Introduction to the OreModules package

Calling Sequence:

OreModules[<function>](args)
<function>(args)

Description:

- *OreModules* is a Maple implementation of algorithms which compute parametrizations, extension modules (ext), resolutions and other algebraic objects for linear systems of differential equations, time-delay systems, etc.
- The algebraic framework for *OreModules* are Ore algebras. In order to deal with modules over Ore algebras computationally, this package is based on the Maple library Mgfun (cf. Ore_algebra, e.g. Ore algebras and non-commutative Groebner bases are developed in Mgfun). Within this unified framework, *OreModules* handles:
  - ordinary differential equations,
  - partial differential equations,
  - multidimensional discrete systems,
  - differential time-delay systems,
  - repetitive systems,
  - multidimensional convolutional codes, etc.
- These systems may be time-invariant or time-varying with polynomial or rational coefficients.
- In the context of *linear control systems*, the main features of *OreModules* are the following:
  - decide controllability and parametrizability,
  - construct (minimal) parametrizations,
  - compute Bezout identities (left/right/generalized inverses),
  - decide flatness (also pi-freeness).
- The package *OreModules*, based on an original program by F. Chyzak and A. Quadrat, is maintained and further developed by A. Quadrat and D. Robertz.
- To use a function of the *OreModules* package, either define that function alone using the command with(OreModules, <function>), or define all *OreModules* functions using the command with(OreModules). Alternatively, invoke the function using the long form OreModules[<function>].
- The functions available in the *OreModules* package are the following:

Define an Ore algebra:

`DefineOreAlgebra`

Module Theory:

`Exti(Rat)`	`Extn(Rat)`
`Torsion(Rat)`	`SyzygyModule(Rat)`
`Resolution(Rat)`	`FreeResolution(Rat)`
`ShorterFreeResolution(Rat)`	`ShortestFreeResolution(Rat)`
`MinimalParametrization(s)(Rat)`	`GeneralizedInverse(Rat)`
`LeftInverse(Rat)`	`RightInverse(Rat)`
`LocalLeftInverse`	`GeneralizedInverse(Rat)`
`OreRank(Rat)`	`Dimension(Rat)`
`ProjectiveDimension(Rat)`	`HilbertSeries(Rat)`
`Complement(Rat)`	`LiftOperators(Rat)`

Linear Systems:

`AutonomousElements(Rat)`	`Brunovsky(Rat)`

Parametrization(Rat) PiPolynomial
IntTorsion(Rat) ParticularSolution(Rat)
FirstIntegral

Control Theory:
ControllabilityMatrix KalmanSystem
TorsionElements(Rat) LQEquations
FinalConditions

Matrix tools:
Mult ApplyMatrix
Involution KroneckerProduct

Tools for modules:
Factorize(Rat) Quotient(Rat)
Elimination(Rat) ReduceMatrix(Rat)
KBasis Connection
Integrability IdealIntersection
PolIntersect

Auxiliary tools:
BoundaryTerms DiffToOre

- For the description of the basic algorithms and for detailed examples, see
F. Chyzak, A. Quadrat, D. Robertz, "Effective algorithms for parametrizing linear control systems over Ore algebras", Applicable Algebra in Engineering, Communication and Computing (AAECC) 16 (2005), pp. 319-376,
A. Quadrat, D. Robertz, "Parametrizing all solutions of uncontrollable multidimensional linear systems", Proceedings of the 16th IFAC World Congress, Prague, 2005,
F. Chyzak, A. Quadrat, D. Robertz, "OreModules: A symbolic package for the study of multidimensional linear systems", in: J. Chiasson, J.-J. Loiseau (eds.), "Applications of Time-Delay Systems", LNCIS 352, Springer, 2006, pp. 233-264,
F. Chyzak, A. Quadrat, D. Robertz, OreModules project, http://wwwb.math.rwth-aachen.de/OreModules.

Examples:

```
> with(OreModules):
```

Example 1: Computation of autonomous elements

Linear differential time-delay system describing a flexible rod (see H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD thesis, University of Orsay, France, 1995):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  shift_action=[delta,t,h]):
> R := evalm([[Dt, -Dt*delta, -1], [2*Dt*delta, -Dt-Dt*delta^2, 0]]);
```

$$R := \begin{bmatrix} Dt & -Dt\,\delta & -1 \\ 2\,Dt\,\delta & -Dt - Dt\,\delta^2 & 0 \end{bmatrix}$$

```
> ApplyMatrix(R, [y1(t),y2(t),u(t)], Alg);
```

$$\begin{bmatrix} \mathrm{D}(y1)(t) - \mathrm{D}(y2)(t-h) - \mathrm{u}(t) \\ 2\,\mathrm{D}(y1)(t-h) - \mathrm{D}(y2)(t) - \mathrm{D}(y2)(t-2\,h) \end{bmatrix}$$

```
> Exti(Involution(R, Alg), Alg, 1);
```

$$\left[\begin{bmatrix} Dt & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -2\,\delta & 1+\delta^2 & 0 \\ -Dt & Dt\,\delta & 1 \\ Dt\,\delta & -Dt & \delta \end{bmatrix}, \begin{bmatrix} 1+\delta^2 \\ 2\,\delta \\ Dt - Dt\,\delta^2 \end{bmatrix}\right]$$

```
> TorsionElements(R, [y1(t),y2(t),u(t)], Alg);
```

$$[[\mathrm{D}(\theta_1)(t) = 0], [\theta_1(t) = -2\,\mathrm{y1}(t-h) + \mathrm{y2}(t) + \mathrm{y2}(t-2\,h)]]$$

Example 2: Study of flatness of linear systems

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l1,l2]):
> R := evalm([[D^2+g/l1, 0, -g/l1], [0, D^2+g/l2, -g/l2]]);
```

$$R := \begin{bmatrix} D^2+\frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2+\frac{g}{l2} & -\frac{g}{l2} \end{bmatrix}$$

Check parametrizability of the system:

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} l1\,D^2+g & 0 & -g \\ 0 & l2\,D^2+g & -g \end{bmatrix}, \begin{bmatrix} D^2\,l2\,g+g^2 \\ D^2\,l1\,g+g^2 \\ D^4\,l2\,l1+D^2\,l2\,g+D^2\,l1\,g+g^2 \end{bmatrix}\right]$$

Since *Ext1*[1] is an identity matrix, the system is (generically) controllable and parametrizable. *Ext1*[3] is a parametrization of the system.

```
> P := Ext1[3];
```

$$P := \begin{bmatrix} D^2\,l2\,g+g^2 \\ D^2\,l1\,g+g^2 \\ D^4\,l2\,l1+D^2\,l2\,g+D^2\,l1\,g+g^2 \end{bmatrix}$$

A left inverse of the parametrization (if it exists) is a *flat output* of the system:

```
> F := LeftInverse(P, Alg);
```

$$F := \begin{bmatrix} -\frac{l1}{g^2\,(-l1+l2)} & \frac{l2}{g^2\,(-l1+l2)} & 0 \end{bmatrix}$$

We want to express the system variables *x1*, *x2*, and *u* in terms of the flat output:

```
> R2 := linalg[stackmatrix](R, F);
```

$$R2 := \begin{bmatrix} D^2+\frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2+\frac{g}{l2} & -\frac{g}{l2} \\ -\frac{l1}{g^2\,(-l1+l2)} & \frac{l2}{g^2\,(-l1+l2)} & 0 \end{bmatrix}$$

```
> E := Elimination(R2, [x1,x2,u], [z1,z2,y], Alg);
```

$$E := \mathrm{table}([1 = \begin{bmatrix} 0 & 0 & l2\,g^2-l1\,g^2 \\ 0 & l2\,g-l1\,g & 0 \\ l2\,g-l1\,g & 0 & 0 \end{bmatrix},$$

$$2 = \begin{bmatrix} l2\,D^2\,l1^2+g\,l1^2 & -l2^2\,g-l2^2\,D^2\,l1 & l2^2\,D^2\,g^3-l2\,D^4\,g^2\,l1^2+l2^2\,D^4\,l1\,g^2-D^2\,g^3\,l1^2-g^4\,l1+g^4\,l2 \\ l1^2 & -l1\,l2 & -D^2\,g^2\,l1^2+l1\,D^2\,g^2\,l2-g^3\,l1+l2\,g^3 \\ l1\,l2 & -l2^2 & -l1\,D^2\,g^2\,l2+D^2\,g^2\,l2^2-g^3\,l1+l2\,g^3 \end{bmatrix}$$

$$])$$

```
> ApplyMatrix(E[1], [x1(t),x2(t),u(t)], Alg)=ApplyMatrix(E[2], [z1(t),z2(t),y(t)], Alg);
```

$$\begin{bmatrix} g^2\,(-l1+l2)\,u(t) \\ (-l1+l2)\,g\,x2(t) \\ (-l1+l2)\,g\,x1(t) \end{bmatrix} =$$

$$\left[l2\,l1^2\left(\frac{d^2}{dt^2}z1(t)\right)+g\,l1^2\,z1(t)-l2^2\,l1\left(\frac{d^2}{dt^2}z2(t)\right)-l2^2\,g\,z2(t)+\left(\frac{d^2}{dt^2}y(t)\right)l2^2\,g^3-\left(\frac{d^2}{dt^2}y(t)\right)g^3\,l1^2-\left(\frac{d^4}{dt^4}y(t)\right)l2\,g^2\,l1^2\right.$$

$$+\left(\frac{d^4}{dt^4}\mathrm{y}(t)\right)l2^2\,l1\,g^2-\mathrm{y}(t)\,g^4\,l1+\mathrm{y}(t)\,g^4\,l2\Big]$$

$$\left[l1^2\,\mathrm{z1}(t)-l1\,l2\,\mathrm{z2}(t)-\left(\frac{d^2}{dt^2}\mathrm{y}(t)\right)l1^2\,g^2+\left(\frac{d^2}{dt^2}\mathrm{y}(t)\right)l1\,g^2\,l2-\mathrm{y}(t)\,g^3\,l1+\mathrm{y}(t)\,l2\,g^3\right]$$

$$\left[l1\,l2\,\mathrm{z1}(t)-l2^2\,\mathrm{z2}(t)-\left(\frac{d^2}{dt^2}\mathrm{y}(t)\right)l1\,g^2\,l2+\left(\frac{d^2}{dt^2}\mathrm{y}(t)\right)l2^2\,g^2-\mathrm{y}(t)\,g^3\,l1+\mathrm{y}(t)\,l2\,g^3\right]$$

Up to invertible constants, the previous equations express *x1*, *x2*, and *u* in terms of the flat output *y* (modulo the system equations).

See Also:

with, Ore_algebra, DefineOreAlgebra, ApplyMatrix, Involution, Elimination, LeftInverse, Exti, TorsionElements, AutonomousElements, Parametrization, LQEquations.

OreModules[ApplyMatrix] - **apply operator to a (vector of) function(s)**

Calling Sequence:

ApplyMatrix(M,v,Alg)

Parameters:

M	-	scalar in **Alg** or matrix with entries in **Alg**
v	-	function or list or vector of functions
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***ApplyMatrix*** applies the operator represented by **M** to **v** which is a function or a list or a vector of functions, i.e. it computes the matrix product of **M** by **v**, where scalar multiplication is replaced by the action of scalar operators (represented by elements in **Alg**) on functions. (The action of elements in **Alg** on functions is determined by the commutation rules in the Ore algebra.)
- If **M** is a scalar in **Alg**, then **M** is applied to **v** if **v** is a function or to every entry in **v** if **v** is a list or vector of functions.
- If **M** is a matrix, then **v** is expected to be a list or a vector and the length of **v** must be equal to the number of columns of **M**.
- **Alg** is expected to be defined using DefineOreAlgebra.
- The result of ***ApplyMatrix*** is a function in case **v** is a function and a vector of functions if **v** is a list or a vector. In the latter case, the length of the result equals the number of rows of **M**., if **M** is a matrix, or equals the number of entries of **v**, if **M** is a scalar in **Alg**.
- This command extends applyopr in Ore_algebra. DiffToOre provides a counterpart to ***ApplyMatrix***. To compose two or more operators, use Mult.

Examples:

```
> with(OreModules):
```

Example 1: Ordinary differential equations

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l1,l2]):
```

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> R := evalm([[D^2+g/l1, 0, -g/l1], [0, D^2+g/l2, -g/l2]]);
```

$$R := \begin{bmatrix} D^2 + \frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2 + \frac{g}{l2} & -\frac{g}{l2} \end{bmatrix}$$

```
> ApplyMatrix(R, [x1(t),x2(t),u(t)], Alg) = matrix([[0],[0]]);
```

$$\begin{bmatrix} \left(\frac{d^2}{dt^2} \mathrm{x1}(t)\right) + \frac{g\,\mathrm{x1}(t)}{l1} - \frac{g\,\mathrm{u}(t)}{l1} \\ \left(\frac{d^2}{dt^2} \mathrm{x2}(t)\right) + \frac{g\,\mathrm{x2}(t)}{l2} - \frac{g\,\mathrm{u}(t)}{l2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Example 2: Differential time-delay systems

```
> Alg := DefineOreAlgebra(diff=[D,t], dual_shift=[delta,s], polynom=[t,s],
  comm=[a,omega,zeta,k], shift_action=[delta,t]):
```

Differential time-delay system describing a wind tunnel (A. Manitius, *Feedback controllers for a wind tunnel model involving a delay: analytical design and numerical simulations*, IEEE Trans. Autom. Contr. vol. 29 (1984), 1058-1068):

```
> R := evalm([[D+a, -k*a*delta, 0, 0], [0, D, -1, 0], [0, omega^2, D+2*zeta*omega,
  -omega^2]]);
```

$$R := \begin{bmatrix} D + a & -k\,a\,\delta & 0 & 0 \\ 0 & D & -1 & 0 \\ 0 & \omega^2 & D + 2\,\zeta\,\omega & -\omega^2 \end{bmatrix}$$

```
> ApplyMatrix(R, [x1(t),x2(t),x3(t),u(t)], Alg) = matrix([[0],[0],[0]]);
```

$$\begin{bmatrix} D(x1)(t) + a\,x1(t) - k\,a\,x2(t-1) \\ D(x2)(t) - x3(t) \\ \omega^2\,x2(t) + D(x3)(t) + 2\,\zeta\,\omega\,x3(t) - \omega^2\,u(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> DiffToOre(lhs(%), [x1,x2,x3,u], Alg);
```

$$\left[\begin{bmatrix} D + a & -k\,a\,\delta & 0 & 0 \\ 0 & D & -1 & 0 \\ 0 & \omega^2 & D + 2\,\zeta\,\omega & -\omega^2 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right]$$

See Also:

DefineOreAlgebra, Ore_algebra[applyopr], Ore_algebra[Ore_to_diff], DiffToOre, Mult, Involution, KroneckerProduct, ReduceMatrix, LeftInverse, RightInverse, GeneralizedInverse.

OreModules[AutonomousElements],

OreModules[AutonomousElementsRat] - **return torsion elements in terms of the system variables and as integrated autonomous elements**

Calling Sequence:

AutonomousElements(R,v,Alg)
AutonomousElementsRat(R,v,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`v`	-	list or vector of functions
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***AutonomousElements*** returns a generating set of the autonomous elements of the linear system of ordinary / partial differential equations represented by the matrix **`R`**, a system of differential equations that defines the autonomous elements as functions, and, if possible, the general solutions to these equations.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`v`** is a list or vector of functions which depend on the independent variable of the ODE system. These functions are interpreted as the system variables.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***AutonomousElements*** is the empty list if there exist no autonomous elements of the linear system, or a list of three vectors otherwise.
- If the result is a list of three vectors, then the first vector consists of a system of differential equations which defines a generating set of the autonomous elements of the system.
- The second entry of the result is a vector that gives the autonomous elements of the linear system as functions.
- A vector whose entries define a generating set of the torsion elements expressed in the system variables given by **`v`** is the third entry of the result. The ith generator is given by the right hand side of the equation which is the ith entry of this vector. The left hand side of this equation is θ_i.
- ***AutonomousElementsRat*** performs the same computations as ***AutonomousElements***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- In addition to the third entry of the result, the autonomous equations that the torsion elements satisfy can be obtained by using TorsionElements. The integration of the torsion elements can also be achieved by using IntTorsion.

Examples:

```
> with(OreModules):
```

Example 1:

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l1,l2]):
> R := evalm([[D^2+g/l1, 0, -g/l1], [0, D^2+g/l2, -g/l2]]);
```

$$R := \begin{bmatrix} D^2 + \frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2 + \frac{g}{l2} & -\frac{g}{l2} \end{bmatrix}$$

```
> AutonomousElements(R, [x1(t),x2(t),u(t)], Alg);
```

$$[\,]$$

There are no autonomous elements, i.e., generically, the bipendulum is controllable. However, if the lengths of the two pendula are equal, there are autonomous elements:

```
> Rmod := subs(l2=l1, evalm(R));
```

$$Rmod := \begin{bmatrix} D^2 + \frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2 + \frac{g}{l1} & -\frac{g}{l1} \end{bmatrix}$$

```
> AutonomousElements(Rmod, [x1(t),x2(t),u(t)], Alg);
```

$$\left[\left[g\,\theta_1(t) + l1\left(\frac{d^2}{dt^2}\theta_1(t)\right) = 0\right], \left[\theta_1 = \_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right) + \_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\right], [\theta_1 = x1(t) - x2(t)]\right]$$

Example 2:

System of linear ordinary differential equations describing two pendula mounted on a cart (J. W. Polderman, J. C. Willems, *Introduction to Mathematical Systems Theory. A Behavioral Approach*, TAM 26, Springer, 1998):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[m1,m2,M,L1,L2,g]):
> R := evalm([[m1*L1*D^2, m2*L2*D^2, -1, (M+m1+m2)*D^2],
              [m1*L1^2*D^2-m1*L1*g, 0, 0, m1*L1*D^2],
              [0, m2*L2^2*D^2-m2*L2*g, 0, m2*L2*D^2]]);
```

$$R := \begin{bmatrix} m1\,L1\,D^2 & m2\,L2\,D^2 & -1 & (M + m1 + m2)\,D^2 \\ m1\,L1^2\,D^2 - m1\,L1\,g & 0 & 0 & m1\,L1\,D^2 \\ 0 & m2\,L2^2\,D^2 - m2\,L2\,g & 0 & m2\,L2\,D^2 \end{bmatrix}$$

```
> AutonomousElements(R, [x1(t),x2(t),x3(t),u(t)], Alg);
```

$$[\,]$$

```
> Rmod := subs(L2=L1, evalm(R));
```

$$Rmod := \begin{bmatrix} m1\,L1\,D^2 & D^2\,L1\,m2 & -1 & (M + m1 + m2)\,D^2 \\ m1\,L1^2\,D^2 - m1\,L1\,g & 0 & 0 & m1\,L1\,D^2 \\ 0 & m2\,L1^2\,D^2 - L1\,g\,m2 & 0 & D^2\,L1\,m2 \end{bmatrix}$$

```
> AutonomousElements(Rmod, [x1(t),x2(t),x3(t),u(t)], Alg);
```

$$\left[\begin{bmatrix} m2\,m1\,L1\,g\,\theta_1(t) - L1\,m2\,\theta_3(t) = 0 \\ L1\,m2\,\theta_2(t) + L1\,m2\,\theta_3(t) = 0 \\ -L1\,m2\left(g\,\theta_3(t) - L1\left(\frac{d^2}{dt^2}\theta_3(t)\right)\right) = 0 \end{bmatrix}, \begin{bmatrix} \theta_1 = \dfrac{\_C1\,\mathbf{e}^{\left(\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} + \_C2\,\mathbf{e}^{\left(-\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)}}{m1\,g} \\ \theta_2 = -\_C1\,\mathbf{e}^{\left(\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} - \_C2\,\mathbf{e}^{\left(-\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} \\ \theta_3 = \_C1\,\mathbf{e}^{\left(\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} + \_C2\,\mathbf{e}^{\left(-\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} \end{bmatrix},\right.$$

$$\left[\begin{array}{c}\theta_1 = \mathrm{x1}(t) - \mathrm{x2}(t)\\ \theta_2 = \mathrm{x2}(t)\,g\,m1 + \mathrm{x2}(t)\,g\,m2 - \mathrm{x3}(t) + M\left(\frac{d^2}{dt^2}\mathrm{u}(t)\right)\\ \theta_3 = -\mathrm{x2}(t)\,g\,m1 - \mathrm{x2}(t)\,g\,m2 - \mathrm{x2}(t)\,g\,M + L1\,M\left(\frac{d^2}{dt^2}\mathrm{x2}(t)\right) + \mathrm{x3}(t)\end{array}\right]$$

```
> TorsionElements(Rmod, [x1(t),x2(t),x3(t),u(t)], Alg);
```

$$\left[\left[\begin{array}{c}-g\,\theta_1(t) + L1\left(\frac{d^2}{dt^2}\theta_1(t)\right) = 0\\ -g\,\theta_2(t) + L1\left(\frac{d^2}{dt^2}\theta_2(t)\right) = 0\\ -g\,\theta_3(t) + L1\left(\frac{d^2}{dt^2}\theta_3(t)\right) = 0\end{array}\right], \left[\begin{array}{c}\theta_1(t) = \mathrm{x1}(t) - \mathrm{x2}(t)\\ \theta_2(t) = \mathrm{x2}(t)\,g\,m1 + \mathrm{x2}(t)\,g\,m2 - \mathrm{x3}(t) + M\left(\frac{d^2}{dt^2}\mathrm{u}(t)\right)\\ \theta_3(t) = -\mathrm{x2}(t)\,g\,m1 - \mathrm{x2}(t)\,g\,m2 - \mathrm{x2}(t)\,g\,M + L1\,M\left(\frac{d^2}{dt^2}\mathrm{x2}(t)\right) + \mathrm{x3}(t)\end{array}\right]\right]$$

See Also:

DefineOreAlgebra, IntTorsion, TorsionElements, Parametrization, MinimalParametrization, Exti, Extn, Torsion, FirstIntegral, PiPolynomial.

OreModules[BoundaryTerms] - return the boundary terms of an integration by parts

Calling Sequence:

BoundaryTerms(z,P,y,Alg)

Parameters:

z	-	function or list or vector of functions
P	-	element of **Alg** or matrix with entries in **Alg**
y	-	function or list or vector of functions
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***BoundaryTerms*** returns an expression in terms of the (entries of the) given **z** and **y** which allows to determine the boundary terms obtained by integrating by parts the product **z P y**, where the second multiplication is the application of the (matrix of) ordinary differential operator(s) in **Alg** to **y** and the first multiplication is just multiplication of functions on the left.
- The boundary terms of the integration by parts of **z P y** are obtained as the difference of the substitutions of the upper and lower bound of the range of integration into the result of ***BoundaryTerms*** (see the examples below).
- **P** is an element of the Ore algebra **Alg** of ordinary differential operators or a matrix with entries in **Alg**.
- For the product **z P y** to be defined, **z** and **y** must be functions of the left indeterminate (i.e. independent variable) of **Alg** and **P** an element of **Alg**, or **z** and **y** must be lists or vectors of functions of the left indeterminate of **Alg** and **P** a matrix with entries in **Alg**. In the latter case, the length of **z** must equal the number of rows of **P** and the length of **y** must equal the number of columns of **P**.
- **Alg** is expected to be defined using DefineOreAlgebra.
- The result of ***BoundaryTerms*** is a sum of certain products of the entries of **z** and **y** and their derivatives.
- ***BoundaryTerms*** is used in LQEquations to determine boundary terms which are introduced by integration by parts when computing the Euler-Lagrange equations.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
> P := D;
```

$$P := D$$

```
> B := BoundaryTerms(f(t), P, g(t), Alg);
```

$$B := f(t)\, g(t)$$

The integral which is considered in this example is the following:

```
> int(f(t)*diff(g(t), t), t=a..b);
```

$$\int_a^b f(t) \left(\frac{d}{dt} g(t)\right) dt$$

The boundary terms of the integration by parts of $f(t)\left(\frac{d}{dt} g(t)\right)$ are obtained by substituting the upper and lower bound of the range of integration into B and taking the difference:

```
> subs(t=b, B)-subs(t=a, B);
```

$$f(b)\, g(b) - f(a)\, g(a)$$

Example 2:

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
> z := vector([f1(t), f2(t)]);
```

$$z := [\mathrm{f1}(t), \mathrm{f2}(t)]$$

```
> P := evalm([[D, 1], [D^2, 0]]);
```

$$P := \begin{bmatrix} \mathrm{D} & 1 \\ \mathrm{D}^2 & 0 \end{bmatrix}$$

```
> y := vector([g1(t), g2(t)]);
```

$$y := [\mathrm{g1}(t), \mathrm{g2}(t)]$$

```
> B := BoundaryTerms(z, P, y, Alg);
```

$$B := \mathrm{f1}(t)\,\mathrm{g1}(t) + \mathrm{f2}(t)\left(\frac{d}{dt}\mathrm{g1}(t)\right) - \left(\frac{d}{dt}\mathrm{f2}(t)\right)\mathrm{g1}(t)$$

The integral which is considered in this example is the following:

```
> int(evalm(z &* ApplyMatrix(P, y, Alg))[1], t=a..b);
```

$$\int_a^b \mathrm{f1}(t)\left(\left(\frac{d}{dt}\mathrm{g1}(t)\right) + \mathrm{g2}(t)\right) + \mathrm{f2}(t)\left(\frac{d^2}{dt^2}\mathrm{g1}(t)\right) dt$$

The boundary terms of the integration by parts of the previous integrand are obtained by substituting the upper and lower bound of the range of integration into B and taking the difference:

```
> subs(t=b, B)-subs(t=a, B);
```

$$\mathrm{f1}(b)\,\mathrm{g1}(b) + \mathrm{f2}(b)\left(\frac{d}{db}\mathrm{g1}(b)\right) - \left(\frac{d}{db}\mathrm{f2}(b)\right)\mathrm{g1}(b) - \mathrm{f1}(a)\,\mathrm{g1}(a) - \mathrm{f2}(a)\left(\frac{d}{da}\mathrm{g1}(a)\right) + \left(\frac{d}{da}\mathrm{f2}(a)\right)\mathrm{g1}(a)$$

See Also:

DefineOreAlgebra, Mult, ApplyMatrix, Involution, LQEquations, FinalConditions, ControllabilityMatrix, Brunovsky, KalmanSystem, TorsionElements.

OreModules[Brunovsky],

OreModules[BrunovskyRat] - **find transformation of controllable linear ODE system to Brunovsky canonical form**

Calling Sequence:

Brunovsky(R,Alg)
BrunovskyRat(R,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***Brunovsky*** returns a matrix which defines a transformation of the system variables such that the given controllable linear system of ordinary differential equations for these system variables transforms to Brunovsky canonical form (or canonical controller form, see e.g. E. D. Sontag, *Mathematical Control Theory*, Springer, 2nd edition, 1998, or T. Kailath, *Linear Systems*, Prentice-Hall, 1980).
- **`R`** is a matrix with entries in the Ore algebra **`Alg`** representing a linear system of ordinary differential equations.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The variable transformation defined by the matrix which is returned by ***Brunovsky*** brings the given linear system to the form $\frac{d}{dt}\mathrm{x}(t) = A\,\mathrm{x}(t) + B\,\mathrm{u}(t)$, where A is a block diagonal matrix with each non-zero block a companion matrix with zero last row, and B is partitioned according to the block structure of A such that every block at position (i, i) in this structure is a column whose only non-zero component is the last one which is 1 (see Example 2 below; see also E. D. Sontag, *Mathematical Control Theory*, Springer, 2nd edition, 1998, p. 191).
- ***BrunovskyRat*** performs the same computations as ***Brunovsky***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1: converting a second order ODE to a first order system

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
```

We consider the following linear ordinary differential equation:

```
> R := matrix([[D^2, -1]]);
```

$$R := \begin{bmatrix} \mathrm{D}^2 & -1 \end{bmatrix}$$

```
> ApplyMatrix(R, [x(t),u(t)], Alg)=evalm([[0]]);
```

$$\left[\left(\frac{d^2}{dt^2}\mathrm{x}(t)\right) - \mathrm{u}(t)\right] = \begin{bmatrix} 0 \end{bmatrix}$$

Using ***Brunovsky***, we find a transformation of the system variables that brings the system to Brunovsky canonical form:

```
> T := Brunovsky(R, Alg);
```

$$T := \begin{bmatrix} 1 & 0 \\ \mathrm{D} & 0 \\ 0 & 1 \end{bmatrix}$$

In terms of the system variables $\mathrm{x}(t)$, $\mathrm{u}(t)$ and the new variables $\mathrm{z1}(t)$, $\mathrm{z2}(t)$, $\mathrm{v}(t)$, this transformation can be written as follows:

```
> matrix(3,1,[z1(t),z2(t),v(t)])=ApplyMatrix(T, [x(t),u(t)], Alg);
```

$$\begin{bmatrix} z1(t) \\ z2(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} x(t) \\ \frac{d}{dt} x(t) \\ u(t) \end{bmatrix}$$

```
> S := linalg[stackmatrix](T, R):
```

To find the Brunovsky canonical form satisfied by the new variables z1(*t*), z2(*t*), v(*t*), we solve *S* (x(*t*), u(*t*))^T = (z1(*t*), z2(*t*), v(*t*)) for (x(*t*), u(*t*))^T:

```
> E := Elimination(S, [x,u], [z1,z2,v,0], Alg):
> ApplyMatrix(E[1], [x(t),u(t)], Alg)=ApplyMatrix(E[2], [z1(t),z2(t),v(t)], Alg);
```

$$\begin{bmatrix} 0 \\ 0 \\ u(t) \\ x(t) \end{bmatrix} = \begin{bmatrix} -\left(\frac{d}{dt} z2(t)\right) + v(t) \\ -\left(\frac{d}{dt} z1(t)\right) + z2(t) \\ v(t) \\ z1(t) \end{bmatrix}$$

Example 2:

Linear system of ordinary differential equations describing a satellite in a circular equatorial orbit (see T. Kailath, *Linear Systems*, Prentice-Hall, 1980, p. 60):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], polynom=[t], comm=[omega,m,r,a,b]):
> R := evalm([[Dt,-1,0,0,0,0], [-3*omega^2,Dt,0,-2*omega*r,-a/m,0], [0,0,Dt,-1,0,0],
  [0,2*omega/r,0,Dt,0,-b/(m*r)]]);
```

$$R := \begin{bmatrix} Dt & -1 & 0 & 0 & 0 & 0 \\ -3\omega^2 & Dt & 0 & -2\omega r & -\frac{a}{m} & 0 \\ 0 & 0 & Dt & -1 & 0 & 0 \\ 0 & \frac{2\omega}{r} & 0 & Dt & 0 & -\frac{b}{m r} \end{bmatrix}$$

```
> ApplyMatrix(R, [seq(x[i](t),i=1..4),u1(t),u2(t)], Alg)=evalm([[0],[0],[0],[0]]);
```

$$\begin{bmatrix} \left(\frac{d}{dt} x_1(t)\right) - x_2(t) \\ -\frac{3\omega^2 x_1(t) m - \left(\frac{d}{dt} x_2(t)\right) m + 2\omega r x_4(t) m + a\, u1(t)}{m} \\ \left(\frac{d}{dt} x_3(t)\right) - x_4(t) \\ \frac{2\omega x_2(t) m + \left(\frac{d}{dt} x_4(t)\right) m r - b\, u2(t)}{m r} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

A transformation of the system variables that brings the system to Brunovsky canonical form is:

```
> T := Brunovsky(R, Alg);
```

$$T := \begin{bmatrix} \frac{1}{b\,a} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{b\,a} & 0 & 0 & 0 & 0 \\ \frac{3\,\omega^2}{a\,b} & 0 & 0 & \frac{2\,\omega\,r}{a\,b} & \frac{1}{b\,m} & 0 \\ 0 & 0 & \frac{1}{b\,a} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{b\,a} & 0 & 0 \\ 0 & -\frac{2\,\omega}{a\,b\,r} & 0 & 0 & 0 & \frac{1}{m\,a\,r} \end{bmatrix}$$

```
> evalm([[z[1](t)],[z[2](t)],[v[1](t)],[z[3](t)],[z[4](t)],[v[2](t)]])=ApplyMatrix(T,
  [seq(x[i](t),i=1..4),u1(t),u2(t)], Alg);
```

$$\begin{bmatrix} z_1(t) \\ z_2(t) \\ v_1(t) \\ z_3(t) \\ z_4(t) \\ v_2(t) \end{bmatrix} = \begin{bmatrix} \frac{x_1(t)}{b\,a} \\ \frac{x_2(t)}{b\,a} \\ \frac{3\,\omega^2\,x_1(t)\,m + 2\,\omega\,r\,x_4(t)\,m + a\,\mathrm{u1}(t)}{m\,b\,a} \\ \frac{x_3(t)}{b\,a} \\ \frac{x_4(t)}{b\,a} \\ -\frac{2\,\omega\,x_2(t)\,m - b\,\mathrm{u2}(t)}{b\,a\,m\,r} \end{bmatrix}$$

```
> S := linalg[stackmatrix](T, R):
> E := Elimination(S, [seq(x[i],i=1..4),u1,u2], [z[1],z[2],v[1],z[3],z[4],v[2],0,0,0,0],
  Alg):
> C := ApplyMatrix(E[2], [[z[1](t)],[z[2](t)],[v[1](t)],[z[3](t)],[z[4](t)],[v[2](t)]],
  Alg):
```

The first four equations in the following system are the equations satisfied by the new variables z1(t), z2(t), z3(t), z4(t), v1(t), v2(t), and the last six equations give the inverse transformation to T:

```
> ApplyMatrix(E[1], [seq(x[i](t),i=1..4),u1(t),u2(t)], Alg)=evalm(C);
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \mathrm{u2}(t) \\ \mathrm{u1}(t) \\ x_4(t) \\ x_3(t) \\ x_2(t) \\ x_1(t) \end{bmatrix} = \begin{bmatrix} -\left(\frac{d}{dt} z_4(t)\right) + v_2(t) \\ -\left(\frac{d}{dt} z_3(t)\right) + z_4(t) \\ -\left(\frac{d}{dt} z_2(t)\right) + v_1(t) \\ -\left(\frac{d}{dt} z_1(t)\right) + z_2(t) \\ 2\, a\, \omega\, m\, z_2(t) + a\, m\, r\, v_2(t) \\ -3\, b\, \omega^2\, m\, z_1(t) + b\, m\, v_1(t) - 2\, b\, \omega\, r\, m\, z_4(t) \\ b\, a\, z_4(t) \\ b\, a\, z_3(t) \\ b\, a\, z_2(t) \\ b\, a\, z_1(t) \end{bmatrix}$$

If we define the matrices *A* and *B* as follows, then it is clearly visible that the new variables satisfy a Brunovsky canonical form:

```
> A := matrix(4,4,[0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0]); B := matrix(4,2,[0,0,1,0,0,0,0,1]);
```

$$A := \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B := \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> ApplyMatrix(linalg[diag](Dt$4), [z1(t),z2(t),z3(t),z4(t)], Alg)=ApplyMatrix(A,
  [z1(t),z2(t),z3(t),z4(t)], Alg)+ApplyMatrix(B, [v1(t),v2(t)], Alg);
```

$$\begin{bmatrix} \frac{d}{dt}\mathrm{z1}(t) \\ \frac{d}{dt}\mathrm{z2}(t) \\ \frac{d}{dt}\mathrm{z3}(t) \\ \frac{d}{dt}\mathrm{z4}(t) \end{bmatrix} = \begin{bmatrix} \mathrm{z2}(t) \\ 0 \\ \mathrm{z4}(t) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \mathrm{v1}(t) \\ 0 \\ \mathrm{v2}(t) \end{bmatrix}$$

Example 3:

Linearized system of ordinary differential equations describing two pendula mounted on a cart (see J. W. Polderman, J. C. Willems, *Introduction to Mathematical Systems Theory. A Behavioral Approach*, Springer, 1998, p. 159-160):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], polynom=[t], comm=[m1, m2, M, L1, L2, g]):
> R := evalm([[m1*L1*Dt^2, m2*L2*Dt^2,(M+m1+m2)*Dt^2,-1],
             [m1*L1^2*Dt^2-m1*L1*g, 0, m1*L1*Dt^2,0],
             [0, m2*L2^2*Dt^2-m2*L2*g, m2*L2*Dt^2,0]]);
```

$$R := \begin{bmatrix} m1\,L1\,Dt^2 & m2\,L2\,Dt^2 & (M+m1+m2)\,Dt^2 & -1 \\ m1\,L1^2\,Dt^2 - m1\,L1\,g & 0 & m1\,L1\,Dt^2 & 0 \\ 0 & m2\,L2^2\,Dt^2 - m2\,L2\,g & m2\,L2\,Dt^2 & 0 \end{bmatrix}$$

