

Macaulay matrix method for GKZ hypergeometric systems

Pfaffian system (Pfaff equation) and Restriction
Version 0.8
April 3, 2023

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

1 About this document

This document explains Risa/Asir functions for Macaulay matrix method for GKZ hypergeometric systems (A-hypergeometric systems).

Loading the package:

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

References cited in this document.

- [amp2022a] V.Chestnov, F.Gasparotto, M.K.Mandal, P.Mastrolia, S.J.Matsubara-Heo, H.J.Munch, N.Takayama, Macaulay Matrix for Feynman Integrals: Linear Relations and Intersection Numbers, [https://doi.org/10.1007/JHEP09\(2022\)18](https://doi.org/10.1007/JHEP09(2022)18). E-attachments can be obtainable at <http://www.math.kobe-u.ac.jp/OpenXM/Math/amp-MM>
- [amp2023a] V.Chestnov, S.J.Matsubara-Heo, H.J.Munch, N.Takayama, Restrictions of Pfaffian Systems for Feynman Integrals. [https://doi.org/10.1007/JHEP11\(2023\)202](https://doi.org/10.1007/JHEP11(2023)202) E-attachments can be obtainable at <http://www.math.kobe-u.ac.jp/OpenXM/Math/amp-Restriction>
- [Barkatou2017] M.Barkatou, Symbolic Methods for Solving Systems of Linear Ordinary Differential Equations (tutorial slides). https://www.impan.pl/~slawek/pisa/Barkatou_p.pdf

2 Pfaff equation

2.1 `mt_mm.find_macaulay`

`mt_mm.find_macaulay(Ideal, Std, Xvars)`

:: It returns a Macaulay type matrix for the *Ideal* with respect to the standard basis *Std*.

return Data for Macaulay matrix.

Ideal Generators of an ideal. It should be a list of distributed polynomials.

Std Standard basis of the ideal *Ideal*. It should be a list of distributed polynomials.

Xvars Independent variables

Option `deg`, which is the starting degree of the Macaulay matrix.

Option `p`, which is the prime number for the probabilistic rank check.

Option `restriction_var`, which constructs a Macaulay matrix to obtain Pfaffian equations for restriction automatically.

Option `restriction_cond`, which is a non-automatic setting to construct a Macaulay matrix for restriction.

- The data for Macaulay matrix is a list of length 4. The list is $[M_1, M_2, \text{Extra}, \text{Std}]$. M_1 and M_2 are M_{Ext} and M_{Std} in the Section 4 of [amp2022a] respectively. *Extra* and *Std* are *Ext* and *Std* in the Section 4 of [amp2022a] respectively.
- A Macaulay matrix is obtained by applying differential monomials ∂^a , $|a| \leq d$ to generators of the ideal *Ideal* where d is a (given) degree. In the default setting, the degree d is automatically increased until the Macaulay matrix is solvable.
- Here is a quick summary of the Section 4 of [amp2022a]. Let *Std* be a column vector of a standard basis and *Ext* a column vector of differential monomials appearing the Macaulay matrix other than *Std*. When we have a Macaulay matrix of sufficiently large degree, there exists a matrix *C* of rational function entries satisfying the identity $d_i \text{Std} - P_i \text{Std} = C(M_1 \text{Ext} + M_2 \text{Std})$, which is 0 modulo *Ideal*, where P_i is the Pfaffian matrix and d_i is the partial differential operator with respect to the i -th variable. $d_i \text{Std}$ can be expressed as $C'_1 \text{Ext} + C'_2 \text{Std}$ where C'_1 and C'_2 are a matrix with 0, 1 entries. It follows from the identity that we have $C'_1 = C M_1$ and $C'_2 - P_i = C M_2$. Finding a matrix *C*, we obtain the Pfaffian matrix P_i .
- The global variable *NT_ES* stores all differentials of *Std* (standard monomials) minus *Std* in the distributed polynomial format. For example, when *Std* is $[1, dx, dx*dy, dy]$, *NT_ES* is $[dx^2, dx^2*y, dx*dy^2, dy^2]$. `mt_mm.get_NT_ES()` returns the value of *NT_ES*.
- The option `restriction_var` is a list of variables to restrict. These variables are set to zero and *NT_ES* are set by using only non-restricted variables. For example, when `Xvars=[x,y,z]; restriction_var=[y,z];`, the variables y, z are set to zero and only dx is applied to *Std*.

