GKZ hypergeometric system

by S-J. Matsubara-Heo, N.Takayama

1 About this document

This document explains Risa/Asir functions for GKZ hypergeometric system (A-hypergeometric system).
Loading the package:

```
import("mt_gkz.rr");
```

References cited in this document.

- [MT2020] Saiei-Jaeyeong Matsubara-Heo, Nobuki Takayama, Algorithms for Pfaffian Systems and Cohomology Intersection Numbers of Hypergeometric Integrals, Lecture Notes in Computer Science 12097 (2020), 73--84. Errata is posted on `http://arxiv.org/abs/???`. E-attachments can be obtainable at `http://www.math.kobe-u.ac.jp/OpenXM/Math/intersection2`

- [GM2020] Yoshiaki Goto, Saiei-Jaeyeong Matsubara-Heo, Homology and cohomology intersection numbers of GKZ systems, arXiv:2006.07848

- [SST1999] M.Saito, B.Sturmfels, N.Takayama, Hypergeometric polynomials and integer programming, Compositio Mathematica, 155 (1999), 185--204

- [SST2000] M.Saito, B.Sturmfels, N.Takayama, Groebner Deformations of Hypergeometric Differential Equations. Springer, 2000.

References for maple packages IntegrableConnections and OreMorphisms.

- [BCEW] M.Barkatou, T.Cluzeau, C.El Bacha, J.-A.Weil, IntegrableConnections a maple package for computing closed form solutions of integrable connections (2012). `https://www.unilim.fr/pages perso/thomas.cluzeau/Packages/IntegrableConnections/PDS.html`

- [CQ] T.Cluzeau and A.Quadrat, OreMorphisms: A homological algebraic package for factoring, reducing and decomposing linear functional systems (2009). `https://who.rocq.inria.fr/Alban.Quadrat/OreMorphisms/index.html`

- [CQ08] T.Cluzeau, A.Quadrat, Factoring and decomposing a class of linear functional systems, Linear Algebra and its Applications (LAA), 428(1): 324-381, 2008.

2 Pfaff equation

2.1 Pfaff equation for given cocycles

2.1.1 `mt_gkz.pfaff_eq`

`mt_gkz.pfaff_eq(A,Beta,Ap,Rvec,DirX)`
:: It returns the Pfaff equation for the GKZ system defined by A and $Beta$ with respect to cocycles defined by $Rvec$.

return   a list of coefficients of the Pfaff equation with respect to the direction *DirX*

A   the matrix A of the GKZ system.

Beta   the parameter vector of the GKZ system.

Ap   See [MT2020].

Rvec   It is used to specify a basis of cocycles as explained below. See also [MT2020].

DirX   a list of dxi's.

- The independent variables are x1, x2, x3, ...

- When $Rvec=[v\_1, v\_2, ..., v\_r]$ where r is the rank of the GKZ system, the set of the cocycles standing for Av_1, Av_2, ..., Av_r (see [MT2020]) is supposed to be the basis to construct the Pfaffian system. The exponents (q_ℓ, q) of the integral representation $\int \prod h_\ell^{-q_\ell} x^q \frac{dx}{x}$ is shifted by Av_i=: A_{v_i} as $(q_\ell, q) + A_{v_i}$. Let a_1, a_2, ..., a_n be the column vectors of the matrix A and v be a column vector (x_1, x_2, ..., x_n)^T. Av is defined as a_1 x_1 + a_2 x_2 + ... + a_n x_n.

- When the columns of A are expressed as $e_i \otimes \alpha_{i_j}$, the columns of Ap is $e_i \otimes 0$ where e_i is the i-th unit vector. See [MT2020] on the definition of Ap. Here are some examples. When A is

  ```
  [[1,1,0,0],
   [0,0,1,1],
   [0,1,0,1]]
  ```
  Ap is
  ```
  [[1,1,0,0],
   [0,0,1,1],
   [0,0,0,0]] <-- zero row
  ```
  When A is
  ```
  [[1,1,1,0,0,0],
   [0,0,0,1,1,1],
   [0,1,0,0,1,0],
   [0,0,1,0,0,1]
  ]
  ```
  Ap is
  ```
  [[1,1,1,0,0,0],
   [0,0,0,1,1,1],
  ```

```
   [0,0,0,0,0,0],  <-- zero row
   [0,0,0,0,0,0]   <-- zero row
  ]
```
See also page 223 of [SST2000].

- Option *xrule*. When the option *xrule* is given, the x variables specified by this option are specialized to numbers.

- Option *shift*. When the matrix A is not normal (the associated toric ideal is not normal), a proper shift vector must be given to obtain an element of the b-ideal. Or, use the option *b_ideal* below. See [SST1999] on the theory.

- Option *b_ideal*. When the matrix A is not normal, the option `b_ideal=1` obtains b-ideals and the first element of each b-ideal is used as the b-function. The option *shift* is ignored.

- Option *cg*. A constant matrix given by this option is used for the Gauge transformation of the Pfaffian system. In other words, the basis of cocycles specified by *Rvec* is transformed by the constant matrix given by this option.

- By mt_gkz.use_hilbert_driven(Rank), the rank of the GKZ system is assumed to be Rank. It makes the computation of Groebner basis by yang.rr faster. This option is disabled by mt_gkz.use_hilbert_driven(0);

Example: Gauss hypergeometric system, see [GM2020] example ??.

```
[1883] import("mt_gkz.rr");
[2657] PP=mt_gkz.pfaff_eq(A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]],
            Beta=[-g1,-g2,-c],
            Ap = [[1,1,0,0],[0,0,1,1],[0,0,0,0]],
            Rvec = [[1,0,0,0],[0,0,1,0]],
            DirX=[dx4,dx3] | xrule=[[x1,1],[x2,1]],
            cg=matrix_list_to_matrix([[1,0],[-1,1]]))$

Bfunctions=[s_1*s_2-s_1*s_3+s_1^2,s_1*s_3,s_2^2+(-s_3+s_1)*s_2,s_3*s_2]
-- snip --
[2658] PP[0];
[ (g2*x3-g2)/(x4-x3) (g2*x3)/(x4-x3) ]
[ ((-g2*x3-c+g2)*x4+(c-g1)*x3+g1)/(x4^2-x3*x4)
            ((-g2*x3-c)*x4+(c-g1)*x3)/(x4^2-x3*x4) ]
[2659] PP[1];
[ (-g2*x4+g2)/(x4-x3) (-g2*x4)/(x4-x3) ]
[ ((g2*x3+c-g2-1)*x4+(-c+g1+1)*x3-g1)/(x3*x4-x3^2)
      ((g2*x3+c-g2-1)*x4+(-c+g1+g2+1)*x3)/(x3*x4-x3^2) ]
```

