Introduction to the JanetOre package

Calling Sequence:

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

Description:

- The JanetOre package provides algorithms for the involutive analysis of left ideals in certain Ore algebras, which are non-commutative rings in general, and more generally for submodules of finitely generated free left modules over such algebras.
- The main algorithm is a version of the involutive basis algorithm by Gerdt and Blinkov adapted to certain Ore algebras. In the particular cases of commutative polynomial rings or the Weyl algebras it is a substantial improvement of Janet's algorithm for analysing systems of linear partial differential equations.
- The Ore algebra in which computations are performed by the procedures of JanetOre can theoretically be defined by an iterative construction of certain skew-polynomial rings from a commutative polynomial ring with (a field extension of) the rational numbers or a finite field as base field. However, for the JanetOre package the Ore algebra is defined by specifying the indeterminates of the commutative polynomial ring and all indeterminates for these Ore extensions in one list and by specifying a list of generating commutation rules for these indeterminates in a prescribed way. All indeterminates for the Ore extensions are assumed to commute with each other, and for each of these indeterminates at most one non-trivial commutation rule with an indeterminate of the commutative polynomial ring is accepted. In case no commutation rule is specified for such an indeterminate, this indeterminate commutes with all elements of the Ore algebra. The list of commutation rules may contain the following expressions (cf. also the examples below):
  - weyl(D, *x*), where *x* is an indeterminate of the commutative polynomial ring and D is an indeterminate for one of the Ore extensions. This defines the Ore extension with commutation rule D**x* = *x**D + 1 (Weyl algebra).
  - shift(δ, *t*), where *t* is an indeterminate of the commutative polynomial ring and δ is an indeterminate for one of the Ore extensions. This defines an Ore extension with commutation rule δ**t* = (*t*+1)*δ (algebra of shift operators).
- JanetOre represents elements of such an Ore algebra as polynomials in the indeterminates x_i of the commutative polynomial ring and the indeterminates D_j for the Ore extensions. It chooses as a vector space basis of the Ore algebra the set of terms which are formed as products of monomials in the x_i times monomials in the D_j, i.e. every element of the Ore algebras has a unique representation as polynomial where in each term a monomial in the x_i is multiplied from the left to a monomial in the D_j. All results of the JanetOre package are given in this representation. However, since the order in which Maple displays factors in a product is not fixed, it may be necessary to sort the variables in each term of such a polynomial according to the previous definition when reading the result. The Maple package Ore_algebra uses the same convention, see e. g. Ore_algebra[diff_algebra].
- The main algorithm for this package, called JBasis, produces standard generators for submodules of free left modules over Ore algebras as described above, which are given by any finite set of generators. These can be used as input for various other commands, e. g., to give quantitative information about the residue class module. They are also used in the command JInvReduce to produce a normal form for representatives of residue classes.
- The base field is defined to be (a field extension of) the rational numbers by default. By means of the command JanetOreOptions it can be changed to integer coefficients or to (an extension of) a field of non-zero characteristic.
- Involutive bases form a special kind of Groebner bases, which are provided e. g. by the Groebner package. The main difference is that involutive division provides a different strategy to obtain deductions and reduce polynomials. The rules, which element of the involutive basis has to be applied first, when performing reduction, are rather strict and governed by the concept of multiplicative and nonmultiplicative variables, i.e. variables which are allowed resp. not allowed as quotients for involutive divisions by an element of the involutive basis. For details see the references below and the explanations in JTabVar.
- To use a function of the JanetOre package, either define that function alone using the command with(JanetOre, <function>), or define all JanetOre functions using the command with(JanetOre). Alternatively, invoke the function using the long form JanetOre[<function>].
- The functions available in the JanetOre package are the following:

Basic commands:

JBasis	JBasisFast
JInvReduce	JInvReduceFast
JTabVar	JHilbertSeries
JFactorModuleBasis	JSubmoduleBasis

Further commands for the computation of involutive bases:

JAddRhs	AssertJBasis
JanetOreOptions	JZeroSets

Commands for special applications:

JMinPoly	JSyzygies
JSyzygyModule	JSyzygyModuleFast
JResolution	JResolutionDim
JEulerChar	JRepres
JAnnihilator	JCoeff
JCoeffList	JWeightedHilbertSeries
JLeftInverse	JRightInverse
JFactorize	JSyzOp
JNotHas/JHas	

Commands for module theory:

JSum	JDirectSum
JIntersection	JSubFactor
JCheckHom	JDefect
JHom	JHomHom
JKernel	JCokernel
JExt1	JExtn
JParametrization	JTorsion

Commands for various invariants derivable from JHilbertSeries:

JIndexRegularity	JDimension
JHilbertPolynomial	JHilbertFunction
JHP	JHF
JMultiplicity	JCartanCharacter
JSubmoduleDimension	JSubmoduleHF
JSubmoduleHP	JSubmoduleHilbertFunction
JSubmoduleHilbertPolynomial	JSubmoduleHilbertSeries
JSubmoduleMultiplicity	

Alternate Groebner basis commands:

JGroebnerBasis	JGroebnerBasisFast

Auxiliary commands:

JLeadingMonomial	JanetOreStats

- For a description of the basic algorithms in the case of commutative polynomial rings, see V. P. Gerdt, "Involutive Algorithms for Computing Groebner Bases", in: S. Cojocaru, G. Pfister, V. Ufnarovski, "Computational Commutative and Non-Commutative Algebraic Geometry", NATO Science Series, IOS Press, 2005, pp. 199-225; or V. P. Gerdt, "Involutive Division Technique: Some Generalizations and Optimizations", Journal of Mathematical Sciences 108(6), 2002, pp. 1034-1051.
- For a description of the modifications of the algorithms towards the non-commutative version for Ore algebras, see D. Robertz, "Formal Computational Methods for Control Theory", PhD thesis, RWTH Aachen, Germany, 2006, or D. Robertz, "Janet Bases and Applications", in: M. Rosenkranz, D. Wang (eds.), "Groebner Bases in Symbolic Analysis", Radon Series on Computational and Applied Mathematics 2, de Gruyter, 2007, pp. 139-168.
- For more details about Ore algebras and the description of algorithms computing Groebner bases for left ideals in Ore algebras, see F. Chyzak, "Fonctions holonomes en calcul formel", PhD thesis, Ecole Polytechnique, France, 1998; F. Chyzak, B. Salvy, "Non-commutative elimination in Ore algebras proves multivariate identities", J. Symbolic Computation, 26, 1998, 187-227.
- For a description of the packages Involutive and Janet, see Y. A. Blinkov, C. F. Cid, V. P. Gerdt, W. Plesken, D. Robertz, "The MAPLE package 'Janet': I. Polynomial Systems, II. Linear Partial Differential Equations", in: V. G. Ganzha, E. W. Mayr, E. V. Vorozhtsov (eds.), Proceedings of Computer Algebra in Scientific Computing CASC 2003, Passau, pp. 31-40 resp. 41-54.
- For a more general description of Janet's philosophy, see W. Plesken, D. Robertz, "Janet's approach to presentations and resolutions

for polynomials and linear pdes", Archiv der Mathematik, 84(1), 2005, pp. 22-37.

Examples:

```
> with(JanetOre):
```

Example 1: Weyl algebra

We are going to consider the Weyl algebra Q[x][D] of polynomial differential operators w.r.t. x with coefficients which are polynomials in x with rational coefficients. Therefore, we define the list of variables consisting of x and D. (Note that the following list also specifies that x > D for the monomial ordering which is used in the following Janet basis computation.)

```
> var := [x,D];
```

$$var := [x, \mathrm{D}]$$

By using the following list we specify that the Ore extension of Q[x] by D is defined with commutation rule D*x = x*D+1:

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x)]$$

This commutation rule can be checked by computing the product D*x:

```
> JMult(D, x, var, ops);
```

$$x\,\mathrm{D} + 1$$

We are going to compute a Janet basis for the left ideal I in Q[x][D] which is generated by the following elements of Q[x][D]:

```
> L := [2*x*D-2+x*D^2, D^2];
```

$$L := [2\,x\,\mathrm{D} - 2 + x\,\mathrm{D}^2, \mathrm{D}^2]$$

The Janet basis with respect to the degree-reverse lexicographical ordering of the left ideal I generated by L:

```
> J := JBasis(L, var, ops);
```

$$J := [\mathrm{D}^2, x\,\mathrm{D} - 1]$$

JTabVar displays the internal data structure which was created by ***JBasis***, in particular containing the list of multiplicative and non-multiplicative variables:

```
> JTabVar();
```

$$[\mathrm{D}^2, [*, \mathrm{D}], \mathrm{D}^2]$$
$$[x\,\mathrm{D} - 1, [x, \mathrm{D}], x\,\mathrm{D}]$$

Enumerate a vector space basis of the factor module Q[x][D] / I:

```
> JFactorModuleBasis(var);
```

$$1 + \frac{x}{1-x} + \mathrm{D}$$

Compute the Hilbert series of the factor module Q[x][D] / I:

```
> JHilbertSeries();
```

$$1 + 2\,s + \frac{s^2}{1-s}$$

Find the normal form of the residue class in Q[x][D] / I which contains D^2+D+1:

```
> JInvReduce(D^2+D+1, J, var, ops);
```

$$\mathrm{D} + 1$$

```
> JInvReduce(D^4, J, var, ops);
```

$$0$$

Example 2: Ore algebra of shift operators

In this example we compute over the Ore algebra Q[t][δ] of polynomials in the operator δ with coefficients which are polynomials in t with rational coefficients.

```
> var := [t,delta];
```

$$var := [t, \delta]$$

By using the following list we specify that the Ore extension of Q[t] by δ is defined with commutation rule δ*t = (t+1)*δ:

```
> ops := [shift(delta,t)];
```

$$ops := [\mathrm{shift}(\delta, t)]$$

The commutation rule can be checked as follows:

```
> JMult(delta, t, var, ops);
```

$$(t+1)\,\delta$$

We are going to compute a Janet basis for the left ideal I in Q[t][δ] which is generated by the following elements of Q[t][δ]:

```
> L := [t^2*delta, delta-t];
```

$$L := [t^2\,\delta, \delta - t]$$

The Janet basis with respect to the degree-reverse lexicographical ordering of the left ideal I generated by L:

```
> J := JBasis(L, var, ops);
```

$$J := [\delta, t]$$

```
> JTabVar();
```

$$[\delta, [*, \delta], \delta]$$

$$[t, [t, \delta], t]$$

```
> JFactorModuleBasis(var);
```

$$[1]$$

See Also:

with, Involutive, Janet.

JanetOre[JanetOreOptions] - set up the options for the current session of JanetOre

Calling Sequence:

JanetOreOptions(s,v)

Parameters:

s - string specifying the option to be affected
v - (optional) the option's new value

Description:

- ***JanetOreOptions*** sets up the options for the current session of JanetOre. The string **s** specifies the option which is to be modified or whose value is to be returned. Possible values for **s** are the following:

```
"char"              "rational"          "AbsolutelySmallestRemainder"
"matrix"            "JanetLike"         "criteria"
"C++"               "Maple"
"InvBasis"          "InvReduce"         "SyzygyModule"
"GBasis"
```

- If no second parameter is provided, then ***JanetOreOptions*** returns the current value of the option specified by the first parameter.
- If the first parameter **s** is the string "char", then a second parameter **v** is expected to be zero or a prime number. Subsequent computations of the ***JanetOre*** package are done in characteristic **v** then. The return value of ***JanetOreOptions*** is the characteristic of the ground field used by the ***JanetOre*** package so far.
- If **s** equals the string "rational", then **v** is expected to be either true or false. The default setting is true which means that involutive bases are computed over (an extension field of) the rational numbers or fields of non-zero characteristic, if the option "char" was modified. If the option "rational" is set to false, then involutive bases are computed over the integers. Note that in this case the value of the option "char" is automatically set to zero and that "rational" is set to true again if a non-zero value is assigned to the option "char" afterwards. The return value of ***JanetOreOptions*** in this case is the former value of the option "rational".
- If **s** equals the string "AbsolutelySmallestRemainder", then **v** is expected to be either true or false. This option is relevant only if the option "rational" has been set to false, i.e. if involutive bases are computed over the integers. In that case the option "AbsolutelySmallestRemainder" determines how ambiguity of normal forms is resolved: If **v** equals false (which is the default), then a term which is reduced modulo another polynomial with positive leading coefficient c, will have a coefficient between 0 and c-1. If **v** equals true, then the resulting coefficient will be between floor($-c/2$)+1 and floor($c/2$), i.e. it will be an absolutely smallest remainder modulo c. The return value of ***JanetOreOptions*** in this case is the former value of the option "AbsolutelySmallestRemainder".
- The keywords "C++" and "Maple" select the method for subsequent computations of involutive bases, involutive reductions, syzygy modules and Groebner bases. The default setting is "Maple". In this case all computations are done using procedures written in Maple. If "C++" methods are chosen, then the command JBasis becomes a synonym for JBasisFast, JInvReduce a synonym for JInvReduceFast, JSyzygyModule a synonym for JSyzygyModuleFast, and JGroebnerBasis a synonym for JGroebnerBasisFast, i.e. all these basic commands invoke the external C++ routines. Note that in this case ***JBasis*** and ***JSyzygyModule*** call AssertJBasis with the output of the C++ routine as parameter which sets up the internal data for the current ***JanetOre*** session. Hence, from the user's point of view, there is no difference in using the "Maple" or the "C++" methods of ***JBasis***, ***JInvReduce***, ***JSyzygyModule***, and ***JGroebnerBasis*** when selecting the methods by means of ***JanetOreOptions***. Note also that this selection affects many commands of the JanetOre package which call these basic procedures (e. g. JResolution) and that the "C++" methods are much faster than the "Maple" routines for big problems. There is no return value of ***JanetOreOptions*** if **s** is either "C++" or "Maple".
- If the parameter **s** is the string "matrix", then **v** is expected to be either the symbol 'matrix' or the symbol 'Matrix'. This option determines the type for matrices returned by the procedures of the JanetOre package (e.g. JResolution, JRepres are affected by this option; however, the result of JLeftInverse, JRightInverse and JCoeff is of the same type as their input). This option is meaningful only for Maple versions that provide both the linalg and the LinearAlgebra package. The default value for this option is the symbol

'Matrix'.

- If **s** equals the string "JanetLike", then **v** is expected to be either true or false. If **v** is true then the command JBasis returns a Janet-like Groebner basis instead of an involutive basis. The default setting is false. The return value of ***JanetOreOptions*** is the former value of the option "JanetLike".
- If **s** equals the string "criteria", then **v** is expected to be a list consisting of some of the integers 1, 2, 3, 4 (or being the empty list). If the integer i is present in v, then involutive basis computations over commutative polynomial rings during the current session of Involutive will apply the i-th involutive criterion to avoid unnecessary reductions. For more details about the involutive criteria, see V. P. Gerdt, D. A. Yanovich, "Experimental Analysis of Involutive Criteria", in: A. Dolzmann, A. Seidl, T. Sturm (eds.), Algorithmic Algebra and Logic, BOD Norderstedt, pp. 105-109.
- By means of the keywords "InvBasis", "InvReduce", "SyzygyModule", and "GBasis" the methods for subsequent computations of involutive bases, involutive reductions, syzygy modules, and Groebner bases are chosen. This is a more advanced way of setting the options described above for "C++" and "Maple". More precisely, if **s** is the string "InvBasis" then **v** is expected to be a procedure. Then every subsequent call of JBasis invokes **v** instead of the default method for involutive basis computation. Similarly, if **s** is "InvReduce" (resp. "SyzygyModule" resp. "GBasis") then **v** is also expected to be of type procedure and every call of JInvReduce (resp. JSyzygyModule resp. JGroebnerBasis) invokes **v** instead of the default method. In each case the return value of ***JanetOreOptions*** is the procedure used so far by the JanetOre package for the respective purpose.

Examples:

```
> with(JanetOre):
```

Example 1: Changing the characteristic of the ground field

```
> var := [D,t];
```

$$var := [D, t]$$

```
> ops := [weyl(D,t)];
```

$$ops := [\mathrm{weyl}(D, t)]$$

```
> L := [t*D-2, D^3];
```

$$L := [t\,D - 2, D^3]$$

```
> JBasis(L, var, ops);
```

$$[t\,D - 2, t\,D^2 - D, D^3]$$

```
> JanetOreOptions("char", 2);
```

$$0$$

```
> JBasis(L, var, ops);
```

$$[t\,D, t\,D^2 + D, D^3]$$

```
> JanetOreOptions("char", 0);
```

$$2$$

```
> JBasis(L, var, ops);
```

$$[t\,D - 2, t\,D^2 - D, D^3]$$

```
> JanetOreOptions("char");
```

$$0$$

Example 2: Computing involutive bases over the integers

```
> L := [3*x, x^2-x];
```

$$L := [3\,x, x^2 - x]$$

```
> JanetOreOptions("rational", false);
```

$$true$$

```
> JBasis(L, [x]);
```

$$[3\,x, x^2 + 2\,x]$$

```
> JTabVar();
```

$$[3\,x, [*], 3\,x]$$
$$[x^2 + 2\,x, [x], x^2]$$

```
> JanetOreOptions("AbsolutelySmallestRemainder", true);
```

$$false$$

```
> JBasis(L, [x]);
```

$$[3x, x^2 - x]$$

```
> JTabVar();
```

$$[3x, [*], 3x]$$

$$[x^2 - x, [x], x^2]$$

```
> JanetOreOptions("rational", true);
```

$$false$$

```
> JBasis(L, [x]);
```

$$[x]$$

Example 3: Selecting fast involutive basis computation method

```
> L := [seq(a[i]^3-a[i+1]-1, i=1..6),seq(a[i]^2-a[i-1]+1, i=2..3)];
```

$$L := [a_1^{\,3} - a_2 - 1, a_2^{\,3} - a_3 - 1, a_3^{\,3} - a_4 - 1, a_4^{\,3} - a_5 - 1, a_5^{\,3} - a_6 - 1, a_6^{\,3} - a_7 - 1, a_2^{\,2} - a_1 + 1, a_3^{\,2} - a_2 + 1]$$

```
> JanetOreOptions("C++");
> JBasis(L, [seq(a[i], i=1..7)]);
```

$$[1]$$

```
> JanetOreOptions("Maple");
> JBasis(L, [seq(a[i], i=1..7)]);
```

$$[1]$$

Example 4: Changing the matrix type of results of procedures

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [x+y, x*y];
```

$$L := [x + y, xy]$$

```
> JResolution(L, var);
```

$$\left[\begin{bmatrix} -y^2 & x+y \end{bmatrix}, \begin{bmatrix} x+y \\ y^2 \end{bmatrix}\right]$$

```
> whattype(%[1]);
```

$$array$$

```
> InvolutiveOptions("matrix", Matrix);
```

$$matrix$$

```
> PolResolution(L, var);
```

$$\left[\begin{bmatrix} -y^2 & x+y \end{bmatrix}, \begin{bmatrix} x+y \\ y^2 \end{bmatrix}\right]$$

```
> whattype(%[1]);
```

$$Matrix$$

```
> InvolutiveOptions("matrix", matrix);
```

$$Matrix$$

Example 5: Computing Janet-like Groebner bases

(see V. P. Gerdt, Y. A. Blinkov, *Janet-like Groebner Bases*, Proceedings of Computer Algebra in Scientific Computing, Springer, 2005, pp. 184-195)

```
> L := [x^7-y^2*z, x^4*w-y^3, x^3*y-z*w];
```

$$L := [x^7 - y^2 z, x^4 w - y^3, x^3 y - z w]$$

```
> JanetOreOptions("JanetLike", true);
```

$$false$$

```
> JBasis(L, [x,y,z,w]);
```

$$[-z w^2 x + y^4, x^3 y - z w, x^4 w - y^3, y x^4 - z w x, x^7 - y^2 z]$$

```
> JanetOreOptions("JanetLike", false);
```

$$true$$

```
> JBasis(L, [x,y,z,w]);
```

$$[-z w^2 x + y^4, x^3 y - z w, x^4 w - y^3, -z w^2 x^2 + y^4 x, y x^4 - z w x, w x^5 - y^3 x, -z w^2 x^3 + y^4 x^2, y x^5 - z w x^2, w x^6 - y^3 x^2, y x^6 - z w x^3,$$

$x^7 - y^2 z]$

```
> JGroebnerBasis(L, [x,y,z,w]);
```

$$[-z w^2 x + y^4, x^3 y - z w, x^4 w - y^3, x^7 - y^2 z]$$

See Also:

JBasis, JanetOreStats, JBasisFast, JInvReduce, JInvReduceFast, JTabVar, JHilbertSeries, JSyzygyModule, JSyzygyModuleFast, JGroebnerBasis, JBasisFast.

JanetOre[JanetOreStats] - display statistics of last application of JBasis

Calling Sequence:

JanetOreStats()

Parameters:

- - none (assumes that JBasis has been called before)

Description:

- ***JanetOreStats*** displays statistical information about the last run of ***JBasis***.

Examples:

```
> with(JanetOre):
> L := [x1+x2+x3+x4, x1*x2+x2*x3+x3*x4+x4*x1, x1*x2*x3+x2*x3*x4+x3*x4*x1+x4*x1*x2,
  x1*x2*x3*x4-1];
```

$$L := [x1 + x2 + x3 + x4, x1\,x2 + x2\,x3 + x3\,x4 + x4\,x1, x1\,x2\,x3 + x2\,x3\,x4 + x3\,x4\,x1 + x4\,x1\,x2, x1\,x2\,x3\,x4 - 1]$$

```
> JBasis(L, [x1,x2,x3,x4]);
```

$$[x1 + x2 + x3 + x4, x2^2 + 2\,x4\,x2 + x4^2, x2\,x3^2 + x3^2\,x4 - x2\,x4^2 - x4^3, x3^2\,x4^2 - x4^3\,x2 - x4^4 + x2\,x3\,x4^2 + x3\,x4^3 - 1,$$
$$x4^4\,x2 + x4^5 - x4 - x2, x4^2\,x3^3 + x4^3\,x3^2 - x3 - x4, x4^4\,x3^2 + x2\,x3 - x4\,x2 + x3\,x4 - 2\,x4^2]$$

```
> JanetOreStats();
```

Number of polynomials in involutive basis , 7

Use of normal form procedure, 41

Number of reductions performed, 94

Number of transfers, 0

Use of first criterion, 0

Use of second criterion, 0

Use of third criterion, 0

Use of fourth criterion, 0

The involutive basis is also a reduced Groebner basis.

See Also:

JBasis, JanetOreOptions, JTabVar

JanetOre[AssertJBasis] - assure the system that given (lists of) polynomials form a Janet basis

Calling Sequence:

AssertJBasis(L,var,ops,ord,mode)

Parameters:

`L`	-	list (or matrix) of generators of the submodule
`var`	-	list of variables (of the Ore algebra)
`ops`	-	(optional) list of commutation rules for the variables of the Ore algebra
`ord`	-	(optional) change of monomial ordering (see below)
`mode`	-	(optional) string specifying options for the computation

Description:

- The internal result of the command ***AssertJBasis*** is that the data structure for the current involutive basis is set up such that commands like JHilbertSeries, JFactorModuleBasis, JTabVar, etc. can be invoked.
- All parameters to ***AssertJBasis*** have the same meaning as in JBasis.
- ***AssertJBasis*** returns the list **L**.
- One typical situation where ***AssertJBasis*** is used is to make an earlier computed involutive basis after at least one further call of ***JBasis*** again to the current involutive basis without recomputing it.
- Another, usually more important, use for ***AssertJBasis*** is for big polynomial systems. In this case one can use JBasisFast and apply ***AssertJBasis*** to the result defining it as the current involutive basis in Maple.

Examples:

```
> with(JanetOre):
```

Example 1: Working with two Janet bases

```
> var := [D1,D2,x1,x2];
```

$$var := [D1, D2, x1, x2]$$

```
> ops := [weyl(D1,x1),weyl(D2,x2)];
```

$$ops := [\mathrm{weyl}(D1, x1), \mathrm{weyl}(D2, x2)]$$

```
> L1 := [D1*D2-x1^2, D2^2-x1*x2];
```

$$L1 := [D1\,D2 - x1^2, D2^2 - x1\,x2]$$

```
> J1 := JBasis(L1, var, ops);
```

$$J1 := [x1, D2]$$

```
> JHilbertSeries(t);
```

$$1 + t\left(\frac{1}{1-t} + \frac{1}{(1-t)^2}\right)$$

```
> L2 := [[D1,-D2], [D2,-D1]];
```

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

```
> J2 := JBasis(L2, var, ops);
```

$$J2 := [[-D2, D1], [D1, -D2]]$$

```
> JHilbertSeries(t);
```

$$2 + t\left(2\frac{1}{1-t} + 2\frac{1}{(1-t)^2} + 2\frac{1}{(1-t)^3}\right)$$

```
> AssertJBasis(J1, var, ops);
```

$$[x1, D2]$$

```
> JHilbertSeries(t);
```

$$1 + t\left(\frac{1}{1-t} + \frac{1}{(1-t)^2}\right)$$

Example 2: Setting up the internal data structure after the use of ***JBasisFast***

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [[x^2, -x*y], [y^3, x^2], [x*y*z, x*y^2]];
```

$$L := [[x^2, -xy], [y^3, x^2], [xyz, xy^2]]$$

```
> IB := JBasisFast(L, var);
```

$$IB := [[x^2, -xy], [y^3, x^2], [xyz, xy^2], [xy^3, x^3], [-xyz^2, x^2 y^2], [0, x^4 + x^3 z], [xyz^3, x^3 y^2]]$$

```
> AssertJBasis(IB, var):
> JFactorModuleBasis(var);
```

$$\left[\frac{y^2}{1-z}+\frac{y}{1-z}+\frac{1}{1-z}+\frac{xy^2}{1-z}+\frac{xy}{1-z}+\frac{x}{1-z}, \frac{xy}{1-z}+\frac{x}{1-z}+\frac{x^2 y}{1-z}+\frac{x^2}{1-z}+\frac{x^3 y}{1-z}+\frac{x^3}{1-z}+\frac{1}{(1-y)(1-z)}\right]$$

```
> JHilbertSeries(t);
```

$$2+6t+11t^2+15t^3+17t^4+t^5\left(17\frac{1}{1-t}+\frac{1}{(1-t)^2}\right)$$

See Also:

JBasis, JTabVar, JanetOreOptions, JInvReduce, JFactorModuleBasis, JHilbertSeries, JWeightedHilbertSeries, JSyzygies, JSyzygyModule, JCartanCharacter.

JanetOre[JAddRhs] - add unit vectors as right hand sides to the entries of a list

Calling Sequence:

JAddRhs(L,R)

Parameters:

L	-	list (of arbitrary entries)
R	-	(optional) list (of arbitrary entries)

Description:

- ***JAddRhs*** substitutes each entry **m** of **L** by **m=e**, where **e** is the *i*-th unit row vector (i.e. a list) if **m** is at position *i* in the list **L**, and returns this new list.
- If the optional second parameter **R** is provided, then the right hand sides to be assigned to the entries of **L** are taken from **R**.
- If **L** is a matrix, then the method described above is applied to the list of rows of **L**.