- The option `restriction_cond` is a list of a list of variables applying to *Std* and a list of restriction rules. For example, when `Xvars=[r1,r2,y1,y2]; restriction_cond=[[r1,r2],[[r1,1/4],[r2,1/5],[y1,1],[y2,0]]]`, the operators *dr1* and *dr2* are applied to *Std* to construct *NT_ES* and the restriction `[[r1,1/4],[r2,1/5],[y1,1],[y2,0]]` is performed.
- `mt_mm.get_NT_info()[3]` returns the argument to construct the Macaulay matrix. This information may be used when the option *restriction_cond* is used. For example, when `Xvars=[r,y]; restriction_cond=[[r],[[r,1/4],[y,1/5]]]`, the returned *MData* is restricted to `[[r,1/4],[y,1/5]]`. In order to get non-restricted *MData*, we may input `mt_mm.my_macaulay(Ideal,Std,Deg,length(Xvars) | restriction_cond=[[r],[[r,1/4],[y,1/5]]])`; where *Deg* is given in `mt_mm.get_NT_info()[3][3]`.

Example: Macaulay matrix of $x_1\partial_1 + 2x_2\partial_2 - b_1\partial_1^2 - \partial_2$ with respect to the standard monomials $1, \partial_2$. The PDE is the GKZ system for the 1 by 2 matrix $A = [1, 2]$.

```
/* We input the followings */
import("mt_mm.rr");
Ideal=[x1*dx1+2*x2*dx2-b1,dx1^2-dx2]$
Std=[<<0,0>>,<<0,1>>]$ Xvars=[x1,x2]$
Id = map(dp_ptod,Ideal,poly_dvar(Xvars));
MData=mt_mm.find_macaulay(Id,Std,Xvars);
P1=mt_mm.find_pfaffian(MData,Xvars,1 | use_orig=1);
P2=mt_mm.find_pfaffian(MData,Xvars,2 | use_orig=1);

[3404]
--- snip ---
[3405] [[(b1)/(x1),(-2*x2)/(x1)],
      [(1/2*b1^2-1/2*b1)/(x1*x2),((-b1+1)*x2-1/2*x1^2)/(x1*x2)]] // P1
[3406] [[0,1],
      [(-1/4*b1^2+1/4*b1)/(x2^2),((b1-3/2)*x2+1/4*x1^2)/(x2^2)]] // P2
```

Note that Macaulay matrix method might give a fake answer under some situations. For example, we input the following non-integrable system. In other words, the *Ideal* below is the trivial ideal. We also input a wrong standard basis *Std*.

```
/* We input the followings */
Ideal=[x1*dx1+x2*dx2-b1,dx1^2-dx2]$
Std=[<<0,0>>,<<0,1>>]$ Xvars=[x1,x2]$
Id = map(dp_ptod,Ideal,poly_dvar(Xvars));
MData=mt_mm.find_macaulay(Id,Std,Xvars);
P1=mt_mm.find_pfaffian(MData,Xvars,1 | use_orig=1);
P2=mt_mm.find_pfaffian(MData,Xvars,2 | use_orig=1);
nd_weyl_gr(Ideal,[x1,x2,dx1,dx2],0,0);

[3240] import("mt_mm.rr");
--- snip ---
[3542] MData=mt_mm.find_macaulay(Id,Std,Xvars);
```

We use a probabilistic method for rank_check with P=65537
Warning: prules are ignored.

```

--- snip ---
NT_info=[rank,[row, col]( of m=transposed MData[0]),indep_columns(L[2])]
Rank=6
The matrix is full rank.
We use a probabilistic method to compute the rank of a matrix with entries of rational
The output is a macaulay matrix of degree 1
--- snip ---
[rank=6,[row,col]=[6,6] (size of t(M1)),Indep_cols] is stored in the variable NT_info.
--- snip ---
[[[0,0,0,0,0,x1],[0,0,1,0,0,0],[0,0,0,x1,x2,0],[0,1,0,0,-1,0],
  [0,0,x1,x2,0,-b1+1],[1,0,0,-1,0,0]],
 [[-b1,x2],[0,-1],[0,-b1+1],[0,0],[0,0],[0,0]],
 [(1)*<<3,0>>,(1)*<<2,1>>,(1)*<<2,0>>,(1)*<<1,1>>,(1)*<<0,2>>,(1)*<<1,0>>],
 [(1)*<<0,0>>,(1)*<<0,1>>]]

[3543] P1=mt_mm.find_pfaffian(MData,Xvars,1 | use_orig=1);
[[ (b1)/(x1), (-x2)/(x1)], [(b1^2-b1)/(x1*x2), ((-b1+1)*x2-x1^2)/(x1*x2)]]

// d/dx1 - P1
[3544] P2=mt_mm.find_pfaffian(MData,Xvars,2 | use_orig=1);
[[0,1], [(-b1^2+b1)/(x2^2), ((2*b1-2)*x2+x1^2)/(x2^2)]]
// d/dx2 - P2
[2545] nd_weyl_gr(Ideal,[x1,x2,dx1,dx2],0,0);
[-1] // But it is a trivial ideal!