Example: The role of shift. When the toric ideal is not normal, a proper shift vector must be given with the option `shift` to find an element of the b-ideal.

```
[1882] load("mt_gkz.rr");
[1883] A=[[1,1,1,1],[0,1,3,4]];
   [[1,1,1,1],[0,1,3,4]]
```

```
[1884] Ap=[[1,1,1,1],[0,0,0,0]];
   [[1,1,1,1],[0,0,0,0]]
[1885] Rvec=[[0,0,0,0],[0,0,1,0],[0,0,0,1],[0,0,0,2]];
   [[0,0,0,0],[0,0,1,0],[0,0,0,1],[0,0,0,2]];
[2674] P=mt_gkz.pfaff_eq(A,[b1,b2],Ap,Rvec,DirX=[dx4
     | xrule=[[x1,1],[x2,2],[x3,4]] )$
dx remains
stopped in step_up at line 342 in file "./mt_gkz/saito-b.rr"
342     if (type(dn(Ans)) > 1) error("dx remains");
(debug) quit
// Since the toric ideal for A is not normal, it stops with the error.
[2675]  P=mt_gkz.pfaff_eq(A,[b1,b2],Ap,Rvec,DirX=[dx4
    | shift=[1,0],xrule=[[x1,1],[x2,2],[x3,4]])$
// It works.
```

Refer to ⟨undefined⟩ [mt_gkz.ff1], page ⟨undefined⟩, ⟨undefined⟩ [mt_gkz.ff2], page ⟨undefined⟩, Section 2.1.2 [mt_gkz.ff], page 4, Section 2.1.3 [mt_gkz.rvec_to_fvec], page 5,

2.1.2 `mt_gkz.ff2`, `mt_gkz.ff1`, `mt_gkz.ff`

`mt_gkz.ff(Rvec0,A,Ap,Beta)`
`mt_gkz.ff1(Rvec0,A,Beta,Ap)`
`mt_gkz.ff2(Rvec0,A,Beta,Ap,BF,C)`

:: `ff` returns a differential operator whose action to 1 gives the cocycle defined by *Rvec0*

return   `ff` returns a differential operator whose action to 1 of $M_A(\beta)$ gives the cocycle defined by *Rvec0*.

return   `ff1` returns a composite of step-down operators for the positive part of *Rvec0*

return   `ff2` returns a composite of step-up operators for the positive part of *Rvec0*

Rvec0   An element of *Rvec* explained in Section 2.1.1 [mt_gkz.pfaff_eq], page 2.

BF   the list of b-functions to all directions.

C   the list of the step up operators for all a_1, a_2, ..., a_n.

Other arguments are same with those of `pfaff_eq`.

- The function `ff` generates the list of b-functions and the list of step up operators and store them in the cache variable. They can be obtained by calling as `S=mt_gkz.get_bf_step_up()` where S[0] is the list of b-functions and S[1] is the list of step up operators. Step up operators are obtained by the algorithm given in [SST1999].
- Option nf. When nf=1, the output operator is reduced to the normal form with respect to the Groebner basis of the GKZ system of the graded reverse lexicographic order.
- Option shift. See Section 2.1.1 [mt_gkz.pfaff_eq], page 2.
- Internal info: The function `mt_gkz.bb` gives the constant so that the step up and step down operators (contiguity operators) give contiguity relations for the integral representation in [MT2020]. Note that `mt_gkz.ff1` and `mt_gkz.ff2` give contiguity relations which are constant multiple of those for hypergeometric polynomials.

- Internal info: `mt_gkz.step_up` generates step up operators of [SST1999] from b-functions by utilizing `mt_gkz.bf2euler` and `mt_gkz.toric`.

Example: Step up operators compatible with the integral representation in [MT2020]. The function hgpoly_res defined in `check-by-hgpoly.rr` returns a multiple of the hypergeometric polynomial which agrees with the residue times a power of $2\pi\sqrt{-1}$ of the integral representation. See [SST1999].

```
[1883] import("mt_gkz.rr")$
[3175] load("mt_gkz/check-by-hgpoly.rr")$
[3176]  A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]]$
[3177]  B=newvect(3,[5,4,7])$ Ap=[[1,1,0,0],[0,0,1,1],[0,0,0,0]]$
[3179]  Beta=[b1,b2,b3]$  R=[0,0,-1,0]$
[3180]  F2=hgpoly_res(A,B,2);  // HG polynomial. 2 is the number of e_i's.
  10*x1^2*x2^3*x4^4+20*x1*x2^4*x3*x4^3+6*x2^5*x3^2*x4^2
[3182]  mt_gkz.ff(R,A,Ap,Beta); // the operator standing for R
  (x3*x4*dx4+x3^2*dx3+x1*x4*dx2+x1*x3*dx1+x3)/(b1+b2-b3+1)
[3184] S=mt_gkz.get_bf_step_up(A); // b-function and non-reduced step up op's
  [[ s_1*s_2-s_1*s_3+s_1^2 s_1*s_3 s_2^2+(-s_3+s_1)*s_2 s_3*s_2 ],
  [ x2*x3*dx4+x1*x3*dx3+x1*x2*dx2+x1^2*dx1+x1
    x2*x4*dx4+x1*x4*dx3+x2^2*dx2+x1*x2*dx1+x2
    x3*x4*dx4+x3^2*dx3+x1*x4*dx2+x1*x3*dx1+x3
    x4^2*dx4+x3*x4*dx3+x2*x4*dx2+x2*x3*dx1+x4 ]]
[3185] Fvec=mt_gkz.ff2(R,A,Beta,Ap,S[0],S[1]);
  (x3*x4*dx4+x3^2*dx3+x1*x4*dx2+x1*x3*dx1+x3)/(b1+b2-b3+1)
[3188] Fvec = base_replace(Fvec,assoc(Beta,vtol(B)));
  1/3*x3*x4*dx4+1/3*x3^2*dx3+1/3*x1*x4*dx2+1/3*x1*x3*dx1+1/3*x3
[3189] R32d = odiff_act(Fvec,F2,[x1,x2,x3,x4]); // Act Fvec to the hg-poly
  10*x1^3*x2^2*x4^5+50*x1^2*x2^3*x3*x4^4+50*x1*x2^4*x3^2*x4^3+10*x2^5*x3^3*x4^2
[3190] red(R32d/hgpoly_res(A,B+newvect(3,[0,1,0])),2));
   // R32d agrees with the HG polynomial with Beta=[5,4,7]+[0,1,0].
  1
```

Refer to   Section 2.1.1 [mt_gkz.pfaff_eq], page 2,

2.1.3 `mt_gkz.rvec_to_fvec`

`mt_gkz.rvec_to_fvec(Rvec,A,Ap,Beta)`
          :: It returns a set of differential operators standing for *Rvec*.

return   It returns a set of differential operators of which action to $1 \in M_A(\beta)$ give cocycles specified by *Rvec*.