In case the lengths $L1$ and $L2$ of the two pendula are not equal, we obtain the following transformation of the system variables that brings the system to Brunovsky canonical form:

```
> T := Brunovsky(R, Alg);
```

$$T := \begin{bmatrix} \frac{L1^2}{g^2\,(-L2+L1)} & -\frac{L2^2}{g^2\,(-L2+L1)} & \frac{1}{g^2} & 0 \\ \frac{Dt\,L1^2}{g^2\,(-L2+L1)} & -\frac{Dt\,L2^2}{g^2\,(-L2+L1)} & \frac{Dt}{g^2} & 0 \\ \frac{L1}{g\,(-L2+L1)} & -\frac{L2}{g\,(-L2+L1)} & 0 & 0 \\ \frac{L1\,Dt}{g\,(-L2+L1)} & -\frac{L2\,Dt}{g\,(-L2+L1)} & 0 & 0 \\ \frac{1}{-L2+L1} & -\frac{1}{-L2+L1} & 0 & 0 \\ \frac{Dt}{-L2+L1} & -\frac{Dt}{-L2+L1} & 0 & 0 \\ \frac{g\,(L2\,M+L2\,m1-m1\,L1)}{M\,L1\,L2\,(-L2+L1)} & -\frac{(M\,L1+L1\,m2-m2\,L2)\,g}{M\,L1\,L2\,(-L2+L1)} & 0 & \frac{1}{L2\,L1\,M} \end{bmatrix}$$

```
> evalm([seq([z[i](t)],i=1..6),[v(t)]])=ApplyMatrix(T, [x1(t),x2(t),x3(t),u(t)], Alg);
```

$$\begin{bmatrix} z_1(t) \\ z_2(t) \\ z_3(t) \\ z_4(t) \\ z_5(t) \\ z_6(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} \dfrac{L1^2\,x1(t) - L2^2\,x2(t) - x3(t)\,L2 + x3(t)\,L1}{g^2\,(-L2 + L1)} \\ \dfrac{L1^2\left(\frac{d}{dt}x1(t)\right) - L2^2\left(\frac{d}{dt}x2(t)\right) - \left(\frac{d}{dt}x3(t)\right)L2 + \left(\frac{d}{dt}x3(t)\right)L1}{g^2\,(-L2 + L1)} \\ \dfrac{L1\,x1(t) - L2\,x2(t)}{g\,(-L2 + L1)} \\ \dfrac{L1\left(\frac{d}{dt}x1(t)\right) - L2\left(\frac{d}{dt}x2(t)\right)}{g\,(-L2 + L1)} \\ \dfrac{x1(t) - x2(t)}{-L2 + L1} \\ \dfrac{\left(\frac{d}{dt}x1(t)\right) - \left(\frac{d}{dt}x2(t)\right)}{-L2 + L1} \\ \dfrac{g\,x1(t)\,L2\,M + g\,x1(t)\,L2\,m1 - g\,x1(t)\,m1\,L1 - g\,x2(t)\,M\,L1 - g\,x2(t)\,L1\,m2 + g\,x2(t)\,m2\,L2 - u(t)\,L2 + u(t)\,L1}{M\,L1\,L2\,(-L2 + L1)} \end{bmatrix}$$

```
> S := linalg[stackmatrix](T, R):
> F := Elimination(S, [x1,x2,x3,u], [seq(z[i],i=1..6),v,0,0,0], Alg):
```

The Brunovsky canonical form can be read off from the following system of equations:

```
> ApplyMatrix(F[1], [x1(t),x2(t),x3(t),u(t)], Alg)=ApplyMatrix(F[2],
  [seq(z[i](t),i=1..6),v(t)], Alg);
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ u(t) \\ x3(t) \\ x2(t) \\ x1(t) \end{bmatrix} = \begin{bmatrix} -\left(\frac{d}{dt}z_6(t)\right) + v(t) \\ -\left(\frac{d}{dt}z_5(t)\right) + z_6(t) \\ -\left(\frac{d}{dt}z_4(t)\right) + z_5(t) \\ -\left(\frac{d}{dt}z_3(t)\right) + z_4(t) \\ -\left(\frac{d}{dt}z_2(t)\right) + z_3(t) \\ -\left(\frac{d}{dt}z_1(t)\right) + z_2(t) \\ z_3(t)\,g^2\,m2 + z_3(t)\,g^2\,M + z_3(t)\,g^2\,m1 - z_5(t)\,L2\,M\,g - z_5(t)\,L2\,m1\,g - z_5(t)\,g\,L1\,m2 - z_5(t)\,g\,L1\,M + L2\,L1\,M\,v(t) \\ g^2\,z_1(t) - z_3(t)\,L2\,g - z_3(t)\,g\,L1 + L2\,L1\,z_5(t) \\ g\,z_3(t) - L1\,z_5(t) \\ g\,z_3(t) - L2\,z_5(t) \end{bmatrix}$$

See Also:

DefineOreAlgebra, Elimination, AutonomousElements, FirstIntegral, PiPolynomial, IntTorsion, ParticularSolution, ControllabilityMatrix, KalmanSystem, TorsionElements, LQEquations, FinalConditions.

OreModules[Complement],

OreModules[ComplementRat],

OreModules[ComplementConstCoeff],

OreModules[AllComplementsConstCoeff] - **return generating set of torsion elements in terms of the system variables**

Calling Sequence:

Complement(T,R,Alg,d)
ComplementRat(T,R,Alg,d)
ComplementConstCoeff(T,R,Alg)
AllComplementsConstCoeff(T,R,Alg)

Parameters:

`T, R`	-	matrices with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)
`d`	-	(optional) non-negative integer

Description:

- ***Complement***, ***ComplementRat***, ***ComplementConstCoeff***, ***AllComplementsConstCoeff*** solve the matrix equation **T** - **T** *S* **T** = *V* **R** for *S* and *V*, where **T** and **R** are given matrices with entries in the Ore algebra **Alg**.
- ***Complement*** solves the above matrix equation only for those matrices *S* and *V* such that the entries of *S* have degree at most **d** in the operators and at most **d** in the coefficients given in the definition of the Ore algebra **Alg**. If the parameter **d** is not provided, the degree in the operators and coefficients of the entries of *S* are bounded by 1 by default.
- ***ComplementConstCoeff*** is applicable only in the case, where each entry of **T** and **R** has constant coefficients as an operator in **Alg**. Then the above equation is solved only for matrices *S* and *V* whose entries have constant coefficients as operators in **Alg**. However, in this case the degrees of the entries is not bounded. Note that even if ***Complement*** and ***ComplementConstCoeff*** are applied to matrices **T** and **R** over **Alg** whose entries have constant coefficients, in general different solutions are returned by ***Complement*** and ***ComplementConstCoeff***.
- If no solution to the above matrix equation could be found, then the result of these procedures is the empty list. Otherwise ***Complement*** and ***ComplementConstCoeff*** return a list of three matrices with entries in **Alg**, and ***AllComplementsConstCoeff*** returns a list of matrices with entries in **Alg**. In this case the result of ***Complement*** and ***ComplementConstCoeff*** is the list [*I* - *S* **T**, *V*, *S*], where (*S*, *V*) is a particular solution to the above matrix equation and *I* is the identity matrix. The result of ***AllComplementsConstCoeff*** in this case contains matrices *I* - *S* **T**, where *I* is the identity matrix and different matrices are substituted for *S*, namely the first *S* coming from a particular solution to **T** - **T** *S* **T** = *V* **R**, and all remaining matrices *S* being obtained from this particular solution by adding each matrix of a basis of solutions for the corresponding homogeneous linear system of equations.
- These procedures are intended for the following purpose: Let the residue classes of the rows of **T** in the left **Alg**-module *M* presented by **R** be a generating set of the torsion submodule of *M*. That means we consider the left **Alg**-module *M* which is the factor module of the free **Alg**-module of row vectors whose length equals the number of columns of **R** modulo the submodule which is generated by the rows of **R**. (A generating set for its torsion submodule can be obtained e.g. using TorsionElements.) Then, from a each solution (*S*, *V*) or **T** - **T** *S* **T** = *V* **R** we obtain a complement of the torsion submodule in *M*, i.e. a submodule *N* of *M* such that *M* is the direct sum of its torsion submodule and *N*. Given *S*, the residue classes in *M* of the rows of the matrix *U* = *I* - *S* **T**, where *I* is the identity matrix, form a generating set of such a submodule *N*.
- **T** and **R** are matrices with entries in the Ore algebra **Alg** and with the same number of columns.

- **`Alg`** is expected to be defined using DefineOreAlgebra.
- ***ComplementRat*** performs the same computations as ***Complement***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- For more details see A. Quadrat, D. Robertz, "Parametrizing all solutions of uncontrollable multidimensional linear systems", Proceedings of the 16th IFAC World Congress, Prague, 2005.

Examples:

```
> with(OreModules):
```

Example 1: Ordinary differential equations

We study a bipendulum, namely a system composed of a bar where two pendula are fixed. Here we only consider the case, where both pendula have the same length *l*.
For more details, see J.-F. Pommaret, *Partial Differential Control Theory*, Kluwer, 2001, p. 569, and the Library of Examples at http://wwwb.math.rwth-aachen.de/OreModules.

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l]):
> R := evalm([[D^2+g/l, 0, -g/l], [0, D^2+g/l, -g/l]]);
```

$$R := \begin{bmatrix} D^2+\frac{g}{l} & 0 & -\frac{g}{l} \\ 0 & D^2+\frac{g}{l} & -\frac{g}{l} \end{bmatrix}$$

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} D^2 l+g & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 & 0 \\ 0 & D^2 l+g & -g \end{bmatrix}, \begin{bmatrix} g \\ g \\ D^2 l+g \end{bmatrix}\right]$$

```
> TorsionElements(R, [x1(t),x2(t),u(t)], Alg);
```

$$\left[\left[g\,\theta_1(t)+l\left(\frac{d^2}{dt^2}\theta_1(t)\right)=0\right], [\theta_1(t)=x1(t)-x2(t)]\right]$$

The residue classes of the rows of **`T`** in the (left) **`Alg`**-module presented by **`R`** generate its torsion submodule:

```
> T := Ext1[2];
```

$$T := \begin{bmatrix} 1 & -1 & 0 \\ 0 & D^2 l+g & -g \end{bmatrix}$$

```
> C := Complement(T, R, Alg);
```

$$C := \left[\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & l \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}\right]$$

```
> U := C[1]: V := C[2]: S := C[3]:
```

We verify that $\mathbf{T} - \mathbf{T}\,S\,\mathbf{T} = \mathbf{T}\,U = V\,\mathbf{R}$:

```
> simplify(Mult(T, U, Alg) - Mult(V, R, Alg));
```

$$0$$

Since the entries of **`T`** and **`R`** have constant coefficients as operators in **`Alg`**, also ***ComplementConstCoeff*** can be applied:

```
> C := ComplementConstCoeff(T, R, Alg);
```

$$C := \left[\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{D^2 l+g}{g} & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & -\frac{1}{g} \end{bmatrix}\right]$$

```
> A := AllComplementsConstCoeff(T, R, Alg);
```

$$A := \left[\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{D^2 l + g}{g} & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{(-1+g)(D^2 l + g)}{g} & g \end{bmatrix}, \begin{bmatrix} g & 1-g & 0 \\ g & 1-g & 0 \\ 0 & -\frac{(-1+g)(D^2 l + g)}{g} & g \end{bmatrix}, \begin{bmatrix} 0 & 1 + D^2 l + g & -g \\ 0 & 1 & 0 \\ 0 & \frac{D^2 l + g}{g} & 0 \end{bmatrix}, \right.$$

$$\left. \begin{bmatrix} 0 & 1 + D^2 l g + g^2 & -g^2 \\ 0 & 1 + g(D^2 l + g) & -g^2 \\ 0 & \frac{(1 + D^2 l g + g^2)(D^2 l + g)}{g} & -D^2 l g - g^2 \end{bmatrix}, \begin{bmatrix} g & 1-g & 0 \\ g & 1-g & 0 \\ D^2 l + g & -\frac{D^2 l g + g^2 - D^2 l - g}{g} & 0 \end{bmatrix}, \begin{bmatrix} D^2 l + g & 1 & -g \\ 0 & 1 & 0 \\ 0 & \frac{D^2 l + g}{g} & 0 \end{bmatrix} \right]$$

Every matrix in the list A defines a complement of the torsion submodule of the (left) **Alg**-module presented by **R**.

```
> map(a->Factorize(Mult(T, a, Alg), R, Alg), A);
```

$$\left[\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & l g \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ l g & 0 \end{bmatrix}, \begin{bmatrix} 0 & l \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} l & 0 \\ 0 & 0 \end{bmatrix}\right]$$

Example 2: Differential time-delay systems

Linear differential time-delay system describing a flexible rod (see H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD thesis, University of Orsay, France, 1995):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  shift_action=[delta,t,h]):
> R := evalm([[Dt, -Dt*delta, -1], [2*Dt*delta, -Dt-Dt*delta^2, 0]]);
```

$$R := \begin{bmatrix} Dt & -Dt\,\delta & -1 \\ 2\,Dt\,\delta & -Dt - Dt\,\delta^2 & 0 \end{bmatrix}$$

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} Dt & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -2\,\delta & 1+\delta^2 & 0 \\ -Dt & Dt\,\delta & 1 \\ Dt\,\delta & -Dt & \delta \end{bmatrix}, \begin{bmatrix} 1+\delta^2 \\ 2\,\delta \\ Dt - Dt\,\delta^2 \end{bmatrix}\right]$$

```
> TorsionElements(R, [y1(t),y2(t),u(t)], Alg);
```

$$[[D(\theta_1)(t) = 0], [\theta_1(t) = -2\,y1(t-h) + y2(t) + y2(t-2\,h)]]$$

The residue classes of the rows of **T** in the (left) **Alg**-module presented by **R** generate its torsion submodule:

```
> T := Ext1[2];
```

$$T := \begin{bmatrix} -2\,\delta & 1+\delta^2 & 0 \\ -Dt & Dt\,\delta & 1 \\ Dt\,\delta & -Dt & \delta \end{bmatrix}$$

```
> C := Complement(T, R, Alg);
```

$$C := \left[\begin{bmatrix} 1+\delta^2 & -\frac{(1+\delta^2)\,\delta}{2} & 0 \\ 2\,\delta & -\delta^2 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ -1 & \frac{\delta}{2} \\ -\delta & \frac{\delta^2}{2} \end{bmatrix}, \begin{bmatrix} \frac{\delta}{2} & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right]$$

```
> U := C[1]: V := C[2]: S := C[3]:
```

We verify that **T** - **T** S **T** = **T** $U = V$ **R**:

```
> simplify(Mult(T, U, Alg) - Mult(V, R, Alg));
```

$$0$$

Since the entries of **T** and **R** have constant coefficients as operators in **Alg**, also ***ComplementConstCoeff*** can be applied:

```
> C := ComplementConstCoeff(T, R, Alg);
```

$$C := \left[\begin{bmatrix} 1+\delta^2 & -\frac{(1+\delta^2)\delta}{2} & 0 \\ 2\delta & -\delta^2 & 0 \\ Dt - Dt\,\delta^2 & -\frac{1}{2}Dt\,\delta + \frac{1}{2}Dt\,\delta^3 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} \frac{\delta}{2} & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -\frac{\delta^2}{2}+1 & \frac{\delta}{2} \end{bmatrix}\right]$$

```
> U := C[1]: V := C[2]: S := C[3]:
> simplify(Mult(T, U, Alg) - Mult(V, R, Alg));
```

$$0$$

See Also:

DefineOreAlgebra, Parametrization, IntTorsion, ParticularSolution, Factorize, MinimalParametrization, TorsionElements, AutonomousElements, Exti, Extn, Torsion.

OreModules[Connection] - return matrix representations of left multiplication maps on a finite dimensional factor module over an Ore algebra

Calling Sequence:

Connection(R,Alg)

Parameters:

R	-	matrix with entries in **Alg**
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***Connection*** returns the list of matrices which represent the left multiplication maps by indeterminates of **Alg** on the finite dimensional left module which is presented by **R** with respect to the vector space basis returned by KBasis.
- Independently of the definition of the coefficient domain of **Alg**, ***Connection*** uses the Ore algebra which is obtained from **Alg** by replacing the coefficient domain by its quotient field, i.e. rational functions. The residue class module which is considered by ***Connection***, namely the module presented by **R**, is the factor module of the free module of row vectors over this Ore algebra whose length equals the number of columns of **R** modulo the submodule which is generated by the rows of **R**.
- The vector space endomorphisms of the module presented by **R** defined by left multiplication by the (remaining) non-invertible indeterminates are represented as matrices with respect to the basis constructed by KBasis, i.e. the i-th row of the j-th resulting matrix is the coefficient row vector of the product of the j-th indeterminate times the i-th element in the vector space basis returned by KBasis with respect to this basis.
- If the factor module is the zero module, then ***Connection*** returns a list of zero times zero matrices.
- **R** is a matrix with entries in the Ore algebra **Alg**.
- **Alg** is expected to be defined using DefineOreAlgebra.
- ***Connection*** returns a list of square matrices with entries in the quotient field of the coefficient domain of **Alg**.
- Note that for ***Connection***, in the same way as for ***KBasis***, the domain of coefficients of the Ore algebra **Alg** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[D1, 0], [D2, D1], [0, D2]]);
```

$$R := \begin{bmatrix} D1 & 0 \\ D2 & D1 \\ 0 & D2 \end{bmatrix}$$

```
> B := KBasis(R, Alg);
```

$$B := [\lambda_1, \lambda_2, \lambda_2\, D1]$$

```
> Connection(R, Alg);
```

$$\left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right]$$

Multiplication by D1 on the residue class module in terms of the vector space basis B:

```
> map(a->Mult(D1, a, Alg), B);
```

$$[\lambda_1\,\mathrm{D1}, \lambda_2\,\mathrm{D1}, \lambda_2\,\mathrm{D1}^2]$$

```
> map(a->[coeff(a, B[1]), coeff(a, B[2])], %);
```

$$[[\mathrm{D1}, 0], [0, \mathrm{D1}], [0, \mathrm{D1}^2]]$$

```
> map(a->ReduceMatrix([a], R, Alg), %);
```

$$[[\ ], [0 \quad \mathrm{D1}], [\ ]]$$

Hence, only the product of D1 by the second basis vector in *B* is non-zero, and the coefficient row vector of this product has only one non-zero coefficient, which is 1.

Multiplication by D2 on the residue class module in terms of the vector space basis *B*:

```
> map(a->Mult(D2, a, Alg), B);
```

$$[\lambda_1\,\mathrm{D2}, \lambda_2\,\mathrm{D2}, \mathrm{D2}\,\lambda_2\,\mathrm{D1}]$$

```
> map(a->[coeff(a, B[1]), coeff(a, B[2])], %);
```

$$[[\mathrm{D2}, 0], [0, \mathrm{D2}], [0, \mathrm{D2}\,\mathrm{D1}]]$$

```
> map(a->ReduceMatrix([a], R, Alg), %);
```

$$[[0 \quad -\mathrm{D1}], [\ ], [\ ]]$$

Hence, only the product of D2 by the first basis vector in *B* is non-zero, and the coefficient row vector of this product has only one non-zero coefficient, which is -1.

Example 2:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := matrix([[D1^3, x1], [D2, x1+D1], [D3, D2]]);
```

$$R := \begin{bmatrix} \mathrm{D1}^3 & x1 \\ \mathrm{D2} & x1+\mathrm{D1} \\ \mathrm{D3} & \mathrm{D2} \end{bmatrix}$$

```
> KBasis(R, Alg);
```

$$[\lambda_1, \mathrm{D1}\,\lambda_1, \mathrm{D1}^2\,\lambda_1, \lambda_2, \lambda_2\,\mathrm{D2}, \lambda_2\,\mathrm{D1}, \mathrm{D1}\,\mathrm{D2}\,\lambda_2, \mathrm{D1}^2\,\lambda_2, \lambda_2\,\mathrm{D1}^3]$$

```
> Connection(R, Alg);
```

$$\left[\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -x1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & x1 & 0 & 0 & -3 & -x1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & -x1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -x1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & -x1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},\right.$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}]$$

Example 3:

```
> R := matrix([[D1^3], [D2+x3], [D3+D1]]);
```

$$R := \begin{bmatrix} \mathrm{D1}^3 \\ \mathrm{D2} + x3 \\ \mathrm{D3} + \mathrm{D1} \end{bmatrix}$$

```
> KBasis(R, Alg);
```

[ ]

```
> Connection(R, Alg);
```

[[ ], [ ], [ ]]

See Also:

DefineOreAlgebra, KBasis, HilbertSeries, Dimension, OreRank, Factorize, Quotient, ReduceMatrix, Elimination, Integrability, Involution, SyzygyModule.

OreModules[ControllabilityMatrix] - return controllability matrix of a linear ODE system

Calling Sequence:

ControllabilityMatrix(F,G,k,Alg)

Parameters:

`F`	-	square matrix with entries in **`Alg`**
`G`	-	matrix with entries in **`Alg`**
`k`	-	natural number
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***ControllabilityMatrix*** returns the controllability matrix for the Kalman system $\frac{d}{dt}\mathrm{x}(t) = F\,\mathrm{x}(t) + G\,\mathrm{u}(t)$. More generally, ***ControllabilityMatrix*** returns the matrix formed by juxtaposing $G_0(t)$, $G_1(t)$, ..., $G_{k-1}(t)$, where $G_0 = \mathrm{G}(t)$ and $G_{i+1}(t) = \mathrm{F}(t)\,\mathrm{G}(t) - \left(\frac{d}{dt} G_i(t)\right)$.
- **`F`** and **`G`** are matrices with entries in the Ore algebra **`Alg`**, where **`F`** is expected to be a square matrix and the number of rows of **`G`** equals the number of columns of **`F`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- If **`F`** is an (n x n)-matrix and **`G`** is an (n x m)-matrix, then the result is an (n x $k\,m$)-matrix.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
```

Example 1:

Consider the Kalman system $\frac{d}{dt}\mathrm{x}(t) = F\,\mathrm{x}(t) + G\,\mathrm{u}(t)$, where:

```
> F := matrix(2,2,[1,-1,0,1]);
```

$$F := \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

```
> G := matrix(2,1,[0,1]);
```

$$G := \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

```
> C := ControllabilityMatrix(F, G, 2, Alg);
```

$$C := \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}$$

```
> linalg[rank](C);
```

$$2$$

Since C has full rank, the above Kalman system is controllable.

Example 2:

Consider the linear ODE system $\frac{d}{dt}\mathrm{x}(t) = \mathrm{F}(t)\,\mathrm{x}(t) + \mathrm{G}(t)\,\mathrm{u}(t)$, where:

```
> F := matrix(2,2,[2,t-1,-1,1]);
```

$$F := \begin{bmatrix} 2 & t-1 \\ -1 & 1 \end{bmatrix}$$

```
> G := matrix(2,1,[t,1-t]);
```

$$G := \begin{bmatrix} t \\ 1-t \end{bmatrix}$$

```
> C := ControllabilityMatrix(F, G, 2, Alg);
```

$$C := \begin{bmatrix} t & 4t-t^2-2 \\ 1-t & -2t+2 \end{bmatrix}$$

```
> subs(t=1, evalm(C));
```

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

Hence, C does not have full rank for $t = 1$. Here we have to consider the larger controllability matrix:

```
> C := ControllabilityMatrix(F, G, 3, Alg);
```

$$C := \begin{bmatrix} t & 4t-t^2-2 & 14t-4t^2-10 \\ 1-t & -2t+2 & -6t+t^2+6 \end{bmatrix}$$

```
> linalg[rank](subs(t=1, evalm(C)));
```

$$2$$

See Also:

DefineOreAlgebra, Mult, ApplyMatrix, Involution, KalmanSystem, TorsionElements, LQEquations, FinalConditions.

OreModules[DefineOreAlgebra] - define an Ore algebra for the current session of OreModules

Calling Sequence:

DefineOreAlgebra($t_1 = l_1$, ... , $t_n = l_n$, options)

Parameters:

t_i	-	types of commutation
l_i	-	lists of indeterminates whose lengths are determined by the corresponding t_i
`options`	-	(optional) options

Description:

- ***DefineOreAlgebra*** sets up a data structure representing an Ore algebra for the current session of OreModules. It extends the command Ore_algebra[skew_algebra] so that most of its parameters and options are the same as in Ore_algebra[skew_algebra].
- Most of the commands in OreModules take one parameter **`Alg`** which is expected to be a result of the ***DefineOreAlgebra*** command.
- The result of ***DefineOreAlgebra*** is a list whose first entry is the result of Ore_algebra[skew_algebra] called with the same parameters and all options as given to ***DefineOreAlgebra*** except for "shift_action". The other entries in the resulting list collect information about the Ore algebra defined by the first entry.
- For the possible types of commutation and possible options, see Ore_algebra[commutation_rules] resp. Ore_algebra[declaration_options].
- ***DefineOreAlgebra*** accepts an additional option "shift_action" which specifies how (matrices of) shift and advance operators are applied to (vectors of) functions. The string "shift_action" is expected as left hand side of an equation whose right hand side is a list with three entries. The first entry specifies an indeterminate δ declared in a preceding argument of ***DefineOreAlgebra*** which represents a shift or advance operator. The second entry sets the indeterminate t on which the previous indeterminate δ acts. Here, t may be different from the indeterminate which was declared together with δ. The third entry defines the length of the shift resp. advance. This option effects the result of ApplyMatrix.

Examples:

```
> with(OreModules):
```

Example 1:

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> Alg1 := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l1,l2]):
> R1 := evalm([[D^2+g/l1, 0, -g/l1], [0, D^2+g/l2, -g/l2]]);
```

$$R1 := \begin{bmatrix} D^2 + \frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2 + \frac{g}{l2} & -\frac{g}{l2} \end{bmatrix}$$

```
> Mult(D, t, Alg1);
```

$$1 + D\,t$$

```
> ApplyMatrix(R1, [x1(t),x2(t),u(t)], Alg1);
```

$$\begin{bmatrix} \dfrac{\left(\frac{d^2}{dt^2}\,x1(t)\right) l1 + g\,x1(t) - g\,u(t)}{l1} \\ -\dfrac{-\left(\frac{d^2}{dt^2}\,x2(t)\right) l2 - g\,x2(t) + g\,u(t)}{l2} \end{bmatrix}$$

Example 2:

Linear differential time-delay system describing a flexible rod (see H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD thesis, University of Orsay, France, 1995):

```
> Alg2 := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  shift_action=[delta,t,h]):
> R2 := evalm([[Dt, -Dt*delta, -1], [2*Dt*delta, -Dt-Dt*delta^2, 0]]);
```

$$R2 := \begin{bmatrix} Dt & -Dt\,\delta & -1 \\ 2\,Dt\,\delta & -Dt - Dt\,\delta^2 & 0 \end{bmatrix}$$

```
> ApplyMatrix(R2, [y1(t),y2(t),u(t)], Alg2);
```

$$\begin{bmatrix} \mathrm{D}(y1)(t) - \mathrm{D}(y2)(t-h) - \mathrm{u}(t) \\ 2\,\mathrm{D}(y1)(t-h) - \mathrm{D}(y2)(t) - \mathrm{D}(y2)(t-2\,h) \end{bmatrix}$$

Example 3:

Linear system of PDEs that appears in mathematical physics, namely in the study of Lie-Poisson structures (see C. M. Bender, G. V. Dunne, L. R. Mead, *Underdetermined systems of partial differential equations*, Journal of Mathematical Physics, vol. 41, no. 9 (2000), pp. 6388-6398 and W. M. Seiler, *Involution analysis of the partial differential equations characterising Hamiltonian vector fields*, Journal of Mathematical Physics, vol. 44 (2003), pp. 1173-1182):

```
> Alg3 := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R3 := evalm([[x1*D3, x2*D3, 0], [-x1*D2+x2*D1, -1, x2*D3], [-1, -x2*D1+x1*D2, x1*D3]]);
```

$$R3 := \begin{bmatrix} x1\,\mathrm{D3} & x2\,\mathrm{D3} & 0 \\ -x1\,\mathrm{D2} + x2\,\mathrm{D1} & -1 & x2\,\mathrm{D3} \\ -1 & -x2\,\mathrm{D1} + x1\,\mathrm{D2} & x1\,\mathrm{D3} \end{bmatrix}$$

```
> ApplyMatrix(R3, [F(x1,x2,x3),G(x1,x2,x3),H(x1,x2,x3)], Alg3);
```

$$\begin{bmatrix} x1\left(\frac{\partial}{\partial x3}\mathrm{F}(x1,x2,x3)\right) + x2\left(\frac{\partial}{\partial x3}\mathrm{G}(x1,x2,x3)\right) \\ -x1\left(\frac{\partial}{\partial x2}\mathrm{F}(x1,x2,x3)\right) + x2\left(\frac{\partial}{\partial x1}\mathrm{F}(x1,x2,x3)\right) - \mathrm{G}(x1,x2,x3) + x2\left(\frac{\partial}{\partial x3}\mathrm{H}(x1,x2,x3)\right) \\ -\mathrm{F}(x1,x2,x3) + x1\left(\frac{\partial}{\partial x2}\mathrm{G}(x1,x2,x3)\right) - x2\left(\frac{\partial}{\partial x1}\mathrm{G}(x1,x2,x3)\right) + x1\left(\frac{\partial}{\partial x3}\mathrm{H}(x1,x2,x3)\right) \end{bmatrix}$$

Example 4:

Linear system involving the Euler operator which occurs in the study of a sphere rolling on a surface (see J. Hadamard, *Sur l'equilibre des plaques elastiques circulaires libres ou appuyees et celui de la sphere isotrope*, Annales scientifiques de l'E. N. S., 3e serie, 18 (1901), pp. 313-342.)

```
> Alg4 := DefineOreAlgebra(euler=[D,rho], polynom=[rho], comm=[lambda,mu]):
> R4 := evalm([[D+1/2,((lambda+mu)/2)*(D-1),1/2,0], [2*D,-(3*lambda+2*mu),D+3,0],
  [-D,lambda,-1,2*mu*(D+1)]]);
```

$$R4 := \begin{bmatrix} \mathrm{D} + \frac{1}{2} & \frac{1}{2}(\lambda+\mu)(\mathrm{D}-1) & \frac{1}{2} & 0 \\ 2\,\mathrm{D} & -3\,\lambda - 2\,\mu & \mathrm{D}+3 & 0 \\ -\mathrm{D} & \lambda & -1 & 2\,\mu\,(\mathrm{D}+1) \end{bmatrix}$$

```
> ApplyMatrix(R4, [theta(rho),sigma(rho),K(rho),G(rho)], Alg4);
```

$$\left[\begin{array}{c} \rho\left(\frac{d}{d\rho}\theta(\rho)\right)+\frac{1}{2}\theta(\rho)+\frac{1}{2}\rho\left(\frac{d}{d\rho}\sigma(\rho)\right)\lambda+\frac{1}{2}\rho\left(\frac{d}{d\rho}\sigma(\rho)\right)\mu-\frac{1}{2}\lambda\sigma(\rho)-\frac{1}{2}\sigma(\rho)\mu+\frac{1}{2}K(\rho) \\ 2\rho\left(\frac{d}{d\rho}\theta(\rho)\right)-3\lambda\sigma(\rho)-2\sigma(\rho)\mu+\rho\left(\frac{d}{d\rho}K(\rho)\right)+3K(\rho) \\ -\rho\left(\frac{d}{d\rho}\theta(\rho)\right)+\lambda\sigma(\rho)-K(\rho)+2\mu\rho\left(\frac{d}{d\rho}G(\rho)\right)+2\mu G(\rho) \end{array}\right]$$

See Also:

OreModules, Ore_algebra[skew_algebra], Ore_algebra[commutation_rules], Ore_algebra[declaration_options], Mult, ApplyMatrix.

OreModules[DiffToOre] - convert a linear (or affine) differential (time-delay) equation to an operator

Calling Sequence:

DiffToOre(L,dvar,Alg)

Parameters:

`L`	-	differential expression or list or vector of differential expressions in the dependent variables
`dvar`	-	list of dependent variables
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***DiffToOre*** converts the linear (or affine) differential (time-delay) equation(s) **`L`** to an operator which is a matrix over the Ore algebra defined by **`Alg`**.
- **`L`** is a differential expression or a list or vector of differential expressions in the functions whose names are provided by the list **`dvar`**. The arguments of these functions may be shifted by constant values, if these shifts are representable by the action of shift operators defined in **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***DiffToOre*** is a list of two matrices. The first matrix has entries in **`Alg`** and represents the linear part of **`L`** as an operator. The second matrix of the result has only one column and consists of differential expressions. It is the difference of **`L`** written as a vector and the linear part of **`L`**.
- This command provides a counterpart to ApplyMatrix and to Ore_to_diff in Ore_algebra. To compose two or more operators, use Mult.

Examples:

```
> with(OreModules):
```

Example 1: Ordinary differential equations

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
> L := [diff(x(t),t,t)-diff(x(t),t)+diff(u(t),t)+a(t), diff(x(t),t)+x(t)-u(t)+b(t)];
```

$$L := \left[\left(\frac{d^2}{dt^2}\mathrm{x}(t)\right)-\left(\frac{d}{dt}\mathrm{x}(t)\right)+\left(\frac{d}{dt}\mathrm{u}(t)\right)+\mathrm{a}(t), \left(\frac{d}{dt}\mathrm{x}(t)\right)+\mathrm{x}(t)-\mathrm{u}(t)+\mathrm{b}(t)\right]$$

```
> M := DiffToOre(L, [x,u], Alg);
```

$$M := \left[\begin{bmatrix} D^2-D & D \\ D+1 & -1 \end{bmatrix}, \begin{bmatrix} \mathrm{a}(t) \\ \mathrm{b}(t) \end{bmatrix}\right]$$

```
> evalm(ApplyMatrix(M[1], [x(t),u(t)], Alg) + M[2]);
```

$$\begin{bmatrix} \left(\frac{d^2}{dt^2}\mathrm{x}(t)\right)-\left(\frac{d}{dt}\mathrm{x}(t)\right)+\left(\frac{d}{dt}\mathrm{u}(t)\right)+\mathrm{a}(t) \\ \left(\frac{d}{dt}\mathrm{x}(t)\right)+\mathrm{x}(t)-\mathrm{u}(t)+\mathrm{b}(t) \end{bmatrix}$$

```
> DiffToOre(L[1], [x,u], Alg);
```

$$[[D^2-D \quad D], \mathrm{a}(t)]$$

Example 2: Partial differential equations

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> L := [diff(y(x1,x2,x3),x1,x2)-diff(z(x1,x2,x3),x2)+diff(u(x1,x2,x3),x1),
  diff(z(x1,x2,x3),x2,x3)-u(x1,x2,x3)];
```

$$L := \left[\left(\frac{\partial^2}{\partial x2\,\partial x1}\, y(x1, x2, x3)\right) - \left(\frac{\partial}{\partial x2}\, z(x1, x2, x3)\right) + \left(\frac{\partial}{\partial x1}\, u(x1, x2, x3)\right), \left(\frac{\partial^2}{\partial x3\,\partial x2}\, z(x1, x2, x3)\right) - u(x1, x2, x3)\right]$$

```
> DiffToOre(L, [y,z,u], Alg);
```

$$\left[\begin{bmatrix} D2\,D1 & -D2 & D1 \\ 0 & D3\,D2 & -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right]$$

Example 3: Differential time-delay systems

```
> Alg := DefineOreAlgebra(diff=[D1,x1], shift=[delta,x2], dual_shift=[tau,x3],
  polynom=[x1,x2,x3], shift_action=[delta,x1], shift_action=[tau,x1]):
> L := [D(y)(x1-1)+z(x1+2)+2*u(x1), (D@@2)(y)(x1+1)-y(x1)-u(x1)];
```

$$L := [D(y)(x1 - 1) + z(x1 + 2) + 2\, u(x1), (D^{(2)})(y)(x1 + 1) - y(x1) - u(x1)]$$

```
> DiffToOre(L, [y,z,u], Alg);
```

$$\left[\begin{bmatrix} D1\,\tau & \delta^2 & 2 \\ D1^2\,\delta - 1 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right]$$

See Also:

DefineOreAlgebra, Ore_algebra[Ore_to_diff], Ore_algebra[skew_algebra], Ore_algebra[commutation_rules], Ore_algebra[declaration_options], Mult, ApplyMatrix, Involution, KroneckerProduct.

OreModules[Dimension],

OreModules[DimensionRat] - **return the Hilbert dimension of a finitely generated module over an Ore algebra**

Calling Sequence:

Dimension(R,Alg)
DimensionRat(R,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***Dimension*** returns the Hilbert dimension of the left module over the Ore algebra **`Alg`** presented by **`R`**. This command extends hilbertdim in Groebner to left modules (however ***Dimension*** always uses the degree-reverse lexicographic termorder tdeg).
- For more details about the Hilbert dimension, see Groebner[hilbertdim].
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- If the left module presented by the matrix **`R`** is the zero module, then -infinity is returned.
- ***DimensionRat*** performs the same computations as ***Dimension***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
```