```

Let us go back to our running example. Homogeneity is reduced by `mt_gkz.gkz_b`. `Id0` is an ODE.

```

/* We input the followings. */
A=[[1,2]]$ Beta=[b1]$ Sigma=[2]$
Id0=mt_gkz.gkz_b(A,Beta,Sigma|partial=1)$
Xvars=[x2]$
Id=map(mt_mm.remove_redundancy,Id0,Xvars);
Std=[1,dx2]$ Std=map(dp_ptod,Std,[dx2])$
MData=mt_mm.find_macaulay(Id,Std,Xvars);
P1=mt_mm.find_pfaffian(MData,Xvars,1 | use_orig=1);

[3244] A=[[1,2]]$ Beta=[b1]$ Sigma=[2]$
Id0=mt_gkz.gkz_b(A,Beta,Sigma|partial=1)$
Xvars=[x2]$
Id=map(mt_mm.remove_redundancy,Id0,Xvars);
[3245] [3246] [3247] [3248]
[3249] [(x1^2)*<<2>>+(1/2*x1^3)*<<1>>+(-1/2*b1*x1^2)*<<0>>]
// This is the ODE.

```

```

[3252] Std=[1,dx2]$ Std=map(dp_ptod,Std,[dx2])$
MData=mt_mm.find_macaulay(Id,Std,Xvars);

[3253] [3254] We use a probabilistic method for rank_check with P=65537
Warning: prules are ignored.
--- snip ---
NT_info=[rank,[row, col]( of m=transposed MData[0]),indep_columns(L[2])]
Rank=2
The matrix is full rank.
We use a probabilistic method to compute the rank of a matrix with entries of rational
The output is a macaulay matrix of degree 1
To draw a picture of the rref of the Macaulay matrix, use Util/show_mat with tmp-mm-*-.
[Std,Xvars] is stored in tmp-mm-9823-Std-Xvars.ab
[ES,Xvars] is stored in tmp-mm-9823-Es-Xvars.ab
[rank=2,[row,col]=[2,2] (size of t(M1)),Indep_cols] is stored in the variable NT_info.
NT_info is stored in tmp-mm-9823-NT_info.ab
MData is stored in tmp-mm-9823-mdata.ab

[[[0,x1^2],[x1^2,1/2*x1^3]],
 [[-1/2*b1*x1^2,1/2*x1^3],[0,-1/2*b1*x1^2]],
 [(1)*<<3>>,(1)*<<2>>],
 [(1)*<<0>>,(1)*<<1>>]]
[3255] P1=mt_mm.find_pfaffian(MData,Xvars,1 | use_orig=1);
[[0,1],[1/2*b1,-1/2*x1]]
// d/d<<1>> - P1
[3256]

```

When the parameters and independent variables are numbers, we can call `find_pfaffian_fast`.

```

/* We input the followings. */
A=[[1,2]]$ Beta=[1/2]$ Sigma=[2]$
Id0=mt_gkz.gkz_b(A,Beta,Sigma|partial=1)$
Xvars=[x2]$
Id=map(mt_mm.remove_redundancy,Id0,Xvars)$
Std=[1,dx2]$ Std=map(dp_ptod,Std,[dx2])$
MData=mt_mm.find_macaulay(Id,Std,Xvars);
P1=mt_mm.find_pfaffian_fast(MData,Xvars,1,mt_mm.get_indep_cols() | xrules=[[x1,1/3]]);

--- snip ---
[3300] Generate a data for linsolv.
Sol rank=2
cmd=time /home/nobuki/bin/linsolv --vars tmp-mm-9823-linsolv-vars.txt <tmp-mm-9823-
--- snip ---
#time of linsolv=0.00681901
Time(find_pfaffian_fast)=0.002101 seconds
[3304] P1;
[[0,1],[1/4,-1/6]]

```

Example of finding a restriction: We consider a left ideal generated by $\partial_x(\theta_x + c_{01} - 1) - (\theta_x + \theta_y + a)(\theta_x + b_{02})$ and $\partial_y(\theta_y + c_{11} - 1) - (\theta_x + \theta_y + a)(\theta_y + b_{12})$ where $\theta_x = x\partial_x$. The Appell function F2 satisfies it. We try to find the restriction to $x=0$. We input the following commands.

```

Ideal=[(-x^2+x)*dx^2+(-y*x*dy+(-a-b_02-1)*x+c_01)*dx-b_02*y*dy-b_02*a,
        (-y*x*dy-b_12*x)*dx+(-y^2+y)*dy^2+((-a-b_12-1)*y+c_11)*dy-b_12*a]$
Xvars=[x,y]$
Std=[(1)*<<0,0>>,(1)*<<0,1>>]$
Id = map(dp_ptod,Ideal,poly_dvar(Xvars))$
MData=mt_mm.find_macaulay(Id,Std,Xvars | restriction_var=[x]);
P2=mt_mm.find_pfafrican(MData,Xvars,2 | use_orig=1);
// Std is a standard basis for the restricted system.