A, Ap, Beta
          Same with Section 2.1.1 [mt_gkz.pfaff_eq], page 2,

- Internal info: this function builds the set of operators by calling Section 2.1.2 [mt_gkz.ff], page 4.

Example: The following two expressions are congruent because $2a_1 - a_2 - a_3 + a_4 = a_1$ for this `A`.

```
[1883] import("mt_gkz.rr");
```

```
[3191] mt_gkz.rvec_to_fvec([[2,-1,-1,1],[0,0,1,0]],
 [[1,1,0,0],[0,0,1,1],[0,1,0,1]],
 [[1,1,0,0],[0,0,1,1],[0,0,0,0]],[b1,b2,b3]);
[(x2*x3*x4^2*dx1^2*dx4^3+((x1*x3*x4^2+x2*x3^2*x4)*dx1^2*dx3
 +(x1*x2*x4^2+x2^2*x3*x4)*dx1^2*dx2+(x1^2*x4^2+2*x1*x2*x3*x4+x2^2*x3^2)*dx1^3
 +(x1*x4^2+3*x2*x3*x4)*dx1^2)*dx4^2+(x1*x3^2*x4*dx1^2*dx3^2
 +((x1^2*x3*x4+x1*x2*x3^2)*dx1^3+(3*x1*x3*x4+x2*x3^2)*dx1^2)*dx3
 +x1*x2^2*x4*dx1^2*dx2^2+((x1^2*x2*x4+x1*x2^2*x3)*dx1^3
 +(3*x1*x2*x4+x2^2*x3)*dx1^2)*dx2+x1^2*x2*x3*dx1^4
 +(x1^2*x4+3*x1*x2*x3)*dx1^3+(x1*x4+x2*x3)*dx1^2)*dx4)
 /(b3*b2*b1^3+(b3*b2^2+(-b3^2-2*b3)*b2)*b1^2+(-b3*b2^2+(b3^2+b3)*b2)*b1),
 (dx3)/(b2)]
[3192] mt_gkz.rvec_to_fvec([[1,0,0,0],[0,0,1,0]],
    [[1,1,0,0],[0,0,1,1],[0,1,0,1]],
    [[1,1,0,0],[0,0,1,1],[0,0,0,0]],[b1,b2,b3]);
[(dx1)/(b1),(dx3)/(b2)]
```

Refer to   Section 2.1.1 [mt_gkz.pfaff_eq], page 2,

2.1.4 `mt_gkz.fvec_to_conn_mat`

`mt_gkz.fvec_to_conn_mat(Fvec,A,Beta,DirX)`
         :: It returns the coefficient matrices of the basis *Fvec* or *DirX*[I]*Fvec in terms of the set of the standard basis.

return   It returns the coefficient matrices of the basis *Fvec* or *DirX*[I]*Fvec in terms of the set of the standard basis of the Groebner basis explained below.

A Beta   Same with Section 2.1.1 [mt_gkz.pfaff_eq], page 2.

DirX   When *DirX* is 1, this function returns the matrix which expresses *Fvec* in terms of the set of the standard monomials of the Groebner basis of the GKZ system in the ring of rational function coefficients with respect to the graded reverse lexicographic order. In other cases, it returns the coefficient matrices of *DirX*[I]'s*Fvec in terms of the set of the standard basis of the Groebner basis.

- It utilizes a Groebner basis computation by the package `yang.rr` and `yang.reduction` to obtain connection matrices.

- This function calls some utility functions `mt_gkz.dmul(Op1,Op2,XvarList)` (multiplication of Op1 and Op2 and `mt_gkz.index_vars(x,Start,End | no_=1)` which generates indexed variables without the underbar "_".

- We note here some other utility functions in this section: `mt_gkz.check_compatibility(P,Q,X,Y)`, which checkes if the sytem d/dX-P, d/dY-Q is compatible.

     Example: The following example illustrates how mt_gkz.pfaff_eq obtains connection matrices.

```
[1883] import("mt_gkz.rr");
[3201] V=mt_gkz.index_vars(x,1,4 | no_=1);
  [x1,x2,x3,x4]
```

```
[3202] mt_gkz.dmul(dx1,x1^2,V);
  x1^2*dx1+2*x1
[3204] A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]]$
   Ap=[[1,1,0,0],[0,0,1,1],[0,0,0,0]]$
   Beta= [b1,b2,b3]$
   Rvec = [[1,0,0,0],[0,0,1,0]]$
   Fvec = mt_gkz.rvec_to_fvec(Rvec,A,Ap,Beta)$
    /* Express cocyles Rvec
        by elements Fvec in the Weyl algebra by contiguity relations. */
   Cg = matrix_list_to_matrix([[1,0],[1,-1]])$
[3208] NN=mt_gkz.fvec_to_conn_mat(Fvec,A,Beta,1);
  // Express Fvec by the standard monomials Std=NN[1].
 1 ooo 2 .ooo
  [[ (x4)/(b1*x1) (b1-b3)/(b1*x1) ]
  [ (-x4)/(b1*x2) (1)/(x3) ],[dx4,1]]
[3209] Std=NN[1];
  [dx4,1]
[3173] NN=NN[0];
  [ (x4)/(b1*x1) (b1-b3)/(b1*x1) ]
  [ (-x4)/(b2*x3) (1)/(x3) ]
[3174] NN1=mt_gkz.fvec_to_conn_mat(Fvec,A,Beta,dx1)[0];
  // Express dx1*Fvec by the standard monomials Std.
 1 ooo 2 .ooo
  [ ((2*b1+b2-b3-1)*x1*x4^2+(-b1+b3+1)*x2*x3*x4)/(b1*x1^3*x4-b1*x1^2*x2*x3)
     ((b1^2+(-2*b3-1)*b1-b3*b2+b3^2+b3)*x1*x4
       +(-b1^2+(2*b3+1)*b1-b3^2-b3)*x2*x3)/(b1*x1^3*x4-b1*x1^2*x2*x3) ]
   [(b1 (-b1*x1*x4^2-b2*x2*x3*x4)/(b2*x1^2*x3*x4-b2*x1*x2*x3^2)
     (b1*x1*x4+(-b1+b3)*x2*x3)/(x1^2*x3*x4-x1*x2*x3^2) ]
[3188] P1=map(red,Cg*NN1*matrix_inverse(NN)*matrix_inverse(Cg));
  [ ((-b2*x3+(b1+b2-b3-1)*x1)*x4+(-b1+b3+1)*x2*x3)/(x1^2*x4-x1*x2*x3)
     (b2*x3*x4)/(x1^2*x4-x1*x2*x3) ]
  [ ((-b2*x3+(b2-b3-1)*x1)*x4+(-b1+b3+1)*x2*x3+b1*x1*x2)/(x1^2*x4-x1*x2*x3)
     ((b2*x3+b1*x1)*x4)/(x1^2*x4-x1*x2*x3) ]

[3191] mt_gkz.pfaff_eq(A,Beta,Ap,Rvec,[dx1]|cg=Cg)[0]-P1;
  [ 0 0 ]
  [ 0 0 ]  // P1 agrees with the output of mt_gkz.pfaff_eq.
```

