Introduction to the Involutive package

Calling Sequence:

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

Description:

- The Involutive package provides algorithms for the involutive analysis of ideals in commutative polynomial rings and more generally for submodules of free modules over polynomial rings. Its main purpose is the analysis of systems of polynomial equations.
- The main algorithm designed by Gerdt and Blinkov is a substantial improvement of Janet's algorithm for analysing systems of linear partial differential equations adapted to polynomial equations. This is based on the observation that both systems of linear partial differential equations with constant coefficients and polynomial equations are two different languages to talk about submodules of free modules over polynomial rings.
- The main algorithm for this package, called InvolutiveBasis, produces standard generators for submodules of free modules over polynomial rings, 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 PolInvReduce to produce a normal form for representatives of residue classes. The polynomial ring is defined over (a field extension of) the rational numbers by default. By means of the command InvolutiveOptions it can be changed to integer coefficients or coefficients in (an extension of) a field of non-zero characteristic.
- Involutive bases are special Groebner bases, 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 PolTabVar.
- To use a function of the Involutive package, either define that function alone using the command with(Involutive, <function>), or define all Involutive functions using the command with(Involutive). Alternatively, invoke the function using the long form Involutive[<function>].
- The functions available in the Involutive package are the following:

Basic commands:

InvolutiveBasis	InvolutiveBasisFast	InvolutiveBasisGINV
PolInvReduce	PolInvReduceFast	PolInvReduceGINV
PolTabVar	PolHilbertSeries	
FactorModuleBasis	SubmoduleBasis	

Further commands for the computation of involutive bases:

AddRhs	AssertInvBasis	InvolutiveOptions
InvolutivePreprocess	Substitute	PolZeroSets

Commands for special applications:

PolMinPoly	Syzygies	PolResolution
SyzygyModule	SyzygyModuleFast	SyzygyModuleGINV
NoetherNormalization	PolShorterResolution	PolShortestResolution
PolResolutionDim	PolEulerChar	PolRepres
Annihilator	PolCoeff	
Repres	PolWeightedHilbertSeries	
PolLeftInverse	PolRightInverse	
PolSyzOp	CoeffList	
PolFactorize	NotHas/Has	

Commands for module theory:

PolSum	PolDirectSum	PolIntersection
PolSubFactor	PolCheckHom	PolDefect
PolHom	PolHomHom	
PolKernel	PolCokernel	
PolExt1	PolExtn	
PolParametrization	PolTorsion	

Commands for various invariants derivable from `PolHilbertSeries` or `SubmoduleHilbertSeries`:

PolIndexRegularity	PolDimension	
PolHilbertPolynomial	PolHilbertFunction	
PolHP	PolHF	
PolCartanCharacter	SubmoduleDimension	
SubmoduleHF	SubmoduleHP	
SubmoduleHilbertFunction	SubmoduleHilbertPolynomial	
SubmoduleHilbertSeries		

Alternate Groebner basis commands:

GroebnerBasis	GroebnerBasisFast	GroebnerBasisGINV

Auxiliary commands:

LeadingMonomial	JanetGraph	Stats

- 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 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. For more applications, see 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.

Examples:

```
> with(Involutive):
```

First we choose the variables of the considered polynomial ring **R**:

```
> var := [x1,x2,x3];
```

$$var := [x1, x2, x3]$$

We want to calculate the Janet basis for the polynomial ideal **I** generated by the following polynomials:

```
> L := [x1+x2+x3-a1, x1*x2 + x2*x3 + x3*x1-a2, x1*x2*x3-a3];
```

$$L := [x1 + x2 + x3 - a1, x1\,x2 + x2\,x3 + x3\,x1 - a2, x1\,x2\,x3 - a3]$$

The Janet basis is computed w. r. t. degree reverse lexicographical order:

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

$$IB := [x1 + x2 + x3 - a1, x2^2 + x2\,x3 + x3^2 - a1\,x2 - a1\,x3 + a2, -a3 + x3^3 - a1\,x3^2 + a2\,x3, -a3\,x2 + x3^3\,x2 - a1\,x3^2\,x2 + a2\,x3\,x2]$$

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

```
> PolTabVar();
```

$$[x1 + x2 + x3 - a1, [x1, x2, x3], x1]$$

$$[x2^2 + x2\,x3 + x3^2 - a1\,x2 - a1\,x3 + a2, [*, x2, x3], x2^2]$$

$$[-a3 + x3^3 - a1\,x3^2 + a2\,x3, [*, *, x3], x3^3]$$

$$[-a3\,x2 + x3^3\,x2 - a1\,x3^2\,x2 + a2\,x3\,x2, [*, *, x3], x3^3\,x2]$$

Compute the Hilbert series of the quotient ring **R/I**:

```
> PolHilbertSeries();
```

$$1 + 2\,s + 2\,s^2 + s^3$$

Next problem: Find the normal forms of the residue classes in **R/I** which contain the following polynomials:

```
> PolInvReduce(x1, IB, var);
  PolInvReduce(x1^2, IB, var);
  PolInvReduce(x1^3, IB, var);
```

$$-x2 - x3 + a1$$

$$a1^2 - a2 - a1\,x3 - a1\,x2 + x2\,x3$$

$$a3 + a1^3 - 2\,a2\,a1 - a1^2\,x3 + a2\,x3 - a1^2\,x2 + a2\,x2 + a1\,x3\,x2$$

```
> PolInvReduce(x1^3-a1*x1^2+a2*x1-a3, IB, var);
```

0

See Also:

with, Janet, JanetOre

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

Calling Sequence:

AddRhs(L,R)

Parameters:

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

Description:

- ***AddRhs*** 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(Involutive):
> L := [x^2,x+y,z^3];
```

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

```
> AddRhs(L);
```

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

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

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

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

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

```
> AddRhs(M);
```

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

See Also:

InvolutiveBasis, Syzygies.

Involutive[Annihilator] - return involutive basis of the annihilator of a submodule of a finitely presented module over a polynomial ring

Calling Sequence:

Annihilator(p,L,var)

Parameters:

`p` - (list (of lists of the same length) of) polynomial(s)
`L` - list (of lists of the same length) of polynomials
`var` - list of variables of the polynomial ring

Description:

- ***Annihilator*** computes the annihilator of the submodule generated by the residue class(es) of **p** in the module presented by **L** over the polynomial ring with variables **var**, i.e. the ideal of those elements in the polynomial ring which satisfy that their product by the module generated by the residue class(es) of **p** lies in the module presented by **L**.
- Residue classes are taken in the factor module given by the free module of tuples over the polynomial ring in **var** modulo the submodule generated by **L**. This means that the annihilator consists of those elements in the polynomial ring whose associated multiplication map sends all elements of the module generated by the residue class(es) of **p** to zero in this factor module.
- 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. In the first case, **p** may be a polynomial. Then the annihilator of the module generated by the residue class of **p** is computed. If *m* is greater than 1, then **p** may be a list of polynomials in which case the annihilator of the module generated by the residue class of **p** is computed. In general, **p** is a list or a list of lists of the same length of polynomials according to the value of *m*. Then the annihilator of the module generated by the residue classes of the elements of **p** is computed.
- The result of ***Annihilator*** is an involutive basis of the annihilator defined above.

Examples:

```
> with(Involutive):
```

Example 1:

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

$$var := [x]$$

```
> Annihilator(x-1, [x^2-1], var);
```

$$[x+1]$$

Example 2:

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

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

```
> Annihilator([x^3, 0], [[x^4, 0], [0, x^4]], var);
```

$$[x]$$

```
> Annihilator([x^3, x^2], [[x^4, 0], [0, x^4]], var);
```

$$[x^2]$$

Example 3:

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

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

```
> P := [x-1, y-2];
```

$$P := [x-1, y-2]$$

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

$$L := [x^2 - 1, yx - 2x + y - 2]$$

```
> Annihilator(P, L, var);
```

$$[x + 1]$$

Example 4:

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

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

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

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

```
> L := [[x+1, 0], [0, y*x-2*x+y-2]];
```

$$L := [[x + 1, 0], [0, yx - 2x + y - 2]]$$

```
> Annihilator(P, L, var);
```

$$[x + 1]$$

See Also:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolHom, PolHomHom, PolExt1, PolExtn, PolTorsion, PolParametrization, PolSyzOp.

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

Calling Sequence:

AssertInvBasis(L,var,ord,mode)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the polynomial ring)
`ord` - (optional) change of monomial ordering (see below)
`mode` - (optional) string specifying options for the computation

Description:

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

Examples:

```
> with(Involutive):
```

Example 1: Working with two Janet bases

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

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

```
> L1 := [x-y];
```

$$L1 := [x - y]$$

```
> J1 := InvolutiveBasis(L1, var):
> PolHilbertSeries(t);
```

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

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

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

```
> J2 := InvolutiveBasis(L2, var):
> PolHilbertSeries(t);
```

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

```
> AssertInvBasis(J1, var):
> PolHilbertSeries(t);
```

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

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

```
> 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 := InvolutiveBasisFast(L, var);
```

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

```
> AssertInvBasis(IB, var):
> FactorModuleBasis(var);
```

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

```
> PolHilbertSeries(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:

InvolutiveBasis, PolTabVar, InvolutiveOptions, PolInvReduce, FactorModuleBasis, PolHilbertSeries, PolWeightedHilbertSeries, Syzygies, SyzygyModule, PolCartanCharacter.

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

Calling Sequence:

CoeffList(p,var,B)

Parameters:

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

Description:

- ***CoeffList*** 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 polynomial ring.
- The list **B** is expected to be a result of a previous call of FactorModuleBasis. If **p** is a polynomial, then **B** is expected to be a factor module basis computed for a residue class module of the polynomial ring in **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 of polynomials in **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 ***CoeffList***. 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(Involutive):
```

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);
```

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

```
> F := FactorModuleBasis(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;
```

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

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

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

The next polynomial is not in the span of F.

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

$$p := x+1$$

```
> CoeffList(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]];
```

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

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

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

```
> F := FactorModuleBasis(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];
```

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

```
> CoeffList(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];
```

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

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

$$[0, x^2]$$

Example 3:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

See Also:

InvolutiveBasis, PolInvReduce, FactorModuleBasis, SubmoduleBasis, PolRepres, Repres.

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

Calling Sequence:

FactorModuleBasis(var,mode)

Parameters:

`var` - list of variables (of the polynomial ring)
`mode` - (optional) string specifying options

Description:

- ***FactorModuleBasis*** returns a vector space basis for the residue class module of the free module over the polynomial ring modulo the submodule generated by the Janet basis of the last call of InvolutiveBasis, 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 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 module. See also the explanation of the option "G" below.
- **`var`** is the list of variables of the polynomial ring that was given as parameter to ***InvolutiveBasis*** 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 ***FactorModuleBasis*** 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 ***FactorModuleBasis*** 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 InvolutiveBasisFast or InvolutiveBasisGINV has been called before. In this case ***FactorModuleBasis*** determines its result using the output data of the last call of ***InvolutiveBasisFast*** resp. ***InvolutiveBasisGINV***, whichever was called last. This means that in this case it is not necessary to update the internal data structure for the current involutive basis using AssertInvBasis before calling ***FactorModuleBasis*** (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 ***InvolutiveBasisFast*** resp. ***InvolutiveBasisGINV***, so that the results of ***FactorModuleBasis*** 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).
- For more information about factor module bases, 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(Involutive):
```

Example 1:

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

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

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

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

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

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

```
> FactorModuleBasis(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:

```
> PolHilbertSeries();
```

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

Example 2:

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

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

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

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

$$L := [xy]$$

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

$$[xy]$$

```
> FactorModuleBasis(var);
```

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

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

$$[1, x]$$

Example 3:

Example for a module over the polynomial ring Q[x,y]:

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

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

```
> L := [[x^2, 0], [y^2, 0], [0, x*y^2], [0, x^3], [0, y^4]];
```

$$L := [[x^2, 0], [y^2, 0], [0, xy^2], [0, x^3], [0, y^4]]$$

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

$$[[y^2, 0], [x^2, 0], [0, xy^2], [xy^2, 0], [0, x^3], [0, y^4], [0, x^2 y^2]]$$

```
> PolHilbertSeries();
```

$$2 + 4s + 4s^2 + 2s^3$$

```
> FactorModuleBasis(var);
```

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

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

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

Example 4:

```
> restart;
> with(Involutive):
> InvolutiveBasisFast([x*y], [x,y]);
```

$$[xy]$$

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

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

See Also:

InvolutiveBasis, PolTabVar, JanetGraph, SubmoduleBasis, PolHilbertSeries, PolWeightedHilbertSeries, PolMinPoly, PolRepres, CoeffList.

Involutive[GroebnerBasis] - return minimal Groebner basis of a submodule of a free module over a polynomial ring

Calling Sequence:

GroebnerBasis(L,var,ord,mode,opt)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the polynomial ring)
`ord` - (optional) change of monomial ordering
`mode` - (optional) string specifying options for the computation
`opt` - (optional) equation specifying options for the computation

Description:

- ***GroebnerBasis*** returns the minimal Groebner basis of a submodule of the free module of *m*-tuples of polynomials given by the generators in **L** with respect to a certain ordering. 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, ***GroebnerBasis*** passes its arguments to InvolutiveBasis and extracts the minimal Groebner basis from this result. Therefore, Janet's data (see PolTabVar) contains information about the involutive basis of the given submodule.
- All parameters to ***GroebnerBasis*** have the same meaning as in InvolutiveBasis.
- 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 InvolutiveOptions one can also choose between two implementations of ***GroebnerBasis***: "Maple" and "C++".

Examples:

```
> with(Involutive):
> 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);
```

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

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

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

```
> PolTabVar();
```

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

See Also:

InvolutiveBasis, GroebnerBasisFast, GroebnerBasisGINV, PolTabVar, JanetGraph, InvolutiveOptions, LeadingMonomial

Involutive[GroebnerBasisFast] - return minimal Groebner basis of a submodule of a free module over a polynomial ring (C++ version)

Calling Sequence:

GroebnerBasisFast(L,var,ord,mode,opt)

Parameters:

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

Description:

- ***GroebnerBasisFast*** computes the minimal Groebner basis of a submodule of the free module of *m*-tuples of polynomials given by the generators in **`L`** with respect to a certain ordering using InvolutiveBasisFast instead of InvolutiveBasis (cf. GroebnerBasis). Up to now, only the algorithm for the degree reverse lexicographical ordering (i.e. **`ord`** is 2 or 4) is implemented in C++. If ***GroebnerBasisFast*** is called choosing a different monomial ordering, then InvolutiveBasis is applied instead of InvolutiveBasisFast.
- All parameters to ***GroebnerBasisFast*** have the same meaning as in InvolutiveBasisFast.
- 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 "JB" instead.)
- Using the option "C++" of InvolutiveOptions, the command ***GroebnerBasis*** is replaced by ***GroebnerBasisFast*** for the current Maple session.

Examples:

```
> with(Involutive):
> 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);
```

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

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

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

See Also:

InvolutiveBasis, InvolutiveBasisFast, GroebnerBasisGINV, InvolutiveOptions, PolTabVar, FactorModuleBasis, PolInvReduce, PolInvReduceFast, GroebnerBasis, SyzygyModule, SyzygyModuleFast.

Involutive[GroebnerBasisGINV] - Python/C++ version of GroebnerBasis

Calling Sequence:

GroebnerBasisGINV(L,var,ord,mode,opt)

Parameters:

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

- ***GroebnerBasisGINV*** computes the minimal Groebner basis of a submodule of the free module of *m*-tuples of polynomials given by the generators in **L** with respect to a certain ordering using InvolutiveBasisGINV instead of InvolutiveBasis (cf. GroebnerBasis), i.e. ***GroebnerBasisGINV*** is a version of the command GroebnerBasis which uses the C++ module **ginv** for Python to perform the involutive basis computation.
- The parameters **L**, **var**, **ord**, and **mode** have the same meaning as in GroebnerBasis.
- 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 corresponding process "python" instead.)
- Possible left hand sides of the optional equations **opt** are the strings "char", "algext", "time", "Name", "denom", "donotread", "MovedBound", and "QlengthBound".
- 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 Groebner basis in characteristic *c*. If further commands like PolMinPoly shall be applied afterwards, the characteristic of the ground field must be changed by using the command InvolutiveOptions.
- The right hand side of an equation "algext"=*p* in **opt** is expected to be a univariate polynomial in an indeterminate ζ which does not occur in **var**. The coefficients of *p* must be algebraic over the ground field in the sense that they are rational expressions in RootOf and indeterminates ξ used in the right hand sides of other equations "algext"=*q* in **opt**. This extends the ground field (defined so far) by ζ which has minimal polynomial *p*, i.e. every occurrence of ζ in **L** is subject to the relation $p = 0$ (cf. Example 3).
- 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 4).
- The right hand side of an equation "Name"=s is expected to be a string. ***GroebnerBasisGINV*** appends *s* to the default name for the temporary file to which the input for **ginv** is written.
- For more information about **ginv**, cf. http://invo.jinr.ru and http://wwwb.math.rwth-aachen.de/Janet.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

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

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

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

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

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

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];
```

$$L := [x+2y+3z, xy+2yz+3zx, xyz-1]$$

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

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

Example 3:

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

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

```
> alias(omega=RootOf(Z^2+Z+1));
```

$$\omega$$

```
> simplify(omega^3);
```

$$1$$

```
> factor(zeta^3+omega*zeta+1, omega);
```

$$\zeta^3+\omega\,\zeta+1$$

```
> minpoly1 := zeta^3+omega*zeta+1;
```

$$minpoly1 := \zeta^3+\omega\,\zeta+1$$

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

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

```
> GroebnerBasisGINV(L, var, "algext"=minpoly1);
```

$$[x^2-y^2, xy^2+(\zeta^2+\omega)y^3, y^4]$$

```
> J := GroebnerBasisGINV(AddRhs(L), var, "algext"=minpoly1);
```

$$J := \left[x^2-y^2=[1,0], xy^2+(\zeta^2+\omega)y^3=[-x, \zeta^2+\omega], y^4=\left[\left(\left(-\frac{2\omega}{3}-\frac{1}{3}\right)\zeta^2+\left(\frac{\omega}{3}+\frac{2}{3}\right)\zeta-\frac{\omega}{3}-\frac{2}{3}\right)x^2+\left(\left(\frac{\omega}{3}+\frac{2}{3}\right)\zeta^2+\left(-\frac{2\omega}{3}-\frac{1}{3}\right)\zeta+\frac{2\omega}{3}+\frac{1}{3}\right)xy+\left(\left(-\frac{2\omega}{3}-\frac{1}{3}\right)\zeta^2+\left(\frac{\omega}{3}+\frac{2}{3}\right)\zeta-\frac{\omega}{3}-\frac{2}{3}\right)y^2, \left(\left(\frac{\omega}{3}+\frac{2}{3}\right)\zeta^2+\left(-\frac{2\omega}{3}-\frac{1}{3}\right)\zeta+\frac{2\omega}{3}+\frac{1}{3}\right)x+\left(\left(-\frac{2\omega}{3}-\frac{1}{3}\right)\zeta^2+\left(\frac{\omega}{3}+\frac{2}{3}\right)\zeta-\frac{\omega}{3}+\frac{1}{3}\right)y\right]\right]$$

```
> simplify(rem(expand(rhs(J[3])[1] * L[1] + rhs(J[3])[2] * L[2]), minpoly1, zeta));
```

$$y^4$$

See Also:

InvolutiveBasis, InvolutiveBasisGINV, AssertInvBasis, GroebnerBasisFast, InvolutiveOptions, PolTabVar, FactorModuleBasis, PolInvReduce, PolInvReduceFast, Syzygies, SyzygyModule, PolHilbertSeries.

Involutive[InvolutiveBasis] - return the (unique) minimal Janet basis of a submodule of a free module over a polynomial ring

Calling Sequence:

InvolutiveBasis(L,var,ord,mode,opt)

Parameters:

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

- ***InvolutiveBasis*** returns the (unique) minimal Janet basis of the submodule of the free module of *m*-tuples of polynomials given by the generators in **L** with respect to a certain ordering. The default ordering is degree reverse lexicographical. ***InvolutiveBasis*** is the main function of the package Involutive.
- The polynomial ring is defined over (a field extension of) the rational numbers by default. By means of the command InvolutiveOptions 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 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 parameter **var** is a list specifying the variables of the polynomial ring. 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 output is a list containing the Janet basis for the submodule generated by **L** with respect to the chosen ordering.
- ***InvolutiveBasis*** saves the information in an internal data structure which can be displayed by PolTabVar.
- As optional third 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 LeadingMonomial.)
- 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 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 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 InvolutiveOptions. It is also possible to compute involutive bases of the ring of integers. By means of the command InvolutiveOptions one can also choose between three implementations of ***InvolutiveBasis***: "Maple", "C++", and "GINV".
- One can specify a right hand side for each generator in order to let ***InvolutiveBasis*** 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 **P_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 Syzygies.
- 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 (cf. Involutive).
- ***InvolutiveBasis*** 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 PolWeightedHilbertSeries instead of PolHilbertSeries. For examples that deal with leading monomials in the case of non-standard degrees see LeadingMonomial.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

```
> L1 := [x+y+z, x*y+y*z+z*x, x*y*z-1];
```

$$L1 := [x+y+z, xy+yz+zx, xyz-1]$$

```
> InvolutiveBasis(L1, var);
```

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

```
> PolTabVar();
```

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

```
> InvolutiveBasis(L1, var, 1);
```

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

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

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

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

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

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

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]]);
```

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

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

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

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];
```

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

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

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

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

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

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

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

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

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

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

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

```
]
```

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];
```

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

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

$$[1]$$

```
> PolZeroSets();
```

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

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

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

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

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

```
> 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];
```

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

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

$$\left[x+\omega y-z-z\,\omega,\ y^2+2yz+2yz\,\omega+z^2+2z^2\,\omega+z^2\,\omega^2,\ \frac{2}{3}+\frac{1}{3}\omega+yz^2+\frac{1}{3}z^3+\frac{2}{3}z^3\,\omega,\ \frac{1}{3}(-1+\omega)(5z+4z\,\omega-2z^4+3y-z^4\,\omega)\right]$$

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

$$[x+\omega y+(-1-\omega)z,\ -\omega y^2+2yz+(1+\omega)z^2,\ -2-\omega-3yz^2+(-1-2\,\omega)z^3,\ -3y+(-4\,\omega-5)z+(2+\omega)z^4]$$

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

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

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

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

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

```
> PolWeightedHilbertSeries([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]];
```

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

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

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

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

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

Example 7: Block ordering

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

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

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

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

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

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

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)];
```

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

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

See Also:

PolTabVar, LeadingMonomial, JanetGraph, InvolutiveBasisFast, InvolutiveBasisGINV, GroebnerBasis, InvolutiveOptions, PolInvReduce, FactorModuleBasis, PolHilbertSeries, PolWeightedHilbertSeries, Syzygies, SyzygyModule, PolCartanCharacter, Has.

Involutive[InvolutiveBasisFast] - return the (unique) minimal Janet basis of a submodule of a free module over a polynomial ring (C++ version)

Calling Sequence:

InvolutiveBasisFast(L,var,ord,mode,opt)

Parameters:

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

- ***InvolutiveBasisFast*** invokes the C++ version of the command InvolutiveBasis. Up to now, only the algorithm for the degree reverse lexicographical ordering (i.e. **`ord`** is 2 or 4) is implemented in C++. If ***InvolutiveBasisFast*** is called choosing a different monomial ordering, then InvolutiveBasis is applied internally to the same data.
- The parameters **`L`**, **`var`**, **`ord`**, and **`mode`** have the same meaning as in InvolutiveBasis.
- 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 "JB" instead.)
- The output of the C++ program is read into the current Maple session. To continue with commands that expect a previous run of InvolutiveBasis (like PolTabVar, FactorModuleBasis, PolHilbertSeries, etc.) the internal data structure for the involutive basis has to be set up by the command AssertInvBasis (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 PolMinPoly shall be applied afterwards, the characteristic of the ground field must be changed by using the command InvolutiveOptions.
- 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. ***InvolutiveBasisFast*** appends *s* to the default name for the temporary file to which the input for the external program JB 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 PolZeroSets. 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 ***InvolutiveBasisFast*** does not read the result produced by the C++ program and does not return a result.
- Using the option "C++" of InvolutiveOptions, the command ***InvolutiveBasis*** is replaced by ***InvolutiveBasisFast*** (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 ***InvolutiveBasis***).

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

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

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

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

