GROEBNER: A Package for Calculating Gröbner Bases, Version 3.0

H. Melenk & W. Neun

Konrad–Zuse–Zentrum
für Informationstechnik Berlin
Takustrasse 7
D–14195 Berlin–Dahlem
Germany

Email: melenk@zib.de

and

H.M. Möller

FB Mathematik
Vogelpothsweg 87
Universität Dortmund
D–44221 Dortmund
Germany

Email: moeller@math.uni–dortmund.de

Gröbner bases are a valuable tool for solving problems in connection with multivariate polynomials, such as solving systems of algebraic equations and analyzing polynomial ideals. For a definition of Gröbner bases, a survey of possible applications and further references, see [**?**]. Examples are given in [**?**], in [**?**] and also in the test file for this package.

The *groebner* package calculates Gröbner bases using the Buchberger algorithm. It can be used over a variety of different coefficient domains, and for different variable and term orderings.

The current version of the package uses parts of a previous version, written by R. Gebauer, A.C. Hearn, H. Kredel and H. M. Möller. The algorithms implemented in the current version are documented in [**?**], [**?**], [**?**] and [**?**].

The operator *saturation* has been implemented in July 2000 (Herbert Melenk).

1 Background

1.1 Variables, Domains and Polynomials

The various functions of the *groebner* package manipulate equations and/or polynomials; equations are internally transformed into polynomials by forming the difference of left-hand side and right-hand side, if equations are given.

All manipulations take place in a ring of polynomials in some variables $x1, \ldots, xn$ over a coefficient domain d:

$$d[x1, \ldots, xn],$$

where d is a field or at least a ring without zero divisors. The set of variables $x1, \ldots, xn$ can be given explicitly by the user or it is extracted automatically from the input expressions.

All REDUCE kernels can play the role of "variables" in this context; examples are

```
x y z22 sin(alpha) cos(alpha) c(1,2,3) c(1,3,2) farina4711
```

The domain d is the current REDUCE domain with those kernels adjoined that are not members of the list of variables. So the elements of d may be complicated polynomials themselves over kernels not in the list of variables; if, however, the variables are extracted automatically from the input expressions, d is identical with the current REDUCE domain. It is useful to regard kernels not being members of the list of variables as "parameters", e.g.

$$a * x + (a - b) * y * *2 \text{ with "variables" } \{x, y\} \text{ and "parameters" } a \text{ and } b \ .$$

The current version of the Buchberger algorithm has two internal modes, a field mode and a ring mode. In the starting phase the algorithm analyzes the domain type; if it recognizes d as being a ring it uses the ring mode, otherwise the field mode is needed. Normally field calculations occur only if all coefficients are numbers and if the current REDUCE domain is a field (e.g. rational numbers, modular numbers modulo a prime). In general, the

ring mode is faster. When no specific REDUCE domain is selected, the ring mode is used, even if the input formulas contain fractional coefficients: they are multiplied by their common denominators so that they become integer polynomials. Zeroes of the denominators are included in the result list.

1.2 Term Ordering

In the theory of Gröbner bases, the terms of polynomials are considered as ordered. Several order modes are available in the current package, including the basic modes:

$$lex,\ gradlex,\ revgradlex$$

All orderings are based on an ordering among the variables. For each pair of variables (a, b) an order relation must be defined, e.g. "$a \gg b$". The greater sign $\gg$ does not represent a numerical relation among the variables; it can be interpreted only in terms of formula representation: "a" will be placed in front of "b" or "a" is more complicated than "b".

The sequence of variables constitutes this order base. So the notion of

$$\{x1, x2, x3\}$$

as a list of variables at the same time means

$$x1 \gg x2 \gg x3$$

with respect to the term order.

If terms (products of powers of variables) are compared with lex, that term is chosen which has a greater variable or a higher degree if the greatest variable is the first in both. With $gradlex$ the sum of all exponents (the total degree) is compared first, and if that does not lead to a decision, the lex method is taken for the final decision. The $revgradlex$ method also compares the total degree first, but afterward it uses the lex method in the reverse direction; this is the method originally used by Buchberger.