Refer to   Section 2.1.1 [mt_gkz.pfaff_eq], page 2,

2.1.5 `mt_gkz.contiguity`, `mt_gkz.contiguity_by_fvec`

`mt_gkz.contiguity(A,Beta,Ap,Rvec1,Rvec2)`
         :: It returns the coefficient matrix P that satisfies $Rvec1 = P\ Rvec2$.

`mt_gkz.contiguity_by_fvec(A,Beta,Ap,Fvec1,Fvec2)`
         :: It returns the coefficient matrix P that satisfies $Fvec1 = P\ Fvec2$.

return   The coefficient matrix P that satisfies $Rvec1 = P\ Rvec2$ or $Fvec1 = P\ Fvec2$

A Beta Ap Rvec1 Rvec2

        Same with Section 2.1.1 [mt_gkz.pfaff_eq], page 2.

- It returns the contiguity relation between *Rvec1* and *Rvec2*

Example:

```
[1883] import("mt_gkz.rr");
[3200] PP=mt_gkz.contiguity(A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]],
            Beta=[-g1,-g2,-c],
            Ap = [[1,1,0,0],[0,0,1,1],[0,0,0,0]],
            Rvec1 = [[1,0,0,0],[0,0,1,0]],
            Rvec2 = [[0,0,1,0],[1,0,0,0]]);

[3366] Fvec411=mt_gkz.rvec_to_fvec(Rvec411=[[1,1,0]],
                  A=[[1,1,1],[1,0,1],[0,1,1]],
                  Ap=[[1,1,1],[0,0,0],[0,0,0]],
                  Beta=[eps,-eps*del,-eps*del])$
 Fvec411d=[mt_gkz.dmul(dx1,Fvec411[0],[x1,x2,x3])];
  [(dx1^2*dx2)/(eps^2-eps)]
[3367] mt_gkz.contiguity_by_fvec(A,Beta,Ap,Fvec411d,Fvec411);
 1 .ooo
[ ((del+1)*eps-1)/(x1) ]
```

Refer to   Section 2.1.1 [mt_gkz.pfaff_eq], page 2, Section 2.1.4 [mt_gkz.fvec_to_conn_mat],
            page 6,

3 b function

3.1 b function and facet polynomial

3.1.1 `mt_gkz.bf`

`mt_gkz.bf(A,Facet_poly,II0)`
:: It returns the b-function with respect to the direction *II0*.

return It returns the b-function introduced Saito with respect to the direction *II0* in case of *A* is normal or an element of b-ideal when a proper shift vector is given in case of *A* is not normal.

A the matrix A of the GKZ system.

Facet_poly
  The set of facet polynomials of the convex hull of *A*.

II0 Direction expressed as 0, 1, 2, ... (not 1, 2, 3, ...) to obtain the b function.

- See [SST1999] on the b-function introduced Saito and b-ideal.
- The facet polynomial must be primitive.

  Example:
```
[1883] import("mt_gkz.rr");

[3193] A;
  [[1,1,0,0],[0,0,1,1],[0,1,0,1]]
[3194] Fpoly=mt_gkz.facet_poly(A);
  [[s_3,s_1,s_2-s_3+s_1,s_2],[[0,0,1],[1,0,0],[1,1,-1],[0,1,0]]]
[3196] mt_gkz.bf(A,Fpoly,0);
  s_1*s_2-s_1*s_3+s_1^2
[3197] mt_gkz.bf(A,Fpoly,1);
  s_1*s_3
```

`Refer to` Section 2.1.2 [mt_gkz.ff], page 4, ⟨undefined⟩ [mt_gkz.facet_poly], page ⟨undefined⟩,

3.1.2 `mt_gkz.facet_poly`

`mt_gkz.facet_poly(A)`
:: It returns the set of facet polynomials and their normal vectors of the cone defined by *A*.

return It returns the set of facet polynomials and their normal vectors of the cone generated by the column vectors of the matrix *A*.

A the matrix A of the GKZ system.

- The facet polynomial f is primitive. In other words, all f(a_i) is integer and min f(a_i)=1 for a_i's not being on f=0. where a_i is the i-th column vector of the matrix *A*. It can be checked by `mt_gkz.is_primitive(At,Facets)` where *At* is the transpose of *A* and *Facets* is the second return value of this function.

- This function utilizes the system polymake `https://polymake.org` on our server.

  Example:

  ```
  [1883] import("mt_gkz.rr");
  [1884] mt_gkz.facet_poly([[1,1,1,1],[0,1,2,3]]);
    oohg_native=0, oohg_curl=1
    [[s_2,-s_2+3*s_1],[[0,1],[3,-1]]]
  ```