```
> AssertInvBasis(IB, var);
```

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

```
> PolTabVar();
```

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

```
> FactorModuleBasis(var);
```

$$[1, z, y, z^2, yz, yz^2]$$

```
> PolHilbertSeries(lambda);
```

$$1+2\lambda+2\lambda^2+\lambda^3$$

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];
```

$$L := [x+2y+3z, xy+2yz+3zx, xyz-1]$$

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

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

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))];
```

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

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

8199

Example 4:

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

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

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

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

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

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

```
> PolZeroSets();
```

$$[5, 3]$$

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

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

```
> PolZeroSets();
```

[5]

See Also:

InvolutiveBasis, AssertInvBasis, InvolutiveBasisGINV, InvolutiveOptions, PolTabVar, FactorModuleBasis, PolInvReduce, PolInvReduceFast, Syzygies, SyzygyModule, SyzygyModuleFast, PolHilbertSeries.

Involutive[InvolutiveBasisGINV] - Python/C++ version of InvolutiveBasis

Calling Sequence:

InvolutiveBasisGINV(L,var,ord,mode,opt)

Parameters:

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

- ***InvolutiveBasisGINV*** invokes the version of the command InvolutiveBasis which uses the C++ module **ginv** for Python to perform the involutive basis computation.
- The parameters **L**, **var**, **ord**, and **mode** have the same meaning as in InvolutiveBasis.
- 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 corresponding process "python" instead.)
- The output of python is read into the current Maple session. To continue with commands that expect a previous run of InvolutiveBasis (like PolTabVar, FactorModuleBasis, PolHilbertSeries, etc.) the internal data structure for the involutive basis has to be set up by the command AssertInvBasis (cf. example below).
- Possible left hand sides of the optional equations **opt** are the strings "char", "algext", "transext", "time", "Name", "quiet", "denom", "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 PolMinPoly shall be applied afterwards, the characteristic of the ground field must be changed by using the command InvolutiveOptions.
- The right hand side of an equation "algext"=*p* in **opt** is expected to be a univariate polynomial in an indeterminate ζ which does not occur in **var**. The coefficients of *p* must be algebraic over the ground field in the sense that they are rational expressions in RootOf and indeterminates ξ used in previously given right hand sides of other equations "algext"=*q* in **opt**. This extends the ground field (defined so far) by ζ which has minimal polynomial *p*, i.e. every occurrence of ζ in **L** is subject to the relation $p = 0$ (cf. Example 3).
- The right hand side of an equation "transext"=*z* in **opt** is expected to be a name for an indeterminate. This extends the ground field (defined so far) by a new transcendental element *z*.
- 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 4).
- The right hand side of an equation "Name"=*s* is expected to be a string. ***InvolutiveBasisGINV*** appends *s* to the default name for the temporary file to which the input for **ginv** is written.
- As right hand side of an equation "quiet"=*b* in **opt**, a boolean value *b* is expected. The default value is false. If *b* equals true, then no intermediate output is produced on the screen by the Python/C++ program.
- 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 Python/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 PolZeroSets. See also Example 5 below.
- The right hand side of an equation "donotread"=b is expected to be a boolean value. If b equals true, then ***InvolutiveBasisGINV*** does

not read the result produced by the Python/C++ program and does not return a result.

- Using the option "GINV" of InvolutiveOptions, the command ***InvolutiveBasis*** is replaced by ***InvolutiveBasisGINV*** for the current Maple session (which has the corresponding effect on all Maple procedures that call ***InvolutiveBasis***).
- For more information about **ginv**, cf. http://invo.jinr.ru and http://wwwb.math.rwth-aachen.de/Janet.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

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

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

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

```
> AssertInvBasis(IB, var);
```

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

```
> PolTabVar();
```

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

```
> FactorModuleBasis(var);
```

$$[1, z, y, z^2, yz, yz^2]$$

```
> PolHilbertSeries(lambda);
```

$$1+2\lambda+2\lambda^2+\lambda^3$$

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];
```

$$L := [x+2y+3z, xy+2yz+3zx, xyz-1]$$

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

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

Example 3:

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

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

```
> alias(omega=RootOf(Z^2+Z+1));
```

$$\omega$$

```
> simplify(omega^3);
```

$$1$$

```
> factor(zeta^3+omega*zeta+1, omega);
```

$$\zeta^3+\omega\zeta+1$$

```
> minpoly1 := zeta^3+omega*zeta+1;
```

$$minpoly1 := \zeta^3+\omega\zeta+1$$

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

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

```
> InvolutiveBasisGINV(L, var, "algext"=minpoly1);
```

$$[x^2-y^2, xy^2+(\zeta^2+\omega)y^3, y^4]$$

```
> J := InvolutiveBasisGINV(AddRhs(L), var, "algext"=minpoly1);
```

$$J := \left[x^2 - y^2 = [1, 0], xy^2 + (\zeta^2 + \omega)y^3 = [-x, \zeta^2 + \omega], y^4 = \left[\left(\left(-\frac{2\omega}{3} - \frac{1}{3} \right)\zeta^2 + \left(\frac{\omega}{3} + \frac{2}{3} \right)\zeta - \frac{\omega}{3} - \frac{2}{3} \right)x^2 + \left(\left(\frac{\omega}{3} + \frac{2}{3} \right)\zeta^2 + \left(-\frac{2\omega}{3} - \frac{1}{3} \right)\zeta + \frac{2\omega}{3} + \frac{1}{3} \right)xy + \left(\left(-\frac{2\omega}{3} - \frac{1}{3} \right)\zeta^2 + \left(\frac{\omega}{3} + \frac{2}{3} \right)\zeta - \frac{\omega}{3} - \frac{2}{3} \right)y^2, \left(\left(\frac{\omega}{3} + \frac{2}{3} \right)\zeta^2 + \left(-\frac{2\omega}{3} - \frac{1}{3} \right)\zeta + \frac{2\omega}{3} + \frac{1}{3} \right)x + \left(\left(-\frac{2\omega}{3} - \frac{1}{3} \right)\zeta^2 + \left(\frac{\omega}{3} + \frac{2}{3} \right)\zeta - \frac{\omega}{3} + \frac{1}{3} \right)y \right] \right]$$

```
> simplify(rem(expand(rhs(J[3])[1] * L[1] + rhs(J[3])[2] * L[2]), minpoly1, zeta));
```

$$y^4$$

Example 4:

```
> var := [seq(a[i], i=1..10)];
```

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

```
> L := [seq(a[i]^5-a[i+1]^4, i=1..9),seq(a[i]^3-a[i-1]^2, i=2..10)];
```

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

```
> J := InvolutiveBasisGINV(L, var, "time"=1):
Warning, computation of involutive basis stopped due to time restriction.
> J := InvolutiveBasisGINV(L, var):
> AssertInvBasis(J, var):
> nops(FactorModuleBasis(var));
```

2055

Example 5:

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

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

```
> L := [a*x*y-a, x-b*y];
```

$$L := [a\,xy - a, x - b\,y]$$

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

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

```
> PolZeroSets();
```

$$[a, b]$$

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

$$[x - b\,y, b\,y^2 - 1]$$

```
> PolZeroSets();
```

$$[a]$$

See Also:

InvolutiveBasis, AssertInvBasis, InvolutiveBasisFast, GroebnerBasisGINV, InvolutiveOptions, PolTabVar, FactorModuleBasis, PolInvReduce, PolInvReduceGINV, Syzygies, SyzygyModule, PolHilbertSeries.

Involutive[InvolutiveOptions] - set up the options for the current session of Involutive

Calling Sequence:

InvolutiveOptions(s,v)

Parameters:

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

Description:

- ***InvolutiveOptions*** sets up the options for the current session of Involutive. 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"
"C++"               "GINV"              "Maple"
"matrix"            "JanetLike"         "criteria"
"InvBasis"          "InvReduce"         "SyzygyModule"
"GBasis"
```

- If the first parameter **s** is one of the strings "C++", "GINV", or "Maple", then ***InvolutiveOptions*** has no return value. For all other possible values of the first parameter **s**, ***InvolutiveOptions*** returns the current value of the option specified by **s**. In these latter cases, if no second parameter **v** is provided, the value of the option addressed by **s** is not changed.

- If **s** equals the string "char", then a second parameter **v** is expected to be zero or a prime number. Subsequent computations of the ***Involutive*** package are done in characteristic **v** then. The return value of ***InvolutiveOptions*** is the characteristic of the ground field used by the ***Involutive*** 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 ***InvolutiveOptions*** 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 ***InvolutiveOptions*** in this case is the former value of the option "AbsolutelySmallestRemainder".

- The keywords "C++", "GINV", 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 InvolutiveBasis becomes a synonym for InvolutiveBasisFast, PolInvReduce a synonym for PolInvReduceFast, SyzygyModule a synonym for SyzygyModuleFast, and GroebnerBasis a synonym for GroebnerBasisFast, i.e. all these basic commands invoke the external C++ routines. Note that in this case ***InvolutiveBasis*** and ***SyzygyModule*** call AssertInvBasis with the output of the C++ routine as parameter which sets up the internal data for the current ***Involutive*** session. Hence, from the user's point of view, there is no difference in using the "Maple" or the "C++" methods of ***InvolutiveBasis***, ***PolInvReduce***, ***SyzygyModule***, and ***GroebnerBasis*** when selecting the methods by means of ***InvolutiveOptions***. The same remarks hold for the keyword "GINV". In that case InvolutiveBasis becomes a synonym for InvolutiveBasisGINV, PolInvReduce a synonym for PolInvReduceGINV, SyzygyModule a synonym for SyzygyModuleGINV, and GroebnerBasis a synonym for GroebnerBasisGINV. Note also that these options affect many commands of the Involutive package which call these basic procedures (e. g. PolResolution) and that the "C++" and "GINV" methods are much faster than the "Maple" routines for big problems. There is no return value of ***InvolutiveOptions*** if **s** is either "C++", "GINV", 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 Involutive package (e.g. PolResolution, PolRepres, Repres are affected by this option; however, the result of PolLeftInverse, PolRightInverse and PolCoeff 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 InvolutiveBasis returns a Janet-like Groebner basis instead of an involutive basis. The default setting is false. The return value of ***InvolutiveOptions*** 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 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++", "GINV", and "Maple". More precisely, if **s** is the string "InvBasis" then **v** is expected to be a procedure. Then every subsequent call of InvolutiveBasis 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 PolInvReduce (resp. SyzygyModule resp. GroebnerBasis) invokes **v** instead of the default method. In each case the return value of ***InvolutiveOptions*** is the procedure used so far by the Involutive package for the respective purpose.

Examples:

```
> with(Involutive):
```

Example 1: Changing the characteristic of the ground field

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

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

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

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

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

$$0$$

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

$$[z+y, x]$$

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

$$2$$

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

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

```
> InvolutiveOptions("char");
```

$$0$$

Example 2: Computing involutive bases over the integers

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

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

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

$$true$$

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

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

```
> PolTabVar();
```

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

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

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

$$false$$

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

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

```
> PolTabVar();
```

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

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

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

$$false$$

```
> InvolutiveBasis(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)];
```

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

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

$$[1]$$

```
> InvolutiveOptions("GINV");
> InvolutiveBasis(L, [seq(a[i], i=1..7)]);
```

$$[1]$$

```
> InvolutiveOptions("Maple");
> InvolutiveBasis(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];
```

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

```
> PolResolution(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];
```

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

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

$$false$$

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

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

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

$$true$$

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

$$[-z\,w^2\,x+y^4, x^3\,y-z\,w, x^4\,w-y^3, -z\,w^2\,x^2+y^4\,x, y\,x^4-z\,w\,x, w\,x^5-y^3\,x, -z\,w^2\,x^3+y^4\,x^2, y\,x^5-z\,w\,x^2, w\,x^6-y^3\,x^2, y\,x^6-z\,w\,x^3, x^7-y^2\,z]$$

```
> GroebnerBasis(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:

InvolutiveBasis, Stats, InvolutiveBasisFast, InvolutiveBasisGINV, PolInvReduce, PolInvReduceFast, PolInvReduceGINV, PolTabVar, PolHilbertSeries, SyzygyModule, SyzygyModuleFast, SyzygyModuleGINV, GroebnerBasis, GroebnerBasisFast, GroebnerBasisGINV.

Involutive[InvolutivePreprocess] - find possibilities to solve relations of a finitely presented module over a polynomial ring for some variables

Calling Sequence:

InvolutivePreprocess(L,var,mode)

Parameters:

L	-	matrix of polynomials in **var** or list of (lists of the same length of) polynomials in **var**
var	-	list of variables (of the polynomial ring)
mode	-	(optional) string specifying options for the computation

Description:

- ***InvolutivePreprocess*** searches through the list **L** in order to find (tuples of) polynomials in **L** which can be solved (linearly) for some of the variables in **var**. If possible, it returns a list of equations that are solved for some variable in terms of the other ones. It is convenient to apply ***InvolutivePreprocess*** to **L** prior to the run of InvolutiveBasis and to possibly use the result of ***InvolutivePreprocess*** to eliminate some of the variables in **var** by substitution in order to reduce the complexity of the involutive basis computation. This is automatized by the command Substitute.
- For each generator *p* of the submodule in **L**, ***InvolutivePreprocess*** checks whether a variable in **var** occurs only linearly in (some component of) *p* with coefficient independent of the other variables. If this is the case, $p = 0$ is solved for this variable, the resulting equation becomes an entry of the list returned by ***InvolutivePreprocess***, and the next generator is examined.
- 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 parameter **var** is a list specifying the variables of the polynomial ring.
- If the letter "A" is present in **mode**, ***InvolutivePreprocess*** finds *all* possibilities to solve (tuples of) polynomials in **L** for variables in **var**, i.e. the search process described above is not stopped when finding a relation for some variable (in some component), but the search is extended to all variables (and all components).
- If the letter "S" is present in **mode**, the program applies evala and simplify to the result of solve. If the polynomials in the input **L** contain nonrational coefficients, more precisely, if the ground field contains algebraic elements over the rationals (RootOf), then evala and simplify are applied to the result of solve automatically by ***InvolutivePreprocess***.
- The result of ***InvolutivePreprocess*** is a (possibly empty) list of equations whose left hand sides are variables in **var** in the ideal case or tuples whose only non-zero entry is a variable in **var**. These equations satisfy that the difference of their left and right hand sides are in the submodule spanned by **L**.
- In addition, for each variable in **var** which does not occur in the generators of the submodule given in **L**, ***InvolutivePreprocess*** gives a warning.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

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

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

$$\left[x = \frac{1}{3}y^2 + \frac{1}{3}z\right]$$

```
> InvolutivePreprocess(L, var, "A");
```

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

```
> InvolutivePreprocess(L, [x,y,z,u]);
Warning, variable u does not occur in given polynomials.
```

$$\left[x=\frac{1}{3}y^2+\frac{1}{3}z\right]$$

Example 2:

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

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

```
> L := [[0,x^3,x-z,y^2-z^2-x],[0,0,x^2-y+z^7*x+1,1],[x^2,y^3,x+x*y,0]];
```

$$L := [[0, x^3, x-z, y^2-z^2-x], [0, 0, x^2-y+z^7 x+1, 1], [x^2, y^3, x+xy, 0]]$$

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

$$[[0, 0, x, 0]=[0, x^3, z, y^2-z^2-x], [0, 0, y, 0]=[0, 0, x^2+z^7 x+1, 1]]$$

```
> InvolutivePreprocess(L, var, "A");
```

$$[[0, 0, x, 0]=[0, x^3, z, y^2-z^2-x], [0, 0, z, 0]=[0, x^3, x, y^2-z^2-x], [0, 0, 0, x]=[0, x^3, x-z, y^2-z^2], [0, 0, y, 0]=[0, 0, x^2+z^7 x+1, 1]]$$

Example 3:

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

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

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

$$0$$

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

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

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

$$[x=y^2 z^2]$$

```
> InvolutiveOptions("char", 3);
```

$$2$$

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

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

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

$$[\ ]$$

See Also:

InvolutiveBasis, Substitute, InvolutiveOptions, PolTabVar, PolInvReduce, PolHilbertSeries, SyzygyModule, GroebnerBasis.

Involutive[ItJanet] - perform iterated Janet division on a polynomial

Calling Sequence:

ItJanet(p,J,var,c)

Parameters:

p	-	polynomial
J	-	Janet basis with right hand sides
var	-	list of variables (of the polynomial ring)
c	-	list of equations containing Janet normal forms modulo **J**

Description:

- ***ItJanet*** performs iterated Janet division on the polynomial **p**. This method can be used when algebraically independent polynomials p_1, ..., p_r in the variables **var** are given such that the factor module of the polynomial ring in the indeterminates **var** modulo the ideal generated by p_1, ..., p_r is a free module over the polynomial ring P generated by the p_1, ..., p_r. Another requirement for the applicability of ***ItJanet*** is that a Janet basis **J** of the ideal generated by p_1, ..., p_r is computed with right hand sides P_1, ..., P_r for p_1, ..., p_r resp., where the P_i are unassigned variables. In order to guarantee convergence of ***ItJanet***, the right hand sides of this Janet basis **J** must have smaller degree in the variables **var** than the corresponding left hand sides. Then by means of ***ItJanet*** one can express any given element **p** of P as polynomial in P_1, ..., P_r.
- Moreover, as fourth parameter **c** a list of equations is accepted that contains the Janet normal forms with right hand sides modulo **J** of a basis of the polynomial ring R in the variables **var** as module over P. If this basis consists of the polynomials q_1, ..., q_s, the list **c** can be constructed by applying PolInvReduce to q_i=Q_i for i=1, ..., s, where Q_1, ..., Q_s are unassigned variables. Then ***ItJanet*** expresses any element **p** of R as linear combination of the Q_i with coefficients that are polynomials in the P_j.
- ***ItJanet*** applies PolInvReduce to **p**=0 (modulo the Janet basis **J**) and reduces the result modulo the vector space generated by the left hand sides of **c** by Gaussian elimination and taking the corresponding linear combinations on the right hand side. In this way the unassigned variables P_i, Q_j are gathered on the right hand side. Let the result be r=q. Then ***ItJanet*** applies this step of the algorithm again to $r - q$. This process is iterated until the degree of $r - q$ in the variables **var** is zero.

Examples:

```
> with(Involutive):
```

Example 1:

```
> var := [x,y]:
> p1 := x^2+y^2; p2 := x+y^3;
```

$$p1 := x^2 + y^2$$

$$p2 := x + y^3$$

```
> J := InvolutiveBasis([p1=P1, p2=P2], var):
> PolTabVar();
```

$$[x^2 + y^2 = P1, [x, y], x^2]$$

$$[x + y^3 = P2, [*, y], y^3]$$

$$[y^3 x - y^2 = P2x - P1, [*, y], y^3 x]$$

```
> ItJanet(p1*p2^2, J, var);
```

$$P1\,P2^2$$

```
> ItJanet(p1^3+p2, J, var);
```

$$P2 + P1^3$$

Example 2: Invariant ring of C3 x C3

```
> with(Invariant):
> var := [w,x,y,z]:
```

```
> t1 := [w=w,x=x,y=-y-z,z=y]; t2 := [w=-w-x,x=w,y=y,z=z];
```

$$t1 := [w = w, x = x, y = -y - z, z = y]$$

$$t2 := [w = -w - x, x = w, y = y, z = z]$$

```
> r := a->Reynolds(a, [t1,t2], 9, 10):
> p1 := y^2+y*z+z^2; p2 := w^2+w*x+x^2;
  p3 := -y^2*z-y*z^2; p4 := -w^2*x-w*x^2;
```

$$p1 := y^2 + yz + z^2$$

$$p2 := w^2 + wx + x^2$$

$$p3 := -y^2 z - yz^2$$

$$p4 := -w^2 x - wx^2$$

```
> J := InvolutiveBasis([p1=P1, p2=P2, p3=P3, p4=P4], var):
> S := SecInvar(r, 9, J, var, Q);
```

$$S := [[1, 3yz^2 - y^3 + z^3, 3wx^2 - w^3 + x^3, 3yz^2x^3 - 3z^2yw^3 - 3y^3x^2w - y^3x^3 - w^3z^3 + x^3z^3 + 3z^3x^2w + y^3w^3 + 9z^2yx^2w], [$$

$$1 = Q1, 3yz^2 = -P1z + P1y + Q2, 3wx^2 = -P2x + P2w + Q3,$$

$$9z^2yx^2w = -P2z^3x + P2z^3w - 3P2xz^2y + 3P2z^2yw - 3P1zx^2w + 3P1x^2wy + P2xy^3 - P2y^3w + Q4]]$$

```
> q2 := S[1,2]; q3 := S[1,3]; q4 := S[1,4];
```

$$q2 := 3yz^2 - y^3 + z^3$$

$$q3 := 3wx^2 - w^3 + x^3$$

$$q4 := 3yz^2x^3 - 3z^2yw^3 - 3y^3x^2w - y^3x^3 - w^3z^3 + x^3z^3 + 3z^3x^2w + y^3w^3 + 9z^2yx^2w$$

```
> c := S[2]:
> ItJanet(p1*p2, J, var, c);
```

$$P1P2$$

```
> ItJanet(p3*q3+p1^2*q4, J, var, c);
```

$$P1^2 Q4 + P3 Q3$$

```
> ItJanet(q3^2, J, var, c);
```

$$-9P4^2 - 3P4Q3 + P2^3$$

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, Reynolds, PrimInvar, SecInvar.

Involutive[JanetGraph] - return the Janet graph which corresponds to the Janet basis

Calling Sequence:

JanetGraph(B,var)

Parameters:

B	-	(optional) Janet basis
var	-	list of variables (of the polynomial ring)

Description:

- Associated with every Janet basis is a Janet graph which is the labeled directed graph whose vertices are the elements of the Janet basis and whose edge set is given as follows: For each element v of the Janet basis and each of its non-multiplicative variables x there is an edge from v to the unique involutive divisor of $x\,v$ in the Janet basis. This edge is labeled by x. The Janet graph contains the same information as the corresponding Janet basis (cf. PolTabVar).
- The command ***JanetGraph*** returns the Janet graph which corresponds to the Janet basis **B**. If **B** is not given, ***JanetGraph*** works on the Janet basis which has been computed by the last call of InvolutiveBasis. The graph is returned as the list of its labeled edges, where each edge is represented as a triple $[v, x, w]$, where v is an element of the Janet basis, x is a non-multiplicative variable of v, and w is the involutive divisor of $x\,v$ in the Janet basis. The order of the edges in the result is the same as the order of the v's in the Janet basis.
- For more information about the Janet graph, 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(Involutive):
> var := [x3,x2,x1];
```

$$var := [x3, x2, x1]$$

```
> L := [x2^2*x3, x1^2*x3^3];
```

$$L := [x2^2\,x3, x1^2\,x3^3]$$

```
> J := InvolutiveBasis(L, var);
```

$$J := [x2^2\,x3, x3^2\,x2^2, x1^2\,x3^3, x3^3\,x2^2, x2\,x1^2\,x3^3]$$

```
> PolTabVar();
```

$$[x2^2\,x3, [*, x2, x1], x2^2\,x3]$$
$$[x3^2\,x2^2, [*, x2, x1], x3^2\,x2^2]$$
$$[x1^2\,x3^3, [x3, *, x1], x1^2\,x3^3]$$
$$[x3^3\,x2^2, [x3, x2, x1], x3^3\,x2^2]$$
$$[x2\,x1^2\,x3^3, [x3, *, x1], x2\,x1^2\,x3^3]$$

```
> JanetGraph(var);
```

$$[[x2^2\,x3, x3, x3^2\,x2^2], [x3^2\,x2^2, x3, x3^3\,x2^2], [x1^2\,x3^3, x2, x2\,x1^2\,x3^3], [x2\,x1^2\,x3^3, x2, x3^3\,x2^2]]$$

See Also:

InvolutiveBasis, FactorModuleBasis, PolTabVar, PolHilbertSeries, PolRepres, PolMinPoly.

Involutive[LeadingMonomial] - determine leading monomial(s) of a (list of) polynomial(s)

Calling Sequence:

LeadingMonomial(L,var,ord,mode)

Parameters:

`L`	-	list of elements of a free module over a polynomial ring
`var`	-	list of variables (of the polynomial ring)
`ord`	-	(optional) change of monomial ordering
`mode`	-	(optional) string specifying the type of information to be returned

Description:

- ***LeadingMonomial*** 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, ***LeadingMonomial*** 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`** and **`ord`** see the corresponding explanations in InvolutiveBasis.
- 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(Involutive):
> var := [x,y,z];
```

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

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

$$L := [3xyz + 4xz^3, 2y^2 + 7x]$$

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

$$[xz^3, y^2]$$

```
> LeadingMonomial(L[1], var);
```

$$xz^3$$

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

$$[xyz, x]$$

```
> LeadingMonomial(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]];
```

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

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

$$[xz^3, x^3]$$

Assign degrees to the variables:

```
> LeadingMonomial(L, [x=1,y=3,z=1]);
```

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

Use "position over term" ordering:

```
> LeadingMonomial(L, [x=1,y=3,z=1], 2);
```

$$[xyz, x^3]$$

Change the sequence of priority of the list entries:

```
> LeadingMonomial(L, [x=1,y=3,z=1,2,1]);
```

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

Return leading monomials in tuples:

```
> LeadingMonomial(L, [x=1,y=3,z=1,2,1], 4, "T");
```

$$[[0, y^2], [x^3, 0]]$$

Assign degrees to standard basis vectors:

```
> LeadingMonomial([[x*y, y]], [x,y,1=0,2=2], 4);
```

$$[y]$$

See Also:

InvolutiveBasis, PolTabVar, Has.