Example:

```
> with(JanetOre):
> L := [x^2,x+y,z^3];
```

$$L := [x^2, x+y, z^3]$$

```
> JAddRhs(L);
```

$$[x^2 = [1, 0, 0], x+y = [0, 1, 0], z^3 = [0, 0, 1]]$$

```
> JAddRhs(L, [a,b,c]);
```

$$[x^2 = a, x+y = b, z^3 = c]$$

```
> M := matrix(map(i->[i], L));
```

$$M := \begin{bmatrix} x^2 \\ x+y \\ z^3 \end{bmatrix}$$

```
> JAddRhs(M);
```

$$[[x^2] = [1, 0, 0], [x+y] = [0, 1, 0], [z^3] = [0, 0, 1]]$$

See Also:

JBasis, JSyzygies.

JanetOre[JBasis] - return the (unique) minimal Janet basis of a submodule of a free left module over an Ore algebra

Calling Sequence:

JBasis(L,var,ops,ord,mode,opt)

Parameters:

`L`	-	list (or matrix) of generators of the submodule
`var`	-	list of variables (of the Ore algebra)
`ops`	-	(optional) list of commutation rules for the variables of the Ore algebra
`ord`	-	(optional) change of monomial ordering (see below)
`mode`	-	(optional) string specifying options for the computation
`opt`	-	(optional) equation specifying options for the computation

Description:

- ***JBasis*** returns the (unique) minimal Janet basis with respect to a certain ordering of the submodule of the free left module of *m*-tuples over the Ore algebra specified by **`var`** and **`ops`** which is given by the generators in **`L`**. The default ordering is degree reverse lexicographical. ***JBasis*** is the main function of the package JanetOre.
- The Ore algebra is defined over (a field extension of) the rational numbers by default. By means of the command JanetOreOptions it can be changed to integer coefficients or coefficients in (an extension of) a field of non-zero characteristic. If the domain of coefficients is a field, then the leading coefficients in the resulting Janet basis are normalized to 1.
- The entries of **`L`** are polynomials in case of a left ideal, i. e. a submodule of the free left module of rank one, or lists of polynomials of length *m*, representing elements of the free left module of *m*-tuples over the Ore algebra. If **`L`** is a matrix, then the generators are extracted from the rows of **`L`**.
- The parameter **`var`** is a list specifying the variables of the Ore algebra. If **`var`** is $[x_1, ..., x_n]$, then the ordering of the variables is defined to be $x_1 > x_2 > ... > x_n$. In the module case, the monomial ordering is extended to tuples giving higher priority to standard basis vectors whose non-zero component comes first. The sequence of priority can be changed by appending a permutation of the numbers 1 to *m* to the variables in **`var`** (cf. Example 6 below).
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- The output is a list containing the Janet basis for the submodule generated by **`L`** with respect to the chosen ordering.
- ***JBasis*** saves the information in an internal data structure which can be displayed by JTabVar.
- As optional fourth parameter natural numbers from 1 to 4 are accepted. If **`ord`** = 1, pure lexicographical ordering (elimination ordering) is applied. In case **`ord`** = 2 the degree reverse lexicographical ordering is chosen. In the module case these two possibilities assume "position over term" order, i.e. the leading term of an *m*-tuple is the leading term of the first non-zero entry. If one prefers to work with the "term over position" order, i.e. the leading term of an *m*-tuple is the greatest of the leading terms of the non-zero entries, then **`ord`** = 1 is replaced by **`ord`** = 3 and **`ord`** = 2 by **`ord`** = 4. The default is **`ord`** = 4. (Further modifications concerning degrees are described below. For examples that illustrate the dependence of the leading monomials of a polynomial on the choice of ordering see JLeadingMonomial.)
- In addition to the orderings described in the preceding paragraph, a block (elimination) ordering can be selected by partitioning the list of variables **`var`**. In this case the argument **`var`** is a list of lists ("blocks") of variables and **`ord`** is a list of natural numbers from 1 to 4 whose length equals the number of blocks. Two monomials are compared w. r. t. the block ordering as follows: The variables of the first block are examined first. If the according parts of the two monomials are different, the ordering specified by the first number in **`ord`** decides which monomial is greater. If the monomials are equal when considering only the first block of variables, then the ordering specified by the second number in **`ord`** is applied to the second block of variables, if the corresponding parts of the monomials are different, and so on (cf. Example 7 below). Note that in the module case, a "position over term" order and a "term over position" order cannot be mixed.
- The fifth argument **`mode`** is a string consisting of letters "N" or "S".

- If the letter "N" is present in **mode**, leading coefficients in the Janet basis are *not* normalized to 1.
- If the letter "S" is present in **mode**, the program uses simplify instead of expand in the normal form procedure. If the polynomials in the input **L** contain nonrational coefficients, more precisely, if the ground field contains algebraic elements over the rationals (RootOf, cf. Example 4 below), then ***simplify*** is used instead of ***expand*** automatically. Note, the program can also work with pure transcendental extensions, i. e. algebraically independent parameters. (If the parameters are algebraically dependent, there is always the danger of division by zero.)
- Possible left hand sides of the optional equations in **opt** are the strings "time" and "Groebner".
- If "time"=*t* is given in **opt**, then *t* is expected to be a non-negative integer. In this case, the involutive basis computation is stopped after *t* seconds. If the program was not able to construct the result before *t* seconds, then a warning is printed (cf. Example 8).
- If "Groebner"=*b* is given in **opt**, then *b* is expected to be a boolean value. If *b* is true, then only the reduced Groebner basis for the module generated by **L** (w.r.t. the chosen monomial ordering) is returned. The Groebner basis is extracted from the computed Janet basis. The default value for *b* is false.
- The ground field over which involutive bases are computed is the field of rational numbers by default. The characteristic of the ground field can be changed by JanetOreOptions. It is also possible to compute involutive bases of the ring of integers. By means of the command JanetOreOptions one can also choose between two implementations of ***JBasis***: "Maple" and "C++".
- One can specify a right hand side for each generator in order to let ***JBasis*** perform any operation on both left and right hand side. Right hand sides are assigned by an equal sign (cf. Example 2 below), usually they are symbols, but also tuples are possible, for instance, if one wants to construct a free resolution. A list **J_HOM** is constructed that contains all expressions being right hand sides in some step of the computation that correspond to zero left hand side. This list is used to find the syzygies among the generators in **L**, see JSyzygies.
- For a description of the basic algorithms, see V. P. Gerdt, "Involutive Algorithms for Computing Groebner Bases", in: S. Cojocaru, G. Pfister, V. Ufnarovski, "Computational Commutative and Non-Commutative Algebraic Geometry", NATO Science Series, IOS Press, 2005, pp. 199-225; or V. P. Gerdt, "Involutive Division Technique: Some Generalizations and Optimizations", Journal of Mathematical Sciences 108(6), 2002, pp. 1034-1051.
- For a description of the modifications of the algorithms towards the non-commutative version for Ore algebras, see D. Robertz, "Formal Computational Methods for Control Theory", PhD thesis, RWTH Aachen, Germany, 2006 (cf. also JanetOre).
- ***JBasis*** allows for assigning degrees other than 1 to the variables and also, in the module case, degrees other than 0 to the standard basis vectors of the free module. The syntax for this is to change **var** from $[x_1, ..., x_n]$ to $[x_1=d_1, ..., x_n=d_n]$ respectively to $[x_1=d_1, ..., x_n=d_n, 1=e_1, ..., m=e_m]$ in the module case. Here d_i is the degree of x_i and e_i the degree of the ith standard basis vector [0,...,0,1,0,...0] with the 1 in the ith position. The d_i must be natural numbers, the e_i integers. Of course, the Janet basis will in general be different from the standard one. Note, to continue with these degrees one has to work with JWeightedHilbertSeries instead of JHilbertSeries. For examples that deal with leading monomials in the case of non-standard degrees see JLeadingMonomial.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,D,delta];
```

$$var := [x, D, \delta]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(D, x), \mathrm{shift}(\delta, x)]$$

```
> L1 := [x*D-2, D^3+delta^3];
```

$$L1 := [xD - 2, D^3 + \delta^3]$$

```
> JBasis(L1, var, ops);
```

$$[xD - 2, \delta^3, D^3, \delta^3 D, \delta^3 x, \delta^3 D^2]$$

```
> JTabVar();
```

$$[xD - 2, [x, D, \delta], xD]$$
$$[\delta^3, [*, *, \delta], \delta^3]$$
$$[D^3, [*, D, \delta], D^3]$$

$$[\delta^3 D, [*, *, \delta], \delta^3 D]$$
$$[\delta^3 x, [x, *, \delta], \delta^3 x]$$
$$[\delta^3 D^2, [*, *, \delta], \delta^3 D^2]$$

```
> JBasis(L1, [D,delta,x], ops);
```

$$[xD-2, xD\delta+D\delta-2\delta, xD^2-D, \delta^3, D^3, xD\delta^2+2D\delta^2-2\delta^2, \delta D^2+x\delta D^2-D\delta, \delta^3 D, 2\delta^2 D^2+x\delta^2 D^2-D\delta^2, \delta^3 D^2]$$

Example 2:

```
> L2a := [[x^2-1, 0], [x*y, x*y], [0, y^2-1]];
```

$$L2a := [[x^2-1, 0], [xy, xy], [0, y^2-1]]$$

```
> JBasis(L2a, [x,y]);
```

$$[[0, y^2-1], [xy, xy], [y^2, x^2], [x^2-1, 0], [y^3-y, 0], [0, -x+xy^2]]$$

The generators of the submodule can also be specified in matrix form:

```
> L2b := matrix(3, 2, [[x^2-1, 0], [x*y, x*y], [0, y^2-1]]);
```

$$L2b := \begin{bmatrix} x^2-1 & 0 \\ xy & xy \\ 0 & y^2-1 \end{bmatrix}$$

```
> JBasis(L2b, [x,y]);
```

$$[[0, y^2-1], [xy, xy], [y^2, x^2], [x^2-1, 0], [y^3-y, 0], [0, -x+xy^2]]$$

Next we see the last example with right hand sides:

```
> L2c := [[x^2-1, 0]=a, [x*y, x*y]=b, [0, y^2-1]=c];
```

$$L2c := [[x^2-1, 0]=a, [xy, xy]=b, [0, y^2-1]=c]$$

```
> JBasis(L2c, [x,y], 2);
```

$$[[0, y^2-1]=c, [0, -x+xy^2]=xc, [0, x^3-x]=xc-xy^2 a+yx^2 b-yb-x^3 c, [0, -x^2+y^2 x^2]=x^2 c, [y, x^2 y]=-ya+xb,$$
$$[xy, xy]=-y^2 x^2 b+xy^3 a+(-a-c)xy+y^2 b+yx^3 c+x^2 b, [x^2-1, 0]=a]$$

```
> JSyzygies(L2c, [x,y]);
```

$$[y^2 x^2 b-xy^3 a+b+(c+a)xy-y^2 b-yx^3 c-x^2 b, -xb+yc x^4+b x^3-b y^2 x^3+a y^3 x^2+(-a-c)yx^2+y^2 bx,$$
$$(-a-c)xy^2+yx^2 b+y^2 x^3 c+y^3 b-yb+y^4 xa-y^3 x^2 b]$$

```
> L2d := [[x^2-1, 0]=[1,0,0], [x*y, x*y]=[0,1,0], [0, y^2-1]=[0,0,1]];
```

$$L2d := [[x^2-1, 0]=[1, 0, 0], [xy, xy]=[0, 1, 0], [0, y^2-1]=[0, 0, 1]]$$

```
> JBasis(L2d, [x,y], 2);
```

$$[[0, y^2-1]=[0, 0, 1], [0, -x+xy^2]=[0, 0, x], [0, x^3-x]=[-xy^2, x^2 y-y, -x^3+x], [0, -x^2+y^2 x^2]=[0, 0, x^2], [y, x^2 y]=[-y, x, 0],$$
$$[xy, xy]=[-xy+y^3 x, -y^2 x^2+y^2+x^2, x^3 y-xy], [x^2-1, 0]=[1, 0, 0]]$$

```
> JSyzygies(L2d, [x,y]);
```

$$[$$
$$[-y^3 x+xy, y^2 x^2-y^2-x^2+1, -x^3 y+xy], [x^2 y^3-x^2 y, -x+x^3-y^2 x^3+xy^2, -x^2 y+x^4 y], [-xy^2+xy^4, x^2 y-y-x^2 y^3+y^3, y^2 x^3-xy^2]$$
$$]$$

Note, these syzygies can be interpreted as a matrix representing a homomorphism of the free module of rank 1 into the free module of rank 3, whose cokernel is the module generated by *L2d*.

Example 3: Algebraically dependent parameters:

```
> L3 := [x^2+y^2 - P1, x^2*y^2 - P2, x*y^3-x^3*y - P3];
```

$$L3 := [x^2+y^2-P1, y^2 x^2-P2, y^3 x-x^3 y-P3]$$

```
> JBasis(L3, [x,y]);
```

$$[1]$$

```
> JZeroSets();
```

$$[-P1^3 P2+4P2^2 P1+P3^2 P1, -P1^2 P2+4P2^2+P3^2, -P3P1^3 P2+4P3P1P2^2+P3^3 P1]$$

```
> map(factor, %);
```

$$[P1(-P1^2 P2+4P2^2+P3^2), -P1^2 P2+4P2^2+P3^2, P3P1(-P1^2 P2+4P2^2+P3^2)]$$

```
> s := P1^2*P2-4*P2^2-P3^2;
```

$$s := P1^2 P2-4P2^2-P3^2$$

```
> expand(subs([P1=x^2+y^2, P2=x^2*y^2, P3=x*y^3-x^3*y], s));
```

0

Example 4: The next example deals with nonrational coefficients:

```
> alias(omega=RootOf(a^2+a+1,a)):
> simplify(omega^2);
```

$$-1-\omega$$

```
> L4 := [x+omega*y+omega^2*z,x*y+y*z+z*x,x*y*z-1];
```

$$L4 := [x+\omega y+\omega^2 z, xy+yz+zx, xyz-1]$$

```
> JBasis(L4, [x,y,z]);
```

$$\left[x+y+2\,\omega y+\omega^2 y-z-z\,\omega,\ \omega\,(-y^2-\omega y^2-2\,\omega yz+z^2),\ \frac{1}{3}(2\,\omega+1)(z^3-2\,yz^2\,\omega-yz^2-\omega),\ -z\,\omega-3\,z+z^4+\omega y-y\right]$$

```
> JBasis(L4, [x,y,z], "N");
```

$$[-\omega x-z+(1+\omega)y,\ (1+\omega)y^2+2\,\omega yz-z^2,\ -z^3+(2\,\omega+1)yz^2+\omega,\ (-\omega-3)z+z^4+(\omega-1)y]$$

Example 5: Assigning weights ("degrees") to the variables:

```
> L5 := [x^2-y^3, x^4+y^6];
```

$$L5 := [x^2-y^3, x^4+y^6]$$

```
> JBasis(L5, [x=3,y=2]);
```

$$[x^2-y^3, y^6, xy^6]$$

```
> JWeightedHilbertSeries([x=3,y=2]);
```

$$1+s^3+s^5+s^7+s^9+s^{11}+s^{13}+s^2+s^4+s^6+s^8+s^{10}$$

Example 6: Changing the priority of tuple entries

```
> L6 := [[x^2,y,0], [x^3,y^2,x-y]];
```

$$L6 := [[x^2, y, 0], [x^3, y^2, x-y]]$$

```
> JBasis(L6, [x,y], 2);
```

$$[[0, xy-y^2, -x+y], [x^2, y, 0]]$$

```
> JBasis(L6, [x,y,2,3,1], 2);
```

$$[[-x^2\,y+x^3, 0, x-y], [x^2, y, 0]]$$

Example 7: Block ordering

```
> L7 := [x*y-z^3, x*y*z-x^2*y^2];
```

$$L7 := [xy-z^3, xyz-y^2\,x^2]$$

```
> JBasis(L7, [x,y,z]);
```

$$[-xy+z^3, -x^2\,y+z^3\,x, -xyz+y^2\,x^2, -x^3\,y+z^3\,x^2, z^3\,yx^2-x^2\,yz]$$

```
> JBasis(L7, [[x,y],[z]], [4,4]);
```

$$[-z^4+z^6, -z^4\,x+z^6\,x, xy-z^3]$$

Example 8: Stop computation of involutive basis within a prescribed time bound

```
> L8 := [seq(a[i]^3-a[i+1]-1, i=1..6),seq(a[i]^2-a[i-1]+1, i=2..3)];
```

$$L8 := [a_1^{\,3}-a_2-1, a_2^{\,3}-a_3-1, a_3^{\,3}-a_4-1, a_4^{\,3}-a_5-1, a_5^{\,3}-a_6-1, a_6^{\,3}-a_7-1, a_2^{\,2}-a_1+1, a_3^{\,2}-a_2+1]$$

```
> JBasis(L8, [seq(a[i], i=1..7)], "time"=5):
Warning, computation of involutive basis stopped due to time restriction.
```

See Also:

JTabVar, JLeadingMonomial, JBasisFast, JGroebnerBasis, JanetOreOptions, JInvReduce, JFactorModuleBasis, JHilbertSeries, JWeightedHilbertSeries, JSyzygies, JSyzygyModule, JCartanCharacter, JHas.

JanetOre[JBasisFast] - return the (unique) minimal Janet Basis of a submodule of a free left module over an Ore algebra (C++ version)

Calling Sequence:

JBasisFast(L,var,ops,ord,mode,opt)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`ord` - (optional) change of monomial ordering
`mode` - (optional) string specifying options for the computation
`opt` - (optional) sequence of equations specifying options for the computation

Description:

- ***JBasisFast*** invokes the C++ version of the command JBasis. Up to now, only the algorithm for the degree reverse lexicographical ordering (i.e. **`ord`** is 2 or 4) is implemented in C++. If ***JBasisFast*** is called choosing a different monomial ordering, then JBasis is applied internally to the same data.
- The parameters **`L`**, **`var`**, **`ops`**, **`ord`**, and **`mode`** have the same meaning as in JBasis.
- The advantage of this command is clearly the enormous speed up. (Note, you cannot interrupt this command from within Maple; you have to kill the process "JBOre" instead.)
- The output of the C++ program is read into the current Maple session. To continue with commands that expect a previous run of JBasis (like JTabVar, JFactorModuleBasis, JHilbertSeries, etc.) the internal data structure for the involutive basis has to be set up by the command AssertJBasis (cf. example below).
- Possible left hand sides of the optional equations **`opt`** are the strings "char", "time", "Name", "denom", and "donotread".
- If an equation "char"=*c* is provided in **`opt`** by the user, then *c* is expected to be zero or a prime number. In this case, the involutive basis is computed in characteristic *c* (cf. Example 2). The purpose of this option is to compute just one involutive basis in characteristic *c*. If further commands like JMinPoly shall be applied afterwards, the characteristic of the ground field must be changed by using the command JanetOreOptions.
- If "time"=*t* is given in **`opt`**, then *t* is expected to be a non-negative integer. In this case, the involutive basis computation is stopped after *t* seconds. If the program was not able to construct the result before *t* seconds, then a warning is printed (cf. Example 3).
- The right hand side of an equation "Name"=*s* is expected to be a string. ***JBasisFast*** appends *s* to the default name for the temporary file to which the input for the external program JBOre is written.
- The right hand side of an equation "denom"=*b* is expected to be either true or false. The default value is false. If *b* equals true, then the C++ program collects all coefficients by which it divides during the computation of the involutive basis (these arise either as contents of polynomials treated by the algorithm or as leading coefficients in the result before normalizing) together with the coefficients that occur in some denominator of the input **`L`**. After the computation is finished and the result is read into Maple, this list of denominators can be obtained via JZeroSets. See also Example 4 below.
- The right hand side of an equation "donotread"=*b* is expected to be a boolean value. If *b* equals true, then ***JBasisFast*** does not read the result produced by the C++ program and does not return a result.
- Using the option "C++" of JanetOreOptions, the command ***JBasis*** is replaced by ***JBasisFast*** (i.e. the former becomes a synonym for the latter) for the current Maple session (which has the corresponding effect on all Maple procedures that call ***JBasis***).

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,D,delta];
```

$$var := [x, \mathrm{D}, \delta]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x), \mathrm{shift}(\delta, x)]$$

```
> L1 := [x*D-2, D^3+delta^3];
```

$$L1 := [x\,\mathrm{D} - 2, \mathrm{D}^3 + \delta^3]$$

```
> J := JBasisFast(L1, var, ops);
```

$$J := [x\,\mathrm{D} - 2, \delta^3, \mathrm{D}^3, \mathrm{D}\,\delta^3, x\,\delta^3, \mathrm{D}^2\,\delta^3]$$

```
> AssertJBasis(J, var, ops);
```

$$[x\,\mathrm{D} - 2, \delta^3, \mathrm{D}^3, \mathrm{D}\,\delta^3, x\,\delta^3, \mathrm{D}^2\,\delta^3]$$

```
> JTabVar();
```

$$[x\,\mathrm{D} - 2, [x, \mathrm{D}, \delta], x\,\mathrm{D}]$$
$$[\delta^3, [*, *, \delta], \delta^3]$$
$$[\mathrm{D}^3, [*, \mathrm{D}, \delta], \mathrm{D}^3]$$
$$[\mathrm{D}\,\delta^3, [*, *, \delta], \mathrm{D}\,\delta^3]$$
$$[x\,\delta^3, [x, *, \delta], x\,\delta^3]$$
$$[\mathrm{D}^2\,\delta^3, [*, *, \delta], \mathrm{D}^2\,\delta^3]$$

```
> JFactorModuleBasis(var);
```

$$1 + \frac{x}{1-x} + \frac{x\,\delta}{1-x} + \frac{x\,\delta^2}{1-x} + \mathrm{D}^2 + \mathrm{D}^2\,\delta + \mathrm{D}^2\,\delta^2 + \mathrm{D} + \mathrm{D}\,\delta + \mathrm{D}\,\delta^2 + \delta + \delta^2$$

```
> JHilbertSeries(lambda);
```

$$1 + 3\,\lambda + 5\,\lambda^2 + 5\,\lambda^3 + 4\,\lambda^4 + \frac{3\,\lambda^5}{1-\lambda}$$

```
> JBasisFast(L1, [D,delta,x], ops);
```

$$[x\,\mathrm{D} - 2, \mathrm{D}\,\delta\,x + \mathrm{D}\,\delta - 2\,\delta, \mathrm{D}^2\,x - \mathrm{D}, \delta^3, \mathrm{D}^3, \mathrm{D}\,\delta^2\,x + 2\,\mathrm{D}\,\delta^2 - 2\,\delta^2, \mathrm{D}^2\,\delta\,x + \mathrm{D}^2\,\delta - \mathrm{D}\,\delta, \mathrm{D}\,\delta^3, \mathrm{D}^2\,\delta^2\,x + 2\,\mathrm{D}^2\,\delta^2 - \mathrm{D}\,\delta^2, \mathrm{D}^2\,\delta^3]$$

Example 2:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x+2*y+3*z, x*y+2*y*z+3*z*x, x*y*z-1];
```

$$L := [x + 2\,y + 3\,z, x\,y + 2\,y\,z + 3\,z\,x, x\,y\,z - 1]$$

```
> JBasisFast(L, var, "char"=7);
```

$$[x + 2\,y + 3\,z, y^2 + z^2, y\,z^2 + 4\,z^3 + 5, z^4 + 3\,y + 2\,z]$$

Example 3:

```
> var := [seq(a[i], i=1..12)];
```

$$var := [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}]$$

```
> L := [seq(a[i]^5-a[i+1]^4, i=1..nops(var)-1),seq(a[i]^3-a[i-1]^2, i=2..nops(var))];
```

$$L := [a_1^5 - a_2^4, a_2^5 - a_3^4, a_3^5 - a_4^4, a_4^5 - a_5^4, a_5^5 - a_6^4, a_6^5 - a_7^4, a_7^5 - a_8^4, a_8^5 - a_9^4, a_9^5 - a_{10}^4, a_{10}^5 - a_{11}^4, a_{11}^5 - a_{12}^4, a_2^3 - a_1^2,$$
$$a_3^3 - a_2^2, a_4^3 - a_3^2, a_5^3 - a_4^2, a_6^3 - a_5^2, a_7^3 - a_6^2, a_8^3 - a_7^2, a_9^3 - a_8^2, a_{10}^3 - a_9^2, a_{11}^3 - a_{10}^2, a_{12}^3 - a_{11}^2]$$

```
> JBasisFast(L, var, "time"=1):
Warning, computation of involutive basis stopped due to time restriction.
> JBasisFast(L, var):
Warning, resulting involutive basis is big; reading it may take a while...
> nops(JFactorModuleBasis(var, "L"));
```

8199

Example 4:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [3*x*y-5, x-5*y];
```

$$L := [3xy - 5, x - 5y]$$

```
> JBasisFast(L, var, "denom"=true);
```

$$\left[x - 5y, y^2 - \frac{1}{3}\right]$$

```
> JZeroSets();
```

$$[5, 3]$$

```
> JBasisFast(L, var, "N", "denom"=true);
```

$$[x - 5y, 3y^2 - 1]$$

```
> JZeroSets();
```

$$[5]$$

See Also:

JBasis, AssertJBasis, JanetOreOptions, JTabVar, JFactorModuleBasis, JInvReduce, JInvReduceFast, JSyzygies, JSyzygyModule, JSyzygyModuleFast, JHilbertSeries.

JanetOre[JCartanCharacter] - compute Cartan characters of a finitely presented module over an Ore algebra

Calling Sequence:

JCartanCharacter(i)
JCartanCharacter()

Parameters:

`i` - " " (empty string) or natural number smaller or equal to the number of indeterminates

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JHilbertSeries. Then the Cartan characters $\alpha(q, i)$ are defined by
$$\sum_{i=0}^{\infty} d_i v^i \sum_{i=0}^{\infty} d_i v^i = \left(\sum_{i=0}^{q-1} d_i v^i\right) + v^q \left(\sum_{j=1}^{n} \frac{\alpha(q,j)}{(1-v)^j}\right),$$
where q is the highest degree of the polynomials in the Janet basis computed by the last call of JBasis. (The same formula holds with q replaced by the index of regularity plus 1, cf. JIndexRegularity, with suitably modified Cartan characters α´s.)
- ***JCartanCharacter***() prints the the highest degree q of the polynomials in the Janet basis computed by the last call JBasis and returns the list of Cartan characters $[\alpha(q, 1), ..., \alpha(q, n)]$ of the factor module of the free module over the Ore algebra modulo the submodule generated by this Janet basis. ***JCartanCharacter***(`i`) returns the Cartan character $\alpha(q, i)$.
- All this information can also be extracted from the command JHilbertSeries.
- ***JCartanCharacter***(" ") simply prints the Cartan characters $\alpha(q, 1), ..., \alpha(q, n)$, where q is as above.