Example 1 with $\{x, y, z\}$:

lex:

$x * y * *3$	$\gg$	$y * *48$	(heavier variable)
$x * *4 * y * *2$	$\gg$	$x * *3 * y * *10$	(higher degree in 1st variable)

gradlex:

$y * *3 * z * *4$	$\gg$	$x * *3 * y * *3$	(higher total degree)
$x * z$	$\gg$	$y * *2$	(equal total degree)

revgradlex:

$y * *3 * z * *4$	$\gg$	$x * *3 * y * *3$	(higher total degree)
$x * z$	$\ll$	$y * *2$	(equal total degree, so reverse order of lex)

The formal description of the term order modes is similar to [**?**]; this description regards only the exponents of a term, which are written as vectors of integers with 0 for exponents of a variable which does not occur:

$(e) = (e1, \ldots, en)$ representing $x1 * *e1\ x2 * *e2 \cdots xn * *en.$
$\deg(e)$ is the sum over all elements of (e)
$(e) \gg (l) \Longleftrightarrow (e) - (l) \gg (0) = (0, \ldots, 0)$

lex:

$$(e) > lex > (0) \implies e_k > 0 \text{ and } e_j = 0 \text{ for } j = 1, \ldots, k-1$$

gradlex:

$$(e) > gl > (0) \implies \deg(e) > 0 \text{ or } (e) > lex > (0)$$

revgradlex:

$$(e) > rgl > (0) \implies \deg(e) > 0 \text{ or } (e) < lex < (0)$$

Note that the *lex* ordering is identical to the standard REDUCE kernel ordering, when *korder* is set explicitly to the sequence of variables.

lex is the default term order mode in the *groebner* package.

It is beyond the scope of this manual to discuss the functionality of the term order modes. See [**?**].

The list of variables is declared as an optional parameter of the *torder* statement (see below). If this declaration is missing or if the empty list has been used, the variables are extracted from the expressions automatically

and the REDUCE system order defines their sequence; this can be influenced by setting an explicit order via the *korder* statement.

The result of a Gröbner calculation is algebraically correct only with respect to the term order mode and the variable sequence which was in effect during the calculation. This is important if several calls to the *groebner* package are done with the result of the first being the input of the second call. Therefore we recommend that you declare the variable list and the order mode explicitly. Once declared it remains valid until you enter a new *torder* statement. The operator *gvars* helps you extract the variables from a given set of polynomials, if an automatic reordering has been selected.

1.3 The Buchberger Algorithm

The Buchberger algorithm of the package is based on Gebauer/Möller [**?**]. Extensions are documented in [**?**] and [**?**].

2 Loading of the Package

The following command loads the package into REDUCE(this syntax may vary according to the implementation):

load_package groebner;

The package contains various operators, and switches for control over the reduction process. These are discussed in the following.

3 The Basic Operators

3.1 Term Ordering Mode

torder $(vl,m,[p_1,p_2,\ldots])$;

where vl is a variable list (or the empty list if no variables are declared explicitly), m is the name of a term ordering mode lex, $gradlex$, $revgradlex$ (or another implemented mode) and $[p_1,p_2,\ldots]$ are additional parameters for the term ordering mode (not needed for the basic modes).

torder sets variable set and the term ordering mode. The default mode is *lex*. The previous description is returned as a list with corresponding elements. Such a list can alternatively be passed as sole argument to *torder*.

If the variable list is empty or if the *torder* declaration is omitted, the automatic variable extraction is activated.

gvars ({*exp*1, *exp*2, ..., *expn*});

where $\{exp1, exp2, \ldots, expn\}$ is a list of expressions or equations.

gvars extracts from the expressions $\{exp1, exp2, \ldots, expn\}$ the kernels, which can play the role of variables for a Gröbner calculation. This can be used e.g. in a *torder* declaration.

3.2 *groebner*: Calculation of a Gröbner Basis

groebner $\{exp1, exp2, \ldots, expm\}$;

where $\{exp1, exp2, \ldots, expm\}$ is a list of expressions or equations.

groebner calculates the Gröbner basis of the given set of expressions with respect to the current *torder* setting.

The Gröbner basis {1} means that the ideal generated by the input polynomials is the whole polynomial ring, or equivalently, that the input polynomials have no zeroes in common.

As a side effect, the sequence of variables is stored as a REDUCE list in the shared variable

gvarslast.

This is important if the variables are reordered because of optimization: you must set them afterwards explicitly as the current variable sequence if you want to use the Gröbner basis in the sequel, e.g. for a *preduce* call. A basis has the property "Gröbner" only with respect to the variable sequences which had been active during its computation.