First we determine the Hilbert dimension of the zero left ideal in **`Alg`**:

```
> R := evalm([[0]]);
```

$$R := [\ 0]$$

```
> Dimension(R, Alg);
```

$$4$$

We replace the domain of coefficients of **`Alg`** by its quotient field, i.e. by the field of rational functions in *x1* and *x2*:

```
> DimensionRat(R, Alg);
```

$$2$$

The Hilbert dimension of the zero module is -infinity:

```
> Dimension([[1]], Alg);
```

$$-\infty$$

Example 2:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := matrix([[D1+x2], [D3+1]]);
```

$$R := \begin{bmatrix} \mathrm{D1}+x2 \\ \mathrm{D3}+1 \end{bmatrix}$$

```
> Dimension(R, Alg);
```

$$4$$

```
> DimensionRat(R, Alg);
```

1

See Also:

DefineOreAlgebra, KBasis, Connection, HilbertSeries, OreRank, Factorize, Quotient, ReduceMatrix, Elimination, Integrability, Involution, SyzygyModule.

OreModules[Elimination],

OreModules[EliminationRat] - eliminate variables in a linear system over an Ore algebra

Calling Sequence:

Elimination(R,v,w,Alg,u)
EliminationRat(R,v,w,Alg,u)

Parameters:

`R`	-	matrix with entries in **`Alg`** or INJ(*n*) or SURJ(*n*) or ZERO, where *n* is a non-negative integer
`v`	-	list of indeterminates
`w`	-	list of indeterminates
`Alg`	-	Ore algebra (given by DefineOreAlgebra)
`u`	-	(optional) sublist of **`v`**

Description:

- ***Elimination*** solves, if possible, the linear system **`R`** $y = z$ for y, where y (resp. z) is the vector whose components are the entries of **`v`** (resp. **`w`**).
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- The number of entries in **`v`** (resp. **`w`**) must equal the number of columns (resp. rows) of **`R`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result is a table T containing two matrices such that $T[1]\,y = T[2]\,z$ is equivalent to **`R`** $y = z$. These matrices are formed by the coefficients in the Groebner basis of the left **`Alg`**-module generated by the left hand sides of **`R`** $y - z = 0$ w.r.t. an elimination order which eliminates the variables **`v`**.
- If **`u`** is given, then **`R`** $y = z$ is solved, if possible, for the indeterminates in **`v`** which are not contained in **`u`**, where y (resp. z) is the vector whose components are the entries of **`v`** (resp. **`w`**).
- ***EliminationRat*** performs the same computations as ***Elimination***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> ivar := x1,x2,x3:
> R := evalm([[D1, 0, -D2], [0, D2, -D2], [D1, 0, D2]]);
```

$$R := \begin{bmatrix} D1 & 0 & -D2 \\ 0 & D2 & -D2 \\ D1 & 0 & D2 \end{bmatrix}$$

```
> ApplyMatrix(R, [y1(ivar),y2(ivar),y3(ivar)],
  Alg)=evalm([[z1(ivar)],[z2(ivar)],[z3(ivar)]]);
```

$$\begin{bmatrix} \left(\frac{\partial}{\partial x1} \mathrm{y1}(x1, x2, x3)\right) - \left(\frac{\partial}{\partial x2} \mathrm{y3}(x1, x2, x3)\right) \\ \left(\frac{\partial}{\partial x2} \mathrm{y2}(x1, x2, x3)\right) - \left(\frac{\partial}{\partial x2} \mathrm{y3}(x1, x2, x3)\right) \\ \left(\frac{\partial}{\partial x1} \mathrm{y1}(x1, x2, x3)\right) + \left(\frac{\partial}{\partial x2} \mathrm{y3}(x1, x2, x3)\right) \end{bmatrix} = \begin{bmatrix} \mathrm{z1}(x1, x2, x3) \\ \mathrm{z2}(x1, x2, x3) \\ \mathrm{z3}(x1, x2, x3) \end{bmatrix}$$

```
> E := Elimination(R, [y1,y2,y3], [z1,z2,z3], Alg);
```

$$E := \mathrm{table}([1 = \begin{bmatrix} 0 & 0 & 2\,\mathrm{D2} \\ 0 & 2\,\mathrm{D2} & 0 \\ 2\,\mathrm{D1} & 0 & 0 \end{bmatrix}, 2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 2 & 1 \\ 1 & 0 & 1 \end{bmatrix}])$$

```
> ApplyMatrix(E[1], [y1(ivar),y2(ivar),y3(ivar)], Alg)=ApplyMatrix(E[2],
  [z1(ivar),z2(ivar),z3(ivar)], Alg);
```

$$\begin{bmatrix} 2\left(\frac{\partial}{\partial x2} \mathrm{y3}(x1, x2, x3)\right) \\ 2\left(\frac{\partial}{\partial x2} \mathrm{y2}(x1, x2, x3)\right) \\ 2\left(\frac{\partial}{\partial x1} \mathrm{y1}(x1, x2, x3)\right) \end{bmatrix} = \begin{bmatrix} -\mathrm{z1}(x1, x2, x3) + \mathrm{z3}(x1, x2, x3) \\ -\mathrm{z1}(x1, x2, x3) + 2\,\mathrm{z2}(x1, x2, x3) + \mathrm{z3}(x1, x2, x3) \\ \mathrm{z1}(x1, x2, x3) + \mathrm{z3}(x1, x2, x3) \end{bmatrix}$$

Example 2: ***Elimination*** computes input-output representation from state space representation

Linear system of ordinary differential equations describing a stirred tank (see H. Kwakernaak & R. Sivan, *Linear Optimal Control Systems*, Wiley-Interscience, 1972).

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[theta,V0,c0,c1,c2]):
```

The state space representation is given by:

```
> R := evalm([[D+1/(2*theta),0,-1,-1],[0,D+1/theta,-(c1-c0)/V0,-(c2-c0)/V0]]);
```

$$R := \begin{bmatrix} \mathrm{D} + \frac{1}{2}\frac{1}{\theta} & 0 & -1 & -1 \\ 0 & \mathrm{D} + \frac{1}{\theta} & -\frac{c1 - c0}{V0} & -\frac{c2 - c0}{V0} \end{bmatrix}$$

```
> ApplyMatrix(R, [x1(t),x2(t),u1(t),u2(t)], Alg);
```

$$\begin{bmatrix} \frac{1}{2}\frac{2\left(\frac{d}{dt}\mathrm{x1}(t)\right)\theta + \mathrm{x1}(t) - 2\,\mathrm{u1}(t)\,\theta - 2\,\mathrm{u2}(t)\,\theta}{\theta} \\ \frac{\left(\frac{d}{dt}\mathrm{x2}(t)\right)\theta\,V0 + \mathrm{x2}(t)\,V0 - \mathrm{u1}(t)\,\theta\,c1 + \mathrm{u1}(t)\,\theta\,c0 - \mathrm{u2}(t)\,\theta\,c2 + \mathrm{u2}(t)\,\theta\,c0}{\theta\,V0} \end{bmatrix}$$

In terms of the states, the output is defined by:

```
> C := evalm([[1/(2*theta),0],[0,1]]);
```

$$C := \begin{bmatrix} \frac{1}{2}\frac{1}{\theta} & 0 \\ 0 & 1 \end{bmatrix}$$

```
> evalm([[y1(t)],[y2(t)]])=ApplyMatrix(C, [x1(t),x2(t)], Alg);
```

$$\begin{bmatrix} \mathrm{y1}(t) \\ \mathrm{y2}(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\frac{\mathrm{x1}(t)}{\theta} \\ \mathrm{x2}(t) \end{bmatrix}$$

To find an input-output representation, we define the following matrix:

```
> Rf := linalg[stackmatrix](R, linalg[augment](C, matrix(2,2,0)));
```

$$Rf := \begin{bmatrix} D+\frac{1}{2}\frac{1}{\theta} & 0 & -1 & -1 \\ 0 & D+\frac{1}{\theta} & -\frac{c1-c0}{V0} & -\frac{c2-c0}{V0} \\ \frac{1}{2}\frac{1}{\theta} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

```
> E := Elimination(Rf, [x1,x2,u1,u2], [0,0,y1,y2], Alg, [u1,u2]);
```

$$E := \text{table}([1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, 2 = \begin{bmatrix} 0 & -V0\,D\,\theta - V0 & \theta\,c1-\theta\,c0 & \theta\,c2-\theta\,c0 \\ -2\,D\,\theta-1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 2\,\theta & 0 & 0 & 0 \end{bmatrix}])$$

```
> ApplyMatrix(E[1], [x1(t),x2(t)], Alg)=ApplyMatrix(E[2], [y1(t),y2(t),u1(t),u2(t)],
  Alg);
```

$$\begin{bmatrix} 0 \\ 0 \\ x2(t) \\ x1(t) \end{bmatrix} = \begin{bmatrix} -\theta\,V0\left(\frac{d}{dt}y2(t)\right) - V0\,y2(t) + u1(t)\,\theta\,c1 - u1(t)\,\theta\,c0 + u2(t)\,\theta\,c2 - u2(t)\,\theta\,c0 \\ -2\,\theta\left(\frac{d}{dt}y1(t)\right) - y1(t) + u1(t) + u2(t) \\ y2(t) \\ 2\,\theta\,y1(t) \end{bmatrix}$$

The first two equations give the input-output behaviour of the system. The third and fourth equations express the state in terms of the output which shows that the system is observable.

Example 3: Application of ***Elimination*** to the study of flatness of linear systems

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l1,l2]):
> R := evalm([[D^2+g/l1, 0, -g/l1], [0, D^2+g/l2, -g/l2]]);
```

$$R := \begin{bmatrix} D^2+\frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2+\frac{g}{l2} & -\frac{g}{l2} \end{bmatrix}$$

Check parametrizability of the system:

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} l1\,D^2+g & 0 & -g \\ 0 & l2\,D^2+g & -g \end{bmatrix}, \begin{bmatrix} D^2\,l2\,g+g^2 \\ D^2\,l1\,g+g^2 \\ D^4\,l2\,l1+D^2\,l2\,g+D^2\,l1\,g+g^2 \end{bmatrix}\right]$$

Since *Ext1*[1] is an identity matrix, the system is (generically) controllable and parametrizable. *Ext1*[3] is a parametrization of the system.

```
> P := Ext1[3];
```

$$P := \begin{bmatrix} D^2\, l2\, g + g^2 \\ D^2\, l1\, g + g^2 \\ D^4\, l2\, l1 + D^2\, l2\, g + D^2\, l1\, g + g^2 \end{bmatrix}$$

A left inverse of the parametrization (if it exists) is a *flat output* of the system:

```
> F := LeftInverse(P, Alg);
```

$$F := \begin{bmatrix} -\frac{l1}{g^2\,(-l1+l2)} & \frac{l2}{g^2\,(-l1+l2)} & 0 \end{bmatrix}$$

We want to express the system variables *x1*, *x2*, and *u* in terms of the flat output:

```
> R2 := linalg[stackmatrix](R, F);
```

$$R2 := \begin{bmatrix} D^2 + \frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2 + \frac{g}{l2} & -\frac{g}{l2} \\ -\frac{l1}{g^2\,(-l1+l2)} & \frac{l2}{g^2\,(-l1+l2)} & 0 \end{bmatrix}$$

```
> E := Elimination(R2, [x1,x2,u], [z1,z2,y], Alg);
```

$$E := \mathrm{table}([1 = \begin{bmatrix} 0 & 0 & l2\, g^2 - l1\, g^2 \\ 0 & l2\, g - l1\, g & 0 \\ l2\, g - l1\, g & 0 & 0 \end{bmatrix},$$

$$2 = \begin{bmatrix} l2\, D^2\, l1^2 + g\, l1^2 & -l2^2\, g - l2^2\, D^2\, l1 & l2^2\, D^2\, g^3 - l2\, D^4\, g^2\, l1^2 + l2^2\, D^4\, l1\, g^2 - D^2\, g^3\, l1^2 - g^4\, l1 + g^4\, l2 \\ l1^2 & -l1\, l2 & -D^2\, g^2\, l1^2 + l1\, D^2\, g^2\, l2 - g^3\, l1 + l2\, g^3 \\ l1\, l2 & -l2^2 & -l1\, D^2\, g^2\, l2 + D^2\, g^2\, l2^2 - g^3\, l1 + l2\, g^3 \end{bmatrix}$$

$$])$$

```
> ApplyMatrix(E[1], [x1(t),x2(t),u(t)], Alg)=ApplyMatrix(E[2], [z1(t),z2(t),y(t)], Alg);
```

$$\begin{bmatrix} g^2\,(-l1+l2)\,\mathrm{u}(t) \\ (-l1+l2)\,g\,\mathrm{x2}(t) \\ (-l1+l2)\,g\,\mathrm{x1}(t) \end{bmatrix} =$$

$$\Big[l2\, l1^2 \left(\frac{d^2}{dt^2}\mathrm{z1}(t)\right) + g\, l1^2\, \mathrm{z1}(t) - l2^2\, l1 \left(\frac{d^2}{dt^2}\mathrm{z2}(t)\right) - l2^2\, g\, \mathrm{z2}(t) + \left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) l2^2\, g^3 - \left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) g^3\, l1^2 - \left(\frac{d^4}{dt^4}\mathrm{y}(t)\right) l2\, g^2\, l1^2$$
$$+ \left(\frac{d^4}{dt^4}\mathrm{y}(t)\right) l2^2\, l1\, g^2 - \mathrm{y}(t)\, g^4\, l1 + \mathrm{y}(t)\, g^4\, l2 \Big]$$
$$\left[l1^2\, \mathrm{z1}(t) - l1\, l2\, \mathrm{z2}(t) - \left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) l1^2\, g^2 + \left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) l1\, g^2\, l2 - \mathrm{y}(t)\, g^3\, l1 + \mathrm{y}(t)\, l2\, g^3 \right]$$
$$\left[l1\, l2\, \mathrm{z1}(t) - l2^2\, \mathrm{z2}(t) - \left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) l1\, g^2\, l2 + \left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) l2^2\, g^2 - \mathrm{y}(t)\, g^3\, l1 + \mathrm{y}(t)\, l2\, g^3 \right]$$

Up to invertible constants, the previous equations express *x1*, *x2*, and *u* in terms of the flat output *y* (modulo the system equations).

See Also:

DefineOreAlgebra, Factorize, Quotient, Integrability, ReduceMatrix, Involution, SyzygyModule, ApplyMatrix, LeftInverse, Parametrization.

OreModules[Exti],

OreModules[ExtiRat] - **compute an extension module of a finitely presented module over an Ore algebra with values in this Ore algebra**

Calling Sequence:

Exti(R,Alg,i)
ExtiRat(R,Alg,i)

Parameters:

`R`	-	matrix with entries in **`Alg`** or INJ(n) or SURJ(n), where n is a non-negative integer
`Alg`	-	Ore algebra (given by DefineOreAlgebra)
`i`	-	non-negative integer

Description:

- ***Exti*** computes the ith extension module with values in **`Alg`** of the left **`Alg`**-module M which is generated by the rows of **`R`**, i.e. the ith homology module of the complex which is obtained from a free resolution of M by applying the hom functor. As a special case, for $i = 0$ the computed extension module is the right **`Alg`**-module of homomorphisms from M into **`Alg`**, returned as a presentation of a left **`Alg`**-module by means of an involution of **`Alg`** (cf. Involution).
- ***Exti*** uses an involution of **`Alg`** in order to turn the complex of right **`Alg`**-modules, obtained from a free resolution of M by applying the hom functor, into a complex of left **`Alg`**-modules. Following the definition of the extension module, if $i > 0$, ***Exti*** computes a part of a free resolution of M of length $i + 1$ (see Resolution) and applies Involution to the ith resp. the $(i + 1)$th matrix in this resolution to obtain the matrix L_i resp. L_{i+1}. Then ***Exti*** computes the syzygy module S of the left **`Alg`**-module presented by L_{i+1} and the annihilator of the generating elements of S in the left **`Alg`**-module presented by L_i (see Quotient). This last step corresponds to the computation of the ith homology module of the before mentioned complex. If $i = 0$, then ***Exti*** only computes the syzygy module of the left **`Alg`** -module presented by L_{i+1}.
- **`R`** is a matrix with entries in **`Alg`** or INJ(n) or SURJ(n), where n is a non-negative integer.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- If $i = 0$, then the result is a matrix with entries in **`Alg`**. After applying Involution to this matrix, one obtains a presentation of the right **`Alg`**-module of homomorphisms from the left **`Alg`**-module presented by **`R`** into **`Alg`**.
- If $i > 0$, then the result is a list of three matrices with entries in **`Alg`**. The residue classes of the rows of the second matrix in the left **`Alg`**-module presented by L_i form a generating set for S (see above). The first matrix gives the annihilator of these residue classes in this module. The third matrix is L_{i+1}. The product of the second matrix by the third matrix is zero by construction. If the part of the free resolution of M computed by Resolution is shorter than $i + 1$, then ***Exti*** returns [*undefined*, *ZERO*, *ZERO*].
- If $i > 0$, the first matrix of the result is a matrix having a block diagonal structure, where each block consists of only one column but may have several rows. The number of blocks equals the number of rows of the second matrix of the result. The entries of the ith block form a Groebner basis (w.r.t. the degree reverse lexicographical ordering on the variables of **`Alg`**) of the annihilator of the ith row of the second matrix in the left **`Alg`**-module presented by L_i (see also Quotient).
- The ith extension module with values in **`Alg`** of M is the zero module if and only if the first matrix of the result of ***Exti*** is an identity matrix. For an additional interpretation of the first extension module, see Torsion.
- ***ExtiRat*** performs the same computations as ***Exti***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- Extn computes several extension modules at once. Torsion is synonymous with ***Exti*** for $i = 1$.
- For more details on the algorithm computing the extension modules over Ore algebras, see F. Chyzak, A. Quadrat, D. Robertz,

"Effective algorithms for parametrizing linear control systems over Ore algebras", Applicable Algebra in Engineering, Communication and Computing (AAECC) 16 (2005), pp. 319-376.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := matrix([[2, -D2], [2*D1, -D1*D2]]);
```

$$R := \begin{bmatrix} 2 & -D2 \\ 2\,D1 & -D1\,D2 \end{bmatrix}$$

```
> Exti(R, Alg, 0);
```

$$[D2 \quad -2]$$

To obtain the generator of the right **Alg**-module of homomorphisms from the left **Alg**-module *M* presented by **R** into **Alg**, one has to apply an involution of **Alg** to the result of ***Exti***:

```
> H := Involution(Exti(R, Alg, 0), Alg);
```

$$H := \begin{bmatrix} -D2 \\ -2 \end{bmatrix}$$

Indeed, the homomorphism of free left **Alg**-modules represented by *H* maps the relations defining *M* to zero in **Alg**.

```
> Mult(R, H, Alg);
```

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Example 2:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := matrix([[-D2+D1+2, -D2], [2, -D2], [2*D1, -D1*D2]]);
```

$$R := \begin{bmatrix} -D2 + D1 + 2 & -D2 \\ 2 & -D2 \\ 2\,D1 & -D1\,D2 \end{bmatrix}$$

```
> Ext := Exti(R, Alg, 1);
```

$$Ext := \left[\begin{bmatrix} D1\,D2 - D2^2 & 0 \\ 0 & D1\,D2 - D2^2 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & D1 \end{bmatrix}, \begin{bmatrix} 0 \\ -D1 \\ -1 \end{bmatrix} \right]$$

In this example, L_1 is given by:

```
> L[1] := Involution(R, Alg);
```

$$L_1 := \begin{bmatrix} D2 - D1 + 2 & 2 & -2\,D1 \\ D2 & D2 & -D1\,D2 \end{bmatrix}$$

Ext[1] is the annihilator of the rows of *Ext*[2] in the **Alg**-module presented by L_1, i.e.:

```
> ReduceMatrix(Mult(Ext[1], Ext[2], Alg), L[1], Alg);
```

[ ]

This annihilator can also be computed by:

```
> Quotient(Ext[2], L[1], Alg);
```

$$\begin{bmatrix} D1\,D2 - D2^2 & 0 \\ 0 & D1\,D2 - D2^2 \end{bmatrix}$$

By construction, *Ext*[3] yields a parametrization of the system $Ext[2]\,y = 0$:

```
> Mult(Ext[2], Ext[3], Alg);
```

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Example 3:

Linear system of partial differential equations with non-constant coefficients appearing in the study of the Lie algebra SU(2) (see C. M. Bender, G. V. Dunne, L. R. Mead, *Underdetermined systems of partial differential equations*, Journal of Mathematical Physics, vol. 41 no. 9 (2000), pp. 6388-6398):

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := Involution(evalm([[x3*D1-x1*D3, x3*D2-x2*D3, -1], [-1, x1*D2-x2*D1, x1*D3-x3*D1],
  [x2*D1-x1*D2, -1, x2*D3-x3*D2]]), Alg);
```

$$R := \begin{bmatrix} x1\,D3 - x3\,D1 & -1 & x1\,D2 - x2\,D1 \\ x2\,D3 - x3\,D2 & x2\,D1 - x1\,D2 & -1 \\ -1 & x3\,D1 - x1\,D3 & x3\,D2 - x2\,D3 \end{bmatrix}$$

```
> Exti(R, Alg, 1);
```

$$\left[\begin{bmatrix} x2\,D3 - x3\,D2 & 0 & 0 \\ x1\,D3 - x3\,D1 & 0 & 0 \\ x1\,D2 - x2\,D1 & 0 & 0 \\ 0 & x2\,D3 - x3\,D2 & 0 \\ 0 & x1\,D3 - x3\,D1 & 0 \\ 0 & x1\,D2 - x2\,D1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} x1 & x2 & x3 \\ D1 & D2 & D3 \\ -1 & x1\,D2 - x2\,D1 & x1\,D3 - x3\,D1 \end{bmatrix}, \begin{bmatrix} x3\,D2 - x2\,D3 \\ x1\,D3 - x3\,D1 \\ x2\,D1 - x1\,D2 \end{bmatrix}\right]$$

```
> Exti(R, Alg, 2);
```

$$\left[\begin{bmatrix} x2\,D3 - x3\,D2 \\ x1\,D3 - x3\,D1 \\ x1\,D2 - x2\,D1 \end{bmatrix}, [\;1], \mathrm{SURJ}(1)\right]$$

```
> Exti(R, Alg, 3);
```

$$[\mathit{undefined}, \mathit{ZERO}, \mathit{ZERO}]$$

Computing extension modules over the Weyl algebra with rational coefficients:

```
> ExtiRat(R, Alg, 1);
```

$$\left[\begin{bmatrix} x3\,D2 - x2\,D3 & 0 \\ x3\,D1 - x1\,D3 & 0 \\ 0 & x3\,D2 - x2\,D3 \\ 0 & x3\,D1 - x1\,D3 \end{bmatrix}, \begin{bmatrix} x1 & x2 & x3 \\ 0 & -x1^2\,D2 + x1\,x2\,D1 - x2 & -x1^2\,D3 + x1\,x3\,D1 - x3 \end{bmatrix}, \begin{bmatrix} x3\,D2 - x2\,D3 \\ x1\,D3 - x3\,D1 \\ x2\,D1 - x1\,D2 \end{bmatrix}\right]$$

```
> ExtiRat(R, Alg, 2);
```

$$\left[\begin{bmatrix} x3\,D2 - x2\,D3 \\ x3\,D1 - x1\,D3 \end{bmatrix}, [\;1], \mathrm{SURJ}(1)\right]$$

See Also:

DefineOreAlgebra, Involution, SyzygyModule, Quotient, Resolution, FreeResolution, ShorterFreeResolution, ShortestFreeResolution, ProjectiveDimension, Extn, Torsion, Parametrization, MinimalParametrization, AutonomousElements, PiPolynomial, TorsionElements.

OreModules[Extn],

OreModules[ExtnRat] - **compute extension modules of a finitely presented module over an Ore algebra with values in this Ore algebra**

Calling Sequence:

Extn(R,Alg,n)
ExtnRat(R,Alg,n)

Parameters:

`R`	-	matrix with entries in **`Alg`** or INJ(*n*) or SURJ(*n*), where *n* is a non-negative integer
`Alg`	-	Ore algebra (given by DefineOreAlgebra)
`n`	-	non-negative integer

Description:

- ***Extn*** computes the first, second, ..., and *n*th extension module with values in **`Alg`** of the left **`Alg`**-module *M* which is generated by the rows of **`R`**, i.e. the first, second, ..., and *n*th homology module of the complex which is obtained from a free resolution of *M* by applying the hom functor.
- **`R`** is a matrix with entries in **`Alg`** or INJ(*n*) or SURJ(*n*), where *n* is a non-negative integer.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result is a table of $n+1$ lists. The entry of the result with index *j* equals the result of ***Exti*** applied to **`R`** with $i = j$.
- ***ExtnRat*** performs the same computations as ***Extn***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- Exti computes the *i*th extension module for given *i*. Torsion is synonymous with ***Exti*** for $i = 1$.
- For more details on the algorithm computing the extension modules over Ore algebras, see F. Chyzak, A. Quadrat, D. Robertz, "Effective algorithms for parametrizing linear control systems over Ore algebras", Applicable Algebra in Engineering, Communication and Computing (AAECC) 16 (2005), pp. 319-376.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := matrix([[-D2+D1+2, -D2], [2, -D2], [2*D1, -D1*D2]]);
```

$$R := \begin{bmatrix} -D2+D1+2 & -D2 \\ 2 & -D2 \\ 2\,D1 & -D1\,D2 \end{bmatrix}$$

```
> Ext := Extn(R, Alg, 2);
```

Ext :=

$$\mathrm{table}\left(\left[0 = [\mathit{undefined}, \mathrm{INJ}(2), \mathit{undefined}],\ 1 = \left[\begin{bmatrix} D1\,D2 - D2^2 & 0 \\ 0 & D1\,D2 - D2^2 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & D1 \end{bmatrix}, \begin{bmatrix} 0 \\ -D1 \\ -1 \end{bmatrix}\right],\ 2 = [[\ 1], [\ 1], \mathrm{SURJ}(1)]\right]\right)$$

```
> Exti(R, Alg, 0);
```

$$\mathrm{INJ}(2)$$

```
> Exti(R, Alg, 1);
```

$$\left[\begin{bmatrix} D1\,D2 - D2^2 & 0 \\ 0 & D1\,D2 - D2^2 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & D1 \end{bmatrix}, \begin{bmatrix} 0 \\ -D1 \\ -1 \end{bmatrix}\right]$$

```
> Exti(R, Alg, 2);
```

$$[[\ 1], [\ 1], \mathrm{SURJ}(1)]$$

Example 2:

Linear system of partial differential equations with non-constant coefficients appearing in the study of the Lie algebra SU(2) (see C. M. Bender, G. V. Dunne, L. R. Mead, *Underdetermined systems of partial differential equations*, Journal of Mathematical Physics, vol. 41 no. 9 (2000), pp. 6388-6398):

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := Involution(evalm([[x3*D1-x1*D3, x3*D2-x2*D3, -1], [-1, x1*D2-x2*D1, x1*D3-x3*D1],
  [x2*D1-x1*D2, -1, x2*D3-x3*D2]]), Alg);
```

$$R := \begin{bmatrix} x1\,D3 - x3\,D1 & -1 & x1\,D2 - x2\,D1 \\ x2\,D3 - x3\,D2 & x2\,D1 - x1\,D2 & -1 \\ -1 & x3\,D1 - x1\,D3 & x3\,D2 - x2\,D3 \end{bmatrix}$$

```
> Extn(R, Alg, 3);
```

table([0 = [*undefined*, [$x1\,D2 - x2\,D1 \quad x2\,D3 - x3\,D2 \quad x3\,D1 - x1\,D3$], *undefined*],

$$1 = \left[\begin{bmatrix} x2\,D3 - x3\,D2 & 0 & 0 \\ x1\,D3 - x3\,D1 & 0 & 0 \\ x1\,D2 - x2\,D1 & 0 & 0 \\ 0 & x2\,D3 - x3\,D2 & 0 \\ 0 & x1\,D3 - x3\,D1 & 0 \\ 0 & x1\,D2 - x2\,D1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} x1 & x2 & x3 \\ D1 & D2 & D3 \\ -1 & x1\,D2 - x2\,D1 & x1\,D3 - x3\,D1 \end{bmatrix}, \begin{bmatrix} x3\,D2 - x2\,D3 \\ x1\,D3 - x3\,D1 \\ x2\,D1 - x1\,D2 \end{bmatrix}\right],$$

$$2 = \left[\begin{bmatrix} x2\,D3 - x3\,D2 \\ x1\,D3 - x3\,D1 \\ x1\,D2 - x2\,D1 \end{bmatrix}, [\ 1], \mathrm{SURJ}(1)\right],$$

3 = [*undefined*, *ZERO*, *ZERO*]

])

Computation of extension modules over the Weyl algebra with rational coefficients:

```
> ExtnRat(R, Alg, 3);
```

table([0 = [*undefined*, [$x2\,D1 - x1\,D2 \quad x3\,D2 - x2\,D3 \quad x1\,D3 - x3\,D1$], *undefined*],

$$1 = \left[\begin{bmatrix} x3\,D2 - x2\,D3 & 0 \\ x3\,D1 - x1\,D3 & 0 \\ 0 & x3\,D2 - x2\,D3 \\ 0 & x3\,D1 - x1\,D3 \end{bmatrix}, \begin{bmatrix} x1 & x2 & x3 \\ 0 & -x2 - x1^2\,D2 + x1\,x2\,D1 & -x3 - x1^2\,D3 + x1\,x3\,D1 \end{bmatrix}, \begin{bmatrix} x3\,D2 - x2\,D3 \\ x1\,D3 - x3\,D1 \\ x2\,D1 - x1\,D2 \end{bmatrix}\right],$$

$$2 = \left[\begin{bmatrix} x3\,D2 - x2\,D3 \\ x3\,D1 - x1\,D3 \end{bmatrix}, [\ 1], \mathrm{SURJ}(1)\right],$$

3 = [*undefined*, *ZERO*, *ZERO*]

])

See Also:

DefineOreAlgebra, Involution, SyzygyModule, Quotient, Resolution, FreeResolution, ShorterFreeResolution, ShortestFreeResolution, ProjectiveDimension, Exti, Torsion, Parametrization, MinimalParametrization, AutonomousElements, PiPolynomial, TorsionElements.

OreModules[Factorize],

OreModules[FactorizeRat] - **right-divide a matrix over an Ore algebra by another one, if possible**

Calling Sequence:

Factorize(M1,M2,Alg)
FactorizeRat(M1,M2,Alg)

Parameters:

M1, M2	-	matrices with entries in **Alg**
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***Factorize*** performs, if possible, a right-division of **M1** by **M2**, i.e., ***Factorize*** returns a matrix *F* with entries in **Alg**, if it exists, such that *F* **M2** = **M1**.
- **M1** and **M2** are matrices with entries in the Ore algebra **Alg** having the same number of columns.
- **Alg** is expected to be defined using DefineOreAlgebra.
- The result is a matrix with entries in **Alg** or the empty list, if right-division failed.
- ***FactorizeRat*** performs the same computations as ***Factorize***, but the domain of coefficients of the Ore algebra **Alg** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D[1],x[1]], diff=[D[2],x[2]], polynom=[x[1],x[2]]):
> M1 := matrix([[-D[1],-D[2],0], [-1,0,-D[2]], [0,-1,D[1]]]);
```

$$M1 := \begin{bmatrix} -D_1 & -D_2 & 0 \\ -1 & 0 & -D_2 \\ 0 & -1 & D_1 \end{bmatrix}$$

```
> M2 := matrix([[-D[1]-2,-D[2]-3,-2*D[2]+3*D[1]], [-2,0,-2*D[2]], [0,17,-17*D[1]]]);
```

$$M2 := \begin{bmatrix} -D_1 - 2 & -D_2 - 3 & -2\,D_2 + 3\,D_1 \\ -2 & 0 & -2\,D_2 \\ 0 & 17 & -17\,D_1 \end{bmatrix}$$

```
> F := Factorize(M1, M2, Alg);
```

$$F := \begin{bmatrix} 1 & -1 & \frac{3}{17} \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{-1}{17} \end{bmatrix}$$

```
> Mult(F, M2, Alg);
```

$$\begin{bmatrix} -D_1 & -D_2 & 0 \\ -1 & 0 & -D_2 \\ 0 & -1 & D_1 \end{bmatrix}$$

```
> M1 := matrix([[D[1]]]);
```

$$M1 := [D_1]$$

```
> M2 := matrix([[D[2]]]);
```

$$M2 := [D_2]$$

```
> Factorize(M1, M2, Alg);
```

$$[\ ]$$

See Also:

DefineOreAlgebra, Quotient, Elimination, Integrability, ReduceMatrix, Involution, SyzygyModule, Resolution, FreeResolution.

OreModules[FinalConditions] - extract the coefficients of variations of functions and their derivatives in a given expression

Calling Sequence:

FinalConditions(B,T)

Parameters:

`B` - expression which is linear in all occuring variations of functions and their derivatives

`T` - value serving as "final time" (to be substituted for the independent variable)

Description:

- ***FinalConditions*** first extracts the coefficients of the variations of functions and their derivatives which occur in **B**. Then **T** is substituted for the independent variable which is the argument of the functions occuring in **B**. The list of the resulting expressions is returned.
- **B** is expected to be an expression which is linear in all occuring variations of functions and their derivatives. Exactly the functions named δ with subscript are interpreted as variations of functions.
- ***FinalConditions*** is intended to be applied to the second entry of the result of LQEquations. It then yields the left hand sides of the equations determined by the boundary terms which were introduced in the computation of the Euler-Lagrange equations.

Examples:

```
> with(OreModules):
```

Example 1:

```
> B := a(t)*delta[xi[1]](t)+b(t)*diff(delta[xi[1]](t),t)+c(t)*diff(delta[xi[1]](t),t,t);
```

$$B := a(t)\,\delta_{\xi_1}(t) + b(t)\left(\frac{d}{dt}\delta_{\xi_1}(t)\right) + c(t)\left(\frac{d^2}{dt^2}\delta_{\xi_1}(t)\right)$$

```
> FinalConditions(B, T);
```

$$[a(T), b(T), c(T)]$$

Example 2:

```
> B := 6*delta[xi[1]](t)*xi[1](t)+12*delta[xi[1]](t)*diff(xi[1](t),t)-4*delta[xi[1]](t)*diff(xi[1](t),t$3)+2*delta[xi[1]](t)*diff(xi[2](t),t)+2*diff(delta[xi[1]](t),t)*xi[1](t)+4*diff(delta[xi[1]](t),t)*diff(xi[1](t),t$2)+6*diff(delta[xi[1]](t),t)*diff(xi[1](t),t)-2*diff(delta[xi[1]](t),t)*xi[2](t)-3*delta[xi[1]](t)*xi[2](t);
```

$$B := 6\,\delta_{\xi_1}(t)\,\xi_1(t) + 12\,\delta_{\xi_1}(t)\left(\frac{d}{dt}\xi_1(t)\right) - 4\,\delta_{\xi_1}(t)\left(\frac{d^3}{dt^3}\xi_1(t)\right) + 2\,\delta_{\xi_1}(t)\left(\frac{d}{dt}\xi_2(t)\right) + 2\left(\frac{d}{dt}\delta_{\xi_1}(t)\right)\xi_1(t) + 4\left(\frac{d}{dt}\delta_{\xi_1}(t)\right)\left(\frac{d^2}{dt^2}\xi_1(t)\right) + 6\left(\frac{d}{dt}\delta_{\xi_1}(t)\right)\left(\frac{d}{dt}\xi_1(t)\right) - 2\left(\frac{d}{dt}\delta_{\xi_1}(t)\right)\xi_2(t) - 3\,\delta_{\xi_1}(t)\,\xi_2(t)$$

```
> FinalConditions(B, T);
```

$$[6\,\xi_1(T) + 12\,D(\xi_1)(T) - 4\,(D^{(3)})(\xi_1)(T) + 2\,D(\xi_2)(T) - 3\,\xi_2(T),\ 2\,\xi_1(T) + 4\,(D^{(2)})(\xi_1)(T) + 6\,D(\xi_1)(T) - 2\,\xi_2(T)]$$

```
> collect(B, [delta[xi[1]](t), diff(delta[xi[1]](t),t)]);
```

$$\left(6\,\xi_1(t) + 12\left(\frac{d}{dt}\xi_1(t)\right) - 4\left(\frac{d^3}{dt^3}\xi_1(t)\right) + 2\left(\frac{d}{dt}\xi_2(t)\right) - 3\,\xi_2(t)\right)\delta_{\xi_1}(t) + \left(2\,\xi_1(t) + 4\left(\frac{d^2}{dt^2}\xi_1(t)\right) + 6\left(\frac{d}{dt}\xi_1(t)\right) - 2\,\xi_2(t)\right)\left(\frac{d}{dt}\delta_{\xi_1}(t)\right)$$

See Also:

LQEquations, BoundaryTerms, Mult, ApplyMatrix, Involution, ControllabilityMatrix, Brunovsky, KalmanSystem, TorsionElements.