Examples:

```
> with(JanetOre):
> var := [x,y,z,v];
```

$$var := [x, y, z, v]$$

```
> L := [x*y+y*z+z*x, x*y*z-v];
```

$$L := [xy+yz+zx, xyz-v]$$

```
> B := JBasis(L, var);
```

$$B := [xy+yz+zx, yz^2+z^2x+v, z^2y^2+vy+zv]$$

```
> JCartanCharacter("");
alpha(4,1) = 15
alpha(4,2) = 6
alpha(4,3) = 0
alpha(4,4) = 0
> JCartanCharacter();
Cartan Character for q = 4
```

$$[15, 6, 0, 0]$$

```
> JCartanCharacter(2);
```

$$6$$

```
> JTabVar();
```

$$[xy+yz+zx, [x, y, z, v], xy]$$
$$[yz^2+z^2x+v, [x, *, z, v], z^2x]$$
$$[z^2y^2+vy+zv, [*, y, z, v], z^2y^2]$$

```
> JHilbertSeries(s);
```

$$1+4s+9s^2+15s^3+s^4\left(15\frac{1}{1-s}+6\frac{1}{(1-s)^2}\right)$$

```
> JIndexRegularity();
```

$$1$$

In particular, the Hilbert series could be rewritten as 1 + 4*t + t^2*(6*1/((1-t)^2) + 3*1/(1-t)).
Note, the Cartan characters depend on the Hilbert series, which is the same for all variable orderings in case one works with the degree reversed lexicographical ordering. However, if one works with the pure lexicographical ordering, one will usually get different

Hilbert series and hence different Cartan characters:

```
> B1 := JBasis(L, [v,x,y,z], 1);
```

$$B1 := [xy + yz + zx, yz^2 + z^2 x + v]$$

```
> JHilbertSeries(s);
```

$$1 + 3s + s^2\left(3\frac{1}{1-s} + 2\frac{1}{(1-s)^2}\right)$$

```
> JCartanCharacter();
Cartan Character for q = 2
```

$$[3, 2, 0, 0]$$

See Also:

JBasis, JTabVar, JHilbertSeries, JHilbertPolynomial, JHP, JHilbertFunction, JHF, JIndexRegularity.

JanetOre[JCoeff] - express module element as linear combination of given module elements

Calling Sequence:

JCoeff(L,G,var,ops)

Parameters:

`L`	-	polynomial or list (of lists of the same length) of polynomials or matrix with polynomial entries
`G`	-	list (of lists of the same length) of polynomials
`var`	-	list of variables of the Ore algebra
`ops`	-	(optional) list of commutation rules for the variables of the Ore algebra

Description:

- ***JCoeff*** expresses (if possible) the polynomials in **L** or the elements in **L** of a free module of tuples over the Ore algebra which is specified by **var** and **ops** as linear combinations of the polynomials resp. the tuples of polynomials of the same length in **G**. The result of ***JCoeff*** is the matrix of coefficients of these linear combinations such that the product of this matrix by the column vector composed of the entries in **G** is the column vector of entries in **L**, if all entries in **L** can be expressed as linear combinations of the entries in **G**. For all entries of **L** that cannot be expressed in this way, the remainder defined by reduction modulo the involutive basis of **G** is ignored, i.e. the matrix product described before only reproduces the part of each entry of **L** that has zero remainder modulo **G**.
- Technically, ***JCoeff*** computes an involutive basis of **G** with right hand sides first (see JBasis, JAddRhs) and then performs involutive reduction on all elements in **L** modulo this involutive basis keeping all coefficients of these reductions (see JInvReduce). The matrix formed by these coefficients is then multiplied by the matrix composed of the right hand sides of the involutive basis which finally yields the expressions of the entries in **L** in terms of the entries in **G** (if all entries in **L** reduce to zero modulo the involutive basis of **G**).
- The module elements to be expressed are given in the first argument **L**. This argument is either a single polynomial, a list of polynomials, a list of lists of polynomials of the same length or a matrix of polynomials. In the latter case, a list of lists of polynomials is extracted from the matrix **L** by interpreting the rows of **L** as tuples of polynomials. A list of polynomials may either stand for several polynomials to be expressed as linear combinations or for one element of a free module of tuples of polynomials to be expressed as a linear combination. The context is clear from the structure of the second argument **G**.
- The commutation rules in the optional list **ops** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- The second argument **G** is either a list of polynomials in the indeterminates **var** or a list of lists of the same length of polynomials in the indeterminates **var**.
- If **L** is a list of length *m* (or represents a single module element) and **G** is a list of length *n*, then the result of ***JCoeff*** is an (*m* x *n*)-matrix (resp. a (1 x *n*)-matrix) with polynomial entries.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> G := [x^2+y^2, x^4+y^4];
```

$$G := [x^2 + y^2, x^4 + y^4]$$

```
> L := x^8+y^8;
```

$$L := x^8 + y^8$$

```
> C := JCoeff(L, G, var);
```

$$C := \left[x^6 - y^2 x^4 + x^2 y^4 - y^6 + 2 y^4 \left(\frac{1}{2} y^2 - \frac{1}{2} x^2 \right) \quad y^4 \right]$$

The matrix *C* gives the coefficients in a linear combination of the polynomials in *G* that equals *L*.

```
> JMult(C, [[x^2+y^2], [x^4+y^4]], var);
```

$$[x^8 + y^8]$$

This can be done simultaneously for several polynomials:

```
> L := [x^6+y^6, x^4*y^4];
```

$$L := [x^6 + y^6, y^4 x^4]$$

```
> C := JCoeff(L, G, var);
```

$$C := \begin{bmatrix} x^4 - x^2 y^2 + y^4 & 0 \\ x^2 y^4 - y^6 + y^4 \left(\frac{1}{2} y^2 - \frac{1}{2} x^2\right) & \frac{1}{2} y^4 \end{bmatrix}$$

```
> B := [[x^2+y^2], [x^4+y^4]];
```

$$B := [[x^2 + y^2], [x^4 + y^4]]$$

```
> map(expand, JMult(C, B, var));
```

$$\begin{bmatrix} x^6 + y^6 \\ y^4 x^4 \end{bmatrix}$$

JCoeff also accepts a matrix L instead of a list L:

```
> M := matrix([[x^6+y^6], [x^4*y^4]]);
```

$$M := \begin{bmatrix} x^6 + y^6 \\ y^4 x^4 \end{bmatrix}$$

```
> C := JCoeff(M, G, var);
```

$$C := \begin{bmatrix} x^4 - x^2 y^2 + y^4 & 0 \\ x^2 y^4 - y^6 + y^4 \left(\frac{1}{2} y^2 - \frac{1}{2} x^2\right) & \frac{1}{2} y^4 \end{bmatrix}$$

```
> B := matrix(map(a->[a], G));
```

$$B := \begin{bmatrix} x^2 + y^2 \\ x^4 + y^4 \end{bmatrix}$$

```
> map(expand, JMult(C, B, var));
```

$$\begin{bmatrix} x^6 + y^6 \\ y^4 x^4 \end{bmatrix}$$

Note that all remainders of reduction modulo the involutive basis of **G** are ignored:

```
> JCoeff(x+y, G, var);
```

$$[0 \quad 0]$$

Example 2:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> G := [[y, x*y*z], [y+1, 0]];
```

$$G := [[y, xyz], [y+1, 0]]$$

```
> L := [-1, x*y*z];
```

$$L := [-1, xyz]$$

```
> C := JCoeff(L, G, var);
```

$$C := [1 \quad -1]$$

```
> JMult(C, G, var);
```

$$[-1 \quad xyz]$$

```
> map(op, convert(%, listlist));
```

$$[-1, xyz]$$

See Also:

JBasis, JTabVar, JInvReduce, JAddRhs, JSyzygies, JSyzygyModule, JLeftInverse, JRightInverse.

JanetOre[JCoeffList] - express a (tuple of) polynomial(s) in a given vector space basis of monomials

Calling Sequence:

JCoeffList(p,var,B)

Parameters:

p	-	(tuple of) polynomial(s) in **var** to be expressed
var	-	list of variables (of the Ore algebra)
B	-	vector space basis given as list of monomials or generating function (result of JFactorModuleBasis)

Description:

- ***JCoeffList*** returns the list of coefficients of the unique representation of **p** in the vector space basis **B**.
- The parameter **p** is either a polynomial in the variables **var** or a tuple given as a list of polynomials in **var**.
- **var** is the list of variables of the Ore algebra.
- The list **B** is expected to be a result of a previous call of JFactorModuleBasis. If **p** is a polynomial, then **B** is expected to be a factor module basis computed for a residue class module of an Ore algebra with variables **var**. If **p** is a list of polynomials of length *m*, then **B** is expected to be a factor module basis computed for a factor module of the free module of tuples over an Ore algebra with variables **var** of rank *m*.
- If **p** is in the span of the vector space basis **B**, then the result is the list of coefficients of the unique representation of **p** in the basis **B**.
- If **p** is not in the span of **B**, then the result is the monomial in **p** which is not an element of **B** and which is encountered first by ***JCoeffList***. If **p** is a list of polynomials, then the result is accordingly a list of the same length with exactly one non-zero entry which is a monomial in **var** (cf. Example 2).
- If **B** is a list, i.e. the vector space basis is finite, then the resulting list has as many entries as **B**, and these entries are in the ground field.
- If **B** is given as generating function, e.g. **B** is the sum of monomials according to a disjoint cone decomposition of a factor module, then the *i*-th entry of the resulting list is a polynomial in the multiplicative variables for the *i*-th cone in the basis **B**, i.e. a polynomial in the variables occurring in the corresponding denominator, where the cones are sorted by their vertices with respect to the degree-reverse lexicographic ordering (cf. Example 3). The number of entries equals the number of cones in this case.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x+y+z, x*y+y*z+z*x, x*y*z-1];
```

$$L := [x+y+z, xy+yz+zx, xyz-1]$$

```
> JBasis(L, var);
```

$$[x+y+z, y^2+yz+z^2, z^3-1, z^3y-y]$$

```
> F := JFactorModuleBasis(var);
```

$$F := [1, z, y, z^2, yz, z^2y]$$

```
> p := a + b*z + c*y + d*z^2 + e*y*z + f*z^2*y;
```

$$p := a+bz+cy+dz^2+eyz+fz^2y$$

```
> JCoeffList(p, var, F);
```

$$[a, b, c, d, e, f]$$

The next polynomial is not in the span of F.

```
> p := x+1;
```

$$p := x+1$$

```
> JCoeffList(p, var, F);
```

$$x$$

Example 2:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L2 := [[x^2-1, 0], [x*y, x*y], [0, y^2-1]];
```

$$L2 := [[x^2 - 1, 0], [xy, xy], [0, y^2 - 1]]$$

```
> JBasis(L2, [x,y]);
```

$$[[0, y^2 - 1], [xy, xy], [y^2, x^2], [x^2 - 1, 0], [y^3 - y, 0], [0, -x + y^2 x]]$$

```
> F := JFactorModuleBasis(var);
```

$$F := [[0, 1], [0, y], [0, x], [0, xy], [1, 0], [y, 0], [x, 0], [y^2, 0]]$$

```
> p := [1+3*y, 5*x+7*x*y];
```

$$p := [1 + 3y, 5x + 7xy]$$

```
> JCoeffList(p, var, F);
```

$$[0, 0, 5, 7, 1, 3, 0, 0]$$

The next tuple is not in the span of F.

```
> p := [1+y^2, x^2];
```

$$p := [1 + y^2, x^2]$$

```
> JCoeffList(p, var, F);
```

$$[0, x^2]$$

Example 3:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [x^2*y-x, x*y^2-y];
```

$$L := [x^2 y - x, y^2 x - y]$$

```
> JBasis(L, var);
```

$$[y^2 x - y, x^2 y - x]$$

```
> F := JFactorModuleBasis(var);
```

$$F := \frac{1}{1-y} + \frac{x^2}{1-x} + x + xy$$

```
> JFactorModuleBasis(var, "C");
```

$$[1, x, xy, x^2]$$

```
> JCoeffList(3 + 2*x + 7*x*y + (-12)*x^2, var, F);
```

$$[3, 2, 7, -12]$$

```
> JCoeffList(3*y^2 + 2*x + 7*x*y + (-12)*x^5, var, F);
```

$$[3y^2, 2, 7, -12x^3]$$

See Also:

JBasis, JInvReduce, JFactorModuleBasis, JSubmoduleBasis, JRepres.

JanetOre[JCokernel] - return presentation of the cokernel of a homomorphism between two finitely presented left modules over an Ore algebra

JanetOre[JSum] - return Janet basis of the sum of two submodules of a free left module over an Ore algebra

Calling Sequence:

JCokernel(A,N,var,ops)
JSum(A,N,var,ops)

Parameters:

`A` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`N` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the Ore algebra
`ops` - (optional) list of commutation rules for the variables of the Ore algebra

Description:

- ***JCokernel*** returns a presentation of the cokernel of the homomorphism represented by **A** between two finitely presented left modules over Ore algebra specified by **`var`** and **`ops`**. The domain of this homomorphism is a factor module of the free left module of tuples of length equal to the number of entries in **A** (resp. the number of rows of **A**, if **A** is a matrix). While the domain of this homomorphism does not need to be specified for ***JCokernel***, a presentation of its range is given by **N**. More precisely, the homomorphism represented by **A** maps into the factor module of the free left module of tuples of length equal to the length of the lists in **A** (resp. 1 if the entries of **A** are polynomials resp. the number of colums of **A**, if **A** is a matrix) modulo its submodule generated by the entries in **N** (or the rows of **N**, if **N** is a matrix). The residue classes of the entries resp. rows of **A** in the module presented by **N** generate the image of this homomorphism.
- The entries of **A** and **N** are polynomials in case the range of the homomorphism is a factor module of the free left module of rank one, or lists of polynomials of length *m*, representing elements of the free left module of *m*-tuples over the Ore algebra.
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- The result is a Janet basis with respect to the ("term over position") degree reverse lexicographical ordering (cf. JBasis). The cokernel of the given homomorphism is the factor module of the free left module of *m*-tuples modulo the submodule generated by the entries of the result of ***JCokernel***.
- ***JSum*** is a synonym for ***JCokernel***. The interpretation of input and output is different in this case: The input **A**, **N** forms two generating sets for submodules of a free left module of tuples over the Ore algebra specified by **`var`** and **`ops`**. The result of ***JSum*** is a Janet basis for the sum of the two submodules.

Examples:

```
> with(JanetOre):
```

Example 1: Presenting the cokernel of a homomorphism

```
> var := [D,x];
```

$$var := [D, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(D, x)]$$

```
> A := matrix([[D^2, D^2, 0], [0, x+1, x+1]]);
```

$$A := \begin{bmatrix} D^2 & D^2 & 0 \\ 0 & x+1 & x+1 \end{bmatrix}$$

```
> JCokernel(A, [[0,0,0]], var, ops);
```

$$[[0, x+1, x+1], [D^2, D^2, 0]]$$

```
> N := [[x, 0, 0], [0, x, 0], [0, 0, x]];
```

$$N := [[x, 0, 0], [0, x, 0], [0, 0, x]]$$

```
> JCokernel(A, N, [x,y]);
```

$$[[0, 1, 1], [1, 0, -1], [0, 0, x]]$$

Example 2: Computing the Janet basis of the sum of two submodules of a free left module over an Ore algebra

```
> var := [delta,x];
```

$$var := [\delta, x]$$

```
> ops := [shift(delta,x)];
```

$$ops := [\mathrm{shift}(\delta, x)]$$

```
> M1 := [[x-1, delta], [0, x^2-delta^2]];
```

$$M1 := [[x-1, \delta], [0, x^2-\delta^2]]$$

```
> M2 := [[x+1, delta], [0, x]];
```

$$M2 := [[x+1, \delta], [0, x]]$$

```
> JSum(M1, M2, var, ops);
```

$$[[1, 0], [0, x], [0, \delta]]$$

See Also:

JBasis, JTabVar, JInvReduce, JSyzygies, JSyzygyModule, JResolution, JDirectSum, JSubFactor, JKernel, JSyzOp.

JanetOre[JDefect] - return presentation of defect of exactness / homology module at a certain position of a chain complex

Calling Sequence:

JDefect(L1,L2,var,ops,v)

Parameters:

`L1` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`L2` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the Ore algebra
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`v` - (optional) name of the indeterminate for the Hilbert series of the homology module (default: 's')

Description:

- ***JDefect*** returns a presentation of the defect of exactness at a certain position in a chain complex of finitely generated free modules over a polynomial ring, i.e., it returns a presentation of a homology module of this chain complex.
- **L1** and **L2** are interpreted as matrices representing homomorphisms between finitely generated free modules over the polynomial ring in the variables **var**, such that their composition (namely the homomorphism represented by **L2** succeeding the homomorphism represented by **L1**) is well defined and gives the zero homomorphism. Row convection is applied, i.e., a polynomial matrix represents the homomorphism defined by multiplication of rows on the left of this matrix. ***JDefect*** computes a presentation of the factor module defined as the kernel of the homomorphism represented by **L2** modulo the image of the homomorphism represented by **L1**.
- The entries of **L1** and **L2** may be polynomials or lists of polynomials of the same length which represent generators for the image of the represented homomorphism into a free module of tuples over the polynomial ring. In the latter case, the common length of the lists in **L1** must equal the number of lists (i.e. the number of generators) in **L2**. If **L1** or **L2** is a matrix, then the generators for the image of the represented homomorphism are extracted from the rows of **L1** resp. **L2**.
- The commutation rules in the optional list **ops** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- Since the result of ***JDefect*** is a presentation of a subfactor module which is computed using JSubFactor, the output of ***JDefect*** is a list which is formatted in the same way as the output of JSubFactor. For a description of the format of such a presentation, cf. JSubFactor.
- The optional fourth argument to ***JDefect*** selects the name of the indeterminate for the Hilbert series given as third entry in the resulting list. The default name is 's' which cannot be affected by a subs command.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L1 := matrix(1, 2, [x^2,x*y]);
```

$$L1 := [x^2 \quad xy]$$

```
> L2 := matrix(2, 1, [y,-x]);
```

$$L2 := \begin{bmatrix} y \\ -x \end{bmatrix}$$

Here **L1** (resp. **L2**) represents a homomorphism from the free module of rank 1 (resp. 2) over the polynomial ring in **var** into the free module of rank 2 (resp. 1) over the polynomial ring in **var**. The composition of the homomorphisms represented by **L1** and **L2** is the zero homomorphism:

```
> evalm(L1 &* L2);
```

$$[0]$$

```
> PolDefect(L1, L2, var);
```

$$\left[[[1]=[x,y]],[x],1+\frac{s}{1-s},[1,0]\right]$$

The defect of exactness is generated by the residue class represented by $[x,y]$; this generator is annihilated by x.

```
> PolDefect(L1, L2, var, lambda);
```

$$\left[[[1]=[x,y]],[x],1+\frac{\lambda}{1-\lambda},[1,0]\right]$$

Changing the first homomorphism as follows, we obtain exactness of the chain complex at the position considered here:

```
> L1a := matrix(1, 2, [x, y]);
```

$$L1a := \begin{bmatrix} x & y \end{bmatrix}$$

```
> PolDefect(L1a, L2, var);
```

$$[[[1]=[0,0]],[1],0,[0,0]]$$

Example 2:

```
> var := [x,y];
```

$$var := [x,y]$$

```
> L1 := [x]; L2 := [0];
```

$$L1 := [x]$$

$$L2 := [0]$$

L1 (resp. **L2**) represent the homomorphism defined as multiplication by x (resp. 0) from and into the polynomial ring in **var**.

```
> PolDefect(L1, L2, var);
```

$$\left[[[1]=[1]],[x],1+\frac{s}{1-s},[1,0]\right]$$

See Also:

JBasis, JInvReduce, JHilbertSeries, JSyzygies, JSyzygyModule, JResolution, JSubFactor, JKernel, JHom, JHomHom, JExt1, JExtn, JTorsion, JParametrization, JSyzOp.

JanetOre[JDimension] - return the dimension of the factor module presented by the last computed Janet basis

Calling Sequence:

JDimension()

Parameters:

- - none (assumes that the involutive basis has been computed before)

Description:

- ***JDimension*** returns the degree of the filtered Hilbert polynomial (as in JHP) of the filtration of the factor module for which a presentation was computed by the last call of JBasis, as explained in JHilbertSeries.
- Note, ***JDimension***()-1 equals the degree of JHilbertPolynomial().

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x*y+y*z+z*x, x*y*z-1];
```

$$L := [xy+yz+zx, xyz-1]$$

```
> JBasis(L, var);
```

$$[xy+yz+zx, yz^2+z^2x+1, z^2y^2+y+z]$$

```
> JTabVar();
```

$$[xy+yz+zx, [x, y, z], xy]$$
$$[yz^2+z^2x+1, [x, *, z], z^2x]$$
$$[z^2y^2+y+z, [*, y, z], z^2y^2]$$

```
> JHP();
```

$$6s-3$$

```
> JDimension();
```

$$1$$

```
> JMultiplicity();
```

$$6$$

```
> JHilbertPolynomial();
```

$$6$$

```
> JHilbertSeries();
```

$$1+3s+5s^2+6s^3+\frac{6s^4}{1-s}$$

Example 2:

```
> var := [Dx,x,Dy,y];
```

$$var := [Dx, x, Dy, y]$$

```
> ops := [weyl(Dx,x),weyl(Dy,y)];
```

$$ops := [\mathrm{weyl}(Dx, x), \mathrm{weyl}(Dy, y)]$$

```
> L := [[x*Dx, -y*Dy], [x*Dy+y*Dx, 0]];
```

$$L := [[Dx\,x, -y\,Dy], [x\,Dy+y\,Dx, 0]]$$

```
> JBasis(L, var, ops);
```

$$[[x\,Dy+y\,Dx, 0], [Dx\,x, -y\,Dy], [y\,Dx^2+Dy, y\,Dy^2+Dy], [Dx^2\,x+Dx, -y\,Dy\,Dx], [-2\,y\,Dx, y^2\,Dy\,Dx+x\,Dy+y\,Dy^2\,x]]$$

```
> JTabVar();
```

$$[[x\,Dy+y\,Dx, 0], [*, x, Dy, y], [x\,Dy, 1]]$$

$$[[Dx\,x, -y\,Dy], [*, x, Dy, y], [Dx\,x, 1]]$$

$$[[y\,Dx^2 + Dy, y\,Dy^2 + Dy], [Dx, *, Dy, y], [y\,Dx^2, 1]]$$

$$[[Dx^2\,x + Dx, -y\,Dy\,Dx], [Dx, x, Dy, y], [Dx^2\,x, 1]]$$

$$[[-2\,y\,Dx, y^2\,Dy\,Dx + x\,Dy + y\,Dy^2\,x], [Dx, x, Dy, y], [y\,Dy^2\,x, 2]]$$

```
> JHP();
```

$$\frac{13}{3}s + 3s^2 + \frac{2}{3}s^3 + 2$$

```
> JDimension();
```

$$3$$

```
> JMultiplicity();
```

$$4$$

```
> JHilbertPolynomial();
```

$$4s + 2s^2 + 2$$

```
> JHilbertSeries();
```

$$2 + 8s + 18s^2 + 32s^3 + s^4\left(\frac{32}{1-s} + \frac{14}{(1-s)^2} + \frac{4}{(1-s)^3}\right)$$

See Also:

JBasis, JTabVar, JMultiplicity, JHilbertSeries, JHilbertPolynomial, JHP, JHilbertFunction, JHF, JIndexRegularity, JCartanCharacter.

JanetOre[JDirectSum] - form the matrix whose rows define the direct sum of given submodules of free left modules over an Ore algebra

Calling Sequence:

JDirectSum(L1, L2, ...)

Parameters:

`L1, L2` - lists (or matrices) of generators of the submodule

Description:

- ***JDirectSum*** returns a matrix whose rows form a generating set of the direct sum of given submodules of free left modules over an Ore algebra.
- The entries of **`L1`**, **`L2`**, ... are polynomials in case of left ideals, i. e. submodules of the free left module of rank one, or lists of polynomials of the same length, representing elements of a free left module of tuples over an Ore algebra. If **`L1`** or **`L2`**, ... is a matrix, then the generators are extracted from the rows of **`L1`** resp. **`L2`**, etc..
- The result is a polynomial block-diagonal matrix.

Examples:

```
> with(JanetOre):
> L1 := [[x^2, y^2+1], [x*y, x*z^2]];
```

$$L1 := [[x^2, y^2+1], [xy, xz^2]]$$

```
> L2 := [[x+y+z, 0, x], [x^3-1, y^2-z^2, 0]];
```

$$L2 := [[x+y+z, 0, x], [x^3-1, y^2-z^2, 0]]$$

```
> JDirectSum(L1, L2);
```

$$\begin{bmatrix} x^2 & y^2+1 & 0 & 0 & 0 \\ xy & xz^2 & 0 & 0 & 0 \\ 0 & 0 & x+y+z & 0 & x \\ 0 & 0 & x^3-1 & y^2-z^2 & 0 \end{bmatrix}$$

See Also:

JBasis, JTabVar, JInvReduce, JSyzygies, JSyzygyModule, JResolution, JCokernel, JSyzOp

JanetOre[JEulerChar] - return Euler characteristic of a factor module of a free module over a polynomial ring

Calling Sequence:

JEulerChar(L,var)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the polynomial ring)

Description:

- ***JEulerChar*** returns the Euler characteristic of the module *M* presented by **L** (i.e., the elements of **L** are considered as elements of a free module over the polynomial ring with indeterminates **`var`** of appropriate rank and *M* is the factor module of this free module modulo the submodule that is generated by the elements of **L**). The Euler characteristic of a finitely presented module *M* over a polynomial ring is defined as the alternating sum of the ranks of the free modules occurring in a free resolution of *M*. It is independent of the choice of the free resolution of *M*.
- The entries of **L** are polynomials in case of an ideal, i.e. a submodule of the free module of rank one, or lists of polynomials of length *m*, representing elements of the free module of *m*-tuples over the polynomial ring. If **L** is a matrix, then the generators are extracted from the rows of **L**.
- The result of ***JEulerChar*** is the non-negative integer obtained as alternating sum of the entries of the list of ranks returned by JResolutionDim (starting with positive sign for the last entry of this list of ranks and running backwards through this list). Since ***JEulerChar*** relies on the result of JResolutionDim, only one involutive basis computation is needed to obtain the Euler characteristic of *M* (instead of several involutive basis computations performed by JResolution).
- For more information about Janet bases and resolutions, see W. Plesken, D. Robertz, "Janet's approach to presentations and resolutions for polynomials and linear pdes", Archiv der Mathematik, 84(1), 2005, 22-37.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x+y+z, x*y+y*z+z*x, x*y*z-1];
```

$$L := [x+y+z,\ xy+yz+zx,\ xyz-1]$$

```
> InvolutiveBasis(L, var);
```

$$[x+y+z,\ y^2+yz+z^2,\ -1+z^3,\ -y+z^3\,y]$$

```
> PolTabVar();
```

$$[x+y+z,\ [x, y, z],\ x]$$
$$[y^2+yz+z^2,\ [*, y, z],\ y^2]$$
$$[-1+z^3,\ [*, *, z],\ z^3]$$
$$[-y+z^3\,y,\ [*, *, z],\ z^3\,y]$$

```
> PolResolutionDim(L, var);
```

$$[2, 5, 4, 1]$$

```
> PolEulerChar(L, var);
```

$$0$$

```
> PolResolution(L, var);
```

$$\left[\begin{bmatrix}1 & z+x & 1 & -y & 0\\ x & -z^2 & -y-z & -z^2 & -1+z^3\end{bmatrix}, \begin{bmatrix}0 & 1-z^3 & z^2 & y+z\\ 0 & 0 & y & -1\\ y-z^3 y & -1+z^3 & -z^2 & x\\ 1-z^3 & 0 & z+x & 1\\ -y^2-yz-z^2 & x+y+z & 0 & 0\end{bmatrix}, \begin{bmatrix}x+y+z\\ y^2+yz+z^2\\ -1+z^3\\ -y+z^3 y\end{bmatrix}\right]$$

Example 2:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [[x^2-y,y^2,0],[x,y,x]];
```

$$L := [[x^2-y, y^2, 0], [x, y, x]]$$