4 Utilities

4.1 Some utility functions

We only show examples on these functions. As for details, please see the source code.

```
[1883] import("mt_gkz.rr");
[2667] mt_gkz.dvar([x1,x2]);  // it generates variables starting with d
  [dx1,dx2]
[2669] mt_gkz.p_true_nf_rat((1/3)*x^3-1,[x^2-1],[x],0);
  [x-3,3]      // p_true_nf does not accept rational number coefficients
[2670] A=[[1,1,1,1],[0,1,3,4]];
  [[1,1,1,1],[0,1,3,4]]
[2671] mt_gkz.reduce_by_toric(dx3^4,A);
  dx1*dx4^3   // reduction by toric ideal defined by A
[2672] nk_toric.toric_ideal(A);
  [-x1*x4+x2*x3,-x2*x4^2+x3^3,x2^2*x4-x1*x3^2,-x1^2*x3+x2^3]
[2673] mt_gkz.yang_gkz_buch(A,[b1,b2]);  // Groebner basis of GKZ system by yang.rr
 1 o 2 ..o 3 ..ooooooooo 4 o 6 ooo 9 o
[[[(x2)*<<0,1,0,0>>+(3*x3)*<<0,0,1,0>>+ ---snip ---*<<0,0,0,0>>,1]],
[dx1,dx2,dx3,dx4],
[(1)*<<0,0,0,2>>,(1)*<<0,0,1,0>>,(1)*<<0,0,0,1>>,(1)*<<0,0,0,0>>]]

[2674] mt_gkz.dp_op_to_coef_vec([x1*<<1,0>>+x1*x2*<<0,1>>,x1+1],[<<1,0>>,<<0,1>>]);
  // x1+1 is the denominator
  [ (x1)/(x1+1) (x1*x2)/(x1+1) ]
[2675] mt_gkz.tk_base_is_equal([1,2],[1,2]);
 1
[2676] mt_gkz.tk_base_is_equal([1,2],[1,x,y]);
 0
[2677] mt_gkz.mdiff(sin(x),x,1);
 cos(x)
[2678] mt_gkz.mdiff(sin(x),x,2); //2nd derivative
 -sin(x)
[3164] mt_gkz.ord_xi(V=[x1,x2,x3],II=1);
// matrix to define graded lexicographic order so that V[II] is the smallest.
[ 1 1 1 ]
[ 0 -1 0 ]
[ -1 0 0 ]
[3166] load("mt_gkz/check-by-hgpoly.rr");
[3187] check_0123(); // check the pfaffian for the A below by hg-polynomial.
  A=[[1,1,1,1],[0,1,2,3]]
  Ap=[[1,1,1,1],[0,0,0,0]]
  --- snip ---
  Bfunctions= --- snip ---
  0 (vector) is expected:
  [[ 0 0 0 ],[ 0 0 0 ]]
```

```
[3188] mt_gkz.get_check_fvec();
 // get the basis of cocycles used in terms of differential operators.
  [1,(dx4)/(b1),(dx4^2)/(b1^2-b1)]
[3189] mt_gkz.clear_bf();
0
[3190] mt_gkz.get_bf_step_up(A=[[1,1,1,1],[0,1,2,3]]);
// b-functions and step-up operators.
// Option b_ideal=1 or shift=... may be used for non-normal case.
  [[ -s_2^3+(9*s_1-3)*s_2^2+ ---snip---
     -s_2^3+(3*s_1+1)*s_2^2-3*s_1*s_2 s_2^3-3*s_2^2+2*s_2 ],
  [ x3^3*dx4^2+ ---snip---
    3*x3^2*x4*dx4^2+ --- snip---]]
[3191] mt_gkz.mytoric_ideal(0 | use_4ti2=1);
// 4ti2 is used to obtain a generator set of the toric ideal
// defined by the matrix A
[3192] mt_gkz.mytoric_ideal(0 | use_4ti2=0);
// A slower method is used to obtain a generator set of the toric ideal
// defined by the matrix A.  4ti2 is not needed. Default.
[3193] mt_gkz.cbase_by_euler(A=[[1,1,1,1],[0,1,3,4]]);
// Cohomology basis of the GKZ system defined by A for generic beta.
// Basis is given by a set of Euler operators.
// It is an implementation of the algorithm in http://dx.doi.org/10.1016/j.aim.2016.10
// beta is set by random numbers. Option: no_prob=1
```

5 Cohomology intersection numbers

5.1 Secondary equation

5.1.1 `mt_gkz.kronecker_prd`

`mt_gkz.kronecker_prd(`*A*`,`*B*`)`
        :: It returns the Kronecker product of A and B.

return   a matrix which is equal to the Kronecker product of A and B (`https://en.`
              `wikipedia.org/wiki/Kronecker_product`).

A,B   list

```
[2644]  A=[[a,b],[c,d]];
[[a,b],[c,d]]
[2645] B=[[e,f],[g,h]];
[[e,f],[g,h]]
[2646] mt_gkz.kronecker_prd(A,B);
[ e*a f*a e*b f*b ]
[ g*a h*a g*b h*b ]
[ e*c f*c e*d f*d ]
[ g*c h*c g*d h*d ]
```

5.1.2 `mt_gkz.secondary_eq`

`mt_gkz.secondary_eq(`*A*`,`*Beta*`,`*Ap*`,`*Rvec*`,`*DirX*`)`
        :: It returns the secondary equation with respect to cocycles defined by Rvec.

return   a list of coefficients of the Pfaffian system corresponding to the secondary equa-
              tion (cf. equation (8) of [MT2020]).

A,Beta,Ap,Rvec,DirX
        see `pfaff_eq`

- The secondary equation is originally a Pfaffian system for an unkwon r by r matrix I
  with $r =$length(Rvec). We set $Y = (I_{11}, I_{12}, ..., I_{1r}, I_{21}, I_{22}, ...)^T$. Then, the secondary
  equation can be seen as a Pfaffian system $\frac{dY}{dx_i} = A_i Y$ with DirX$=\{dx_i\}_i$. The function
  mt_gkz.secondary_eq(A,*Beta*,*Ap*,*Rvec*,*DirX*) outputs a list obtained by aligning the
  matrices A_i.

- Let $F := (\omega_i)_i$ be a column vector whose entries are given by the cohomology classes
  specified by entries of Rvec. Then, `pfaff_eq` computes the Pfaffian matrices P_i so
  that $\frac{dF}{dx_i} = P_i F$. If Q_i denotes the matrix obtained by replacing Beta by -Beta, we
  have $A_i =$`mt_gkz.kronecker_prd`(E,P_i)+`mt_gkz.kronecker_prd`(Q_i,E) where E is the
  identity matrix of size length(Rvec).

- Options xrule, shift, b_ideal,cg. Same as `pfaff_eq`.