OreModules[FirstIntegral] - compute first integrals for linear systems of ordinary differential equations

Calling Sequence:

FirstIntegral(R,v,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`v`	-	list or vector of functions
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***FirstIntegral*** returns the first integrals of the linear system of ordinary differential equations represented by **`R`**, i.e. the autonomous elements of the system which are annihilated by the differential operator w.r.t. the independent variable. In other words, the first integrals are those left **`Alg`**-linear combinations of the system variables whose derivative is a consequence of the system equations.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`** of ordinary differential operators.
- **`v`** is a list or vector of functions which depend on the independent variable of the ODE system. These functions are interpreted as the system variables.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***FirstIntegral*** is a function of the independent variable of the ODE system which is given in terms of the system variables specified by **`v`**. Since it is computed by solving a linear system of ordinary differential equations, the result depends on some constants introduced by dsolve.
- For more information, see J.-F. Pommaret, A. Quadrat, "Localization and parametrization of linear multidimensional control systems", Systems & Control Letters, 37 (1999), pp. 247-260.

Examples:

```
> with(OreModules):
```

Example 1:

(See Example 9 in J.-F. Pommaret, A. Quadrat, *Localization and parametrization of linear multidimensional control systems*, Systems & Control Letters, 37 (1999), pp. 247-260.)

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
> R := evalm([[D, -1, -1], [-1, D, 1]]);
```

$$R := \begin{bmatrix} D & -1 & -1 \\ -1 & D & 1 \end{bmatrix}$$

```
> z := FirstIntegral(R, [eta1(t),eta2(t),eta3(t)], Alg);
```

$$z := \_C1\, e^{(-t)} (\eta 1(t) + \eta 2(t))$$

```
> S := ApplyMatrix(R, [eta1(t),eta2(t),eta3(t)], Alg);
```

$$S := \begin{bmatrix} \left(\frac{d}{dt}\eta 1(t)\right) - \eta 2(t) - \eta 3(t) \\ -\eta 1(t) + \left(\frac{d}{dt}\eta 2(t)\right) + \eta 3(t) \end{bmatrix}$$

```
> diff(z, t) - _C1*exp(-t)*(S[1,1]+S[2,1]);
```

$$-\_C1\, e^{(-t)} (\eta 1(t) + \eta 2(t)) + \_C1\, e^{(-t)} \left(\left(\frac{d}{dt}\eta 1(t)\right) + \left(\frac{d}{dt}\eta 2(t)\right)\right) - \_C1\, e^{(-t)} \left(\left(\frac{d}{dt}\eta 1(t)\right) - \eta 2(t) - \eta 1(t) + \left(\frac{d}{dt}\eta 2(t)\right)\right)$$

```
> simplify(%);
```

$$0$$

Example 2:

We study a bipendulum, namely a system composed of a bar where two pendula are fixed, one of length *l1* and one of length *l2*. See J.-F. Pommaret, *Partial Differential Control Theory*, Kluwer, 2001, p. 569. Here we consider the case, where $l1 = l2$:

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g, l1, l2]):
> R := subs(l2=l1, evalm([[D^2+g/l1, 0, -g/l1], [0, D^2+g/l2, -g/l2]]));
```

$$R := \begin{bmatrix} D^2 + \frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2 + \frac{g}{l1} & -\frac{g}{l1} \end{bmatrix}$$

```
> AutonomousElements(R, [x1(t),x2(t),u(t)], Alg);
```

$$\left[\left[g\,\theta_1(t) + l1\left(\frac{d^2}{dt^2}\theta_1(t)\right) = 0\right], \left[\theta_1 = \_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right) + \_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\right], [\theta_1 = x1(t) - x2(t)]\right]$$

```
> V := FirstIntegral(R, [x1(t),x2(t),u(t)], Alg);
```

$$V := -\left(\left(\frac{d}{dt}x1(t)\right)\_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\sqrt{l1} + \left(\frac{d}{dt}x1(t)\right)\_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\sqrt{l1} - x1(t)\_C1 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\sqrt{g} + x1(t)\_C2 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\sqrt{g} - \left(\frac{d}{dt}x2(t)\right)\_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\sqrt{l1} - \left(\frac{d}{dt}x2(t)\right)\_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\sqrt{l1} + x2(t)\_C1 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\sqrt{g} - x2(t)\_C2 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\sqrt{g}\right) \Big/ \sqrt{l1}$$

```
> S := ApplyMatrix(R, [x1(t),x2(t),u(t)], Alg);
```

$$S := \begin{bmatrix} \dfrac{g\,x1(t) + \left(\frac{d^2}{dt^2}x1(t)\right)l1 - g\,u(t)}{l1} \\ \dfrac{g\,x2(t) + \left(\frac{d^2}{dt^2}x2(t)\right)l1 - g\,u(t)}{l1} \end{bmatrix}$$

```
> L := expand(evalm([coeff(diff(V, t), diff(x1(t),t,t)),
                     -coeff(diff(V, t), diff(x1(t),t,t))] &* S)[1]);
```

$$L := -\frac{\_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right) g\,x1(t)}{l1} - \_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\left(\frac{d^2}{dt^2}x1(t)\right) - \frac{\_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right) g\,x1(t)}{l1} - \_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\left(\frac{d^2}{dt^2}x1(t)\right) + \frac{\_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right) g\,x2(t)}{l1} + \_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\left(\frac{d^2}{dt^2}x2(t)\right) + \frac{\_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right) g\,x2(t)}{l1} + \_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l1}}\right)\left(\frac{d^2}{dt^2}x2(t)\right)$$

```
> simplify(diff(V, t)-L);
```

$$0$$

Example 3:

The linearized ordinary differential equations for the satellite in a circular orbit (see T. Kailath, *Linear Systems*, Prentice-Hall, 1980, p. 60 and p. 145). We consider the case where $a = 0$ and $b = 1$, i.e., the case where we only have a tangential thrust:

```
> Alg := DefineOreAlgebra(diff=[Dt,t], polynom=[t], comm=[omega,m,r,a,b]):
> Rab := evalm([[Dt,-1,0,0,0,0], [-3*omega^2,Dt,0,-2*omega*r,-a/m,0], [0,0,Dt,-1,0,0],
  [0,2*omega/r,0,Dt,0,-b/(m*r)]]);
```

$$Rab := \begin{bmatrix} Dt & -1 & 0 & 0 & 0 & 0 \\ -3\,\omega^2 & Dt & 0 & -2\,\omega\,r & -\frac{a}{m} & 0 \\ 0 & 0 & Dt & -1 & 0 & 0 \\ 0 & \frac{2\,\omega}{r} & 0 & Dt & 0 & -\frac{b}{m\,r} \end{bmatrix}$$

```
> R := linalg[submatrix](subs(a=1,b=0,evalm(Rab)), 1..4, 1..5);
```

$$R := \begin{bmatrix} Dt & -1 & 0 & 0 & 0 \\ -3\,\omega^2 & Dt & 0 & -2\,\omega\,r & -\frac{1}{m} \\ 0 & 0 & Dt & -1 & 0 \\ 0 & \frac{2\,\omega}{r} & 0 & Dt & 0 \end{bmatrix}$$

```
> AutonomousElements(R, [x1(t),x2(t),x3(t),x4(t),u1(t)], Alg);
```

$$\left[\begin{bmatrix} 3\,\omega\,m\,\theta_1(t) - \theta_2(t) = 0 \\ \frac{d}{dt}\theta_2(t) = 0 \end{bmatrix}, \begin{bmatrix} \theta_1 = \frac{\_C1}{3\,\omega\,m} \\ \theta_2 = \_C1 \end{bmatrix}, \begin{bmatrix} \theta_1 = 2\,\omega\,\mathrm{x1}(t) + r\,\mathrm{x4}(t) \\ \theta_2 = 2\,m\left(\frac{d}{dt}\mathrm{x2}(t)\right) - \omega\,r\,m\,\mathrm{x4}(t) - 2\,\mathrm{u1}(t) \end{bmatrix}\right]$$

```
> FirstIntegral(R, [x1(t),x2(t),x3(t),x4(t),u1(t)], Alg);
```

$$\frac{1}{2}\,\frac{\_C1\,(2\,\omega\,\mathrm{x1}(t) + r\,\mathrm{x4}(t))}{\omega}$$

Example 4:

System of linear ordinary differential equations describing two pendula mounted on a cart (J. W. Polderman, J. C. Willems, *Introduction to Mathematical Systems Theory. A Behavioral Approach*, TAM 26, Springer, 1998):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[m1,m2,M,L1,L2,g]):
> R := subs(L2=L1, evalm([[m1*L1*D^2, m2*L2*D^2, -1, (M+m1+m2)*D^2],
  [m1*L1^2*D^2-m1*L1*g, 0, 0, m1*L1*D^2], [0, m2*L2^2*D^2-m2*L2*g, 0, m2*L2*D^2]]));
```

$$R := \begin{bmatrix} m1\,L1\,\mathrm{D}^2 & \mathrm{D}^2\,L1\,m2 & -1 & (M + m1 + m2)\,\mathrm{D}^2 \\ m1\,L1^2\,\mathrm{D}^2 - m1\,L1\,g & 0 & 0 & m1\,L1\,\mathrm{D}^2 \\ 0 & m2\,L1^2\,\mathrm{D}^2 - L1\,g\,m2 & 0 & \mathrm{D}^2\,L1\,m2 \end{bmatrix}$$

```
> AutonomousElements(R, [x1(t),x2(t),x3(t),u1(t)], Alg);
```

$$\left[\begin{bmatrix} L1\,m2\,m1\,g\,\theta_1(t) - L1\,m2\,\theta_3(t) = 0 \\ L1\,m2\,\theta_2(t) + L1\,m2\,\theta_3(t) = 0 \\ -L1\,m2\left(g\,\theta_3(t) - L1\left(\frac{d^2}{dt^2}\theta_3(t)\right)\right) = 0 \end{bmatrix}, \begin{bmatrix} \theta_1 = \frac{\_C1\,\mathbf{e}^{\left(\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} + \_C2\,\mathbf{e}^{\left(-\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)}}{m1\,g} \\ \theta_2 = -\_C1\,\mathbf{e}^{\left(\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} - \_C2\,\mathbf{e}^{\left(-\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} \\ \theta_3 = \_C1\,\mathbf{e}^{\left(\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} + \_C2\,\mathbf{e}^{\left(-\frac{\sqrt{g}\,t}{\sqrt{L1}}\right)} \end{bmatrix},\right.$$

$$\left[\begin{array}{c} \theta_1 = \mathrm{x1}(t) - \mathrm{x2}(t) \\ \theta_2 = \mathrm{x2}(t)\, g\, m1 + \mathrm{x2}(t)\, g\, m2 - \mathrm{x3}(t) + M\left(\frac{d^2}{dt^2}\mathrm{u1}(t)\right) \\ \theta_3 = -\mathrm{x2}(t)\, g\, m1 - \mathrm{x2}(t)\, g\, m2 - \mathrm{x2}(t)\, g\, M + L1\, M\left(\frac{d^2}{dt^2}\mathrm{x2}(t)\right) + \mathrm{x3}(t) \end{array}\right]$$

```
> FirstIntegral(R, [x1(t),x2(t),x3(t),u1(t)], Alg);
```

$$m1\, L1\left(L1\left(\frac{d}{dt}\mathrm{x1}(t)\right)\_C1\, \mathbf{e}^{\left(\frac{2\sqrt{g}\, t}{\sqrt{L1}}\right)} + L1\left(\frac{d}{dt}\mathrm{x1}(t)\right)\_C2 - \sqrt{L1}\,\mathrm{x1}(t)\_C1\sqrt{g}\,\mathbf{e}^{\left(\frac{2\sqrt{g}\, t}{\sqrt{L1}}\right)} + \sqrt{L1}\,\mathrm{x1}(t)\_C2\sqrt{g} - L1\left(\frac{d}{dt}\mathrm{x2}(t)\right)\_C1\,\mathbf{e}^{\left(\frac{2\sqrt{g}\, t}{\sqrt{L1}}\right)} - L1\left(\frac{d}{dt}\mathrm{x2}(t)\right)\_C2 + \sqrt{L1}\,\mathrm{x2}(t)\_C1\sqrt{g}\,\mathbf{e}^{\left(\frac{2\sqrt{g}\, t}{\sqrt{L1}}\right)} - \sqrt{L1}\,\mathrm{x2}(t)\_C2\sqrt{g}\right)\mathbf{e}^{\left(-\frac{\sqrt{g}\, t}{\sqrt{L1}}\right)}$$

See Also:

DefineOreAlgebra, AutonomousElements, Brunovsky, KalmanSystem, TorsionElements, Parametrization, MinimalParametrization, Exti, Extn, Torsion, PiPolynomial.

OreModules[FreeResolution],

OreModules[FreeResolutionRat] - compute a free resolution of a finitely presented module over an Ore algebra

Calling Sequence:

FreeResolution(R,Alg)
FreeResolutionRat(R,Alg)

Parameters:

R	-	matrix with entries in **Alg**
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***FreeResolution*** iterates the computation of syzygy modules of the left module over the Ore algebra **Alg** which is presented by **R**, i.e. of the factor module of the free **Alg**-module of tuples whose length equals the number of columns of **R** modulo the submodule which is generated by the rows of **R**. That means that ***FreeResolution*** computes a free resolution of the left module presented by **R**.
- At first, ***FreeResolution*** computes a matrix the rows of which generate all left **Alg**-linear relations of the rows of **R**. Then ***FreeResolution*** repeats the same for the matrix which has just been defined instead of **R**. This construction is iterated as long as there exist non-trivial left **Alg**-linear relations of the rows of the matrix which has just been computed.
- **R** is a matrix with entries in the Ore algebra **Alg**.
- **Alg** is expected to be defined using DefineOreAlgebra.
- The result is a table which contains matrices with entries **Alg** and the name INJ(r) as last entry of the table, where r is the number of rows of the last matrix occuring in the table. The matrix with index 1 in the result is **R** and the matrix with index i is the result of SyzygyModule applied to the matrix with index $i-1$, $i>1$, i.e., the rows of the matrix with index i generate the syzygy module of the left module generated by the rows of the matrix with index $i-1$.
- In order to stop the computation of syzygy modules after a given number of iterations, Resolution can be used.
- ***FreeResolutionRat*** performs the same computations as ***FreeResolution***, but the domain of coefficients of the Ore algebra **Alg** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[D1],[D2],[D3]]);
```

$$R := \begin{bmatrix} D1 \\ D2 \\ D3 \end{bmatrix}$$

```
> Res := FreeResolution(R, Alg);
```

$$Res := \text{table}([1 = \begin{bmatrix} D1 \\ D2 \\ D3 \end{bmatrix}, 2 = \begin{bmatrix} -D3 & 0 & D1 \\ -D2 & D1 & 0 \\ 0 & -D3 & D2 \end{bmatrix}, 3 = [-D2 \quad D3 \quad D1], 4 = \text{INJ}(1)])$$

```
> Mult(Res[2], Res[1], Alg);
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> Mult(Res[3], Res[2], Alg);
```

$$[0 \quad 0 \quad 0]$$

Example 2:

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s]):
> R := matrix([[0,Dt*delta], [t*Dt,t*delta], [Dt,Dt]]);
```

$$R := \begin{bmatrix} 0 & Dt\,\delta \\ t\,Dt & t\,\delta \\ Dt & Dt \end{bmatrix}$$

```
> Res := FreeResolution(R, Alg, 3);
```

$$Res := \text{table}([1 = \begin{bmatrix} 0 & Dt\,\delta \\ t\,Dt & t\,\delta \\ Dt & Dt \end{bmatrix}, 2 = \begin{bmatrix} -Dt\,t^2 + \delta\,t^2 & \delta - t\,Dt\,\delta & Dt\,\delta\,t^2 \\ 2\,\delta - Dt^2\,t - 2\,Dt + t\,Dt\,\delta & -Dt^2\,\delta & Dt^2\,\delta\,t + 2\,Dt\,\delta \end{bmatrix}, 3 = [Dt \quad -t], 4 = \text{INJ}(1)])$$

```
> Mult(Res[2], Res[1], Alg);
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> Mult(Res[3], Res[2], Alg);
```

$$[0 \quad 0 \quad 0]$$

See Also:

DefineOreAlgebra, SyzygyModule, ShorterFreeResolution, ShortestFreeResolution, Resolution, ProjectiveDimension, LiftOperators, Exti, Extn, Torsion, Parametrization, MinimalParametrization, Involution, Quotient, Integrability.

OreModules[GeneralizedInverse], OreModules[GeneralizedInverseRat] - compute a generalized inverse of a matrix over an Ore algebra

Calling Sequence:

GeneralizedInverse(M,Alg)
GeneralizedInverseRat(M,Alg)

Parameters:

`M` - matrix with entries in **`Alg`**
`Alg` - Ore algebra (given by DefineOreAlgebra)

Description:

- ***GeneralizedInverse*** computes (if possible) a generalized inverse of the matrix **`M`**, i.e. a matrix *G* with entries in **`Alg`** such that the product **`M`** *G* **`M`** equals **`M`**.
- If no generalized inverse of **`M`** exists, ***GeneralizedInverse*** returns the empty list.
- **`M`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- ***GeneralizedInverseRat*** performs the same computations as ***GeneralizedInverse***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- Left (right) inverses of matrices over Ore algebras are computed by LeftInverse (RightInverse).

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[Dt,t], polynom=[t]):
> R := evalm([[0, 1], [0, 0]]);
```

$$R := \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

```
> G := GeneralizedInverse(R, Alg);
```

$$G := \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

```
> Mult(R, G, R, Alg);
```

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Example 2:

Linear differential time-delay system describing a flexible rod (see H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD thesis, University of Orsay, France, 1995):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  shift_action=[delta,t,h]):
> R := evalm([[Dt, -Dt*delta, -1], [2*Dt*delta, -Dt-Dt*delta^2, 0]]);
```

$$R := \begin{bmatrix} Dt & -Dt\,\delta & -1 \\ 2\,Dt\,\delta & -Dt - Dt\,\delta^2 & 0 \end{bmatrix}$$

```
> Ext := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext := \left[\begin{bmatrix} Dt & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -2\,\delta & 1+\delta^2 & 0 \\ -Dt & Dt\,\delta & 1 \\ Dt\,\delta & -Dt & \delta \end{bmatrix}, \begin{bmatrix} 1+\delta^2 \\ 2\,\delta \\ -Dt\,\delta^2 + Dt \end{bmatrix} \right]$$

```
> G := GeneralizedInverse(Ext[2], Alg);
```

$$G := \begin{bmatrix} \frac{1}{2}\delta & 0 & 0 \\ 1 & 0 & 0 \\ -\frac{1}{2} Dt\,\delta & 1 & 0 \end{bmatrix}$$

```
> Mult(Ext[2], G, Ext[2], Alg) - Ext[2];
```

$$0$$

See Also:

DefineOreAlgebra, LeftInverse, RightInverse, LocalLeftInverse, Mult, ApplyMatrix, Involution, KroneckerProduct, Factorize, Quotient, Elimination, Integrability, ReduceMatrix.

OreModules[HilbertSeries],

OreModules[HilbertSeriesRat] - return Hilbert series of a finitely presented module over an Ore algebra

Calling Sequence:

HilbertSeries(R,Alg,s)
HilbertSeriesRat(R,Alg,s)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)
`s`	-	indeterminate for the Hilbert series

Description:

- ***HilbertSeries*** returns the Hilbert series of the left module over **`Alg`** which is presented by **`R`**.
- The left **`Alg`**-module which is considered by ***HilbertSeries*** is the factor module of the free **`Alg`**-module of row vectors whose length equals the number of columns of **`R`** modulo the submodule which is generated by the rows of **`R`**.
- The Hilbert series of a finitely presented left module over **`Alg`** is defined as follows: The Ore algebra **`Alg`** has a increasing filtration defined by the order of its elements, namely an increasing sequence of vector subspaces of **`Alg`** such that the union of these vector subspaces is **`Alg`** and the product of any element in the i-th vector subspace by any element in the j-th vector subspace lies in the $(i+j)$-th vector subspace, where the k-th vector subspace consists of zero and all elements of **`Alg`** of order less than or equal to k. (For the Weyl algebra this is the Bernstein filtration). The Hilbert series of a finitely presented left **`Alg`**-module is the generating function of the (vector space) dimensions of the homogeneous components of the graded module associated to this left **`Alg`**-module with respect to the above filtration.
- The result of ***HilbertSeries*** is a formal power series in **`s`** such that the coefficient of **`s`**^i is the dimension of the vector space containing zero and all elements of order i (in the sense explained above) in the left **`Alg`**-module presented by **`R`**.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- **`s`** is the indeterminate for the resulting Hilbert series.
- ***HilbertSeriesRat*** performs the same computations as ***HilbertSeries***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
```

The Hilbert series of the Ore algebra **`Alg`** is:

```
> HilbertSeries([[0]], Alg, lambda);
```

$$\frac{1}{(-1+\lambda)^4}$$

The Hilbert series of the algebra obtained by replacing the domain of coefficients of **`Alg`** by its quotient field is:

```
> HilbertSeriesRat([[0]], Alg, lambda);
```

$$\frac{1}{(-1+\lambda)^2}$$

Example 2:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[D1, 0], [D2, D1], [0, D2]]);
```

$$R := \begin{bmatrix} D1 & 0 \\ D2 & D1 \\ 0 & D2 \end{bmatrix}$$

```
> HilbertSeries(R, Alg, lambda);
```

$$\frac{1}{(-1+\lambda)^2} + \frac{\lambda+1}{(-1+\lambda)^2}$$

```
> taylor(%, lambda=0, 12);
```

$$2 + 5\lambda + 8\lambda^2 + 11\lambda^3 + 14\lambda^4 + 17\lambda^5 + 20\lambda^6 + 23\lambda^7 + 26\lambda^8 + 29\lambda^9 + 32\lambda^{10} + 35\lambda^{11} + O(\lambda^{12})$$

```
> HilbertSeriesRat(R, Alg, lambda);
```

$$2 + \lambda$$

Hence, the dimension of the vector space containing zero and all elements of order zero in the left module presented by **R** over the Ore algebra **Alg** with rational functions in *x1*, *x2* as coefficients is 2. Similarly, the dimension of the vector space containing zero and all elements of order one is 1. The list which is returned by KBasis contains bases of these vector spaces:

```
> KBasis(R, Alg);
```

$$[\lambda_1, \lambda_2, \lambda_2\, D1]$$

Example 3:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := matrix([[D1^3, x1], [D2, x1+D1], [D3, D2]]);
```

$$R := \begin{bmatrix} D1^3 & x1 \\ D2 & x1 + D1 \\ D3 & D2 \end{bmatrix}$$

```
> HilbertSeries(R, Alg, lambda);
```

$$-\frac{\lambda^2+\lambda+1}{(-1+\lambda)^3} - \frac{\lambda^3+2\lambda^2+2\lambda+1}{(-1+\lambda)^3}$$

```
> HilbertSeriesRat(R, Alg, lambda);
```

$$3\lambda^2 + 3\lambda + 2 + \lambda^3$$

```
> KBasis(R, Alg);
```

$$[\lambda_1, D1\,\lambda_1, D1^2\,\lambda_1, \lambda_2, \lambda_2\, D2, \lambda_2\, D1, D1\, D2\, \lambda_2, D1^2\,\lambda_2, \lambda_2\, D1^3]$$

Example 4:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := matrix([[D1^3], [D2], [D3+D1]]);
```

$$R := \begin{bmatrix} D1^3 \\ D2 \\ D3 + D1 \end{bmatrix}$$

```
> HilbertSeriesRat(R, Alg, lambda);
```

$$\lambda^2 + \lambda + 1$$

```
> KBasis(R, Alg);
```

$$[\lambda_1, \lambda_1\, D3, D3^2\,\lambda_1]$$

```
> R := matrix([[D1^3], [D2+x1], [D3+D1]]);
```

$$R := \begin{bmatrix} D1^3 \\ D2 + x1 \\ D3 + D1 \end{bmatrix}$$

```
> HilbertSeries(R, Alg, lambda);
```

$$0$$

```
> KBasis(R, Alg);
```

$$[\ ]$$

See Also:

DefineOreAlgebra, KBasis, Connection, Dimension, OreRank, Factorize, Quotient, ReduceMatrix, Elimination, Integrability, Involution, SyzygyModule.

OreModules[IdealIntersection] - intersect two left ideals of an Ore algebra

Calling Sequence:

IdealIntersection(L1,L2,Alg)

Parameters:

`L1`	-	list of polynomials in **`Alg`**
`L2`	-	list of polynomials in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***IdealIntersection*** computes a Groebner basis (w.r.t. the degree-reverse lexicographical term order) of the intersection of the left ideals generated by **`L1`** and **`L2`** in the Ore algebra defined by **`Alg`**.
- **`L1`** and **`L2`** are lists whose entries are polynomials in **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***IdealIntersection*** is a list of polynomials in **`Alg`**.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> L1 := [x1*D2+x2*D1-1]; L2 := [D1-D2];
```

$$L1 := [x1\, D2 + x2\, D1 - 1]$$

$$L2 := [D1 - D2]$$

```
> IdealIntersection(L1, L2, Alg);
```

$$[x1\, D2\, D1 - x1\, D2^2 + x2\, D1^2 - x2\, D2\, D1 - 2\, D1 + 2\, D2]$$

```
> L1 := [D2^2+D1,D1^2-1]; L2 := [D1^2-D2-1];
```

$$L1 := [D2^2 + D1, D1^2 - 1]$$

$$L2 := [D1^2 - D2 - 1]$$

```
> IdealIntersection(L1, L2, Alg);
```

$$[D1^3 - D1\, D2 - D1 + D2^2\, D1^2 - D2^3 - D2^2, -2\, D1^2 + D2 + 1 + D1^4 - D2\, D1^2]$$

See Also:

DefineOreAlgebra, PolIntersect, Mult, ApplyMatrix, Involution, KroneckerProduct, Factorize, Quotient, Elimination, Integrability, ReduceMatrix, SyzygyModule.

OreModules[Integrability] - compute Groebner basis of the rows of a matrix over an Ore algebra

Calling Sequence:

Integrability(R,Alg)

Parameters:

`R` - matrix with entries in **`Alg`** or INJ(*n*) or SURJ(*n*), where *n* is a non-negative integer

`Alg` - Ore algebra (given by DefineOreAlgebra)

Description:

- ***Integrability*** returns the Groebner basis of the left module over the Ore algebra **`Alg`** which is generated by the left hand sides of **`R`** $\lambda - \mu = 0$, where λ and μ are vectors of suitable dimensions. The Groebner basis is computed w.r.t. an elimination ordering which eliminates the indeterminates introduced to represent the standard basis vectors, i.e. the λ_i.
- Each elements of the resulting Groebner basis which contains no λ_i (i.e., which is a linear combination of the μ_i only) is a syzygy of the rows of **`R`** (cf. also SyzygyModule). Hence, ***Integrability*** can be used to study formal integrability of linear systems of PDE.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result is a list of linear combinations of $\lambda_1, ..., \lambda_p, \mu_1, ..., \mu_q$ with coefficients in **`Alg`**. Here *p* (resp. *q*) equals the number of columns (resp. rows) of **`R`**.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := matrix([[D1, 0, 0], [-D2, D1, 1], [0, D1, 1]]);
```

$$R := \begin{bmatrix} D1 & 0 & 0 \\ -D2 & D1 & 1 \\ 0 & D1 & 1 \end{bmatrix}$$

```
> G := Integrability(R, Alg);
```

$$G := [D1\,\mu_2 + D2\,\mu_1 - D1\,\mu_3, \lambda_2\,D1 + \lambda_3 - \mu_3, \lambda_1\,D2 + \mu_2 - \mu_3, \lambda_1\,D1 - \mu_1]$$

```
> remove(has, parse(convert(G, string)), lambda);
```

$$[D1\,\mu_2 + D2\,\mu_1 - D1\,\mu_3]$$

```
> SyzygyModule(R, Alg);
```

$$\begin{bmatrix} D2 & D1 & -D1 \end{bmatrix}$$

See Also:

DefineOreAlgebra, Factorize, Quotient, ReduceMatrix, Elimination, Involution, SyzygyModule.

OreModules[IntTorsion],

OreModules[IntTorsionRat] - integrate the torsion elements of a linear system of partial differential equations

Calling Sequence:

IntTorsion(R,Alg)
IntTorsionRat(R,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- In order to find a parametrization of a linear system **`R`** $y = 0$ of partial differential equations having autonomous elements, **`R`** can be split as a product $R1\,R2$ such that the system **`R`** $y = 0$ is equivalent to $R1\,\tau = 0$ and $\tau = R2\,\eta$, namely $R2$ is a presentation matrix of the left **`Alg`** -module M which is associated with **`R`** $y = 0$ modulo its torsion submodule. $R2$ can be obtained as the second entry of the result of applying Exti for $i = 1$ to the formal adjoint (see Involution) of **`R`**, and $R1$ can be computed by applying Factorize to **`R`** and $R2$.
- ***IntTorsion*** integrates the torsion elements, i.e., it solves (essentially) the homogeneous linear system $R1\,\tau = 0$ for the vector of functions τ.
- Those rows of $R2$ whose residue classes in M are zero are omitted by ***IntTorsion***. Hence, strictly speaking, ***IntTorsion*** solves $R1\,\tau = 0$ for the functions in the vector τ which correspond to non-zero torsion elements in M.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- First ***IntTorsion*** computes a generating set of the torsion submodule of the left **`Alg`**-module M which is associated with **`R`** $y = 0$. The torsion elements of M are in bijective correspondence to the autonomous elements of the system (see also TorsionElements, AutonomousElements). Then ***IntTorsion*** computes a generating set of the left **`Alg`**-relations satisfied by the generating set of autonomous elements. This linear system of partial differential equations is then solved for the autonomous elements using pdsolve.
- The result of ***IntTorsion*** is a list with three entries. If the left **`Alg`**-module M which is associated with the linear system is torsion-free, i.e. if the system has no autonomous elements, then the first two entries of the result equal the empty list.
- In any case, the third entry of the result is a matrix with entries in **`Alg`** having the same number of columns as **`R`**. The residue classes in M of the rows of this matrix generate the torsion submodule of M.
- If the given linear system has autonomous elements, then the first entry of the result of ***IntTorsion*** is a matrix with entries in **`Alg`** whose rows generate the left **`Alg`**-relations satisfied by the generating set of autonomous elements which corresponds to the generating set of torsion elements given by the rows of the third entry of the result, i.e. this matrix constitutes the linear system of partial differential equations which is solved by ***IntTorsion***.
- The second entry of the result of ***IntTorsion*** is the solution (which may depend on arbitrary functions and constants) to the above linear system of partial differential equations if it could be found by pdsolve. Otherwise the second entry of the result is the empty list. Even in case pdsolve could only partially solve the system, the second entry of the result gives the result of pdsolve.
- To continue parametrizing the linear system **`R`** $y = 0$, a particular solution to $R2\,\eta = \tau$ has to be found. This is done by ParticularSolution . ***IntTorsion*** and ParticularSolution are used by Parametrization, if the system has autonomous elements.
- ***IntTorsionRat*** performs the same computations as ***IntTorsion***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- For more details see A. Quadrat, D. Robertz, "Parametrizing all solutions of uncontrollable multidimensional linear systems", Proceedings of the 16th IFAC World Congress, Prague, 2005.

Examples:

```
> with(OreModules):
```

Example 1: Ordinary differential equations

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l]):
> R := evalm([[D^2+g/l, 0, -g/l], [0, D^2+g/l, -g/l]]);
```

$$R := \begin{bmatrix} D^2 + \frac{g}{l} & 0 & -\frac{g}{l} \\ 0 & D^2 + \frac{g}{l} & -\frac{g}{l} \end{bmatrix}$$

A generating set of the torsion submodule of the (left) **`Alg`**-module *M* which is associated with the given linear system can be obtained as follows:

```
> TorsionElements(R, [x1(t),x2(t),u(t)], Alg);
```

$$\left[\left[g\,\theta_1(t) + l\left(\frac{d^2}{dt^2}\theta_1(t)\right) = 0\right], [\theta_1(t) = x1(t) - x2(t)]\right]$$

Equivalently, one can compute the first extension module with values in **`Alg`** of left **`Alg`**-module presented by the formal adjoint of **`R`** :

```
> Ext := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext := \left[\begin{bmatrix} D^2\,l + g & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 & 0 \\ 0 & D^2\,l + g & -g \end{bmatrix}, \begin{bmatrix} g \\ g \\ D^2\,l + g \end{bmatrix}\right]$$

Then, **`R`** can be split as a product *R1 R2* as follows:

```
> R2 := evalm(Ext[2]);
```

$$R2 := \begin{bmatrix} 1 & -1 & 0 \\ 0 & D^2\,l + g & -g \end{bmatrix}$$

```
> R1 := Factorize(R, R2, Alg);
```

$$R1 := \begin{bmatrix} D^2 + \frac{g}{l} & \frac{1}{l} \\ 0 & \frac{1}{l} \end{bmatrix}$$

```
> Mult(R1, R2, Alg);
```

$$\begin{bmatrix} \frac{D^2\,l + g}{l} & 0 & -\frac{g}{l} \\ 0 & \frac{D^2\,l + g}{l} & -\frac{g}{l} \end{bmatrix}$$

The residue class in *M* of the second row of *R2* is zero. The residue class of the first row of *R2* is a non-zero torsion element of *M*. Hence, ***IntTorsion*** solves $R1_{1,1}\,\tau_1 = 0$.

```
> IntTorsion(R, Alg);
```

$$\left[[D^2\,l + g], \left[\_C1 \sin\left(\frac{\sqrt{g}\,t}{\sqrt{l}}\right) + \_C2 \cos\left(\frac{\sqrt{g}\,t}{\sqrt{l}}\right)\right], [1 \quad -1 \quad 0]\right]$$

We find: A generating set of the left **`Alg`**-relations satisfied by the autonomous elements, the integrated torsion elements, and a generating set of torsion elements of *M*.