Example 2

```
torder({},lex)$
groebner{3*x**2*y + 2*x*y + y + 9*x**2 + 5*x - 3,
2*x**3*y - x*y - y + 6*x**3 - 2*x**2 - 3*x + 3,
x**3*y + x**2*y + 3*x**3 + 2*x**2 };
```

```
           2
{8*x - 2*y  + 5*y + 3,

     3      2
 2*y  - 3*y  - 16*y + 21}
```

This example used the default system variable ordering, which was $\{x, y\}$. With the other variable ordering, a different basis results:

```
torder({y,x},lex)$
groebner{3*x**2*y + 2*x*y + y + 9*x**2 + 5*x - 3,
2*x**3*y - x*y - y + 6*x**3 - 2*x**2 - 3*x + 3,
x**3*y + x**2*y + 3*x**3 + 2*x**2 };

           2
{2*y + 2*x  - 3*x - 6,

     3      2
 2*x  - 5*x  - 5*x}
```

Another basis yet again results with a different term ordering:

```
torder({x,y},revgradlex)$
groebner{3*x**2*y + 2*x*y + y + 9*x**2 + 5*x - 3,
2*x**3*y - x*y - y + 6*x**3 - 2*x**2 - 3*x + 3,
x**3*y + x**2*y + 3*x**3 + 2*x**2 };

   2
{2*y  - 5*y - 8*x - 3,

 y*x - y + x + 3,

   2
 2*x  + 2*y - 3*x - 6}
```

The operation of *groebner* can be controlled by the following switches:

groebopt – If set *on*, the sequence of variables is optimized with respect to execution speed; the algorithm involved is described in [**?**]; note that the final list of variables is available in *gvarslast*.

An explicitly declared dependency supersedes the variable optimization. For example

$$depend\ a,\ x,\ y;$$

guarantees that a will be placed in front of x and y. So *groebopt* can be used even in cases where elimination of variables is desired.

By default *groebopt* is off, conserving the original variable sequence.

groebfullreduction – If set off, the reduction steps during the *groebner* operation are limited to the pure head term reduction; subsequent terms are reduced otherwise.

By default *groebfullreduction* is on.

gltbasis – If set on, the leading terms of the result basis are extracted. They are collected in a basis of monomials, which is available as value of the global variable with the name *gltb*.

glterms – If $\{exp_1, \ldots, exp_m\}$ contain parameters (symbols which are not member of the variable list), the share variable *glterms* contains a list of expression which during the calculation were assumed to be nonzero. A Gröbner basis is valid only under the assumption that all these expressions do not vanish.

The following switches control the print output of *groebner*; by default all these switches are set off and nothing is printed.

groebstat – A summary of the computation is printed including the computing time, the number of intermediate h–polynomials and the counters for the hits of the criteria.

trgroeb – Includes *groebstat* and the printing of the intermediate h-polynomials.

trgroebs – Includes *trgroeb* and the printing of intermediate s–polynomials.

*trgroeb*1 – The internal pairlist is printed when modified.

3.3 *Gzerodim*?: Test of dim = 0

gzerodim!? *bas*

where *bas* is a Gröbner basis in the current setting. The result is *nil*, if *bas* is the basis of an ideal of polynomials with more than finitely many common zeros. If the ideal is zero dimensional, i. e. the polynomials of the ideal have only finitely many zeros in common, the result is an

integer k which is the number of these common zeros (counted with multiplicities).

3.4 *gdimension*, *gindependent_sets*: compute dimension and independent variables

The following operators can be used to compute the dimension and the independent variable sets of an ideal which has the Gröbner basis *bas* with arbitrary term order:

gdimension bas

gindependent_sets bas *gindependent_sets* computes the maximal left independent variable sets of the ideal, that are the variable sets which play the role of free parameters in the current ideal basis. Each set is a list which is a subset of the variable list. The result is a list of these sets. For an ideal with dimension zero the list is empty. *gdimension* computes the dimension of the ideal, which is the maximum length of the independent sets.

The switch *groebopt* plays no role in the algorithms *gdimension* and *gindependent_sets*. It is set off during the processing even if it is set *on* before. Its state is saved during the processing.

The "Kredel-Weispfenning" algorithm is used (see [**?**], extended to general ordering in [**?**].

3.5 Conversion of a Gröbner Basis

3.5.1 *glexconvert*: Conversion of an Arbitrary Gröbner Basis of a Zero Dimensional Ideal into a Lexical One

glexconvert $(\{exp, \ldots, expm\}\,[, \{var1 \ldots, varn\}]\,[, maxdeg = mx]$
$[, newvars = \{nv1, \ldots, nvk\}])$
where $\{exp1, \ldots, expm\}$ is a Gröbner basis with $\{var1, \ldots, varn\}$ as variables in the current term order mode, mx is an integer, and $\{nv1, \ldots, nvk\}$ is a subset of the basis variables. For this operator the source and target variable sets must be specified explicitly.

glexconvert converts a basis of a zero-dimensional ideal (finite number of isolated solutions) from arbitrary ordering into a basis under *lex* ordering.