```
> PolResolutionDim(L, var);
```

$$[1, 3, 3]$$

```
> PolEulerChar(L, var);
```

$$1$$

```
> PolResolution(L, var);
```

$$\left[\begin{bmatrix}-1 & x & -y\end{bmatrix}, \begin{bmatrix}0 & -y^2 x+x^2 y-y^2 & x^3-xy\\ y & -y^2+xy & x^2\\ x & y & x\end{bmatrix}\right]$$

See Also:

JBasis, JBasisFast, JTabVar, JInvReduce, JInvReduceFast, JSyzygies, JSyzygyModule, JSyzygyModuleFast, JResolution.

JanetOre[JFactorModuleBasis] - return a vector space basis for the residue class module (or a generating function for it)

Calling Sequence:

JFactorModuleBasis(var,mode)

Parameters:

`var` - list of variables (of the Ore algebra)
`mode` - (optional) string specifying options

Description:

- ***JFactorModuleBasis*** returns a vector space basis for the residue class module of the free left module over the Ore algebra modulo the submodule generated by the Janet basis of the last call of JBasis, in case the factor module is finite dimensional as a vector space.
- If the factor module is infinite dimensional, a generating function is produced whose monomial summands are the representatives of the standard monomial basis vectors of the residue class module: A term of the form $m/((1-x_1)...(1-x_n))\ e_i$ stands for the residues of those vectors which, according to the geometric series expansion, are all left multiples of $m\ e_i$ by any monomial in the variables $x_1,...,x_n$. Here m stands for a monomial in the indeterminates **`var`** and e_i for the i-th standard basis vector of the free left module. See also the explanation of the option "G" below.
- **`var`** is the list of variables of the Ore algebra that was given as parameter to ***JBasis*** before.
- The optional argument **`mode`** is a string which may contain the letters "C", "G", "L" and "M".
- If the letter "G" is present in **`mode`**, but the letter "C" is not, then ***JFactorModuleBasis*** is forced to return a generating function as described above even if the factor module is finite dimensional (cf. Example 3 below).
- If the factor module is infinite dimensional and the letter "C" is contained in **`mode`**, then the numerators m of the resulting generating function are returned in a list (cf. Example 2 below). If additionally the letter "M" is present in **`mode`**, then the result of ***JFactorModuleBasis*** is a list of lists $[m, v]$, where m is the numerator as above and v is the list of multiplicative variables for the cone with vertex m.
- The presence of the letter "L" in **`mode`** only has an effect if JBasisFast has been called before. In this case ***JFactorModuleBasis*** computes its result using the output data of the last call of ***JBasisFast***. This means that in this case it is not necessary to update the internal data structure for the current involutive basis using AssertJBasis before calling ***JFactorModuleBasis*** (cf. Example 4 below). However, the user must be aware that the current involutive basis in Maple may then be different from the involutive basis which was computed by the last call of ***JBasisFast***, so that the results of ***JFactorModuleBasis*** with and without option "L" are in general different.
- The resulting basis contains monomials with coefficient 1. The list is sorted using degree reverse lexicographical ordering ("position over term" in the module case).

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [x^2, x*y^2, y^3];
```

$$L := [x^2, xy^2, y^3]$$

```
> JBasis(L, var);
```

$$[x^2, y^3, xy^2]$$

```
> JFactorModuleBasis(var);
```

$$[1, y, x, y^2, xy]$$

Note, the sum of the coefficients of the Hilbert series is the length of the basis for the residue class module:

```
> JHilbertSeries();
```

$$1+2s+2s^2$$

```
> JFactorModuleBasis(var, "G");
```

$$1+x+xy+y+y^2$$

Example 2:

Here is an example, where the factor module is infinite dimensional. ***JFactorModuleBasis*** returns the corresponding generating function:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [x*y];
```

$$L := [xy]$$

```
> JBasis(L, var);
```

$$[xy]$$

```
> JFactorModuleBasis(var);
```

$$\frac{1}{1-y}+\frac{x}{1-x}$$

```
> JFactorModuleBasis(var, "C");
```

$$[1, x]$$

Example 3:

```
> var := [D,x];
```

$$var := [\mathrm{D}, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x)]$$

```
> L := [[x*D-1, 0], [D^2, 0], [0, x*D-2], [0, D^3]];
```

$$L := [[x\mathrm{D}-1, 0], [\mathrm{D}^2, 0], [0, x\mathrm{D}-2], [0, \mathrm{D}^3]]$$

```
> JBasis(L, var, ops);
```

$$[[0, x\mathrm{D}-2], [x\mathrm{D}-1, 0], [\mathrm{D}^2, 0], [0, x\mathrm{D}^2-\mathrm{D}], [0, \mathrm{D}^3]]$$

```
> JFactorModuleBasis(var);
```

$$\left[\mathrm{D}+\frac{1}{1-x}, \mathrm{D}+\mathrm{D}^2+\frac{1}{1-x}\right]$$

```
> JHilbertSeries(t);
```

$$2+4t+3t^2+2\frac{t^3}{1-t}$$

```
> L := [[x, 0], [D^2, 0], [0, x], [0, D^3]];
```

$$L := [[x, 0], [\mathrm{D}^2, 0], [0, x], [0, \mathrm{D}^3]]$$

```
> JBasis(L, var, ops);
```

$$[[0, 1], [1, 0]]$$

```
> JFactorModuleBasis(var);
```

$$[\,]$$

```
> JFactorModuleBasis(var, "G");
```

$$0$$

Example 4:

```
> restart;
> with(JanetOre):
> JBasisFast([x*y], [x,y]);
```

$$[xy]$$

```
> JFactorModuleBasis([x,y], "L");
```

$$\frac{1}{1-y}+\frac{x}{1-x}$$

See Also:

JBasis, JTabVar, JSubmoduleBasis, JHilbertSeries, JWeightedHilbertSeries, JMinPoly, JRepres, JCoeffList

JanetOre[JGroebnerBasis] - return minimal Groebner basis of a submodule of a free left module over an Ore algebra

Calling Sequence:

JGroebnerBasis(L,var,ops,ord,mode,opt)

Parameters:

`L`	-	list (or matrix) of generators of the submodule
`var`	-	list of variables (of the Ore algebra)
`ops`	-	(optional) list of commutation rules for the variables of the Ore algebra
`ord`	-	(optional) change of monomial ordering
`mode`	-	(optional) string specifying options for the computation
`opt`	-	(optional) equation specifying options for the computation

Description:

- ***JGroebnerBasis*** returns the minimal Groebner basis with respect to a certain monomial ordering of a submodule of the free left module of *m*-tuples over the Ore algebra specified by **`var`** and **`ops`** which is given by the generators in **`L`**. The default ordering is degree reverse lexicographical. The leading coefficients in the resulting Groebner basis are normalized to 1.
- The Groebner basis is extracted from the involutive basis of the given submodule (which is, in general, a redundant Groebner basis because of the separation of the variables into multiplicative and non-multiplicative ones). Hence, ***JGroebnerBasis*** passes its arguments to JBasis and extracts the minimal Groebner basis from this result. Therefore, Janet's data (see JTabVar) contains information about the involutive basis of the given submodule.
- All parameters to ***JGroebnerBasis*** have the same meaning as in JBasis.
- The output is a list containing the Groebner basis for the submodule generated by **`L`** with respect to the chosen ordering.
- By means of the command JanetOreOptions one can also choose between two implementations of ***JGroebnerBasis***: "Maple" and "C++".

Examples:

```
> with(JanetOre):
> var := [D1,D2,x1,x2];
```

$$var := [D1, D2, x1, x2]$$

```
> ops := [weyl(D1,x1),weyl(D2,x2)];
```

$$ops := [\mathrm{weyl}(D1, x1), \mathrm{weyl}(D2, x2)]$$

```
> L := [x1*D1*D2^2-D2^2, D1^2*D2^2, x2*D2-1];
```

$$L := [x1\,D1\,D2^2 - D2^2, D1^2\,D2^2, x2\,D2 - 1]$$

```
> JBasis(L, var, ops);
```

$$[x2\,D2 - 1, -D1 + x2\,D2\,D1, x2\,D1\,D2^2, -D1^2 + x2\,D2\,D1^2, x1\,D1\,D2^2 - D2^2, D1^2\,D2^2]$$

```
> JGroebnerBasis(L, var, ops);
```

$$[x2\,D2 - 1, x1\,D1\,D2^2 - D2^2, D1^2\,D2^2]$$

```
> JTabVar();
```

$$[x2\,D2 - 1, [*, D2, x1, x2], x2\,D2]$$
$$[-D1 + x2\,D2\,D1, [*, *, x1, x2], x2\,D2\,D1]$$
$$[x2\,D1\,D2^2, [*, D2, *, x2], x2\,D1\,D2^2]$$
$$[-D1^2 + x2\,D2\,D1^2, [D1, *, x1, x2], x2\,D2\,D1^2]$$
$$[x1\,D1\,D2^2 - D2^2, [*, D2, x1, x2], x1\,D1\,D2^2]$$
$$[D1^2\,D2^2, [D1, D2, x1, x2], D1^2\,D2^2]$$

See Also:

JBasis, JBasisFast, JTabVar, JanetOreOptions, JLeadingMonomial, JGroebnerBasisFast

JanetOre[JGroebnerBasisFast] - return minimal Groebner basis of a submodule of a free left module over an Ore algebra (C++ version)

Calling Sequence:

JGroebnerBasisFast(L,var,ops,ord,mode,opt)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`ord` - (optional) change of polynomial ordering
`mode` - (optional) string specifying options for the computation
`opt` - (optional) equation specifying options for the computation

Description:

- ***JGroebnerBasisFast*** computes the minimal Groebner basis with respect to a certain monomial ordering of a submodule of the free left module of *m*-tuples over the Ore algebra specified by **`var`** and **`ops`** which is given by the generators in **`L`**. Here JBasisFast is used instead of JBasis (cf. JGroebnerBasis). Up to now, only the algorithm for the degree reverse lexicographical ordering (i.e. **`ord`** is 2 or 4) is implemented in C++. If ***JGroebnerBasisFast*** is called choosing a different monomial ordering, then JBasis is applied instead of JBasisFast.
- All parameters to ***JGroebnerBasisFast*** have the same meaning as in JGroebnerBasis and JBasis.
- The advantage of this command is clearly the enormous speed up. (Note, you cannot interrupt this command from within Maple; you have to kill the process "JBOre" instead.)
- Using the option "C++" of JanetOreOptions, the command ***JGroebnerBasis*** is replaced by ***JGroebnerBasisFast*** for the current Maple session.

Examples:

```
> with(JanetOre):
> var := [D1,D2,x1,x2];
```

$$var := [D1, D2, x1, x2]$$

```
> ops := [weyl(D1,x1),weyl(D2,x2)];
```

$$ops := [\mathrm{weyl}(D1, x1), \mathrm{weyl}(D2, x2)]$$

```
> L := [x1*D1*D2^2-D2^2, D1^2*D2^2, x2*D2-1];
```

$$L := [x1\,D1\,D2^2 - D2^2, D1^2\,D2^2, x2\,D2 - 1]$$

```
> JBasisFast(L, var, ops);
```

$$[x2\,D2 - 1, x2\,D2\,D1 - D1, x2\,D1\,D2^2, x2\,D2\,D1^2 - D1^2, x1\,D1\,D2^2 - D2^2, D1^2\,D2^2]$$

```
> JGroebnerBasisFast(L, var, ops);
```

$$[x2\,D2 - 1, x1\,D1\,D2^2 - D2^2, D1^2\,D2^2]$$

See Also:

JBasis, JBasisFast, JanetOreOptions, JTabVar, JFactorModuleBasis, JInvReduce, JInvReduceFast, JGroebnerBasis, JSyzygyModule, JSyzygyModuleFast.

JanetOre[JHF] - compute the filtered Hilbert function for the factor module

Calling Sequence:

JHF(p)
JHF()

Parameters:

p - " " (empty string) or natural number

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JHilbertSeries. Then ***JHF*(p)** returns $\sum_{i=0}^{p} d_i$ for natural numbers **p** and prints the corresponding function in case **p** is the empty string.
- JHilbertFunction, of which the present command is a summed up version and which refers to the induced grading rather than to the filtration, must not be confused with ***JHF()***.
- ***JHF***() returns a function expecting one parameter **p** which computes ***JHF*(p)**.

Examples:

```
> with(JanetOre):
> var := [x,y,z,v];
```

$$var := [x, y, z, v]$$

```
> L := [x*y+y*z+z*x, x*y*z-v];
```

$$L := [xy+yz+zx, xyz-v]$$

```
> B := JBasis(L, var);
```

$$B := [xy+yz+zx, yz^2+z^2x+v, z^2y^2+vy+zv]$$

```
> JHilbertSeries();
```

$$1+4s+9s^2+15s^3+s^4\left(15\frac{1}{1-s}+6\frac{1}{(1-s)^2}\right)$$

```
> f := JHF();
```

$$f := JanetOre/JHF$$

```
> f(2);
```

14

```
> f(20);
```

1202

```
> JHF(20);
```

1202

```
> JHF("");
s = 0: 1
s = 1: 5
s = 2: 14
s = 3: 29
s >= 4: 3*s^2+2
> JHP();
```

$$3s^2+2$$

```
> JHilbertFunction("");
Dim(M.0) = 1
Dim(M.1) = 4
Dim(M.2) = 9
Dim(M.3) = 15
Dim(M.s) = -3+6*s, for s >= 4
```

See Also:

JBasis, JTabVar, JHilbertSeries, JHilbertFunction, JHilbertPolynomial, JHP, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHP, JSubmoduleHF.

JanetOre[JHP] - compute the filtered Hilbert polynomial for the factor module

Calling Sequence:

JHP(p)
JHP()

Parameters:

p - natural number or name of an indeterminate

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JHilbertSeries. Then ***JHP*(p)** returns $\sum_{i=0}^{p} d_i$ for natural numbers **p** larger than the index of regularity and the corresponding polynomial in **p** inducing this function in case **p** is an indeterminate. The index of regularity can be computed by JIndexRegularity. Note, all this information can also be extracted from the command JHF.
- ***JHP()*** returns the above polynomial with ´s´ as the default name of the indeterminate. ´s´ cannot be affected by a subs command.
- JHilbertPolynomial, of which the present command is a summed up version and which refers to the induced grading rather than to the filtration, must not be confused with ***JHP()***.

Examples:

```
> with(JanetOre):
> var := [x,y,z,v];
```

$$var := [x, y, z, v]$$

```
> L := [x*y+y*z+z*x, x*y*z-v];
```

$$L := [xy+yz+zx, xyz-v]$$

```
> B := JBasis(L, var);
```

$$B := [xy+yz+zx, yz^2+z^2x+v, z^2y^2+vy+zv]$$

```
> JHilbertSeries();
```

$$1+4s+9s^2+15s^3+s^4\left(15\frac{1}{1-s}+6\frac{1}{(1-s)^2}\right)$$

```
> JHP();
```

$$3s^2+2$$

```
> JHP(20);
```

$$1202$$

```
> JHF("");
s = 0: 1
s = 1: 5
s = 2: 14
s = 3: 29
s >= 4: 3*s^2+2
> JHP(lambda);
```

$$3\lambda^2+2$$

```
> subs(lambda=3, %);
```

$$29$$

See Also:

JBasis, JTabVar, JHilbertSeries, JHilbertPolynomial, JHilbertFunction, JHF, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHP, JSubmoduleHF.

JanetOre[JHilbertFunction] - compute the graded Hilbert function for the factor module

Calling Sequence:

JHilbertFunction(p)
JHilbertFunction()

Parameters:

p - " " (empty string) or natural number

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JHilbertSeries. Then ***JHilbertFunction*(p)** returns d_p in case **p** is a natural number and prints the function $s \to d_s$ in case **p** is the empty string.
- JHF, which is a summed up version of the present command and refers to the filtration rather than to the induced grading, must not be confused with ***JHilbertFunction***.
- ***PolHilbertFunction***() returns a function expecting one parameter **p** which computes ***JHilbertFunction*(p)**.

Examples:

```
> with(JanetOre):
> var := [x,y,z,v];
```

$$var := [x, y, z, v]$$

```
> L := [x*y+y*z+z*x, x*y*z-v^3];
```

$$L := [xy+yz+zx, xyz-v^3]$$

```
> B := JBasis(L, var);
```

$$B := [xy+yz+zx, yz^2+z^2x+v^3, z^2y^2+v^3y+zv^3]$$

```
> JHilbertSeries();
```

$$1+4s+9s^2+15s^3+s^4\left(15\frac{1}{1-s}+6\frac{1}{(1-s)^2}\right)$$

```
> f := JHilbertFunction();
```

$$f := JanetOre/JHilbertFunction$$

```
> f(2);
```

$$9$$

```
> f(20);
```

$$117$$

```
> JHilbertFunction(20);
```

$$117$$

```
> JHilbertFunction("");
Dim(M.0) = 1
Dim(M.1) = 4
Dim(M.2) = 9
Dim(M.3) = 15
Dim(M.s) = -3+6*s, for s >= 4
```

See Also:

JBasis, JTabVar, JHilbertSeries, JHF, JHilbertPolynomial, JHP, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHP, JSubmoduleHF.

JanetOre[JHilbertPolynomial] - compute the graded Hilbert polynomial for the factor module

Calling Sequence:

JHilbertPolynomial(p)
JHilbertPolynomial()

Parameters:

p - natural number or name of an indeterminate

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JHilbertSeries. Then ***JHilbertPolynomial*(p)** returns d_p in case **p** is a natural number greater than or equal to the index of regularity. If **p** is the name of an indeterminate, then the Hilbert polynomial in **p** is returned. The information is derived from the last call of JBasis. Note, this same information can be extracted from the command JHilbertFunction.
- JHP, which is a summed up version of the present command and refers to the filtration rather than to the induced grading, must not be confused with ***JHilbertPolynomial***.
- ***JHilbertPolynomial***() returns the graded Hilbert polynomial of the associated graded module of the residue class module modulo the left module whose involutive basis has been computed last by JBasis. Note, this same information can be extracted from the command JHilbertFunction.
- As optional parameter a name **p** for the indeterminate of the Hilbert polynomial can be given. The default name of the indeterminate is 's'. It will not be affected by a subs command.

Examples:

```
> with(JanetOre):
> var := [x,y,z,v];
```

$$var := [x, y, z, v]$$

```
> L := [x*y+y*z+z*x, x*y*z-v^3];
```

$$L := [xy+yz+zx, xyz-v^3]$$

```
> B := JBasis(L, var);
```

$$B := [xy+yz+zx, yz^2+z^2x+v^3, z^2y^2+v^3y+zv^3]$$

```
> JHilbertSeries();
```

$$1+4s+9s^2+15s^3+s^4\left(15\frac{1}{1-s}+6\frac{1}{(1-s)^2}\right)$$

```
> JHilbertPolynomial();
```

$$-3+6s$$

```
> JHilbertPolynomial(6);
```

$$33$$

```
> JHP(6);
```

$$110$$

```
> JHilbertFunction("");
Dim(M.0) = 1
Dim(M.1) = 4
Dim(M.2) = 9
Dim(M.3) = 15
Dim(M.s) = -3+6*s, for s >= 4
> JHilbertPolynomial(lambda);
```

$$-3+6\lambda$$

```
> subs(lambda=3, %);
```

$$15$$

See Also:

JBasis, JTabVar, JHilbertSeries, JHP, JHilbertFunction, JHF, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHP, JSubmoduleHF.

JanetOre[JHilbertSeries] - Hilbert series of the factor module presented by the last computed Janet basis

Calling Sequence:

JHilbertSeries(v)

Parameters:

v - (optional) name of the indeterminate (default: 's')

Description:

- ***JHilbertSeries*** returns the generating function counting - according to the standard degrees - the standard monomial basis vectors of the factor module *F* of the free left module of *m*-tuples over the Ore algebra modulo the submodule *M* generated by the Janet basis produced by the last call of JBasis.
- The Ore algebra defined by **var** and **ops** in the last call of ***JBasis*** has an (increasing) filtration *B* defined by the total degree. (In case of the Weyl algebras, this filtration is called the Bernstein filtration.) The graded algebra of this Ore algebra with respect to this filtration *B* is the commutative polynomial ring over the ground field with as many indeterminates as there are variables in **var**.
- For a non-zero element of the free left module of *m*-tuples over the Ore algebra, we define the total degree to be the maximal total degree of its non-zero entries. Then this free left module has an (increasing) *B*-filtration Γ defined by the total degree, i.e. left multiplication of elements of B_i to elements of Γ_j yields elements of Γ_{i+j}. The corresponding graded module over the graded algebra is then a direct sum of finite dimensional vector spaces over the ground field. The Hilbert series of this graded module is the formal power series whose *i*-th coefficient is the vector space dimension of the *i*-th graded piece of the module.
- ***JHilbertSeries*** returns the Hilbert series of the residue class module of the graded module of the free left module modulo the graded module of the submodule of leading terms of *M*, where *M* is the submodule generated by the Janet basis computed by the last call of JBasis. Note, this submodule, and therefore also its Hilbert series, depends on the term order chosen in the call of JBasis. The returned Hilbert series agrees with the generating function described above, since the standard bases for both the free module of *m*-tuples and the residue class module are represented by the same monomial elements in the module of *m*-tuples.
- The output is the corresponding Hilbert series $\sum_{i=0}^{\infty} d_i v^i$, where the d_i are the dimensions of the homogeneous components of the graded residue class module considered above.
- The default name of the indeterminate **v** is 's'. It cannot be affected by a subs command.
- Note, if one has assigned non-standard degrees to the variables or to the standard basis vectors, the command ***JHilbertSeries*** will proceed from the leading terms computed by JBasis but then reassign the degrees 1 for the variables and 0 for the basis vectors. This is usually not what one wants: To proceed with the introduced grading one has to work with JWeightedHilbertSeries.

Examples:

```
> with(JanetOre):
```

In the case of an ideal of a commutative polynomial ring, the Hilbert series is simply the Hilbert series of the graded ring given by the polynomial ring modulo the ideal of the leading monomials as listed in the last items of the tuples in the output of JTabVar.

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x+y^2, y+z^2];
```

$$L := [x+y^2, y+z^2]$$

```
> JBasis(L, var);
```

$$[y+z^2, x+y^2, z^2\,y-x]$$

```
> JTabVar();
```

$$[y+z^2, [x, *, z], z^2]$$
$$[x+y^2, [x, y, z], y^2]$$
$$[z^2\,y-x, [x, *, z], z^2\,y]$$

```
> JHilbertSeries();
```

$$1+3s+4s^2+4\frac{s^3}{1-s}$$

Note, the Hilbert series changes, if one works with the pure lexicographical order:

```
> JBasis(L, var, 1);
```

$$[y+z^2, x+z^4]$$

```
> JHilbertSeries();
```

$$1+\frac{s}{1-s}$$

```
> JTabVar();
```

$$[y+z^2, [*, y, z], y]$$
$$[x+z^4, [x, y, z], x]$$

Example for a module of tuples:

```
> L2 := [[x,-y],[y,x]];
```

$$L2 := [[x, -y], [y, x]]$$

```
> JBasis(L2, [x,y]);
```

$$[[y, x], [x, -y]]$$

```
> JHilbertSeries(lambda);
```

$$2+2\frac{\lambda}{1-\lambda}$$

```
> JTabVar();
```

$$[[y, x], [x, y], [x, 2]]$$
$$[[x, -y], [x, y], [x, 1]]$$

Example for the Weyl algebra:

```
> var := [D,x];
```

$$var := [D, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(D, x)]$$

```
> L := [x*D-1, D^2];
```

$$L := [xD-1, D^2]$$

```
> JBasis(L, var, ops);
```

$$[xD-1, D^2]$$

```
> JHilbertSeries(lambda);
```

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

See Also:

JBasis, JTabVar, JFactorModuleBasis, JHilbertPolynomial, JHP, JHilbertFunction, JHF, JIndexRegularity, JCartanCharacter, JWeightedHilbertSeries, JSubmoduleBasis, JSubmoduleHilbertSeries.

JanetOre[JIndexRegularity] - return index of regularity of the graded module of a residue class module

Calling Sequence:

JIndexRegularity()

Parameters:

- - none (assumes that the involutive basis has been computed before)

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JHilbertSeries. Then ***JIndexRegularity*(p)** returns the index of regularity *r*, i. e. *r* is the biggest integer for which the (graded) Hilbert polynomial and the (graded) Hilbert function give different values, cf. JHilbertPolynomial, JHilbertFunction, i. e. the smallest *r* such that the filtered Hilbert function JHF and the filtered Hilbert polynomial JHP agree on all integers greater or equal to *r*.
- The command refers to the last call of JBasis.

Examples:

```
> with(JanetOre):
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x*y+y*z+z*x, x*y*z-1];
```

$$L := [xy + yz + zx, xyz - 1]$$

```
> JBasis(L, var);
```

$$[xy + yz + zx, yz^2 + z^2 x + 1, z^2 y^2 + y + z]$$

```
> JIndexRegularity();
```

$$2$$

```
> JHilbertSeries(lambda);
```

$$1 + 3\lambda + 5\lambda^2 + 6\lambda^3 + 6\frac{\lambda^4}{1-\lambda}$$

```
> JTabVar();
```

$$[xy + yz + zx, [x, y, z], xy]$$
$$[yz^2 + z^2 x + 1, [x, *, z], z^2 x]$$
$$[z^2 y^2 + y + z, [*, y, z], z^2 y^2]$$

```
> JHilbertPolynomial();
```

$$6$$

```
> JHilbertFunction("");
Dim(M.0) = 1
Dim(M.1) = 3
Dim(M.2) = 5
Dim(M.3) = 6
Dim(M.s) = 6, for s >= 4
> JHP();
```

$$6s - 3$$

```
> JHF("");
s = 0: 1
s = 1: 4
s = 2: 9
s = 3: 15
s >= 4: 6*s-3
```

See Also:

JBasis, JTabVar, JHilbertPolynomial, JHP, JHilbertFunction, JHF, JCartanCharacter.