The information given by ***IntTorsion*** is used by Parametrization if the considered system has autonomous elements:

```
> Parametrization(R, Alg);
```

$$\begin{bmatrix} \_C1 \sin\left(\frac{\sqrt{g}\, t}{\sqrt{l}}\right) + \_C2 \cos\left(\frac{\sqrt{g}\, t}{\sqrt{l}}\right) + g\, \xi_1(t) \\ g\, \xi_1(t) \\ g\, \xi_1(t) + l \left(\frac{d^2}{dt^2} \xi_1(t)\right) \end{bmatrix}$$

Example 2: Partial differential equations

Linear system of PDEs that appears in mathematical physics, namely in the study of Lie-Poisson structures (see C. M. Bender, G. V. Dunne, L. R. Mead, *Underdetermined systems of partial differential equations*, Journal of Mathematical Physics, vol. 41, no. 9 (2000), pp. 6388-6398 and W. M. Seiler, *Involution analysis of the partial differential equations characterising Hamiltonian vector fields*, Journal of Mathematical Physics, vol. 44 (2003), pp. 1173-1182):

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[x1*D3, x2*D3, 0], [-x1*D2+x2*D1, -1, x2*D3], [-1, -x2*D1+x1*D2, x1*D3]]);
```

$$R := \begin{bmatrix} x1\, \mathrm{D3} & x2\, \mathrm{D3} & 0 \\ -x1\, \mathrm{D2} + x2\, \mathrm{D1} & -1 & x2\, \mathrm{D3} \\ -1 & -x2\, \mathrm{D1} + x1\, \mathrm{D2} & x1\, \mathrm{D3} \end{bmatrix}$$

A generating set of the torsion submodule of the left **Alg**-module *M* which is associated with the given linear system can be obtained as follows:

```
> TorsionElements(R, [F(x1,x2,x3),G(x1,x2,x3),H(x1,x2,x3)], Alg);
```

$$\left[\begin{bmatrix} \frac{\partial}{\partial x3} \theta_1(x1, x2, x3) = 0 \\ -x2 \left(\frac{\partial}{\partial x1} \theta_1(x1, x2, x3)\right) + x1 \left(\frac{\partial}{\partial x2} \theta_1(x1, x2, x3)\right) = 0 \\ x2 \left(\frac{\partial}{\partial x3} \theta_2(x1, x2, x3)\right) = 0 \\ x1 \left(\frac{\partial}{\partial x3} \theta_2(x1, x2, x3)\right) = 0 \\ -x2 \left(\frac{\partial}{\partial x1} \theta_2(x1, x2, x3)\right) + x1 \left(\frac{\partial}{\partial x2} \theta_2(x1, x2, x3)\right) = 0 \end{bmatrix}, \begin{bmatrix} \theta_1(x1, x2, x3) = x1\, \mathrm{F}(x1, x2, x3) + x2\, \mathrm{G}(x1, x2, x3) \\ \theta_2(x1, x2, x3) = \left(\frac{\partial}{\partial x1} \mathrm{F}(x1, x2, x3)\right) + \left(\frac{\partial}{\partial x2} \mathrm{G}(x1, x2, x3)\right) + \left(\frac{\partial}{\partial x3} \mathrm{H}(x1, x2, x3)\right) \end{bmatrix}\right]$$

Equivalently, one can compute the first extension module with values in **Alg** of left **Alg**-module presented by the formal adjoint of **R** :

```
> Ext := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext := \left[\begin{bmatrix} D3 & 0 & 0 \\ -x2\,D1 + x1\,D2 & 0 & 0 \\ 0 & x2\,D3 & 0 \\ 0 & x1\,D3 & 0 \\ 0 & -x2\,D1 + x1\,D2 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} x1 & x2 & 0 \\ D1 & D2 & D3 \\ -1 & -x2\,D1 + x1\,D2 & x1\,D3 \end{bmatrix}, \begin{bmatrix} -x2\,D3 \\ x1\,D3 \\ -x1\,D2 + x2\,D1 \end{bmatrix}\right]$$

Then, **R** can be split as a product $R1\,R2$ as follows:

```
> R2 := evalm(Ext[2]);
```

$$R2 := \begin{bmatrix} x1 & x2 & 0 \\ D1 & D2 & D3 \\ -1 & -x2\,D1 + x1\,D2 & x1\,D3 \end{bmatrix}$$

```
> R1 := Factorize(R, R2, Alg);
```

$$R1 := \begin{bmatrix} D3 & 0 & 0 \\ -D2 & x2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> Mult(R1, R2, Alg);
```

$$\begin{bmatrix} x1\,D3 & x2\,D3 & 0 \\ -x1\,D2 + x2\,D1 & -1 & x2\,D3 \\ -1 & -x2\,D1 + x1\,D2 & x1\,D3 \end{bmatrix}$$

The residue class in M of the third row of $R2$ is zero. The residue classes of the first and second row of $R2$ are non-zero torsion elements of M. Hence, ***IntTorsion*** solves $R1\,\tau = 0$ for the first two components of τ.

```
> IntTorsion(R, Alg);
```

$$\left[\begin{bmatrix} D3 & 0 \\ D2 & -x2 \\ D1 & -x1 \\ 0 & x2\,D3 \\ 0 & x1\,D3 \\ 0 & -x2\,D1 + x1\,D2 \end{bmatrix}, \begin{bmatrix} \int x1\,\_F1(x1^2 + x2^2)\,dx1 + \int x2\left(-2\int D(\_F1)(x1^2 + x2^2)\,x1\,dx1 + \_F1(x1^2 + x2^2)\right)dx2 + \_C1 \\ \_F1(x1^2 + x2^2) \end{bmatrix}, \begin{bmatrix} x1 & x2 & 0 \\ D1 & D2 & D3 \end{bmatrix}\right]$$

We find: A generating set of the left **`Alg`**-relations satisfied by the autonomous elements, the integrated torsion elements, and a generating set of torsion elements of M.

The information given by ***IntTorsion*** is used by Parametrization if the considered system has autonomous elements:

```
> Parametrization(R, Alg);
```

$$\text{table}([1 = \begin{bmatrix} \eta_1(x1, x2, x3) - x2 \left(\frac{\partial}{\partial x3} \xi_1(x1, x2, x3)\right) \\ \eta_2(x1, x2, x3) + x1 \left(\frac{\partial}{\partial x3} \xi_1(x1, x2, x3)\right) \\ \eta_3(x1, x2, x3) + x2 \left(\frac{\partial}{\partial x1} \xi_1(x1, x2, x3)\right) - x1 \left(\frac{\partial}{\partial x2} \xi_1(x1, x2, x3)\right) \end{bmatrix},$$

$$2 = \left(\begin{bmatrix} x1\, \eta_1(x1, x2, x3) + x2\, \eta_2(x1, x2, x3) \\ \left(\frac{\partial}{\partial x1} \eta_1(x1, x2, x3)\right) + \left(\frac{\partial}{\partial x2} \eta_2(x1, x2, x3)\right) + \left(\frac{\partial}{\partial x3} \eta_3(x1, x2, x3)\right) \\ -\eta_1(x1, x2, x3) - x2 \left(\frac{\partial}{\partial x1} \eta_2(x1, x2, x3)\right) + x1 \left(\frac{\partial}{\partial x2} \eta_2(x1, x2, x3)\right) + x1 \left(\frac{\partial}{\partial x3} \eta_3(x1, x2, x3)\right) \end{bmatrix} = \right.$$

$$\left.\begin{bmatrix} \int x1\, \_\text{F1}(x1^2 + x2^2)\, dx1 + \int x2 \left(-2 \int \text{D}(\_F1)(x1^2 + x2^2)\, x1\, dx1 + \_\text{F1}(x1^2 + x2^2)\right) dx2 + \_C1 \\ \_\text{F1}(x1^2 + x2^2) \\ 0 \end{bmatrix}\right)$$

$$])$$

See Also:

DefineOreAlgebra, Parametrization, ParticularSolution, Complement, MinimalParametrization, Exti, Extn, Torsion, TorsionElements, Factorize, PiPolynomial, AutonomousElements.

OreModules[Involution] - apply involution of the Ore algebra to a matrix

Calling Sequence:

Involution(M,Alg)

Parameters:

`M` - matrix with entries in **`Alg`** or INJ(n) or SURJ(n) or ZERO, where n is a non-negative integer

`Alg` - Ore algebra (given by DefineOreAlgebra)

Description:

- ***Involution*** applies an involution of **`Alg`** to **`M`**.
- For each Ore algebra **`Alg`** the involution is fixed in OreModules. The matrix **`M`** is transposed and (an extension of) Ore_algebra[dual_polynomial] is applied to each entry. Then, indeterminates representing differential or shift operators are mapped to their negative, whereas the indeterminates occuring on the left in normal forms of elements in **`Alg`** are mapped to themselves. If the option "shift+dual_shift" is used in DefineOreAlgebra to declare a shift and an advance operator at the same time, then the involution is just transposition of matrices.
- **`M`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result is a matrix whose shape is the transposed shape of **`M`**.

Examples:

```
> with(OreModules):
```

Example 1: Ordinary differential equations

```
> Alg := DefineOreAlgebra(diff=[D,x], polynom=[x]):
> M := matrix([[D, x], [0, x*D+D^2]]);
```

$$M := \begin{bmatrix} D & x \\ 0 & x\,D + D^2 \end{bmatrix}$$

```
> Involution(M, Alg);
```

$$\begin{bmatrix} -D & 0 \\ x & -1 - x\,D + D^2 \end{bmatrix}$$

Example 2: Shift and advance operator

```
> Alg := DefineOreAlgebra(‘shift+dual_shift‘=[tau,delta,t]):
> M := matrix([[delta, 0, t], [0, t*delta+tau^2, 1]]);
```

$$M := \begin{bmatrix} \delta & 0 & t \\ 0 & t\,\delta + \tau^2 & 1 \end{bmatrix}$$

```
> Involution(M, Alg);
```

$$\begin{bmatrix} \delta & 0 \\ 0 & t\,\delta + \tau^2 \\ t & 1 \end{bmatrix}$$

See Also:

DefineOreAlgebra, Ore_algebra[dual_polynomial], linalg[transpose], Mult, ApplyMatrix, KroneckerProduct, ReduceMatrix, LeftInverse, RightInverse, GeneralizedInverse.

OreModules[KalmanSystem] - check structural properties of a Kalman system

Calling Sequence:

KalmanSystem(A,B,Alg)
KalmanSystem(n,m,Alg)

Parameters:

`A, B`	-	matrices with entries in **`Alg`**
`n, m`	-	positive integers
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***KalmanSystem*** returns structural information about a linear system of ordinary differential equations in Kalman form. The system in Kalman form is either given explicitly in terms of two matrices **A** and **B** with entries in **`Alg`**, or a generic Kalman system of prescribed dimension is considered.
- **A** and **B** are matrices with entries in the Ore algebra **`Alg`**. They represent a linear system of ordinary differential equations in Kalman form $D\,x = \mathbf{A}\,x + \mathbf{B}\,u$, where x is the system variable representing the state of the system, u is the system variable representing the input of the system, and D is the diagonal matrix with the differential operator with respect to time on the diagonal. Therefore, **A** must be a square matrix. If **A** and **B** are provided, then **n** is set to the number of columns of **A**, and **m** is set to the number of columns of **B**.
- **n** and **m** are positive integers which define the dimension of state space respectively input space. If ***KalmanSystem*** is called with positive integers **n** and **m**, then an **n** x **n** matrix **A** with entries *A1*, *A2*, ..., *A*(**n*****n**) (enumerated row by row) and an **n** x **m** matrix **B** with entries *B1*, *B2*, ..., *B*(**n*****m**) are defined. In this case, the parameter **`Alg`** is optional. If **`Alg`** is not provided, then a suitable Ore algebra with indeterminates D and *t* is defined.
- In any case, the system matrix is formed by juxtaposing D - **A** and **B**, where D is the diagonal matrix with the differential operator with respect to time.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***KalmanSystem*** is a list with five entries.
- The first entry of the result gives the result of TorsionElements applied to the system matrix of the Kalman system and system variables *x1*, ..., *xn*, *u1*, ..., *um*, where **n** is the dimension of the state space and **m** is the dimension of the input space.
- The second entry of the result gives the result of Parametrization applied to the system matrix of the Kalman system.
- The third entry of the result gives the result of LeftInverse applied to the third entry of Exti applied to the adjoint of the system matrix (obtained by Involution) and $i = 1$, i.e. the left inverse (if it exists) of the parametrization of the system computed along with the first extension module with values in **`Alg`** of the adjoint of the left **`Alg`**-module associated with the system.
- The fourth entry of the result gives the result of RightInverse applied to the system matrix of the Kalman system.
- The fifth entry of the result is the Ore algebra which is used by ***KalmanSystem***. (In case, **A** and **B** are defined by ***KalmanSystem*** as above, then **`Alg`** needs to be adapted.)

Examples:

```
> with(OreModules):
```

Example 1:

We study the linear Kalman system of a satellite in a circular equatorial orbit in one go (see T. Kailath, *Linear Systems*, Prentice-Hall, 1980, p. 60 and 145; here mass and radius are set to 1; ω is the angular velocity):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[omega]):
> mA := evalm([[0,1,0,0],[3*omega^2,0,0,2*omega],[0,0,0,1],[0,-2*omega,0,0]]);
```

$$mA := \begin{bmatrix} 0 & 1 & 0 & 0 \\ 3\,\omega^2 & 0 & 0 & 2\,\omega \\ 0 & 0 & 0 & 1 \\ 0 & -2\,\omega & 0 & 0 \end{bmatrix}$$

```
> mB := evalm([[0,0],[1,0],[0,0],[0,1]]);
```

$$mB := \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> ApplyMatrix(evalm([[D,0,0,0],[0,D,0,0],[0,0,D,0],[0,0,0,D]]),
  [x1(t),x2(t),x3(t),x4(t)], Alg)=
  ApplyMatrix(mA, [x1(t),x2(t),x3(t),x4(t)], Alg) + ApplyMatrix(mB, [u1(t),u2(t)], Alg);
```

$$\begin{bmatrix} \frac{d}{dt}\,\mathrm{x1}(t) \\ \frac{d}{dt}\,\mathrm{x2}(t) \\ \frac{d}{dt}\,\mathrm{x3}(t) \\ \frac{d}{dt}\,\mathrm{x4}(t) \end{bmatrix} = \begin{bmatrix} \mathrm{x2}(t) \\ 3\,\omega^2\,\mathrm{x1}(t) + 2\,\omega\,\mathrm{x4}(t) \\ \mathrm{x4}(t) \\ -2\,\omega\,\mathrm{x2}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \mathrm{u1}(t) \\ 0 \\ \mathrm{u2}(t) \end{bmatrix}$$

```
> K := KalmanSystem(mA, mB, Alg):
```

There are no torsion elements of the system, i.e. the system is controllable:

```
> K[1];
```

$$[\;]$$

Parametrization of the system:

```
> K[2];
```

$$\begin{bmatrix} \xi_1(t) \\ \frac{d}{dt}\,\xi_1(t) \\ \xi_2(t) \\ \frac{d}{dt}\,\xi_2(t) \\ -3\,\omega^2\,\xi_1(t) + \left(\frac{d^2}{dt^2}\,\xi_1(t)\right) - 2\,\omega\left(\frac{d}{dt}\,\xi_2(t)\right) \\ 2\,\omega\left(\frac{d}{dt}\,\xi_1(t)\right) + \left(\frac{d^2}{dt^2}\,\xi_2(t)\right) \end{bmatrix}$$

There is a parametrization of the system using two free parameters ξ_1, ξ_2.

A flat output of the system can be obtained, if possible, as a left inverse of a parametrization. Then it is returned as third entry of the result:

```
> K[3];
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Right inverse of system matrix:

```
> K[4];
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -D & -1 & 2\,\omega & 0 \\ -2\,\omega & 0 & -D & -1 \end{bmatrix}$$

Example 2:

We study a bipendulum, namely a system composed of a bar where two pendula are fixed. Here we only consider the case, where both pendula have the same length *l*.

For more details, see J.-F. Pommaret, *Partial Differential Control Theory*, Kluwer, 2001, p. 569, and the Library of Examples at http://wwwb.math.rwth-aachen.de/OreModules.

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g, l]):
> mA := evalm([[0,0,1,0], [0,0,0,1], [-g/l,0,0,0], [0,-g/l,0,0]]);
```

$$mA := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{g}{l} & 0 & 0 & 0 \\ 0 & -\frac{g}{l} & 0 & 0 \end{bmatrix}$$

```
> mB := evalm([[0], [0], [g/l], [g/l]]);
```

$$mB := \begin{bmatrix} 0 \\ 0 \\ \frac{g}{l} \\ \frac{g}{l} \end{bmatrix}$$

```
> ApplyMatrix(evalm([[D,0,0,0],[0,D,0,0],[0,0,D,0],[0,0,0,D]]),
  [x1(t),x2(t),x3(t),x4(t)], Alg)=
  ApplyMatrix(mA, [x1(t),x2(t),x3(t),x4(t)], Alg)+ApplyMatrix(mB, [u(t)], Alg);
```

$$\begin{bmatrix} \frac{d}{dt}\,\mathrm{x1}(t) \\ \frac{d}{dt}\,\mathrm{x2}(t) \\ \frac{d}{dt}\,\mathrm{x3}(t) \\ \frac{d}{dt}\,\mathrm{x4}(t) \end{bmatrix} = \begin{bmatrix} \mathrm{x3}(t) \\ \mathrm{x4}(t) \\ -\frac{g\,\mathrm{x1}(t)}{l} \\ -\frac{g\,\mathrm{x2}(t)}{l} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{g\,\mathrm{u}(t)}{l} \\ \frac{g\,\mathrm{u}(t)}{l} \end{bmatrix}$$

```
> K := KalmanSystem(mA, mB, Alg):
```

Torsion elements in terms of the system variables *x1*, *x2*, *x3*, *x4*, *u*:

```
> K[1];
```

$$\left[\left[\begin{array}{l} g\,\theta_1(t)+l\left(\frac{d^2}{dt^2}\theta_1(t)\right)=0 \\ g\,\theta_4(t)+l\left(\frac{d^2}{dt^2}\theta_4(t)\right)=0 \end{array}\right], \left[\begin{array}{l} \theta_1(t)=\mathrm{x1}(t)-\mathrm{x2}(t) \\ \theta_4(t)=\mathrm{x3}(t)-\mathrm{x4}(t) \end{array}\right]\right]$$

Parametrization of the linear system obtained by equating the torsion elements to zero:

```
> K[2];
```

$$\left[\begin{array}{c} \frac{-\sqrt{l}\,\_C1\cos\left(\frac{\sqrt{g}\,t}{\sqrt{l}}\right)+\sqrt{l}\,\_C2\sin\left(\frac{\sqrt{g}\,t}{\sqrt{l}}\right)+g^{(3/2)}\xi_1(t)}{\sqrt{g}} \\ g\,\xi_1(t) \\ \_C1\sin\left(\frac{\sqrt{g}\,t}{\sqrt{l}}\right)+\_C2\cos\left(\frac{\sqrt{g}\,t}{\sqrt{l}}\right)+g\left(\frac{d}{dt}\xi_1(t)\right) \\ g\left(\frac{d}{dt}\xi_1(t)\right) \\ g\,\xi_1(t)+l\left(\frac{d^2}{dt^2}\xi_1(t)\right) \end{array}\right]$$

Flat output of the linear system obtained by equating the torsion elements to zero:

```
> K[3];
```

$$\left[\begin{array}{ccccc} 0 & \frac{1}{g} & 0 & 0 & 0 \end{array}\right]$$

A right inverse of system matrix does not exist:

```
> K[4];
```

[ ]

Example 3:

We only give the dimensions of state and input space and let OreModules check the corresponding generic Kalman system:

```
> K := KalmanSystem(2, 1):
```

Generically, there are no torsion elements:

```
> K[1];
```

[ ]

Parametrization of the generic Kalman system:

```
> K[2];
```

$$\left[\begin{array}{c} \xi_1(t)\,A2\,B2-\xi_1(t)\,B1\,A4+B1\left(\frac{d}{dt}\xi_1(t)\right) \\ \xi_1(t)\,B1\,A3-\xi_1(t)\,B2\,A1+B2\left(\frac{d}{dt}\xi_1(t)\right) \\ -\xi_1(t)\,A2\,A3+\xi_1(t)\,A1\,A4-\left(\frac{d}{dt}\xi_1(t)\right)A1-\left(\frac{d}{dt}\xi_1(t)\right)A4+\left(\frac{d^2}{dt^2}\xi_1(t)\right) \end{array}\right]$$

Flat output:

```
> K[3];
```

$$\left[\begin{array}{ccc} \frac{B2}{-B1^2\,A3+A2\,B2^2-B2\,B1\,A4+B1\,A1\,B2} & -\frac{B1}{-B1^2\,A3+A2\,B2^2-B2\,B1\,A4+B1\,A1\,B2} & 0 \end{array}\right]$$

Right inverse of system matrix:

```
> K[4];
```

$$\begin{bmatrix} -\frac{B2\,B1}{-B1^2\,A3+A2\,B2^2-B2\,B1\,A4+B1\,A1\,B2} & \frac{B1^2}{-B1^2\,A3+A2\,B2^2-B2\,B1\,A4+B1\,A1\,B2} \\ -\frac{B2^2}{-B1^2\,A3+A2\,B2^2-B2\,B1\,A4+B1\,A1\,B2} & \frac{B2\,B1}{-B1^2\,A3+A2\,B2^2-B2\,B1\,A4+B1\,A1\,B2} \\ -\frac{-B1\,A3+B2\,\mathrm{D}-B2\,A4}{-B1^2\,A3+A2\,B2^2-B2\,B1\,A4+B1\,A1\,B2} & \frac{-A2\,B2-B1\,A1+B1\,\mathrm{D}}{-B1^2\,A3+A2\,B2^2-B2\,B1\,A4+B1\,A1\,B2} \end{bmatrix}$$

See Also:

DefineOreAlgebra, Exti, Extn, Torsion, TorsionElements, AutonomousElements, Parametrization, LeftInverse, RightInverse, Mult, ApplyMatrix, ControllabilityMatrix, FirstIntegral, LQEquations, FinalConditions.

OreModules[KBasis] - return a vector space basis of a finite dimensional factor module over an Ore algebra

Calling Sequence:

KBasis(R,Alg)

Parameters:

R	-	matrix with entries in **Alg** or INJ(*n*) or SURJ(*n*), where *n* is a non-negative integer
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***KBasis*** returns a vector space basis of the finite dimensional left module which is presented by **R**.
- Independently of the definition of the coefficient domain of **Alg**, ***KBasis*** uses the Ore algebra which is obtained from **Alg** by replacing the coefficient domain by its quotient field, i.e. rational functions. The left module which is considered by ***KBasis***, namely the module presented by **R**, is the factor module of the free module of row vectors over this Ore algebra whose length equals the number of columns of **R** modulo the submodule which is generated by the rows of **R**.
- If the module presented by **R** is the zero module, then ***KBasis*** returns the empty list. Otherwise the result is a list of multiples of λ_1, ..., λ_p by monomials in **Alg**, where *p* is the number of columns of **R**. If λ_1, ..., λ_p are interpreted as the standard basis vectors of the free module of rank *p*, then the residue classes which are represented by the entries of the result of ***KBasis*** form a vector space basis of the module presented by **R**.
- **R** is a matrix with entries in the Ore algebra **Alg**.
- **Alg** is expected to be defined using DefineOreAlgebra.
- Note that, for ***KBasis***, the domain of coefficients of the Ore algebra **Alg** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[D1, 0], [D2, D1], [0, D2]]);
```

$$R := \begin{bmatrix} \mathrm{D1} & 0 \\ \mathrm{D2} & \mathrm{D1} \\ 0 & \mathrm{D2} \end{bmatrix}$$

```
> KBasis(R, Alg);
```

$$[\lambda_1, \lambda_2, \lambda_2\,\mathrm{D1}]$$

Example 2:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := matrix([[D1^3, x1], [D2, x1+D1], [D3, D2]]);
```

$$R := \begin{bmatrix} \mathrm{D1}^3 & x1 \\ \mathrm{D2} & x1+\mathrm{D1} \\ \mathrm{D3} & \mathrm{D2} \end{bmatrix}$$

```
> KBasis(R, Alg);
```

$$[\lambda_1, \mathrm{D1}\,\lambda_1, \mathrm{D1}^2\,\lambda_1, \lambda_2, \lambda_2\,\mathrm{D2}, \lambda_2\,\mathrm{D1}, \mathrm{D1}\,\mathrm{D2}\,\lambda_2, \mathrm{D1}^2\,\lambda_2, \lambda_2\,\mathrm{D1}^3]$$

Example 3:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := matrix([[D1^3], [D2], [D3+D1]]);
```

$$R := \begin{bmatrix} D1^3 \\ D2 \\ D3 + D1 \end{bmatrix}$$

```
> KBasis(R, Alg);
```

$$[\lambda_1, \lambda_1 \, D3, D3^2 \, \lambda_1]$$

```
> R := matrix([[D1^3], [D2+x1], [D3+D1]]);
```

$$R := \begin{bmatrix} D1^3 \\ D2 + x1 \\ D3 + D1 \end{bmatrix}$$

```
> KBasis(R, Alg);
```

$$[\,]$$

See Also:

DefineOreAlgebra, Connection, HilbertSeries, Dimension, OreRank, Factorize, Quotient, ReduceMatrix, Elimination, Integrability, Involution, SyzygyModule.

OreModules[KroneckerProduct] - return the Kronecker product of two matrices over an Ore algebra

Calling Sequence:

KroneckerProduct(A,B,Alg)

Parameters:

`A, B`	-	matrices with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***KroneckerProduct*** returns the Kronecker product of the matrices **A** and **B**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D1,x1], polynom=[x1]):
> A := matrix(2,2,[D1,x1,D1,x1]);
```

$$A := \begin{bmatrix} \mathrm{D1} & x1 \\ \mathrm{D1} & x1 \end{bmatrix}$$

```
> B := matrix(2,2,[x1*D1+1,x1^2,0,D1^2]);
```

$$B := \begin{bmatrix} x1\,\mathrm{D1}+1 & x1^2 \\ 0 & \mathrm{D1}^2 \end{bmatrix}$$

```
> KroneckerProduct(A, B, Alg[1]);
```

$$\begin{bmatrix} x1\,\mathrm{D1}^2+2\,\mathrm{D1} & \mathrm{D1}\,x1^2+2\,x1 & \mathrm{D1}\,x1^2+x1 & x1^3 \\ 0 & \mathrm{D1}^3 & 0 & x1\,\mathrm{D1}^2 \\ x1\,\mathrm{D1}^2+2\,\mathrm{D1} & \mathrm{D1}\,x1^2+2\,x1 & \mathrm{D1}\,x1^2+x1 & x1^3 \\ 0 & \mathrm{D1}^3 & 0 & x1\,\mathrm{D1}^2 \end{bmatrix}$$

See Also:

DefineOreAlgebra, Mult, ApplyMatrix, DiffToOre, Involution, ReduceMatrix, LeftInverse, RightInverse, GeneralizedInverse.

OreModules[LeftInverse],

OreModules[LeftInverseRat] - compute a left inverse of a matrix over an Ore algebra

Calling Sequence:

LeftInverse(M,Alg)
LeftInverseRat(M,Alg)

Parameters:

`M`	-	matrix with entries in **`Alg`** or INJ(*n*) or SURJ(*n*), where *n* is a non-negative integer
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***LeftInverse*** computes (if possible) a left inverse of the matrix **`M`**, i.e. a matrix *L* with entries in **`Alg`** such that the product of *L* by **`M`** is the identity matrix.
- If no left inverse of **`M`** exists, ***LeftInverse*** returns the empty list.
- **`M`** is a matrix with entries in **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- ***LeftInverseRat*** performs the same computations as ***LeftInverse***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- Right inverses of matrices over Ore algebras are computed by RightInverse. Generalized inverses of matrices over Ore algebras are computed by GeneralizedInverse.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> M1 := evalm([[0,1],[1,0],[0,1]]);
```

$$M1 := \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> L1 := LeftInverse(M1, Alg);
```

$$L1 := \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> Mult(L1, M1, Alg);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> M2 := evalm([[x2*D1+1], [D2]]);
```

$$M2 := \begin{bmatrix} x2\,\mathrm{D1} + 1 \\ \mathrm{D2} \end{bmatrix}$$

```
> L2 := LeftInverse(M2, Alg);
```

$$L2 := [1 - x2\,\mathrm{D2} \quad x2^2\,\mathrm{D1} + x2]$$

```
> Mult(L2, M2, Alg);
```

$$[\ 1]$$

```
> M3 := Involution(M2, Alg);
```

$$M3 := [-x2\,\mathrm{D1} + 1 \quad -\mathrm{D2}]$$

```
> LeftInverse(M3, Alg);
```

$$[\,]$$

```
> LeftInverse(SURJ(3), Alg);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> LeftInverse(INJ(2), Alg);
```

$$ZERO$$

See Also:

DefineOreAlgebra, RightInverse, LocalLeftInverse, GeneralizedInverse, Mult, ApplyMatrix, Involution, KroneckerProduct, Factorize, Quotient, Elimination, Integrability, ReduceMatrix.

OreModules[LiftOperators],

OreModules[LiftOperatorsRat] - computes the lift operators for a linear operator defining a projective D-module

Calling Sequence:

LiftOperators(R,Alg)
LiftOperatorsRat(R,Alg)

Parameters:

`R` - matrix with entries in **`Alg`**
`Alg` - Ore algebra (given by DefineOreAlgebra)

Description:

- The fact that the linear operator represented by the matrix **`R`** over the Ore algebra **`Alg`** defines a projective module, i.e. that the cokernel of the map which multiplies rows to the left of the matrix **`R`** is projective, is characterized by the existence of a lift operator for this operator. A lift operator *P1* for the operator D1 represented by **`R`** is defined by the property $\mathrm{D1}\,P1\,\mathrm{D1}=\mathrm{D1}$ (as a composition of operators), i.e. in terms of matrices, a lift operator is represented by a generalized inverse of **`R`**. If **`R`** admits a right inverse, then such a right inverse represents a lift operator.
- ***LiftOperators*** constructs a free resolution of the left **`Alg`**-module presented by **`R`** using FreeResolution. Then, ***LiftOperators*** tries to compute a right inverse of the last morphism in the free resolution. If such a right inverse does not exist, then ***LiftOperators*** returns the empty list. Otherwise, ***LiftOperators*** returns a table of matrices, where the last matrix is a right inverse of the last morphism of the free resolution and the previous ones represent lift operators of the corresponding operators represented by the morphisms in the free resolution.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- ***LiftOperatorsRat*** performs the same computations as ***LiftOperators***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- Left (resp. right) inverses of matrices over Ore algebras are computed by LeftInverse (resp. RightInverse). Generalized inverses of matrices over Ore algebras are computed by GeneralizedInverse.
- For more details about the computation of lift operators, cf. J.-F. Pommaret, A. Quadrat, *Generalized Bezout Identity*, Applicable Algebra in Engineering, Communication and Computing 9 (1998), pp. 91-116.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[-x2*D1+1, D2]]);
```

$$R := \begin{bmatrix} -x2\,\mathrm{D1}+1 & \mathrm{D2} \end{bmatrix}$$

```
> L := LiftOperators(R, Alg);
```

$$L := \mathrm{table}([1 = \begin{bmatrix} 2+\mathrm{D2}\,x2 \\ x2^2\,\mathrm{D1}-x2 \end{bmatrix}])$$

```
> RightInverse(R, Alg);
```

$$\begin{bmatrix} 2+\mathrm{D2}\,x2 \\ x2^2\,\mathrm{D1}-x2 \end{bmatrix}$$

```
> simplify(Mult(R, L[1], R, Alg) - R);
```

0

Example 2:

(See Examples 4, 9, 10 in J.-F. Pommaret, A. Quadrat, *Generalized Bezout Identity*, Applicable Algebra in Engineering, Communication and Computing 9 (1998), pp. 91-116.)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[D1, -1, 0], [D2, 0, -1], [0, D2, -D1]]);
```

$$R := \begin{bmatrix} D1 & -1 & 0 \\ D2 & 0 & -1 \\ 0 & D2 & -D1 \end{bmatrix}$$

```
> L := LiftOperators(R, Alg);
```

$$L := \mathrm{table}([1 = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}, 2 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}])$$

```
> F := FreeResolution(R, Alg);
```

$$F := \mathrm{table}([1 = \begin{bmatrix} D1 & -1 & 0 \\ D2 & 0 & -1 \\ 0 & D2 & -D1 \end{bmatrix}, 2 = [-D2 \quad D1 \quad -1], 3 = \mathrm{INJ}(1)])$$

```
> RightInverse(F[2], Alg);
```

$$\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

```
> GeneralizedInverse(R, Alg);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

See Also:

DefineOreAlgebra, Mult, ApplyMatrix, LeftInverse, RightInverse, GeneralizedInverse, SyzygyModule, FreeResolution, Resolution, ProjectiveDimension, ShorterFreeResolution, ShortestFreeResolution.

OreModules[LocalLeftInverse] - **compute left inverse of a matrix over a localization of an Ore algebra**

Calling Sequence:

LocalLeftInverse(M,v,Alg)

Parameters:

`M`	-	matrix with entries in **`Alg`** or INJ(*n*) or SURJ(*n*), where *n* is a non-negative integer
`v`	-	list containing a single indeterminate
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***LocalLeftInverse*** computes (if possible) a left inverse of the matrix **`M`** over the localization of the Ore algebra **`Alg`** with respect to the multiplicatively closed set of all powers of the indeterminate given in **`v`**, i.e., ***LocalLeftInverse*** returns (if possible) a matrix *L* whose entries are fractions whose numerators are in **`Alg`** and whose denominators are powers of the indeterminate given in **`v`**, such that the product of *L* by **`M`** is the identity matrix.
- If no left inverse of **`M`** over the localization of **`Alg`** described above exists, ***LocalLeftInverse*** returns the empty list.
- **`M`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`v`** is a list containing one of the indeterminates which were used to define **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- Left (resp. right) inverses of matrices over Ore algebras are computed by LeftInverse (resp. RightInverse). Generalized inverses of matrices over Ore algebras are computed by GeneralizedInverse.