During the call of *glexconvert* the original ordering of the input basis must be still active!

newvars defines the new variable sequence. If omitted, the original variable sequence is used. If only a subset of variables is specified here, the partial ideal basis is evaluated. For the calculation of a univariate polynomial, *newvars* should be a list with one element.

maxdeg is an upper limit for the degrees. The algorithm stops with an error message, if this limit is reached.

A warning occurs if the ideal is not zero dimensional.

glexconvert is an implementation of the FLGM algorithm by Faugère, Gianni, Lazard and Mora [**?**]. Often, the calculation of a Gröbner basis with a graded ordering and subsequent conversion to *lex* is faster than a direct *lex* calculation. Additionally, *glexconvert* can be used to transform a *lex* basis into one with different variable sequence, and it supports the calculation of a univariate polynomial. If the latter exists, the algorithm is even applicable in the non zero-dimensional case, if such a polynomial exists. If the polynomial does not exist, the algorithm computes until *maxdeg* has been reached.

```
torder({{w,p,z,t,s,b},gradlex)

g  :=  groebner  { f1 := 45*p + 35*s -165*b -36,
      35*p + 40*z + 25*t - 27*s, 15*w + 25*p*s +30*z -18*t
     -165*b**2, -9*w + 15*p*t  + 20*z*s,
     w*p + 2*z*t - 11*b**3, 99*w - 11*s*b +3*b**2,
     b**2 + 33/50*b + 2673/10000};

g := {60000*w + 9500*b + 3969,

    1800*p - 3100*b - 1377,

    18000*z + 24500*b + 10287,

    750*t - 1850*b + 81,

    200*s - 500*b - 9,
           2
```

```
   10000*b  + 6600*b + 2673}

glexconvert(g,{w,p,z,t,s,b},maxdeg=5,newvars={w});

            2
 100000000*w  + 2780000*w + 416421

glexconvert(g,{w,p,z,t,s,b},maxdeg=5,newvars={p});

       2
 6000*p  - 2360*p + 3051
```

3.5.2 *groebner_walk*: Conversion of a (General) Total Degree Basis into a Lex One

The algorithm *groebner_walk* convertes from an arbitrary polynomial system a *graduated* basis of the given variable sequence to a *lex* one of the same sequence. The job is done by computing a sequence of Gröbner bases of correspondig monomial ideals, lifting the original system each time. The algorithm has been described (more generally) by [**?**],[**?**],[**?**] and [**?**]. *groebner_walk* should be only called, if the direct calculation of a *lex* Gröbner base does not work. The computation of *groebner_walk* includes some overhead (e. g. the computation divides polynomials). Normally *torder* must be called before to define the variables and the variable sorting. The reordering of variables makes no sense with *groebner_walk*; so do not call *groebner_walk* with *groebopt on*!

groebner_walk g

where g is a polynomial ideal basis computed under *gradlex* or under *weighted* with a one–element, non zero weight vector with only one element, repeated for each variable. The result is a corresponding *lex* basis (if that is computable), independet of the degree of the ideal (even for non zero degree ideals). The variabe *gvarslast* is not set.

3.6 *groebnerf*: Factorizing Gröbner Bases

3.6.1 Background

If Gröbner bases are computed in order to solve systems of equations or to find the common roots of systems of polynomials, the factorizing version of the Buchberger algorithm can be used. The theoretical background is simple: if a polynomial p can be represented as a product of two (or more) polynomials, e.g. $h = f * g$, then h vanishes if and only if one of the factors vanishes. So if during the calculation of a Gröbner basis h of the above form is detected, the whole problem can be split into two (or more) disjoint branches. Each of the branches is simpler than the complete problem; this saves computing time and space. The result of this type of computation is a list of (partial) Gröbner bases; the solution set of the original problem is the union of the solutions of the partial problems, ignoring the multiplicity of an individual solution. If a branch results in a basis $\{1\}$, then there is no common zero, i.e. no additional solution for the original problem, contributed by this branch.

3.6.2 *groebnerf* Call

The syntax of *groebnerf* is the same as for *groebner*.

$$groebnerf(\{exp1, exp2, \ldots, expm\}[, \{\}, \{nz1, \ldots nzk\}]);$$

where $\{exp1, exp2, \ldots, expm\}$ is a given list of expressions or equations, and $\{nz1, \ldots nzk\}$ is an optional list of polynomials known to be non-zero.

groebnerf tries to separate polynomials into individual factors and to branch the computation in a recursive manner (factorization tree). The result is a list of partial Gröbner bases. If no factorization can be found or if all branches but one lead to the trivial basis $\{1\}$, the result has only one basis; nevertheless it is a list of lists of polynomials. If no solution is found, the result will be $\{\{1\}\}$. Multiplicities (one factor with a higher power, the same partial basis twice) are deleted as early as possible in order to speed up the calculation. The factorizing is controlled by some switches.