```

Then, we have the following output.

```

[3618] We use a probabilistic method for rank_check with P=65537
--- snip ---
Rank=6
Have allocated 0 M bytes.
0-th rank condition is OK.
We use a probabilistic method to compute the rank of a matrix with entries of rational
The output is a macaulay matrix of degree 1
--- snip ---
[rank=6,[row,col]=[8,6] (size of t(M1)),Indep_cols] is stored in the variable NT_info.
[[[0,0,0,0,0,0,0,c_01],[0,0,0,0,0,0,-y^2+y,0],[0,0,0,0,0,c_01,-b_02*y,0],
  [0,0,0,-y^2+y,0,0,(-a-b_12-3)*y+c_11+1,0],
  [0,0,0,0,c_01+1,(-b_02-1)*y,0,(-b_02-1)*a-b_02-1],
  [0,0,-y^2+y,0,0,(-a-b_12-2)*y+c_11,0,-b_12*a-b_12]],

  [[-b_02*a,-b_02*y],[-b_12*a,(-a-b_12-1)*y+c_11],[0,-b_02*a-b_02],
  [0,(-b_12-1)*a-b_12-1],[0,0],[0,0]],

  [(1)*<<3,0>>,(1)*<<2,1>>,(1)*<<1,2>>,(1)*<<0,3>>,(1)*<<2,0>>,(1)*<<1,1>>,
  (1)*<<0,2>>,(1)*<<1,0>>],
  [(1)*<<0,0>>,(1)*<<0,1>>]]

[3619] [[0,1],[(-b_12*a)/(y^2-y),((-a-b_12-1)*y+c_11)/(y^2-y)]]

```

An example of finding a restriction on the singular locus: We consider the system of differential equations for the Appell function $F_4(a, b, c_{01}, c_{11}; x, y)$, which is Id_p below. The curve $xy((x - y)^2 - 2(x + y) + 1) = 0$ is the singular locus of this system and $(x, y) = (1/4, 1/4)$ is on the locus. Let F be a holomorphic solution around the point. The values of $\text{poly_dact}(dx*dy, F, [x, y]), \text{poly_dact}(dx^2, F, [x, y]), \text{poly_dact}(dy^2, F, [x, y])$ (parial derivatives of f) can be obtained from the values of $\text{poly_dact}(1, F, [x, y]), \text{poly_dact}(dx, F, [x, y]), \text{poly_dact}(dy, F, [x, y])$ (standing for Std) on the singular locus by the Macaulay matrix in MData2 by executing the following codes.


```

Id_p=[(-x^2+x)*dx^2+(-2*y*x*dy+(-a-b-1)*x+c_01)*dx-y^2*dy^2+(-a-b-1)*y*dy-b*a,
      -x^2*dx^2+(-2*y*x*dy+(-a-b-1)*x)*dx+(-y^2+y)*dy^2+((-a-b-1)*y+c_11)*dy-b*a];
Xvars=[x,y];
//Restriction at x=y=1/4 included in sing (x-y)^2-2(x+y)+1
Std=map(dp_ptod,[1,dx,dy],[dx,dy])$          // It succeeds at degree 1.
MData=mt_mm.find_macaulay(Id_p,Std,Xvars | restriction_cond=[[x,y],[[x,1/4],[y,1/4]]])
Args=mt_mm.get_NT_info()[3]$
yang.define_ring(["partial",Xvars])$
MData2=mt_mm.my_macaulay(map(dp_ptod,Id_p,Dvars),Std,Args[3],Args[4]
    | restriction_cond=[[x,y],[[]]])$

```

Refer to [\[mt_mm.find_pfaffian_fast\]](#), page [\[undefined\]](#), [\[undefined\]](#)
[\[mt_mm.find_pfaffian\]](#), page [\[undefined\]](#), Section 2.1 [\[mt_mm.find_macaulay\]](#),
page 2,

2.2 mt_mm.my_macaulay

mt_mm.my_macaulay(*Id*, *Std*, *Deg*, *N*)
:: Multiply all differential operators upto *Deg* to *Id* and return a Macaulay matrix.

return [*M1*, *M2*, *Ext*, *Std*], *M1***Ext*+*M2***Std*=0 modulo *Id* holds.

Id A list of generators of the ideal by the distributed polynomial format. Independent variables must be *x1*, *x2*, *x3*, ...

Std A list of Standard monomials by the distributed polynomial format.

Deg Degree to generate a Macaulay matrix.

N *N* is the number of differential variables = the size of the distributed polynomials=the number of independent variables.

option *restriction_var*, see Section 2.1 [\[mt_mm.find_macaulay\]](#), page 2.

option *restriction_cond*, see Section 2.1 [\[mt_mm.find_macaulay\]](#), page 2.

- This function is for an internal use.
- This function is called from the function `mt_mm.find_macaulay`. The function `mt_mm.find_macaulay` also calls the function `mt_mm.rank_check_ff` with `base_set_minus([dxi*Std | for all i], Std)`, whose value is obtained by `ES=mt_mm.get_NT.ES()`. When the system equations by the Macaulay matrix can be solved with respect to *ES*, we are done.