  Example:

```
[2647] Beta=[b1,b2,b3]$
[2648] DirX=[dx1,dx4]$
```

```
[2649] Rvec=[[1,0,0,0],[0,0,1,0]]$
[2650] A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]]$
[2651] Ap=[[1,1,0,0],[0,0,1,1],[0,0,0,0]]$
[2652] Xrule=[[x2,1],[x3,1]]$
[2653] P=mt_gkz.secondary_eq(A,Beta,Ap,Rvec,DirX|xrule=Xrule)$
--snip--
[2654] length(P);
2
[2655] P[0];
[[(-2*x1^3*x4^2+4*x1^2*x4-2*x1)/(x1^4*x4^2-2*x1^3*x4+x1^2),(b2*x4)/(x1^2*x4-x1),
(-b2*x4)/(x1^2*x4-x1),0],[(b1)/(x1*x4-1),
((b2-4/3)*x1^2*x4^2+(-b1-b2+8/3)*x1*x4+b1-4/3)/(x1^3*x4^2-2*x1^2*x4+x1),0,
(-b2*x4)/(x1^2*x4-x1)],[(-b1)/(x1*x4-1),0,
((-b2-2/3)*x1^2*x4^2+(b1+b2+4/3)*x1*x4-b1-2/3)/(x1^3*x4^2-2*x1^2*x4+x1),
(b2*x4)/(x1^2*x4-x1)],[0,(-b1)/(x1*x4-1),(b1)/(x1*x4-1),0]]
<--- Paffian matrix in x1 direction.
[2656] P[1];
[[0,(b2)/(x1*x4-1),(-b2)/(x1*x4-1),0],[(b1*x1)/(x1*x4^2-x4),
((b2-1/3)*x1^2*x4^2+(-b1-b2+2/3)*x1*x4+b1-1/3)/(x1^2*x4^3-2*x1*x4^2+x4),0,
(-b2)/(x1*x4-1)],[(-b1*x1)/(x1*x4^2-x4),0,
((-b2+1/3)*x1^2*x4^2+(b1+b2-2/3)*x1*x4-b1+1/3)/(x1^2*x4^3-2*x1*x4^2+x4),
(b2)/(x1*x4-1)],[0,(-b1*x1)/(x1*x4^2-x4),(b1*x1)/(x1*x4^2-x4),0]]
<--- Paffian matrix in x4 direction.
```
Refer to   Section 2.1.1 [mt_gkz.pfaff_eq], page 2,

5.1.3 `mt_gkz.generate_maple_file_IC`

`mt_gkz.generate_maple_file_IC(`*A,Beta,Ap,Rvec,DirX*`)`
>   :: It returns the maple input for a solver of a Pfaffian system IntegrableConnections[RationalSolutions].

return   a maple input file for the function IntegrableConnections[RationalSolutions] (cf. [BCEW]) for the Pfaffian system mt_gkz.secondary_eq(*A,Beta,Ap,Rvec,DirX*).

A,Beta,Ap,Rvec,DirX
>   see `pfaff_eq`.

- A maple package IntegrableConnections is available in [BCEW]. In order to use IntegrableConnections, you need to add the global path to the file IntegrableConnections.m to libname on maple. See [BCEW].

- If Beta contains unkwon variables, they are regarded as generic parameters. For example, if Beta=[b1,b2,1/5,1/7,b5,...], parameters are [b1,b2,b5,...].

- Options xrule, shift, b_ideal,cg. Same as `pfaff_eq`.

- Option filename. You can specify the file name by specifying the option variable filename.   If you do not specify it, `generate_maple_file_IC` generates a file "auto-generated-IC.ml".

Example:
```
[2681] Beta=[b1,b2,1/3]$
```

```
[2682] DirX=[dx1,dx4]$
[2683] Rvec=[[1,0,0,0],[0,0,1,0]]$
[2684] A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]]$
[2685] Ap=[[1,1,0,0],[0,0,1,1],[0,0,0,0]]$
[2687] Xrule=[[x2,1],[x3,1]]$
[2688] mt_gkz.generate_maple_file_IC(A,Beta,Ap,Rvec,DirX|xrule=Xrule,filename="Test.ml
```

```
//A file named Test.ml is automatically generated as follows:
```

```
with(OreModules);
with(IntegrableConnections);
with(linalg);
C:=[Matrix([[(-2*x1^3*x4^2+4*x1^2*x4-2*x1)/(x1^4*x4^2-2*x1^3*x4+x1^2),
(b2*x4)/(x1^2*x4-x1),(-b2*x4)/(x1^2*x4-x1),0],[(b1)/(x1*x4-1),
((b2-4/3)*x1^2*x4^2+(-b1-b2+8/3)*x1*x4+b1-4/3)/(x1^3*x4^2-2*x1^2*x4+x1),0,
(-b2*x4)/(x1^2*x4-x1)],[(-b1)/(x1*x4-1),0,
((-b2-2/3)*x1^2*x4^2+(b1+b2+4/3)*x1*x4-b1-2/3)/(x1^3*x4^2-2*x1^2*x4+x1),
(b2*x4)/(x1^2*x4-x1)],[0,(-b1)/(x1*x4-1),(b1)/(x1*x4-1),0]]),
Matrix([[0,(b2)/(x1*x4-1),(-b2)/(x1*x4-1),0],[(b1*x1)/(x1*x4^2-x4),
((b2-1/3)*x1^2*x4^2+(-b1-b2+2/3)*x1*x4+b1-1/3)/(x1^2*x4^3-2*x1*x4^2+x4),0,
(-b2)/(x1*x4-1)],[(-b1*x1)/(x1*x4^2-x4),0,
((-b2+1/3)*x1^2*x4^2+(b1+b2-2/3)*x1*x4-b1+1/3)/(x1^2*x4^3-2*x1*x4^2+x4),
(b2)/(x1*x4-1)],[0,(-b1*x1)/(x1*x4^2-x4),(b1*x1)/(x1*x4^2-x4),0]])];
RatSols:=RationalSolutions(C,[x1,x4],['param',[b1,b2]]);
```

```
/*
If you run the output file on maple, you obtain a rational solution of
the secondary equation.
*/
```

```
          [b2*(3*b1-1)/(b1*x1^2)]
RatSols:=[3*b2/x1               ]
          [3*b2/x1               ]
          [3*b2-1                ]
```

```
/*
Note that the 4 entries of this vector correspond to entries of a 2 by 2 matrix.
They are aligned as (1,1), (1,2), (2,1) (2,2) from the top.
*/
```

```
Refer to   Section 2.1.1 [mt_gkz.pfaff_eq], page 2,
```

5.1.4 `mt_gkz.generate_maple_file_MR`

`mt_gkz.generate_maple_file_MR(A,Beta,Ap,Rvec,DirX,D1,D2)`
:: It returns the maple input for a solver of a Pfaffian system Morphism-sRat[OreMorphisms].

return   a maple input file for the function MorphismsRat[OreMorphisms] (cf. [CQ]) for the Pfaffian system obtained by `secondary_eq`. If you run the output file on maple, you obtain a rational solution of the secondary equation.

A,Beta,Ap,Rvec,DirX
see `pfaff_eq`.

D1,D2   Positive integers. D1 (resp. D2) is the upper bound of the degree of the numerator (resp. denominator) of the solution.