As a side effect, the sequence of variables is stored as a REDUCE list in the shared variable

gvarslast .

If *gltbasis* is on, a corresponding list of leading term bases is also produced and is available in the variable *gltb*.

The third parameter of *groebnerf* allows one to declare some polynomials nonzero. If any of these is found in a branch of the calculation the branch is cancelled. This can be used to save a substantial amount of computing time. The second parameter must be included as an empty list if the third parameter is to be used.

```
torder({x,y},lex)$
groebnerf { 3*x**2*y + 2*x*y + y + 9*x**2 + 5*x = 3,
            2*x**3*y - x*y - y + 6*x**3 - 2*x**2 - 3*x = -3,
             x**3*y + x**2*y + 3*x**3 + 2*x**2 \};
```

```
    {{y - 3,x},

                      2
 {2*y + 2*x - 1,2*x  - 5*x - 5}}
```

It is obvious here that the solutions of the equations can be read off immediately.

All switches from *groebner* are valid for *groebnerf* as well:

groebopt
gltbasis
groebfullreduction
groebstat
trgroeb
trgroebs
*rgroeb*1

Additional switches for *groebnerf*:

trgroebr – All intermediate partial basis are printed when detected.

By default *trgroebr* is off.

groebmonfac groebresmax groebrestriction

These variables are described in the following paragraphs.

3.6.3 Suppression of Monomial Factors

The factorization in $groebnerf$ is controlled by the following switches and variables. The variable $groebmonfac$ is connected to the handling of "monomial factors". A monomial factor is a product of variable powers occurring as a factor, e.g. $x**2*y$ in $x**3*y-2*x**2*y**2$. A monomial factor represents a solution of the type "$x = 0$ or $y = 0$" with a certain multiplicity. With $groebnerf$ the multiplicity of monomial factors is lowered to the value of the shared variable

$$groebmonfac$$

which by default is 1 (= monomial factors remain present, but their multiplicity is brought down). With

$$groebmonfac := 0$$

the monomial factors are suppressed completely.

3.6.4 Limitation on the Number of Results

The shared variable

$$groebresmax$$

controls the number of partial results. Its default value is 300. If $groebresmax$ partial results are calculated, the calculation is terminated. $groebresmax$ counts all branches, including those which are terminated (have been computed already), give no contribution to the result (partial basis 1), or which are unified in the result with other (partial) bases. So the resulting number may be much smaller. When the limit of $groeresmax$ is reached, a warning

GROEBRESMAX limit reached

is issued; this warning in any case has to be taken as a serious one. For "normal" calculations the $groebresmax$ limit is not reached. $groebresmax$ is a shared variable (with an integer value); it can be set in the algebraic mode to a different (positive integer) value.

3.6.5 Restriction of the Solution Space

In some applications only a subset of the complete solution set of a given set of equations is relevant, e.g. only nonnegative values or positive definite values for the variables. A significant amount of computing time can be saved if nonrelevant computation branches can be terminated early.

Positivity: If a polynomial has no (strictly) positive zero, then every system containing it has no nonnegative or strictly positive solution. Therefore, the Buchberger algorithm tests the coefficients of the polynomials for equal sign if requested. For example, in $13 * x + 15 * y * z$ can be zero with real nonnegative values for x, y and z only if $x = 0$ and $y = 0$ or $z = 0$; this is a sort of "factorization by restriction". A polynomial $13 * x + 15 * y * z + 20$ never can vanish with nonnegative real variable values.

Zero point: If any polynomial in an ideal has an absolute term, the ideal cannot have the origin point as a common solution.

By setting the shared variable

groebrestriction

groebnerf is informed of the type of restriction the user wants to impose on the solutions:

groebrestiction:=nonnegative;
only nonnegative real solutions are of interest

groebrestriction:=positive;
only nonnegative and nonzero solutions are of interest

groebrestriction:=zeropoint;
only solution sets which contain the point $\{0, 0, \ldots, 0\}$ are or interest.

If *groebnerf* detects a polynomial which formally conflicts with the restriction, it either splits the calculation into separate branches, or, if a violation of the restriction is determined, it cancels the actual calculation branch.