JanetOre[JIntersection] - intersect two submodules of a free left module over an Ore algebra

Calling Sequence:

JIntersection(L1,L2,var,ops)

Parameters:

`L1` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`L2` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the Ore algebra
`ops` - (optional) list of commutation rules for the variables of the Ore algebra

Description:

- ***JIntersection*** computes a Janet basis of the intersection of the submodules generated by **`L1`** and **`L2`** in a free left module of tuples over the Ore algebra specified by **`var`** and **`ops`**.
- The entries of **`L1`** and **`L2`** are polynomials in **`var`** in case of left ideals, i. e. submodules of the free left module of rank one, or lists of polynomials in **`var`** of length *m*, representing elements of the free left module of *m*-tuples over the Ore algebra. In the latter case, the lists in **`L1`** and **`L2`** must be of the same length. If **`L1`** or **`L2`** is a matrix, then the generators are extracted from the rows of **`L1`** resp. **`L2`**.
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- The result of ***JIntersection*** is a list of polynomials or a list of lists of polynomials according to the structure of the input.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L1 := [x^2+y^2-1]; L2 := [x-y];
```

$$L1 := [x^2 + y^2 - 1]$$

$$L2 := [x - y]$$

```
> JIntersection(L1, L2, var);
```

$$[-y x^2 - y^3 + y + x^3 + y^2 x - x]$$

```
> factor(%[1]);
```

$$(x - y)(x^2 + y^2 - 1)$$

Example 2:

```
> var := [D,x];
```

$$var := [\mathrm{D}, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x)]$$

```
> L1 := [D^3, x*D-2];
```

$$L1 := [\mathrm{D}^3, x\,\mathrm{D} - 2]$$

```
> JBasis(L1, var, ops);
```

$$[x\,\mathrm{D} - 2, x\,\mathrm{D}^2 - \mathrm{D}, \mathrm{D}^3]$$

```
> L2 := [D^7, x*D-6];
```

$$L2 := [\mathrm{D}^7, x\,\mathrm{D} - 6]$$

```
> JBasis(L2, var, ops);
```

$$[x\,\mathrm{D} - 6, x\,\mathrm{D}^2 - 5\,\mathrm{D}, x\,\mathrm{D}^3 - 4\,\mathrm{D}^2, x\,\mathrm{D}^4 - 3\,\mathrm{D}^3, x\,\mathrm{D}^5 - 2\,\mathrm{D}^4, x\,\mathrm{D}^6 - \mathrm{D}^5, \mathrm{D}^7]$$

```
> JIntersection(L1, L2, var, ops);
```

$$[-7\,x\,\mathrm{D}+12+x^2\,\mathrm{D}^2,\ x^2\,\mathrm{D}^3-5\,x\,\mathrm{D}^2+5\,\mathrm{D},\ x\,\mathrm{D}^4-3\,\mathrm{D}^3,\ x\,\mathrm{D}^5-2\,\mathrm{D}^4,\ x\,\mathrm{D}^6-\mathrm{D}^5,\ \mathrm{D}^7]$$

Example 3:

```
> var := [D,x];
```

$$var := [\mathrm{D}, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x)]$$

```
> L1 := [[D, x], [x, 0]];
```

$$L1 := [[\mathrm{D}, x], [x, 0]]$$

```
> L2 := [[D, x*D]];
```

$$L2 := [[\mathrm{D}, x\,\mathrm{D}]]$$

```
> J := JIntersection(L1, L2, var, ops);
```

$$J := \left[[x^3\,\mathrm{D}, x^4\,\mathrm{D}], \left[x^2\,\mathrm{D}^2-\frac{1}{3}x^2\,\mathrm{D}+3\,x\,\mathrm{D}, -\frac{1}{3}x^3\,\mathrm{D}+4\,x^2\,\mathrm{D}+x^3\,\mathrm{D}^2\right]\right]$$

See Also:

JBasis, JInvReduce, JHilbertSeries, JSyzygies, JResolution, JLeftInverse, JRightInverse, JKernel, JSum, JDirectSum, JSubFactor, JSyzOp.

JanetOre[JInvReduce] - return the normal form with respect to a Janet basis

Calling Sequence:

JInvReduce(f,B,var,ops,ord,mode)

Parameters:

`f` - (tuple of) polynomial(s) (or list of such) to be reduced
`B` - Janet basis
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`ord` - (optional) type of monomial ordering
`mode` - (optional) string specifying options for the computation

Description:

- ***JInvReduce*** returns the normal form representing the residue class of **`f`** modulo the submodule of the free left module of *m*-tuples generated by the Janet basis **`B`**. This is done by involutive reduction. Note, if *m*=1, brackets can be omitted: one deals with a left ideal in an Ore algebra. If **`f`** is a list of (tuples of) polynomials, then the list of the corresponding normal forms is returned.
- The Janet basis **`B`** is given as a list of lists of polynomials in **`var`** in the module case and as a list of polynomials in the ideal case. Note, the program does not check whether **`B`** is a Janet basis with respect to **`var`** and **`ord`**. (Changing **`ord`** is not so critical, however, changing the ordering in **`var`** can result in wrong answers.)
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- As optional fifth parameter the values 1 to 4 are accepted which might affect the sequential order in which the reduction steps are performed. It does not affect the final coset representative. If **`ord`** = 1, highest terms with respect to the pure lexicographical ordering are reduced first, even if the Janet basis is taken with respect to degree reverse lexicographical ordering. In case **`ord`** = 2 the default degree reverse lexicographical order is taken. The values 3 and 4 select pure lexicographical ordering and degree reverse lexicographical ordering resp., but change from "position over term" order to "term over position" order.
- If JBasis was called with user defined degrees for variables and / or standard basis vectors, the corresponding parameter **`var`** has to be specified here in the same manner.
- If the letter "C" is present in **`mode`**, then ***JInvReduce*** additionally returns the coefficients of the elements subtracted from the input to obtain the normal form representative (remainder) with respect to the Janet basis. (For a more comfortable way of using this option, see JCoeff)
- If the letter "S" is present in **`mode`**, the program uses simplify instead of expand in the normal form procedure. If the polynomials in the input **`B`** contain nonrational coefficients, more precisely, if the ground field contains algebraic elements over the rationals (RootOf), then ***simplify*** is used instead of ***expand*** automatically.
- If **`B`** is a Janet basis with right hand sides (cf. JBasis), one can specify a right hand side for **`f`** in order to let ***JInvReduce*** perform any operation on both left and right hand side. For instance the input **`f=f`** is turned into the equation of the normal form representative (remainder) on the left hand side and **`f`** minus an explicit linear combination of the right hand sides of the elements of the Janet basis corresponding to the reduction. Usually the right hand sides of the Janet basis will express the elements of the Janet basis in term of the original generators. Therefore the right hand side of the equation will then also express the reducing element in terms of the original module generators.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,x];
```

$$var := [D, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x)]$$

```
> L := [D^2+x*D-1, 3*D^2];
```

$$L := [\mathrm{D}^2 + x\mathrm{D} - 1, 3\,\mathrm{D}^2]$$

```
> B := JBasis(L, var, ops);
```

$$B := [x\mathrm{D} - 1, \mathrm{D}^2]$$

```
> JInvReduce(x^2*D, B, var, ops);
```

$$x$$

```
> f := x*D^2-x^3*D-1;
```

$$f := x\mathrm{D}^2 - x^3\,\mathrm{D} - 1$$

```
> JInvReduce(f, B, var, ops);
```

$$-x^2 - 1$$

How can the remainder $-x^2 - 1$ be expressed as linear combination in **f** and the basis **B**?

```
> JInvReduce(f, B, var, ops, "C");
```

$$[-x^2 - 1, [-x^2, x]]$$

```
> f - expand(JMult(-x^2, B[1], var, ops) + JMult(x, B[2], var, ops));
```

$$-x^2 - 1$$

If one wants the coefficients with respect to the original generators, one has to give them names as follows:

```
> L1 := [L[1]=a, L[2]=b];
```

$$L1 := [\mathrm{D}^2 + x\mathrm{D} - 1 = a, 3\,\mathrm{D}^2 = b]$$

```
> B1 := JBasis(L1, var, ops);
```

$$B1 := \left[x\mathrm{D} - 1 = -\frac{1}{3}b + a, \mathrm{D}^2 = \frac{1}{3}b\right]$$

```
> f;
```

$$x\mathrm{D}^2 - x^3\,\mathrm{D} - 1$$

```
> JInvReduce(f=f, B1, var);
```

$$-x^2 - 1 = -\frac{1}{3}x\,b + x\mathrm{D}^2 - x^3\,\mathrm{D} + x^2\left(-\frac{1}{3}b + a\right) - 1$$

A list of polynomials can be reduced in one step:

```
> JInvReduce([x^2*D, -1, 7*x^3*D], B1, var);
```

$$[x, -1, 7x^2]$$

Example 2: A sample calculation for modules over the polynomial ring Q[*x*,*y*]:

```
> L2 := [[x^2-1, 0], [x*y, x*y], [0, y^2-1]];
```

$$L2 := [[x^2 - 1, 0], [xy, xy], [0, y^2 - 1]]$$

```
> B2 := JBasis(L2, [x,y]);
```

$$B2 := [[0, y^2 - 1], [xy, xy], [y^2, x^2], [x^2 - 1, 0], [y^3 - y, 0], [0, y^2\,x - x]]$$

```
> JInvReduce([x*y^3,0], B2, [x,y]);
```

$$[0, -xy]$$

Example 3: Using transcendental elements to express an element in terms of the generators:

```
> L3 := [x+y-a, x^2+y^2-b];
```

$$L3 := [x + y - a, x^2 + y^2 - b]$$

```
> B3 := JBasis(L3, [x,y]);
```

$$B3 := \left[x + y - a, y^2 - y\,a + \frac{1}{2}a^2 - \frac{1}{2}b\right]$$

```
> JInvReduce(x*y, B3, [x,y]);
```

$$\frac{1}{2}a^2 - \frac{1}{2}b$$

Note this only works well because $x + y$ and $x^2 + y^2$ are algebraically independent. In case of algebraically dependent generators division by zero might occur. This can be overcome by using equations as in Example 1 above.

Example 4:

```
> L4 := [x^2-y^3, x^4+y^6];
```

$$L4 := [x^2 - y^3, x^4 + y^6]$$

```
> B4 := JBasis(L4, [x=3,y=2]);
```

$$B4 := [x^2 - y^3, y^6, xy^6]$$

```
> JInvReduce(x^4, B4, [x=3,y=2]);
```

$$0$$

See Also:

JBasis, JBasisFast, JTabVar, JInvReduceFast, JFactorModuleBasis, JCoeff.

JanetOre[JInvReduceFast] - return the normal form with respect to a Janet basis (C++ version)

Calling Sequence:

JInvReduceFast(f,B,var,ops,ord,mode,opt)

Parameters:

`f`	-	(tuple of) polynomial(s) (or list of such) to be reduced
`B`	-	Janet basis
`var`	-	list of variables (of the Ore algebra)
`ops`	-	(optional) list of commutation rules for the variables of the Ore algebra
`ord`	-	(optional) type of monomial ordering
`mode`	-	(optional) string specifying options for the computation
`opt`	-	(optional) equation specifying options for the computation

Description:

- ***JInvReduceFast*** invokes the C++ version of the command JInvReduce. Up to now, only the algorithm for the standard setting (degree reverse lexicographical ordering (i.e. **`ord`** is 2 or 4) and default degrees) is implemented in C++. If ***JInvReduceFast*** is called with a non-standard option, then JInvReduce is applied internally to the same data.
- The parameter **`B`** should be the result of JBasisFast. If this is not the case, JBasisFast is applied to **`B`** before starting the involutive reductions. (See, however, the description of the option "L" below.)
- The parameters **`var`**, **`ops`**, **`ord`** have the same meaning as in JInvReduce.
- The advantage of this command is clearly the enormous speed up. (Note, you cannot interrupt this command from within Maple; you have to kill the process "JBOre" instead.)
- If the letter "C" is given in **`mode`**, then ***JInvReduceFast*** additionally returns the coefficients of the elements subtracted from the input to obtain the normal form representative (remainder) with respect to the Janet basis.
- If the letter "L" is present in **`mode`**, then ***JInvReduceFast*** does *not* check whether the given involutive basis **`B`** equals the one which was computed by the last call of JBasisFast. This option should speed up the repetitive use of ***JInvReduceFast***. Note that, even if the computations of ***JInvReduceFast*** rely upon the basis computed by the C++ program, the parameter **`B`** must match this basis, since certain data (e.g. the number of entries of the tuples in the module case) are determined from **`B`**.
- The only possible left hand side of the optional equation **`opt`** is the string "char". If JBasisFast has been run before using the option "char"=*c*, then this option must also be given to ***JInvReduceFast*** in order to perform involutive reductions in characteristic *c* (cf. Example 3).
- Using the option "C++" of JanetOreOptions, the command ***JInvReduce*** is replaced by ***JInvReduceFast*** for the current Maple session (which has the corresponding effect on all Maple procedures that call ***JInvReduce***).

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,D,delta];
```

$$var := [x, D, \delta]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(D, x), \mathrm{shift}(\delta, x)]$$

```
> L1 := [x*D-2, D^3+delta^3];
```

$$L1 := [xD - 2, D^3 + \delta^3]$$

```
> J := JBasis(L1, var, ops);
```

$$J := [xD - 2, \delta^3, D^3, \delta^3 D, \delta^3 x, \delta^3 D^2]$$

```
> JInvReduceFast(D^4+1, J, var, ops);
```

$$1$$

```
> JInvReduceFast(x*D^2, J, var, ops);
```

$$D$$

Example 2:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x+y+z=[1,0,0], x*y+y*z+z*x=[0,1,0], x*y*z-1=[0,0,1]];
```

$$L := [x+y+z=[1, 0, 0], xy+yz+zx=[0, 1, 0], xyz-1=[0, 0, 1]]$$

```
> B := JBasisFast(L, var);
```

$$B := [x+y+z=[1, 0, 0], y^2+yz+z^2=[y+z, -1, 0], z^3-1=[z^2, -z, 1], yz^3-y=[yz^2, -yz, y]]$$

```
> JInvReduceFast(z^4=[0,0,0], B, var);
```

$$z=[-z^3, z^2, -z]$$

```
> JInvReduceFast(z^4, B, var);
```

$$z$$

Example 3:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x+2*y+3*z, x*y+2*y*z+3*z*x, x*y*z-1];
```

$$L := [x+2y+3z, xy+2yz+3zx, xyz-1]$$

```
> B := JBasisFast(L, var, "char"=7);
```

$$B := [x+2y+3z, y^2+z^2, yz^2+4z^3+5, z^4+3y+2z]$$

```
> JInvReduceFast(x^3, B, var, "char"=7);
```

$$4z^3+6$$

See Also:

JBasisFast, JBasis, JanetOreOptions, JTabVar, JFactorModuleBasis, JInvReduce, JHilbertSeries, JSyzygies, JSyzygyModule, JSyzygyModuleFast.

JanetOre[JLeadingMonomial] - determine leading monomial(s) of a (list of) polynomial(s)

Calling Sequence:

JLeadingMonomial(L,var,ops,ord,mode)

Parameters:

`L`	-	polynomial in **`var`** or list of polynomials in **`var`** representing element(s) of a free left module over an Ore algebra
`var`	-	list of variables (of the Ore algebra)
`ops`	-	(optional) list of commutation rules for the variables of the Ore algebra
`ord`	-	(optional) change of monomial ordering
`mode`	-	(optional) string specifying the type of information to be returned

Description:

- ***JLeadingMonomial*** returns the leading monomial of **`L`** with respect to a certain monomial ordering, if **`L`** is a polynomial. If **`L`** is a list of polynomials, ***JLeadingMonomial*** returns the list of the respective leading monomials. The default monomial ordering is the degree reverse lexicographical ordering ("term over position" in the case of tuples). The monomial ordering is determined by the optional parameter **`ord`**.
- For a description of all possible values of the parameters **`var`**, **`ops`** and **`ord`** see the corresponding explanations in JBasis.
- The optional parameter **`ops`** is accepted by ***JLeadingMonomial*** only for coherence to the other commands of the JanetOre package. For determining the leading monomial of a module element it is not needed, and if provided it is ignored.
- As optional fourth parameter **`mode`** a string consisting of letters "C" and "T" is accepted. If **`mode`** contains the letter "C", the leading monomials are returned with leading coefficients (i. e. leading terms). If **`mode`** contains "T", tuples of the same length as the tuples in **`L`** are returned (length 1 if **`L`** consists of polynomials), where the leading monomial is in the same component where it occurs in the corresponding element of **`L`** and the other components are zero.

Examples:

```
> with(JanetOre):
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [3*x*y*z+4*x*z^3, 2*y^2+7*x];
```

$$L := [3xyz + 4xz^3, 2y^2 + 7x]$$

```
> JLeadingMonomial(L, var);
```

$$[xz^3, y^2]$$

```
> JLeadingMonomial(L[1], var);
```

$$xz^3$$

```
> JLeadingMonomial(L, var, 1);
```

$$[xyz, x]$$

```
> JLeadingMonomial(L, var, 1, "C");
```

$$[3xyz, 7x]$$

Examples for elements of the free module of rank 2:

```
> L := [[x*y*z+x*z^3, y^2], [x^3,y]];
```

$$L := [[xyz + xz^3, y^2], [x^3, y]]$$

```
> JLeadingMonomial(L, var);
```

$$[xz^3, x^3]$$

Assign degrees to the variables:

```
> JLeadingMonomial(L, [x=1,y=3,z=1]);
```

$$[y^2, x^3]$$

Use "position over term" ordering:

```
> JLeadingMonomial(L, [x=1,y=3,z=1], 2);
```

$$[xyz, x^3]$$

Change the sequence of priority of the list entries:

```
> JLeadingMonomial(L, [x=1,y=3,z=1,2,1]);
```

$$[y^2, x^3]$$

Return leading monomials in tuples:

```
> JLeadingMonomial(L, [x=1,y=3,z=1,2,1], 4, "T");
```

$$[[0, y^2], [x^3, 0]]$$

Assign degrees to standard basis vectors:

```
> JLeadingMonomial([[x*y, y]], [x,y,1=0,2=2], 4);
```

$$[y]$$

See Also:

JBasis, JTabVar, JHas.

JanetOre[JLeftInverse] - compute left inverse of a matrix with entries in an Ore algebra

Calling Sequence:

JLeftInverse(M,var,ops)

Parameters:

`M` - matrix of polynomials in **`var`** or list of lists of the same length of polynomials in **`var`**
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra

Description:

- ***JLeftInverse*** computes (if possible) a left inverse of the matrix **`M`**, i.e. a matrix L whose entries are polynomials in **`var`** such that the product of L by **`M`** is the identity matrix.
- The first parameter **`M`** is expected to be a matrix whose entries are polynomials in the variables **`var`** or a list of lists of polynomials in **`var`**, where each list is of the same length. In the second case, ***JLeftInverse*** forms a matrix by taking the lists in **`M`** as rows and computes a left inverse of this matrix.
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- If no left inverse of **`M`** exists, ***JLeftInverse*** returns FAIL.
- If a left inverse L of **`M`** exists, ***JLeftInverse*** returns such an L as a matrix if **`M`** is a matrix, or returns the list of the rows of L if **`M`** is a list as explained above.
- Right inverses of polynomial matrices are computed by JRightInverse.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,x];
```

$$var := [\mathrm{D}, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x)]$$

```
> M := matrix([[x*D+1], [D^2]]);
```

$$M := \begin{bmatrix} x\mathrm{D}+1 \\ \mathrm{D}^2 \end{bmatrix}$$

```
> L := JLeftInverse(M, var, ops);
```

$$L := \begin{bmatrix} -\frac{1}{2}x\mathrm{D}+1 & \frac{1}{2}x^2 \end{bmatrix}$$

```
> JMult(L, M, var, ops);
```

$$[\ 1]$$

Example 2:

```
> var := [delta,t];
```

$$var := [\delta, t]$$

```
> ops := [shift(delta,t)];
```

$$ops := [\mathrm{shift}(\delta, t)]$$

```
> M := [[delta^2+1, 1], [t, delta+1], [delta+1, 0]];
```

$$M := [[\delta^2+1, 1], [t, \delta+1], [\delta+1, 0]]$$

```
> L := JLeftInverse(M, var, ops);
```

$$L := [[2\,\delta + 1 + \delta^2, -1 - \delta, t - \delta^3 - \delta^2 - \delta], [-4\,\delta - 1 - 2\,\delta^2, 2\,\delta + 2, -2\,t + 1 + 2\,\delta^2 + 2\,\delta^3 + \delta]]$$

```
> JMult(L, M, var, ops);
```

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

See Also:

JBasis, JTabVar, JInvReduce, JInvolution, JRightInverse, JAddRhs.

JanetOre[JMinPoly] - minimal polynomial of left multiplication by a ring element on a residue class module

Calling Sequence:

JMinPoly(m,B,var,ops,mode)

Parameters:

`m`	-	polynomial in **`var`**
`B`	-	Janet basis
`var`	-	list of variables (of the Ore algebra)
`ops`	-	(optional) list of commutation rules for the variables of the Ore algebra
`mode`	-	(optional) string or equation whose left hand side is a string

Description:

- ***JMinPoly*** returns the minimal polynomial for the left multiplication by **`m`** on the residue class module presented by the Janet basis **`B`** in case its degree does not exceed a certain positive integer. By default this integer is 30.
- **`var`** is the list of variables of the Ore algebra. (***JMinPoly*** does not check whether **`B`** is a Janet basis with respect to **`var`**.)
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- By default the result of ***JMinPoly*** is a polynomial in the indeterminate λ. The name of the indeterminate can be changed by the option described below.
- The optional parameter **`mode`** may occur repeatedly. It may be equal to the string "S" or to an equation whose left hand side is one of the following strings: "degree", "var", "subs".
- If **`mode`** equals "S", then ***JMinPoly*** uses simplify instead of expand in the normal form procedure.
- If **`mode`** is given as equation "degree"=*d*, where *d* is a positive integer, then the upper bound for the degree of the minimal polynomial to be computed is set to *d*.
- If **`mode`** equals "var"=*z*, where *z* is a name for an indeterminate, then the resulting minimal polynomial is returned as a polynomial in the variable *z*.
- If **`mode`** is the equation "subs"=*s*, then *s* is substituted for the indeterminate in the resulting minimal polynomial.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [x^3-x^2, y^2];
```

$$L := [x^3 - x^2, y^2]$$

```
> B := JBasis(L, var);
```

$$B := [y^2, x y^2, x^3 - x^2, x^2 y^2]$$

```
> JMinPoly(x, B, var);
```

$$\lambda^3 - \lambda^2$$

```
> JMinPoly(x^2+y, B, var);
```

$$\lambda^4 + \lambda^2 - 2\lambda^3$$

Example 2:

```
> var := [D,x];
```

$$var := [D, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(D, x)]$$

```
> L := [x*D-1, D^2];
```

$$L := [x D - 1, D^2]$$

```
> J := JBasis(L, var, ops);
```

$$J := [x D - 1, D^2]$$

```
> JFactorModuleBasis(var);
```

$$\frac{1}{1-x} + D$$

```
> JMinPoly(D, J, var, ops);
```

$$\lambda^2$$

Example 3:

```
> var := [a,b];
```

$$var := [a, b]$$

```
> L2 := [a*b-b, a+b^2];
```

$$L2 := [a\,b - b, a + b^2]$$

```
> B2 := JBasis(L2, var);
```

$$B2 := [a + b^2, a\,b - b, a^2 - a]$$

```
> JFactorModuleBasis(var);
```

$$[1, b, a]$$

```
> JMinPoly(a, B2, var, "var"=X);
```

$$X^2 - X$$

```
> JMinPoly(a, B2, var, "subs"=X+Y);
```

$$(X+Y)^2 - X - Y$$

```
> B3 := JBasis([a^31], [a]);
```

$$B3 := [a^{31}]$$

```
> JMinPoly(a, B3, [a]);
Error, (in JanetOre/JMinPoly) stopped calculation of minimal polynomial since upper bound for the degree is
reached.
```

```
> JMinPoly(a, B3, [a], "var"=lambda, "degree"=40);
```

$$\lambda^{31}$$

See Also:

JBasis, JFactorModuleBasis, JTabVar, JRepres, JCoeffList, JHilbertSeries.

JanetOre[JMultiplicity] - return the multiplicity of the factor module presented by the last computed Janet basis

Calling Sequence:

JMultiplicity()

Parameters:

- - none (assumes that the involutive basis has been computed before)

Description:

- ***JMultiplicity*** returns the leading coefficient of the filtered Hilbert polynomial (as in JHP), times the factorial of its degree, of the filtration of the factor module for which a presentation was computed by the last call of JBasis, as explained in JHilbertSeries.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x*y+y*z+z*x, x*y*z-1];
```

$$L := [xy + yz + zx, xyz - 1]$$

```
> JBasis(L, var);
```

$$[xy + yz + zx, yz^2 + z^2 x + 1, z^2 y^2 + y + z]$$

```
> JTabVar();
```

$$[xy + yz + zx, [x, y, z], xy]$$
$$[yz^2 + z^2 x + 1, [x, *, z], z^2 x]$$
$$[z^2 y^2 + y + z, [*, y, z], z^2 y^2]$$

```
> JHP();
```

$$6s - 3$$

```
> JDimension();
```

$$1$$

```
> JMultiplicity();
```

$$6$$

Example 2:

```
> var := [Dx,x,Dy,y];
```

$$var := [Dx, x, Dy, y]$$

```
> ops := [weyl(Dx,x),weyl(Dy,y)];
```

$$ops := [\mathrm{weyl}(Dx, x), \mathrm{weyl}(Dy, y)]$$

```
> L := [[x*Dx, -y*Dy], [x*Dy+y*Dx, 0]];
```

$$L := [[x\,Dx, -y\,Dy], [x\,Dy + y\,Dx, 0]]$$

```
> JBasis(L, var);
```

$$[[x\,Dy + y\,Dx, 0], [x\,Dx, -y\,Dy], [y\,Dx^2, Dy^2\,y], [Dx^2\,x, -Dx\,y\,Dy], [0, y^2\,Dx\,Dy + x\,Dy^2\,y]]$$

```
> JTabVar();
```

$$[[x\,Dy + y\,Dx, 0], [*, x, Dy, y], [x\,Dy, 1]]$$
$$[[x\,Dx, -y\,Dy], [*, x, Dy, y], [x\,Dx, 1]]$$
$$[[y\,Dx^2, Dy^2\,y], [Dx, *, Dy, y], [y\,Dx^2, 1]]$$
$$[[Dx^2\,x, -Dx\,y\,Dy], [Dx, x, Dy, y], [Dx^2\,x, 1]]$$
$$[[0, y^2\,Dx\,Dy + x\,Dy^2\,y], [Dx, x, Dy, y], [x\,Dy^2\,y, 2]]$$

```
> JHP();
```

$$\frac{13}{3}s + 3s^2 + \frac{2}{3}s^3 + 2$$

```
> JDimension();
```

3

```
> JMultiplicity();
```

4

See Also:

JBasis, JTabVar, JDimension, JHilbertSeries, JHilbertPolynomial, JHP, JHilbertFunction, JHF, JIndexRegularity, JCartanCharacter.

JanetOre[JHas],

JanetOre[JNotHas] - take a certain sublist of a list of elements of a free left module over an Ore algebra

Calling Sequence:

JHas(B,var,vi,ord)
JNotHas(B,var,vi,ord)

Parameters:

`B` - list of module elements, typically an involutive basis
`var` - list of variables (of the Ore algebra)
`vi` - list of variables to be inspected
`ord` - (optional) change of monomial ordering

Description:

- ***JNotHas*** respectively ***JHas*** returns the list of polynomials of **`B`** of which the leading monomial does not contain the variables in **`vi`** resp. does contain variables in **`vi`**.
- The elements in **`B`** must be polynomials in the variables **`var`** or lists (of the same length) of such polynomials.
- Typically **`B`** is an involutive basis of polynomials, cf. JBasis, computed with respect to pure lexicographical ordering, and **`vi`** are the first variables according to this ordering. In this case, ***JNotHas*** returns the polynomials of **`B`** not containing any variables of **`vi`**.
- With an optional fourth parameter the monomial ordering which affects the selection of the leading term can be chosen. The default ordering is degree reverse lexicographical (with "position over term" ordering in the module case). For a description of all possible orderings and assignment of degrees to variables and basis vectors (by means of parameter **`var`**) see JBasis.
- To extract the sublist consisting of those elements of **`B`** that does not contain any of the variables in **`vi`** at all one can use the Maple function remove (resp. ***select***) combined with has (see examples below).