Involutive[NoetherNormalization] - find invertible transformation of the variables which puts given ideal in Noether position

Calling Sequence:

NoetherNormalization(L,var,mode,opt)

Parameters:

`L`	-	list of polynomials
`var`	-	list of variables of the polynomial ring
`mode`	-	(optional) sequence of strings "L" or "P"
`opt`	-	(optional) sequence of options to be handed over to involutive basis computation

Description:

- ***NoetherNormalization*** constructs a Noether normalization of the ideal generated by the elements of **`L`** in the polynomial ring with indeterminates **`var`**. More precisely, an automorphism of the polynomial ring is determined such that the transformed residue class ring is a finite extension of a polynomial ring generated by a certain number of new variables.
- ***NoetherNormalization*** returns (as part of its output) a list encoding such an automorphism. This list consists of equations which represent a substitution of the variables in **`var`**. In each equation, the left hand side is a variable in **`var`**, and the right hand side is understood as a polynomial in new variables, which is the image of the variable on the left hand side under the automorphism. However, to simplify further processing of the output, the new variables again get the names given in **`var`**. If *d* is the Krull dimension of the given residue class ring, then the new variables whose names are the last *d* entries in **`var`** form a polynomial ring over which the transformed residue class ring is finite.
- If the characteristic of the ground field is changed for the current Involutive session using InvolutiveOptions, then ***NoetherNormalization*** computes over the same chosen ground field.
- If the characteristic of the ground field is zero, then the images of the variables in **`var`** under the constructed automorphism are linear combinations of the new variables, i.e. the right hand sides defined above are linear polynomials. In particular, if the given ideal is homogeneous, then its image under the constructed automorphism is again homogeneous. In non-zero characteristic, a non-homogeneous transformation may be necessary.
- The result of ***NoetherNormalization*** is a list of two lists. The second list defines the automorphism described above in terms of equations, whose *i*-th right hand side is the image of the *i*-th variable in **`var`**. This coordinate transformation achieves a Noether normalization of the residue class ring modulo the ideal generated by **`L`**. The first list in the result is an involutive basis for the transformed ideal (the result of InvolutiveBasis) with respect to the degree-reverse lexicographical ordering.
- The implemented method first computes an involutive basis of the given ideal and then compares the number of variables occurring in the denominators of the generalized Hilbert series for the complement of the ideal of leading terms (see FactorModuleBasis) with the Krull dimension of the residue class ring. As long as the former number is greater than the latter number, a sparse invertible transformation of the variables is determined from inspection of the leading terms of the involutive basis such that the difference of the corresponding numbers for the involutive basis of the transformed ideal is smaller. This process is iterated, where the given ideal is replaced by the transformed ideal, until the difference is zero.
- The string "L" is accepted as an option (see Example 5 below). If it is given in **`mode`**, ***NoetherNormalization*** ensures that the last *d* variables are algebraically independent modulo the given ideal in the new coordinates, where *d* is the Krull dimension of the residue class ring.
- If the string "P" is given in **`mode`** (see Example 6 below), ***NoetherNormalization*** determines after the computation of the first involutive basis whether the given ideal is principal. In that case it checks whether the unique element in the involutive basis is a monic polynomial in some of the variables in **`var`** (starting from the first variable). If this is true, then ***NoetherNormalization*** immediately returns the list consisting of the involutive basis, a list encoding the identity map of the polynomial ring, and a variable in **`var`** as third entry such that the above generator of the principal ideal is monic as a polynomial in that variable, and it is not monic as a polynomial of lower degree in another variable in **`var`**. If this test fails, then ***NoetherNormalization*** goes on with the process outlined above.

- For more information about this method to construct a Noether normalization, see D. Robertz, "Noether normalization guided by monomial cone decompositions", Journal of Symbolic Computation, 44(10), 2009, pp. 1359-1373.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

$$L := [xyz]$$

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

$$[xyz]$$

```
> FactorModuleBasis(var);
```

$$\frac{1}{(1-y)(1-z)} + \frac{x}{(1-x)(1-z)} + \frac{xy}{(1-x)(1-y)}$$

```
> N := NoetherNormalization(L, var);
```

$$N := [[x^3 - yx^2 - zx^2 + xyz], [x = x, y = y - x, z = z - x]]$$

```
> AssertInvBasis(N[1], var):
> FactorModuleBasis(var);
```

$$\frac{1}{(1-y)(1-z)} + \frac{x}{(1-y)(1-z)} + \frac{x^2}{(1-y)(1-z)}$$

```
> InvolutiveBasis(subs(N[2], L), var);
```

$$[x^3 - yx^2 - zx^2 + xyz]$$

Example 2:

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

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

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

$$L := [xyz, xz^2, xy^2, w^2 x^2]$$

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

$$\frac{z^2}{(1-z)(1-y)(1-w)} + \frac{yz}{(1-y)(1-w)} + \frac{z}{1-w} + \frac{zx}{1-w} + \frac{zx^2}{1-x} + \frac{zx^2 w}{1-x} + \frac{y^2}{(1-y)(1-w)} + \frac{y}{1-w} + \frac{xy}{1-w} + \frac{yx^2}{1-x} + \frac{yx^2 w}{1-x} + \frac{1}{1-w} + \frac{x}{1-w} + \frac{x^2}{1-x} + \frac{x^2 w}{1-x}$$

```
> N := NoetherNormalization(L, var);
```

$$N := [[y^2 z - xy^2, yz^2 - xyz, z^3 - xz^2, z^2 w^2 - 2w^2 xz + w^2 x^2, zw^2 x^2 - w^2 x^3, yw^2 xz - yw^2 x^2], [z = z, y = y, x = x - z, w = w]]$$

```
> AssertInvBasis(N[1], var);
```

$$[y^2 z - xy^2, yz^2 - xyz, z^3 - xz^2, z^2 w^2 - 2w^2 xz + w^2 x^2, zw^2 x^2 - w^2 x^3, yw^2 xz - yw^2 x^2]$$

```
> FactorModuleBasis(var);
```

$$\frac{1}{(1-y)(1-x)(1-w)} + \frac{z^2}{1-x} + \frac{z^2 w}{1-x} + \frac{zy}{1-w} + \frac{xyz}{1-x} + \frac{zyxw}{1-x} + \frac{z}{1-w} + \frac{zx}{1-w} + \frac{zx^2}{1-x} + \frac{zx^2 w}{1-x}$$

Example 3: (involutive basis computations can be switched from Maple to C++)

```
> InvolutiveOptions("C++"):
> var := [w,x,y,z];
```

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

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

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

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

$$\frac{wx}{1-z}+\frac{x^2w}{1-x}+z^2w^2+\frac{w^2xz}{1-x}+\frac{w^2xy}{1-x}+w^3z^2+wyz^2+wy+wy^2z+\frac{xy}{1-y}+w^2yz+wxy+wyz+w^3yz+w^2z+wy^2+w^3z$$
$$+w^3y+w^2z^3+w^2y+\frac{x^3}{(1-x)(1-z)}+\frac{x^2y}{1-y}+\frac{w^2x}{1-x}+\frac{w^2y^2}{1-y}+\frac{wy^3}{1-y}+\frac{x^3y}{1-x}+\frac{w^4z}{1-w}+\frac{w^4y}{1-w}+\frac{w^3y^2}{1-y}+\frac{x^3y^2}{1-x}+\frac{1}{1-z}+\frac{y}{1-z}$$
$$+\frac{y^4}{1-y}+y^2+w^2+\frac{w^4}{1-w}+\frac{zx^2w}{1-x}+\frac{yx^2w}{1-x}+w^3+y^3+y^2z+\frac{wx^2z^2}{1-x}+\frac{w^4yz}{1-w}+\frac{w^4y^2}{(1-w)(1-y)}+y^3z+y^2z^2+\frac{x}{1-z}+\frac{w}{1-z}$$
$$+\frac{x^2}{1-z}$$

```
> N := NoetherNormalization(L, var):
> N[2];
```

$$[w=w, x=x-w, y=y-x, z=z-x]$$

```
> AssertInvBasis(N[1], var):
> FactorModuleBasis(var);
```

$$\frac{wx}{1-z}+\frac{x}{(1-y)(1-z)}+\frac{1}{(1-y)(1-z)}+xwy^2+w^2y^2z+w^2xy+w^2y^2+y^3x^2+x^3+x^2y^2+\frac{wy}{1-z}+\frac{wy^2}{1-z}+\frac{x^2y}{1-z}+\frac{w^2y}{1-z}+\frac{w^2x}{1-z}$$
$$+\frac{wy^3}{1-z}+\frac{w^3y}{1-z}+\frac{w^3x}{1-z}+\frac{y^4w}{1-z}+x^3z+\frac{wxy}{1-z}+\frac{wy^5z}{1-y}+z^2x^3+y^3x^2z+\frac{w^2}{1-z}+\frac{w^3}{1-z}+x^2y^2z+\frac{wy^5}{1-y}+x^3y+\frac{w}{1-z}+\frac{x^2}{1-z}$$

```
> InvolutiveOptions("Maple");
```

Example 4: (Noether normalization over a finite ground field sometimes requires a nonlinear transformation)

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

$$0$$

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

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

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

$$L := [y^2+xy]$$

```
> N := NoetherNormalization(L, var);
```

$$N := [[x^4+x^3+xy+y^2], [x=x, y=y-x^2]]$$

```
> AssertInvBasis(N[1], var):
> FactorModuleBasis(var);
```

$$\frac{1}{1-y}+\frac{x}{1-y}+\frac{x^2}{1-y}+\frac{x^3}{1-y}$$

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

$$2$$

Example 5: (ensure that last *d* variables are algebraically independent modulo the ideal, where *d* is the Krull dimension)

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

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

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

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

```
> N := NoetherNormalization(L, var);
```

$$N := [[z^3, x^3-xy^2, xz^3, z^3x^2], [x=x, y=y-x, z=z]]$$

```
> AssertInvBasis(N[1], var):
> FactorModuleBasis(var);
```

$$\frac{x^2}{1-y}+\frac{zx^2}{1-y}+\frac{x^2z^2}{1-y}+\frac{x}{1-y}+\frac{zx}{1-y}+\frac{xz^2}{1-y}+\frac{1}{1-y}+\frac{z}{1-y}+\frac{z^2}{1-y}$$

```
> N := NoetherNormalization(L, var, "L");
```

$$N := [[y^3, x^3-xz^2, y^3x, y^3x^2], [x=x, y=z-x, z=y]]$$

```
> AssertInvBasis(N[1], var):
> FactorModuleBasis(var);
```

$$\frac{x^2}{1-z}+\frac{x^2y}{1-z}+\frac{x^2y^2}{1-z}+\frac{x}{1-z}+\frac{xy}{1-z}+\frac{xy^2}{1-z}+\frac{1}{1-z}+\frac{y}{1-z}+\frac{y^2}{1-z}$$

Example 6: (check for principal ideal)

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

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

```
> L := [w*x*y*z + y^3 + w*x + z^2];
```

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

```
> N := NoetherNormalization(L, var);
```

$$N := [[w^4 + w^3 - w^3\,y - w^3\,z - w^3\,x + y\,w^2\,x + w^2\,x\,z - 3\,y\,w^2 + w^2\,y\,z - z\,y\,x\,w + 2\,z\,w + 3\,w\,y^2 - w\,x - y^3 - z^2],$$
$$[w = w, x = x - w, y = y - w, z = z - w]]$$

```
> AssertInvBasis(N[1], var):
> FactorModuleBasis(var);
```

$$\frac{1}{(1-x)(1-y)(1-z)} + \frac{w}{(1-x)(1-y)(1-z)} + \frac{w^2}{(1-x)(1-y)(1-z)} + \frac{w^3}{(1-x)(1-y)(1-z)}$$

```
> NoetherNormalization(L, var, "P");
```

$$[[z\,y\,x\,w + y^3 + w\,x + z^2], [w = w, x = x, y = y, z = z], z]$$

See Also:

InvolutiveBasis, PolTabVar, FactorModuleBasis, SubmoduleBasis, PolHilbertSeries, PolMinPoly, PolRepres, InvolutiveOptions.

Involutive[Has],

Involutive[NotHas] - takes a certain sublist of a list of elements of a free module over a polynomial ring

Calling Sequence:

Has(B,var,vi,ord)
NotHas(B,var,vi,ord)

Parameters:

`B` - list of module elements, typically an involutive basis
`var` - list of variables (of the polynomial ring)
`vi` - list of variables to be inspected
`ord` - (optional) change of monomial ordering

Description:

- ***NotHas*** respectively ***Has*** 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. InvolutiveBasis, computed with respect to pure lexicographical ordering, and **`vi`** are the first variables according to this ordering. In this case, ***NotHas*** 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 InvolutiveBasis.
- 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(Involutive):
```

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 := InvolutiveBasis(L, var);
```

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

PolTabVar displays the involutive basis, multiplicativity of variables and leading monomials:

```
> PolTabVar();
```

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

```
> NotHas(B, var, [x]);
```

$$[y^2z+z^2x+yz^2]$$

(Note, w. r. t. pure lex. ordering the summands containing x are the greatest, so we get:)

```
> NotHas(B, var, [x], 1);
```

$$[\,]$$

```
> Has(B, var, [x]);
```

$$[xy+yz+zx, x^2+zx+yz]$$

```
> Has(B, var, [x,y]);
```

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

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 := InvolutiveBasis(L, var);
```

$$B := [y^2-z^2, xy+z^2, z^2x+yz^2]$$

```
> PolTabVar();
```

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

```
> NotHas(B, var, [z]);
```

$$[y^2-z^2, xy+z^2]$$

```
> remove(has, B, [z]);
```

$$[\,]$$

```
> Has(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 ***NotHas*** 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 := InvolutiveBasis(L, var, 1);
```

$$L := \Big[ba^2-4b^2-c^2, ba^2y-4b^2y-c^2y, y^2ba^2-4y^2b^2-c^2y^2, y^3ba^2-4y^3b^2-c^2y^3, b+y^4-y^2a, cx+y^3a-ya^2+2by,$$
$$cax+y^3a^2-ya^3+2bay, ca^2x+y^3a^3-ya^4+8b^2y+2c^2y, cy^3a-ca^2y+bxa^2-4b^2x+2cyb, cyx+2by^2-ba,$$
$$cayx+2by^2a-4b^2-c^2, -4bxy+2cy^2-ca+ya^2x, cy^2x+2by^3-bay, bxy^2-\frac{1}{2}cy^3+\frac{1}{2}cay-\frac{1}{2}bxa, -2bx+y^2ax+cy,$$
$$xy^3+\frac{1}{2}c-\frac{1}{2}yax, x^2+y^2-a\Big]$$

```
> NotHas(L, var, [x,y], 1);
```

$$[ba^2-4b^2-c^2]$$

This of course is interpreted as a ring relation between x^2+y^2, x^2y^2, and x^3y-xy^3.

See Also:

InvolutiveBasis, PolTabVar, Syzygies.

Involutive[PolCartanCharacter] - compute Cartan characters of a finitely presented module over a polynomial ring

Calling Sequence:

PolCartanCharacter(i)
PolCartanCharacter()

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 PolHilbertSeries. 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 InvolutiveBasis. (The same formula holds with q replaced by the index of regularity plus 1, cf. PolIndexRegularity, with suitably modified Cartan characters α's.)
- ***PolCartanCharacter***() prints the the highest degree q of the polynomials in the Janet basis computed by the last call InvolutiveBasis and returns the list of Cartan characters $[\alpha(q, 1), ..., \alpha(q, n)]$ of the factor module of the free module over the polynomial ring modulo the submodule generated by this Janet basis. ***PolCartanCharacter***(`i`) returns the Cartan character $\alpha(q, i)$.
- All this information can also be extracted from the command PolHilbertSeries.
- ***PolCartanCharacter***(" ") simply prints the Cartan characters $\alpha(q, 1), ..., \alpha(q, n)$, where q is as above.

Examples:

```
> with(Involutive):
> 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 := InvolutiveBasis(L, var);
```

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

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

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

```
> PolCartanCharacter(2);
```

$$6$$

```
> PolTabVar();
```

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

```
> PolHilbertSeries(s);
```

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

```
> PolIndexRegularity();
```

$$1$$

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 := InvolutiveBasis(L, [v,x,y,z], 1);
```

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

```
> PolHilbertSeries(s);
```

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

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

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

See Also:

InvolutiveBasis, PolTabVar, PolHilbertSeries, PolHilbertPolynomial, PolHP, PolHilbertFunction, PolHF, PolIndexRegularity.

Involutive[PolCheckHom] - check whether a matrix represents a homomorphism between two finitely presented modules over a polynomial ring

Calling Sequence:

PolCheckHom(M,A,N,var)

Parameters:

`M` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`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 polynomial ring

Description:

- By means of ***PolCheckHom*** one can determine whether **A** represents a (well defined) homomorphism from the module presented by **M** to the module presented by **N** (i.e., the elements of **M** and **N** are considered as elements of a free module of tuples over the polynomial ring with indeterminates **var** of appropriate rank, and the modules presented by **M** resp. **N** are the factor modules of the respective free modules modulo the submodules generated by the elements of **M** resp. **N**).
- ***PolCheckHom*** computes an involutive basis of the module generated by **N** first. Then it applies involutive reduction modulo this involutive basis to the images of the elements of **M** under **A** (considered as a homomorphism between free modules mapping row vectors). ***PolCheckHom*** returns the list of normal forms computed by PolInvReduce of these images. Note that the result is always a list of lists even in case **A** maps into a free module of rank 1.
- Given the result of ***PolCheckHom***, one therefore can check whether the submodule of relations generated by **M** maps to zero under the map between free modules represented by **A**. Then, **A** represents a homomorphism from the module presented by **M** to the module presented by **N** if and only if the result of ***PolCheckHom*** consists of lists of zeros only.
- If **M** and **N** are lists, then the entries of **M** and **N** are polynomials in case of ideals, i.e. submodules of the free module of rank one, or lists of polynomials of length *m* (resp. *n*), representing elements of the free module of *m*-tuples (resp. *n*-tuples) over the polynomial ring. If **M** or **N** is a matrix, then the generators for the submodules are extracted from the rows of **M** resp. **N**.
- The parameter **A** is either a matrix whose number of rows equals the rank of the free module which contains the elements of **M** and whose number of columns equals the rank of the free module which contains the elements of **N**, or **A** is a list of lists of polynomials in var of the same length. In the latter case, the corresponding matrix is formed by taking the entries of **A** as rows. In any case, row convection is applied, i.e., a polynomial matrix represents the homomorphism defined by multiplication of rows on the left of this matrix.

Examples:

```
> with(Involutive):
```

Example 1:

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

$$var := [x]$$

```
> M := [x]; A := [1]; N := [x+1];
```

$$M := [x]$$

$$A := [1]$$

$$N := [x+1]$$

```
> PolCheckHom(M, A, N, var);
```

$$[[-1]]$$

Since the result is different from the zero tuple, *A* does not represent a homomorphism from the module presented by *M* to the module presented by *N*. In fact, there is no non-zero homomorphism between these modules:

```
> PolHom(M, N, var);
```

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

Example 2:

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

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

```
> M := [[x-y,0],[0,x^3]]; N := [[x^2-y^2,0],[0,x^4]];
```

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

$$N := [[x^2 - y^2, 0], [0, x^4]]$$

```
> A := linalg[diag](2*x+2*y, x^3);
```

$$A := \begin{bmatrix} 2x + 2y & 0 \\ 0 & x^3 \end{bmatrix}$$

```
> PolCheckHom(M, A, N, var);
```

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

The matrix A represents a homomorphism from the module presented by M to the module presented by N. But the following matrix B does not:

```
> B := linalg[diag](x+2*y, x^3);
```

$$B := \begin{bmatrix} x + 2y & 0 \\ 0 & x^3 \end{bmatrix}$$

```
> PolCheckHom(M, B, N, var);
```

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

See Also:

InvolutiveBasis, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolKernel, PolCokernel, PolHom, PolHomHom, PolExt1, PolExtn, PolParametrization, PolTorsion, PolSyzOp.

Involutive[PolCoeff] - express module element as linear combination of given module elements

Calling Sequence:

PolCoeff(L,G,var)

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 polynomial ring

Description:

- ***PolCoeff*** expresses (if possible) the polynomials in **L** or the elements of a free module of tuples of polynomials in the polynomial ring with indeterminates **var** given in **L** as linear combinations of the polynomials resp. the tuples of polynomials of the same length in **G**. The result of ***PolCoeff*** 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, ***PolCoeff*** computes an involutive basis of **G** with right hand sides first (see InvolutiveBasis, AddRhs) and then performs involutive reduction on all elements in **L** modulo this involutive basis keeping all coefficients of these reductions (see PolInvReduce). 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 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 ***PolCoeff*** is an (*m* x *n*)-matrix (resp. a (1 x *n*)-matrix) with polynomial entries.

Examples:

```
> with(Involutive):
```

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 := PolCoeff(L, G, var);
```

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

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

```
> map(expand, evalm(C &* G));
```

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

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 := PolCoeff(L, G, var);
```

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

```
> map(expand, evalm(C &* G));
```

$$[x^6 + y^6, y^4 x^4]$$

PolCoeff 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 := PolCoeff(M, G, var);
```

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

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

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

```
> map(expand, evalm(C &* B));
```

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

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

```
> PolCoeff(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 := PolCoeff(L, G, var);
```

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

```
> evalm(C &* G);
```

$$[-1 \quad xyz]$$

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

$$[-1, xyz]$$

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, AddRhs, Syzygies, SyzygyModule, PolLeftInverse, PolRightInverse.

Involutive[PolCokernel] - return presentation of the cokernel of a homomorphism between two finitely presented modules over a polynomial ring

Involutive[PolSum] - return Janet basis of the sum of two submodules of a free module over a polynomial ring

Calling Sequence:

PolCokernel(A,N,var)
PolSum(A,N,var)

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 polynomial ring

Description:

- ***PolCokernel*** returns a presentation of the cokernel of the homomorphism represented by **A** between two finitely presented modules over the polynomial ring in **`var`**. The domain of this homomorphism is a factor module of the free 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 ***PolCokernel***, a presentation of its range is given by **N**. More precisely, the homomorphism represented by **A** maps into the factor module of the free 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 module of rank one, or lists of polynomials of length *m*, representing elements of the free module of *m*-tuples over the polynomial ring.
- The result is a Janet basis with respect to the ("term over position") degree reverse lexicographical ordering (cf. InvolutiveBasis). The cokernel of the given homomorphism is the factor module of the free module of *m*-tuples modulo the submodule generated by the entries of the result of ***PolCokernel***.
- ***PolSum*** is a synonym for ***PolCokernel***. The interpretation of input and output is different in this case: The input **A**, **N** forms two generating sets for submodules of a free module of tuples over the polynomial ring in **`var`**. The result of ***PolSum*** is a Janet basis for the sum of the two submodules.

Examples:

```
> with(Involutive):
```

Example 1: Presenting the cokernel of a homomorphism

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

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

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

$$A := \begin{bmatrix} x^2 & y^2 & 0 \\ 0 & x^2 & y^2 \end{bmatrix}$$

```
> PolCokernel(A, [[0,0,0]], var);
```

$$[[0, x^2, y^2], [x^2, y^2, 0]]$$

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

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

```
> PolCokernel(A, N, var);
```

$$[[0, 0, x], [0, x, 0], [x, 0, 0], [0, 0, y^2], [0, y^2, 0]]$$

Example 2: Computing the Janet basis of the sum of two submodules of a free module over a polynomial ring

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

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

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

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

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

$$M2 := [[x + 1, y], [0, x]]$$

```
> PolSum(M1, M2, var);
```

$$[[1, 0], [0, y], [0, x]]$$

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolDirectSum, PolSubFactor, PolKernel, PolSyzOp.

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

Calling Sequence:

PolDefect(L1,L2,var,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 polynomial ring
`v` - (optional) name of the indeterminate for the Hilbert series of the homology module (default: 's')

Description:

- ***PolDefect*** 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. ***PolDefect*** 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**.
- Since the result of ***PolDefect*** is a presentation of a subfactor module which is computed using PolSubFactor, the output of ***PolDefect*** is a list which is formatted in the same way as the output of PolSubFactor. For a description of the format of such a presentation, cf. PolSubFactor.
- The optional fourth argument to ***PolDefect*** 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(Involutive):
```

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:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolKernel, PolHom, PolHomHom, PolExt1, PolExtn, PolTorsion, PolParametrization, PolSyzOp.