Example: Consider the left ideal generated by $x_1\partial_1 - 1/2, x_2\partial_2 - 1/3$. Input the following codes.

```

import("mt_mm.rr");
yang.define_ring(["partial",[x1,x2]]);
Id=[x1*<<2,0>>-1/2, x2*<<0,1>>-1/3];
Std=[<<0,0>>,<<1,0>>];

```

```
T0=mt_mm.my_macaulay(Id,Std,0,2);
T1=mt_mm.my_macaulay(Id,Std,1,2);
```

Then, we have

```
[3988] T0;
[[[x1,0],[0,x2]],
 [[-1/2,0],[-1/3,0]],
 [(1)*<<2,0>>,(1)*<<0,1>>],
 [(1)*<<0,0>>,(1)*<<1,0>>]]
[3989] T1;
[[[0,0,x1,0,0,0],[0,0,0,0,0,x2],[0,x1,0,0,0,-1/2],[0,0,0,0,x2,2/3],
 [x1,0,1,0,0,0],[0,0,0,x2,0,0]],

 [[-1/2,0],[-1/3,0],[0,0],[0,0],[0,-1/2],[0,-1/3]],

 [(1)*<<3,0>>,(1)*<<2,1>>,(1)*<<2,0>>,(1)*<<1,1>>,(1)*<<0,2>>,(1)*<<0,1>>],
 [(1)*<<0,0>>,(1)*<<1,0>>]]
```

For example, the 5th rows of $T1[0]$ and $T1[1]$, stands for $\partial_1(x_1\partial_1^2 - 1/2) = x_1\partial_1^3 + \partial_1^2 - 1/2\partial_1$.

Refer to Section 2.1 [mt_mm.find_macaulay], page 2,

3 Invariant subvector space

3.1 `mt_mm.ediv_set_field`

`mt_mm.ediv_set_field(Mode)`

:: It sets the type of coefficient field.

Mode When *Mode* is 1, the elementary divisor is factorized in $\mathbf{Q}[x]$ (rational number coefficient polynomial ring). Here *x* can be changed by setting the global variable *InvX* by calling `mt_mm.set_InvX(X)`. When *Mode* is 0, the elementary divisor is factorized in $\mathbf{Q}(a, b, \dots)[x]$ where *a, b, ...* are parameter variables automatically determined from the input.

Refer to Section 3.2 [`mt_mm.mat_inv_space`], page 9,

3.2 `mt_mm.mat_inv_space`

`mt_mm.mat_inv_space(Mat)`

:: It returns sets of basis vectors of invariant subvector spaces of *Mat*

return a list of sets of basis vectors of invariant subvector spaces by the action of *Mat*.

Mat A square matrix.

ediv Option. When *ediv*=1, it returns [a list of sets of basis vectors, [ED, L, R]] where ED is the elementary divisor.

- Let *Mat* be a matrix of rational number entries. When the characteristic polynomial of the matrix *Mat* can be factored over $\mathbf{Q}[x]$ into first order polynomials, it returns bases of eigen vector spaces. In general, it returns bases of invariant subvector spaces corresponding to irreducible factors of the characteristic polynomial.
- *x* is the reserved variable of computing the elementary divisor of *Mat*. Then, *x* cannot be used as a parameter vector.
- Note that the returned basis does not give basis for the Jordan canonical form in general even when the characteristic polynomial is factored into first order polynomials.
- When the option *ediv*=1 is given, it returns [a list of sets of basis vectors, [ED, L, R]]. Note that the format of return value is changed. L and R are matrices such that $L(x - A)R = D$ where *A* is the argument *Mat* and *D* is the matrix ED whose diagonal consists of the elementary divisor.

Example: We have 3 invariant subvector spaces of the 3 by 3 matrix L below. Then, the matrix can be diagonalized by them.

```
[3118] import("invlin0.rr");
Base field is Q.
[3168] B=mt_mm.mat_inv_space(L=[[6,-3,-7],[-1,2,1],[5,-3,-6]]);
[[[-2/25 2/25 -2/25]],[[6/25 3/25 3/25]],[[-1/25 0 -1/25]]]
[3170] length(B);
3
[3171] L=newmat(3,3,L);
```

```

[ 6 -3 -7 ]
[ -1 2 1 ]
[ 5 -3 -6 ]
[3174] P=matrix_transpose(newmat(3,3,[vtol(B[0][0]),vtol(B[1][0]),vtol(B[2][0])]));
[ -2/25 6/25 -1/25 ]
[ 2/25 3/25 0 ]
[ -2/25 3/25 -1/25 ]
[3176] matrix_inverse(P)*L*P;
[ 2 0 0 ]
[ 0 1 0 ]
[ 0 0 -1 ]
[3177] map(print,check5(L))$ // The above can be done by check5.
[[[ -2/25 2/25 -2/25 ]],[[ 6/25 3/25 3/25 ]],[[ -1/25 0 -1/25 ]]]
// invariant subvector spaces.

[ -2/25 6/25 -1/25 ]
[ 2/25 3/25 0 ]
[ -2/25 3/25 -1/25 ]
// transformation matrix

[ 2 0 0 ]
[ 0 1 0 ]
[ 0 0 -1 ]
// diagonalized matrix

[3178] B=mat_inv_space(L | ediv=1)$
[3179] B[1][0]; // The elementary divisor is the diagonal.
[ 1 0 0 ]
[ 0 6/25 0 ]
[ 0 0 x^3-2*x^2-x+2 ]
[3180] fctr(B[1][0][2][2]);
[[1,1],[x-2,1],[x-1,1],[x+1,1]]