3.7 *greduce*, *preduce*: Reduction of Polynomials

3.7.1 Background

Reduction of a polynomial "p" modulo a given sets of polynomials "b" is done by the reduction algorithm incorporated in the Buchberger algorithm. Informally it can be described for polynomials over a field as follows:

loop1: % head term elimination
if there is one polynomial b in B such that the leading
term of p is a multiple of the leading term of P do
$p := p - lt(p)/lt(b) * b$ (the leading term vanishes)
do this loop as long as possible;
loop2: % elimination of subsequent terms
for each term s in p do
if there is one polynomial b in B such that s is a
multiple of the leading term of p do
$p := p - s/lt(b) * b$ (the term s vanishes)
do this loop as long as possible;

If the coefficients are taken from a ring without zero divisors we cannot divide by each possible number like in the field case. But using that in the field case, $c * p$ is reduced to $c * q$, if p is reduced to q, for arbitrary numbers c, the reduction for the ring case uses the least c which makes the (field) reduction for $c * p$ integer. The result of this reduction is returned as (ring) reduction of p eventually after removing the content, i.e. the greatest common divisor of the coefficients. The result of this type of reduction is also called a pseudo reduction of p.

3.7.2 Reduction via Gröbner Basis Calculation

$$greduce(exp, \{exp1, exp2, \ldots, expm\}]);$$

where *exp* is an expression, and $\{exp1, exp2, \ldots, expm\}$ is a list of any number of expressions or equations.

greduce first converts the list of expressions $\{exp1, \ldots, expn\}$ to a Gröbner basis, and then reduces the given expression modulo that basis. An error results if the list of expressions is inconsistent. The returned value is an

expression representing the reduced polynomial. As a side effect, *greduce* sets the variable *gvarslast* in the same manner as *groebner* does.

3.7.3 Reduction with Respect to Arbitrary Polynomials

$$preduce(exp, \{exp1, exp2, \ldots, expm\});$$

where $expm$ is an expression, and $\{exp1, exp2, \ldots, expm\}$ is a list of any number of expressions or equations.

preduce reduces the given expression modulo the set $\{exp1, \ldots, expm\}$. If this set is a Gröbner basis, the obtained reduced expression is uniquely determined. If not, then it depends on the subsequence of the single reduction steps (see 3.7.1). *preduce* does not check whether $\{exp1, exp2, \ldots, expm\}$ is a Gröbner basis in the actual order. Therefore, if the expressions are a Gröbner basis calculated earlier with a variable sequence given explicitly or modified by optimization, the proper variable sequence and term order must be activated first.

Example 3(*preduce* called with a Gröbner basis):

```
torder({x,y},lex);
gb:=groebner{3*x**2*y + 2*x*y + y + 9*x**2 + 5*x - 3,
             2*x**3*y - x*y - y + 6*x**3 - 2*x**2 - 3*x + 3,
             x**3*y + x**2*y + 3*x**3 + 2*x**2}$
preduce (5*y**2 + 2*x**2*y + 5/2*x*y + 3/2*y
           + 8*x**2 + 3/2*x - 9/2, gb);

    2
   y
```

3.7.4 *greduce_orders*: Reduction with several term orders

The shortest polynomial with different polynomial term orders is computed with the operator *greduce_orders*:

greduce_orders (exp, $\{exp1$, $exp2$, …, $expm\}$ [,$\{v_1$,v_2 … $v_n\}$]);

where exp is an expression and $\{exp1, exp2, \ldots, expm\}$ is a list of any number of expressions or equations. The list of variables $v_1, v_2 \ldots v_n$ may be omitted; if set, the variables must be a list.

The expression *exp* is reduced by *greduce* with the orders in the shared variable *gorders*, which must be a list of term orders (if set). By default it is set to

$$\{revgradlex, gradlex, lex\}$$

The shortest polynomial is the result. The order with the shortest polynomial is set to the shared variable *gorder*. A Gröbner basis of the system $\{exp1, exp2, \ldots, expm\}$ is computed for each element of $orders$. With the default setting *gorder* in most cases will be set to *revgradlex*. If the variable set is given, these variables are taken; otherwise all variables of the system $\{exp1, exp2, \ldots, expm\}$ are extracted.

The Gröbner basis computations can take some time; if interrupted, the intermediate result of the reduction is set to the shared variable $greduce\_result$, if one is done already. However, this is not nesessarily the minimal form.

If the variable *gorders* should be set to orders with a parameter, the term oder has to be replaced by a list; the first element is the term oder selected, followed by its parameter(s), e.g.

$$orders := \{\{gradlexgradlex, 2\}, \{lexgradlex, 2\}\}$$

3.7.5 Reduction Tree

In some case not only are the results produced by *greduce* and *preduce* of interest, but the reduction process is of some value too. If the switch

$$groebprot$$

is set on, *groebner*, *greduce* and *preduce* produce as a side effect a trace of their work as a REDUCE list of equations in the shared variable

$$groebprotfile.$$

Its value is a list of equations with a variable "candidate" playing the role of the object to be reduced. The polynomials are cited as "$poly1$", "$poly2$", ... If read as assignments, these equations form a program which leads from the reduction input to its result. Note that, due to the pseudo reduction with a ring as the coefficient domain, the input coefficients may be changed by global factors.