Involutive[PolDimension] - return the dimension of the factor module presented by the last computed Janet basis

Calling Sequence:

PolDimension()

Parameters:

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

Description:

- ***PolDimension*** returns the degree of the filtered Hilbert polynomial (as in PolHP) of the filtration of the factor module for which a presentation was computed by the last call of InvolutiveBasis, as explained in PolHilbertSeries.
- Note, ***PolDimension***()-1 equals the degree of PolHilbertPolynomial().

Examples:

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

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

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

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

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

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

```
> PolTabVar();
```

$$[xy+yz+zx, [1, 2, 3], xy]$$
$$[1+yz^2+z^2x, [1, *, 3], z^2x]$$
$$[y^2z^2+y+z, [*, 2, 3], y^2z^2]$$

```
> PolDimension();
```

$$1$$

```
> PolHP();
```

$$6s-3$$

```
> PolHilbertPolynomial();
```

$$6$$

```
> PolHilbertSeries();
```

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

See Also:

InvolutiveBasis, PolTabVar, PolHilbertSeries, PolHilbertPolynomial, PolHP, PolHilbertFunction, PolHF, PolIndexRegularity, PolCartanCharacter.

Involutive[PolDirectSum] - form the matrix whose rows define the direct sum of given submodules of free modules over a polynomial ring

Calling Sequence:

PolDirectSum(L1, L2, ...)

Parameters:

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

Description:

- ***PolDirectSum*** returns a matrix whose rows form a generating set of the direct sum of given submodules of free modules over a polynomial ring.
- The entries of **`L1`**, **`L2`**, ... are polynomials in case of ideals, i. e. submodules of the free module of rank one, or lists of polynomials of the same length, representing elements of a free module of tuples over the polynomial ring. 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(Involutive):
> 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]];
```

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

```
> PolDirectSum(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:

InvolutiveBasis, PolTabVar, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolCokernel, PolSyzOp.

Involutive[PolEulerChar] - return Euler characteristic of a factor module of a free module over a polynomial ring

Calling Sequence:

PolEulerChar(L,var)

Parameters:

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

Description:

- ***PolEulerChar*** 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 ***PolEulerChar*** is the non-negative integer obtained as alternating sum of the entries of the list of ranks returned by PolResolutionDim (starting with positive sign for the last entry of this list of ranks and running backwards through this list). Since ***PolEulerChar*** relies on the result of PolResolutionDim, only one involutive basis computation is needed to obtain the Euler characteristic of *M* (instead of several involutive basis computations performed by PolResolution).
- 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(Involutive):
```

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:

InvolutiveBasis, PolTabVar, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolShorterResolution, PolShortestResolution, PolResolutionDim.

Involutive[PolExt1] - return presentation of the first extension module of a finitely presented module over a polynomial ring

Calling Sequence:

PolExt1(M,var,v)

Parameters:

M	-	list (of lists of the same length) of polynomials or matrix with polynomial entries
var	-	list of variables of the polynomial ring
v	-	(optional) name of the indeterminate for the Hilbert series of the subfactor (default: 's')

Description:

- ***PolExt1*** returns a presentation of the first extension module with values in the polynomial ring with indeterminates **var** of the module presented by **M** (i.e., the elements of **M** are considered as elements of a free module of tuples over the polynomial ring with indeterminates **var** of appropriate rank, and ***PolExt1*** computes the extension module of the factor modules of the respective free module modulo the submodule generated by the elements of **M**).
- If **M** is a list, then the entries of **M** are polynomials in the 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 **M** is a matrix, then the generators for the submodule are extracted from the rows of **M**.
- Since the result of ***PolExt1*** is a presentation for a defect of exactness of a chain complex (i.e. a homology module), ***PolExt1*** computes this presentation using PolSubFactor. The output of ***PolExt1*** is therefore a list which is formatted in the same way as the output of PolSubFactor. For a description of the format of such a presentation, cf. PolSubFactor.
- The optional third argument to ***PolExt1*** selects the name of the indeterminate for the Hilbert series. The default name is 's' which cannot be affected by a subs command.

Examples:

```
> with(Involutive):
```

Example 1:

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

$$var := [x]$$

```
> L := [[x,1,1], [1,x,1]];
```

$$L := [[x, 1, 1], [1, x, 1]]$$

```
> PolExt1(L, var);
```

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

```
> L := [[x,1], [1,x], [1,1]];
```

$$L := [[x, 1], [1, x], [1, 1]]$$

```
> PolExt1(L, var);
```

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

Example 2:

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

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

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

$$L := [[y, xy, 0], [y, 0, y^2]]$$

```
> PolExt1(L, var);
```

$$\left[[[1, 0] = [0, 1], [0, 1] = [1, 0]], [[y, y], [0, y^2], [0, xy]], 2 + 3s + 2\frac{s^2}{1-s}, [2, 0]\right]$$

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

$$L := [[y, y], [xy, 0], [0, y^2]]$$

```
> PolExt1(L, var);
```

$$\left[[[1, 0] = [1, 0, y], [0, 1] = [1, x, 0]], [[0, y], [y, 0]], 2 + 2\frac{s}{1-s}, [2, 0]\right]$$

See Also:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolKernel, PolCokernel, PolHom, PolHomHom, PolExtn, PolParametrization, PolTorsion, PolSyzOp.

Involutive[PolExtn] - return presentation of an extension module of a finitely presented module over a polynomial ring

Calling Sequence:

PolExtn(q,M,var,v)

Parameters:

`q` - non-negative integer
`M` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the polynomial ring
`v` - (optional) name of the indeterminate for the Hilbert series of the subfactor (default: 's')

Description:

- ***PolExtn*** returns a presentation of the **q**-th extension module with values in the polynomial ring with indeterminates **var** of the module presented by **M** (i.e., the elements of **M** are considered as elements of a free module of tuples over the polynomial ring with indeterminates **var** of appropriate rank, and ***PolExtn*** computes the **q**-th extension module of the factor modules of the respective free module modulo the submodule generated by the elements of **M**).
- If **M** is a list, then the entries of **M** are polynomials in the 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 **M** is a matrix, then the generators for the submodule are extracted from the rows of **M**.
- Since the result of ***PolExtn*** is a presentation for a defect of exactness of a chain complex (i.e. a homology module), ***PolExtn*** computes this presentation using PolSubFactor. The output of ***PolExtn*** is therefore a list which is formatted in the same way as the output of PolSubFactor. For a description of the format of such a presentation, cf. PolSubFactor.
- The optional fourth argument to ***PolExtn*** selects the name of the indeterminate for the Hilbert series. The default name is 's' which cannot be affected by a subs command.

Examples:

```
> with(Involutive):
```

Example 1:

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

$$var := [x]$$

```
> L := [[x,1,1], [1,x,1]];
```

$$L := [[x, 1, 1], [1, x, 1]]$$

```
> PolExtn(0, L, var);
```

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

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

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

```
> PolExtn(2, L, var);
```

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

Example 2:

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

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

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

$$L := [[y, xy, 0], [y, 0, y^2]]$$

```
> PolExtn(0, L, var);
```

$$\left[\left[\left[1\right]=\begin{bmatrix} -xy \\ y \\ x \end{bmatrix}\right], [0], \frac{1}{(1-s)^2}, [0, 1]\right]$$

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

$$\left[[[1, 0]=[0, 1], [0, 1]=[1, 0]], [[y, y], [0, y^2], [0, xy]], 2+3s+2\frac{s^2}{1-s}, [2, 0]\right]$$

```
> PolExtn(2, L, var);
```

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

See Also:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolKernel, PolCokernel, PolHom, PolHomHom, PolExt1, PolParametrization, PolTorsion, PolSyzOp.

Involutive[PolHF] - compute the filtered Hilbert function for the factor module

Calling Sequence:

PolHF(p)
PolHF()

Parameters:

p - " " (empty string) or natural number

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in PolHilbertSeries. Then ***PolHF*(p)** returns $\sum_{i=0}^{p} d_i$ for natural numbers **p** and prints the corresponding function in case **p** is the empty string.
- PolHilbertFunction, 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 ***PolHF()***.
- ***PolHF***() returns a function expecting one parameter **p** which computes ***PolHF*(p)**.

Examples:

```
> with(Involutive):
> 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 := InvolutiveBasis(L, var);
```

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

```
> PolHilbertSeries();
```

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

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

$$f := PolHF$$

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

14

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

1202

```
> PolHF(20);
```

1202

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

$$3s^2+2$$

```
> PolHilbertFunction("");
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:

InvolutiveBasis, PolTabVar, PolHilbertSeries, PolHilbertFunction, PolHilbertPolynomial, PolHP, SubmoduleHilbertSeries, SubmoduleHilbertPolynomial, SubmoduleHilbertFunction, SubmoduleHP, SubmoduleHF.

Involutive[PolHP] - compute the filtered Hilbert polynomial for the factor module

Calling Sequence:

PolHP(p)
PolHP()

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 PolHilbertSeries. Then ***PolHP*(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 PolIndexRegularity. Note, all this information can also be extracted from the command PolHF.
- ***PolHP()*** returns the above polynomial with ´s´ as the default name of the indeterminate. ´s´ cannot be affected by a subs command.
- PolHilbertPolynomial, 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 ***PolHP()***.

Examples:

```
> with(Involutive):
> 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 := InvolutiveBasis(L, var);
```

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

```
> PolHilbertSeries();
```

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

```
> PolHP();
```

$$3s^2+2$$

```
> PolHP(20);
```

$$1202$$

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

$$3\lambda^2+2$$

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

$$29$$

See Also:

InvolutiveBasis, PolTabVar, PolHilbertSeries, PolHilbertPolynomial, PolHilbertFunction, PolHF, SubmoduleHilbertSeries, SubmoduleHilbertPolynomial, SubmoduleHilbertFunction, SubmoduleHP, SubmoduleHF.

Involutive[PolHilbertFunction] - compute the graded Hilbert function

Calling Sequence:

PolHilbertFunction(p)
PolHilbertFunction()

Parameters:

p - " " (empty string) or natural number

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in PolHilbertSeries. Then ***PolHilbertFunction*(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.
- PolHF, 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 ***PolHilbertFunction***.
- ***PolHilbertFunction***() returns a function expecting one parameter **p** which computes ***PolHilbertFunction*(p)**.

Examples:

```
> with(Involutive):
> 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 := InvolutiveBasis(L, var);
```

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

```
> PolHilbertSeries();
```

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

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

$$f := Involutive/PolHilbertFunction$$

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

$$9$$

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

$$117$$

```
> PolHilbertFunction(20);
```

$$117$$

```
> PolHilbertFunction("");
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:

InvolutiveBasis, PolTabVar, PolHilbertSeries, PolHF, PolHilbertPolynomial, PolHP, SubmoduleHilbertSeries, SubmoduleHilbertPolynomial, SubmoduleHilbertFunction, SubmoduleHP, SubmoduleHF.

Involutive[PolHilbertPolynomial] - compute the graded Hilbert polynomial for the factor module

Calling Sequence:

PolHilbertPolynomial(p)
PolHilbertPolynomial()

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 PolHilbertSeries. Then ***PolHilbertPolynomial*(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 InvolutiveBasis. Note, this same information can be extracted from the command PolHilbertFunction.
- PolHP, 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 ***PolHilbertPolynomial***.
- ***PolHilbertPolynomial***() returns the graded Hilbert polynomial of the associated graded module of the residue class module modulo the module whose involutive basis has been computed last by InvolutiveBasis. Note, this same information can be extracted from the command PolHilbertFunction.
- 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(Involutive):
> 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 := InvolutiveBasis(L, var);
```

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

```
> PolHilbertSeries();
```

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

```
> PolHilbertPolynomial();
```

$$-3+6s$$

```
> PolHilbertPolynomial(6);
```

$$33$$

```
> PolHP(6);
```

$$110$$

```
> PolHilbertFunction("");
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
> PolHilbertPolynomial(lambda);
```

$$-3+6\lambda$$

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

$$15$$

See Also:

InvolutiveBasis, PolTabVar, PolHilbertSeries, PolHP, PolHilbertFunction, PolHF, SubmoduleHilbertSeries,

SubmoduleHilbertPolynomial, SubmoduleHilbertFunction, SubmoduleHP, SubmoduleHF.

Involutive[PolHilbertSeries] - Hilbert series of the factor module presented by the last computed Janet basis

Calling Sequence:

PolHilbertSeries(v)

Parameters:

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

Description:

- ***PolHilbertSeries*** returns the generating function counting - according to the standard degrees - the standard monomial basis vectors of the factor module *F* of the free module of *m*-tuples over the polynomial ring modulo the submodule *M* generated by the Janet basis produced by the last call of InvolutiveBasis.
- The free module of *m*-tuples over the polynomial ring is graded by the standard grading (maximal degree of the components) and therefore induces a grading on its residue class module *G* modulo the submodule of the leading terms of *M* (but unfortunately in general not on *F*). Note, this submodule, and therefore also its Hilbert series, depends on the term order chosen in the call of InvolutiveBasis. ***PolHilbertSeries*** returns the Hilbert series of the graded module *G*. This Hilbert series agrees with the generating function described above, since the standard bases for both *F* and *G* are represented by the same monomial elements in the module of *m*-tuples.
- In the standard case of degree reverse lexicographical monomial order (default value or parameter 2 in the call of ***InvolutiveBasis***) a change of the ordering of the variables does not change the resulting Hilbert series though it might change the graded factor module *G*. In the special case, where the Janet basis consists of homogeneous elements, i. e. *M* is a graded submodule and therefore *F* also inherits a grading, the Hilbert series computed by the present command also is the Hilbert series of *F*.
- In the non-standard case of pure lexicographical monomial order (parameter 1 or 3 in the call of ***InvolutiveBasis***) the result highly depends on the order of the variables and is usually different from the Hilbert series taken in the standard case.
- 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 *G* 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 ***PolHilbertSeries*** will proceed from the leading terms computed by InvolutiveBasis 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 PolWeightedHilbertSeries.

Examples:

```
> with(Involutive):
```

In the ideal case, the Hilbert series is 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 PolTabVar.

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

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

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

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

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

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

```
> PolTabVar();
```

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

```
> PolHilbertSeries();
```

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

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

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

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

```
> PolHilbertSeries();
```

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

```
> PolTabVar();
```

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

Here is a module example:

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

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

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

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

```
> PolHilbertSeries(lambda);
```

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

```
> PolTabVar();
```

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

See Also:

InvolutiveBasis, PolTabVar, FactorModuleBasis, JanetGraph, PolHilbertPolynomial, PolHP, PolHilbertFunction, PolHF, PolIndexRegularity, PolCartanCharacter, PolWeightedHilbertSeries, SubmoduleBasis, SubmoduleHilbertSeries.

Involutive[PolHom] - return presentation of the module of homomorphisms between two finitely presented modules over a polynomial ring

Calling Sequence:

PolHom(M,N,var)

Parameters:

`M` - 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 polynomial ring

Description:

- ***PolHom*** returns a presentation of the module of homomorphisms from the module presented by **`M`** to the module presented by **`N`** (i.e., the elements of **`M`** and **`N`** are considered as elements of a free module of tuples over the polynomial ring with indeterminates **`var`** of appropriate rank, and ***PolHom*** computes the homomorphisms between the factor modules of the respective free modules modulo the submodules generated by the elements of **`M`** resp. **`N`**).
- If **`M`** and **`N`** are lists, then the entries of **`M`** and **`N`** are polynomials in case of ideals, i.e. submodules of the free module of rank one, or lists of polynomials of length *m* (resp. *n*), representing elements of the free module of *m*-tuples (resp. *n*-tuples) over the polynomial ring. If **`M`** or **`N`** is a matrix, then the generators for the submodules are extracted from the rows of **`M`** resp. **`N`**.
- The result of ***PolHom*** is a list with four entries. The first one defines the abstract generators of the constructed presentation of the module of homomorphisms. The second entry is a list of the relations imposed on the abstract generators of the presentation. Finally, the third and the fourth entry of the result give the Hilbert series (see PolHilbertSeries) resp. the Cartan characters (see PolCartanCharacter) of the module of homomorphisms.
- We denote by *F1* resp. *F2* the free module of tuples over the polynomial ring with indeterminates **`var`**, whose rank equals the length of the lists in **`M`** resp. **`N`** (where the length is 1 in the case of ideals), and we denote by *F* the module of homomorphisms from *F1* to *F2* (represented here by matrices with polynomial entries). Then the first entry of the result of ***PolHom*** also gives an embedding of the presented module of homomorphisms into the factor module of *F* modulo the diagonal embedding of **`N`** into *F*. In other words, the first entry of the result establishes a correspondence of the abstract generators of the presentation to representatives of residue classes in the factor module quoted before.
- 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 is a matrix representing a residue class in the factor module described in the previous point which corresponds to the *i*th abstract generator. Hence, the right hand sides of the first entry provide a generating set for the module of homomorphisms.
- 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 annihilator 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 *R* of the free module *S* over the polynomial ring with indeterminates **`var`**, where the rank of *S* equals the number of abstract generators, such that the module of homomorphisms is isomorphic to the factor module *S* / *R*.
- The third entry of the result is the Hilbert series (according to standard degrees) of the module of homomorphisms, see PolHilbertSeries.
- The fourth entry of the result is the list of Cartan characters of the module of homomorphisms as defined in PolCartanCharacter.

Examples:

```
> with(Involutive):
```

Example 1:

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

$$var := [x]$$

```
> M := [x]; N := [x+1];
```

$$M := [x]$$

$$N := [x+1]$$

```
> PolHom(M, N, var);
```

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

There is no non-zero homomorphism from the module presented by M over the polynomial ring with indeterminate x to the module presented by N (because the former is a torsion module).

Example 2:

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

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

```
> M := matrix([[x,y]]);
```

$$M := [x \quad y]$$

```
> N := matrix([[0,x^2,y^2], [0,x*y^2,x^2*y]]);
```

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

```
> H := PolHom(M, N, var);
```

$$H := \left[\left[[1,0,0,0,0] = \begin{bmatrix} 0 & 0 & -y \\ 0 & 0 & x \end{bmatrix}, [0,1,0,0,0] = \begin{bmatrix} 0 & -y & 0 \\ 0 & x & 0 \end{bmatrix}, [0,0,1,0,0] = \begin{bmatrix} -y & 0 & 0 \\ x & 0 & 0 \end{bmatrix}, [0,0,0,1,0] = \begin{bmatrix} 0 & x & 0 \\ 0 & 0 & y \end{bmatrix}, [0,0,0,0,1] = \begin{bmatrix} 0 & y^2 & xy \\ 0 & 0 & 0 \end{bmatrix}\right], [[0,0,0,0,x], [-y,0,0,x,0], [0,x,0,y,0], [xy,y^2,0,0,y]], 5+7s+s^2\left(7\frac{1}{1-s}+\frac{1}{(1-s)^2}\right), [7,1]\right]$$

The presentation of the module of homomorphisms between the modules presented by M and N involves 5 abstract generators which correspond to

```
> H[1];
```

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

The relations of the abstract generators in the presentation are given by:

```
> H[2];
```

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

The Hilbert series (according to standard degrees) of the module of homomorphisms is:

```
> H[3];
```

$$5+7s+s^2\left(7\frac{1}{1-s}+\frac{1}{(1-s)^2}\right)$$

The Cartan characters of the module of homomorphisms are:

```
> H[4];
```

$$[7, 1]$$

See Also:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolKernel, PolCokernel, PolHomHom, PolExt1, PolExtn, PolParametrization, PolTorsion, PolSyzOp.

Involutive[PolHomHom] - represent the canonical homomorphism from a finitely presented module to its bidual as a matrix

Calling Sequence:

PolHomHom(M,var,H)

Parameters:

`M` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the polynomial ring
`H` - (optional) symbol to which the result of ***PolHom*** (applied to the relations in ***PolHom***(**M**,**var**)) is assigned

Description:

- ***PolHomHom*** returns a matrix which represents the canonical homomorphism from the module presented by **M** to its bidual, i.e. to the module of homomorphisms that map homomorphisms, from the module presented by **M** to the polynomial ring in **var**, to the polynomial ring in **var**.
- The entries (or rows) of **M** are considered as elements of a free module of tuples over the polynomial ring with indeterminates **var** of appropriate rank, and the module presented by **M** is the factor module of this free module modulo the submodule generated by the entries (or rows) of **M**.
- The result of ***PolHomHom*** is a matrix which represents the homomorphism which maps an element of **M** to the map which evaluates homomorphisms from the module presented by **M** to the polynomial ring at this element of **M**.
- Since row convention is applied, a homomorphism from the module presented by **M** to the polynomial ring is represented by a column (i.e. the homomorphism is given by multiplication of rows on the left of this column). Row convention is retained for the result of ***PolHomHom***: Multiplying a row *r*, which represents a residue class in **M**, to the left of the resulting matrix, one obtains a row whose entries are the values of the residue class represented by *r* under the homomorphisms represented by the generators in the presentation of the module of homomorphisms computed by PolHom applied to **M**. Hence, if homomorphisms from the module presented by **M** to the polynomial ring are now represented by columns w.r.t. the basis of generators given by ***PolHom***, then the multiplication of the row defined before by the column representing the homomorphism yields the value of the residue class represented by *r* under this homomorphism.
- If **M** is a list, then the entries of **M** 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 **M** is a matrix, then the generators for the submodule are extracted from the rows of **M**.
- As a first step, ***PolHomHom*** computes a presentation of the module of homomorphisms from the module presented by **M** to the polynomial ring in **var** using PolHom. Then ***PolHom*** is applied to the relations of this presentation so that a presentation of the range of the canonical homomorphism under consideration is obtained. If the optional argument **H** is present, then ***PolHomHom*** assigns the latter presentation to the symbol **H**.

Examples:

```
> with(Involutive):
```

Example 1:

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

$$var := [x]$$

```
> M := [[x,1,1], [1,x,1]];
```

$$M := [[x, 1, 1], [1, x, 1]]$$

```
> H := PolHom(M, [0], var);
```

$$H := \left[\left[\left[\left[1\right] = \begin{bmatrix} 1 \\ 1 \\ -x-1 \end{bmatrix}\right], [0], \frac{1}{1-s}, [1]\right]\right]$$

```
> e := PolHomHom(M, var);
```

$$e := \begin{bmatrix} 1 \\ 1 \\ -x-1 \end{bmatrix}$$

```
> r := [[1,0,0]];
```

$$r := [[1, 0, 0]]$$

```
> evalm(r &* e);
```

$$[\ 1]$$

Here, all homomorphisms from the module presented by **M** to the polynomial ring are represented by multiples of the generator given in the first entry of *H*. The evaluation of this generator at the residue class represented by *r* gives 1, and each multiple of this generator has the according multiple of 1 as value.