```

Example: The characteristic polynomial is factored into $(x^2 + 1)^2$ in $\mathbf{Q}[x]$. Then, we obtain two invariant subspaces.

```

[3196] B=mt_mm.mat_inv_space([[0,-1,0,0],[1,0,0,0],[0,0,0,1],[0,0,-1,0]]);
[[[ 1 0 0 0 ],[ 0 1 0 0 ]],[[ 0 0 1 0 ],[ 0 0 0 -1 ]]]
[3197] length(B);
2

```

Example: Block diagonalization. The matrix L is congruent to $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix}$.

```

[3352] L=matrix_list_to_matrix([[4,0,1],[2,3,2],[0,-2,0]]);
[ 4 0 1 ]
[ 2 3 2 ]
[ 0 -2 0 ]
[3348] B=mt_mm.check5(L);

```

```

[[[ 4 6 -4 ]],[ -3 -6 4 ],[ -8 -16 12 ]],[ 4 -3 -8 ]
[ 6 -6 -16 ]
[ -4 4 12 ],[ 3 0 0 ]
[ 0 0 -4 ]
[ 0 1 4 ]]
[3350] B[0];
[[[ 4 6 -4 ]],[ -3 -6 4 ],[ -8 -16 12 ]]]
[3351] map(length,B[0]);
[1,2]
//There are one dimensional and two dimensional invariant subspaces for L
[3353] matrix_inverse(B[1])*L*B[1];
[ 3 0 0 ]
[ 0 0 -4 ]
[ 0 1 4 ]
[3354] B[2];
[ 3 0 0 ]
[ 0 0 -4 ]
[ 0 1 4 ]
// the matrix L is block diagonalized as 1 by 1 matrix and 2 by 2 matrix.
[3355]

```

Example: When the matrix contains parameters, you need to change the base field. L below is congruent to $\begin{pmatrix} a & 1 \\ 0 & 1 \end{pmatrix}$.

```

[3110] import("mt_mm.rr");
[3111] mt_mm.ediv_set_field(0);
Base field is Q(params).
Warning: computation over Q(params) may require huge memory space and time.
0
[3112] B=mt_mm.mat_inv_space(L=[[a-6,-8],[9/2,a+6]]);
[[[ 1 0 ],[ a-6 9/2 ]]]
[3113] length(B);
1 // one invariant subvector space for the linear map L
[3357] B=check5(L);
[[[ 1 0 ],[ a-6 9/2 ]],[ 1 a-6 ]
[ 0 9/2 ],[ 0 -a^2 ]
[ 1 2*a ]]
[3359] fctr(matrix_det(B[2]-x*matrix_identity_matrix(2)));
[[1,1],[x-a,2]] // characteristic polynomial is (x-a)^2

```

Refer to Section 3.1 [mt_mm.ediv_set_field], page 9,

3.3 mt_mm.is_integral_difference_sol

```

mt_mm.is_integral_difference_sol(F1,F2)
:: Check if the irreducible polynomials F1 and F2 in x have a common solution
with integral difference.

```

return If there is a common solution with integral difference, it returns $[1, Y]$ where Y is an integer and $x_0 + Y$ and x_0 are solutions of $F1$ and $F2$ respectively. If there is no such solution, it returns 0.

F1 A polynomial in x .

F2 A polynomial in x .

- x is the default value of *InvX*. It can be changed by the function `mt_mm.set_InvX()`.
- It called `poly_prime_dec` for the check.

Example: Two examples of irreducible polynomials over \mathbb{Q} and $\mathbb{Q}(a, b)$.

```
[2695] is_integral_difference_sol(x^2+1,(x++1)*(x-+1));
```

```
[1,-1]
```

```
[2696] is_integral_difference_sol(x^2+(b/(a+1))^2,(x-(b/(a+1))*+3)*(x+(b/(a+1))*+3));
```

```
[1,3]
```

Refer to Section 3.2 [mt_mm.mat_inv_space], page 9, [\[mt_mm.eDiv\]](#),
page [\[mt_mm.set_InvX\]](#), page [\[mt_mm.set_InvX\]](#), page [\[mt_mm.set_InvX\]](#),

4 Moser reduction::

4.1 mt_mm.moser_reduction

`mt_mm.moser_reduction(A,X)`

:: It returns the Moser reduced matrix of A with respect to X

return Moser reduced matrix

A Coefficient matrix of the ODE $dF/dX - AF$ where F is an unknown vector valued function.