Example 4

on groebprot $
preduce $(5 * y * *2 + 2 * x * *2 * y + 5/2 * x * y + 3/2 * y + 8 * x * *2$
$+3/2 * x - 9/2, gb)$;

```
 2
y
```

groebprotfile;

```
                2          2                      2
{candidate=4*x *y + 16*x  + 5*x*y + 3*x + 10*y  + 3*y - 9,

          2
 poly1=8*x - 2*y  + 5*y + 3,

          3      2
 poly2=2*y  - 3*y  - 16*y + 21,
 candidate=2*candidate,
 candidate= - x*y*poly1 + candidate,
 candidate= - 4*x*poly1 + candidate,
 candidate=4*candidate,

               3
 candidate= - y *poly1 + candidate,
 candidate=2*candidate,

                2
 candidate= - 3*y *poly1 + candidate,
 candidate=13*y*poly1 + candidate,
 candidate=candidate + 6*poly1,

                2
 candidate= - 2*y *poly2 + candidate,
 candidate= - y*poly2 + candidate,
 candidate=candidate + 6*poly2}
```

This means

$$16(5y^2 + 2x^2y + \frac{5}{2}xy + \frac{3}{2}y + 8x^2 + \frac{3}{2}x - \frac{9}{2}) =$$

$$(-8xy - 32x - 2y^3 - 3y^2 + 13y + 6)\text{poly1}$$
$$+(-2y^2 - 2y + 6)\text{poly2} \quad + y^2.$$

3.8 Tracing with *groebnert* and *preducet*

Given a set of polynomials $\{f_1, \ldots, f_k\}$ and their Gröbner basis $\{g_1, \ldots, g_l\}$, it is well known that there are matrices of polynomials C_{ij} and D_{ji} such that

$$f_i = \sum_j C_{ij} g_j \quad \text{and } g_j = \sum_i D_{ji} f_i$$

and these relations are needed explicitly sometimes. In BUCHBERGER [**?**], such cases are described in the context of linear polynomial equations. The standard technique for computing the above formulae is to perform Gröbner reductions, keeping track of the computation in terms of the input data. In the current package such calculations are performed with (an internally hidden) cofactor technique: the user has to assign unique names to the input expressions and the arithmetic combinations are done with the expressions and with their names simultaneously. So the result is accompanied by an expression which relates it algebraically to the input values.

There are two complementary operators with this feature: *groebnert* and *preducet*; functionally they correspond to *groebner* and *produce*. However, the sets of expressions here **must be** equations with unique single identifiers on their left side and the *lhs* are interpreted as names of the expressions. Their results are sets of equations (*groebnert*) or equations (*preducet*), where a *lhs* is the computed value, while the *rhs* is its equivalent in terms of the input names.

Example 5

We calculate the Gröbner basis for an ellipse (named "$p1$") and a line (named "$p2$"); $p2$ is member of the basis immediately and so the corresponding first result element is of a very simple form; the second member is a combination of $p1$ and $p2$ as shown on the *rhs* of this equation:

```
gb1:=groebnert {p1=2*x**2+4*y**2-100,p2=2*x-y+1};

gb1 := {2*x - y + 1=p2,
           2
        9*y  - 2*y - 199= - 2*x*p2 - y*p2 + 2*p1 + p2}
```

Example 6

We want to reduce the polynomial `x**2` *wrt* the above Gröbner basis and need knowledge about the reduction formula. We therefore extract the basis polynomials from $gb1$, assign unique names to them (here $g1$, $g2$) and call *preducet*. The polynomial to be reduced here is introduced with the name Q, which then appears on the *rhs* of the result. If the name for the polynomial is omitted, its formal value is used on the right side too.

```
gb2 := for k := 1:length gb1 collect
      mkid(g,k) = lhs part(gb1,k)$
preducet (q=x**2,gb2);

 - 16*y + 208= - 18*x*g1 - 9*y*g1 + 36*q + 9*g1 - g2
```

This output means

$$x^2 = (\frac{1}{2}x + \frac{1}{4}y - \frac{1}{4})g1 + \frac{1}{36}g2 + (-\frac{4}{9}y + \frac{52}{9}).$$

Example 7

If we reduce a polynomial which is member of the ideal, we consequently get a result with *lhs* zero:

```
preducet(q=2*x**2+4*y**2-100,gb2);

0= - 2*x*g1 - y*g1 + 2*q + g1 - g2
```

This means

$$q = (x + \frac{1}{2}y - \frac{1}{2})g1 + \frac{1}{2}g2.$$

With these operators the matrices C_{ij} and D_{ji} are available implicitly, D_{ji} as side effect of *groebnert*T, c_{ij} by *calls* of *preducet* of f_i *wrt* $\{g_j\}$. The latter by definition will have the *lhs* zero and a *rhs* with linear f_i.