Example 2:

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

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

```
> M := matrix([[0,x,-y,0], [-x,0,z,0], [y,-z,0,0]]);
```

$$M := \begin{bmatrix} 0 & x & -y & 0 \\ -x & 0 & z & 0 \\ y & -z & 0 & 0 \end{bmatrix}$$

```
> H := PolHom(M, [0], var);
```

$$H := \left[\left[\left[\left[1, 0\right] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, [0, 1] = \begin{bmatrix} -z \\ -y \\ -x \\ 0 \end{bmatrix}\right], [[0, 0]], 2\frac{1}{(1-s)^3}, [0, 0, 2]\right]\right]$$

```
> HH := PolHom(H[2], [0], var);
```

$$HH := \left[\left[\left[\left[1, 0\right] = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, [0, 1] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right], [[0, 0]], 2\frac{1}{(1-s)^3}, [0, 0, 2]\right]\right]$$

```
> e := PolHomHom(M, var, HHM);
```

$$e := \begin{bmatrix} -z & 0 \\ -y & 0 \\ -x & 0 \\ 0 & 1 \end{bmatrix}$$

```
> HHM;
```

$$\left[\left[\left[\left[1, 0\right] = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, [0, 1] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right], [[0, 0]], 2\frac{1}{(1-s)^3}, [0, 0, 2]\right]\right]$$

```
> r := [[1,0,1,1]];
```

$$r := [[1, 0, 1, 1]]$$

```
> v := evalm(r &* e);
```

$$v := [-z-x \quad 1]$$

```
> h := evalm([[2],[-1]]);
```

$$h := \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

```
> evalm(v &* h);
```

$$[-2z - 2x - 1]$$

Here, the module of homomorphisms from the module presented by **M** to the polynomial ring is presented with two generators $\phi 1$, $\phi 2$, see the first component of *H*. The image of the residue class represented by *r* in **M** under the canonical homomorphism represented by *e* is represented by *v*. In this example the homomorphism $2\,\phi 1 - \phi 2$ is evaluated at the residue class represented by *r* by multiplying the row *v* by the column *h*.

See Also:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolKernel, PolCokernel, PolHom, PolExt1, PolExtn, PolParametrization, PolTorsion, PolSyzOp.

Involutive[PolIndexRegularity] - return index of regularity of the graded module of a residue class module

Calling Sequence:

PolIndexRegularity()

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 PolHilbertSeries. Then ***PolIndexRegularity*(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. PolHilbertPolynomial, PolHilbertFunction, i. e. the smallest *r* such that the filtered Hilbert function PolHF and the filtered Hilbert polynomial PolHP agree on all integers greater or equal to *r*.
- The command refers to the last call of InvolutiveBasis.

Examples:

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

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

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

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

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

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

```
> PolIndexRegularity();
```

$$2$$

```
> PolHilbertSeries(lambda);
```

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

```
> PolTabVar();
```

$$[xy+yz+zx, [1, 2, 3], xy]$$
$$[1+yz^2+z^2x, [1, *, 3], z^2x]$$
$$[y^2z^2+y+z, [*, 2, 3], y^2z^2]$$

```
> PolHilbertPolynomial();
```

$$6$$

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

$$6s-3$$

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

See Also:

InvolutiveBasis, PolTabVar, PolHilbertPolynomial, PolHP, PolHilbertFunction, PolHF, PolCartanCharacter.

Involutive[PolIntersection] - intersect two submodules of a free module over a polynomial ring

Calling Sequence:

PolIntersection(L1,L2,var)

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 polynomial ring

Description:

- ***PolIntersection*** computes a Janet basis of the intersection of the submodules generated by **`L1`** and **`L2`** in a free module of tuples over the polynomial ring in the variables **`var`**.
- The entries of **`L1`** and **`L2`** are polynomials in case of ideals, i. e. submodules 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. 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 result of ***PolIntersection*** is a list of polynomials or a list of lists of polynomials according to the structure of the input.

Examples:

```
> with(Involutive):
```

Example 1: intersection of ideals

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

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

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

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

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

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

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

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

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

```
> L1 := [x^2+y^2-4,x^2-y^2]; L2 := [x^2-x-1];
```

$$L1 := [x^2 + y^2 - 4, x^2 - y^2]$$

$$L2 := [x^2 - x - 1]$$

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

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

Example 2: intersection of two submodules of the free module of rank 2 over the polynomial ring in the variables *x, y, z*

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

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

```
> L1 := [[x^2+y^2, z^2], [x^4, 0]]; L2 := [[z^2, x^2+y^2], [0, x^4]];
```

$$L1 := [[x^2 + y^2, z^2], [x^4, 0]]$$

$$L2 := [[z^2, x^2 + y^2], [0, x^4]]$$

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

$$[[0, z^2\,x^4], [z^2\,x^4, 0]]$$

See Also:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, PolResolution, PolLeftInverse, PolRightInverse, PolKernel, PolSum, PolDirectSum, PolSubFactor, PolSyzOp.

Involutive[PolInvReduce] - return the normal form with respect to a Janet basis

Calling Sequence:

PolInvReduce(f,B,var,ord,mode)

Parameters:

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

Description:

- ***PolInvReduce*** returns the normal form representing the residue class of **`f`** modulo the submodule of the free 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 an ideal in the polynomial ring. 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.)
- As optional fourth 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 InvolutiveBasis 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 ***PolInvReduce*** 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 PolCoeff.)
- 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. InvolutiveBasis), one can specify a right hand side for **`f`** in order to let ***PolInvReduce*** 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(Involutive):
```

Example 1:

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

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

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

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

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

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

```
> PolInvReduce(z^4, B, var);
```

$$z$$

```
> f := x^2+y^2;
```

$$f := x^2+y^2$$

```
> PolInvReduce(f, B, var);
```

$$-z^2$$

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

```
> PolInvReduce(f, B, var, "C");
```

$$[-z^2, [x-y-z, 2, 0, 0]]$$

```
> f - expand((x-y-z)*B[1] + 2*B[2]);
```

$$-z^2$$

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

```
> L1 := [x+y+z=a, x*y+y*z+z*x=b, x*y*z-1=c];
```

$$L1 := [x+y+z=a, xy+yz+zx=b, xyz-1=c]$$

```
> B1 := InvolutiveBasis(L1, var);
```

$$B1 := [x+y+z=a, y^2+yz+z^2=za+ya-b, z^3-1=z^2 a-zb+c, -y+z^3 y=cy+az^2 y-bzy]$$

```
> PolTabVar();
```

$$[x+y+z=a, [x, y, z], x]$$
$$[y^2+yz+z^2=za+ya-b, [*, y, z], y^2]$$
$$[z^3-1=z^2 a-zb+c, [*, *, z], z^3]$$
$$[-y+z^3 y=cy+az^2 y-bzy, [*, *, z], z^3 y]$$

```
> f;
```

$$x^2+y^2$$

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

$$-z^2 = -za-ya+2b-xa+x^2+y^2$$

A list of polynomials can be reduced in one step:

```
> PolInvReduce([x^2+y^2, x*y+y*z=p, x*y*z], B1, var);
```

$$[-z^2, yz+z^2=za-b+p, 1]$$

Changing the polynomial ordering and the ordering of variables respectively:

```
> PolInvReduce(f, B1, var, 1);
```

$$-z^2$$

```
> varnew := [z,y,x]: PolInvReduce(f, B, varnew);
```

$$x^2+y^2$$

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 := InvolutiveBasis(L2, [x,y]);
```

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

```
> PolInvReduce([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 := InvolutiveBasis(L3, [x,y]);
```

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

```
> PolInvReduce(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 := InvolutiveBasis(L4, [x=3,y=2]);
```

$$B4 := [x^2 - y^3, y^6, xy^6]$$

```
> PolInvReduce(x^4, B4, [x=3,y=2]);
```

$$0$$

See Also:

InvolutiveBasis, InvolutiveBasisFast, InvolutiveBasisGINV, PolTabVar, PolInvReduceFast, FactorModuleBasis, PolCoeff.

Involutive[PolInvReduceFast] - return the normal form with respect to a Janet basis (C++ version)

Calling Sequence:

PolInvReduceFast(f,B,var,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 polynomial ring)
`ord`	-	(optional) type of monomial ordering
`mode`	-	(optional) string specifying options for the computation
`opt`	-	(optional) equation specifying options for the computation

Description:

- ***PolInvReduceFast*** invokes the C++ version of the command PolInvReduce. 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 ***PolInvReduceFast*** is called with a non-standard option, then PolInvReduce is applied internally to the same data.
- The parameter **`B`** should be the result of InvolutiveBasisFast. If this is not the case, InvolutiveBasisFast is applied to **`B`** before starting the involutive reductions. (See, however, the description of the option "L" below.)
- The parameters **`var`** and **`ord`** have the same meaning as in PolInvReduce.
- 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 "JB" instead.)
- If the letter "C" is given in **`mode`**, then ***PolInvReduceFast*** 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 ***PolInvReduceFast*** does *not* check whether the given involutive basis **`B`** equals the one which was computed by the last call of InvolutiveBasisFast. This option should speed up the repetitive use of ***PolInvReduceFast***. Note that, even if the computations of ***PolInvReduceFast*** 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 InvolutiveBasisFast has been run before using the option "char"=*c*, then this option must also be given to ***PolInvReduceFast*** in order to perform involutive reductions in characteristic *c* (cf. Example 3).
- Using the option "C++" of InvolutiveOptions, the command ***PolInvReduce*** is replaced by ***PolInvReduceFast*** for the current Maple session (which has the corresponding effect on all Maple procedures that call ***PolInvReduce***).

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

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

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

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

```
> PolInvReduceFast(z^4, B, var);
```

$$z$$

```
> PolInvReduceFast(x^2+y^2, B, var);
```

$$-z^2$$

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 := InvolutiveBasisFast(L, var);
```

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

```
> PolInvReduceFast(z^4=[0,0,0], B, var);
```

$$z=[-z^3, z^2, -z]$$

```
> PolInvReduceFast(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 := InvolutiveBasisFast(L, var, "char"=7);
```

$$B := [x+2y+3z, y^2+z^2, yz^2+4z^3+5, z^4+3y+2z]$$

```
> PolInvReduceFast(x^3, B, var, "char"=7);
```

$$4z^3+6$$

See Also:

InvolutiveBasisFast, InvolutiveBasis, InvolutiveOptions, PolTabVar, FactorModuleBasis, PolInvReduce, PolHilbertSeries, Syzygies, SyzygyModule, SyzygyModuleFast.

Involutive[PolInvReduceGINV] - Python/C++ version of PolInvReduce

Calling Sequence:

PolInvReduceGINV(f,B,var,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 polynomial ring)
`ord`	-	(optional) type of monomial ordering
`mode`	-	(optional) string specifying options for the computation
`opt`	-	(optional) equation specifying options for the computation

Description:

- ***PolInvReduceGINV*** invokes the version of the command PolInvReduce which uses the C++ module **ginv** for Python to perform the involutive reduction.
- The parameter **`B`** should be the result of InvolutiveBasisGINV. If this is not the case, InvolutiveBasisGINV is applied to **`B`** before starting the involutive reductions. (See, however, the description of the option "L" below.)
- The parameters **`var`** and **`ord`** have the same meaning as in PolInvReduce.
- 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 corresponding process "python" instead.)
- If the letter "C" is given in **`mode`**, then ***PolInvReduceGINV*** 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 ***PolInvReduceGINV*** does *not* check whether the given involutive basis **`B`** equals the one which was computed by the last call of InvolutiveBasisGINV. This option should speed up the repetitive use of ***PolInvReduceGINV***. Note that, even if the computations of ***PolInvReduceGINV*** rely upon the basis computed by the Python/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`**.
- Possible left hand sides of the optional equations **`opt`** are the strings "char", "algext", "transext", "Name", "quiet", "donotread",
- If InvolutiveBasisGINV has been run before using the option "char"=*c*, then this option must also be given to ***PolInvReduceFast*** in order to perform involutive reductions in characteristic *c* (cf. Example 3).
- The right hand side of an equation "algext"=*p* in **`opt`** is expected to be a univariate polynomial in an indeterminate ζ which does not occur in **`var`**. The coefficients of *p* must be algebraic over the ground field in the sense that they are rational expressions in RootOf and indeterminates ξ used in previously given right hand sides of other equations "algext"=*q* in **`opt`**. This extends the ground field (defined so far) by ζ which has minimal polynomial *p*, i.e. every occurrence of ζ in **`L`** is subject to the relation $p = 0$ (cf. Example 3).
- The right hand side of an equation "transext"=*z* in **`opt`** is expected to be a name for an indeterminate. This extends the ground field (defined so far) by a new transcendental element *z*.
- Using the option "GINV" of InvolutiveOptions, the command ***PolInvReduce*** is replaced by ***PolInvReduceGINV*** for the current Maple session (which has the corresponding effect on all Maple procedures that call ***PolInvReduce***).
- The right hand side of an equation "Name"=s is expected to be a string. ***PolInvReduceGINV*** appends *s* to the default name for the temporary file to which the input for **ginv** is written.
- The right hand side of an equation "donotread"=s is expected to be a boolean value. If s equals true, then ***PolInvReduceGINV*** does not read the result produced by the Python/C++ program and does not return a result.
- As right hand side of an equation "quiet"=t in **`opt`**, a boolean value t is expected. The default value is false. If t equals true, then no intermediate output is produced on the screen by the Python/C++ program.

- For more information about **ginv**, cf. http://invo.jinr.ru and http://wwwb.math.rwth-aachen.de/Janet.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

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

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

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

```
> PolInvReduceGINV(z^4, B, var);
```

$$z$$

```
> PolInvReduceGINV(x^2+y^2, B, var);
```

$$-z^2$$

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 := InvolutiveBasisGINV(L, var);
```

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

```
> PolInvReduceGINV(z^4=[0,0,0], B, var);
```

$$z=[-z^3, z^2, -z]$$

```
> PolInvReduceGINV(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 := InvolutiveBasisGINV(L, var, "char"=7);
```

$$B := [x+2y+3z, y^2+z^2, yz^2+4z^3+5, z^4+3y+2z]$$

```
> PolInvReduceGINV(x^3, B, var, "char"=7);
```

$$4z^3+6$$

See Also:

InvolutiveBasis, InvolutiveBasisGINV, InvolutiveOptions, PolTabVar, FactorModuleBasis, PolInvReduce, PolInvReduceFast, PolHilbertSeries, Syzygies, SyzygyModule

Involutive[PolKernel] - return presentation of the kernel of a homomorphism between two finitely presented modules over a polynomial ring

Calling Sequence:

PolKernel(M,A,N,var,K)

Parameters:

M	-	list (of lists of the same length) of polynomials or matrix with polynomial entries
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 polynomial ring
K	-	(optional) symbol

Description:

- ***PolKernel*** returns a presentation of the kernel of the homomorphism which is represented by **A**. The homomorphism is understood as a map from the module presented by **M** to the module presented by **N** (i.e., the elements of **M** and **N** are considered as elements of a free module of tuples over the polynomial ring with indeterminates **var** of appropriate rank, and the domain and the range of the homomorphism are the factor modules of the respective free modules modulo the submodules generated by the elements of **M** resp. **N**).
- If **M** and **N** are lists, then the entries of **M** and **N** are polynomials in case of ideals, i.e. submodules of the free module of rank one, or lists of polynomials of length *m* (resp. *n*), representing elements of the free module of *m*-tuples (resp. *n*-tuples) over the polynomial ring. If **M** or **N** is a matrix, then the generators are extracted from the rows of **M** resp. **N**.
- The parameter **A** represents a homomorphism from the free module of *m*-tuples to the free module of *n*-tuples which maps the submodule generated by **M** into the submodule generated by **N**. If **A** is a matrix, then this homomorphism is given by multiplying **A** from the right to *m*-tuples.
- If **A** is a list, then it contains one polynomial in **var** in the case of ideals or a list of lists of polynomials in **var** of the same length. In the latter case the number of lists must be equal to *m*, and the common length of these lists must be equal to *n*. If **A** is a matrix, then the number of rows must be equal to *m*, and the number of columns must be equal to *n*.
- The result of ***PolKernel*** is a list with four entries. The first one defines the abstract generators of the constructed presentation of the kernel. The second entry is a list of the relations imposed on the abstract generators of the presentation. Finally, the third and the fourth entry of the result give the Hilbert series (see PolHilbertSeries) resp. the Cartan characters (see PolCartanCharacter) of the kernel.
- 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 is a list of polynomials representing a residue class in the module presented by **M**. It corresponds to the *i*th abstract generator. Hence, the right hand sides of the first entry provide a generating set for the kernel.
- 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 annihilator 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 *R* of the free module *S* over the polynomial ring with indeterminates **var**, where the rank of *S* equals the number of abstract generators, such that the kernel is isomorphic to the factor module *S* / *R*.
- The third entry of the result is the Hilbert series (according to standard degrees) of the kernel, see PolHilbertSeries.
- The fourth entry of the result is the list of Cartan characters of the kernel as defined in PolCartanCharacter.

- If the optional fifth parameter **K** is provided, then a matrix is formed whose rows are the generators given in the presentation of the kernel, i.e. the right hand sides in the first entry of the output list, and this matrix is assigned to the symbol **K**.

Examples:

```
> with(Involutive):
```

Example 1:

Let R be the univariate polynomial ring in x with rational coefficients.

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

$$var := [x]$$

```
> M := [x^2-1]; N := [x-1];
```

$$M := [x^2 - 1]$$

$$N := [x - 1]$$

```
> A := [1];
```

$$A := [1]$$

represents the homomorphism (a -> a) from $R / (x^2 - 1)$ to $R / (x - 1)$. A presentation of the kernel of this map is:

```
> PolKernel(M, A, N, var);
```

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

Hence, the kernel is generated by one element which, multiplied by $(x + 1)$, gives zero in $R / (x^2 - 1)$.

```
> PolKernel(M, A, N, var, 'K');
```

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

```
> evalm(K);
```

$$[-x + 1]$$

Example 2:

Let again R be the univariate polynomial ring in x with rational coefficients.

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

$$var := [x]$$

```
> M := [[x,1,1], [1,x,1]];
```

$$M := [[x, 1, 1], [1, x, 1]]$$

```
> A := matrix([[x,0,0], [0,x,0], [0,0,x]]);
```

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

represents the multiplication by x on the module $R^3 / < M >$.

```
> PolKernel(M, A, M, var);
```

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

The kernel of this homomorphism is zero.

Now we consider the map from $R^3 / < M >$ to the zero module which multiplies every element of $R^3 / < M >$ by x.

```
> B := matrix([[x], [x], [x]]);
```

$$B := \begin{bmatrix} x \\ x \\ x \end{bmatrix}$$

```
> PolKernel(M, B, [0], var, 'K');
```

$$\left[[[1, 0] = [-1, 0, 1], [0, 1] = [-1, 1, 0]], [[0, x - 1]], 2 + \frac{s}{1 - s}, [1]\right]$$

We obtain another presentation of the module $R^3 / < M >$.

```
> evalm(K);
```

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

```
> evalm(K &* B);
```

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

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolHom, PolHomHom, PolExt1, PolExtn, PolParametrization, PolTorsion, PolSyzOp.

Involutive[PolLeftInverse] - compute left inverse of a polynomial matrix

Calling Sequence:

PolLeftInverse(M,var)

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 polynomial ring)

Description:

- ***PolLeftInverse*** computes (if possible) a left inverse of the polynomial matrix **`M`**, i.e. a polynomial matrix L 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, ***PolLeftInverse*** forms a matrix by taking the lists in **`M`** as rows and computes a left inverse of this matrix.
- If no left inverse of **`M`** exists, ***PolLeftInverse*** returns FAIL.
- If a left inverse L of **`M`** exists, ***PolLeftInverse*** 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 PolRightInverse.

Examples:

```
> with(Involutive):
```

In the first example we give the input as a matrix:

```
> M := matrix([[2*x^2, 4*x^2-2, 0], [x^2, 0, x^2-1], [-1, 0, 0], [2*x^2, 2*x^2, x^2]]);
```

$$M := \begin{bmatrix} 2x^2 & 4x^2-2 & 0 \\ x^2 & 0 & x^2-1 \\ -1 & 0 & 0 \\ 2x^2 & 2x^2 & x^2 \end{bmatrix}$$

```
> PolLeftInverse(M, [x]);
```

$$\begin{bmatrix} 0 & 0 & -1 & 0 \\ -\frac{1}{2}+x^2 & 2x^2 & 3x^2 & -2x^2+2 \\ -x^2 & -2x^2-1 & -3x^2 & 2x^2-1 \end{bmatrix}$$

A list of lists of the same length of polynomials is also accepted and interpreted as the list of rows of a matrix:

```
> M := [[2*x^2, 4*x^2-2, 0], [x^2, 0, x^2-1], [-1, 0, 0], [2*x^2, 2*x^2, x^2]];
```

$$M := [[2x^2, 4x^2-2, 0], [x^2, 0, x^2-1], [-1, 0, 0], [2x^2, 2x^2, x^2]]$$

```
> PolLeftInverse(M, [x]);
```

$$\left[[0, 0, -1, 0], \left[-\frac{1}{2}+x^2, 2x^2, 3x^2, -2x^2+2\right], [-x^2, -2x^2-1, -3x^2, 2x^2-1]\right]$$

Substitute y for x in the second column of the matrix in the preceding example and consider y as a parameter, i.e. as element of the ground field:

```
> M := matrix([[2*x^2, 4*y^2-2, 0], [x^2, 0, x^2-1], [-1, 0, 0], [2*x^2, 2*y^2, x^2]]);
```

$$M := \begin{bmatrix} 2x^2 & 4y^2-2 & 0 \\ x^2 & 0 & x^2-1 \\ -1 & 0 & 0 \\ 2x^2 & 2y^2 & x^2 \end{bmatrix}$$

```
> PolLeftInverse(M, [x]);
```

$$\begin{bmatrix} 0 & 0 & -1 & 0 \\ \frac{1}{2}\frac{1}{2y^2-1} & 0 & \frac{x^2}{2y^2-1} & 0 \\ -\frac{y^2}{2y^2-1} & -1 & -\frac{x^2}{2y^2-1} & 1 \end{bmatrix}$$

If M is considered as a polynomial matrix in the variables x and y, then M has no left inverse:

```
> PolLeftInverse(M, [x,y]);
```

FAIL

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, PolRightInverse, AddRhs.