X Independent variable of the ODE.

- It is an implementation of the Moser reduction algorithm presented in [Barkatou2017]. The Moser reduction algorithm translates a differential equation with the regular singularity at the origin into a form of the coefficient matrix has only the simple pole at the origin. We call the coefficient matrix the Moser reduced matrix.

Example: the following input outputs the Moser reduced form $A2[0]$ of A .

```
import("mt_mm.rr");
A=newmat(4,4, [[-2/x,0,1/(x^2),0],
               [x^2,-(x^2-1)/x,x^2,-x^3],
               [0,x^(-2),x,0],
               [x^2,1/x,0,-(x^2+1)/x]]);
A2=mt_mm.moser_reduction(A,x);

[3405] A2;
[[ (-x^2-1)/(x) x^2 0 x ]
 [ 0 (-2)/(x) (1)/(x) 0 ]
 [ 0 0 (x^2-1)/(x) (1)/(x) ]
 [ -x 1 x (-x^2-1)/(x) ],[ 0 1 0 0 ]
 [ 0 0 0 x^2 ]
 [ 0 0 x 0 ]
 [ 1 0 0 0 ]]
// A2[1] gives the Gauge transformation to obtain the Moser reduced form.
[3406] A2[0]-mt_mm.gauge_transform(A,A2[1],x);
[ 0 0 0 0 ]
[ 0 0 0 0 ]
[ 0 0 0 0 ]
[ 0 0 0 0 ]
```

Refer to `<undefined> [mt_mm.gauge_transform]`, page `<undefined>`,

5 Restriction::

5.1 mt_mm.restriction_to_pt_

`mt_mm.restriction_to_pt_(Ideal, Gamma, KK, V)`

:: It returns the restriction of *Ideal* to the origin by a probabilistic method.

return a list R. R[0]: a basis of the restriction. R[1]: column indices for the basis. R[2]: a basis of $\mathbf{C}^{c(\gamma)}$. R[3]: echelon form of a matrix for the restriction.

Ideal Generators of an ideal in the ring of differential operators

Gamma Approximation parameter γ for the restriction. In order to obtain the exact answer, it must be larger than or equal to $\max(s_0, s_1)$ where s_0 is the maximal non-negative root of b-function and s_1 is the maximal order with respect to $(-1, 1)$ weight of the Groebner basis of the *Ideal*

KK Approximation parameter k for the restriction. This parameter must be larger than γ . If k is sufficiently large, this function returns the exact answer.

- This function is an implementation of the Algorithm 4.7.
- The echelon form is computed by rational arithmetic by default. When we give an option `p=n`, the echelon form is computed in the finite field of size `pari(nextprime,n)`.
- When the option `save_mem=1` is given, it becomes slower to save memory.
- Parameters in *Ideal* should be replaced to rational numbers.
- $c(\gamma)$ is the number of monomials whose total degree is less than or equal to γ . For example, `mt_mm.eset(3,[x,y])` returns of monomials of dx and dy whose total degree is less than or equation to 3 and $c(3) = 10$.
- `mt_mm.restriction_to_pt_by_linsolv` accepts same argument and returns the result in the same format. It computes the echelon form by `linsolv`. When the option `nproc=n` is given, n processes are created to construct a matrix for the restriction from the ideal.

Restriction of the system of Appell F1.

```
[2077] import("mt_mm.rr");
[3599] Ideal=mt_mm.appell_F1();
      [(-x^2+x)*dx^2+((-y*x+y)*dy+(-a-b1-1)*x+c)*dx-b1*y*dy-b1*a,
      ((-y+1)*x*dy-b2*x)*dx+(-y^2+y)*dy^2+((-a-b2-1)*y+c)*dy-b2*a,
      ((x-y)*dy-b2)*dx+b1*dy]
[3600] Ideal2=base_replace(Ideal, [[a,1/2],[b1,1/3],[b2,1/5],[c,1/7]]);
      [(-x^2+x)*dx^2+((-y*x+y)*dy-11/6*x+1/7)*dx-1/3*y*dy-1/6,
      ((-y+1)*x*dy-1/5*x)*dx+(-y^2+y)*dy^2+(-17/10*y+1/7)*dy-1/10,
      ((x-y)*dy-1/5)*dx+1/3*dy]
[3601] T=mt_mm.restriction_to_pt_(Ideal2,3,4,[x,y] | p=10^10)$
[3602] T[0];
      [1] // The basis of the restriction at the origin is 1

[3604] Ideal3=base_replace(Ideal2, [[x,x+2],[y,y+3]]);
      [(-x^2-3*x-2)*dx^2+((-y-3)*x-y-3)*dy-11/6*x-74/21)*dx+(-1/3*y-1)*dy-1/6,
```



```

      (((-y-2)*x-2*y-4)*dy-1/5*x-2/5)*dx+(-y^2-5*y-6)*dy^2+(-17/10*y-347/70)*dy-1/10,
      ((x-y-1)*dy-1/5)*dx+1/3*dy]
[3605] T3=mt_mm.restriction_to_pt_(Ideal3,3,4,[x,y] | p=10^10)$
[3606] T3[0];
      [dx,dy,1]    // The basis of the restriction at (x,y)=(2,3)
[3607]