- We use the same notation as the explanation of `generate_maple_file_IC`. Let D denote the ring of linear differential operators with coeffiecients in the field of rational functions. We consider D-modules $R := D^{1 \times l} / \sum_{dx_i \in DirX} D^{1 \times l}(\partial_i E - P_i)$ and $S := D^{1 \times l} / \sum_{dx_i \in DirX} D^{1 \times l}(\partial_i E + Q_i^T)$ where $l =$length(Rvec). Then, computing a rational solution of the secondary equation is equivalent to computing a D-morphism from R to S represented by rational function matrix (cf. pp12-13 of [CQ08]).

- A maple package OreMorphisms is available in [CQ]. In order to use OreMorphisms, you need to add the global path to the file OreMorphisms.m to libname on maple.

- Options xrule, shift, b_ideal,cg. Same as `pfaff_eq`.

- Option filename. You can specify the file name as in `generate_maple_file_IC`.

- The difference between `generate_maple_file_IC` and `generate_maple_file_MR` is the appearence of auxilliary variables D1 and D2. If you can guess the degree of the numerator and the denominator of the solution of the secondary equation, Morphism-sRat[OreMorphisms] can be faster than RationalSolutions[IntegrableConnections].

Example:
```
[2668] Beta=[b1,b2,1/3]$
[2669] DirX=[dx1,dx4]$
[2670] Rvec=[[1,0,0,0],[0,0,1,0]]$
[2671] A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]]$
[2672] Ap=[[1,1,0,0],[0,0,1,1],[0,0,0,0]]$
[2673] Xvar=[x1,x4]$
[2674] Xrule=[[x2,1],[x3,1]]$
[2675] mt_gkz.generate_maple_file_MR(A,Beta,Ap,Rvec,DirX,2,2|xrule=Xrule)$


//A file "auto-generated-MR.ml" is automatically generated as follows:


with(OreModules);
with(OreMorphisms);
with(linalg);
Alg:=DefineOreAlgebra(diff=[dx1,x1],diff=[dx4,x4],polynom=[x1,x4],comm=[b1,b2]);
```

```
P:=Matrix([[dx1,0],[0,dx1],[dx4,0],[0,dx4]])
-Matrix([[((b1+b2-4/3)*x1*x4-b1+4/3)/(x1^2*x4-x1),(-b2*x4)/(x1^2*x4-x1)],
[(-b1)/(x1*x4-1),(b1*x4)/(x1*x4-1)],[(b2*x1)/(x1*x4-1),(-b2)/(x1*x4-1)],
[(-b1*x1)/(x1*x4^2-x4),(1/3*x1*x4+b1-1/3)/(x1*x4^2-x4)]]);
Q:=Matrix([[dx1,0],[0,dx1],[dx4,0],[0,dx4]])
+Matrix([[((-b1-b2-2/3)*x1*x4+b1+2/3)/(x1^2*x4-x1),(b1)/(x1*x4-1)],
[(b2*x4)/(x1^2*x4-x1),(-b1*x4)/(x1*x4-1)],[(-b2*x1)/(x1*x4-1),(b1*x1)/(x1*x4^2-x4)],
[(b2)/(x1*x4-1),(-1/3*x1*x4-b1+1/3)/(x1*x4^2-x4)]]);
RatSols:=MorphismsRat(P,Q,Alg,0,2,2);


/*
If you run the output file on maple, you obtain a vector RatSols.
RatSols[1] is the rational solution of the secondary equation:
*/

RatSols[1]:=[(1/3)*n_{2_{1_{3_1}}}*(3*b1-1)/(b1*x1^2*d_{6_1})   n_{2_{1_{3_1}}}/(x1*d_{6_1})]
            [n_{2_{1_{3_1}}}/(x1*d_{6_1})            (1/3)*n_{2_{1_{3_1}}}*(3*b2-1)/(b2*d_{6_1})]

/*
Here, n_{2_{1_{3_1}}} and d_{6_1} are arbitrary constants. We can take n_{2_{1_{3_1}}} = 3*b2 and d_{6_1} =
1 to obtain the rational solution of the secondary equation which is iden-
tical to the one obtained from generate_maple_file_IC.
*/
```

Refer to   Section 2.1.1 [mt_gkz.pfaff_eq], page 2, Section 5.1.3 [mt_gkz.generate_maple_file_IC],
            page 14.

5.2 Normalizing the cohomology intersection matrix

5.2.1 mt_gkz.principal_normalizing_constant

mt_gkz.principal_normalizing_constant(A,T,$Beta$,K)

> :: It returns the normalizing constant of the cohomology intersection matrix in terms of a regular triangulation T.

return   a rational function which is the cohomology intersection number $\frac{1}{(2\pi\sqrt{-1})^n}\langle[\frac{dt}{t}],[\frac{dt}{t}]\rangle_{ch}$ in terms of the regular triangulation T. Here, n is the number of integration variables and $\frac{dt}{t}$ is the volume form $\frac{dt_1}{t_1}\wedge\cdots\wedge\frac{dt_n}{t_n}$ of the complex n-torus.

A,Beta   see pfaff_eq.

T   a regular triangulation of A.

K   The number of polynomial factors in the integrand. see [MT2020].

- This function is useful when the basis of the cohomology group $\{\omega_i\}_{i=1}^r$ is given so that $\omega_1 = [\frac{dt}{t}]$.

- One can find a regular triangulation by using a function `mt_gkz.regular_triangulation`.
- `mt_gkz.leading_terms` can be used more generally.

  Example:
```
[2676] A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]]$
[2677] Beta=[b1,b2,b3]$
[2678] K=2$
[2679] T=[[1,2,3],[2,3,4]]$
[2680] mt_gkz.principal_normalizing_constant(A,T,Beta,K);
(-b1-b2)/(b3*b1+b3*b2-b3^2)
```

Refer to   Section 5.2.2 [mt_gkz.leading_terms], page 18.

5.2.2 `mt_gkz.leading_terms`

`mt_gkz.leading_terms(A,Beta,W,Q1,Q2,K,N)`
          :: It returns the W-leading terms of a cohomology intersection number specified by Q1 and Q2 up to W-degree=(minimum W-degree)+N.

return   a list [[C1,DEG1],[C2,DEG2],...]. Each CI is a rational function depending on Beta times a monomial x^m in x-variables. DEGI is the W-degree of x^m. The cohomology intersection number $\frac{1}{(2\pi\sqrt{-1})^n}\langle[h^{-q_1'}t^{q_1''}\frac{dt}{t}],[h^{-q_2'}t^{q_2''}\frac{dt}{t}]\rangle_{ch}$ has a Laurent expansion of the form C1+C2+....