Involutive[PolMinPoly] - minimal polynomial of an element of the residue class ring

Calling Sequence:

PolMinPoly(m,B,var,mode)

Parameters:

`m` - element of the residue class ring (a polynomial in **`var`**)
`B` - Janet basis
`var` - list of variables (of the polynomial ring)
`mode` - (optional) string or equation whose left hand side is a string

Description:

- ***PolMinPoly*** returns the minimal polynomial for the residue class represented by **`m`** in the residue class ring of the polynomial ring modulo the ideal generated by **`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 polynomial ring. (***PolMinPoly*** does not check whether **`B`** is a Janet basis with respect to **`var`**.)
- By default the result of ***PolMinPoly*** is a polynomial in the indeterminate λ. The name of the indeterminate can be changed by the option described below.
- Note, minimal polynomials are defined for polynomial rings only. So this command cannot be applied to modules.
- 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 ***PolMinPoly*** 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(Involutive):
```

Example 1:

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

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

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

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

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

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

```
> PolMinPoly(y, B, var);
```

$$\lambda^3$$

```
> PolMinPoly(x^2+y, B, var);
```

$$\lambda^5 + \lambda^3 - 2\lambda^4$$

Example 2:

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

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

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

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

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

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

```
> FactorModuleBasis(var);
```

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

```
> PolMinPoly(x*y, B, var);
```

$$\lambda^2$$

Example 3:

```
> var := [a,b];
```

$$var := [a, b]$$

```
> L2 := [a*b-b, a+b^2];
```

$$L2 := [a\,b - b, a + b^2]$$

```
> B2 := InvolutiveBasis(L2, var);
```

$$B2 := [a + b^2, a\,b - b, a^2 - a]$$

```
> FactorModuleBasis(var);
```

$$[1, b, a]$$

```
> PolMinPoly(a, B2, var, "var"=X);
```

$$X^2 - X$$

```
> PolMinPoly(a, B2, var, "subs"=X+Y);
```

$$(X+Y)^2 - X - Y$$

```
> B3 := InvolutiveBasis([a^31], [a]);
```

$$B3 := [a^{31}]$$

```
> PolMinPoly(a, B3, [a]);
Error, (in Involutive/PolMinPoly) stopped calculation of minimal polynomial since upper bound for the degree is
reached.
> PolMinPoly(a, B3, [a], "var"=lambda, "degree"=40);
```

$$\lambda^{31}$$

See Also:

InvolutiveBasis, FactorModuleBasis, PolTabVar, PolRepres, CoeffList, PolHilbertSeries.

Involutive[PolParametrization] - return a matrix whose rows generate a module whose syzygy module equals a given module over a polynomial ring

Calling Sequence:

PolParametrization(L,var)

Parameters:

`L` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the polynomial ring

Description:

- ***PolParametrization*** returns a polynomial matrix *M* such that the module generated by **L** over the polynomial ring with indeterminates **`var`** is the syzygy module of the module generated by the rows 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 ***PolParametrization*** is a matrix *M* such that the product of the matrix composed of the rows of **L** by *M* is a zero matrix. Moreover, the columns of *M* generate the kernel of the linear map of column vectors with polynomial entries which is defined by **L**. In this sense, the result of ***PolParametrization*** parametrizes the kernel of this linear map.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

```
> L1 := [[x,y,z]];
```

$$L1 := [[x, y, z]]$$

```
> L2 := PolParametrization(L1, var);
```

$$L2 := \begin{bmatrix} -y & -z & 0 \\ x & 0 & -z \\ 0 & x & y \end{bmatrix}$$

```
> L3 := PolParametrization(L2, var);
```

$$L3 := \begin{bmatrix} z \\ -y \\ x \end{bmatrix}$$

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

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

```
> evalm(L2 &* L3);
```

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

Example 2:

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

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

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

$$L := [[y, xy, 0], [y, 0, y^2]]$$

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

$$\begin{bmatrix} -xy \\ y \\ x \end{bmatrix}$$

See Also:

InvolutiveBasis, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolSyzOp, PolKernel, PolHom, PolHomHom, PolExt1, PolExtn, PolTorsion, Parametrization.

Involutive[PolRepres] - matrix representation with respect to a factor module basis

Calling Sequence:

PolRepres(m,B,var,FB,ord,mode)

Parameters:

`m`	-	polynomial in **`var`**
`B`	-	Janet basis
`var`	-	list of variables (of the polynomial ring)
`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:

- ***PolRepres*** returns the matrix (in column convention) of the multiplication of **`m`** on the free module over the polynomial ring 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 FactorModuleBasis.
- Note, the ground field is allowed to be the field of complex numbers or the field of rational functions (in one or several variables) over it.
- 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 standard monomials 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 FactorModuleBasis(**`var`**, "C")). The number of rows and the number of columns equal the number of cones in this case.
- If the Janet basis **`B`** has been computed with respect to a monomial ordering different from the default one (see InvolutiveBasis), then the argument **`ord`** is expected to be the same as the one given to the previous call of InvolutiveBasis. Otherwise leading monomials might be determined incorrectly, resulting in an error message of ***PolRepres*** when the normal form of an **`m`**-multiple of an element of **`FB`** is not expressible as linear combination in terms of **`FB`** (cf. Example 4 below).
- 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 ***PolRepres*** 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(Involutive):
```

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 := InvolutiveBasis(L, var);
```

$$B := \left[x^2 - 2y, -\frac{1}{2}y^2 + y^3, xy^2 - y^2\right]$$

```
> FB := FactorModuleBasis(var);
```

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

```
> Mx := PolRepres(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 := PolRepres(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);
```

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

```
> PolMinPoly(y, B, var);
```

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

Example 2: Computations over a field of rational functions

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

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

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

$$L := [yx^2 + y^2 + z^2 - 1, y^2 + z^2 - 1]$$

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

$$B := [y^2 + z^2 - 1, x^2, (z^2 - 1)x + xy^2]$$

```
> FB := FactorModuleBasis(var);
```

$$FB := [1, y, x, xy]$$

```
> Mx := PolRepres(x, B, var, FB);
```

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

```
> My := PolRepres(y, B, var, FB);
```

$$My := \begin{bmatrix} 0 & -z^2 + 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -z^2 + 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

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];
```

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

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

$$B := [xy^2 - y, yx^2 - x]$$

```
> FB := FactorModuleBasis(var);
```

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

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

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

```
> PolRepres(x, B, var, FB);
```

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

```
> PolRepres(y, B, var, FB);
```

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

Example 4: Using a monomial ordering which is not the default one

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

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

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

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

```
> B := InvolutiveBasis(L, var, 1);
```

$$B := [y^5 + y, -y^4 + xy, x^2 + y^2]$$

```
> FB := FactorModuleBasis(var);
```

$$FB := [1, y, x, y^2, y^3, y^4]$$

```
> PolRepres(x, B, var, FB);
Error, (in Involutive/PolRepres) the given vector space basis is not the factor module basis for the residue class
module under consideration.
```

```
> PolRepres(x, B, var, FB, 1);
```

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

Example 5: Matrix representation with respect to an infinite basis of tuples

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

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

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

$$L := [[yx^2 + y^2 + z^2 - 1, x], [y, y^2 + z^2 - 1]]$$

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

$$B := [[y, y^2 + z^2 - 1], [yx^2 + y^2 + z^2 - 1, x]]$$

```
> FB := FactorModuleBasis(var);
```

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

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

$$[[0, 1], [0, y], [1, 0], [x, 0], [x^2, 0]]$$

```
> Mx := PolRepres(x, B, var, FB);
```

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

```
> My := PolRepres(y, B, var, FB);
```

$$My := \begin{bmatrix} 0 & -z^2 + 1 & 0 & 0 & -x \\ 1 & 0 & 0 & 0 & 0 \\ 0 & -y & y & 0 & -y^2 - z^2 + 1 \\ 0 & 0 & 0 & y & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

See Also:

InvolutiveBasis, FactorModuleBasis, PolTabVar, PolHilbertSeries, PolMinPoly, CoeffList.

Involutive[PolResolution] - return free resolution of a factor module of a free module over a polynomial ring

Calling Sequence:

PolResolution(L,var,mode,tr)

Parameters:

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

- ***PolResolution*** computes a free resolution of *R*^*m*/<**L**> 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.
- As optional third 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, ***PolResolution*** 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(Involutive):
```

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();
```

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

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

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

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

$$[[5, 6], [5, 4, 5, 4, 3], [1, 2, 3, 4]]$$

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

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

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

$$[[6], [3, 5, 4], [1, 2, 3]]$$

```
> r := PolResolution(L, var, "G");
```

$$r := \left[\left[\begin{bmatrix} 1 & -z^2 & y+z & -1 \\ y & -yz^2+z^3-1 & -z^2 & z+x \end{bmatrix}, \begin{bmatrix} 0 & xyz-1 & -xy-yz-zx \\ xy+yz+zx & -x-y-z & 0 \\ xz^2+yz^2+1 & -zx-yz-z^2 & x+y+z \\ y^2z^2+y+z & -1-y^2z-yz^2 & y^2+yz+z^2 \end{bmatrix}, \begin{bmatrix} x+y+z \\ xy+yz+zx \\ xyz-1 \end{bmatrix}\right]\right]$$

```
> map(expand, evalm(r[1] &* r[2])); map(expand, evalm(r[2] &* r[3]));
```

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

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

Example 2:

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

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

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

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

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

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

```
> PolResolution(L, var, "G", 1);
```

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

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, Syzygies, SyzygyModule, PolShorterResolution, PolShortestResolution, PolResolutionDim, PolEulerChar.

Involutive[PolResolutionDim] - return ranks of the free modules in a free resolution of a factor module of a free module over a polynomial ring

Calling Sequence:

PolResolutionDim(L,var,tr)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the polynomial ring)
`tr` - (optional) positive integer (truncate resolution to length **`tr`**)

Description:

- ***PolResolutionDim*** returns the list of ranks of the free modules over the polynomial ring in the variables **`var`** occurring in the free resolution constructed by PolResolution applied to **`L`**.
- The ranks of the free modules are computed from the number of non-multiplicative variables for the elements in the Janet basis of **`L`**. Therefore, in contrast to PolResolution, only one involutive basis computation has to be performed. The rank of the domain of the *i*-th homomorphism computed by PolResolution (represented by the last but *i*-th matrix in its resulting list) equals the sum over *j*, of the number of possible choices of *i* variables from the set of non-multiplicative variables of the *j*-th element of the Janet basis of **`L`**.
- 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`**.
- If the optional parameter **`tr`** is supplied, ***PolResolutionDim*** returns only the list of ranks of the first (**`tr`**+1) free modules constructed by PolResolution.
- The result of ***PolResolutionDim*** is a list of positive integers. The last integer in the resulting list equals 1 if **`L`** generates an ideal in the polynomial ring in the variables **`var`**; otherwise **`L`** generates a submodule of a free modules of tuples over this polynomial ring in which case the last integer equals the length of these tuples. For *i* greater than 1, the last but *i*-th entry in the result of ***PolResolutionDim*** equals the rank of the domain of the $(i-1)$-th homomorphism computed by PolResolution (represented by the last but $(i-1)$-th matrix in the result of PolResolution).
- 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(Involutive):
```

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);
```

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

```
> PolResolutionDim(L, var, 2);
```

$$[5, 4, 1]$$

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);
```

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

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolShorterResolution, PolShortestResolution, PolEulerChar.

Involutive[PolRightInverse] - compute right inverse of a polynomial matrix

Calling Sequence:

PolRightInverse(M,var)

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 polynomial ring)

Description:

- ***PolRightInverse*** computes (if possible) a right inverse of the polynomial matrix **`M`**, i.e. a polynomial matrix *R* 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, ***PolRightInverse*** forms a matrix by taking the lists in **`M`** as rows and computes a right inverse of this matrix.
- If no right inverse of **`M`** exists, ***PolRightInverse*** returns FAIL.
- If a right inverse *R* of **`M`** exists, ***PolRightInverse*** 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 PolLeftInverse.

Examples:

```
> with(Involutive):
```

In the first example we give the input as a matrix:

```
> M := matrix([[2*x^2, x^2, -1, 2*x^2], [4*x^2-2, 0, 0, 2*x^2], [0, x^2-1, 0, x^2]]);
```

$$M := \begin{bmatrix} 2x^2 & x^2 & -1 & 2x^2 \\ 4x^2-2 & 0 & 0 & 2x^2 \\ 0 & x^2-1 & 0 & x^2 \end{bmatrix}$$

```
> PolRightInverse(M, [x]);
```

$$\begin{bmatrix} 0 & -\frac{1}{2}+x^2 & -x^2 \\ 0 & 2x^2 & -2x^2-1 \\ -1 & 3x^2 & -3x^2 \\ 0 & -2x^2+2 & 2x^2-1 \end{bmatrix}$$

A list of lists of the same length of polynomials is also accepted and interpreted as the list of rows of a matrix:

```
> M := [[2*x^2, x^2, -1, 2*x^2], [4*x^2-2, 0, 0, 2*x^2], [0, x^2-1, 0, x^2]];
```

$$M := [[2x^2, x^2, -1, 2x^2], [4x^2-2, 0, 0, 2x^2], [0, x^2-1, 0, x^2]]$$

```
> PolRightInverse(M, [x]);
```

$$\left[\left[0, -\frac{1}{2}+x^2, -x^2\right], [0, 2x^2, -2x^2-1], [-1, 3x^2, -3x^2], [0, -2x^2+2, 2x^2-1]\right]$$

Substitute *y* for *x* in the second row of the matrix in the preceding example and consider *y* as a parameter, i.e. as element of the ground field:

```
> M := matrix([[2*x^2, x^2, -1, 2*x^2], [4*y^2-2, 0, 0, 2*y^2], [0, x^2-1, 0, x^2]]);
```

$$M := \begin{bmatrix} 2x^2 & x^2 & -1 & 2x^2 \\ 4y^2-2 & 0 & 0 & 2y^2 \\ 0 & x^2-1 & 0 & x^2 \end{bmatrix}$$

```
> PolRightInverse(M, [x]);
```

$$\begin{bmatrix} 0 & \frac{1}{2}\frac{1}{2y^2-1} & -\frac{y^2}{2y^2-1} \\ 0 & 0 & -1 \\ -1 & \frac{x^2}{2y^2-1} & -\frac{x^2}{2y^2-1} \\ 0 & 0 & 1 \end{bmatrix}$$

If M is considered as a polynomial matrix in the variables x and y, then M has no right inverse:

```
> PolRightInverse(M, [x,y]);
```

FAIL

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, PolLeftInverse, AddRhs.

Involutive[PolShorterResolution] - shorten (if possible) a free resolution of a finitely presented module over a polynomial ring

Calling Sequence:

PolShorterResolution(F,var)

Parameters:

`F` - list of matrices whose entries are polynomials in **`var`**

`var` - list of variables (of the polynomial ring)

Description:

- Given a (finite) free resolution of a finitely presented module over the polynomial ring in the variables **`var`**, ***PolShorterResolution*** tries to construct a shorter free resolution of the same module. This is possible whenever the last homomorphism between free modules in this free resolution admits a right inverse (see PolRightInverse).
- If the length m of the free resolution given by **F** is at least 3 and if the last morphism R_m between free modules given in **F** admits a right inverse S_m, then a shorter free resolution is obtained by removing the last free module, augmenting the last but first morphism R_{m-1} with S_m, i.e. replacing it by $(R_{m-1}\ S_m)$, and replacing the last but second morphism R_{m-2} by the transpose of $(R_{m-2}\ 0)$ in a compatible way (note also that the last but second free module in the given free resolution must be adjusted).
- If the length m of the free resolution given by **F** equals 2 and if the last morphism R_2 between free modules given in **F** admits a right inverse S_2, then a presentation of the module resolved by **F** is obtained by removing the last free module and augmenting the last but first morphism R_1 with S_2, i.e. by defining the presentation matrix $(R_1\ S_2)$.
- If the length m of the free resolution given by **F** is less than 2, then ***PolShorterResolution*** returns **F**.
- **F** is a list of matrices representing a free resolution of a finitely presented module over the polynomial ring in the variables **`var`**. Most commonly, **F** is the result of PolResolution.
- The result of ***PolShorterResolution*** is of the same format as the input **F**, i.e. a list representing a free resolution of the finitely presented module, which is shorter than the given one or equals the given one.
- The procedure described above can be iterated using the command PolShortestResolution.
- For more details, see A. Quadrat, D. Robertz, "Computation of bases of free modules over the Weyl algebras", Journal of Symbolic Computation 42 (11-12), 2007, pp. 1113-1141.

Examples:

```
> with(Involutive):
```

Example:

(see J.-F. Pommaret, *Partial Differential Equations and Group Theory: New Perspectives for Applications Kluwer*, 1994, p. 162)

```
> var := [D1,D2,D3];
```

$$var := [D1, D2, D3]$$

```
> R := [1,D1,D2,D3];
```

$$R := [1, D1, D2, D3]$$

```
> F1 := PolResolution(R, var, "G");
```

$$F1 := \left[\begin{bmatrix} D1 & -1 & D3 & -D2 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 & D2 & -D3 & 0 \\ -D1 & D2 & -D3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & D1 & -D2 \\ 0 & -1 & 0 & D1 & 0 & -D3 \end{bmatrix}, \begin{bmatrix} 0 & 0 & D3 & -D2 \\ 0 & D3 & 0 & -D1 \\ 0 & D2 & -D1 & 0 \\ D3 & 0 & 0 & -1 \\ D2 & 0 & -1 & 0 \\ D1 & -1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix} \right]$$

```
> F2 := PolShorterResolution(F1, var);
```

$$F2 := \left[\begin{bmatrix} -1 & 0 & 0 & D2 & -D3 & 0 & 0 \\ -D1 & D2 & -D3 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & D1 & -D2 & 0 \\ 0 & -1 & 0 & D1 & 0 & -D3 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & D3 & -D2 \\ 0 & D3 & 0 & -D1 \\ 0 & D2 & -D1 & 0 \\ D3 & 0 & 0 & -1 \\ D2 & 0 & -1 & 0 \\ D1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix} \right]$$

```
> F3 := PolShorterResolution(F2, var);
```

$$F3 := \left[\begin{bmatrix} 0 & 0 & D3 & -D2 & -1 & 0 & 0 & 0 \\ 0 & D3 & 0 & -D1 & 0 & 0 & 0 & -1 \\ 0 & D2 & -D1 & 0 & 0 & 0 & -1 & 0 \\ D3 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D1 & -1 & D3 & -D2 \end{bmatrix}, \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right]$$

```
> F4 := PolShorterResolution(F3, var);
```

$$F4 := \left[\left[\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ D2 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ D3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & D2 & -D3 & 0 & 0 \\ 0 & -D1 & D2 & -D3 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & D1 & -D2 & 0 \\ 0 & 0 & -1 & 0 & D1 & 0 & -D3 & 0 \end{bmatrix} \right] \right]$$

```
> PolShorterResolution(F4, var);
```

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

Hence, it was possible to reduce the length of the free resolution represented by **F1** in each step, finally arriving at a free resolution of length 1. These steps can be done at once by calling PolShortestResolution:

```
> F := PolShortestResolution(F1, var);
```

$$F := \left[\left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ D2 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ D3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & D2 & -D3 & 0 & 0 \\ 0 & -D1 & D2 & -D3 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & D1 & -D2 & 0 \\ 0 & 0 & -1 & 0 & D1 & 0 & -D3 & 0 \end{array}\right]\right]$$

In fact, the module presented by *R* is stably free because a right inverse of the presentation matrix obtained by PolShortestResolution admits a right inverse:

```
> PolRightInverse(F[1], var);
```

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

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolShortestResolution, PolResolutionDim, PolEulerChar.

Involutive[PolShortestResolution] - return a shortest free resolution of a finitely presented module over a polynomial ring

Calling Sequence:

PolShortestResolution(F,var)

Parameters:

`F` - list of matrices whose entries are polynomials in **`var`**

`var` - list of variables (of the polynomial ring)

Description:

- ***PolShortestFreeResolution*** iterates the application of PolShorterResolution to a (finite) free resolution of a finitely presented module over the polynomial ring in the variables **`var`** and returns a free resolution of the same module which cannot be shortened in this way anymore.
- **F** is either a matrix with polynomial entries or a list of matrices representing a free resolution of a finitely presented module over the polynomial ring in the variables **`var`**. In the first case, a free resolution of the module presented by **F** is computed first. In the second case, most commonly, **F** is the result of PolResolution. Then, in both cases, PolShorterResolution is applied repeatedly to the resolution until PolShorterResolution does not change the resolution anymore.
- The result of ***PolShortestFreeResolution*** is a list representing a free resolution of the finitely presented module.
- For more details, see A. Quadrat, D. Robertz, "Computation of bases of free modules over the Weyl algebras", Journal of Symbolic Computation 42 (11-12), 2007, pp. 1113-1141.

Examples:

```
> with(Involutive):
```

Example:

(see J.-F. Pommaret, *Partial Differential Equations and Group Theory: New Perspectives for Applications Kluwer*, 1994, p. 162)

```
> var := [D1,D2,D3];
```

$$var := [D1, D2, D3]$$

```
> R := [1,D1,D2,D3];
```

$$R := [1, D1, D2, D3]$$

```
> F1 := PolResolution(R, var, "G");
```

$$F1 := \left[\begin{bmatrix} D1 & -1 & D3 & -D2 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 & D2 & -D3 & 0 \\ -D1 & D2 & -D3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & D1 & -D2 \\ 0 & -1 & 0 & D1 & 0 & -D3 \end{bmatrix}, \begin{bmatrix} 0 & 0 & D3 & -D2 \\ 0 & D3 & 0 & -D1 \\ 0 & D2 & -D1 & 0 \\ D3 & 0 & 0 & -1 \\ D2 & 0 & -1 & 0 \\ D1 & -1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix} \right]$$

```
> F2 := PolShorterResolution(F1, var);
```

$$F2 := \left[\begin{bmatrix} -1 & 0 & 0 & D2 & -D3 & 0 & 0 \\ -D1 & D2 & -D3 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & D1 & -D2 & 0 \\ 0 & -1 & 0 & D1 & 0 & -D3 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & D3 & -D2 \\ 0 & D3 & 0 & -D1 \\ 0 & D2 & -D1 & 0 \\ D3 & 0 & 0 & -1 \\ D2 & 0 & -1 & 0 \\ D1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \end{bmatrix} \right]$$

```
> F3 := PolShorterResolution(F2, var);
```

$$F3 := \left[\begin{bmatrix} 0 & 0 & D3 & -D2 & -1 & 0 & 0 & 0 \\ 0 & D3 & 0 & -D1 & 0 & 0 & 0 & -1 \\ 0 & D2 & -D1 & 0 & 0 & 0 & -1 & 0 \\ D3 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ D2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ D1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D1 & -1 & D3 & -D2 \end{bmatrix}, \begin{bmatrix} 1 \\ D1 \\ D2 \\ D3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right]$$

```
> F4 := PolShorterResolution(F3, var);
```

$$F4 := \left[\left[\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ D2 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ D3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & D2 & -D3 & 0 & 0 \\ 0 & -D1 & D2 & -D3 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & D1 & -D2 & 0 \\ 0 & 0 & -1 & 0 & D1 & 0 & -D3 & 0 \end{bmatrix} \right] \right]$$

```
> PolShorterResolution(F4, var);
```

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

Hence, it was possible to reduce the length of the free resolution represented by **F1** in each step, finally arriving at a free resolution of length 1. These steps can be done at once by calling ***PolShortestResolution***:

```
> F := PolShortestResolution(F1, var);
```

$$F := \left[\left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ D2 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ D3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & D2 & -D3 & 0 & 0 \\ 0 & -D1 & D2 & -D3 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & D1 & -D2 & 0 \\ 0 & 0 & -1 & 0 & D1 & 0 & -D3 & 0 \end{array}\right]\right]$$

In fact, the module presented by *R* is stably free because a right inverse of the presentation matrix obtained by ***PolShortestResolution*** admits a right inverse:

```
> PolRightInverse(F[1], var);
```

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

See Also:

InvolutiveBasis, PolTabVar, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolShorterResolution, PolResolutionDim, PolEulerChar.

Involutive[PolSubFactor] - return presentation of a subfactor of a finitely presented module over a polynomial ring

Calling Sequence:

PolSubFactor(L1,L2,var,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 polynomial ring
`v` - (optional) name of the indeterminate for the Hilbert series of the subfactor (default: 's')

Description:

- ***PolSubFactor*** returns a presentation of the epimorphic image of the sum of the modules generated by **L1** and **L2** in the module presented by **L2**. This is a subfactor module of the factor module given by the free module of *m*-tuples over the polynomial ring in **var** 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 module generated by **L2** is a submodule of the module generated by **L1**. Then ***PolSubFactor*** returns a presentation of the module generated by the residue classes represented by the entries of **L1** in the module presented by **L2**.
- The entries of **L1** and **L2** are polynomials in case of ideals, i.e. submodules 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. 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 result of ***PolSubFactor*** 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 relations imposed on the abstract generators of the presentation. Finally, the third and the fourth entry of the result give the Hilbert series (see PolHilbertSeries) resp. the Cartan characters (see PolCartanCharacter) 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 annihilator 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 (according to standard degrees) of the given subfactor module, see PolHilbertSeries.
- The optional fourth argument to ***PolSubFactor*** 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 PolCartanCharacter.

Examples:

```
> with(Involutive):
> var := [x];
```

$$var := [x]$$

```
> PolSubFactor([x], [x^2], var);
```

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

```
> PolSubFactor([1], [x], var);
```

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

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

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

```
> PolSubFactor([[x^3+y^3, x^2]], [[x^4, 0], [0, x^4]], var);
```

$$\left[[[1]=[x^3+y^3, x^2]],[x^4],1+2s+3s^2+4s^3+4\frac{s^4}{1-s},[4,0]\right]$$

```
> PolSubFactor([x^3+y^2,x^2+1], [y^2,x^4], var, lambda);
```

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

See Also:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, PolResolution, PolKernel, PolHom, PolHomHom, PolExt1, PolExtn, PolTorsion, PolParametrization, PolSyzOp.

Involutive[PolSyzOp] - return a matrix whose rows generate the syzygy module of a finitely presented module over a polynomial ring

Calling Sequence:

PolSyzOp(L,var)

Parameters:

`L` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the polynomial ring

Description:

- ***PolSyzOp*** 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 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**). This matrix is constructed by computing the beginning of a free resolution of *M* using PolResolution. For more information about syzygies cf. also Syzygies.
- 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 in the indeterminates **var**. If **L** is a matrix, then the generators are extracted from the rows of **L**.
- The result of ***PolSyzOp*** 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 ***PolSyzOp*** is motivated by the corresponding command SyzOp in the package Janet.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