Examples:

```
> with(OreModules):
```

Example 1:

Linear differential time-delay system of a wind tunnel model (see A. Manitius, *Feedback controllers for a wind tunnel model involving a delay: analyical design and numerical simulations*, IEEE Trans. Autom. Contr. vol. 29 (1984), pp. 1058-1068):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  comm=[a,omega,zeta,k], shift_action=[delta,t,h]):
> R := evalm([[Dt+a, -k*a*delta, 0, 0], [0, Dt, -1, 0], [0, omega^2, Dt+2*zeta*omega,
  -omega^2]]);
```

$$R := \begin{bmatrix} Dt+a & -k\,a\,\delta & 0 & 0 \\ 0 & Dt & -1 & 0 \\ 0 & \omega^2 & Dt+2\,\zeta\,\omega & -\omega^2 \end{bmatrix}$$

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} Dt+a & -k\,a\,\delta & 0 & 0 \\ 0 & \omega^2 & Dt+2\,\zeta\,\omega & -\omega^2 \\ 0 & Dt & -1 & 0 \end{bmatrix}, \begin{bmatrix} -\omega^2\,k\,a\,\delta \\ -Dt\,\omega^2 - a\,\omega^2 \\ -\omega^2\,Dt^2 - \omega^2\,a\,Dt \\ -Dt\,\omega^2 - a\,\omega^2 - Dt^3 - 2\,Dt^2\,\zeta\,\omega - a\,Dt^2 - 2\,a\,Dt\,\zeta\,\omega \end{bmatrix} \right]$$

Ext1[3] provides us with a parametrization of the system. We try to compute a left inverse of Ext1[3]:

```
> LeftInverse(Ext1[3], Alg);
```

$$[\,]$$

Hence, no left inverse of Ext1[3] over **`Alg`** exists. The obstructions are given by the following possible π-polynomials:

```
> PiPolynomial(R, Alg);
```

$$[\delta, Dt+a]$$

We consider the localization of **Alg** with respect to the multiplicatively closed set of powers of δ and compute a left inverse of Ext1[3] over this localization:

```
> L := LocalLeftInverse(Ext1[3], [delta], Alg);
```

$$L := \begin{bmatrix} -\frac{1}{\delta\,\omega^2\,k\,a} & 0 & 0 & 0 \end{bmatrix}$$

```
> Mult(L, Ext1[3], Alg);
```

$$[\ 1]$$

Hence, we obtain a flat output of the system over the localized ring:

```
> evalm([[xi1(t)]])=ApplyMatrix(L, [x1(t),x2(t),x3(t),u(t)], Alg);
```

$$[\xi 1(t)] = \begin{bmatrix} -\frac{\mathrm{x1}(t+h)}{\omega^2\,k\,a} \end{bmatrix}$$

Example 2:

Linear differential time-delay system describing a satellite in a circular equatorial orbit (see T. Kailath, *Linear Systems*, Prentice-Hall, 1980, p. 60 and p. 145, and H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD Thesis, University of Orsay, France, 1995, p. 6 and p. 11):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  comm=[omega,m,r,a,b], shift_action=[delta,t]):
> R := evalm([[Dt,-1,0,0,0,0], [-3*omega^2,Dt,0,-2*omega*r,-a*delta/m,0],
  [0,0,Dt,-1,0,0], [0,2*omega/r,0,Dt,0,-b*delta/(m*r)]]);
```

$$R := \begin{bmatrix} Dt & -1 & 0 & 0 & 0 & 0 \\ -3\,\omega^2 & Dt & 0 & -2\,\omega\,r & -\frac{a\,\delta}{m} & 0 \\ 0 & 0 & Dt & -1 & 0 & 0 \\ 0 & \frac{2\,\omega}{r} & 0 & Dt & 0 & -\frac{b\,\delta}{m\,r} \end{bmatrix}$$

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -3\,m\,\omega^2 & m\,Dt & 0 & -2\,\omega\,r\,m & -a\,\delta & 0 \\ Dt & -1 & 0 & 0 & 0 & 0 \\ 0 & 2\,m\,\omega & 0 & m\,r\,Dt & 0 & -b\,\delta \\ 0 & 0 & Dt & -1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} b\,a\,\delta & 0 \\ b\,a\,\delta\,Dt & 0 \\ 0 & b\,a\,\delta \\ 0 & b\,a\,\delta\,Dt \\ -3\,b\,m\,\omega^2 + Dt^2\,b\,m & -2\,Dt\,b\,\omega\,r\,m \\ 2\,a\,Dt\,m\,\omega & a\,Dt^2\,m\,r \end{bmatrix} \right]$$

We find a parametrization of the system in Ext1[3]. We try to compute a left inverse of Ext1[3] over **Alg**:

```
> LeftInverse(Ext1[3], Alg);
```

$$[\ ]$$

Therefore, no left inverse of Ext1[3] over **Alg** exists. The obstructions are given by the following π-polynomial:

```
> PiPolynomial(R, Alg, [delta]);
```

$$[\delta]$$

We consider the localization of **Alg** with respect to the multiplicatively closed set of powers of δ and compute a left inverse of Ext1[3] over this localization:

```
> L := LocalLeftInverse(Ext1[3], [delta], Alg);
```

$$L := \begin{bmatrix} 0 & 0 & -\frac{Dt\,r\,(4\,\omega^2 + Dt^2)}{6\,\delta\,a\,\omega^3\,b} & 0 & -\frac{1}{3\,\omega^2\,b\,m} & \frac{Dt}{6\,a\,\omega^3\,m} \\ 0 & 0 & \frac{1}{\delta\,b\,a} & 0 & 0 & 0 \end{bmatrix}$$

```
> Mult(L, Ext1[3], Alg);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Hence, we obtain a flat output of the system if we introduce the time-advance operator:

```
> evalm([[xi1(t)],[xi2(t)]])=ApplyMatrix(L, [x1(t),x2(t),x3(t),x4(t),u1(t),u2(t)], Alg);
```

$$\begin{bmatrix} \xi 1(t) \\ \xi 2(t) \end{bmatrix} = \begin{bmatrix} -\frac{1}{6}\frac{r\,(D^{(3)})(x3)(t+1)\,m + 4\,r\,D(x3)(t+1)\,\omega^2\,m + 2\,u1(t)\,a\,\omega - D(u2)(t)\,b}{a\,\omega^3\,b\,m} \\ \frac{x3(t+1)}{b\,a} \end{bmatrix}$$

See Also:

DefineOreAlgebra, LeftInverse, RightInverse, GeneralizedInverse, Mult, ApplyMatrix, Involution, KroneckerProduct, Factorize, Quotient, Elimination, Integrability, ReduceMatrix.

OreModules[LQEquations],

OreModules[LQEquationsRat] - **derive Euler-Lagrange equations for a linear optimal control problem with quadratic cost functional**

Calling Sequence:

LQEquations(R,Q,Alg)
LQEquationsRat(R,Q,Alg)

Parameters:

`R` - matrix with entries in **`Alg`**
`Q` - square matrix with entries in **`Alg`**
`Alg` - Ore algebra (given by DefineOreAlgebra)

Description:

- ***LQEquations*** transforms a linear optimal control problem to a variational problem without constraints. This requires that it is possible to parametrize the solutions of the linear system of ordinary differential equations under consideration (see Parametrization). The controllability of the given linear system is a sufficient condition.
- ***LQEquations*** constructs a parametrization of the linear system and substitutes this parametrization into the cost functional, in order to obtain a variational problem without constraints. Then the Euler-Lagrange equations of the variational problem are derived.
- The result of ***LQEquations*** is a list containing three entries. The first entry is a matrix containing the integrand of the variation of the cost functional, i.e. the Euler-Lagrange equations, obtained after integrating by parts in order to eliminate the derivatives of the variations. From the second entry of the result the boundary terms which are introduced by this integration by parts can be determined. The third entry of the result gives the parametrization of the linear system which has been substituted into the cost functional.
- One way to solve the linear optimal control problem is to solve the necessary conditions given by the Euler-Lagrange equations and the boundary terms, i.e. given by the first and second entry of the result of ***LQEquations*** (see the example below; for more detailed examples see also the Library of Examples at http://wwwb.math.rwth-aachen.de/OreModules).
- The linear system of ordinary differential equations is given by the matrix **`R`**. The quadratic cost functional is defined as $\frac{1}{2}$ times the integral from 0 to T of z^{tr} **`Q`** z, where z is the vector of system variables.
- **`R`** and **`Q`** are matrices with entries in the Ore algebra **`Alg`**, where **`Q`** is expected to be a square matrix. Although in most applications **`Q`** will be symmetric, this is not required for ***LQEquations***.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- ***LQEquationsRat*** performs the same computations as ***LQEquations***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- For more information, see A. Quadrat, "Analyse algebrique des systemes lineaires multidimensionnels", PhD thesis, Ecole Nationale des Ponts et Chaussees, 1999, and J.-F. Pommaret, A. Quadrat, "A differential operator approach to multidimensional optimal control ", Int. J. Control, Vol. 77 (2004), No. 9., pp. 821-836.

Examples:

```
> with(OreModules):
```

Example:

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
```

We consider the linear system $\frac{d}{dt}x(t) = -x(t) + u(t)$.

```
> R := evalm([[D+1, -1]]);
```

$$R := [D+1 \quad -1]$$

```
> ApplyMatrix(R, [x(t),u(t)], Alg)=evalm([[0]]);
```

$$\left[x(t) + \left(\frac{d}{dt}x(t)\right) - u(t)\right] = [\ 0]$$

```
> TorsionElements(R, [x(t),u(t)], Alg);
```

$$[\ ]$$

Since there are no torsion elements, the given linear system is controllable and parametrizable.

```
> Parametrization(R, Alg);
```

$$\begin{bmatrix} -\xi_1(t) \\ -\xi_1(t) - \left(\frac{d}{dt}\xi_1(t)\right) \end{bmatrix}$$

The quadratic cost functional is defined by means of the following matrix:

```
> Q := evalm([[1,0],[0,1]]);
```

$$Q := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> L := LQEquations(R, Q, Alg);
```

$$L := \left[\left[2\,\xi_1(t) - \left(\frac{d^2}{dt^2}\xi_1(t)\right)\right], \delta_{\xi_1}(t)\left(\xi_1(t) + \left(\frac{d}{dt}\xi_1(t)\right)\right), \begin{bmatrix} -\xi_1(t) \\ -\xi_1(t) - \left(\frac{d}{dt}\xi_1(t)\right) \end{bmatrix}\right]$$

The left hand side of the Euler-Lagrange equation is:

```
> E := L[1][1,1];
```

$$E := 2\,\xi_1(t) - \left(\frac{d^2}{dt^2}\xi_1(t)\right)$$

```
> sol := dsolve(E, xi[1](t));
```

$$sol := \xi_1(t) = \_C1\,\mathbf{e}^{(\sqrt{2}\,t)} + \_C2\,\mathbf{e}^{(-\sqrt{2}\,t)}$$

```
> sol := rhs(sol);
```

$$sol := \_C1\,\mathbf{e}^{(\sqrt{2}\,t)} + \_C2\,\mathbf{e}^{(-\sqrt{2}\,t)}$$

The boundary terms which were introduced by integration by parts of the variation of the cost function are:

```
> B := L[2];
```

$$B := \delta_{\xi_1}(t)\left(\xi_1(t) + \left(\frac{d}{dt}\xi_1(t)\right)\right)$$

The constants $\_C1$ and $\_C2$ of the general solution sol to the Euler-Lagrange equation are determined from the initial condition $\xi_1(0) = -x(0) = -x0$ and the final condition $\xi_1(T) + D(\xi_1)(T) = 0$ given by B:

```
> FinalConditions(B, T);
```

$$[\xi_1(T) + D(\xi_1)(T)]$$

```
> solve({subs(t=0, sol)+x0=0, subs(t=T, sol+diff(sol, t))=0}, {_C1,_C2});
```

$$\left\{\_C2 = -\frac{\mathbf{e}^{(\sqrt{2}\,T)}(\sqrt{2}+1)\,x0}{(\mathbf{e}^{(\sqrt{2}\,T)} + \sqrt{2}\,\mathbf{e}^{(\sqrt{2}\,T)} - \mathbf{e}^{(-\sqrt{2}\,T)} + \sqrt{2}\,\mathbf{e}^{(-\sqrt{2}\,T)})\,\mathbf{e}^0}, \_C1 = -\frac{x0\,\mathbf{e}^{(-\sqrt{2}\,T)}(\sqrt{2}-1)}{(\mathbf{e}^{(\sqrt{2}\,T)} + \sqrt{2}\,\mathbf{e}^{(\sqrt{2}\,T)} - \mathbf{e}^{(-\sqrt{2}\,T)} + \sqrt{2}\,\mathbf{e}^{(-\sqrt{2}\,T)})\,\mathbf{e}^0}\right\}$$

According to the parametrization used by LQEquations, which is given in L[3], we have:

```
> x = simplify(-subs(%, sol));
```

$$x = \frac{x0\,(-\mathbf{e}^{(-\sqrt{2}\,(T-t))} + \mathbf{e}^{(-\sqrt{2}\,(T-t))}\sqrt{2} + \mathbf{e}^{(\sqrt{2}\,(T-t))} + \mathbf{e}^{(\sqrt{2}\,(T-t))}\sqrt{2})}{\mathbf{e}^{(\sqrt{2}\,T)} + \sqrt{2}\,\mathbf{e}^{(\sqrt{2}\,T)} - \mathbf{e}^{(-\sqrt{2}\,T)} + \sqrt{2}\,\mathbf{e}^{(-\sqrt{2}\,T)}}$$

```
> u = simplify(subs(%%, -sol-diff(sol, t)));
```

$$u = \frac{x0\,(\mathbf{e}^{(-\sqrt{2}\,(T-t))} - \mathbf{e}^{(\sqrt{2}\,(T-t))})}{\mathbf{e}^{(\sqrt{2}\,T)} + \sqrt{2}\,\mathbf{e}^{(\sqrt{2}\,T)} - \mathbf{e}^{(-\sqrt{2}\,T)} + \sqrt{2}\,\mathbf{e}^{(-\sqrt{2}\,T)}}$$

See Also:

DefineOreAlgebra, FinalConditions, BoundaryTerms, Mult, ApplyMatrix, Involution, ControllabilityMatrix, Brunovsky, TorsionElements, KalmanSystem.

OreModules[MinimalParametrization], OreModules[MinimalParametrizationRat], OreModules[MinimalParametrizations], OreModules[MinimalParametrizationsRat] - return minimal parametrization(s) of a linear system over an Ore algebra

Calling Sequence:

MinimalParametrization(R,Alg)
MinimalParametrizationRat(R,Alg)
MinimalParametrizations(R,Alg)
MinimalParametrizationsRat(R,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***MinimalParametrization*** constructs a minimal parametrization of the torsion-free left module over **`Alg`** which is presented by **`R`**, namely a matrix *Q* with entries in **`Alg`** such that the rows of **`R`** generate all left **`Alg`**-relations (i.e. syzygies, see SyzygyModule) of the rows of *Q*, and among all matrices with entries in **`Alg`** satisfying this property the result of ***MinimalParametrization*** has the least number of columns. In particular, the product of **`R`** by *Q* is the zero matrix. The minimality of *Q* means that the left **`Alg`**-module presented by *Q* is either the zero module or a torsion left **`Alg`**-module.

- ***MinimalParametrizations*** returns a list of several minimal parametrizations of the left **`Alg`**-module presented by **`R`**.

- The left module which is considered by ***MinimalParametrization(s)***, namely the module presented by **`R`**, is the factor module of the free module of row vectors with entries in **`Alg`** whose length equals the number of columns of **`R`** modulo the submodule which is generated by the rows of **`R`**.

- In the terminology of linear systems over Ore algebras, ***MinimalParametrization*** constructs a matrix which represents an operator *P* such that all solutions of the parametrizable linear system **`R`** $y = 0$ are obtained as $y =$ **`P`** z for some vector of functions z.

- Note that ***MinimalParametrization*** does not check whether the left **`Alg`**-module presented by **`R`** is torsion-free, i.e. whether the linear system **`R`** $y = 0$ is parametrizable. If it is not, then the result will be a minimal parametrization of the left **`Alg`**-module presented by **`R`** modulo its torsion submodule, i.e. a minimal parametrization of the linear system obtained from **`R`** $y = 0$ by adding a suitable set of equations such that all autonomous elements of the system are set to zero. (See also Example 3 below).

- First ***MinimalParametrization*** computes the syzygy module of the left **`Alg`**-module presented by the formal adjoint of **`R`** (i.e. SyzygyModule is applied to the result of Involution applied to **`R`**). Then the rank *r* of the left **`Alg`**-module presented by the syzygies is determined using OreRank. The rank *r* gives the number of columns of the resulting matrix *Q*. Then Involution is applied to the first matrix, found by selecting *r* rows of the matrix of syzygies computed before, whose rows do not satisfy any non-trivial left **`Alg`**-linear relation, and the result is returned.

- ***MinimalParametrizations*** returns the list of formal adjoints of all matrices, found by selecting *r* rows of the matrix of syzygyies (see previous point), whose rows do not satisfy any non-trivial left **`Alg`**-linear relation, i.e. which are left **`Alg`**-linearly independent.

- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.

- **`Alg`** is expected to be defined using DefineOreAlgebra.

- The result of ***MinimalParametrization*** is a matrix with entries in **`Alg`** whose number of rows equals the number of columns of **`R`**. The result of ***MinimalParametrizations*** is a list of matrices with entries in **`Alg`** whose numbers of rows equal the number of columns of **`R`**.

- ***MinimalParametrization(s)Rat*** performs the same computations as ***MinimalParametrization(s)***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.

- For more details about minimal parametrizations, see F. Chyzak, A. Quadrat, D. Robertz, "Effective algorithms for parametrizing linear control systems over Ore algebras", Applicable Algebra in Engineering, Communication and Computing (AAECC) 16 (2005), pp. 319-376.

Examples:

```
> with(OreModules):
```

Example 1:

We compute a minimal parametrization for the divergence operator:

```
> Alg := DefineOreAlgebra(diff=[d[1],x[1]], diff=[d[2],x[2]], diff=[d[3],x[3]],
  polynom=[x[1],x[2],x[3]]):
> R := evalm([[d[1], d[2], d[3]]]);
```

$$R := [d_1 \quad d_2 \quad d_3]$$

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[[\;1], [d_1 \quad d_2 \quad d_3], \begin{bmatrix} d_3 & d_2 & 0 \\ 0 & -d_1 & d_3 \\ -d_1 & 0 & -d_2 \end{bmatrix} \right]$$

We find that the linear system given by the divergence operator is parametrizable, and *Ext1*[3] is a parametrization of the system:

```
> Mult(R, Ext1[3], Alg);
```

$$[0 \quad 0 \quad 0]$$

In fact, up to the sign and a permutation of the columns, *Ext1*[3] is the curl operator. However, this parametrization is not minimal. We have that the rank of the **`Alg`**-module which is associated with the system is 2, i.e., there exists a parametrization of the system which depends on two arbitrary functions only:

```
> OreRank(R, Alg);
```

$$2$$

```
> P := MinimalParametrization(R, Alg);
```

$$P := \begin{bmatrix} d_3 & d_2 \\ 0 & -d_1 \\ -d_1 & 0 \end{bmatrix}$$

```
> Mult(R, P, Alg);
```

$$[\,0 \quad 0]$$

```
> MinimalParametrizations(R, Alg);
```

$$\left[\begin{bmatrix} d_3 & d_2 \\ 0 & -d_1 \\ -d_1 & 0 \end{bmatrix}, \begin{bmatrix} d_3 & 0 \\ 0 & d_3 \\ -d_1 & -d_2 \end{bmatrix}, \begin{bmatrix} d_2 & 0 \\ -d_1 & d_3 \\ 0 & -d_2 \end{bmatrix} \right]$$

Example 2:

We consider the first set of Maxwell equations (for more details, see the Library of Examples at http://wwwb.math.rwth-aachen.de/OreModules):

```
> Alg := DefineOreAlgebra(diff=[d[1],x[1]], diff=[d[2],x[2]], diff=[d[3],x[3]],
  diff=[d[4],x[4]], polynom=[x[1],x[2],x[3],x[4]]):
> R := evalm([[d[4], 0, 0, 0, -d[3], d[2]],
              [0, d[4], 0, d[3], 0, -d[1]],
              [0, 0, d[4],-d[2], d[1], 0],
              [d[1], d[2], d[3], 0, 0, 0]]);
```

$$R := \begin{bmatrix} d_4 & 0 & 0 & 0 & -d_3 & d_2 \\ 0 & d_4 & 0 & d_3 & 0 & -d_1 \\ 0 & 0 & d_4 & -d_2 & d_1 & 0 \\ d_1 & d_2 & d_3 & 0 & 0 & 0 \end{bmatrix}$$

In terms of equations, the first set of Maxwell equations is given by:

```
> ApplyMatrix(R, [seq(B[i](seq(x[j],j=1..4)),i=1..3),seq(E[i](seq(x[j],j=1..4)),i=1..3)],
  Alg)=evalm([[0]$4]);
```

$$\begin{bmatrix} \left(\frac{\partial}{\partial x_4} B_1(x_1,x_2,x_3,x_4)\right) - \left(\frac{\partial}{\partial x_3} E_2(x_1,x_2,x_3,x_4)\right) + \left(\frac{\partial}{\partial x_2} E_3(x_1,x_2,x_3,x_4)\right) \\ \left(\frac{\partial}{\partial x_4} B_2(x_1,x_2,x_3,x_4)\right) + \left(\frac{\partial}{\partial x_3} E_1(x_1,x_2,x_3,x_4)\right) - \left(\frac{\partial}{\partial x_1} E_3(x_1,x_2,x_3,x_4)\right) \\ \left(\frac{\partial}{\partial x_4} B_3(x_1,x_2,x_3,x_4)\right) - \left(\frac{\partial}{\partial x_2} E_1(x_1,x_2,x_3,x_4)\right) + \left(\frac{\partial}{\partial x_1} E_2(x_1,x_2,x_3,x_4)\right) \\ \left(\frac{\partial}{\partial x_1} B_1(x_1,x_2,x_3,x_4)\right) + \left(\frac{\partial}{\partial x_2} B_2(x_1,x_2,x_3,x_4)\right) + \left(\frac{\partial}{\partial x_3} B_3(x_1,x_2,x_3,x_4)\right) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Let us check whether or not the first set of Maxwell equations is parametrizable. In order to do that, let us introduce the formal adjoint *R_adj* of *R*:

```
> R_adj := Involution(R, Alg);
```

$$R\_adj := \begin{bmatrix} -d_4 & 0 & 0 & -d_1 \\ 0 & -d_4 & 0 & -d_2 \\ 0 & 0 & -d_4 & -d_3 \\ 0 & -d_3 & d_2 & 0 \\ d_3 & 0 & -d_1 & 0 \\ -d_2 & d_1 & 0 & 0 \end{bmatrix}$$

To check whether the system of Maxwell equations is parametrizable, we compute the first extension module with values in *Alg* of the left *Alg*-module *N* which is associated with *R_adj*:

```
> Ext1 := Exti(R_adj, Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} d_4 & 0 & 0 & 0 & -d_3 & d_2 \\ d_1 & d_2 & d_3 & 0 & 0 & 0 \\ 0 & -d_4 & 0 & -d_3 & 0 & d_1 \\ 0 & 0 & d_4 & -d_2 & d_1 & 0 \end{bmatrix}, \begin{bmatrix} d_3 & d_2 & 0 & 0 \\ 0 & -d_1 & d_3 & 0 \\ -d_1 & 0 & -d_2 & 0 \\ 0 & 0 & -d_4 & -d_1 \\ d_4 & 0 & 0 & -d_2 \\ 0 & -d_4 & 0 & -d_3 \end{bmatrix} \right]$$

Since *Ext1*[1] is the identity matrix, we see that the module *M*, which is associated with *R*, is torsion-free. Equivalently, the system of Maxwell equations is parametrizable and *Ext1*[3] is a parametrization of the system. In what follows, we shall see that this parametrization is not minimal. We first compute a free resolution of the *Alg*-module *M* associated with *R*:

```
> FreeResolution(R, Alg);
```

$$\text{table}([1 = \begin{bmatrix} d_4 & 0 & 0 & 0 & -d_3 & d_2 \\ 0 & d_4 & 0 & d_3 & 0 & -d_1 \\ 0 & 0 & d_4 & -d_2 & d_1 & 0 \\ d_1 & d_2 & d_3 & 0 & 0 & 0 \end{bmatrix}, 2 = [d_1 \quad d_2 \quad d_3 \quad -d_4], 3 = \text{INJ}(1)])$$

In particular, by summing alternatingly the number of columns of all the entries in this free resolution, we find that the rank of M is 6 - 4 + 1 = 3. This result can also be obtained using *OreRank*:

```
> OreRank(R, Alg);
```

$$3$$

Hence, a minimal parametrization of the system involves only three potentials. Let us compute some minimal parametrizations of the system:

```
> Pmin := MinimalParametrizations(R, Alg);
```

$$Pmin := \left[\begin{bmatrix} d_3 & d_2 & 0 \\ 0 & -d_1 & d_3 \\ -d_1 & 0 & -d_2 \\ 0 & 0 & -d_4 \\ d_4 & 0 & 0 \\ 0 & -d_4 & 0 \end{bmatrix}, \begin{bmatrix} d_3 & d_2 & 0 \\ 0 & -d_1 & 0 \\ -d_1 & 0 & 0 \\ 0 & 0 & -d_1 \\ d_4 & 0 & -d_2 \\ 0 & -d_4 & -d_3 \end{bmatrix}, \begin{bmatrix} d_3 & 0 & 0 \\ 0 & d_3 & 0 \\ -d_1 & -d_2 & 0 \\ 0 & -d_4 & -d_1 \\ d_4 & 0 & -d_2 \\ 0 & 0 & -d_3 \end{bmatrix}, \begin{bmatrix} d_2 & 0 & 0 \\ -d_1 & d_3 & 0 \\ 0 & -d_2 & 0 \\ 0 & -d_4 & -d_1 \\ 0 & 0 & -d_2 \\ -d_4 & 0 & -d_3 \end{bmatrix}\right]$$

Example 3:

If the left module which is associated with the linear system is a finite dimensional vector space (e.g. if the solution space of an analytic linear system of PDEs is finite-dimensional over the field of constants), then this module is a torsion module:

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
> R := evalm([[D,0],[0,D]]);
```

$$R := \begin{bmatrix} D & 0 \\ 0 & D \end{bmatrix}$$

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} D & 0 \\ 0 & D \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{SURJ}(2)\right]$$

```
> MinimalParametrization(R, Alg);
```

$$\text{SURJ}(2)$$

Here we use Parametrization to obtain a parametrization of this finite-dimensional solution space:

```
> Parametrization(R, Alg);
```

$$\begin{bmatrix} \_C1 \\ \_C2 \end{bmatrix}$$

See Also:

DefineOreAlgebra, Parametrization, IntTorsion, ParticularSolution, Complement, Involution, OreRank, SyzygyModule, Resolution, FreeResolution, Exti, Extn, Torsion, AutonomousElements, PiPolynomial, TorsionElements.

OreModules[Mult] - multiply scalars or matrices over an Ore algebra

Calling Sequence:

Mult(M1,M2,...,Alg)

Parameters:

`M1, M2, ...`	-	scalars in **`Alg`** or matrices with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***Mult*** returns the product **`M1 M2`** ..., where **`M1`**, **`M2`**, ... are scalars in **`Alg`** or matrices with entries in **`Alg`** (expecting that their product in the given order is defined).
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***Mult*** is a scalar in **`Alg`** if all arguments (except **`Alg`**) are scalars. It is a matrix with entries in **`Alg`** if at least one matrix occurs in the arguments of ***Mult***.
- This command extends skew_product in Ore_algebra.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D[1],x[1]], diff=[D[2],x[2]], polynom=[x[1],x[2]]):
```

Multiplying scalars:

```
> Mult(D[1], x[1], Alg);
```

$$1 + D_1\, x_1$$

```
> Mult(3, D[2], x[1]+1, Alg);
```

$$3\, D_2\, (x_1 + 1)$$

Multiplying matrices:

```
> L1 := evalm([[x[2]*D[1]+1, D[2]], [0, D[1]]]);
```

$$L1 := \begin{bmatrix} x_2\, D_1 + 1 & D_2 \\ 0 & D_1 \end{bmatrix}$$

```
> R1 := evalm([[2+x[2]*D[2], x[1]], [x[2]^2*D[1]-x[2], x[1]+x[2]]]);
```

$$R1 := \begin{bmatrix} 2 + x_2\, D_2 & x_1 \\ x_2^{\,2}\, D_1 - x_2 & x_1 + x_2 \end{bmatrix}$$

```
> Mult(L1, R1, Alg);
```

$$\begin{bmatrix} 4\, x_2\, D_1 + 2\, x_2^{\,2}\, D_1\, D_2 + 1 & x_1\, x_2\, D_1 + x_1 + x_2 + 1 + D_2\, x_1 + x_2\, D_2 \\ D_1\, x_2\, (x_2\, D_1 - 1) & 1 + D_1\, x_1 + x_2\, D_1 \end{bmatrix}$$

Multiplying scalar and matrices:

```
> Mult(7, L1, R1, Alg);
```

$$\begin{bmatrix} 28\, x_2\, D_1 + 14\, x_2^{\,2}\, D_1\, D_2 + 7 & 7\, x_1\, x_2\, D_1 + 7\, x_1 + 7\, x_2 + 7 + 7\, D_2\, x_1 + 7\, x_2\, D_2 \\ 7\, D_1\, x_2\, (x_2\, D_1 - 1) & 7 + 7\, D_1\, x_1 + 7\, x_2\, D_1 \end{bmatrix}$$

See Also:

DefineOreAlgebra, Ore_algebra[skew_product], ApplyMatrix, Involution, KroneckerProduct, ReduceMatrix, LeftInverse, RightInverse, GeneralizedInverse.

OreModules[OreRank],

OreModules[OreRankRat] - compute the rank of a finitely presented left module over an Ore algebra

Calling Sequence:

OreRank(R,Alg)
OreRankRat(R,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***OreRank*** returns the rank of the left module over the Ore algebra **`Alg`** which is presented by **`R`**, namely the vector space dimension of the tensor product of this module by the quotient (skew) field of **`Alg`**.
- The left module which is considered by ***OreRank***, namely the left module presented by **`R`**, is the factor module of the free module of row vectors over **`Alg`** whose length equals the number of columns of **`R`** modulo the submodule which is generated by the rows of **`R`**.
- The rank of the left module M presented by **`R`** is computed by constructing a finite free resolution of M and summing up alternatingly the ranks of the free modules in this resolution. The resulting number, which is the Euler characteristic of M, equals the rank of M and is returned by ***OreRank***.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- ***OreRankRat*** performs the same computations as ***OreRank***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R1 := matrix([[1, D2+1], [D1, D2]]);
```

$$R1 := \begin{bmatrix} 1 & D2+1 \\ D1 & D2 \end{bmatrix}$$

```
> OreRank(R1, Alg);
```

$$0$$

The rank of the left module presented by $R1$ is zero. In fact, this module is a torsion module, as one obtains a generating set of torsion elements for the module by computing the first extension module with values in **`Alg`** of the adjoint module:

```
> Exti(Involution(R1, Alg), Alg, 1);
```

$$\left[\begin{bmatrix} -D2+D1\,D2+D1 & 0 \\ 0 & -D2+D1\,D2+D1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathrm{SURJ}(2)\right]$$

Example 2:

```
> R2 := matrix([[1, D2], [D1, D1*D2]]);
```

$$R2 := \begin{bmatrix} 1 & D2 \\ D1 & D1\,D2 \end{bmatrix}$$

```
> OreRank(R2, Alg);
```

$$1$$

```
> R3 := matrix([[1, D2, 0], [D1, D1*D2, 0]]);
```

$$R3 := \begin{bmatrix} 1 & D2 & 0 \\ D1 & D1\,D2 & 0 \end{bmatrix}$$

```
> OreRank(R3, Alg);
```

$$2$$

See Also:

DefineOreAlgebra, KBasis, Connection, HilbertSeries, Dimension, Involution, SyzygyModule, Resolution, FreeResolution, Exti, Extn, Torsion, Elimination, Factorize.

OreModules[Parametrization],

OreModules[ParametrizationRat] - **return, if possible, a parametrization of a linear system over an Ore algebra**

Calling Sequence:

Parametrization(R,Alg)
ParametrizationRat(R,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***Parametrization*** constructs, if possible, a parametrization of the linear system represented by the matrix **R**, i.e. in particular, it constructs a vector *z* which depends on some arbitrary functions and possibly on some arbitrary constants such that **R** $y = 0$ holds identically. Of course, the problem of parametrizing all solutions of **R** $y = 0$ in this way depends on the space of functions under consideration, i.e. the space where the entries of *y* are searched for. For several types of linear systems and spaces of functions, ***Parametrization*** yields a parametrization of the solutions of the linear system in the sense that *all* solutions of **R** $y = 0$ are found by substituting appropriate functions into the parameters in *z*. For instance, if **R** $y = 0$ is a linear system of partial differential equations with constant coefficients and the function space is chosen to be the set of all smooth functions, then all solutions of **R** $y = 0$ are parametrized by the result *z* of ***Parametrization***.
- If the left **`Alg`**-module *M* associated with the linear system is torsion-free, i.e. the factor module of the free left **`Alg`**-module of row vectors whose length equals the number of columns of **R** modulo the submodule which is generated by the rows of **R** contains no non-zero torsion elements, then ***Parametrization*** applies the parametrization obtained by Exti to a vector of arbitrary functions ξ_i and returns the resulting vector of functions.
- If the left **`Alg`**-module *M* associated with the linear system is not torsion-free, i.e. the linear system under consideration has some non-trivial autonomous elements, then ***Parametrization*** tries to integrate the torsion elements using IntTorsion and tries to glue these integrated torsion elements with the parametrization (computed by Exti) of the linear system obtained from the given one by equating all autonomous elements to zero. In the module-theoretic language, this can be achieved if the torsion submodule of *M* has a complement in *M* (see Complement). The latter condition is always satisfied for linear systems of ordinary differential equations. Up to now, ***Parametrization*** only uses ParticularSolution to glue the integrated torsion elements with the parametrization of the linear system without non-trivial autonomous elements (see also ParticularSolution).
- **R** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- Depending on the structural properties of the linear system defined by **R** and the possibilities to integrate the resulting equations, ***Parametrization*** returns either a matrix or a table of (equations of) matrices.
- If the given linear system is parametrizable by means of arbitrary functions (and/or constants) only, or if present torsion elements can be integrated and glued with the parametrization of the corresponding linear system without non-trivial autonomous elements (see above), then the result of ***Parametrization*** is a matrix *P* whose entries are linear expressions in arbitrary functions ξ_i of the independent variables and in constants *_Cj* such that **R** $P = 0$, where multiplication is the action of the operators in **`Alg`** on functions. For the issue of parametrizing *all* solutions in this way see the first paragraph.
- If the given linear system has autonomous elements and it is not possible to integrate them, ***Parametrization*** returns a table with three entries. The first entry gives a parametrization of the linear system obtained from the given one by equating all autonomous elements to zero. The second entry is the equation $R2\,\eta = \zeta$ and the third entry is the equation $R1\,\zeta = 0$ (for the connection of these equations with the glueing of parametrizations see ParticularSolution).
- If the given linear system has autonomous elements and it is possible to solve $R1\,\tau = 0$ (i.e. to integrate the torsion elements; for the

notation see ParticularSolution), but no particular solution to the linear system $R2\,\eta = \tau$ can be found, then ***Parametrization*** returns a table with two entries. The first entry gives a parametrization of the linear system obtained from the given one by equating all autonomous elements to zero. The second entry of the result is the equation $R2\,\eta = \tau$.

- ***ParametrizationRat*** performs the same computations as ***Parametrization***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- For more details see A. Quadrat, D. Robertz, "Parametrizing all solutions of uncontrollable multidimensional linear systems", Proceedings of the 16th IFAC World Congress, Prague, 2005.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
> R := evalm([[D]]);
```

$$R := [\mathrm{D}]$$

```
> ApplyMatrix(R, [x(t)], Alg);
```

$$\left[\frac{d}{dt}\mathrm{x}(t)\right]$$

```
> Parametrization(evalm([[D]]), Alg);
```

$$[\_C1]$$

Every solution of $\frac{d}{dt}\mathrm{x}(t) = 0$ is a constant.

Example 2: Poincare sequence

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R1 := evalm([[D1, D2, D3]]);
```

$$R1 := [\mathrm{D1}\quad \mathrm{D2}\quad \mathrm{D3}]$$

```
> P1 := Parametrization(R1, Alg);
```

$$P1 := \begin{bmatrix} \left(\frac{\partial}{\partial x3}\xi_1(x1,x2,x3)\right) + \left(\frac{\partial}{\partial x2}\xi_2(x1,x2,x3)\right) \\ -\left(\frac{\partial}{\partial x1}\xi_2(x1,x2,x3)\right) + \left(\frac{\partial}{\partial x3}\xi_3(x1,x2,x3)\right) \\ -\left(\frac{\partial}{\partial x1}\xi_1(x1,x2,x3)\right) - \left(\frac{\partial}{\partial x2}\xi_3(x1,x2,x3)\right) \end{bmatrix}$$

The solutions of the divergence operator are parametrized (up to signs and permutation of columns) by the curl operator.

```
> R2 := DiffToOre(P1, [xi[1],xi[2],xi[3]], Alg)[1];
```

$$R2 := \begin{bmatrix} \mathrm{D3} & \mathrm{D2} & 0 \\ 0 & -\mathrm{D1} & \mathrm{D3} \\ -\mathrm{D1} & 0 & -\mathrm{D2} \end{bmatrix}$$

```
> P2 := Parametrization(R2, Alg);
```

$$P2 := \begin{bmatrix} \frac{\partial}{\partial x2}\xi_1(x1,x2,x3) \\ -\left(\frac{\partial}{\partial x3}\xi_1(x1,x2,x3)\right) \\ -\left(\frac{\partial}{\partial x1}\xi_1(x1,x2,x3)\right) \end{bmatrix}$$

The solutions of the curl operator are parametrized (up to signs) by the gradient operator:

```
> R3 := DiffToOre(P2, [xi[1]], Alg)[1];
```

$$R3 := \begin{bmatrix} D2 \\ -D3 \\ -D1 \end{bmatrix}$$

```
> Mult(R1, R2, Alg);
```

$$[0 \quad 0 \quad 0]$$

```
> Mult(R2, R3, Alg);
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Example 3: Differential time-delay system

Linear differential time-delay system describing a flexible rod (see H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD thesis, University of Orsay, France, 1995):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  shift_action=[delta,t,h]):
> R := evalm([[Dt, -Dt*delta, -1], [2*Dt*delta, -Dt-Dt*delta^2, 0]]);
```

$$R := \begin{bmatrix} Dt & -Dt\,\delta & -1 \\ 2\,Dt\,\delta & -Dt - Dt\,\delta^2 & 0 \end{bmatrix}$$

```
> ApplyMatrix(R, [y1(t),y2(t),u(t)], Alg);
```

$$\begin{bmatrix} \mathrm{D}(y1)(t) - \mathrm{D}(y2)(t-h) - \mathrm{u}(t) \\ 2\,\mathrm{D}(y1)(t-h) - \mathrm{D}(y2)(t) - \mathrm{D}(y2)(t-2\,h) \end{bmatrix}$$

```
> P := Parametrization(R, Alg);
```

$$P := \begin{bmatrix} \frac{\_C1}{2} + \xi_1(t) + \xi_1(t-2\,h) \\ \_C1 + 2\,\xi_1(t-h) \\ -\mathrm{D}(\xi_1)(t-2\,h) + \mathrm{D}(\xi_1)(t) \end{bmatrix}$$

We find that P is a solution of $\mathbf{R}\,y = 0$ for all smooth functions ξ_1:

```
> ApplyMatrix(R, P, Alg);
```

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Example 4: Partial differential equations

Linear system of PDEs that appears in mathematical physics, namely in the study of Lie-Poisson structures (see C. M. Bender, G. V. Dunne, L. R. Mead, *Underdetermined systems of partial differential equations*, Journal of Mathematical Physics, vol. 41, no. 9 (2000), pp. 6388-6398 and W. M. Seiler, *Involution analysis of the partial differential equations characterising Hamiltonian vector fields*, Journal of Mathematical Physics, vol. 44 (2003), pp. 1173-1182):

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[x1*D3, x2*D3, 0], [-x1*D2+x2*D1, -1, x2*D3], [-1, -x2*D1+x1*D2, x1*D3]]);
```

$$R := \begin{bmatrix} x1\,\mathrm{D3} & x2\,\mathrm{D3} & 0 \\ -x1\,\mathrm{D2} + x2\,\mathrm{D1} & -1 & x2\,\mathrm{D3} \\ -1 & -x2\,\mathrm{D1} + x1\,\mathrm{D2} & x1\,\mathrm{D3} \end{bmatrix}$$

In this example, no particular solution of $R2\,\eta = \tau$ is found to glue the integrated torsion elements with the parametrization of the linear system obtained from the given linear system by equating all autonomous elements to zero (for the notation see ParticularSolution):

```
> Parametrization(R, Alg);
```

$$\text{table}([1 = \begin{bmatrix} \eta_1(x1,x2,x3) - x2\left(\frac{\partial}{\partial x3}\xi_1(x1,x2,x3)\right) \\ \eta_2(x1,x2,x3) + x1\left(\frac{\partial}{\partial x3}\xi_1(x1,x2,x3)\right) \\ \eta_3(x1,x2,x3) + x2\left(\frac{\partial}{\partial x1}\xi_1(x1,x2,x3)\right) - x1\left(\frac{\partial}{\partial x2}\xi_1(x1,x2,x3)\right) \end{bmatrix},$$

$$2 = \left(\begin{bmatrix} x1\,\eta_1(x1,x2,x3) + x2\,\eta_2(x1,x2,x3) \\ \left(\frac{\partial}{\partial x1}\eta_1(x1,x2,x3)\right) + \left(\frac{\partial}{\partial x2}\eta_2(x1,x2,x3)\right) + \left(\frac{\partial}{\partial x3}\eta_3(x1,x2,x3)\right) \\ -\eta_1(x1,x2,x3) - x2\left(\frac{\partial}{\partial x1}\eta_2(x1,x2,x3)\right) + x1\left(\frac{\partial}{\partial x2}\eta_2(x1,x2,x3)\right) + x1\left(\frac{\partial}{\partial x3}\eta_3(x1,x2,x3)\right) \end{bmatrix} =$$

$$\begin{bmatrix} \int x1\,\_F1(x1^2+x2^2)\,dx1 + \int -x2\left(2\int D(\_F1)(x1^2+x2^2)\,x1\,dx1 - \_F1(x1^2+x2^2)\right)dx2 + \_C1 \\ \_F1(x1^2+x2^2) \\ 0 \end{bmatrix}\Bigg)$$

$$])$$

The first entry of this table is a parametrization of the linear system obtained from the given linear system by equating all autonomous elements to zero. The second entry is the equation $R2\,\eta = \tau$, where τ is the general solution of the homogeneous linear system $R1\,\tau = 0$ (see ParticularSolution).

```
> ParametrizationRat(R, Alg);
```

$$\text{table}([1 = \begin{bmatrix} \eta_1(x1,x2,x3) + x2\left(\frac{\partial}{\partial x3}\xi_1(x1,x2,x3)\right) \\ \eta_2(x1,x2,x3) - x1\left(\frac{\partial}{\partial x3}\xi_1(x1,x2,x3)\right) \\ \eta_3(x1,x2,x3) - x2\left(\frac{\partial}{\partial x1}\xi_1(x1,x2,x3)\right) + x1\left(\frac{\partial}{\partial x2}\xi_1(x1,x2,x3)\right) \end{bmatrix},$$

$$2 = \left(\begin{bmatrix} x1\,\eta_1(x1,x2,x3) + x2\,\eta_2(x1,x2,x3) \\ -x2\,\eta_2(x1,x2,x3) + x2\,x1\left(\frac{\partial}{\partial x1}\eta_2(x1,x2,x3)\right) - x1^2\left(\frac{\partial}{\partial x2}\eta_2(x1,x2,x3)\right) - x1^2\left(\frac{\partial}{\partial x3}\eta_3(x1,x2,x3)\right) \end{bmatrix} = \begin{bmatrix} -\_F1(x1^2+x2^2) \\ \_F1(x1^2+x2^2) \end{bmatrix}\right)$$

$$])$$

See Also:

DefineOreAlgebra, IntTorsion, ParticularSolution, Complement, MinimalParametrization, DiffToOre, SyzygyModule, Resolution, FreeResolution, ShorterFreeResolution, ShortestFreeResolution, ProjectiveDimension, Exti, Extn, Torsion, AutonomousElements, PiPolynomial, TorsionElements.