A,Beta   see `pfaff_eq`.

W   a positive and integral weight vector.

Q1,Q2 $Q1 = (q_1',q_1'')^T$, $Q2 = (q_2',q_2'')^T$ are integer vectors. The lengths of q_1' and q_2' are both equal to K.

K   The number of polynomial factors in the integrand. see [MT2020].

N   A positive integer.

- For a monomial $x^m = x_1^{m_1} \cdots x_n^{m_n}$ and a weight vector $W = (w_1, \ldots, w_n)$, the W-degree of x^m is given by the dot product $m \cdot W = m_1 w_1 + \cdots + m_n w_n$.
- The W-leading terms of the cohomology intersection number $\frac{1}{(2\pi\sqrt{-1})^n}\langle[h^{-q_1'}t^{q_1''}\frac{dt}{t}],[h^{-q_2'}t^{q_2''}\frac{dt}{t}]\rangle_{ch}$ can be computed by means of Theorem 2.6 of [GM2020]. See also Theorem 3.4.2 of [SST2000].
- If the weight vector is not generic, you will receive an error message such as "WARNING(initial_mon): The weight may not be generic". In this case, the output may be wrong and you should retake a suitable W. To be more precise, W should be chosen from an open cone of the Groebner fan.
- Option xrule. Same as `pfaff_eq`.

  Example:
```
[2922] Beta=[b1,b2,1/3];
[b1,b2,1/3]
[2923] Q=[[1,0,0],[0,1,0]];
[[1,0,0],[0,1,0]]
```

```
[2924] A=[[1,1,0,0],[0,0,1,1],[0,1,0,1]];
[[1,1,0,0],[0,0,1,1],[0,1,0,1]]
[2925] W=[1,0,0,0];
[1,0,0,0]
[2926] K=2;
2
[2927] N=2;
2
[2928] NC=mt_gkz.leading_terms(A,Beta,W,Q[0],Q[1],K,N|xrule=[[x2,1],[x3,1],[x4,1]])$
--snip--
[2929] NC;
[[(-3)/(x1),-5],[0,-4],[0,-3]]
```

```
/*
This output means that the W-leading term of the (1,2) entry of the cohomology
intersection matrix is (-3)/(x1)× (2π√-1). In view of examples of generate_
maple_file_IC or generate_maple_file_MR, we can conclude that the cohomology
intersection matrix is given by
*/
```

```
[-(3*b1-1)/(b1*x1^2)  -3/x1           ]
[-3/x1                -(3*b2-1)/b2]]
```

//divided by $2\pi\sqrt{-1}$.

Refer to Section 5.2.2 [mt_gkz.leading_terms], page 18, Section 5.1.3 [mt_gkz.generate_maple_file_IC], page 14, Section 5.1.4 [mt_gkz.generate_maple_file_MR], page 16.

5.2.3 `mt_gkz.leading_term_rat`

`mt_gkz.leading_term_rat(P,W,V)`
:: It returns the W-leading term of a rational function P depending on variables V.

return   It returns the W-leading term of a rational function P.

P   a rational function.

W   a weight vector.

V   a list of variables of P.

- This function is supposed to be combined with `leading_terms` to compute the leading term of a cohomology intersection number.
- If W is chose so that there are several initial terms, you will receive an error message "WARNING(leading_term_rat):The weight vector may not be generic."

Refer to Section 5.2.2 [mt_gkz.leading_terms], page 18.

5.3 Regular triangulations

5.3.1 `mt_gkz.toric_gen_initial`, `mt_gkz.regular_triangulation`, `mt_gkz.top_standard_pairs`

`mt_gkz.toric_gen_initial(A,W)`
`mt_gkz.regular_triangulation(A,W)`
`mt_gkz.top_standard_pairs(A,W)`
          :: utility functions for computing ring theoretic invariants: generic initial ideal
          for the toric ideal specified by the matrix A and a weight W, its associated
          regular triangulation, and its associated top-dimensional standard pairs.

return   `toric_gen_initial` returns a list [L1,L2] of length 2. L1 is a list of generators
          of the W-initial ideal of the toric ideal I_A specified by A. L2 is a list of variables
          of I_A.

return   `regular_triangulation` returns a list of simplices of a regular triangulation
          T_W specified by the weight W.

return   `top_standard_pairs` returns a list of the form [[L1,S1],[L2,S2],...]. Each SI is
          a simplex of T_W. Each LI is a list of exponents.

A   a configuration matrix.

W   a positive weight vector.

- As for the definition of the standard pair, see Chapter 3 of [SST00].
- We set n=length(A) and set BS1:=$\{1, 2, ..., n\} \setminus S1$. Then, each L1[I] is an exponent **k**
  of a top-dimensional standard pair $(\partial_{BS1}^{\mathbf{k}}, S1)$. Here, **k** is a list of length n-length(S1)
  and $\partial_{BS1} = (\partial_J)_{J \in BS1}$.
- If the weight vector is not generic, you will receive an error message such as "WARN-
  ING(initial_mon): The weight may not be generic". See also `leading_terms`.
- These functions are utilized in `leading_terms`.

  Example: An example of a non-unimodular triangulation and non-trivial standard pairs.
```
[3256] A=[[1,1,1,1,1],[0,1,0,2,0],[0,0,1,0,2]];
[[1,1,1,1,1],[0,1,0,2,0],[0,0,1,0,2]]
[3257] W=[2,0,1,2,2];
[2,0,1,2,2]
[3258] mt_gkz.toric_gen_initial(A,W);
--snip--
[[x1*x5,x1*x4,x3^2*x4],[x1,x2,x3,x4,x5]]
[3259] mt_gkz.regular_triangulation(A,W);
--snip--
[[2,4,5],[2,3,5],[1,2,3]]
[3260] mt_gkz.top_standard_pairs(A,W);
--snip--
[[[[0,0],[0,1]],[2,4,5]],[[[0,0]],[2,3,5]],[[[0,0]],[1,2,3]]]

/*
This means that the regular triangulation of the configuration matrix A is
```

given by $T = \{\{2,4,5\}, \{2,3,5\}, \{1,2,3\}\}$. The normalized volumes of these simplices
are 2,1 and 1. Moreover, the top-dimensional standard pairs are
$(1, \{2,4,5\}), (\partial_3, \{2,4,5\})$, $(1, \{2,3,5\}), (1, \{1,2,3\})$.
*/

Refer to   Section 5.2.2 [mt_gkz.leading_terms], page 18.

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

Table of Contents