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x*y+y*z+z*x, x^2-x*y];
```

$$L := [xy+yz+zx, x^2-xy]$$

```
> B := JBasis(L, var);
```

$$B := [xy+yz+zx, x^2+zx+yz, z^2x+yz^2+y^2z]$$

JTabVar displays the involutive basis, multiplicativity of variables and leading monomials:

```
> JTabVar();
```

$$[xy+yz+zx, [*, y, z], xy]$$
$$[x^2+zx+yz, [x, y, z], x^2]$$
$$[z^2x+yz^2+y^2z, [*, y, z], y^2z]$$

```
> JNotHas(B, var, [x]);
```

$$[z^2x+yz^2+y^2z]$$

(Note, with respect to the pure lexicographic ordering the summands containing x are the greatest, so we get:)

```
> JNotHas(B, var, [x], 1);
```

$$[\,]$$

```
> JHas(B, var, [x]);
```

$$[xy+yz+zx, x^2+zx+yz]$$

```
> JHas(B, var, [x,y]);
```

$$[xy+yz+zx, x^2+zx+yz, z^2x+yz^2+y^2z]$$

Example 2: Comparison to Maple functions remove / ***select*** / has

```
> L := [y^2+x*y, y^2-z^2];
```

$$L := [y^2+xy, y^2-z^2]$$

```
> B := JBasis(L, var);
```

$$B := [y^2-z^2, xy+z^2, z^2x+yz^2]$$

```
> JTabVar();
```

$$[y^2-z^2, [*, y, z], y^2]$$
$$[xy+z^2, [x, y, z], xy]$$
$$[z^2x+yz^2, [x, *, z], z^2x]$$

```
> JNotHas(B, var, [z]);
```

$$[y^2-z^2, xy+z^2]$$

```
> remove(has, B, [z]);
```

$$[\,]$$

```
> JHas(B, var, [z]);
```

$$[z^2x+yz^2]$$

```
> select(has, B, [z]);
```

$$[y^2-z^2, xy+z^2, z^2x+yz^2]$$

Example 3: Typically the command ***JNotHas*** comes in the context of elimination as follows:

```
> var := [x,y,a,b,c];
```

$$var := [x, y, a, b, c]$$

```
> L := [x^2+y^2-a, x^2*y^2-b, x^3*y-x*y^3-c];
```

$$L := [x^2+y^2-a, x^2y^2-b, x^3y-xy^3-c]$$

```
> L := JBasis(L, var, 1);
```

$$L := \Big[-4b^2+a^2b-c^2, a^2by-c^2y-4b^2y, -4b^2y^2+a^2by^2-y^2c^2, a^2by^3-c^2y^3-4b^2y^3, y^4-y^2a+b, cx+ay^3-ya^2+2by,$$
$$acx+a^2y^3-ya^3+2bya, a^2cx+a^3y^3-ya^4+8b^2y+2c^2y, a^2bx-4b^2x+acy^3-a^2cy+2byc, ycx+2by^2-ab,$$
$$acyx+2aby^2-4b^2-c^2, ya^2x-ac-4bxy+2y^2c, y^2cx+2by^3-bya, \frac{1}{2}acy-\frac{1}{2}abx+by^2x-\frac{1}{2}cy^3, y^2ax+yc-2bx,$$
$$xy^3-\frac{1}{2}yax+\frac{1}{2}c, x^2+y^2-a\Big]$$

```
> JNotHas(L, var, [x,y], 1);
```

$$[-4b^2+a^2b-c^2]$$

This of course is interpreted as a ring relation between x^2+y^2, x^2y^2, and x^3y-xy^3.

See Also:

JBasis, JTabVar, JSyzygies.

JanetOre[JRepres] - matrix representation with respect to a factor module basis

Calling Sequence:

JRepres(m,B,var,ops,FB,ord,mode)

Parameters:

`m`	-	polynomial in **`var`**
`B`	-	Janet basis
`var`	-	list of variables (of the Ore algebra)
`ops`	-	(optional) list of commutation rules for the variables of the Ore algebra
`FB`	-	factor module basis given as list of monomials or generating function
`ord`	-	(optional) change of monomial ordering
`mode`	-	(optional) string specifying options for the computation

Description:

- ***JRepres*** returns the matrix (in column convention) of the left multiplication of **`m`** on the free left module of the Ore algebra of appropriate rank modulo the submodule generated by the Janet basis **`B`**. The matrix is written with respect to the ground field basis **`FB`** usually computed with the command JFactorModuleBasis.
- Note, the ground field is allowed to be the field of complex numbers or field of rational functions (in one or several variables) over it.
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- If **`FB`** is a list, i.e. the factor module basis is finite, then the resulting matrix has shape *n* x *n*, where *n* is the length of the factor module basis, and it has entries in the ground field.
- If **`FB`** is given as generating function, i.e. **`FB`** is the sum of the monomials according to a disjoint cone decomposition of the factor module, then an entry in the *i*-th row of the resulting matrix is a polynomial in the multiplicative variables for the *i*-th cone of the factor module basis (in the order given by JFactorModuleBasis(**`var`**, "C")). The number of rows and the number of columns equal the number of cones in this case.
- The optional parameter **`mode`** may occur repeatedly. It may be equal to the string "S" or to the string "listlist".
- If **`mode`** equals "S", then ***JRepres*** uses simplify instead of expand in the normal form procedure.
- If **`mode`** equals "listlist", then the resulting matrix is returned as a listlist.
- For more information about disjoint cone decompositions of the factor module, see W. Plesken, D. Robertz, "Janet's approach to presentations and resolutions for polynomials and linear pdes", Archiv der Mathematik, 84(1), 2005, 22-37.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [x^2-2*y, x*y^2-y^2];
```

$$L := [x^2 - 2y, xy^2 - y^2]$$

```
> B := JBasis(L, var);
```

$$B := \left[x^2 - 2y, -\frac{1}{2}y^2 + y^3, xy^2 - y^2\right]$$

```
> FB := JFactorModuleBasis(var);
```

$$FB := [1, y, x, y^2, xy]$$

```
> Mx := JRepres(x, B, var, FB);
```

$$Mx := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

```
> My := JRepres(y, B, var, FB);
```

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

```
> linalg[minpoly](My, lambda);
```

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

```
> JMinPoly(y, B, var);
```

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

Example 2:

```
> var := [delta,t];
```

$$var := [\delta, t]$$

```
> ops := [shift(delta,t)];
```

$$ops := [\mathrm{shift}(\delta, t)]$$

```
> L := [t*delta^3+delta^2+t^2, delta^4-t^2];
```

$$L := [t\,\delta^3 + \delta^2 + t^2, \delta^4 - t^2]$$

```
> B := JBasis(L, var, ops);
```

$$B := [t^2, \delta^2, \delta\,t^2 + 2\,\delta\,t + \delta]$$

```
> FB := JFactorModuleBasis(var);
```

$$FB := [1, t, \delta, \delta\,t]$$

```
> Mdelta := JRepres(delta, B, var, ops, FB);
```

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

```
> Mt := JRepres(t, B, var, ops, FB);
```

$$Mt := \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & -2 \end{bmatrix}$$

```
> JInvReduce(JMult(t, t*delta, var, ops), B, var, ops);
```

$$-2\,\delta\,t - \delta$$

Example 3: Matrix representation with respect to an infinite basis

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [x^2*y-x, x*y^2-y];
```

$$L := [x^2 y - x, x y^2 - y]$$

```
> J := JBasis(L, var);
```

$$J := [x y^2 - y, x^2 y - x]$$

```
> F := JFactorModuleBasis(var);
```

$$F := \frac{1}{1-y} + \frac{x^2}{1-x} + x + x y$$

```
> JFactorModuleBasis(var, "C");
```

$$[1, x, x y, x^2]$$

```
> JRepres(x, J, var, F);
```

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

See Also:

JBasis, JFactorModuleBasis, JTabVar, JHilbertSeries, JMinPoly, JCoeffList

JanetOre[JResolution] - return free resolution of a factor module of a free left module over an Ore algebra

Calling Sequence:

JResolution(L,var,ops,mode,tr)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`mode` - (optional) string specifying options for the computation and the type of information to be returned
`tr` - (optional) positive integer (truncate resolution to length **`tr`**)

Description:

- ***JResolution*** computes a free resolution of the left *R*-module *R*^*m*/<**L**>, where *R* is the Ore algebra specified by **`var`** and **`ops`**, by first computing the minimal Janet basis for <**L**>, say of *k*(1) elements. These elements are given in form of a matrix representing a homomorphism *R*^*k*(1) -> *R*^*m* with cokernel *R*^*m*/<**L**>. It then computes a generating set **L**(1) of the kernel of this homomorphism and proceeds with *R*^*k*(1) and **L**(1) in the same way as it did with *R*^*m* and **L**. It terminates once the kernel is trivial.
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- As optional fourth parameter a string consisting of letters "C", "D", "G", "M", "O", and "S" is accepted that does not contain "D" and "M" at the same time.
- If **`mode`** contains the letter "M", then the output is the list of matrices that were computed as kernels of the above homomorphisms (in the reversed order they were constructed). This is the default mode. If **`mode`** contains the letter "D", then the output is the list containing lists of integers $[[d_{r,1}, ..., d_{r,n_r}], ..., [d_{1,1}, ..., d_{1,n_1}]]$, where $d_{i,j}$ is the degree of the *j*-th generator (row in matrix) of the *i*-th free module in the free resolution.
- Note, for matrices the row convention is used, i. e. *R*^*m* is identified with *R*^{1 x *m*} and the matrices are multiplied to rows from the right.
- If the letter "O" is present in **`mode`**, minimal Groebner bases are computed instead of minimal Janet bases in each step.
- If **`mode`** contains the letter "G", then the first matrix (i.e. the last one in the output) is formed using the given generating set **L**, i.e. the computation of a Janet basis is suppressed in the first step. If also the letter "C" is present in **`mode`**, then the minimal Janet basis is still computed in the first step and the first matrix is formed using the smaller generating set of **L** and its minimal Janet basis.
- If the letter "S" is present in **`mode`**, the program uses simplify instead of expand in the normal form procedure. If the polynomials in the input **L** contain nonrational coefficients, more precisely, if the ground field contains algebraic elements over the rationals (RootOf), then ***simplify*** is used instead of ***expand*** automatically.
- If the optional parameter **`tr`** is supplied, ***JResolution*** stops after having computed **`tr`** kernels as described above.
- For more information about Janet bases and resolutions, see W. Plesken, D. Robertz, "Janet's approach to presentations and resolutions for polynomials and linear pdes", Archiv der Mathematik, 84(1), 2005, 22-37.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,x];
```

$$var := [D, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(D, x)]$$

```
> L := [x*D-1,D^2];
```

$$L := [xD - 1, D^2]$$

```
> JBasis(L, var, ops);
```

$$[xD - 1, D^2]$$

```
> JResolution(L, var, ops);
```

$$\left[\left[\begin{bmatrix} D & -x \end{bmatrix}, \begin{bmatrix} xD-1 \\ D^2 \end{bmatrix}\right]\right]$$

Example 2:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> ops := [shift(x,z)];
```

$$ops := [\mathrm{shift}(x, z)]$$

```
> L := [x+y+z, x*y+y*z+z*x, x*y*z];
```

$$L := [x + y + z, xy + yz + zx, xyz]$$

```
> JBasis(L, var, ops);
```

$$[y + z, x, z^2]$$

```
> JTabVar();
```

$$[y + z, [*, y, z], y]$$
$$[x, [x, y, z], x]$$
$$[z^2, [*, *, z], z^2]$$

```
> R := JResolution(L, var, ops);
```

$$R := \left[\left[\begin{bmatrix} x & -y-z-1 & z^2+2z+1 \end{bmatrix}, \begin{bmatrix} -z^2 & 0 & y+z \\ 0 & -z^2-2z-1 & x \\ x & -y-z-1 & 0 \end{bmatrix}, \begin{bmatrix} y+z \\ x \\ z^2 \end{bmatrix}\right]\right]$$

```
> JResolution(L, var, ops, "D");
```

$$[[4], [3, 3, 2], [1, 1, 2]]$$

```
> JMult(R[1], R[2], var, ops); JMult(R[2], R[3], var, ops);
```

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

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

Example 3:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x^2,y^2,z^2];
```

$$L := [x^2, y^2, z^2]$$

```
> JResolution(L, var, "G");
```

$$\left[\left[\begin{bmatrix} -x^2 & 0 & -z^2 & y \\ 0 & y & 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & z^2 & -y^2 \\ z^2 & 0 & -x^2 \\ y^2 & -x^2 & 0 \\ z^2 y & 0 & -yx^2 \end{bmatrix}, \begin{bmatrix} x^2 \\ y^2 \\ z^2 \end{bmatrix}\right]\right]$$

```
> JResolution(L, var, "G", 1);
```

$$\left[\left[\begin{bmatrix} 0 & z^2 & -y^2 \\ z^2 & 0 & -x^2 \\ y^2 & -x^2 & 0 \\ z^2 y & 0 & -y x^2 \end{bmatrix}, \begin{bmatrix} x^2 \\ y^2 \\ z^2 \end{bmatrix}\right]\right]$$

```
> JResolution(L, var, "O");
```

$$\left[\begin{bmatrix} x^2 & z^2 & -y^2 \end{bmatrix}, \begin{bmatrix} y^2 & -z^2 & 0 \\ 0 & x^2 & -y^2 \\ x^2 & 0 & -z^2 \end{bmatrix}, \begin{bmatrix} z^2 \\ y^2 \\ x^2 \end{bmatrix}\right]$$

```
> JResolution(L, var, "DO");
```

[[6], [4, 4, 4], [2, 2, 2]]

See Also:

JBasis, JBasisFast, JTabVar, JMult, JInvReduce, JInvReduceFast, JSyzygies, JSyzygyModule, JSyzygyModuleFast, JResolutionDim, JEulerChar.

JanetOre[JRightInverse] - compute right inverse of a matrix with entries in an Ore algebra

Calling Sequence:

JRightInverse(M,var,ops)

Parameters:

`M` - matrix of polynomials in **`var`** or list of lists of the same length of polynomials in **`var`**
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra

Description:

- ***JRightInverse*** computes (if possible) a right inverse of the matrix **`M`**, i.e. a matrix *R* whose entries are polynomials in **`var`** such that the product of **`M`** by *R* is the identity matrix.
- The first parameter **`M`** is expected to be a matrix whose entries are polynomials in the variables **`var`** or a list of lists of polynomials in **`var`**, where each list is of the same length. In the second case, ***JRightInverse*** forms a matrix by taking the lists in **`M`** as rows and computes a right inverse of this matrix.
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- If no right inverse of **`M`** exists, ***JRightInverse*** returns FAIL.
- If a right inverse *R* of **`M`** exists, ***JRightInverse*** returns such an *R* as a matrix if **`M`** is a matrix, or returns the list of the rows of *R* if **`M`** is a list as explained above.
- Left inverses of polynomial matrices are computed by JLeftInverse.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,x];
```

$$var := [\mathrm{D}, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x)]$$

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

$$M := [x\,\mathrm{D} - 1 \quad \mathrm{D}^2]$$

```
> R := JRightInverse(M, var, ops);
```

$$R := \begin{bmatrix} -\frac{1}{6}x\,\mathrm{D} - \frac{2}{3} \\ \frac{1}{6}x^2 \end{bmatrix}$$

```
> JMult(M, R, var, ops);
```

$$[\ 1]$$

Example 2:

```
> var := [delta,t];
```

$$var := [\delta, t]$$

```
> ops := [shift(delta,t)];
```

$$ops := [\mathrm{shift}(\delta, t)]$$

```
> M := [[delta^2+1, -t, delta+1], [1, delta+1, 0]];
```

$$M := [[\delta^2 + 1, -t, \delta + 1], [1, \delta + 1, 0]]$$

```
> R := JRightInverse(M, var, ops);
```

$$R := [[2\,\delta + 1 + \delta^2, -4\,\delta - 1 - 2\,\delta^2], [-\delta - 1, 2 + 2\,\delta], [-t - \delta^2 - \delta^3 - \delta, 2\,t + 2\,\delta^2 + 1 + 2\,\delta^3 + \delta]]$$

```
> JMult(M, R, var, ops);
```

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

See Also:

JBasis, JTabVar, JInvReduce, JInvolution, JLeftInverse, JAddRhs.

JanetOre[JSubFactor] - return presentation of a subfactor of a finitely presented left module over an Ore algebra

Calling Sequence:

JSubFactor(L1,L2,var,ops,v)

Parameters:

`L1` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`L2` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the Ore algebra
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`v` - (optional) name of the indeterminate for the Hilbert series of the subfactor (default: 's')

Description:

- ***JSubFactor*** returns a presentation of the epimorphic image of the sum of the left modules generated by **L1** and **L2** in the left module presented by **L2**. This is a subfactor module of the factor module given by the free left module of *m*-tuples over the Ore algebra specified by **var** and **ops** modulo the submodule generated by **L2**, where *m* is the common length of the lists resp. rows in **L1** and **L2**. In many situations the left module generated by **L2** is a submodule of the left module generated by **L1**. Then ***JSubFactor*** returns a presentation of the left module generated by the residue classes represented by the entries of **L1** in the left module presented by **L2**.
- The entries of **L1** and **L2** are polynomials in case of left ideals, i.e. submodules of the free left module of rank one, or lists of polynomials of length *m*, representing elements of the free left module of *m*-tuples over the Ore algebra. In the latter case, the lists in **L1** and **L2** must be of the same length. If **L1** or **L2** is a matrix, then the generators are extracted from the rows of **L1** resp. **L2**.
- The commutation rules in the optional list **ops** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- The result of ***JSubFactor*** is a list with four entries. The first one defines the abstract generators of the constructed presentation of the subfactor module in terms of representatives of residue classes in the given subfactor module. The second entry is a list of the (left-linear) relations imposed on the abstract generators of the presentation. Finally, the third and the fourth entry of the result give the Hilbert series (see JHilbertSeries) resp. the Cartan characters (see JCartanCharacter) of the subfactor module.
- The first entry of the result is a list of equations, where the left hand sides are standard basis vectors in their canonical order, i.e. lists having exactly one entry equal to 1, the other entries being 0. The common length of these lists is the number of abstract generators in the presentation to be defined, and the left hand side of the *i*th equation is the *i*th standard basis vector. The right hand side of the *i*th equation gives a representative of the residue class in the subfactor module generated by the residue classes of the sum of the modules generated by **L1** and **L2** in the factor module presented by **L2** which corresponds to the *i*th abstract generator. Hence, the subfactor module is generated by the set of right hand sides in this first entry modulo the module generated by **L2**.
- The second entry of the result is a list of polynomials if the constructed presentation involves only one abstract generator and a list of lists of polynomials of the same length if there are more than one abstract generator. In the latter case, the common length of these lists equals the number of abstract generators. If the constructed presentation involves only one abstract generator, then the polynomials in this second list of the result generate the (lannihilator of this single generator in the polynomial ring. More generally in the case of several abstract generators, the lists of polynomials correspond to linear combinations of the abstract generators, where the coefficient of the *i*th generator is the *i*th polynomial in the list. All these linear combinations then generate all relations of the abstract generators, i.e. generate the submodule N of the free module M over the polynomial ring with indeterminates **var**, where the rank of M equals the number of abstract generators, such that the given subfactor module is isomorphic to the factor module M / N.
- The third entry of the result is the Hilbert series of the given subfactor module as explained in JHilbertSeries.
- The optional fifth argument to ***JSubFactor*** selects the name of the indeterminate for the Hilbert series. The default name is 's' which cannot be affected by a subs command.
- The fourth entry of the result is the list of Cartan characters of the given subfactor module as defined in JCartanCharacter.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x];
```

$$var := [x]$$

```
> JSubFactor([x], [x^2], var);
```

$$[[[1] = [x]], [x], 1, [0]]$$

```
> JSubFactor([1], [x], var);
```

$$[[[1] = [1]], [x], 1, [0]]$$

Example 2:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> JSubFactor([[x^3+y^3, x^2]], [[x^4, 0], [0, x^4]], var);
```

$$\left[[[1] = [x^3 + y^3, x^2]], [x^4], 1 + 2\,s + 3\,s^2 + 4\,s^3 + 4\,\frac{s^4}{1-s}, [4, 0]\right]$$

```
> JSubFactor([x^3+y^2,x^2+1], [y^2,x^4], var, lambda);
```

$$[[[1, 0] = [x^3], [0, 1] = [x^2 + 1]], [[x, 0], [0, y^2], [y^2, 0], [0, x y^2], [-1, x^3], [0, x^2 y^2]], 2 + 3\,\lambda + 2\,\lambda^2 + \lambda^3, [0, 0]]$$

Example 3:

```
> var := [D1,D2,x1,x2];
```

$$var := [D1, D2, x1, x2]$$

```
> ops := [weyl(D1,x1),weyl(D2,x2)];
```

$$ops := [weyl(D1, x1), weyl(D2, x2)]$$

```
> JSubFactor([D1, D2], [x1*D2, x2*D1], var, ops);
```

$$\left[[[1, 0] = [D1], [0, 1] = [D2]], [[x2, 0], [0, x1], [x1, -x2], [-D2, D1], [0, x2^2], [0, 2 + x2\,D2], [0, x1\,D2]],\right.$$
$$\left. 2 + 4\,s + s^2\left(3\,\frac{1}{1-s} + \frac{1}{(1-s)^2}\right), [3, 1, 0, 0]\right]$$

See Also:

JBasis, JInvReduce, JHilbertSeries, JSyzygies, JResolution, JKernel, JHom, JHomHom, JExt1, JExtn, JTorsion, JParametrization, JSyzOp.

JanetOre[JSubmoduleBasis] - return a vector space basis for the left module generated by the last computed Janet basis as a generating function

Calling Sequence:

JSubmoduleBasis(var,subs)

Parameters:

`var` - list of variables (of the Ore algebra)
`subs` - (optional) equation "subs"=expression

Description:

- ***JSubmoduleBasis*** returns a generating function which enumerates (the leading monomials of) a vector space basis for the submodule of the free left module over the Ore algebra generated by the Janet basis of the last call of JBasis.
- A term of the form $m/((1-x_1)...(1-x_n))\ e_i$ in the result enumerates all (tuples of) polynomials which are obtained as multiples of the unique Janet basis element with leading monomial m (in the i-th entry, in case of tuples) by polynomials in $x_1, ..., x_n$. Here m stands for a monomial in the indeterminates **`var`** and e_i for the i-th standard basis vector of the free module of tuples. Note that if the rank of this free module is greater than one, the result of ***JSubmoduleBasis*** is accordingly a list of generating functions.
- The result of ***JSubmoduleBasis*** can also be easily read off from the information given by JTabVar. It is just the sum of the leading monomials of the Janet basis, each multiplied by the geometric series $1/((1-x_{\mu_1})...(1-x_{\mu_k})$, where $\{ x_{\mu_1}, ..., x_{\mu_k} \}$ is the corresponding set of multiplicative variables for the respective Janet basis element.
- **`var`** is expected to be the list of variables of the polynomial ring that was given as parameter to ***JBasis*** before.
- If an optional equation "subs"=expression is provided, then ***JSubmoduleBasis*** substitutes 'expression' for all variables in **`var`** in the result (cf. Example 1 below).
- For more information about generalized Hilbert series, see W. Plesken, D. Robertz, "Janet's approach to presentations and resolutions for polynomials and linear pdes", Archiv der Mathematik, 84(1), 2005, 22-37.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> JBasis([x,y], var);
```

$$[y, x]$$

```
> JTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> JSubmoduleBasis(var);
```

$$\frac{y}{1-y} + \frac{x}{(1-x)(1-y)}$$

```
> JSubmoduleBasis(var, "subs"=t);
```

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

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20$$

$$t^{19} + O(t^{20})$$

```
> JSubmoduleHilbertFunction(0);
```

$$0$$

```
> JSubmoduleHilbertFunction(1);
```

$$2$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> JSubmoduleHilbertPolynomial(s);
```

$$1 + s$$

Example 2:

```
> var := [D,delta,x];
```

$$var := [\mathrm{D}, \delta, x]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x), \mathrm{shift}(\delta, x)]$$

```
> L := [x*D, x*delta];
```

$$L := [x\,\mathrm{D}, x\,\delta]$$

```
> JBasis(L, var, ops);
```

$$[\delta, x\,\mathrm{D}, \mathrm{D}\,\delta]$$

```
> JTabVar();
```

$$[\delta, [*, \delta, x], \delta]$$
$$[x\,\mathrm{D}, [\mathrm{D}, *, x], x\,\mathrm{D}]$$
$$[\mathrm{D}\,\delta, [\mathrm{D}, \delta, x], \mathrm{D}\,\delta]$$

```
> JSubmoduleBasis(var);
```

$$\frac{\delta}{(1-\delta)(1-x)} + \frac{x\,\mathrm{D}}{(1-\mathrm{D})(1-x)} + \frac{\mathrm{D}\,\delta}{(1-\mathrm{D})(1-\delta)(1-x)}$$

```
> JSubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 8\,t^3 + 13\,t^4 + 19\,t^5 + 26\,t^6 + 34\,t^7 + 43\,t^8 + 53\,t^9 + 64\,t^{10} + 76\,t^{11} + 89\,t^{12} + 103\,t^{13} + 118\,t^{14} + 134\,t^{15} + 151\,t^{16} + 169\,t^{17} + 188\,t^{18} + 208\,t^{19} + O(t^{20})$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.s) = 3/2*s-1+1/2*s^2, for s >= 2
> JSubmoduleHilbertPolynomial(s);
```

$$\frac{3}{2}s - 1 + \frac{1}{2}s^2$$

Example 3:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [[x^2,0], [x-y, z]];
```

$$L := [[x^2, 0], [x - y, z]]$$

```
> JBasis(L, var);
```

$$[[x - y, z], [y^2, -z\,y - z\,x], [0, z\,x^2]]$$

```
> JTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -z\,y - z\,x], [*, y, z], [y^2, 1]]$$
$$[[0, z\,x^2], [x, y, z], [z\,x^2, 2]]$$

```
> JSubmoduleBasis(var);
```

$$\left[\frac{y^2}{(1-y)(1-z)} + \frac{x}{(1-x)(1-y)(1-z)}, \frac{z\,x^2}{(1-x)(1-y)(1-z)}\right]$$

```
> JSubmoduleBasis(var, "subs"=t);
```

$$\left[\frac{t^2}{(1-t)^2}+\frac{t}{(1-t)^3},\frac{t^3}{(1-t)^3}\right]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t+4t^2+9t^3+16t^4+25t^5+36t^6+49t^7+64t^8+81t^9+100t^{10}+121t^{11}+144t^{12}+169t^{13}+196t^{14}+225t^{15}+256t^{16}+289$$
$$t^{17}+324t^{18}+361t^{19}+O(t^{20})$$

```
> JSubmoduleHilbertFunction(0);
```

$$0$$

```
> JSubmoduleHilbertFunction(1);
```

$$1$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> JSubmoduleHilbertPolynomial(s);
```

$$s^2$$

See Also:

JBasis, JTabVar, JSubmoduleHilbertSeries, JSubmoduleHilbertFunction, JSubmoduleHilbertPolynomial, JSubmoduleHF, JSubmoduleHP, JFactorModuleBasis, JHilbertSeries.