OreModules[ParticularSolution],

OreModules[ParticularSolutionRat] - **for parametrizing a linear system, find a particular solution after integration of the torsion elements**

Calling Sequence:

ParticularSolution(R,Alg)
ParticularSolutionRat(R,Alg)

Parameters:

R	-	matrix with entries in **Alg**
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- In order to find a parametrization of a linear system **R** $y = 0$ of partial differential equations having autonomous elements, **R** can be split as a product *R1 R2* such that the system **R** $y = 0$ is equivalent to $R1\,\tau = 0$ and $\tau = R2\,\eta$, namely *R2* is a presentation matrix of the left **Alg** -module *M* which is associated with **R** $y = 0$ modulo its torsion submodule. *R2* can be obtained as the second entry of the result of applying Exti for $i = 1$ to the formal adjoint (see Involution) of **R**, and *R1* can be computed by applying Factorize to **R** and *R2*.
- ***ParticularSolution*** first calls IntTorsion to obtain the general solution τ of the homogeneous linear system $R1\,\tau = 0$ and then tries to find a particular solution η of the inhomogeneous linear system $R2\,\eta = \tau$.
- A particular solution η is obtained by applying a generalized inverse of *R2* (see GeneralizedInverse, but see also Complement), if it exists, to the vector of integrated torsion elements computed by IntTorsion.
- **R** is a matrix with entries in the Ore algebra **Alg**.
- **Alg** is expected to be defined using DefineOreAlgebra.
- The result of ***ParticularSolution*** is a list with three entries.
- The first entry of the result is a matrix with entries in **Alg** having the same number of columns as **R**. The residue classes of the rows of this matrix in the module associated with the given system generate the torsion submodule (see also TorsionElements). This matrix equals the second entry of the result of applying Exti for i = 1 to the formal adjoint of **R**.
- The second entry of the result of ***ParticularSolution*** is a particular solution eta of $R2\,\eta = \tau$, if a generalized inverse of *R2* (see GeneralizedInverse) exists. Otherwise this entry is the empty list.
- The third entry of the result is the vector τ, if IntTorsion succeeded to integrate the torsion elements. Otherwise this entry is the empty list.
- The general solution of the homogeneous linear system $R1\,\tau = 0$ can be computed using IntTorsion. The commands IntTorsion and ***ParticularSolution*** are used by Parametrization, if the system has autonomous elements.
- ***ParticularSolutionRat*** performs the same computations as ***ParticularSolution***, but the domain of coefficients of the Ore algebra **Alg** is replaced by its quotient field, i.e. rational functions.
- For more details see A. Quadrat, D. Robertz, "Parametrizing all solutions of uncontrollable multidimensional linear systems", Proceedings of the 16th IFAC World Congress, Prague, 2005.

Examples:

```
> with(OreModules):
```

Example 1: Ordinary differential equations

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l]):
> R := evalm([[D^2+g/l, 0, -g/l], [0, D^2+g/l, -g/l]]);
```

$$R := \begin{bmatrix} D^2+\frac{g}{l} & 0 & -\frac{g}{l} \\ 0 & D^2+\frac{g}{l} & -\frac{g}{l} \end{bmatrix}$$

```
> P := ParticularSolution(R, Alg);
```

$$P := \left[\begin{bmatrix} 1 & -1 & 0 \\ 0 & D^2\, l+g & -g \end{bmatrix}, \begin{bmatrix} \_C1 \sin\left(\frac{\sqrt{g}\, t}{\sqrt{l}}\right) + \_C2 \cos\left(\frac{\sqrt{g}\, t}{\sqrt{l}}\right) \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \_C1 \sin\left(\frac{\sqrt{g}\, t}{\sqrt{l}}\right) + \_C2 \cos\left(\frac{\sqrt{g}\, t}{\sqrt{l}}\right) \\ 0 \end{bmatrix} \right]$$

We find: $R2$, a particular solution eta of $R2\,\eta = \tau$, and the general solution τ of $R1\,\tau = 0$.

```
> Exti(Involution(R, Alg), Alg, 1)[2];
```

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & D^2\, l+g & -g \end{bmatrix}$$

```
> ApplyMatrix(P[1], P[2], Alg);
```

$$\begin{bmatrix} \_C1 \sin\left(\frac{\sqrt{g}\, t}{\sqrt{l}}\right) + \_C2 \cos\left(\frac{\sqrt{g}\, t}{\sqrt{l}}\right) \\ 0 \end{bmatrix}$$

Example 2: Partial differential equations

Linear system of PDEs that appears in mathematical physics, namely in the study of Lie-Poisson structures (see C. M. Bender, G. V. Dunne, L. R. Mead, *Underdetermined systems of partial differential equations*, Journal of Mathematical Physics, vol. 41, no. 9 (2000), pp. 6388-6398 and W. M. Seiler, *Involution analysis of the partial differential equations characterising Hamiltonian vector fields*, Journal of Mathematical Physics, vol. 44 (2003), pp. 1173-1182):

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[x1*D3, x2*D3, 0], [-x1*D2+x2*D1, -1, x2*D3], [-1, -x2*D1+x1*D2, x1*D3]]);
```

$$R := \begin{bmatrix} x1\, D3 & x2\, D3 & 0 \\ -x1\, D2 + x2\, D1 & -1 & x2\, D3 \\ -1 & -x2\, D1 + x1\, D2 & x1\, D3 \end{bmatrix}$$

```
> P := ParticularSolution(R, Alg);
```

$P :=$

$$\left[\begin{bmatrix} x1 & x2 & 0 \\ D1 & D2 & D3 \\ -1 & -x2\, D1 + x1\, D2 & x1\, D3 \end{bmatrix}, [\ ], \begin{bmatrix} \int x1\, \_F1(x1^2+x2^2)\, dx1 + \int x2 \left(-2 \int D(\_F1)(x1^2+x2^2)\, x1\, dx1 + \_F1(x1^2+x2^2) \right) dx2 + \_C1 \\ \_F1(x1^2+x2^2) \\ 0 \end{bmatrix} \right]$$

We find: $R2$, a particular solution η of $R2\,\eta = \tau$, and the general solution τ of $R1\,\tau = 0$.

```
> Exti(Involution(R, Alg), Alg, 1)[2];
```

$$\begin{bmatrix} x1 & x2 & 0 \\ D1 & D2 & D3 \\ -1 & -x2\, D1 + x1\, D2 & x1\, D3 \end{bmatrix}$$

```
> ParticularSolutionRat(R, Alg);
```

$$\left[\begin{bmatrix} x1 & x2 & 0 \\ 0 & x1\,x2\,\mathrm{D1} - x1^2\,\mathrm{D2} - x2 & -x1^2\,\mathrm{D3} \end{bmatrix}, [\,], \begin{bmatrix} -\_\mathrm{F1}(x1^2 + x2^2) \\ \_\mathrm{F1}(x1^2 + x2^2) \end{bmatrix}\right]$$

```
> ExtiRat(Involution(R, Alg), Alg, 1)[2];
```

$$\begin{bmatrix} x1 & x2 & 0 \\ 0 & x1\,x2\,\mathrm{D1} - x1^2\,\mathrm{D2} - x2 & -x1^2\,\mathrm{D3} \end{bmatrix}$$

See Also:

DefineOreAlgebra, Parametrization, IntTorsion, Complement, MinimalParametrization, Exti, Extn, Torsion, TorsionElements, AutonomousElements, PiPolynomial.

OreModules[PiPolynomial] - return a Groebner basis of the ideal of π-polynomials of a given linear system with constant coefficients

Calling Sequence:

PiPolynomial(R,Alg,v)

Parameters:

`R`	-	matrix with entries in **`Alg`** with constant coefficients
`Alg`	-	Ore algebra (given by DefineOreAlgebra)
`v`	-	indeterminate or list of indeterminates

Description:

- ***PiPolynomial*** returns a Groebner basis of the ideal of the (commutative) polynomial ring in the indeterminate(s) **`v`** such that for every of its non-zero elements π the localization of the **`Alg`**-module which is presented by **`R`** with respect to the multiplicatively closed set of all powers of π is free. The command ***PiPolynomial*** is restricted to matrices **`R`** over **`Alg`** whose entries have constant coefficients (and hence commute).
- Each non-zero element of the ideal generated by the result of ***PiPolynomial*** is called a π-polynomial for the given system over the Ore algebra **`Alg`** with constant coefficients. For every π-polynomial π, the tensor product of the localization **`Alg`**$[\pi^{(-1)}]$ with respect to the set of powers of π with the **`Alg`**-module presented by **`R`** is a free **`Alg`**$[\pi^{(-1)}]$-module.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`v`** is one of the indeterminates which were used to define **`Alg`** or a list of those.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- For more details about π-polynomials, see H. Mounier, "Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques", PhD Thesis, University of Orsay, France, 1995, and F. Chyzak, A. Quadrat, D. Robertz, "Effective algorithms for parametrizing linear control systems over Ore algebras", Applicable Algebra in Engineering, Communication and Computing (AAECC) 16 (2005), pp. 319-376.

Examples:

```
> with(OreModules):
```

Example 1:

Linear differential time-delay system of a wind tunnel model (see A. Manitius, *Feedback controllers for a wind tunnel model involving a delay: analyical design and numerical simulations*, IEEE Trans. Autom. Contr. vol. 29 (1984), pp. 1058-1068):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  comm=[a,omega,zeta,k], shift_action=[delta,t,h]):
> R := evalm([[Dt+a, -k*a*delta, 0, 0], [0, Dt, -1, 0], [0, omega^2, Dt+2*zeta*omega,
  -omega^2]]);
```

$$R := \begin{bmatrix} Dt+a & -k\,a\,\delta & 0 & 0 \\ 0 & Dt & -1 & 0 \\ 0 & \omega^2 & Dt+2\,\zeta\,\omega & -\omega^2 \end{bmatrix}$$

```
> Ext1 := Exti(Involution(R, Alg), Alg, 1);
```

$$Ext1 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} Dt+a & -k\,a\,\delta & 0 & 0 \\ 0 & \omega^2 & Dt+2\,\zeta\,\omega & -\omega^2 \\ 0 & Dt & -1 & 0 \end{bmatrix}, \begin{bmatrix} -\omega^2\,k\,a\,\delta \\ -Dt\,\omega^2 - a\,\omega^2 \\ -\omega^2\,Dt^2 - \omega^2\,a\,Dt \\ -Dt^3 - 2\,Dt^2\,\zeta\,\omega - a\,Dt^2 - Dt\,\omega^2 - 2\,a\,Dt\,\zeta\,\omega - a\,\omega^2 \end{bmatrix}\right]$$

Ext1[3] provides us with a parametrization of the system. We try to compute a left inverse of Ext1[3]:

```
> LeftInverse(Ext1[3], Alg);
```

$$[\,]$$

Hence, no left inverse of Ext1[3] over **Alg** exists. The obstructions are given by the following possible π-polynomials:

```
> PiPolynomial(R, Alg);
```

$$[\delta, Dt+a]$$

We consider the localization of **Alg** with respect to the multiplicatively closed set of powers of δ and compute a left inverse of Ext1[3] over this localization:

```
> L := LocalLeftInverse(Ext1[3], [delta], Alg);
```

$$L := \begin{bmatrix} -\frac{1}{\delta\,\omega^2\,k\,a} & 0 & 0 & 0 \end{bmatrix}$$

```
> Mult(L, Ext1[3], Alg);
```

$$[\;1]$$

Hence, we obtain a flat output of the system over the localized ring, i.e. the localization of the corresponding module is free:

```
> evalm([[xi1(t)]])=ApplyMatrix(L, [x1(t),x2(t),x3(t),u(t)], Alg);
```

$$[\xi 1(t)] = \begin{bmatrix} -\frac{\mathrm{x1}(t+h)}{\omega^2\,k\,a} \end{bmatrix}$$

Example 2:

Differential time-delay system describing an electric transmission line (see D. Salamon, *Control and Observation of Neutral Systems*, Pitman, 1984, and H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD Thesis, University of Orsay, France, 1995):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], diff=[delta,s], polynom=[t,s],
  comm=[a[0],a[1],a[2],a[3],a[4],a[5],b[0]]):
> R := evalm([[Dt+a[0], -(a[4]*Dt+a[0])*delta, -a[0], 0, -b[0]*Dt],
              [-delta*(a[5]*Dt+a[1]), Dt+a[1], 0, a[1], 0],
              [a[2], -a[2]*a[4]*delta, Dt, 0, -a[2]*b[0]],
              [a[3]*a[5]*delta, -a[3], 0, Dt, 0]]);
```

$$R := \begin{bmatrix} Dt+a_0 & -(a_4\,Dt+a_0)\,\delta & -a_0 & 0 & -b_0\,Dt \\ -\delta\,(a_5\,Dt+a_1) & Dt+a_1 & 0 & a_1 & 0 \\ a_2 & -a_2\,a_4\,\delta & Dt & 0 & -a_2\,b_0 \\ a_3\,a_5\,\delta & -a_3 & 0 & Dt & 0 \end{bmatrix}$$

```
> Exti(Involution(R, Alg), Alg, 1)[1];
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hence, the first extension module of the transposed module of the module presented by **R** with values in **Alg** is zero.

```
> Exti(Involution(R, Alg), Alg, 2)[1];
```

$$[\delta^3\,a_1^{\;2}\,a_2\,a_0 + a_0^{\;2}\,a_5^{\;2}\,a_2^{\;2}\,\delta^3 - 2\,a_0\,a_5^{\;2}\,a_2\,a_1\,a_3\,\delta^3 + a_1^{\;2}\,a_3^{\;2}\,a_5^{\;2}\,\delta^3 - \delta\,Dt\,a_2\,a_1\,a_0 + \delta\,Dt\,a_1^{\;2}\,a_3 + \delta\,a_0\,a_5\,a_2\,a_1\,Dt - \delta\,a_1^{\;2}\,a_3\,a_5\,Dt$$
$$+ 2\,a_0\,a_2\,a_1\,a_3\,a_5\,\delta - \delta\,a_1^{\;2}\,a_3^{\;2}\,a_5 - a_0^{\;2}\,a_5\,a_2^{\;2}\,\delta - a_1^{\;2}\,\delta\,a_2\,a_0]$$

$$[\delta^2\,Dt\,a_1 - Dt^2 + a_1\,a_3\,a_5\,\delta^2 - a_1\,Dt - a_1\,a_3 - a_5\,\delta^2\,a_2\,a_0]$$

$$[\delta Dt^2 + \delta a_2 a_0]$$

$$[a_1 Dt^3 + a_1^2 Dt^2 + a_0 a_5 a_2 Dt^2 - a_5 a_1 Dt^2 a_3 + \delta^2 a_1^2 a_2 a_0 + a_0^2 a_5^2 a_2^2 \delta^2 - 2 a_0 a_5^2 a_2 a_1 a_3 \delta^2 + a_1^2 a_3^2 a_5^2 \delta^2 + Dt a_1^2 a_3 + a_0 a_5 a_2 a_1 Dt - a_1^2 a_3 a_5 Dt + a_0 a_1 a_2 a_3 a_5 - a_5 a_1^2 a_3^2]$$

But the second extension module of the transposed module of the module presented by **R** with values in **Alg** is non-zero.

```
> pi := PiPolynomial(R, Alg, [delta]);
```

$$\pi := [a_0^2 a_2^2 \delta^5 a_5^2 - 2 a_0^2 a_2^2 a_5 \delta^3 + a_0^2 a_2^2 \delta - 2 a_0 a_2 a_1 a_3 \delta^5 a_5^2 + 4 a_0 a_2 a_1 a_3 \delta^3 a_5 + a_1^2 \delta a_2 a_0 - 2 \delta^3 a_1^2 a_2 a_0 - 2 \delta a_1 a_3 a_2 a_0 + a_0 a_2 \delta^5 a_1^2 + a_1^2 a_3^2 \delta^5 a_5^2 - 2 a_1^2 a_3^2 \delta^3 a_5 + a_1^2 a_3^2 \delta]$$

```
> factor(pi);
```

$$[\delta (a_0^2 a_5^2 a_2^2 \delta^4 - 2 \delta^2 a_5 a_0^2 a_2^2 + a_0^2 a_2^2 - 2 a_0 a_5^2 a_2 a_1 a_3 \delta^4 + 4 a_5 \delta^2 a_3 a_0 a_2 a_1 + a_1^2 a_2 a_0 - 2 \delta^2 a_1^2 a_2 a_0 - 2 a_1 a_3 a_0 a_2 + \delta^4 a_1^2 a_2 a_0 + a_1^2 a_3^2 a_5^2 \delta^4 - 2 \delta^2 a_5 a_1^2 a_3^2 + a_1^2 a_3^2)]$$

```
> pi := PiPolynomial(R, Alg, [Dt]);
```

$$\pi := [a_1 a_3 Dt^2 + a_0 a_2 a_1 Dt + a_1 a_3 a_0 a_2 + a_1 Dt^3 + Dt^2 a_0 a_2 + Dt^4]$$

```
> factor(pi);
```

$$[(a_1 Dt + a_1 a_3 + Dt^2)(Dt^2 + a_0 a_2)]$$

See Also:

DefineOreAlgebra, Exti, Extn, LeftInverse, RightInverse, LocalLeftInverse, Brunovsky, FirstIntegral, ControllabilityMatrix, Parametrization, IntTorsion, ParticularSolution.

OreModules[PolIntersect] - intersect two left ideals of an Ore algebra

Calling Sequence:

PolIntersect(L,v,Alg)

Parameters:

`L`	-	list of polynomials in **`Alg`**
`v`	-	list of indeterminates in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***PolIntersect*** computes a Groebner basis (w.r.t. the degree-reverse lexicographical term order) of the intersection of the left ideal generated by **`L`** in the Ore algebra defined by **`Alg`** and the (skew) polynomial ring with indeterminates in **`v`**.
- **`L`** is a list of polynomials in **`Alg`** which generate the left ideal of **`Alg`** to be intersected with the (skew) polynomial ring in the variables **`v`**.
- The indeterminates in the list **`v`** must be among those indeterminates which were used to define **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***PolIntersect*** is a list of polynomials in **`Alg`**.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> L := [x1*D2, x2*D1];
```

$$L := [x1\, D2, x2\, D1]$$

```
> v := [x2,D2];
```

$$v := [x2, D2]$$

```
> PolIntersect(L, v, Alg);
```

$$[2\, D2 + x2\, D2^2, x2^2\, D2]$$

```
> Alg := DefineOreAlgebra(diff=[D,t], shift=[delta,s], polynom=[s,t]):
> L := [D*delta+s, delta^2];
```

$$L := [D\,\delta + s, \delta^2]$$

```
> v := [s];
```

$$v := [s]$$

```
> PolIntersect(L, v, Alg);
```

$$[s^2 + s]$$

```
> PolIntersect(L, [D,delta,t,s], Alg);
```

$$[s^2 + s, \delta\, s + \delta, \delta^2, D\,\delta + s]$$

See Also:

DefineOreAlgebra, IdealIntersection, Mult, ApplyMatrix, Involution, KroneckerProduct, Factorize, Quotient, Elimination, Integrability, ReduceMatrix, SyzygyModule.

OreModules[ProjectiveDimension],

OreModules[ProjectiveDimensionRat] - **compute the projective dimension of a finitely presented module over an Ore algebra**

Calling Sequence:

ProjectiveDimension(R,Alg)
ProjectiveDimensionRat(R,Alg)

Parameters:

`R` - matrix with entries in **`Alg`**
`Alg` - Ore algebra (given by DefineOreAlgebra)

Description:

- ***ProjectiveDimension*** returns the (left) projective dimension of the left module over the Ore algebra **`Alg`** which is presented by **`R`**.
- The (left) projective dimension of a finitely presented left **`Alg`**-module *M* is the minimal length of a projective resolution of *M*. All Ore algebras in the scope of OreModules have finite (left) global dimension which is an upper bound on the (left) projective dimension of left **`Alg`**-modules. Hence, the result of ***ProjectiveDimension*** is always a non-negative integer.
- The left **`Alg`**-module which is considered by ***ProjectiveDimension*** is the factor module of the free **`Alg`**-module of row vectors whose length equals the number of columns of **`R`** modulo the submodule which is generated by the rows of **`R`**.
- ***ProjectiveDimension*** computes a free resolution of the left **`Alg`**-module presented by **`R`** and reduces the length of this resolution as much as possible using the same methods as ShorterFreeResolution and ShortestFreeResolution. As soon as the resolution cannot be shortened anymore (i.e. when the last morphism in the resolution does not admit a right inverse), ***ProjectiveDimension*** returns the length of this resolution. If ***ProjectiveDimension*** arrives at a free resolution of length 1 and the presentation matrix still admits a right inverse, then it returns 0.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- ***ProjectiveDimensionRat*** performs the same computations as ***ProjectiveDimension***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- For more details, see T. Y. Lam, "Lectures on Modules and Rings", Springer, 1999, and A. Quadrat, D. Robertz, "Computation of bases of free modules over the Weyl algebras", Journal of Symbolic Computation 42 (11-12), 2007, pp. 1113-1141.

Examples:

```
> with(OreModules):
```

Example 1:

(see J.-F. Pommaret, *Partial Differential Equations and Group Theory: New Perspectives for Applications Kluwer*, 1994, p. 162)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[1], [D1], [D2], [D3]]);
```

$$R := \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix}$$

```
> ShortestFreeResolution(R, Alg);
```

$$\text{table}([1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ 0 & D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 2 = \text{INJ}(8)])$$

```
> ProjectiveDimension(R, Alg);
```

$$0$$

Hence, the (left) **`Alg`**-module which is presented by **`R`** is projective. For the details, see ShorterFreeResolution, Example 1.

Example 2:

(see J.-F. Pommaret, *Partial Differential Control Theory*, Kluwer, 2001, p. 665)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[x2*D1, 1], [x2*D2, 0], [D1, D2]]);
```

$$R := \begin{bmatrix} x2\,D1 & 1 \\ x2\,D2 & 0 \\ D1 & D2 \end{bmatrix}$$

```
> ShortestFreeResolution(R, Alg);
```

$$\text{table}([1 = \begin{bmatrix} x2\,D1 & 1 & 0 \\ x2\,D2 & 0 & 0 \\ D1 & D2 & 1 \end{bmatrix}, 2 = \text{INJ}(3)])$$

```
> ProjectiveDimension(R, Alg);
```

$$1$$

See Also:

DefineOreAlgebra, SyzygyModule, FreeResolution, ShorterFreeResolution, ShortestFreeResolution, Resolution, Exti, Extn, Torsion, Parametrization, MinimalParametrization, Involution, Quotient, Integrability.

OreModules[Quotient],

OreModules[QuotientRat] - **return annihilators of elements in a finitely presented module over an Ore algebra**

Calling Sequence:

Quotient(R1,R2,Alg)
QuotientRat(R1,R2,Alg)

Parameters:

`R1, R2`	-	matrices with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- For each row of **`R1`**, ***Quotient*** computes the left ideal of **`Alg`** containing all elements λ such that the left λ-multiple of the row of **`R1`** is in the left **`Alg`**-module generated by the rows of **`R2`**, i.e., ***Quotient*** computes the annihilators of the residue classes of the rows of **`R1`** in the left module over **`Alg`** which is presented by **`R2`**, i.e. of the factor module of the free **`Alg`**-module of tuples whose length equals the number of columns of **`R2`** modulo the submodule which is generated by the rows of **`R2`**.
- **`R1`** and **`R2`** are matrices with entries in the Ore algebra **`Alg`** having the same number of columns.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***Quotient*** is a matrix having a block diagonal structure, where each block consists of only one column but may have several rows. The number of blocks equals the number of rows of **`R1`**. The entries of the *i*th block form a Groebner basis (w.r.t. the degree reverse lexicographical ordering on the variables of **`Alg`**) of the annihilator of the residue class of the *i*th row of **`R1`** in the left **`Alg`**-module presented by **`R2`**.
- ***QuotientRat*** performs the same computations as ***Quotient***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- ReduceMatrix computes the normal form of each row in a given matrix over **`Alg`** modulo the Groebner basis of the rows of a second matrix over **`Alg`**.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t]):
> R1 := evalm([[D, t], [0, D]]);
```

$$R1 := \begin{bmatrix} D & t \\ 0 & D \end{bmatrix}$$

```
> R2 := Mult(evalm([[1,1],[0,1]]), R1, Alg);
```

$$R2 := \begin{bmatrix} D & t+D \\ 0 & D \end{bmatrix}$$

```
> Quotient(R1, R2, Alg);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Hence, the left *Alg*-modules generated by the rows of *R1* resp. *R2* are equal.

```
> R3 := evalm([[D, t], [1, 0]]);
```

$$R3 := \begin{bmatrix} D & t \\ 1 & 0 \end{bmatrix}$$

```
> Quotient(R1, R3, Alg);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & t^2 \\ 0 & 2 + t\,\mathrm{D} \end{bmatrix}$$

Hence, the first row of *R1* is an element of the left *Alg*-module *M3* generated by the rows of *R3*, and λ times the second row of *R1* lies in *M3* if and only if λ is a left *Alg*-linear combination of t^2 and $2 + t\,\mathrm{D}$.

```
> Quotient(R3, R1, Alg);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & -\mathrm{D} + t\,\mathrm{D}^2 \\ 0 & \mathrm{D}^3 \end{bmatrix}$$

Hence, the first row of *R3* is an element of the left *Alg*-module *M1* generated by the rows of *R1*, and λ times the second row in *R3* lies in *M1* if and only if λ is a left *Alg*-linear combination of $-\mathrm{D} + t\,\mathrm{D}^2$ and D^3.

See Also:

DefineOreAlgebra, Factorize, Elimination, Integrability, ReduceMatrix, Involution, SyzygyModule, Exti.

OreModules[ReduceMatrix],

OreModules[ReduceMatrixRat] - **reduce the rows of a matrix over an Ore algebra modulo the rows of another one**

Calling Sequence:

ReduceMatrix(R1,R2,Alg)
ReduceMatrixRat(R1,R2,Alg)

Parameters:

`R1, R2`	-	matrices with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***ReduceMatrix*** computes the normal form of each row in **`R1`** modulo the Groebner basis of the rows of **`R2`** (w.r.t. the degree-reverse lexicographical ordering on the variables in **`Alg`**) and returns the matrix whose rows are these normal forms. Zero rows are omitted in the result.
- **`R1`** and **`R2`** are matrices with entries in the Ore algebra **`Alg`** having the same number of columns.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***ReduceMatrix*** is a matrix with the same number of columns as **`R1`** and **`R2`**. The number of rows of the result may be zero.
- ***ReduceMatrixRat*** performs the same computations as ***ReduceMatrix***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- Quotient computes the annihilators of the rows of a given matrix over **`Alg`** in the left **`Alg`**-module presented by a second matrix over **`Alg`**.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R1 := evalm([[D1, 0, D2], [1, D1+D2, D2+1], [0, D2, 0]]);
```

$$R1 := \begin{bmatrix} D1 & 0 & D2 \\ 1 & D1+D2 & D2+1 \\ 0 & D2 & 0 \end{bmatrix}$$

```
> R2 := evalm([[1, D1, D2], [0, D2, 1]]);
```

$$R2 := \begin{bmatrix} 1 & D1 & D2 \\ 0 & D2 & 1 \end{bmatrix}$$

```
> ReduceMatrix(R1, R2, Alg);
```

$$\begin{bmatrix} D1 & 0 & D2 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> ReduceMatrix(R1, R1, Alg);
```

[ ]

See Also:

DefineOreAlgebra, Factorize, Quotient, Elimination, Integrability, Involution, SyzygyModule.

OreModules[Resolution],

OreModules[ResolutionRat] - **compute a given number of left modules in a free resolution of a finitely presented module over an Ore algebra**

Calling Sequence:

Resolution(R,Alg,n)
ResolutionRat(R,Alg,n)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`Alg`	-	Ore algebra (given by DefineOreAlgebra)
`n`	-	natural number

Description:

- ***Resolution*** iterates the computation of syzygy modules of the left module over the Ore algebra **`Alg`** which is presented by **`R`**, i.e. of the factor module of the free **`Alg`**-module of tuples whose length equals the number of columns of **`R`** modulo the submodule which is generated by the rows of **`R`**. That means that ***Resolution*** constructs the beginning of a free resolution of the left module presented by **`R`**.
- If **`n`** > 1, then ***Resolution*** first computes a matrix the rows of which generate all left **`Alg`**-linear relations of the rows of **`R`**. If **`n`** > 2, then ***Resolution*** repeats the same for the matrix which has just been defined instead of **`R`**. All in all, this construction is iterated **`n`**-1 times, i.e. **`n`**-1 new matrices are constructed such that the rows of every matrix generate all left **`Alg`**-linear relations of the rows of the preceding matrix.
- If the rows of one of the matrices that are computed by ***Resolution*** do not satisfy any non-trivial left **`Alg`**-linear relation, then the following entry of the result (if requested) is not a matrix, but the name INJ(r), where r is the number of rows of the previous matrix. If more terms of the free resolution are to be constructed, then the following entries of the result will be the names ZERO.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result is a table which contains matrices with entries **`Alg`** and possibly names INJ(r) and ZERO. The matrix with index 1 in the result is **`R`** and the matrix with index i is the result of SyzygyModule applied to the matrix with index $i-1$, $i>1$, i.e., the rows of the matrix with index i generate the syzygy module of the left module generated by the rows of the matrix with index $i-1$.
- In order to iterate the computation of syzygy modules described above as long as possible, FreeResolution can be used.
- ***ResolutionRat*** performs the same computations as ***Resolution***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
```

Example 1:

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := matrix([[D1], [D2], [D3]]);
```

$$R := \begin{bmatrix} D1 \\ D2 \\ D3 \end{bmatrix}$$

```
> Resolution(R, Alg, 3);
```

$$\mathrm{table}([1 = R, 2 = \begin{bmatrix} -D3 & 0 & D1 \\ -D2 & D1 & 0 \\ 0 & -D3 & D2 \end{bmatrix}, 3 = [-D2 \quad D3 \quad D1]])$$

```
> Res := Resolution(R, Alg, 4);
```

$$Res := \mathrm{table}([1 = R, 2 = \begin{bmatrix} -D3 & 0 & D1 \\ -D2 & D1 & 0 \\ 0 & -D3 & D2 \end{bmatrix}, 3 = [-D2 \quad D3 \quad D1], 4 = \mathrm{INJ}(1)])$$

```
> Mult(Res[2], Res[1], Alg);
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> Mult(Res[3], Res[2], Alg);
```

$$[0 \quad 0 \quad 0]$$

Example 2:

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s]):
> R := matrix([[0,Dt*delta], [t*Dt,t*delta], [Dt,Dt]]);
```

$$R := \begin{bmatrix} 0 & Dt\,\delta \\ t\,Dt & t\,\delta \\ Dt & Dt \end{bmatrix}$$

```
> Res := Resolution(R, Alg, 3);
```

$$Res := \mathrm{table}([1 = R, 2 = \begin{bmatrix} -Dt\,t^2 + \delta\,t^2 & \delta - t\,Dt\,\delta & Dt\,\delta\,t^2 \\ 2\,\delta - Dt^2\,t - 2\,Dt + t\,Dt\,\delta & -Dt^2\,\delta & Dt^2\,\delta\,t + 2\,Dt\,\delta \end{bmatrix}, 3 = [Dt \quad -t]])$$

```
> Mult(Res[2], Res[1], Alg);
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> Mult(Res[3], Res[2], Alg);
```

$$[0 \quad 0 \quad 0]$$

See Also:

DefineOreAlgebra, SyzygyModule, FreeResolution, ShorterFreeResolution, ShortestFreeResolution, ProjectiveDimension, LiftOperators, Exti, Extn, Torsion, Parametrization, MinimalParametrization, Involution, Quotient, Integrability.

OreModules[RightInverse],

OreModules[RightInverseRat] - compute a right inverse of a matrix over an Ore algebra

Calling Sequence:

RightInverse(M,Alg)
RightInverseRat(M,Alg)

Parameters:

M	-	matrix with entries in **Alg** or INJ(n) or SURJ(n), where n is a non-negative integer
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***RightInverse*** computes (if possible) a right inverse of the matrix **M**, i.e. a matrix R with entries in **Alg** such that the product of **M** by R is the identity matrix.
- If no right inverse of **M** exists, ***RightInverse*** returns the empty list.
- **M** is a matrix with entries in **Alg**.
- **Alg** is expected to be defined using DefineOreAlgebra.
- ***RightInverseRat*** performs the same computations as ***RightInverse***, but the domain of coefficients of the Ore algebra **Alg** is replaced by its quotient field, i.e. rational functions.
- Left inverses of matrices over Ore algebras are computed by LeftInverse. Generalized inverses of matrices over Ore algebras are computed by GeneralizedInverse.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> M1 := evalm([[0,1,0],[1,0,0]]);
```

$$M1 := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

```
> R1 := RightInverse(M1, Alg);
```

$$R1 := \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> Mult(M1, R1, Alg);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> M2 := evalm([[-x2*D1+1, D2]]);
```

$$M2 := \begin{bmatrix} -x2\,\mathrm{D1}+1 & \mathrm{D2} \end{bmatrix}$$

```
> R2 := RightInverse(M2, Alg);
```

$$R2 := \begin{bmatrix} 2+x2\,\mathrm{D2} \\ x2^2\,\mathrm{D1}-x2 \end{bmatrix}$$

```
> Mult(M2, R2, Alg);
```

$$\begin{bmatrix} 1 \end{bmatrix}$$

```
> M3 := Involution(M2, Alg);
```

$$M3 := \begin{bmatrix} x2\,\mathrm{D1} + 1 \\ -\mathrm{D2} \end{bmatrix}$$

```
> RightInverse(M3, Alg);
```

$$[\ ]$$

```
> RightInverse(SURJ(3), Alg);
```

$$ZERO$$

```
> RightInverse(INJ(2), Alg);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

See Also:

DefineOreAlgebra, LeftInverse, LocalLeftInverse, GeneralizedInverse, Mult, ApplyMatrix, Involution, KroneckerProduct, Factorize, Quotient, Elimination, Integrability, ReduceMatrix.