```
> S := PolSyzOp([x, y], var);
```

$$S := [-y \quad x]$$

```
> L := matrix([[x], [y]]);
```

$$L := \begin{bmatrix} x \\ y \end{bmatrix}$$

```
> evalm(S &* L);
```

$$[\ 0]$$

Example 2:

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

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

```
> L1 := matrix([[x], [y], [z]]);
```

$$L1 := \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

```
> L2 := PolSyzOp(L1, var);
```

$$L2 := \begin{bmatrix} -y & x & 0 \\ -z & 0 & x \\ 0 & -z & y \end{bmatrix}$$

```
> PolSyzOp(L2, var);
```

$$[z \quad -y \quad x]$$

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

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

See Also:

InvolutiveBasis, PolInvReduce, Syzygies, SyzygyModule, PolResolution, PolParametrization, PolHom, PolHomHom, PolExt1, PolExtn, PolTorsion, PolKernel, PolLeftInverse, PolRightInverse.

Involutive[PolTabVar] - display Janet's data, i. e. the generators, their leading monomials, multiplicative variables etc.

Calling Sequence:

PolTabVar()

Parameters:

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

Description:

- ***PolTabVar*** displays the data constructed by InvolutiveBasis. Therefore it is necessary to call ***InvolutiveBasis*** first. The data structure is a list of lists each corresponding to an element of the Janet basis.
- In the ideal case 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 polynomial ring, 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(Involutive):
> 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);
```

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

```
> PolTabVar();
```

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

Note some effects of the other monomial ordering:

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

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

```
> PolTabVar();
```

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

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];
```

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

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

$$[x+y+z=a, y^2+yz+z^2=za+ya-b, z^3-1=z^2 a-zb+c, z^3 y-y=cy-bzy+az^2 y]$$

```
> PolTabVar();
```

$$[x+y+z=a, [x, y, z], x]$$
$$[y^2+yz+z^2=za+ya-b, [*, y, z], y^2]$$
$$[z^3-1=z^2 a-zb+c, [*, *, z], z^3]$$
$$[z^3 y-y=cy-bzy+az^2 y, [*, *, z], z^3 y]$$

For modules the output is slightly more involved:

```
> InvolutiveBasis([[x^2-1, 0], [x*y, x*y], [0, y^2-1]], [x,y]);
```

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

```
> PolTabVar();
```

$$[[0, y^2-1], [*, y], [y^2, 2]]$$

$$[[xy, xy], [*, y], [xy, 1]]$$

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

$$[[x^2-1, 0], [x, y], [x^2, 1]]$$

$$[[y^3-y, 0], [*, y], [y^3, 1]]$$

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

See Also:

InvolutiveBasis, PolZeroSets, Stats, InvolutiveOptions, JanetGraph, PolInvReduce.

Involutive[PolTorsion] - return torsion submodule of a finitely presented module over a polynomial ring

Calling Sequence:

PolTorsion(L,var,v)

Parameters:

`L` - list (of lists of the same length) of polynomials or matrix with polynomial entries
`var` - list of variables of the polynomial ring
`v` - (optional) name of the indeterminate for the Hilbert series of the torsion submodule (default: 's')

Description:

- ***PolTorsion*** constructs a presentation of the torsion submodule 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 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 ***PolTorsion*** is a list with four entries. The first one defines the abstract generators of the constructed presentation of the torsion submodule in terms of representatives of residue classes in the given module. The second entry is a list of the relations imposed on the abstract generators of the presentation. Finally, the third and the fourth entry of the result give the Hilbert series (see PolHilbertSeries) resp. the Cartan characters (see PolCartanCharacter) of the torsion submodule.
- 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 torsion submodule which corresponds to the *i*th abstract generator. Hence, the torsion submodule is generated by the set of right hand sides in this first entry modulo **L**.
- 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 annihilator 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 torsion submodule is isomorphic to the factor module *M* / *N*.
- The third entry of the result is the Hilbert series (according to standard degrees) of the torsion submodule, see PolHilbertSeries.
- The optional third argument to ***PolTorsion*** 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 torsion submodule as defined in PolCartanCharacter.

Examples:

```
> with(Involutive):
```

Example 1:

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

$$var := [x]$$

```
> L := [[x,1,1], [1,x,1]];
```

$$L := [[x, 1, 1], [1, x, 1]]$$

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

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

The torsion submodule is generated by the residue class modulo **L** which is represented by $[1, -1, 0]$, and this element is annihilated by $x - 1$. The dimension of the torsion submodule as a vector space is 1. Its Cartan character is 0.

Example 2:

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

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

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

$$L := [[y, xy, 0], [y, 0, y^2]]$$

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

$$\left[[[1, 0] = [1, 0, y], [0, 1] = [1, x, 0]], [[0, y], [y, 0]], 2 + 2\frac{s}{1 - s}, [2, 0]\right]$$

The torsion submodule is generated by the residue classes modulo **L** which are represented by $[1, 0, y]$, $[1, x, 0]$, and these elements are annihilated by y. The third entry of the result gives the Hilbert series of the torsion submodule, the fourth entry gives its Cartan characters. The indeterminate of the Hilbert series can be changed via the optional third argument:

```
> PolTorsion(L, var, lambda);
```

$$\left[[[1, 0] = [1, 0, y], [0, 1] = [1, x, 0]], [[0, y], [y, 0]], 2 + 2\frac{\lambda}{1 - \lambda}, [2, 0]\right]$$

Example 3:

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

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

```
> L := [[y+x, x*y+1, 0], [x, 0, y^2]];
```

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

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

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

The torsion submodule is trivial.

See Also:

InvolutiveBasis, PolInvReduce, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution, PolSubFactor, PolKernel, PolHom, PolHomHom, PolExt1, PolExtn, PolParametrization, PolSyzOp.

Involutive[PolWeightedHilbertSeries] - Hilbert series of the module generated by the last InvolutiveBasis command (weighted version)

Calling Sequence:

PolWeightedHilbertSeries(degrees,v)

Parameters:

`degrees` - list of variables associated with degrees (weights)
`v` - (optional) name of the indeterminate (default 's')

Description:

- The command ***PolWeightedHilbertSeries*** is completely analogous to PolHilbertSeries with the only exception that the standard grading of the module of m-tuples over the polynomial ring is replaced by the grading defined by **`degrees`**. The information is derived from the last call of InvolutiveBasis.
- 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 module. The above syntax is therefore extended to $[x_1=d_1, ..., x_n=d_n, 1=e_1, ..., m=e_m]$, cf. also InvolutiveBasis.
- The special case where all degrees are equal to 1 yields the same result (in a different expansion) as PolHilbertSeries.
- The default name of the indeterminate is 's'. It will not be affected by a subs command.

Examples:

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

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

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

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

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

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

```
> PolWeightedHilbertSeries([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);
```

$$1+3\lambda+5\lambda^2+6\lambda^3+6\lambda^4+6\lambda^5+6\lambda^6+O(\lambda^7)$$

```
> PolHilbertSeries(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]];
```

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

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

$$[[x^2y^2, xy], [x^3-y^6, y^3], [y^8, -y^5+x^2y], [xy^8, yx^3-y^5x], [0, -xy^7+yx^4-y^5x^2]]$$

```
> PolWeightedHilbertSeries(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 := FactorModuleBasis(var);
```

$$F := \left[y^7+y^6+y^5+y^4+y^3+y^2+y+1+xy^7+xy^6+y^5x+xy^4+xy^3+xy^2+xy+x+x^2y+x^2, \frac{x^4}{1-x}+\frac{x^3}{1-y}+\frac{x^2}{1-y}+\frac{x}{1-y}+\frac{1}{1-y}\right]$$

```
> 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:

InvolutiveBasis, PolTabVar, JanetGraph, PolHilbertSeries, PolHilbertPolynomial, PolHilbertFunction, PolCartanCharacters, FactorModuleBasis.

Involutive[PolZeroSets] - return the coefficients by which the involutive basis algorithm had to divide

Calling Sequence:

PolZeroSets()

Parameters:

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

Description:

- ***PolZeroSets*** 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 InvolutiveBasis to a module defined over a polynomial ring whose coefficient domain consists of rational functions.
- The result of ***PolZeroSets*** 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 ***PolZeroSets*** is similar to that of ZeroSets in the Janet package.

Example:

```
> with(Involutive):
```

Example 1:

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

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

```
> L := [a*x*y-a, x-b*y];
```

$$L := [a\,x\,y - a, x - b\,y]$$

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

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

```
> PolTabVar();
```

$$[x - b\,y, [x, y], x]$$
$$\left[\frac{-1 + y^2\,b}{b}, [*, y], y^2\right]$$

```
> PolZeroSets();
```

$$[a, b]$$

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

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

```
> PolZeroSets();
```

$$[a]$$

Example 2:

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

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

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

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

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

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

```
> PolTabVar();
```

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

```
> PolZeroSets();
```

[*a*]

See Also:

InvolutiveBasis, InvolutiveOptions, PolTabVar, Stats, PolInvReduce, FactorModuleBasis, PolHilbertSeries, ZeroSets

Involutive[Repres] - express polynomials in a given vector space basis of polynomials

Calling Sequence:

Repres(L,B,var)

Parameters:

L	-	list of polynomials
B	-	vector space basis as list of polynomials
var	-	list of variables (of the polynomial ring)

Description:

- ***Repres*** expresses the polynomials in **L** as linear combinations of the polynomials in **B**, if possible. More precisely, the *i*-th column of the result represents the *i*-th polynomial in **L** with respect to the vector space basis **B**, if it lies in the vector space spanned by **B**.
- The result is a matrix whose number of rows equals the number of elements in **B** and whose number of columns equals the number of elements in **L**.
- If a polynomial in **L** does not lie in the vector space which is generated by **B**, then the corresponding column in the result is the zero column.
- ***Repres*** computes an involutive basis of **B** with right hand sides, applies PolInvReduce to **L** and uses the right hand sides to express the polynomials in **L** in terms of **B**, if the remainder returned by ***PolInvReduce*** is zero.

Examples:

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

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

```
> B := [x^2+y^2, x*z+y*u, z^2+u^2];
```

$$B := [x^2+y^2,\ xz+yu,\ z^2+u^2]$$

```
> Repres(B, B, var);
```

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

```
> Repres(B, B, [x,y]);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{y^2}{z^2+u^2}+\frac{x^2}{z^2+u^2} & \frac{uy}{z^2+u^2}+\frac{zx}{z^2+u^2} & 1 \end{bmatrix}$$

```
> L := [0, x^2+y^2, 2*x*z+2*y*u, x^2+y^2+z^2+u^2];
```

$$L := [0,\ x^2+y^2,\ 2xz+2yu,\ x^2+y^2+z^2+u^2]$$

```
> Repres(L, B, var);
```

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

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

```
> Repres(L, B, var);
```

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

The input need not consist of homogeneous polynomials:

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

$$B := [x^2 + 1, y^2 + z - 1]$$

```
> L := [x^2+y^2+z, 1/7*x^2+1/7];
```

$$L := \left[x^2 + y^2 + z, \frac{x^2}{7} + \frac{1}{7}\right]$$

```
> Repres(L, B, [x,y,z]);
```

$$\begin{bmatrix} 1 & \frac{1}{7} \\ 1 & 0 \end{bmatrix}$$

See Also:

PolRepres, coeffmatrix, getbas.

Involutive[Stats] - display statistics of last application of InvolutiveBasis

Calling Sequence:

Stats()

Parameters:

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

Description:

- ***Stats*** displays statistical information about the last run of ***InvolutiveBasis***.

Examples:

```
> with(Involutive):
> 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];
```

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

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

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

```
> Stats();
```

Number of polynomials in involutive basis, 7

Use of normal form procedure, 30

Number of reductions performed, 38

Number of transfers, 0

Use of first criterion, 6

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:

InvolutiveBasis, InvolutiveOptions, PolTabVar.

Involutive[Substitute] - eliminate variables from a system of polynomial equations by substitution

Calling Sequence:

Substitute(L,var)

Parameters:

`L` - list of polynomials in **`var`**
`var` - list of variables

Description:

- ***Substitute*** tries to eliminate variables from a system of polynomial equations in **`var`** by solving some of these equations for variables that occur only linearly and substituting the resulting expressions for these variables into the remaining equations.
- ***Substitute*** applies repeatedly InvolutivePreprocess to the list **`L`** of left hand sides of the polynomial equations. As long as InvolutivePreprocess finds a variable in **`var`** that occurs only linearly in some left hand side, the resulting expression for this variable is substituted into the other left hand sides in **`L`** and InvolutivePreprocess is applied again to the resulting list of left hand sides with one variable less and so on.
- It is convenient to apply ***Substitute*** to **`L`** prior to the run of InvolutiveBasis in order to reduce the complexity of the involutive basis computation.
- **`L`** is the list of left hand sides of the polynomial equations $p_1 = 0, ..., p_n = 0$.
- **`var`** is a list specifying the variables occurring in the system of polynomial equations.
- The result of ***Substitute*** is a list with three entries. The first entry is the list of left hand sides of a system of polynomial equations which has finally been obtained by the substitution process described above, i.e. the last list of left hand sides which could not be solved linearly for any variable anymore. The second entry of the result is the list of equations which have been used to eliminate variables from the system of polynomial equations. The third entry of the result is the list of remaining variables, i.e. the complement in **`var`** of the set of variables occurring as left hand sides in the second entry of the result.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

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

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

$$\left[x = \frac{1}{3}y^2 + \frac{1}{3}z\right]$$

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

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

Example 2:

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

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

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

$$L := [xy - z, z - xy + y - 1, x^2 - y^2]$$

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

$$[[x^2 - 1], [z = xy, y = 1], [x]]$$

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

$$[z = xy, z = xy - y + 1]$$

```
> L2 := subs(z=x*y, L);
```

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

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

$$[y = 1]$$

```
> subs(y=1, L2);
```

$$[0, 0, x^2 - 1]$$

Example 3:

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

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

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

$$0$$

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

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

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

$$[x = y^2 z^2]$$

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

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

```
> InvolutiveOptions("char", 3);
```

$$2$$

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

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

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

$$[\ ]$$

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

$$[[xy, 3x - y^2 z^2, y^2 - z^2], [\ ], [x, y, z]]$$

See Also:

InvolutiveBasis, InvolutivePreprocess, InvolutiveOptions, PolTabVar, PolInvReduce, PolHilbertSeries, SyzygyModule, GroebnerBasis.

Involutive[SubmoduleBasis] - return a vector space basis for the module generated by the last computed Janet basis as a generating function

Calling Sequence:

SubmoduleBasis(var,subs)

Parameters:

`var` - list of variables (of the polynomial ring)
`subs` - (optional) equation "subs"=expression

Description:

- ***SubmoduleBasis*** returns a generating function which enumerates (the leading monomials of) a vector space basis for the submodule of the free module over the polynomial ring generated by the Janet basis of the last call of InvolutiveBasis.
- 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 ***SubmoduleBasis*** is accordingly a list of generating functions.
- The result of ***SubmoduleBasis*** can also be easily read off from the information given by PolTabVar. 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 ***InvolutiveBasis*** before.
- If an optional equation "subs"=expression is provided, then ***SubmoduleBasis*** 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(Involutive):
```

Example 1:

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

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

```
> InvolutiveBasis([x,y], var);
```

$$[y, x]$$

```
> PolTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> SubmoduleBasis(var);
```

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

```
> SubmoduleBasis(var, "subs"=t);
```

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

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20$$

$$t^{19} + O(t^{20})$$

```
> SubmoduleHilbertFunction(0);
```

$$0$$

```
> SubmoduleHilbertFunction(1);
```

$$2$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> SubmoduleHilbertPolynomial(s);
```

$$1 + s$$

Example 2:

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

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

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

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

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

$$[xy + xz, x^2 z, yx^2, z^3 x]$$

```
> PolTabVar();
```

$$[xy + xz, [*, y, z], xy]$$
$$[x^2 z, [x, *, z], x^2 z]$$
$$[yx^2, [x, y, z], yx^2]$$
$$[z^3 x, [*, *, z], z^3 x]$$

```
> SubmoduleBasis(var);
```

$$\frac{xy}{(1-y)(1-z)} + \frac{x^2 z}{(1-x)(1-z)} + \frac{yx^2}{(1-x)(1-y)(1-z)} + \frac{z^3 x}{1-z}$$

```
> SubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t^2 + 4t^3 + 9t^4 + 14t^5 + 20t^6 + 27t^7 + 35t^8 + 44t^9 + 54t^{10} + 65t^{11} + 77t^{12} + 90t^{13} + 104t^{14} + 119t^{15} + 135t^{16} + 152t^{17} + 170t^{18} + 189t^{19} + O(t^{20})$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 2
Dim(M.2) = 1
Dim(M.3) = 4
Dim(M.s) = -1+1/2*s+1/2*s^2, for s >= 4
> SubmoduleHilbertPolynomial(s);
```

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

Example 3:

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

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

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

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

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

$$[[x - y, z], [y^2, -yz - xz], [0, x^2 z]]$$

```
> PolTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -yz - xz], [*, y, z], [y^2, 1]]$$
$$[[0, x^2 z], [x, y, z], [x^2 z, 2]]$$

```
> SubmoduleBasis(var);
```

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

```
> SubmoduleBasis(var, "subs"=t);
```

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

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t+4\,t^2+9\,t^3+16\,t^4+25\,t^5+36\,t^6+49\,t^7+64\,t^8+81\,t^9+100\,t^{10}+121\,t^{11}+144\,t^{12}+169\,t^{13}+196\,t^{14}+225\,t^{15}+256\,t^{16}+289\,t^{17}+324\,t^{18}+361\,t^{19}+O(t^{20})$$

```
> SubmoduleHilbertFunction(0);
```

$$0$$

```
> SubmoduleHilbertFunction(1);
```

$$1$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> SubmoduleHilbertPolynomial(s);
```

$$s^2$$

See Also:

InvolutiveBasis, PolTabVar, SubmoduleHilbertSeries, SubmoduleHilbertFunction, SubmoduleHilbertPolynomial, SubmoduleHF, SubmoduleHP, FactorModuleBasis, PolHilbertSeries.

Involutive[SubmoduleDimension] - return the dimension of the module generated by the last computed Janet basis

Calling Sequence:

SubmoduleDimension()

Parameters:

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

Description:

- ***SubmoduleDimension*** returns the degree of the filtered Hilbert polynomial (as in SubmoduleHP) of the filtration of the factor module for which a presentation was computed by the last call of InvolutiveBasis, as explained in SubmoduleHilbertSeries.
- Note, ***SubmoduleDimension***()-1 equals the degree of SubmoduleHilbertPolynomial().

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

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

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

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

```
> PolTabVar();
```

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

```
> SubmoduleDimension();
```

$$3$$

```
> SubmoduleHP();
```

$$-\frac{25}{6}s + \frac{1}{6}s^3 + 4 + s^2$$

```
> SubmoduleHilbertPolynomial();
```

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

Example 2:

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

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

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

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

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

$$[[y, z, x], [x, y, z]]$$

```
> SubmoduleDimension();
```

$$2$$

```
> SubmoduleHP();
```

$$s + s^2$$

```
> SubmoduleHilbertPolynomial();
```

$$2s$$

See Also:

InvolutiveBasis, PolTabVar, SubmoduleHilbertSeries, SubmoduleHilbertPolynomial, SubmoduleHilbertFunction, SubmoduleHP, SubmoduleHF, PolHilbertSeries, PolDimension.

Involutive[SubmoduleHilbertFunction] - compute the graded Hilbert function for the module generated by the last computed Janet basis

Calling Sequence:

SubmoduleHilbertFunction(p)
SubmoduleHilbertFunction()

Parameters:

p - " " (empty string) or natural number

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in SubmoduleHilbertSeries. Then ***SubmoduleHilbertFunction*(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.
- SubmoduleHF, 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 ***SubmoduleHilbertFunction***.
- ***SubmoduleHilbertFunction***() returns a function expecting one parameter **p** which computes ***SubmoduleHilbertFunction*(p)**.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

```
> InvolutiveBasis([x,y], var);
```

$$[y, x]$$

```
> PolTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> SubmoduleHilbertFunction(1);
```

$$2$$

```
> SubmoduleHilbertFunction(9);
```

$$10$$

Example 2:

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

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

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

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

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

$$[xy + xz, x^2 z, yx^2, z^3 x]$$

```
> PolTabVar();
```

$$[xy+xz,[*,y,z],xy]$$
$$[x^2z,[x,*,z],x^2z]$$
$$[yx^2,[x,y,z],yx^2]$$
$$[z^3x,[*,*,z],z^3x]$$

```
> SubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t^2+4t^3+9t^4+14t^5+20t^6+27t^7+35t^8+44t^9+54t^{10}+65t^{11}+77t^{12}+90t^{13}+104t^{14}+119t^{15}+135t^{16}+152t^{17}+170t^{18}+189t^{19}+O(t^{20})$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 2
Dim(M.2) = 1
Dim(M.3) = 4
Dim(M.s) = -1+1/2*s+1/2*s^2, for s >= 4
> SubmoduleHilbertFunction(5);
```

14

Example 3:

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

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

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

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

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

$$[[x-y,z],[y^2,-yz-xz],[0,x^2z]]$$

```
> PolTabVar();
```

$$[[x-y,z],[x,y,z],[x,1]]$$
$$[[y^2,-yz-xz],[*,y,z],[y^2,1]]$$
$$[[0,x^2z],[x,y,z],[x^2z,2]]$$

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t+4t^2+9t^3+16t^4+25t^5+36t^6+49t^7+64t^8+81t^9+100t^{10}+121t^{11}+144t^{12}+169t^{13}+196t^{14}+225t^{15}+256t^{16}+289t^{17}+324t^{18}+361t^{19}+O(t^{20})$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> SubmoduleHilbertFunction(8);
```

64

See Also:

InvolutiveBasis, PolTabVar, SubmoduleBasis, SubmoduleHilbertSeries, SubmoduleHilbertPolynomial, SubmoduleHP, SubmoduleHF, FactorModuleBasis, PolHilbertSeries, PolHilbertFunction.