JanetOre[JSubmoduleDimension] - return the dimension of the module generated by the last computed Janet basis

Calling Sequence:

JSubmoduleDimension()

Parameters:

- - none (assumes that the involutive basis has been computed before)

Description:

- ***JSubmoduleDimension*** returns the degree of the filtered Hilbert polynomial (as in JSubmoduleHP) of the filtration of the factor module for which a presentation was computed by the last call of JBasis, as explained in JSubmoduleHilbertSeries.
- Note, ***JSubmoduleDimension***()-1 equals the degree of JSubmoduleHilbertPolynomial().

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,delta,x];
```

$$var := [D, \delta, x]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(D, x), \mathrm{shift}(\delta, x)]$$

```
> L := [x*D, x*delta];
```

$$L := [x D, x \delta]$$

```
> JBasis(L, var, ops);
```

$$[\delta, x D, D \delta]$$

```
> JTabVar();
```

$$[\delta, [*, \delta, x], \delta]$$
$$[x D, [D, *, x], x D]$$
$$[D \delta, [D, \delta, x], D \delta]$$

```
> JSubmoduleDimension();
```

$$3$$

```
> JSubmoduleHP();
```

$$-\frac{1}{6}s + s^2 + \frac{1}{6}s^3$$

```
> JSubmoduleHilbertPolynomial();
```

$$\frac{3}{2}s - 1 + \frac{1}{2}s^2$$

Example 2:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [[x,y,z], [y,z,x]];
```

$$L := [[x, y, z], [y, z, x]]$$

```
> JBasis(L, var);
```

$$[[y, z, x], [x, y, z]]$$

```
> JSubmoduleDimension();
```

$$2$$

```
> JSubmoduleHP();
```

$$s + s^2$$

```
> JSubmoduleHilbertPolynomial();
```

$$2s$$

See Also:

JBasis, JTabVar, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHP, JSubmoduleHF, JHilbertSeries, JDimension.

JanetOre[JSubmoduleHilbertFunction] - compute the graded Hilbert function for the left module generated by the last computed Janet basis

Calling Sequence:

JSubmoduleHilbertFunction(p)
JSubmoduleHilbertFunction()

Parameters:

p - " " (empty string) or natural number

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JSubmoduleHilbertSeries. Then ***JSubmoduleHilbertFunction*(p)** returns d_p in case **p** is a natural number and prints the function $s \to d_s$ in case **p** is the empty string.
- JSubmoduleHF, which is a summed up version of the present command and refers to the filtration rather than to the induced grading, must not be confused with ***JSubmoduleHilbertFunction***.
- ***JSubmoduleHilbertFunction***() returns a function expecting one parameter **p** which computes ***JSubmoduleHilbertFunction*(p)**.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> JBasis([x,y], var);
```

$$[y, x]$$

```
> JTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> JSubmoduleHilbertFunction(1);
```

$$2$$

```
> JSubmoduleHilbertFunction(9);
```

$$10$$

Example 2:

```
> var := [D,delta,x];
```

$$var := [\mathrm{D}, \delta, x]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x), \mathrm{shift}(\delta, x)]$$

```
> L := [x*D, x*delta];
```

$$L := [x\mathrm{D}, x\delta]$$

```
> JBasis(L, var, ops);
```

$$[\delta, x\,D, D\,\delta]$$

```
> JTabVar();
```

$$[\delta, [*, \delta, x], \delta]$$
$$[x\,D, [D, *, x], x\,D]$$
$$[D\,\delta, [D, \delta, x], D\,\delta]$$

```
> JSubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 8\,t^3 + 13\,t^4 + 19\,t^5 + 26\,t^6 + 34\,t^7 + 43\,t^8 + 53\,t^9 + 64\,t^{10} + 76\,t^{11} + 89\,t^{12} + 103\,t^{13} + 118\,t^{14} + 134\,t^{15} + 151\,t^{16} + 169\,t^{17} + 188\,t^{18} + 208\,t^{19} + O(t^{20})$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.s) = 3/2*s-1+1/2*s^2, for s >= 2
> JSubmoduleHilbertFunction(5);
```

19

Example 3:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [[x^2,0], [x-y, z]];
```

$$L := [[x^2, 0], [x - y, z]]$$

```
> JBasis(L, var);
```

$$[[x - y, z], [y^2, -z\,y - z\,x], [0, z\,x^2]]$$

```
> JTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -z\,y - z\,x], [*, y, z], [y^2, 1]]$$
$$[[0, z\,x^2], [x, y, z], [z\,x^2, 2]]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 9\,t^3 + 16\,t^4 + 25\,t^5 + 36\,t^6 + 49\,t^7 + 64\,t^8 + 81\,t^9 + 100\,t^{10} + 121\,t^{11} + 144\,t^{12} + 169\,t^{13} + 196\,t^{14} + 225\,t^{15} + 256\,t^{16} + 289\,t^{17} + 324\,t^{18} + 361\,t^{19} + O(t^{20})$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> JSubmoduleHilbertFunction(8);
```

64

See Also:

JBasis, JTabVar, JSubmoduleBasis, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHP, JSubmoduleHF, JFactorModuleBasis, JHilbertSeries, JHilbertFunction.

JanetOre[JSubmoduleHilbertPolynomial] - graded Hilbert polynomial for the left module generated by the last computed Janet basis

Calling Sequence:

JSubmoduleHilbertPolynomial(p)
JSubmoduleHilbertPolynomial()

Parameters:

p - natural number or name of an indeterminate

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JSubmoduleHilbertSeries. Then ***JSubmoduleHilbertPolynomial*(p)** returns d_p in case **p** is a natural number greater than or equal to the maximal (standard) degree of the elements in the Janet basis computed by the last call of JBasis. If **p** is the name of an indeterminate, then the Hilbert polynomial in **p** is returned. The information is derived from the last call of JBasis. Note, this same information can be extracted from the command JSubmoduleHilbertFunction.
- JSubmoduleHP, which is a summed up version of the present command and refers to the filtration rather than to the induced grading, must not be confused with ***JSubmoduleHilbertPolynomial***.
- ***JSubmoduleHilbertPolynomial***() returns the graded Hilbert polynomial of the module of the leading terms of the left module for which an involutive basis has been computed last by JBasis.
- As optional parameter a name **p** for the indeterminate of the Hilbert polynomial can be given. The default name of the indeterminate is 's'. It will not be affected by a subs command.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> JBasis([x,y], var);
```

$$[y, x]$$

```
> JTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> JSubmoduleHilbertPolynomial(s);
```

$$1 + s$$

```
> JSubmoduleHilbertPolynomial(1);
```

$$2$$

```
> JSubmoduleHilbertPolynomial(9);
```

$$10$$

Example 2:

```
> var := [D,delta,x];
```

$$var := [\mathrm{D}, \delta, x]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x), \mathrm{shift}(\delta, x)]$$

```
> L := [x*D, x*delta];
```

$$L := [x\,\mathrm{D}, x\,\delta]$$

```
> JBasis(L, var, ops);
```

$$[\delta, x\,\mathrm{D}, \mathrm{D}\,\delta]$$

```
> JTabVar();
```

$$[\delta, [*, \delta, x], \delta]$$
$$[x\,\mathrm{D}, [\mathrm{D}, *, x], x\,\mathrm{D}]$$
$$[\mathrm{D}\,\delta, [\mathrm{D}, \delta, x], \mathrm{D}\,\delta]$$

```
> JSubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 8\,t^3 + 13\,t^4 + 19\,t^5 + 26\,t^6 + 34\,t^7 + 43\,t^8 + 53\,t^9 + 64\,t^{10} + 76\,t^{11} + 89\,t^{12} + 103\,t^{13} + 118\,t^{14} + 134\,t^{15} + 151\,t^{16} + 169\,t^{17} + 188\,t^{18} + 208\,t^{19} + \mathrm{O}(t^{20})$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.s) = 3/2*s-1+1/2*s^2, for s >= 2
> JSubmoduleHilbertPolynomial(s);
```

$$\frac{3}{2}s - 1 + \frac{1}{2}s^2$$

```
> JSubmoduleHilbertPolynomial(5);
```

19

Example 3:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [[x^2,0], [x-y, z]];
```

$$L := [[x^2, 0], [x - y, z]]$$

```
> JBasis(L, var);
```

$$[[x - y, z], [y^2, -yz - zx], [0, zx^2]]$$

```
> JTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -yz - zx], [*, y, z], [y^2, 1]]$$
$$[[0, zx^2], [x, y, z], [zx^2, 2]]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 9\,t^3 + 16\,t^4 + 25\,t^5 + 36\,t^6 + 49\,t^7 + 64\,t^8 + 81\,t^9 + 100\,t^{10} + 121\,t^{11} + 144\,t^{12} + 169\,t^{13} + 196\,t^{14} + 225\,t^{15} + 256\,t^{16} + 289\,t^{17} + 324\,t^{18} + 361\,t^{19} + \mathrm{O}(t^{20})$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> JSubmoduleHilbertPolynomial(s);
```

$$s^2$$

```
> JSubmoduleHilbertPolynomial(8);
```

64

See Also:

JBasis, JTabVar, JSubmoduleBasis, JSubmoduleHilbertSeries, JSubmoduleHilbertFunction, JSubmoduleHF, JSubmoduleHP, JFactorModuleBasis, JHilbertSeries, JHilbertPolynomial.

JanetOre[JSubmoduleHilbertSeries] - Hilbert series of the module generated by the last computed Janet basis

Calling Sequence:

JSubmoduleHilbertSeries(v)

Parameters:

v - (optional) name of the indeterminate (default: 's')

Description:

- ***JSubmoduleHilbertSeries*** returns a generating function counting - according to the standard degrees - the leading monomials of the left module *M* generated by the Janet basis produced by the last call of JBasis.
- The Ore algebra defined by **var** and **ops** in the last call of ***JBasis*** has an (increasing) filtration *B* defined by the total degree. (In case of the Weyl algebras, this filtration is called the Bernstein filtration.) The graded algebra of this Ore algebra with respect to this filtration *B* is the commutative polynomial ring over the ground field with as many indeterminates as there are variables in **var**.
- For a non-zero element of the free left module of *m*-tuples over the Ore algebra, we define the total degree to be the maximal total degree of its non-zero entries. Then this free left module has an (increasing) *B*-filtration Γ defined by the total degree, i.e. left multiplication of elements of B_i to elements of Γ_j yields elements of Γ_{i+j}. The corresponding graded module over the graded algebra is then a direct sum of finite dimensional vector spaces over the ground field. The Hilbert series of this graded module is the formal power series whose *i*-th coefficient is the vector space dimension of the *i*-th graded piece of the module. ***JSubmoduleHilbertSeries*** returns the Hilbert series of the graded module which is associated with the left module *M* generated by the Janet basis computed by the last call of JBasis. Note that it depends on the term order chosen in that call of ***JBasis***.
- The output is the corresponding Hilbert series $\sum_{i=0}^{\infty} d_i v^i$, where the d_i are the dimensions of the homogeneous components of the graded module defined above.
- The default name of the indeterminate **v** is 's'. It cannot be affected by a subs command.
- Note, if one has assigned non-standard degrees to the variables or to the standard basis vectors, the command ***JSubmoduleHilbertSeries*** will proceed from the leading terms computed by JBasis but then reassign the degrees 1 for the variables and 0 for the basis vectors.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> JBasis([x,y], var);
```

$$[y, x]$$

```
> JTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> JSubmoduleBasis(var);
```

$$\frac{y}{1-y} + \frac{x}{(1-x)(1-y)}$$

```
> JSubmoduleBasis(var, "subs"=t);
```

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

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t+3t^2+4t^3+5t^4+6t^5+7t^6+8t^7+9t^8+10t^9+11t^{10}+12t^{11}+13t^{12}+14t^{13}+15t^{14}+16t^{15}+17t^{16}+18t^{17}+19t^{18}+20t^{19}+O(t^{20})$$

```
> JSubmoduleHilbertFunction(0);
```

$$0$$

```
> JSubmoduleHilbertFunction(1);
```

$$2$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> JSubmoduleHilbertPolynomial(s);
```

$$1+s$$

Example 2:

```
> var := [D,delta,x];
```

$$var := [\mathrm{D}, \delta, x]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x), \mathrm{shift}(\delta, x)]$$

```
> L := [x*D, x*delta];
```

$$L := [x\,\mathrm{D}, x\,\delta]$$

```
> JBasis(L, var, ops);
```

$$[\delta, x\,\mathrm{D}, \mathrm{D}\,\delta]$$

```
> JTabVar();
```

$$[\delta, [*, \delta, x], \delta]$$
$$[x\,\mathrm{D}, [\mathrm{D}, *, x], x\,\mathrm{D}]$$
$$[\mathrm{D}\,\delta, [\mathrm{D}, \delta, x], \mathrm{D}\,\delta]$$

```
> JSubmoduleBasis(var);
```

$$\frac{\delta}{(1-\delta)(1-x)}+\frac{x\,\mathrm{D}}{(1-\mathrm{D})(1-x)}+\frac{\mathrm{D}\,\delta}{(1-\mathrm{D})(1-\delta)(1-x)}$$

```
> JSubmoduleHilbertSeries(t);
```

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

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.s) = 3/2*s-1+1/2*s^2, for s >= 2
> JSubmoduleHilbertPolynomial(s);
```

$$\frac{3}{2}s-1+\frac{1}{2}s^2$$

Example 3:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [[x^2,0], [x-y, z]];
```

$$L := [[x^2, 0], [x-y, z]]$$

```
> JBasis(L, var);
```

$$[[x-y, z], [y^2, -z\,y-z\,x], [0, z\,x^2]]$$

```
> JTabVar();
```

$$[[x-y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -z\,y-z\,x], [*, y, z], [y^2, 1]]$$
$$[[0, z\,x^2], [x, y, z], [z\,x^2, 2]]$$

```
> JSubmoduleBasis(var);
```

$$\left[\frac{y^2}{(1-y)(1-z)}+\frac{x}{(1-x)(1-y)(1-z)},\frac{z\,x^2}{(1-x)(1-y)(1-z)}\right]$$

```
> JSubmoduleBasis(var, "subs"=t);
```

$$\left[\frac{t^2}{(1-t)^2}+\frac{t}{(1-t)^3},\frac{t^3}{(1-t)^3}\right]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t+4\,t^2+9\,t^3+16\,t^4+25\,t^5+36\,t^6+49\,t^7+64\,t^8+81\,t^9+100\,t^{10}+121\,t^{11}+144\,t^{12}+169\,t^{13}+196\,t^{14}+225\,t^{15}+256\,t^{16}+289\,t^{17}+324\,t^{18}+361\,t^{19}+O(t^{20})$$

```
> JSubmoduleHilbertFunction(0);
```

$$0$$

```
> JSubmoduleHilbertFunction(1);
```

$$1$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> JSubmoduleHilbertPolynomial(s);
```

$$s^2$$

See Also:

JBasis, JTabVar, JSubmoduleBasis, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHP, JSubmoduleHF, JFactorModuleBasis, JHilbertSeries.

JanetOre[JSubmoduleHF] - compute the filtered Hilbert function for the left module generated by the last computed Janet basis

Calling Sequence:

JSubmoduleHF(p)
JSubmoduleHF()

Parameters:

p - " " (empty string) or natural number

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JSubmoduleHilbertSeries. Then ***JSubmoduleHF*(p)** returns $\sum_{i=0}^{p} d_i$ for natural numbers **p** and prints the corresponding function in case **p** is the empty string.
- JSubmoduleHilbertFunction, of which the present command is a summed up version and which refers to the induced grading rather than to the filtration, must not be confused with ***JSubmoduleHF()***.
- ***JSubmoduleHF***() returns a function expecting one parameter **p** which computes ***JSubmoduleHF*(p)**.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> JBasis([x,y], var);
```

$$[y, x]$$

```
> JTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> JSubmoduleHF("");
s < 1: 0
s >= 1: 3/2*s+1/2*s^2
> JSubmoduleHF(1);
```

$$2$$

```
> JSubmoduleHF(2);
```

$$5$$

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> JSubmoduleHilbertFunction(1);
```

$$2$$

```
> JSubmoduleHilbertFunction(2);
```

$$3$$

Example 2:

```
> var := [D,delta,x];
```

$$var := [\mathrm{D}, \delta, x]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x), \mathrm{shift}(\delta, x)]$$

```
> L := [x*D, x*delta];
```

$$L := [x\,\mathrm{D}, x\,\delta]$$

```
> JBasis(L, var, ops);
```

$$[\delta, x\,\mathrm{D}, \mathrm{D}\,\delta]$$

```
> JTabVar();
```

$$[\delta, [*, \delta, x], \delta]$$
$$[x\,\mathrm{D}, [\mathrm{D}, *, x], x\,\mathrm{D}]$$
$$[\mathrm{D}\,\delta, [\mathrm{D}, \delta, x], \mathrm{D}\,\delta]$$

```
> JSubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 8\,t^3 + 13\,t^4 + 19\,t^5 + 26\,t^6 + 34\,t^7 + 43\,t^8 + 53\,t^9 + 64\,t^{10} + 76\,t^{11} + 89\,t^{12} + 103\,t^{13} + 118\,t^{14} + 134\,t^{15} + 151\,t^{16} + 169\,t^{17} + 188\,t^{18} + 208\,t^{19} + \mathrm{O}(t^{20})$$

```
> JSubmoduleHF("");
s < 1: 0
s = 1: 1
s >= 2: -1/6*s+s^2+1/6*s^3
> JSubmoduleHF(5);
```

45

```
> JSubmoduleHF(6);
```

71

```
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.s) = 3/2*s-1+1/2*s^2, for s >= 2
> JSubmoduleHilbertFunction(5);
```

19

```
> JSubmoduleHilbertFunction(6);
```

26

Example 3:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [[x^2,0], [x-y, z]];
```

$$L := [[x^2, 0], [x - y, z]]$$

```
> JBasis(L, var);
```

$$[[x - y, z], [y^2, -yz - zx], [0, zx^2]]$$

```
> JTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -yz - zx], [*, y, z], [y^2, 1]]$$
$$[[0, zx^2], [x, y, z], [zx^2, 2]]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 9\,t^3 + 16\,t^4 + 25\,t^5 + 36\,t^6 + 49\,t^7 + 64\,t^8 + 81\,t^9 + 100\,t^{10} + 121\,t^{11} + 144\,t^{12} + 169\,t^{13} + 196\,t^{14} + 225\,t^{15} + 256\,t^{16} + 289\,t^{17} + 324\,t^{18} + 361\,t^{19} + \mathrm{O}(t^{20})$$

```
> JSubmoduleHF("");
s < 1: 0
s = 1: 1
s = 2: 5
s >= 3: 1/6*s+1/2*s^2+1/3*s^3
> JSubmoduleHF(3);
```

14

```
> JSubmoduleHF(4);
                                                    30
> JSubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> JSubmoduleHilbertFunction(3);
                                                     9
> JSubmoduleHilbertFunction(4);
                                                    16
```

See Also:

JBasis, JTabVar, JSubmoduleBasis, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHP, JFactorModuleBasis, JHilbertSeries, JHF.

JanetOre[JSubmoduleHP] - compute the filtered Hilbert polynomial for the left module generated by the last computed Janet basis

Calling Sequence:

JSubmoduleHP(p)
JSubmoduleHP()

Parameters:

p - natural number or name of an indeterminate

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in JSubmoduleHilbertSeries. Then ***JSubmoduleHP*(p)** returns $\sum_{i=0}^{p} d_i$ for natural numbers **p** greater than or equal to the maximal (standard) degree of the elements in the Janet basis computed by the last call of JBasis, and the corresponding polynomial in **p** inducing this function in case **p** is an indeterminate. Note, all this information can also be extracted from the command JSubmoduleHF.
- ***JSubmoduleHP()*** returns the above polynomial with ´s´ as the default name of the indeterminate. ´s´ cannot be affected by asubs command.
- JSubmoduleHilbertPolynomial, of which the present command is a summed up version and which refers to the induced grading rather than to the filtration, must not be confused with ***JSubmoduleHP()***.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> JBasis([x,y], var);
```

$$[y, x]$$

```
> JTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> JSubmoduleHP(s);
```

$$\frac{3}{2}s + \frac{1}{2}s^2$$

```
> JSubmoduleHP(1);
```

$$2$$

```
> JSubmoduleHP(2);
```

$$5$$

```
> JSubmoduleHilbertPolynomial(s);
```

$$1 + s$$

```
> JSubmoduleHilbertPolynomial(1);
```

$$2$$

```
> JSubmoduleHilbertPolynomial(2);
```

Example 2:

```
> var := [D,delta,x];
```

$$var := [\mathrm{D}, \delta, x]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x), \mathrm{shift}(\delta, x)]$$

```
> L := [x*D, x*delta];
```

$$L := [x\,\mathrm{D}, x\,\delta]$$

```
> JBasis(L, var, ops);
```

$$[\delta, x\,\mathrm{D}, \mathrm{D}\,\delta]$$

```
> JTabVar();
```

$$[\delta, [*, \delta, x], \delta]$$
$$[x\,\mathrm{D}, [\mathrm{D}, *, x], x\,\mathrm{D}]$$
$$[\mathrm{D}\,\delta, [\mathrm{D}, \delta, x], \mathrm{D}\,\delta]$$

```
> JSubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 8\,t^3 + 13\,t^4 + 19\,t^5 + 26\,t^6 + 34\,t^7 + 43\,t^8 + 53\,t^9 + 64\,t^{10} + 76\,t^{11} + 89\,t^{12} + 103\,t^{13} + 118\,t^{14} + 134\,t^{15} + 151\,t^{16} + 169\,t^{17} + 188\,t^{18} + 208\,t^{19} + \mathrm{O}(t^{20})$$

```
> JSubmoduleHP(s);
```

$$-\frac{1}{6}s + s^2 + \frac{1}{6}s^3$$

```
> JSubmoduleHP(9);
```

$$201$$

```
> JSubmoduleHP(10);
```

$$265$$

```
> JSubmoduleHilbertPolynomial(s);
```

$$\frac{3}{2}s - 1 + \frac{1}{2}s^2$$

```
> JSubmoduleHilbertPolynomial(9);
```

$$53$$

```
> JSubmoduleHilbertPolynomial(10);
```

$$64$$

Example 3:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [[x^2,0], [x-y, z]];
```

$$L := [[x^2, 0], [x - y, z]]$$

```
> JBasis(L, var);
```

$$[[x - y, z], [y^2, -z\,y - z\,x], [0, z\,x^2]]$$

```
> JTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -z\,y - z\,x], [*, y, z], [y^2, 1]]$$
$$[[0, z\,x^2], [x, y, z], [z\,x^2, 2]]$$

```
> JSubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4\,t^2 + 9\,t^3 + 16\,t^4 + 25\,t^5 + 36\,t^6 + 49\,t^7 + 64\,t^8 + 81\,t^9 + 100\,t^{10} + 121\,t^{11} + 144\,t^{12} + 169\,t^{13} + 196\,t^{14} + 225\,t^{15} + 256\,t^{16} + 289\,t^{17} + 324\,t^{18} + 361\,t^{19} + \mathrm{O}(t^{20})$$

```
> JSubmoduleHP(s);
```

$$\frac{1}{2}s^2+\frac{1}{6}s+\frac{1}{3}s^3$$

```
> JSubmoduleHP(3);
```

14

```
> JSubmoduleHP(4);
```

30

```
> JSubmoduleHilbertPolynomial(s);
```

s^2

```
> JSubmoduleHilbertPolynomial(3);
```

9

```
> JSubmoduleHilbertPolynomial(4);
```

16

See Also:

JBasis, JTabVar, JSubmoduleBasis, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHF, JFactorModuleBasis, JHilbertSeries, JHP.

JanetOre[JSubmoduleMultiplicity] - return the multiplicity of the module generated by the last computed Janet basis

Calling Sequence:

JSubmoduleMultiplicity()

Parameters:

- - none (assumes that the involutive basis has been computed before)

Description:

- ***JSubmoduleDimension*** returns the leading coefficient of the filtered Hilbert polynomial (as in JSubmoduleHP), times the factorial of its degree, of the filtration of the factor module for which a presentation was computed by the last call of JBasis, as explained in JSubmoduleHilbertSeries.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,delta,x];
```

$$var := [D, \delta, x]$$

```
> ops := [weyl(D,x), shift(delta,x)];
```

$$ops := [\mathrm{weyl}(D, x), \mathrm{shift}(\delta, x)]$$

```
> L := [x*D, x*delta];
```

$$L := [x D, x \delta]$$

```
> JBasis(L, var, ops);
```

$$[\delta, x D, D \delta]$$

```
> JTabVar();
```

$$[\delta, [*, \delta, x], \delta]$$
$$[x D, [D, *, x], x D]$$
$$[D \delta, [D, \delta, x], D \delta]$$

```
> JSubmoduleHP();
```

$$-\frac{1}{6}s + s^2 + \frac{1}{6}s^3$$

```
> JSubmoduleDimension();
```

$$3$$

```
> JSubmoduleMultiplicity();
```

$$1$$

Example 2:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [[x,y,z], [y,z,x]];
```

$$L := [[x, y, z], [y, z, x]]$$

```
> JBasis(L, var);
```

$$[[y, z, x], [x, y, z]]$$

```
> JSubmoduleHP();
```

$$s + s^2$$

```
> JSubmoduleDimension();
```

$$2$$

```
> JSubmoduleMultiplicity();
```

$$2$$

See Also:

JBasis, JTabVar, JSubmoduleHilbertSeries, JSubmoduleHilbertPolynomial, JSubmoduleHilbertFunction, JSubmoduleHP, JSubmoduleHF, JHilbertSeries, JDimension.

JanetOre[JSyzOp] - return a matrix whose rows generate the syzygy module of a finitely presented left module over an Ore algebra

Calling Sequence:

JSyzOp(L,var,ops)

Parameters:

`L` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the Ore algebra
`ops` - (optional) list of commutation rules for the variables of the Ore algebra

Description:

- ***JSyzOp*** constructs a matrix whose rows generate the syzygy module of the module *M* presented by **L** (i.e., the elements of **L** are considered as elements of a free left module over the Ore algebra specified by **`var`** and **`ops`** of appropriate rank and *M* is the factor module of this free left module modulo the submodule that is generated by the elements of **L**). This matrix is constructed by computing the beginning of a free resolution of *M* using JResolution. For more information about syzygies cf. also JSyzygies.
- The entries of **L** are polynomials in case of a left ideal, i. e. a submodule of the free left module of rank one, or lists of polynomials of length *m*, representing elements of the free left module of *m*-tuples over the Ore algebra which is specified by **`var`** and **`ops`**. If **L** is a matrix, then the generators are extracted from the rows of **L**.
- The commutation rules in the optional list **`ops`** which are accepted are listed in the introduction to the JanetOre package. Moreover, the way ***JanetOre*** represents elements of the Ore algebra as polynomials is also explained in the introduction to the JanetOre package.
- The result of ***JSyzOp*** is a matrix with polynomial entries such that the product of this matrix by the matrix whose rows are the entries in **L** is a zero matrix.
- The name of the command ***JSyzOp*** is motivated by the corresponding command SyzOp in the package Janet.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> S := JSyzOp([x, y], var);
```

$$S := [-y \quad x]$$

```
> L := matrix([[x], [y]]);
```

$$L := \begin{bmatrix} x \\ y \end{bmatrix}$$

```
> JMult(S, L, var);
```

$$[0]$$

Example 2:

```
> var := [D1,D2,x1,x2];
```

$$var := [\mathrm{D1}, \mathrm{D2}, x1, x2]$$

```
> ops := [weyl(D1,x1),weyl(D2,x2)];
```

$$ops := [\mathrm{weyl}(\mathrm{D1}, x1), \mathrm{weyl}(\mathrm{D2}, x2)]$$

```
> L1 := matrix([[x1*D2], [x2*D1]]);
```

$$L1 := \begin{bmatrix} x1\,\mathrm{D2} \\ x2\,\mathrm{D1} \end{bmatrix}$$

```
> L2 := JSyzOp(L1, var, ops);
```

$$L2 := \begin{bmatrix} x2\,\mathrm{D2}\,\mathrm{D1}^2 + 2\,\mathrm{D1}^2 & -2\,\mathrm{D2}^2 - \mathrm{D2}^2\,x1\,\mathrm{D1} \\ -2 - x2\,\mathrm{D2} + x1\,\mathrm{D1}\,x2\,\mathrm{D2} + 2\,x1\,\mathrm{D1} & -x1^2\,\mathrm{D2}^2 \\ x2^2\,\mathrm{D1}^2 & 2 - 2\,x2\,\mathrm{D2} + x1\,\mathrm{D1} - x1\,\mathrm{D1}\,x2\,\mathrm{D2} \\ -x2^2 + x1\,x2^2\,\mathrm{D1} & -x1^2\,x2\,\mathrm{D2} + x1^2 \end{bmatrix}$$

```
> JMult(L2, L1, var, ops);
```

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

See Also:

JBasis, JInvReduce, JSyzygies, JSyzygyModule, JResolution, JParametrization, JHom, JHomHom, JExt1, JExtn, JTorsion, JKernel, JLeftInverse, JRightInverse.