OreModules[ShorterFreeResolution],

OreModules[ShorterFreeResolutionRat] - **shorten (if possible) a free resolution of a finitely presented module over an Ore algebra**

Calling Sequence:

ShorterFreeResolution(F,Alg)
ShorterFreeResolutionRat(F,Alg)

Parameters:

`F`	-	table representing a free resolution of a finitely presented module over **`Alg`** (e.g. given by FreeResolution)
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- Given a (finite) free resolution of a finitely presented left module over the Ore algebra **`Alg`**, ***ShorterFreeResolution*** tries to construct a shorter free resolution of the same module. This is possible whenever the last morphism between free modules in this free resolution admits a right inverse (see RightInverse).
- If the length m of the free resolution given by **`F`** is at least 3 and if the last morphism R_m between free modules given in **`F`** admits a right inverse S_m, then a shorter free resolution is obtained by removing the last free module, augmenting the last but first morphism R_{m-1} with S_m, i.e. replacing it by $(R_{m-1}\ S_m)$, and replacing the last but second morphism R_{m-2} by the transpose of $(R_{m-2}\ 0)$ in a compatible way (note also that the last but second free module in the given free resolution must be adjusted).
- If the length m of the free resolution given by **`F`** equals 2 and if the last morphism R_2 between free modules given in **`F`** admits a right inverse S_2, then a presentation of the module resolved by **`F`** is obtained by removing the last free module and augmenting the last but first morphism R_1 with S_2, i.e. by defining the presentation matrix $(R_1\ S_2)$.
- If the length m of the free resolution given by **`F`** is less than 2, then ***ShorterFreeResolution*** returns **`F`**.
- **`F`** is a table which represents a free resolution of a finitely presented left module over **`Alg`**. Most commonly, **`F`** is the result of either FreeResolution or Resolution.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***ShorterFreeResolution*** is of the same format as the input **`F`**, i.e. a table representing a free resolution of a finitely presented left module over **`Alg`** (see FreeResolution), which is shorter than the given one or equals the given one.
- The procedure described above can be iterated using the command ShortestFreeResolution.
- ***ShorterFreeResolutionRat*** performs the same computations as ***ShorterFreeResolution***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- For more details, see A. Quadrat, D. Robertz, "Computation of bases of free modules over the Weyl algebras", Journal of Symbolic Computation 42 (11-12), 2007, pp. 1113-1141.

Examples:

```
> with(OreModules):
```

Example 1:

(see J.-F. Pommaret, *Partial Differential Equations and Group Theory: New Perspectives for Applications Kluwer*, 1994, p. 162)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[1], [D1], [D2], [D3]]);
```

$$R := \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix}$$

We start with a free resolution of the (left) **`Alg`**-module presented by *R*:

```
> F := FreeResolution(R, Alg);
```

$$F := \text{table}([1 = \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix}, 2 = \begin{bmatrix} D3 & 0 & 0 & -1 \\ D2 & 0 & -1 & 0 \\ D1 & -1 & 0 & 0 \\ 0 & -D3 & 0 & D1 \\ 0 & -D2 & D1 & 0 \\ 0 & 0 & -D3 & D2 \end{bmatrix}, 3 = \begin{bmatrix} D2 & -D3 & 0 & 0 & 0 & 1 \\ D1 & 0 & -D3 & 1 & 0 & 0 \\ 0 & D1 & -D2 & 0 & 1 & 0 \\ 0 & 0 & 0 & -D2 & D3 & D1 \end{bmatrix}, 5 = \text{INJ}(1), 4 = [D1 \quad -D2 \quad D3 \quad -1]])$$

```
> ShorterFreeResolution(F, Alg);
```

$$\text{table}([1 = \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix}, 2 = \begin{bmatrix} D3 & 0 & 0 & -1 \\ D2 & 0 & -1 & 0 \\ D1 & -1 & 0 & 0 \\ 0 & -D3 & 0 & D1 \\ 0 & -D2 & D1 & 0 \\ 0 & 0 & -D3 & D2 \\ 0 & 0 & 0 & 0 \end{bmatrix}, 3 = \begin{bmatrix} D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 4 = \text{INJ}(4)])$$

```
> ShorterFreeResolution(%, Alg);
```

$$\text{table}([1 = \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, 2 = \begin{bmatrix} D3 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -D3 & 0 & D1 & 0 & 1 & 0 & 0 \\ 0 & -D2 & D1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -D3 & D2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D1 & -D2 & D3 & -1 \end{bmatrix}, 3 = \text{INJ}(7)])$$

```
> ShorterFreeResolution(%, Alg);
```

$$\mathrm{table}\left(\left[1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ 0 & D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 2 = \mathrm{INJ}(8)\right]\right)$$

```
> ShorterFreeResolution(%, Alg);
```

$$\mathrm{table}\left(\left[1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ 0 & D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 2 = \mathrm{INJ}(8)\right]\right)$$

Hence, it was possible to reduce the length of the free resolution represented by **F** in each step, finally arriving at a free resolution of length 1. These steps can be done at once by calling ShortestFreeResolution:

```
> ShortestFreeResolution(F, Alg);
```

$$\mathrm{table}\left(\left[1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ 0 & D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 2 = \mathrm{INJ}(8)\right]\right)$$

In fact, the module presented by *R* is stably free because a right inverse of the presentation matrix obtained by ShortestFreeResolution admits a right inverse:

```
> RightInverse(%[1], Alg);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D3 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -D3 & 0 & D1 & 0 & 1 & 0 & 0 \\ 0 & -D2 & D1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -D3 & D2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D1 & -D2 & D3 & -1 \end{bmatrix}$$

In particular, this module is projective, which can also be checked via ProjectiveDimension:

```
> ProjectiveDimension(R, Alg);
```

$$0$$

Example 2:

Spencer operator (see J.-F. Pommaret, *Partial Differential Equations and Group Theory: New Perspectives for Applications*, Kluwer, 1994, p. 163)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[1], [D1], [D2], [D1^2], [D1*D2], [D2^2]]);
```

$$R := \begin{bmatrix} 1 \\ D1 \\ D2 \\ D1^2 \\ D1\,D2 \\ D2^2 \end{bmatrix}$$

We start again with a free resolution of the (left) **`Alg`**-module presented by *R*:

```
> F := FreeResolution(R, Alg);
```

$$F := \text{table}([1 = \begin{bmatrix} 1 \\ D1 \\ D2 \\ D1^2 \\ D1\,D2 \\ D2^2 \end{bmatrix}, 2 = \begin{bmatrix} D2 & 0 & -1 & 0 & 0 & 0 \\ D1 & -1 & 0 & 0 & 0 & 0 \\ 0 & D2 & 0 & 0 & -1 & 0 \\ 0 & D1 & 0 & -1 & 0 & 0 \\ 0 & 0 & D2 & 0 & 0 & -1 \\ 0 & 0 & D1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -D2 & D1 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D1 \end{bmatrix}, 3 = \begin{bmatrix} D1 & -D2 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & D1 & -D2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & D1 & -D2 & 0 & 1 \end{bmatrix}, 4 = \text{INJ}(3)])$$

```
> ShorterFreeResolution(F, Alg);
```

$$\mathrm{table}\left(\left[1=\begin{bmatrix}1\\ D1\\ D2\\ D1^{2}\\ D1\,D2\\ D2^{2}\\ 0\\ 0\\ 0\end{bmatrix},\ 2=\begin{bmatrix}D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0\\ D1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & D2 & 0 & 0 & -1 & 0 & 0 & 0 & 0\\ 0 & D1 & 0 & -1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & D2 & 0 & 0 & -1 & 0 & 0 & 0\\ 0 & 0 & D1 & 0 & -1 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & -D2 & D1 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & -D2 & D1 & D2 & 0 & 1\end{bmatrix},\ 3=\mathrm{INJ}(8)\right]\right)$$

```
> ShorterFreeResolution(%, Alg);
```

$$\mathrm{table}\left(\left[1=\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ D1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0\\ D2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ D1^{2} & 0 & -D1 & 0 & -1 & 0 & 0 & 0 & 0\\ D1\,D2 & 0 & -D2 & -1 & 0 & 0 & 0 & 0 & 0\\ D2^{2} & -D2 & 0 & 0 & 0 & -1 & 0 & 0 & 0\\ 0 & D1 & -D2 & -1 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & D1 & -D2 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & D1 & -D2 & 0 & 1\end{bmatrix},\ 2=\mathrm{INJ}(9)\right]\right)$$

```
> ShorterFreeResolution(%, Alg);
```

$$\mathrm{table}\left(\left[1=\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ D1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0\\ D2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ D1^{2} & 0 & -D1 & 0 & -1 & 0 & 0 & 0 & 0\\ D1\,D2 & 0 & -D2 & -1 & 0 & 0 & 0 & 0 & 0\\ D2^{2} & -D2 & 0 & 0 & 0 & -1 & 0 & 0 & 0\\ 0 & D1 & -D2 & -1 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & D1 & -D2 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & D1 & -D2 & 0 & 1\end{bmatrix},\ 2=\mathrm{INJ}(9)\right]\right)$$

```
> ShortestFreeResolution(F, Alg);
```

$$\mathrm{table}\left(\left[1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ D2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1^2 & 0 & -D1 & 0 & -1 & 0 & 0 & 0 & 0 \\ D1\,D2 & 0 & -D2 & -1 & 0 & 0 & 0 & 0 & 0 \\ D2^2 & -D2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & D1 & -D2 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & D1 & -D2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & D1 & -D2 & 0 & 1 \end{bmatrix}, 2 = \mathrm{INJ}(9)\right]\right)$$

Again we arrived at a free resolution of the presented module of length 1. The presented module is stably free because the presentation matrix admits a right inverse:

```
> RightInverse(%[1], Alg);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & D1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & D2 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & D1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -D2 & D1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D1 & D2 & 0 & 1 \end{bmatrix}$$

In particular, the module is projective (ProjectiveDimension actually performs the same steps as above to compute the projective dimension):

```
> ProjectiveDimension(R, Alg);
```

$$0$$

Example 3:

(see J.-F. Pommaret, *Partial Differential Control Theory*, Kluwer, 2001, p. 665)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[x2*D1, 1], [x2*D2, 0], [D1, D2]]);
```

$$R := \begin{bmatrix} x2\,D1 & 1 \\ x2\,D2 & 0 \\ D1 & D2 \end{bmatrix}$$

```
> F := FreeResolution(R, Alg);
```

$$F := \mathrm{table}\left(\left[1 = \begin{bmatrix} x2\,D1 & 1 \\ x2\,D2 & 0 \\ D1 & D2 \end{bmatrix}, 2 = \begin{bmatrix} -D2 & D1 & 1 \end{bmatrix}, 3 = \mathrm{INJ}(1)\right]\right)$$

```
> ShorterFreeResolution(F, Alg);
```

$$\text{table}([1 = \begin{bmatrix} x2\,\text{D1} & 1 & 0 \\ x2\,\text{D2} & 0 & 0 \\ \text{D1} & \text{D2} & 1 \end{bmatrix}, 2 = \text{INJ}(3)])$$

```
> ShorterFreeResolution(%, Alg);
```

$$\text{table}([1 = \begin{bmatrix} x2\,\text{D1} & 1 & 0 \\ x2\,\text{D2} & 0 & 0 \\ \text{D1} & \text{D2} & 1 \end{bmatrix}, 2 = \text{INJ}(3)])$$

Here we arrive at a free resolution of the left module presented by *R* of length 1, but the presentation matrix does not admit a right inverse:

```
> RightInverse(%[1], Alg);
```

[ ]

The presented module is not stably free. It is not projective either as shown by the following computation of its projective dimension:

```
> ProjectiveDimension(R, Alg);
```

1

See Also:

DefineOreAlgebra, SyzygyModule, FreeResolution, ShortestFreeResolution, Resolution, ProjectiveDimension, Exti, Extn, Torsion, Parametrization, MinimalParametrization, Involution, Quotient, Integrability.

OreModules[ShortestFreeResolution],

OreModules[ShortestFreeResolutionRat] - **return a shortest free resolution of a finitely presented module over an Ore algebra**

Calling Sequence:

ShortestFreeResolution(F,Alg)
ShortestFreeResolutionRat(F,Alg)

Parameters:

`F` - matrix with entries in **`Alg`** or table representing a free resolution of a finitely presented module over **`Alg`** (e.g. given by FreeResolution)

`Alg` - Ore algebra (given by DefineOreAlgebra)

Description:

- ***ShortestFreeResolution*** iterates the application of ShorterFreeResolution to a finite free resolution of a finitely presented left module over the Ore algebra **`Alg`** and returns a free resolution of the same module which cannot be shortened in this way anymore.
- **`F`** is either a matrix with entries in **`Alg`** or a table which represents a free resolution of a finitely presented left module over **`Alg`**. In the first case, a free resolution of the left **`Alg`**-module presented by **`F`** is computed first. In the second case, most commonly, **`F`** is the result of either FreeResolution or Resolution. Then, in both cases, ShorterFreeResolution is applied repeatedly to the resolution until ShorterFreeResolution does not change the resolution anymore.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***ShortestFreeResolution*** is of the same format as the input **`F`**, i.e. a table representing a free resolution of a finitely presented left module over **`Alg`** (see FreeResolution).
- ***ShortestFreeResolutionRat*** performs the same computations as ***ShortestFreeResolution***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- For more details, see A. Quadrat, D. Robertz, "Computation of bases of free modules over the Weyl algebras", Journal of Symbolic Computation 42 (11-12), 2007, pp. 1113-1141.

Examples:

```
> with(OreModules):
```

Example 1:

(see J.-F. Pommaret, *Partial Differential Equations and Group Theory: New Perspectives for Applications Kluwer*, 1994, p. 162)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[1], [D1], [D2], [D3]]);
```

$$R := \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix}$$

```
> ShortestFreeResolution(R, Alg);
```

$$\mathrm{table}([1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ 0 & D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 2 = \mathrm{INJ}(8)])$$

We show how ***ShortestFreeResolution*** applies ShorterFreeResolution to a free resolution of the (left) **Alg**-module presented by *R* (for more details see ShorterFreeResolution):

```
> F := FreeResolution(R, Alg);
```

$$F := \mathrm{table}([1 = \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix}, 2 = \begin{bmatrix} D3 & 0 & 0 & -1 \\ D2 & 0 & -1 & 0 \\ D1 & -1 & 0 & 0 \\ 0 & -D3 & 0 & D1 \\ 0 & -D2 & D1 & 0 \\ 0 & 0 & -D3 & D2 \end{bmatrix}, 3 = \begin{bmatrix} D2 & -D3 & 0 & 0 & 0 & 1 \\ D1 & 0 & -D3 & 1 & 0 & 0 \\ 0 & D1 & -D2 & 0 & 1 & 0 \\ 0 & 0 & 0 & -D2 & D3 & D1 \end{bmatrix}, 5 = \mathrm{INJ}(1), 4 = [D1 \;\; -D2 \;\; D3 \;\; -1]])$$

```
> ShorterFreeResolution(F, Alg);
```

$$\mathrm{table}([1 = \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix}, 2 = \begin{bmatrix} D3 & 0 & 0 & -1 \\ D2 & 0 & -1 & 0 \\ D1 & -1 & 0 & 0 \\ 0 & -D3 & 0 & D1 \\ 0 & -D2 & D1 & 0 \\ 0 & 0 & -D3 & D2 \\ 0 & 0 & 0 & 0 \end{bmatrix}, 3 = \begin{bmatrix} D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 4 = \mathrm{INJ}(4)])$$

```
> ShorterFreeResolution(%, Alg);
```

$$\mathrm{table}([1 = \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, 2 = \begin{bmatrix} D3 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -D3 & 0 & D1 & 0 & 1 & 0 & 0 \\ 0 & -D2 & D1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -D3 & D2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D1 & -D2 & D3 & -1 \end{bmatrix}, 3 = \mathrm{INJ}(7)])$$

```
> ShorterFreeResolution(%, Alg);
```

$$\text{table}([1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ 0 & D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 2 = \text{INJ}(8)])$$

```
> ShorterFreeResolution(%, Alg);
```

$$\text{table}([1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & D2 & -D3 & 0 & 0 & 0 & 1 & 0 \\ 0 & D1 & 0 & -D3 & 1 & 0 & 0 & 0 \\ 0 & 0 & D1 & -D2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -D2 & D3 & D1 & -1 \end{bmatrix}, 2 = \text{INJ}(8)])$$

Example 2:

Spencer operator (see J.-F. Pommaret, *Partial Differential Equations and Group Theory: New Perspectives for Applications*, Kluwer, 1994, p. 163)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[1], [D1], [D2], [D1^2], [D1*D2], [D2^2]]);
```

$$R := \begin{bmatrix} 1 \\ D1 \\ D2 \\ D1^2 \\ D2\,D1 \\ D2^2 \end{bmatrix}$$

```
> ShortestFreeResolution(R, Alg);
```

$$\text{table}([1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ D2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1^2 & 0 & -D1 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2\,D1 & 0 & -D2 & -1 & 0 & 0 & 0 & 0 & 0 \\ D2^2 & -D2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & D1 & -D2 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & D1 & -D2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & D1 & -D2 & 0 & 1 \end{bmatrix}, 2 = \text{INJ}(9)])$$

For the details, see ShorterFreeResolution, Example 2.

Example 3:

(see J.-F. Pommaret, *Partial Differential Control Theory*, Kluwer, 2001, p. 665)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := evalm([[x2*D1, 1], [x2*D2, 0], [D1, D2]]);
```

$$R := \begin{bmatrix} x2\,D1 & 1 \\ x2\,D2 & 0 \\ D1 & D2 \end{bmatrix}$$

```
> ShortestFreeResolution(R, Alg);
```

$$\text{table}([1 = \begin{bmatrix} x2\,D1 & 1 & 0 \\ x2\,D2 & 0 & 0 \\ D1 & D2 & 1 \end{bmatrix}, 2 = \text{INJ}(3)])$$

```
> F := FreeResolution(R, Alg);
```

$$F := \text{table}([1 = \begin{bmatrix} x2\,D1 & 1 \\ x2\,D2 & 0 \\ D1 & D2 \end{bmatrix}, 2 = [-D2 \quad D1 \quad 1], 3 = \text{INJ}(1)])$$

```
> ShortestFreeResolution(F, Alg);
```

$$\text{table}([1 = \begin{bmatrix} x2\,D1 & 1 & 0 \\ x2\,D2 & 0 & 0 \\ D1 & D2 & 1 \end{bmatrix}, 2 = \text{INJ}(3)])$$

See Also:

DefineOreAlgebra, SyzygyModule, FreeResolution, Resolution, ShorterFreeResolution, ProjectiveDimension, Exti, Extn, Torsion, Parametrization, MinimalParametrization, Involution, Quotient, Integrability.

OreModules[SyzygyModule],

OreModules[SyzygyModuleRat] - **return syzygy module of a finitely presented left module over an Ore algebra**

Calling Sequence:

SyzygyModule(R,Alg)
SyzygyModuleRat(R,Alg)

Parameters:

`R` - matrix with entries in **`Alg`** or INJ(*n*) or SURJ(*n*) or ZERO, where *n* is a non-negative integer
`Alg` - Ore algebra (given by DefineOreAlgebra)

Description:

- ***SyzygyModule*** computes the syzygy module of the left module over the Ore algebra given by **`Alg`** which is presented by **`R`**, i.e. of the factor module of the free **`Alg`**-module of tuples whose length equals the number of columns of **`R`** modulo the submodule which is generated by the rows of **`R`**. Hence, the rows of the resulting matrix generate all left **`Alg`**-linear relations of the rows of **`R`**.
- **`R`** is either a matrix with entries in the Ore algebra **`Alg`** or INJ(*n*) or SURJ(*n*) or ZERO, where *n* is a non-negative integer.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***SyzygyModule*** is a matrix whose rows form a Groebner basis of the syzygy module of the left **`Alg`**-module which is presented by **`R`**.
- ***SyzygyModuleRat*** performs the same computations as ***SyzygyModule***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.

Examples:

```
> with(OreModules):
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], polynom=[x1,x2]):
> R := matrix([[D1, 0, 0], [-D2, D1, 1], [0, D1, 1]]);
```

$$R := \begin{bmatrix} D1 & 0 & 0 \\ -D2 & D1 & 1 \\ 0 & D1 & 1 \end{bmatrix}$$

```
> SyzygyModule(R, Alg);
```

$$\begin{bmatrix} D2 & D1 & -D1 \end{bmatrix}$$

```
> R := matrix([[-D2, D1, 1], [0, D1, 1]]);
```

$$R := \begin{bmatrix} -D2 & D1 & 1 \\ 0 & D1 & 1 \end{bmatrix}$$

```
> SyzygyModule(R, Alg);
```

$$\mathrm{INJ}(2)$$

```
> SyzygyModule(INJ(3), Alg);
```

$$ZERO$$

```
> SyzygyModule(SURJ(2), Alg);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> R := matrix([[x1], [D1]]);
```

$$R := \begin{bmatrix} x1 \\ D1 \end{bmatrix}$$

```
> SyzygyModule(R, Alg);
```

$$\begin{bmatrix} x1\, D1 - 1 & -x1^2 \\ D1^2 & -2 - x1\, D1 \end{bmatrix}$$

```
> SyzygyModuleRat(R, Alg);
```

$$[x1\, D1 - 1 \quad -x1^2]$$

See Also:

DefineOreAlgebra, Resolution, FreeResolution, ShorterFreeResolution, MinimalResolution, ProjectiveDimension, Exti, Extn, Torsion, Parametrization, MinimalParametrization, Quotient, Factorize, Elimination, Integrability, Involution, ReduceMatrix.

OreModules[TorsionElements],

OreModules[TorsionElementsRat] - return generating set of torsion elements in terms of the system variables

Calling Sequence:

TorsionElements(R,v,Alg)
TorsionElementsRat(R,v,Alg)

Parameters:

`R`	-	matrix with entries in **`Alg`**
`v`	-	list or vector of functions
`Alg`	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***TorsionElements*** returns a generating set of torsion elements of the left module over **`Alg`** which is presented by **`R`** and a generating set of autonomous equations that these torsion elements satisfy. The torsion elements are expressed in terms of the system variables given by **`v`**.
- **`R`** is a matrix with entries in the Ore algebra **`Alg`**.
- **`v`** is a list or vector of functions which depend on the independent variable of the ODE system. These functions are interpreted as the system variables.
- **`Alg`** is expected to be defined using DefineOreAlgebra.
- The result of ***TorsionElements*** is the empty list if no torsion elements exist in the left module over **`Alg`** which is presented by **`R`**, or a list containing two vectors otherwise.
- If the result is a list of two vectors, then the first contains a generating set of autonomous equations that the torsion elements given by the second vector satisfy. The second vector gives a generating set of torsion elements θ_i in terms of the system variables given by **`v`**.
- ***TorsionElementsRat*** performs the same computations as ***TorsionElements***, but the domain of coefficients of the Ore algebra **`Alg`** is replaced by its quotient field, i.e. rational functions.
- For linear systems of ODEs or PDEs, the command AutonomousElements returns the generating torsion elements as integrated autonomous elements. This integration can also be achieved by using IntTorsion.

Examples:

```
> with(OreModules):
```

Example 1: Ordinary differential equations

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l1,l2]):
> R1 := evalm([[D^2+g/l1, 0, -g/l1], [0, D^2+g/l2, -g/l2]]);
```

$$R1 := \begin{bmatrix} D^2 + \frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2 + \frac{g}{l2} & -\frac{g}{l2} \end{bmatrix}$$

```
> TorsionElements(R1, [x1(t),x2(t),u(t)], Alg);
```

[ ]

There are no torsion elements, i.e. no autonomous elements of the systems, which means that, generically, the bipendulum is controllable. However, if the lengths of the two pendula are equal, there are autonomous elements:

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l]):
```

```
> R2 := evalm([[D^2+g/l, 0, -g/l], [0, D^2+g/l, -g/l]]);
```

$$R2 := \begin{bmatrix} D^2 + \frac{g}{l} & 0 & -\frac{g}{l} \\ 0 & D^2 + \frac{g}{l} & -\frac{g}{l} \end{bmatrix}$$

```
> TorsionElements(R2, [x1(t),x2(t),u(t)], Alg);
```

$$\left[\left[l\left(\frac{d^2}{dt^2}\theta_1(t)\right) + g\,\theta_1(t) = 0\right], [\theta_1(t) = x1(t) - x2(t)]\right]$$

Example 2: Differential time-delay systems

Linear differential time-delay system describing a flexible rod (see H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD thesis, University of Orsay, France, 1995):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  shift_action=[delta,t,h]):
> R := evalm([[Dt, -Dt*delta, -1], [2*Dt*delta, -Dt-Dt*delta^2, 0]]);
```

$$R := \begin{bmatrix} Dt & -Dt\,\delta & -1 \\ 2\,Dt\,\delta & -Dt - Dt\,\delta^2 & 0 \end{bmatrix}$$

```
> TorsionElements(R, [y1(t),y2(t),u(t)], Alg);
```

$$[[D(\theta_1)(t) = 0], [\theta_1(t) = -2\,y1(t-h) + y2(t) + y2(t-2\,h)]]$$

Example 3: Partial differential equations

Linear system of partial differential equations that appears in mathematical physics, namely in the study of Lie-Poisson structures. (See C. M. Bender, G. V. Dunne, L. R. Mead, *Underdetermined systems of partial differential equations*, Journal of Mathematical Physics, vol. 41, no. 9 (2000), 6388-6398 and
W. M. Seiler, *Involution analysis of the partial differential equations characterising Hamiltonian vector fields*, Journal of Mathematical Physics, vol. 44 (2003), 1173-1182.)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[x1*D3, x2*D3, 0], [-x1*D2+x2*D1, -1, x2*D3], [-1, -x2*D1+x1*D2, x1*D3]]);
```

$$R := \begin{bmatrix} x1\,D3 & x2\,D3 & 0 \\ -x1\,D2 + x2\,D1 & -1 & x2\,D3 \\ -1 & -x2\,D1 + x1\,D2 & x1\,D3 \end{bmatrix}$$

```
> TorsionElements(R, [F(x1,x2,x3), G(x1,x2,x3), H(x1,x2,x3)], Alg);
```

$$\left[\left[\begin{array}{c} \frac{\partial}{\partial x3}\theta_1(x1,x2,x3)=0 \\ x1\left(\frac{\partial}{\partial x2}\theta_1(x1,x2,x3)\right)-x2\left(\frac{\partial}{\partial x1}\theta_1(x1,x2,x3)\right)=0 \\ x2\left(\frac{\partial}{\partial x3}\theta_2(x1,x2,x3)\right)=0 \\ x1\left(\frac{\partial}{\partial x3}\theta_2(x1,x2,x3)\right)=0 \\ x1\left(\frac{\partial}{\partial x2}\theta_2(x1,x2,x3)\right)-x2\left(\frac{\partial}{\partial x1}\theta_2(x1,x2,x3)\right)=0 \end{array}\right], \left[\begin{array}{c} \theta_1(x1,x2,x3)=x1\,F(x1,x2,x3)+x2\,G(x1,x2,x3) \\ \theta_2(x1,x2,x3)=\left(\frac{\partial}{\partial x1}F(x1,x2,x3)\right)+\left(\frac{\partial}{\partial x2}G(x1,x2,x3)\right)+\left(\frac{\partial}{\partial x3}H(x1,x2,x3)\right) \end{array}\right]\right]$$

```
> TorsionElementsRat(R, [F(x1,x2,x3), G(x1,x2,x3), H(x1,x2,x3)], Alg);
```

$$\left[\left[\begin{array}{c} \frac{\partial}{\partial x3}\theta_1(x1,x2,x3)=0 \\ -x1\left(\frac{\partial}{\partial x2}\theta_1(x1,x2,x3)\right)+x2\left(\frac{\partial}{\partial x1}\theta_1(x1,x2,x3)\right)=0 \\ \frac{\partial}{\partial x3}\theta_2(x1,x2,x3)=0 \\ -x1\left(\frac{\partial}{\partial x2}\theta_2(x1,x2,x3)\right)+x2\left(\frac{\partial}{\partial x1}\theta_2(x1,x2,x3)\right)=0 \end{array}\right], \left[\begin{array}{c} \theta_1(x1,x2,x3)=x1\,F(x1,x2,x3)+x2\,G(x1,x2,x3) \\ \theta_2(x1,x2,x3)=-x2\,G(x1,x2,x3)-x1^2\left(\frac{\partial}{\partial x2}G(x1,x2,x3)\right)+x1\,x2\left(\frac{\partial}{\partial x1}G(x1,x2,x3)\right)-x1^2\left(\frac{\partial}{\partial x3}H(x1,x2,x3)\right) \end{array}\right]\right]$$

See Also:

DefineOreAlgebra, Parametrization, MinimalParametrization, AutonomousElements, IntTorsion, Exti, Extn, Torsion, PiPolynomial.

OreModules[Torsion],

OreModules[TorsionRat] - return generating set for torsion submodule and annihilators of torsion elements

Calling Sequence:

Torsion(R,Alg)
TorsionRat(R,Alg)

Parameters:

R	-	matrix with entries in **Alg** or INJ(*n*) or SURJ(*n*), where *n* is a non-negative integer
Alg	-	Ore algebra (given by DefineOreAlgebra)

Description:

- ***Torsion*** computes the first extension module with values in **Alg** of the left **Alg**-module presented by **R**. It returns the same result as ***Exti*** applied to **R** for $i = 1$.
- **R** is a matrix with entries in the Ore algebra **Alg**.
- **Alg** is expected to be defined using DefineOreAlgebra.
- For a general description of the result of ***Torsion*** in terms of the computation of the first extension module, see Exti. If **R** is the result of Involution applied to some matrix *R1* with entries in **Alg**, then the second matrix of the result is a presentation of $M / t(M)$, where M is the left **Alg**-module presented by *R1* and $t(M)$ is its torsion submodule. The first matrix of the result gives the annihilators of the residue classes of the rows of the second matrix in *M*. Hence, non-zero torsion elements correspond to columns of the first matrix which generate a proper left ideal of **Alg**.
- ***TorsionRat*** performs the same computations as ***Torsion***, but the domain of coefficients of the Ore algebra **Alg** is replaced by its quotient field, i.e. rational functions.
- The same information can be obtained using Exti, but Exti also computes higher extension modules (cf. also Extn).

Examples:

```
> with(OreModules):
```

Example 1: Ordinary differential equations

System of linear ordinary differential equations describing a bipendulum (J.-F. Pommaret, *Partial Differential Control Theory*, 2001):

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l1,l2]):
> R1 := evalm([[D^2+g/l1, 0, -g/l1], [0, D^2+g/l2, -g/l2]]);
```

$$R1 := \begin{bmatrix} D^2 + \frac{g}{l1} & 0 & -\frac{g}{l1} \\ 0 & D^2 + \frac{g}{l2} & -\frac{g}{l2} \end{bmatrix}$$

```
> Torsion(Involution(R1, Alg), Alg);
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} D^2\, l1 + g & 0 & -g \\ 0 & D^2\, l2 + g & -g \end{bmatrix}, \begin{bmatrix} D^2\, l2\, g + g^2 \\ D^2\, l1\, g + g^2 \\ D^4\, l2\, l1 + D^2\, l2\, g + D^2\, l1\, g + g^2 \end{bmatrix} \right]$$

Since the first matrix is an identity matrix, there are no torsion elements, i.e., there are no autonomous elements of the systems, which means that, generically, the bipendulum is controllable. However, if the lengths of the two pendula are equal, there are autonomous elements:

```
> Alg := DefineOreAlgebra(diff=[D,t], polynom=[t], comm=[g,l]):
```

```
> R2 := evalm([[D^2+g/l, 0, -g/l], [0, D^2+g/l, -g/l]]);
```

$$R2 := \begin{bmatrix} D^2+\frac{g}{l} & 0 & -\frac{g}{l} \\ 0 & D^2+\frac{g}{l} & -\frac{g}{l} \end{bmatrix}$$

```
> Torsion(Involution(R2, Alg), Alg);
```

$$\left[\begin{bmatrix} D^2\,l+g & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 & 0 \\ 0 & D^2\,l+g & -g \end{bmatrix}, \begin{bmatrix} g \\ g \\ D^2\,l+g \end{bmatrix}\right]$$

The residue class r of the first row of the second matrix generates the torsion submodule of the **`Alg`**-module M presented by *R2*. This residue class r satisfies $(l\,D^2+g)\,r=0$ in M.

Example 2: Differential time-delay systems

Linear differential time-delay system describing a flexible rod (see H. Mounier, *Proprietes structurelles des systemes lineaires a retards: aspects theoriques et pratiques*, PhD thesis, University of Orsay, France, 1995):

```
> Alg := DefineOreAlgebra(diff=[Dt,t], dual_shift=[delta,s], polynom=[t,s],
  shift_action=[delta,t,h]):
> R := evalm([[Dt, -Dt*delta, -1], [2*Dt*delta, -Dt-Dt*delta^2, 0]]);
```

$$R := \begin{bmatrix} Dt & -Dt\,\delta & -1 \\ 2\,Dt\,\delta & -Dt-Dt\,\delta^2 & 0 \end{bmatrix}$$

```
> Torsion(Involution(R, Alg), Alg);
```

$$\left[\begin{bmatrix} Dt & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -2\,\delta & 1+\delta^2 & 0 \\ -Dt & Dt\,\delta & 1 \\ Dt\,\delta & -Dt & \delta \end{bmatrix}, \begin{bmatrix} 1+\delta^2 \\ 2\,\delta \\ Dt-Dt\,\delta^2 \end{bmatrix}\right]$$

The torsion submodule of the **`Alg`**-module M presented by **R** is generated by the residue class r of the first row of the second matrix. We have $Dt\,r=0$ in M.

Example 3: Partial differential equations

Linear system of partial differential equations that appears in mathematical physics, namely in the study of Lie-Poisson structures. (See C. M. Bender, G. V. Dunne, L. R. Mead, *Underdetermined systems of partial differential equations*, Journal of Mathematical Physics, vol. 41, no. 9 (2000), 6388-6398 and
W. M. Seiler, *Involution analysis of the partial differential equations characterising Hamiltonian vector fields*, Journal of Mathematical Physics, vol. 44 (2003), 1173-1182.)

```
> Alg := DefineOreAlgebra(diff=[D1,x1], diff=[D2,x2], diff=[D3,x3], polynom=[x1,x2,x3]):
> R := evalm([[x1*D3, x2*D3, 0], [-x1*D2+x2*D1, -1, x2*D3], [-1, -x2*D1+x1*D2, x1*D3]]);
```

$$R := \begin{bmatrix} x1\,D3 & x2\,D3 & 0 \\ -x1\,D2+x2\,D1 & -1 & x2\,D3 \\ -1 & -x2\,D1+x1\,D2 & x1\,D3 \end{bmatrix}$$

```
> Torsion(Involution(R, Alg), Alg);
```

$$\left[\begin{bmatrix} D3 & 0 & 0 \\ -x2\,D1 + x1\,D2 & 0 & 0 \\ 0 & x2\,D3 & 0 \\ 0 & x1\,D3 & 0 \\ 0 & -x2\,D1 + x1\,D2 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} x1 & x2 & 0 \\ D1 & D2 & D3 \\ -1 & -x2\,D1 + x1\,D2 & x1\,D3 \end{bmatrix}, \begin{bmatrix} -x2\,D3 \\ x1\,D3 \\ -x1\,D2 + x2\,D1 \end{bmatrix}\right]$$

Computation of the torsion submodule over the Weyl algebra with rational coefficients:

```
> TorsionRat(Involution(R, Alg), Alg);
```

$$\left[\begin{bmatrix} D3 & 0 \\ -x1\,D2 + x2\,D1 & 0 \\ 0 & D3 \\ 0 & -x1\,D2 + x2\,D1 \end{bmatrix}, \begin{bmatrix} x1 & x2 & 0 \\ 0 & x1\,x2\,D1 - x1^2\,D2 - x2 & -x1^2\,D3 \end{bmatrix}, \begin{bmatrix} x2\,D3 \\ -x1\,D3 \\ -x2\,D1 + x1\,D2 \end{bmatrix}\right]$$

See Also:

DefineOreAlgebra, Involution, SyzygyModule, Resolution, FreeResolution, ShorterFreeResolution, ShortestFreeResolution, ProjectiveDimension, Exti, Extn, Parametrization, MinimalParametrization, AutonomousElements, PiPolynomial, TorsionElements.