```

Rational restriction of the system of Appell F2 to $x=0$. The rank 2 system of ODE is stored in the variable P2.

```

import("mt_mm.rr")$
Ideal = [(-x^2+x)*dx^2+(-y*x)*dx*dy+((-a-b1-1)*x+c1)*dx-b1*y*dy-b1*a,
        (-y^2+y)*dy^2+(-x*y)*dy*dx+((-a-b2-1)*y+c2)*dy-b2*x*dx-b2*a]$
Xvars = [x,y]$
//Rule for a probabilistic determination of RStd (Std for the restriction)
Rule=[[y,y+1/3],[a,1/2],[b1,1/3],[b2,1/5],[c1,1/7],[c2,1/11]]$
Ideal_p = base_replace(Ideal,Rule);
RStd=mt_mm.restriction_to_pt_by_linsolv(Ideal_p,Gamma=2,KK=4,[x,y]);
RStd=reverse(map(dp_ptod,RStd[0],[dx,dy]));
Id = map(dp_ptod,Ideal,poly_dvar(Xvars))$
MData = mt_mm.find_macaulay(Id,RStd,Xvars | restriction_var=[x]);
P2 = mt_mm.find_pfaaffian(MData,Xvars,2 | use_orig=1);
end$

```

Refer to Section 2.1 [mt_mm.find_macaulay], page 2,

5.2 mt_mm.v_by_eset

`mt_mm.v_by_eset(L,Eset,V)`
 :: It returns coefficients vector of L with respect to $Eset$

return a list of coefficients of L with respect to $Eset$.

L a differential operator.

$Eset$ a list of monomials of differential variable.

V a list of variables.

- $Eset$ must be sorted in the descending order. For example, $[dx^2,dy*dx,dy^2,dx,dy,1]$ is OK, but $[1,dy,dx,dy^2,dy*dx,dx^2]$ is not accepted.
- It returns coefficients of L with respect to $Eset$.
- This function is used for the preprocess of the function `mt_mm.restriction_to_pt_`.

```

[1883] import("mt_mm.rr");
[3600] mt_mm.v_by_eset(x*dx+y*dy+a,reverse(mt_mm.eset(2,[x,y])),[x,y]);
      [ 0 0 0 x y a ]

[3601] Eset=mt_mm.eset(2,[x,y]);
      [1,dy,dx,dy^2,dy*dx,dx^2]
[3602] T=mt_mm.dshift_by_eset(x*dx+y*dy+a,Eset,[x,y]);
      // Eset is applied to x*dx+y*dy+a and the result is

```

```

[x*dx+y*dy+a,
 x*dy*dx+y*dy^2+(a+1)*dy,
 x*dx^2+(y*dy+a+1)*dx,
 x*dy^2*dx+y*dy^3+(a+2)*dy^2,
 x*dy*dx^2+(y*dy^2+(a+2)*dy)*dx,
 x*dx^3+(y*dy+a+2)*dx^2]

[3603] T2=map(mt_mm.v_by_eset,T,reverse(mt_mm.eset(3,[x,y])),[x,y]);
[[ 0 0 0 0 0 0 0 x y a ],[ 0 0 0 0 0 x y 0 a+1 0 ],
 [ 0 0 0 0 x y 0 a+1 0 0 ],[ 0 0 x y 0 0 a+2 0 0 0 ],
 [ 0 x y 0 0 a+2 0 0 0 0 ],[ x y 0 0 a+2 0 0 0 0 0 ]]
[3604]

```

Refer to Section 5.1 [mt_mm.restriction_to_pt_], page 14,

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

1	About this document	1
2	Pfaff equation	2
3	Invariant subvector space	9
4	Moser reduction::.....	13
5	Restriction::.....	14
	Index	17

Table of Contents

1	About this document	1
2	Pfaff equation.....	2
2.1	mt_mm.find_macaulay.....	2
2.2	mt_mm.my_macaulay.....	7
3	Invariant subvector space.....	9
3.1	mt_mm.ediv_set_field.....	9
3.2	mt_mm.mat_inv_space.....	9
3.3	mt_mm.is_integral_difference_sol.....	11
4	Moser reduction::.....	13
4.1	mt_mm.moser_reduction.....	13
5	Restriction::.....	14
5.1	mt_mm.restriction_to_pt.....	14
5.2	mt_mm.v_by_eset.....	15
	Index.....	17