JanetOre[JSyzygies] - return generating set for the syzygies of a finite generating set for a left module over an Ore algebra

Calling Sequence:

JSyzygies(L,var,ops,ord,mode)

Parameters:

`L` - list of generators of the submodule with right hand sides
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`ord` - (optional) change of monomial ordering
`mode` - (optional) string "S", use ***simplify*** instead of ***expand*** internally

Description:

- ***JSyzygies*** returns the list of expressions which occurred as right hand sides corresponding to zero left hand side during computation of the last call of JBasis or during the reduction of the original generators and non-multiplicative prolongations of the Janet basis elements. Note, **L** has to be the same as in the last call of ***JBasis*** and must be given with right hand sides naming the generators: generator=name.
- In terms of modules, ***JSyzygies*** constructs the syzygies among the generators **L** of the module, i.e. it returns a generating set for the (left) linear relations satisfied by the entries of **L**.
- In general, the result of ***JSyzygies*** is not a Janet basis for the syzygies of **L**. The command JSyzygyModule returns a Janet basis for the syzygies.
- The parameters **`var`**, **`ops`**, **`ord`** have the same meaning as in JBasis. In particular one has the possibility via **`var`** to work with other than the standard degrees for the variables and basis vectors, provided one has done that already in JBasis.
- If the string "S" is given as parameter **`mode`**, the program uses simplify instead of expand in the normal form procedure. If the polynomials in the input **L** contain nonrational coefficients, more precisely, if the ground field contains algebraic elements over the rationals (RootOf), then ***simplify*** is used instead of ***expand*** automatically.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,x];
```

$$var := [\mathrm{D}, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(\mathrm{D}, x)]$$

```
> L := [D^4=a, x*D^2=b, x^2=c];
```

$$L := [\mathrm{D}^4 = a, x\,\mathrm{D}^2 = b, x^2 = c]$$

```
> JBasis(L, var, ops);
```

$$\left[1 = \frac{1}{3}\,\mathrm{D}\,x^2\,b - \frac{1}{3}\,x\,c\,\mathrm{D}^3 + \frac{1}{2}\,c\,\mathrm{D}^2 + \frac{7}{6}\,x\,b\right]$$

```
> S := JSyzygies(L, var, ops);
```

$$S := [-2\,x^4\,\mathrm{D}\,b + 2\,x^3\,c\,\mathrm{D}^3 - 3\,x^2\,c\,\mathrm{D}^2 - 7\,x^3\,b + 6\,c, -2\,\mathrm{D}^3\,x^3\,b + 2\,x^2\,c\,\mathrm{D}^5 + x\,c\,\mathrm{D}^4 - 15\,\mathrm{D}^2\,x^2\,b - 18\,\mathrm{D}\,x\,b + 6\,b,$$
$$-2\,\mathrm{D}^5\,x^2\,b + 2\,x\,c\,\mathrm{D}^7 + 5\,c\,\mathrm{D}^6 - 23\,\mathrm{D}^4\,x\,b - 52\,\mathrm{D}^3\,b + 6\,a, -x^3\,b + x^2\,c\,\mathrm{D}^2 - 4\,x\,c\,\mathrm{D} + 6\,c, \mathrm{D}\,x^3\,b - x^2\,c\,\mathrm{D}^3 + 2\,x\,c\,\mathrm{D}^2 + 3\,x^2\,b - 2\,c\,\mathrm{D},$$
$$-\mathrm{D}^2\,x^2\,b + x\,c\,\mathrm{D}^4 - 6\,\mathrm{D}\,x\,b - 6\,b]$$

```
> collect(S, [a,b,c]);
```

$$[(-7\,x^3 - 2\,x^4\,\mathrm{D})\,b + (6 + 2\,x^3\,\mathrm{D}^3 - 3\,x^2\,\mathrm{D}^2)\,c, (-2\,x^3\,\mathrm{D}^3 - 15\,x^2\,\mathrm{D}^2 - 18\,\mathrm{D}\,x + 6)\,b + (2\,x^2\,\mathrm{D}^5 + x\,\mathrm{D}^4)\,c,$$
$$6\,a + (-2\,x^2\,\mathrm{D}^5 - 23\,x\,\mathrm{D}^4 - 52\,\mathrm{D}^3)\,b + (2\,x\,\mathrm{D}^7 + 5\,\mathrm{D}^6)\,c, -x^3\,b + (x^2\,\mathrm{D}^2 - 4\,\mathrm{D}\,x + 6)\,c, (3\,x^2 + \mathrm{D}\,x^3)\,b + (-2\,\mathrm{D} - x^2\,\mathrm{D}^3 + 2\,x\,\mathrm{D}^2)\,c,$$
$$(-x^2\,\mathrm{D}^2 - 6\,\mathrm{D}\,x - 6)\,b + x\,c\,\mathrm{D}^4]$$

```
> JMult(-7*x^3-2*x^4*D, x*D^2, var, ops)+JMult(6+2*x^3*D^3-3*x^2*D^2, x^2, var, ops);
```

$$0$$

Example 2:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [[x^2,0]=[1,0,0], [0,y]=[0,1,0], [x^2,y]=[0,0,1]];
```

$$L := [[x^2, 0] = [1, 0, 0], [0, y] = [0, 1, 0], [x^2, y] = [0, 0, 1]]$$

```
> JBasis(L, var);
```

$$[[0, y] = [0, 1, 0], [x^2, 0] = [1, 0, 0]]$$

```
> JSyzygies(L, var);
```

$$[[-1, -1, 1]]$$

See Also:

JBasis, JBasisFast, JAddRhs, JTabVar, JInvReduce, JInvReduceFast, JSyzygyModule, JSyzygyModuleFast, JResolution, JResolutionDim, JEulerChar.

JanetOre[JSyzygyModule] - return Janet basis of syzygy module of a generating set of a left module over an Ore algebra

Calling Sequence:

JSyzygyModule(L,var,ops,ord,mode,rel)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`ord` - (optional) change of monomial ordering
`mode` - (optional) string specifying options for the computation
`rel` - (optional) equation "mod" = list of generators of a submodule

Description:

- ***JSyzygyModule*** returns the minimal Janet basis with respect to a certain ordering of the syzygy module of the generating set **`L`** of a submodule of a free left module of tuples over the Ore algebra specified by **`var`** and **`ops`**. If the optional parameter **`rel`** is specified, then the entries of **`L`** are interpreted as representatives of residue classes modulo the submodule generated by the right hand side of **`rel`**.
- The entries of **`L`** are polynomials in case of a left ideal, i. e. a submodule of the free left module of rank one, or lists of polynomials of length *m*, representing elements of the free left module of *m*-tuples over the Ore algebra. If **`L`** is a matrix, then the generators are extracted from the rows of **`L`**. If the optional parameter **`rel`** is present, then the right hand side in **`rel`** is expected to contain polynomials in **`var`** or lists of polynomials of length *m* according to the entries in **`L`**.
- The parameters **`var`**, **`ops`**, **`ord`** and have the same meaning as in JBasis.
- The fourth argument **`mode`** is a string consisting of letters "N" or "S".
- If the letter "N" is present in **`mode`**, leading coefficients in the Janet basis of the syzygy module are *not* normalized to 1 (cf. also JBasis).
- If the letter "S" is present in **`mode`**, the program uses simplify instead of expand in the normal form procedure. If the polynomials in the input **`L`** contain nonrational coefficients, more precisely, if the ground field contains algebraic elements over the rationals (RootOf), then ***simplify*** is used instead of ***expand*** automatically.
- By means of the command JanetOreOptions one can also choose between two implementations of ***JSyzygyModule***: "Maple" and "C++".

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,x];
```

$$var := [D, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(D, x)]$$

```
> L1 := [D^4, x*D^2, x^2];
```

$$L1 := [D^4, x D^2, x^2]$$

```
> JSyzygyModule(L1, var, ops);
```

$$[[-x D - 3, D^3, 0], [0, -x^3, x^2 D^2 - 4 x D + 6], [-x^3, x^2 D^2 - 2 x D + 2, 0], [-3 x^2, 2 x D^2 - 12 D, D^4],$$
$$[0, -x^3 D - 3 x^2, x^2 D^3 - 2 x D^2 + 2 D]]$$

Janet basis of syzygy module with respect to "position over term" ordering:

```
> JSyzygyModule(L1, var, ops, 2);
```

$$\left[[0, x^3, -x^2\,D^2 + 4\,x\,D - 6], [0, x^3\,D + 3\,x^2, -x^2\,D^3 + 2\,x\,D^2 - 2\,D], [0, x^2\,D^2 + 6\,x\,D + 6, -x\,D^4], \left[1, \frac{1}{6}\,x\,D^4 + \frac{4}{3}\,D^3, -\frac{1}{6}\,D^6\right]\right]$$

Example 2: A sample calculation for modules over the polynomial ring Q[x,y]:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L2 := [[x^2,0], [0,y], [x^2,y]];
```

$$L2 := [[x^2, 0], [0, y], [x^2, y]]$$

```
> JSyzygyModule(L2, var);
```

$$[[1, 1, -1]]$$

Example 3: Syzygies of a generating set of residue classes of a factor module

```
> var := [x,y];
```

$$var := [x, y]$$

```
> R := [[x^3,0], [0,x^3]];
```

$$R := [[x^3, 0], [0, x^3]]$$

```
> L3 := [[x^2,0], [0,y], [x^2,y]];
```

$$L3 := [[x^2, 0], [0, y], [x^2, y]]$$

```
> S := JSyzygyModule(L3, var, "mod"=R);
```

$$S := [[1, 1, -1], [0, x, -x], [0, 0, x^3]]$$

```
> JInvReduce([x,0,0], S, var);
```

$$[0, 0, 0]$$

Example 4: The next example deals with nonrational coefficients:

```
> alias(omega=RootOf(a^2+a+1,a));
```

$$I, \omega$$

```
> simplify(omega^2);
```

$$-1 - \omega$$

```
> L4 := [x+omega*y+omega^2*z,x*y+y*z+z*x,x*y*z];
```

$$L4 := [x + \omega\,y + \omega^2\,z, y\,x + y\,z + z\,x, x\,y\,z]$$

```
> JSyzygyModule(L4, [x,y,z]);
```

$$[[y\,x + y\,z + z\,x, -x - \omega\,y + z + \omega\,z, 0], [y\,z^2 + x\,z^2, -z\,(x + \omega\,y - z - \omega\,z), x + \omega\,y - z - \omega\,z], [0, x\,y\,z, -y\,x - z\,x - y\,z],$$
$$[z^2\,y^2, -y\,z\,\omega\,(\omega\,z + y), \omega\,(y^2 + 2\,y\,z + 2\,\omega\,y\,z + z^2\,\omega)]]$$

```
> map(a->simplify(evalm([a] &* vector(L4))), %);
```

$$[[0], [0], [0], [0]]$$

See Also:

JBasis, JBasisFast, JAddRhs, JTabVar, JanetOreOptions, JInvReduce, JInvReduceFast, JSyzygies, JSyzygyModuleFast, JResolution, JResolutionDim, JEulerChar.

JanetOre[JSyzygyModuleFast] - return Janet basis of syzygy module of a generating set of a left module over an Ore algebra (C++ version)

Calling Sequence:

JSyzygyModuleFast(L,var,ops,ord,mode)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the Ore algebra)
`ops` - (optional) list of commutation rules for the variables of the Ore algebra
`ord` - (optional) change of polynomial ordering (see below)
`mode` - (optional) sequence of equations specifying options for the computation

Description:

- ***JSyzygyModuleFast*** computes the minimal Janet basis with respect to a certain ordering of the syzygy module of the generating set **`L`** of a submodule *M* of a free left module of tuples over the Ore algebra specified by **`var`** and **`ops`** by using the C++ version of the command JBasis (cf. JBasisFast). Up to now, only the algorithm for the degree reverse lexicographical ordering (i.e., **`ord`** is 2 or 4) is implemented in C++.
- All parameters to ***JSyzygyModuleFast*** have the same meaning as in JSyzygyModule.
- The advantage of this command is clearly the enormous speed up. (Note, you cannot interrupt this command from within Maple; you have to kill the process "JBOre" instead.)
- If an equation with left hand side "mod" occurs in **`mode`**, then its right hand side is expected to be a list of generators of a submodule *N* of the module *M* generated by **`L`**. In this case, the given generators are internally appended to **`L`**, the Janet basis of syzygies for the extended list is computed, but the terms in syzygies which correspond to coefficients for the generators of *N* are neglected. In this way, a Janet basis for the syzygy module of the factor module *M* / *N* is obtained. See also Example 3 below.
- The right hand side of an equation "denom"=*b* in **`mode`** is expected to be either true or false. The default value is false. If *b* equals true, then the C++ program collects all coefficients by which it divides during the computation of the involutive basis (these arise either as contents of polynomials treated by the algorithm or as leading coefficients in the result before normalizing) together with the coefficients that occur in some denominator of the input **`L`**. After the computation is finished and the result is read into Maple, this list of denominators can be obtained via JZeroSets. See also Example 4 below.
- Using the option "C++" of JanetOreOptions, the command ***JSyzygyModule*** is replaced by ***JSyzygyModuleFast*** for the current Maple session.

Examples:

```
> with(JanetOre):
```

Example 1:

```
> var := [D,x];
```

$$var := [D, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(D, x)]$$

```
> L1 := [D^4, x*D^2, x^2];
```

$$L1 := [D^4, x D^2, x^2]$$

```
> JSyzygyModuleFast(L1, var, ops);
```

$$[[-D x - 3, D^3, 0], [0, -x^3, D^2 x^2 - 4 D x + 6], [-x^3, D^2 x^2 - 2 D x + 2, 0], [-3 x^2, 2 x D^2 - 12 D, D^4],$$
$$[0, -D x^3 - 3 x^2, D^3 x^2 - 2 x D^2 + 2 D]]$$

Janet basis of syzygy module with respect to "position over term" ordering:

```
> JSyzygyModuleFast(L1, var, ops, 2);
```

$$\left[[0, x^3, -D^2 x^2 + 4 D x - 6], [0, D x^3 + 3 x^2, -D^3 x^2 + 2 x D^2 - 2 D], [0, D^2 x^2 + 6 D x + 6, -D^4 x], \left[1, \frac{1}{6} D^4 x + \frac{4}{3} D^3, -\frac{D^6}{6}\right]\right]$$

Example 2: A sample calculation for modules over the polynomial ring Q[x,y]:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L2 := [[x^2,0], [0,y], [x^2,y]];
```

$$L2 := [[x^2, 0], [0, y], [x^2, y]]$$

```
> JSyzygyModuleFast(L2, var);
```

$$[[1, 1, -1]]$$

Example 3: Syzygies of a generating set of residue classes of a factor module

```
> var := [x,y];
```

$$var := [x, y]$$

```
> R := [[x^3,0], [0,x^3]];
```

$$R := [[x^3, 0], [0, x^3]]$$

```
> L3 := [[x^2,0], [0,y], [x^2,y]];
```

$$L3 := [[x^2, 0], [0, y], [x^2, y]]$$

```
> S := JSyzygyModuleFast(L3, var, "mod"=R);
```

$$S := [[1, 1, -1], [0, x, -x], [0, 0, x^3]]$$

```
> JInvReduceFast([x,0,0], S, var);
```

$$[0, 0, 0]$$

Example 4: Example 1 with keeping track of denominators

```
> var := [D,x];
```

$$var := [D, x]$$

```
> ops := [weyl(D,x)];
```

$$ops := [\mathrm{weyl}(D, x)]$$

```
> L1 := [D^4, x*D^2, x^2];
```

$$L1 := [D^4, x D^2, x^2]$$

```
> JSyzygyModuleFast(L1, var, ops, "denom"=true);
```

$$[[-D x - 3, D^3, 0], [0, -x^3, D^2 x^2 - 4 D x + 6], [-x^3, D^2 x^2 - 2 D x + 2, 0], [-3 x^2, 2 x D^2 - 12 D, D^4],$$
$$[0, -D x^3 - 3 x^2, D^3 x^2 - 2 x D^2 + 2 D]]$$

```
> JZeroSets();
```

$$[2, 3]$$

See Also:

JBasis, JBasisFast, JanetOreOptions, JTabVar, JInvReduce, JInvReduceFast, JHilbertSeries, JSyzygies, JSyzygyModule, JResolution, JResolutionDim, JEulerChar.

JanetOre[JTabVar] - display Janet's data, i. e. the generators, their leading monomials, multiplicative variables etc.

Calling Sequence:

JTabVar()

Parameters:

- - none (assumes that the involutive basis has been computed before)

Description:

- ***JTabVar*** displays the data constructed by JBasis. Therefore it is necessary to call ***JBasis*** first. The data structure is a list of lists each corresponding to an element of the Janet basis.
- In the case of left ideals the entries of each list are the basis polynomial, the list of multiplicative / non-multiplicative variables and the leading monomial. The variables occurring in the second entry are the multiplicative variables of the respective leading monomial. Non-multiplicative variables are represented by '*'.
- In the module case the first entry is a list of polynomials representing an element of the free module of tuples over the Ore algebra, the second entry indicates multiplicative and non-multiplicative variables as above, the third entry is a list with first entry the leading monomial and second entry its position within the tuple.

Examples:

```
> with(JanetOre):
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x+y+z, x*y+y*z+z*x, x*y*z-1];
```

$$L := [x+y+z, xy+yz+zx, xyz-1]$$

```
> JBasis(L, var);
```

$$[x+y+z, y^2+yz+z^2, z^3-1, -y+z^3 y]$$

```
> JTabVar();
```

$$[x+y+z, [x, y, z], x]$$
$$[y^2+yz+z^2, [*, y, z], y^2]$$
$$[z^3-1, [*, *, z], z^3]$$
$$[-y+z^3 y, [*, *, z], z^3 y]$$

Note some effects of the other monomial ordering:

```
> JBasis(L, var, 1);
```

$$[z^3-1, -y+z^3 y, y^2+yz+z^2, x+y+z]$$

```
> JTabVar();
```

$$[z^3-1, [*, *, z], z^3]$$
$$[-y+z^3 y, [*, *, z], z^3 y]$$
$$[y^2+yz+z^2, [*, y, z], y^2]$$
$$[x+y+z, [x, y, z], x]$$

Note, right hand sides are displayed as well:

```
> L := [x+y+z=a, x*y+y*z+z*x=b, x*y*z-1=c];
```

$$L := [x+y+z=a, xy+yz+zx=b, xyz-1=c]$$

```
> JBasis(L, var);
```

$$[x+y+z=a, y^2+yz+z^2=ya+za-b, z^3-1=z^2 a-zb+c, -y+z^3 y=az^2 y-bzy+cy]$$

```
> JTabVar();
```

$$[x+y+z=a, [x, y, z], x]$$
$$[y^2+yz+z^2=ya+za-b, [*, y, z], y^2]$$
$$[z^3-1=z^2 a-zb+c, [*, *, z], z^3]$$
$$[-y+z^3 y=az^2 y-bzy+cy, [*, *, z], z^3 y]$$

For modules the output is slightly more involved:

```
> JBasis([[x^2-1, 0], [x*y, x*y], [0, y^2-1]], [x,y]);
```

$$[[0, y^2-1], [xy, xy], [y^2, x^2], [x^2-1, 0], [y^3-y, 0], [0, y^2 x-x]]$$

```
> JTabVar();
```

$$[[0, y^2-1], [*, y], [y^2, 2]]$$
$$[[xy, xy], [*, y], [xy, 1]]$$
$$[[y^2, x^2], [x, y], [x^2, 2]]$$
$$[[x^2-1, 0], [x, y], [x^2, 1]]$$
$$[[y^3-y, 0], [*, y], [y^3, 1]]$$
$$[[0, y^2 x-x], [*, y], [y^2 x, 2]]$$

See Also:

JBasis, JZeroSets, JanetOreStats, JanetOreOptions, JInvReduce.

JanetOre[JWeightedHilbertSeries] - Hilbert series of the module generated by the last JBasis command (weighted version)

Calling Sequence:

JWeightedHilbertSeries(degrees,v)

Parameters:

`degrees` - list of variables associated with degrees (weights)
`v` - (optional) name of the indeterminate (default 's')

Description:

- The command ***JWeightedHilbertSeries*** is completely analogous to JHilbertSeries with the only exception that the standard grading of the left module of *m*-tuples over the Ore algebra is replaced by the grading defined by **`degrees`**. The information is derived from the last call of JBasis.
- The parameter **`degrees`** is a list of the form [<variable1>=<degree1>, <variable2>=<degree2>, ...]. In this way a degree is assigned to each variable. In the module case degrees other than 0 can also be assigned to the standard basis vectors of the free left module. The above syntax is therefore extended to $[x_1=d_1, ..., x_n=d_n, 1=e_1, ..., m=e_m]$, cf. also JBasis.
- The special case where all degrees are equal to 1 yields the same result (in a different expansion) as JHilbertSeries.
- The default name of the indeterminate is 's'. It will not be affected by a subs command.

Examples:

```
> with(JanetOre):
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x*y+y*z+z*x,x*y*z-1];
```

$$L := [xy+yz+zx, xyz-1]$$

```
> JBasis(L, var);
```

$$[xy+yz+zx, yz^2+z^2x+1, z^2y^2+y+z]$$

```
> JWeightedHilbertSeries([x=1,y=1,z=1], lambda);
```

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

```
> taylor(%, lambda, 7);
```

$$1+3\lambda+5\lambda^2+6\lambda^3+6\lambda^4+6\lambda^5+6\lambda^6+O(\lambda^7)$$

```
> JHilbertSeries(lambda);
```

$$1+3\lambda+5\lambda^2+6\lambda^3+6\frac{\lambda^4}{1-\lambda}$$

The next examples deals with graded modules:

```
> var := [x=2,y=1,1=0,2=3];
```

$$var := [x=2, y=1, 1=0, 2=3]$$

```
> L := [[x^2*y^2, x*y], [x^3-y^6,y^3]];
```

$$L := [[x^2y^2, xy], [x^3-y^6, y^3]]$$

```
> JBasis(L, var);
```

$$[[x^2y^2, xy], [x^3-y^6, y^3], [y^8, -y^5+x^2y], [xy^8, yx^3-y^5x], [0, -y^7x+yx^4-y^5x^2]]$$

```
> JWeightedHilbertSeries(var, lambda);
```

$$2\lambda^7+2\lambda^6+3\lambda^5+3\lambda^4+2\lambda^3+2\lambda^2+\lambda+1+\lambda^9+\lambda^8+\frac{\lambda^{11}}{1-\lambda^2}+\frac{\lambda^9}{1-\lambda}+\frac{\lambda^7}{1-\lambda}+\frac{\lambda^5}{1-\lambda}+\frac{\lambda^3}{1-\lambda}$$

```
> F := JFactorModuleBasis(var);
```

$$F := \left[y^7+y^6+y^5+y^4+y^3+y^2+y+1+y^7x+xy^6+y^5x+xy^4+xy^3+xy^2+xy+x+x^2y+x^2, \frac{x^4}{1-x}+\frac{x^3}{1-y}+\frac{x^2}{1-y}+\frac{x}{1-y}+\frac{1}{1-y}\right]$$

```
> subs([x=lambda^2, y=lambda], F[1]+F[2]*lambda^3);
```

$$2\lambda^7+2\lambda^6+3\lambda^5+3\lambda^4+2\lambda^3+2\lambda^2+\lambda+1+\lambda^9+\lambda^8+\lambda^3\left(\frac{\lambda^8}{1-\lambda^2}+\frac{\lambda^6}{1-\lambda}+\frac{\lambda^4}{1-\lambda}+\frac{\lambda^2}{1-\lambda}+\frac{1}{1-\lambda}\right)$$

See Also:

JBasis, JTabVar, JHilbertSeries, JHilbertPolynomial, JHilbertFunction, JCartanCharacters, JFactorModuleBasis.

JanetOre[JZeroSets] - return the coefficients by which the involutive basis algorithm had to divide

Calling Sequence:

JZeroSets()

Parameters:

- - none (assumes that the involutive basis has been computed before)

Description:

- ***JZeroSets*** returns the list of elements which are transcendental over the ground field of the last involutive basis computation and by which some polynomials had to be divided during this last involutive basis computation. This command is useful when applying JBasis to a module defined over a polynomial ring whose coefficient domain consists of rational functions.
- The result of ***JZeroSets*** is a list which does not contain multiple entries, i.e. each (transcendental) denominator of the last involutive basis computation occurs only once in the resulting list.
- The purpose of ***JZeroSets*** is similar to that of ZeroSets in the Janet package.

Example:

```
> with(JanetOre):
```

Example 1:

```
> var := [x,y];
```

$$var := [x, y]$$

```
> L := [a*x*y-a, x-b*y];
```

$$L := [a\,x\,y - a, x - b\,y]$$

```
> JBasis(L, var);
```

$$\left[x - b\,y, y^2 - \frac{1}{b}\right]$$

```
> JTabVar();
```

$$[x - b\,y, [x, y], x]$$
$$\left[y^2 - \frac{1}{b}, [*, y], y^2\right]$$

```
> JZeroSets();
```

$$[a, b]$$

```
> JBasis(L, var, "N");
```

$$[x - b\,y, b\,y^2 - 1]$$

```
> JZeroSets();
```

$$[a]$$

Example 2:

```
> var := [x,y,z];
```

$$var := [x, y, z]$$

```
> L := [x^2-y, x*y-a*z];
```

$$L := [x^2 - y, x\,y - a\,z]$$

```
> JBasis(L, var);
```

$$[y^2 - a\,z\,x, x\,y - a\,z, x^2 - y]$$

```
> JTabVar();
```

$$[y^2 - a\,z\,x, [*, y, z], y^2]$$
$$[x\,y - a\,z, [*, y, z], x\,y]$$
$$[x^2 - y, [x, y, z], x^2]$$

```
> JZeroSets();
```

[*a*]

See Also:

JBasis, JanetOreOptions, JTabVar, JanetOreStats, JInvReduce, JFactorModuleBasis, JHilbertSeries, ZeroSets