Involutive[SubmoduleHilbertPolynomial] - graded Hilbert polynomial for the module generated by the last computed Janet basis

Calling Sequence:

SubmoduleHilbertPolynomial(p)
SubmoduleHilbertPolynomial()

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 SubmoduleHilbertSeries. Then ***SubmoduleHilbertPolynomial*(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 InvolutiveBasis. 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 InvolutiveBasis. Note, this same information can be extracted from the command SubmoduleHilbertFunction.
- SubmoduleHP, 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 ***SubmoduleHilbertPolynomial***.
- ***SubmoduleHilbertPolynomial***() returns the graded Hilbert polynomial of the module of the leading terms of the module for which an involutive basis has been computed last by InvolutiveBasis.
- 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(Involutive):
```

Example 1:

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

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

```
> InvolutiveBasis([x,y], var);
```

$$[y, x]$$

```
> PolTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> SubmoduleHilbertPolynomial(s);
```

$$1 + s$$

```
> SubmoduleHilbertPolynomial(1);
```

$$2$$

```
> SubmoduleHilbertPolynomial(9);
```

$$10$$

Example 2:

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

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

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

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

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

$$[xy + xz, x^2 z, yx^2, z^3 x]$$

```
> PolTabVar();
```

$$[xy + xz, [*, y, z], xy]$$
$$[x^2 z, [x, *, z], x^2 z]$$
$$[yx^2, [x, y, z], yx^2]$$
$$[z^3 x, [*, *, z], z^3 x]$$

```
> SubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t^2 + 4t^3 + 9t^4 + 14t^5 + 20t^6 + 27t^7 + 35t^8 + 44t^9 + 54t^{10} + 65t^{11} + 77t^{12} + 90t^{13} + 104t^{14} + 119t^{15} + 135t^{16} + 152t^{17} + 170t^{18} + 189t^{19} + O(t^{20})$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 2
Dim(M.2) = 1
Dim(M.3) = 4
Dim(M.s) = -1+1/2*s+1/2*s^2, for s >= 4
> SubmoduleHilbertPolynomial(s);
```

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

```
> SubmoduleHilbertPolynomial(5);
```

14

Example 3:

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

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

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

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

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

$$[[x - y, z], [y^2, -yz - xz], [0, x^2 z]]$$

```
> PolTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -yz - xz], [*, y, z], [y^2, 1]]$$
$$[[0, x^2 z], [x, y, z], [x^2 z, 2]]$$

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4t^2 + 9t^3 + 16t^4 + 25t^5 + 36t^6 + 49t^7 + 64t^8 + 81t^9 + 100t^{10} + 121t^{11} + 144t^{12} + 169t^{13} + 196t^{14} + 225t^{15} + 256t^{16} + 289t^{17} + 324t^{18} + 361t^{19} + O(t^{20})$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> SubmoduleHilbertPolynomial(s);
```

$$s^2$$

```
> SubmoduleHilbertPolynomial(8);
```

64

See Also:

InvolutiveBasis, PolTabVar, SubmoduleBasis, SubmoduleHilbertSeries, SubmoduleHilbertFunction, SubmoduleHF, SubmoduleHP, FactorModuleBasis, PolHilbertSeries, PolHilbertPolynomial.

Involutive[SubmoduleHilbertSeries] - Hilbert series of the module generated by the last computed Janet basis

Calling Sequence:

SubmoduleHilbertSeries(v)

Parameters:

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

Description:

- ***SubmoduleHilbertSeries*** returns a generating function counting - according to the standard degrees - the leading monomials of the module *M* generated by the Janet basis produced by the last call of InvolutiveBasis.
- The free module of *m*-tuples over the polynomial ring is graded by the standard grading (maximal degree of the components) and the submodule of the leading monomials of *M* inherits a grading from this graded free module. Note, this submodule, and therefore also its Hilbert series, depends on the term order chosen in the call of InvolutiveBasis. ***SubmoduleHilbertSeries*** returns the Hilbert series of the submodule of leading monomials of *M*.
- 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 module of leading monomials of *M*.
- 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 ***SubmoduleHilbertSeries*** will proceed from the leading terms computed by InvolutiveBasis but then reassign the degrees 1 for the variables and 0 for the basis vectors.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

```
> InvolutiveBasis([x,y], var);
```

$$[y, x]$$

```
> PolTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> SubmoduleBasis(var);
```

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

```
> SubmoduleBasis(var, "subs"=t);
```

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

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> SubmoduleHilbertFunction(0);
```

$$0$$

```
> SubmoduleHilbertFunction(1);
```

$$2$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> SubmoduleHilbertPolynomial(s);
```

$$1+s$$

Example 2:

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

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

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

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

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

$$[xy+xz, x^2 z, yx^2, z^3 x]$$

```
> PolTabVar();
```

$$[xy+xz, [*, y, z], xy]$$
$$[x^2 z, [x, *, z], x^2 z]$$
$$[yx^2, [x, y, z], yx^2]$$
$$[z^3 x, [*, *, z], z^3 x]$$

```
> SubmoduleBasis(var);
```

$$\frac{xy}{(1-y)(1-z)}+\frac{x^2 z}{(1-x)(1-z)}+\frac{yx^2}{(1-x)(1-y)(1-z)}+\frac{z^3 x}{1-z}$$

```
> SubmoduleHilbertSeries(t);
```

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

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 2
Dim(M.2) = 1
Dim(M.3) = 4
Dim(M.s) = -1+1/2*s+1/2*s^2, for s >= 4
> SubmoduleHilbertPolynomial(s);
```

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

Example 3:

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

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

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

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

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

$$[[x-y, z], [y^2, -yz-xz], [0, x^2 z]]$$

```
> PolTabVar();
```

$$[[x-y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -yz-xz], [*, y, z], [y^2, 1]]$$
$$[[0, x^2 z], [x, y, z], [x^2 z, 2]]$$

```
> SubmoduleBasis(var);
```

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

```
> SubmoduleBasis(var, "subs"=t);
```

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

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4t^2 + 9t^3 + 16t^4 + 25t^5 + 36t^6 + 49t^7 + 64t^8 + 81t^9 + 100t^{10} + 121t^{11} + 144t^{12} + 169t^{13} + 196t^{14} + 225t^{15} + 256t^{16} + 289 t^{17} + 324t^{18} + 361t^{19} + O(t^{20})$$

```
> SubmoduleHilbertFunction(0);
```

$$0$$

```
> SubmoduleHilbertFunction(1);
```

$$1$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> SubmoduleHilbertPolynomial(s);
```

$$s^2$$

See Also:

InvolutiveBasis, PolTabVar, SubmoduleBasis, SubmoduleHilbertPolynomial, SubmoduleHilbertFunction, SubmoduleHP, SubmoduleHF, FactorModuleBasis, PolHilbertSeries.

Involutive[SubmoduleHF] - compute the filtered Hilbert function for the module generated by the last computed Janet basis

Calling Sequence:

SubmoduleHF(p)
SubmoduleHF()

Parameters:

p - " " (empty string) or natural number

Description:

- Let $\sum_{i=0}^{\infty} d_i v^i$ be the Hilbert series as discussed in SubmoduleHilbertSeries. Then ***SubmoduleHF*(p)** returns $\sum_{i=0}^{p} d_i$ for natural numbers **p** and prints the corresponding function in case **p** is the empty string.
- SubmoduleHilbertFunction, 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 ***SubmoduleHF()***.
- ***SubmoduleHF***() returns a function expecting one parameter **p** which computes ***SubmoduleHF*(p)**.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

```
> InvolutiveBasis([x,y], var);
```

$$[y, x]$$

```
> PolTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> SubmoduleHF("");
s < 1: 0
s >= 1: 3/2*s+1/2*s^2
> SubmoduleHF(1);
```

$$2$$

```
> SubmoduleHF(2);
```

$$5$$

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.s) = 1+s, for s >= 1
> SubmoduleHilbertFunction(1);
```

$$2$$

```
> SubmoduleHilbertFunction(2);
```

$$3$$

Example 2:

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

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

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

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

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

$$[xy + xz, x^2 z, yx^2, z^3 x]$$

```
> PolTabVar();
```

$$[xy + xz, [*, y, z], xy]$$
$$[x^2 z, [x, *, z], x^2 z]$$
$$[yx^2, [x, y, z], yx^2]$$
$$[z^3 x, [*, *, z], z^3 x]$$

```
> SubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t^2 + 4t^3 + 9t^4 + 14t^5 + 20t^6 + 27t^7 + 35t^8 + 44t^9 + 54t^{10} + 65t^{11} + 77t^{12} + 90t^{13} + 104t^{14} + 119t^{15} + 135t^{16} + 152t^{17} + 170t^{18} + 189t^{19} + O(t^{20})$$

```
> SubmoduleHF("");
s < 2: 0
s = 2: 1
s = 3: 5
s >= 4: 1/2*s^2-2/3*s-2+1/6*s^3
> SubmoduleHF(5);
```

28

```
> SubmoduleHF(6);
```

48

```
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 2
Dim(M.2) = 1
Dim(M.3) = 4
Dim(M.s) = -1+1/2*s+1/2*s^2, for s >= 4
> SubmoduleHilbertFunction(5);
```

14

```
> SubmoduleHilbertFunction(6);
```

20

Example 3:

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

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

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

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

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

$$[[x - y, z], [y^2, -yz - xz], [0, x^2 z]]$$

```
> PolTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -yz - xz], [*, y, z], [y^2, 1]]$$
$$[[0, x^2 z], [x, y, z], [x^2 z, 2]]$$

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4t^2 + 9t^3 + 16t^4 + 25t^5 + 36t^6 + 49t^7 + 64t^8 + 81t^9 + 100t^{10} + 121t^{11} + 144t^{12} + 169t^{13} + 196t^{14} + 225t^{15} + 256t^{16} + 289t^{17} + 324t^{18} + 361t^{19} + O(t^{20})$$

```
> SubmoduleHF("");
s < 1: 0
s = 1: 1
s = 2: 5
s >= 3: 1/6*s+1/2*s^2+1/3*s^3
> SubmoduleHF(3);
```

```
                                                    14
> SubmoduleHF(4);
                                                    30
> SubmoduleHilbertFunction("");
Dim(M.s) = 0, for s < 1
Dim(M.1) = 1
Dim(M.2) = 4
Dim(M.s) = s^2, for s >= 3
> SubmoduleHilbertFunction(3);
                                                     9
> SubmoduleHilbertFunction(4);
                                                    16
```

See Also:

InvolutiveBasis, PolTabVar, SubmoduleBasis, SubmoduleHilbertSeries, SubmoduleHilbertPolynomial, SubmoduleHilbertFunction, SubmoduleHP, FactorModuleBasis, PolHilbertSeries, PolHF.

Involutive[SubmoduleHP] - compute the filtered Hilbert polynomial for the module generated by the last computed Janet basis

Calling Sequence:

SubmoduleHP(p)
SubmoduleHP()

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 SubmoduleHilbertSeries. Then ***SubmoduleHP*(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 InvolutiveBasis, 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 SubmoduleHF.
- ***SubmoduleHP()*** returns the above polynomial with ´s´ as the default name of the indeterminate. ´s´ cannot be affected by asubs command.
- SubmoduleHilbertPolynomial, 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 ***SubmoduleHP()***.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

```
> InvolutiveBasis([x,y], var);
```

$$[y, x]$$

```
> PolTabVar();
```

$$[y, [*, y], y]$$
$$[x, [x, y], x]$$

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$2t + 3t^2 + 4t^3 + 5t^4 + 6t^5 + 7t^6 + 8t^7 + 9t^8 + 10t^9 + 11t^{10} + 12t^{11} + 13t^{12} + 14t^{13} + 15t^{14} + 16t^{15} + 17t^{16} + 18t^{17} + 19t^{18} + 20t^{19} + O(t^{20})$$

```
> SubmoduleHP(s);
```

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

```
> SubmoduleHP(1);
```

$$2$$

```
> SubmoduleHP(2);
```

$$5$$

```
> SubmoduleHilbertPolynomial(s);
```

$$1 + s$$

```
> SubmoduleHilbertPolynomial(1);
```

$$2$$

```
> SubmoduleHilbertPolynomial(2);
```

Example 2:

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

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

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

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

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

$$[xy + xz, x^2 z, yx^2, z^3 x]$$

```
> PolTabVar();
```

$$[xy + xz, [*, y, z], xy]$$
$$[x^2 z, [x, *, z], x^2 z]$$
$$[yx^2, [x, y, z], yx^2]$$
$$[z^3 x, [*, *, z], z^3 x]$$

```
> SubmoduleHilbertSeries(t);
```

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

```
> taylor(%, t=0, 20);
```

$$t^2 + 4t^3 + 9t^4 + 14t^5 + 20t^6 + 27t^7 + 35t^8 + 44t^9 + 54t^{10} + 65t^{11} + 77t^{12} + 90t^{13} + 104t^{14} + 119t^{15} + 135t^{16} + 152t^{17} + 170t^{18} + 189t^{19} + O(t^{20})$$

```
> SubmoduleHP(s);
```

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

```
> SubmoduleHP(9);
```

154

```
> SubmoduleHP(10);
```

208

```
> SubmoduleHilbertPolynomial(s);
```

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

```
> SubmoduleHilbertPolynomial(9);
```

44

```
> SubmoduleHilbertPolynomial(10);
```

54

Example 3:

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

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

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

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

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

$$[[x - y, z], [y^2, -yz - xz], [0, x^2 z]]$$

```
> PolTabVar();
```

$$[[x - y, z], [x, y, z], [x, 1]]$$
$$[[y^2, -yz - xz], [*, y, z], [y^2, 1]]$$
$$[[0, x^2 z], [x, y, z], [x^2 z, 2]]$$

```
> SubmoduleHilbertSeries("var"=t);
```

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

```
> taylor(%, t=0, 20);
```

$$t + 4t^2 + 9t^3 + 16t^4 + 25t^5 + 36t^6 + 49t^7 + 64t^8 + 81t^9 + 100t^{10} + 121t^{11} + 144t^{12} + 169t^{13} + 196t^{14} + 225t^{15} + 256t^{16} + 289t^{17} + 324t^{18} + 361t^{19} + O(t^{20})$$

```
> SubmoduleHP(s);
```

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

```
> SubmoduleHP(3);
```

14

```
> SubmoduleHP(4);
```

30

```
> SubmoduleHilbertPolynomial(s);
```

$$s^2$$

```
> SubmoduleHilbertPolynomial(3);
```

9

```
> SubmoduleHilbertPolynomial(4);
```

16

See Also:

InvolutiveBasis, PolTabVar, SubmoduleBasis, SubmoduleHilbertSeries, SubmoduleHilbertPolynomial, SubmoduleHilbertFunction, SubmoduleHF, FactorModuleBasis, PolHilbertSeries, PolHP.

Involutive[Syzygies] - return generating set for the syzygies of a finite generating set for a module over a polynomial ring

Calling Sequence:

Syzygies(L,var,ord,mode)

Parameters:

`L` - list of generators of the submodule with right hand sides
`var` - list of variables (of the polynomial ring)
`ord` - (optional) change of monomial ordering
`mode` - (optional) string "S", use ***simplify*** instead of ***expand*** internally

Description:

- ***Syzygies*** returns the list of expressions which occurred as right hand sides corresponding to zero left hand side during computation of the last call of InvolutiveBasis 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 ***InvolutiveBasis*** and must be given with right hand sides naming the generators: generator=name.
- The expressions in the output list can be interpreted as necessary conditions to be satisfied by the right hand sides for solvability of the inhomogeneous system of algebraic equations with the given left hand sides.
- In terms of modules, ***Syzygies*** constructs the syzygies among the generators **L** of the module.
- In general, the result of ***Syzygies*** is not a Janet basis for the syzygies of **L**. The command SyzygyModule returns a Janet basis for the syzygies.
- The parameters **var** and **ord** have the same meaning as in InvolutiveBasis. 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 InvolutiveBasis.
- 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(Involutive):
```

Example 1:

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

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

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

$$L := [x^2 = a, xy - x = b, y^2 = c]$$

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

$$[x = -b - yb + xc, y^2 = c]$$

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

$$[xc + y^2 b - yxc, xb + xyb - x^2 c + a, y^2 b + y^3 b - y^2 xc + xc]$$

Substituting the standard basis vectors for a and b yields the matrix of the homomorphism $R^{3x1} \to R^{3x1}$ whose image consists of the relations among the generators in L:

```
> Ls := expand(subs([a=[1,0,0], b=[0,1,0], c=[0,0,1]], [-c*x*y+b*y^2+c*x,
  a-c*x^2+b*y*x+b*x, c*x-c*x*y^2+b*y^3+b*y^2]));
```

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

```
> s := linalg[transpose](matrix(Ls));
```

$$s := \begin{bmatrix} 0 & 1 & 0 \\ y^2 & xy + x & y^3 + y^2 \\ -xy + x & -x^2 & x - y^2 x \end{bmatrix}$$

```
> simplify(linalg[multiply]([x^2 , x*y-x , y^2],s));
```

$$[0, 0, 0]$$

Note, the third column of *s* is redundant.

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]];
```

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

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

$$[[0, y] = [0, 1, 0], [x^2, 0] = [1, 0, 0]]$$

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

$$[[-1, -1, 1]]$$

Example 3:

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

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

```
> L := [x^2+y^2-1=a, x+y-1=b, x^2-y^2=c];
```

$$L := [x^2 + y^2 - 1 = a, x + y - 1 = b, x^2 - y^2 = c]$$

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

$$[1 = x\,b + (b - 2\,c)\,y - 2\,y^2\,b + 2\,x\,y\,b + c + b - 2\,a]$$

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

$$[-x^3\,b + (-b - c + 2\,a)\,x^2 + (c + b - 2\,a)\,y^2 + x\,y^2\,b + (-b + 2\,c)\,y\,x^2 + (b - 2\,c)\,y^3 + c + 2\,x^2\,y^2\,b - 2\,x^3\,y\,b + 2\,y^3\,x\,b - 2\,y^4\,b,$$
$$(-c + 2\,a)\,x + (-3\,c + 2\,a)\,y - b\,x^2 + (-3\,b + 2\,c)\,y^2 + 2\,y\,x\,c - 2\,x^2\,y\,b + 2\,y^3\,b + 2\,b + c - 2\,a, -x^3\,b + x\,b + (b - 2\,c)\,y + (-b - c + 2\,a)\,x^2$$
$$+ (-3\,b - c + 2\,a)\,y^2 + 2\,x\,y\,b - x\,y^2\,b + (-b + 2\,c)\,y\,x^2 + (-b + 2\,c)\,y^3 + b - a + c + 2\,x^2\,y^2\,b - 2\,x^3\,y\,b - 2\,y^3\,x\,b + 2\,y^4\,b,$$
$$-x\,b + (b + 2\,c - 2\,a)\,y + (2\,b - 2\,c)\,y^2 + 2\,x\,y^2\,b - 2\,y^3\,b + a - b,$$
$$(c + b - 2\,a)\,x + (-b + c)\,y + b\,x^2 + y^2\,b - 2\,y\,x\,c - 2\,x\,y^2\,b + 2\,x^2\,y\,b - b + a - c]$$

See Also:

InvolutiveBasis, InvolutiveBasisFast, AddRhs, PolTabVar, PolInvReduce, PolInvReduceFast, SyzygyModule, SyzygyModuleFast, PolResolution, PolResolutionDim, PolEulerChar.

Involutive[SyzygyModule] - return Janet basis of syzygy module of a generating set of a module over a polynomial ring

Calling Sequence:

SyzygyModule(L,var,ord,mode,rel)

Parameters:

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

- ***SyzygyModule*** returns the minimal Janet basis of the syzygy module of the generating set **`L`** of a submodule of the free module of tuples of polynomials in **`var`** with respect to a certain ordering. 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 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`**. 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`**, **`ord`** and have the same meaning as in InvolutiveBasis.
- 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 InvolutiveBasis).
- 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 InvolutiveOptions one can also choose between two implementations of ***SyzygyModule***: "Maple" and "C++".

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

$$L1 := [x^2, xy - x, y^2]$$

```
> SyzygyModule(L1, var);
```

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

Janet basis of syzygy module with respect to "position over term" ordering:

```
> SyzygyModule(L1, var, 2);
```

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

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]];
```

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

```
> SyzygyModule(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 := SyzygyModule(L3, var, "mod"=R);
```

$$S := [[1, 1, -1], [0, x, -x], [0, 0, x^3]]$$

```
> PolInvReduce([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];
```

$$L4 := [x + \omega\, y + \omega^2\, z, x\, y + y\, z + z\, x, x\, y\, z]$$

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

$$[[x\, y + y\, z + z\, x, -x - \omega\, y + z + z\, \omega, 0], [y\, z^2 + x\, z^2, -z\, (x + \omega\, y - z - z\, \omega), x + \omega\, y - z - z\, \omega], [0, x\, y\, z, -x\, y - y\, z - z\, x],$$
$$[y^2\, z^2, -y\, z\, \omega\, (z\, \omega + y), \omega\, (2\, y\, z + z^2\, \omega + y^2 + 2\, y\, z\, \omega)]]$$

```
> map(a->simplify(evalm([a] &* vector(L4))), %);
```

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

See Also:

InvolutiveBasis, InvolutiveBasisFast, AddRhs, PolTabVar, InvolutiveOptions, PolInvReduce, PolInvReduceFast, Syzygies, SyzygyModuleFast, PolResolution, PolResolutionDim, PolEulerChar.

Involutive[SyzygyModuleFast] - return Janet basis of syzygy module of a generating set of a module over a polynomial ring (C++ version)

Calling Sequence:

SyzygyModuleFast(L,var,ord,mode)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the polynomial ring)
`ord` - (optional) change of polynomial ordering (see below)
`mode` - (optional) sequence of equations specifying options for the computation

Description:

- ***SyzygyModuleFast*** computes the minimal Janet basis of the syzygy module of the generating set **`L`** of a submodule *M* of the free module of tuples of polynomials in **`var`** with respect to a certain ordering by using the C++ version of the command InvolutiveBasis (cf. InvolutiveBasisFast). 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 ***SyzygyModuleFast*** have the same meaning as in SyzygyModule.
- 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 "JB" 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 PolZeroSets. See also Example 4 below.
- Using the option "C++" of InvolutiveOptions, the command ***SyzygyModule*** is replaced by ***SyzygyModuleFast*** for the current Maple session.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

$$L1 := [x^2, xy - x, y^2]$$

```
> SyzygyModuleFast(L1, var);
```

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

Janet basis of syzygy module w.r.t. "position over term" ordering:

```
> SyzygyModuleFast(L1, var, 2);
```

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

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]];
```

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

```
> SyzygyModuleFast(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 := SyzygyModuleFast(L3, var, "mod"=R);
```

$$S := [[1, 1, -1], [0, x, -x], [0, 0, x^3]]$$

```
> PolInvReduceFast([x,0,0], S, var);
```

$$[0, 0, 0]$$

Example 4: Keeping track of denominators

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

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

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

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

```
> SyzygyModuleFast(L4, var, "denom"=true);
```

$$\left[\left[3, x + \frac{10\,y}{3}, \frac{-10}{3}\right], [0, y^2, 3x - y], [-y^2, 0, x^2 + 3xy]\right]$$

```
> PolZeroSets();
```

$$[9, 3]$$

```
> SyzygyModuleFast(L4, var, "N", "denom"=true);
```

$$[[9, 3x + 10y, -10], [0, y^2, 3x - y], [-y^2, 0, x^2 + 3xy]]$$

```
> PolZeroSets();
```

$$[9]$$

See Also:

InvolutiveBasis, InvolutiveBasisFast, InvolutiveOptions, PolTabVar, PolInvReduce, PolInvReduceFast, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution, PolResolutionDim, PolEulerChar.

Involutive[SyzygyModuleGINV] - Python/C++ version of SyzygyModule

Calling Sequence:

SyzygyModuleGINV(L,var,ord,mode,opt,rel)

Parameters:

`L` - list (or matrix) of generators of the submodule
`var` - list of variables (of the polynomial ring)
`ord` - (optional) change of polynomial ordering (see below)
`mode` - (optional) string specifying options for the computation
`opt` - (optional) sequence of equations specifying options for the computation
`rel` - (optional) equation "mod" = list of generators of a submodule

Description:

- ***SyzygyModuleGINV*** computes the minimal Janet basis of the syzygy module of the generating set **`L`** of a submodule of the free module of tuples of polynomials in **`var`** with respect to a certain ordering by using the Python/C++ version of the command InvolutiveBasis (cf. InvolutiveBasisGINV).
- All parameters for ***SyzygyModule*** are valid for ***SyzygyModuleGINV*** with the same meaning as in SyzygyModule. Additionally, possible left hand sides for equations in **`opt`** are "char", "algext", "transext", "Name", "quiet" with the same meaning as in InvolutiveBasisGINV.
- 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 corresponding process "python" instead.)
- 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 Python/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 PolZeroSets. See also Example 4 below.
- Using the option "GINV" of InvolutiveOptions, the command ***SyzygyModule*** is replaced by ***SyzygyModuleGINV*** for the current Maple session.
- For more information about **ginv**, cf. http://invo.jinr.ru and http://wwwb.math.rwth-aachen.de/Janet.

Examples:

```
> with(Involutive):
```

Example 1:

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

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

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

$$L1 := [x^2, xy - x, y^2]$$

```
> SyzygyModuleGINV(L1, var);
```

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

Janet basis of syzygy module w.r.t. "position over term" ordering:

```
> SyzygyModuleGINV(L1, var, 2);
```

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

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]];
```

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

```
> SyzygyModuleGINV(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 := SyzygyModuleGINV(L3, var, "mod"=R);
```

$$S := [[1, 1, -1], [0, x, -x], [0, 0, x^3]]$$

```
> PolInvReduceGINV([x,0,0], S, var);
```

$$[0, 0, 0]$$

Example 4: Keeping track of denominators

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

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

```
> L4 := [1/5*x^2+3*x*y, y-3*x, y^2];
```

$$L4 := \left[\frac{1}{5}x^2 + 3xy, y - 3x, y^2\right]$$

```
> SyzygyModuleGINV(L4, var, "denom"=true);
```

$$[[45, 3x + 46y, -46], [0, y^2, 3x - y], [-5y^2, 0, x^2 + 15xy]]$$

```
> PolZeroSets();
```

$$[5, 9]$$

See Also:

InvolutiveBasis, InvolutiveBasisGINV, InvolutiveOptions, PolTabVar, PolInvReduce, PolInvReduceGINV, PolHilbertSeries, Syzygies, SyzygyModule, PolResolution.