If $\{1\}$ is the Gröbner basis, the *groebnert* calculation gives a "proof", showing, how 1 can be computed as combination of the input polynomials.

Remark: Compared to the non-tracing algorithms, these operators are much more time consuming. So they are applicable only on small sized problems.

3.9 Gröbner Bases for Modules

Given a polynomial ring, e.g. $r = z[x_1 \cdots x_k]$ and an integer $n > 1$: the vectors with n elements of r form a *module* under vector addition (= componentwise addition) and multiplication with elements of r. For a submodule given by a finite basis a Gröbner basis can be computed, and the facilities of the *groebner* package can be used except the operators *groebnerf* and *groesolve*.

The vectors are encoded using auxiliary variables which represent the unit vectors in the module. E.g. using v_1, v_2, v_3 the module element $[x_1^2, 0, x_1 - x_2]$ is represented as $x_1^2 v_1 + x_1 v_3 - x_2 v_3$. The use of v_1, v_2, v_3 as unit vectors is set up by assigning the set of auxiliary variables to the share variable *gmodule*, e.g.

```
gmodule := {v1,v2,v3};
```

After this declaration all monomials built from these variables are considered as an algebraically independent basis of a vector space. However, you had best use them only linearly. Once *gmodule* has been set, the auxiliary variables automatically will be added to the end of each variable list (if they are not yet member there). Example:

```
torder({x,y,v1,v2,v3},lex)$
gmodule := {v1,v2,v3}$
g:=groebner{x^2*v1 + y*v2,x*y*v1 - v3,2y*v1 + y*v3};

      2
g := {x *v1 + y*v2,

              2
     x*v3 + y *v2,

      3
     y *v2 - 2*v3,

     2*y*v1 + y*v3}

preduce((x+y)^3*v1,g);

           1   3         2
```

```
 - x*y*v2 - ---*y *v3 - 3*y *v2 + 3*y*v3
             2
```

In many cases a total degree oriented term order will be adequate for computations in modules, e.g. for all cases where the submodule membership is investigated. However, arranging the auxiliary variables in an elimination oriented term order can give interesting results. E.g.

```
   p1:=(x-1)*(x^2-x+3)$  p2:=(x-1)*(x^2+x-5)$
   gmodule := {v1,v2,v3};
   torder({v1,x,v2,v3},lex)$
   gb:=groebner {p1*v1+v2,p2*v1+v3};

gb := {30*v1*x - 30*v1 + x*v2 - x*v3 + 5*v2 - 3*v3,

        2       2
       x *v2 - x *v3 + x*v2 + x*v3 - 5*v2 - 3*v3}

   g:=coeffn(first gb,v1,1);

g := 30*(x - 1)

   c1:=coeffn(first gb,v2,1);

c1 := x + 5

   c2:=coeffn(first gb,v3,1);

c2 :=  - x - 3

   c1*p1 + c2*p2;

30*(x - 1)
```

Here two polynomials are entered as vectors $[p_1, 1, 0]$ and $[p_2, 0, 1]$. Using a term ordering such that the first dimension ranges highest and the other components lowest, a classical cofactor computation is executed just as in the extended Euclidean algorithm. Consequently the leading polynomial in

the resulting basis shows the greatest common divisor of p_1 and p_2, found as a coefficient of v_1 while the coefficients of v_2 and v_3 are the cofactors c_1 and c_2 of the polynomials p_1 and p_2 with the relation $gcd(p_1, p_2) = c_1p_1 + c_2p_2$.

3.10 Additional Orderings

Besides the basic orderings, there are ordering options that are used for special purposes.

3.10.1 Separating the Variables into Groups

It is often desirable to separate variables and formal parameters in a system of polynomials. This can be done with a *lex* Gröbner basis. That however may be hard to compute as it does more separation than necessary. The following orderings group the variables into two (or more) sets, where inside each set a classical ordering acts, while the sets are handled via their total degrees, which are compared in elimination style. So the Gröbner basis will eliminate the members of the first set, if algebraically possible. *torder* here gets an additional parameter which describe the grouping

torder (vl,gradlexgradlex, n)
torder (vl,gradlexrevgradlex,n)
torder (vl,lexgradlex, n)
torder (vl,lexrevgradlex, n)

Here the integer n is the number of variables in the first group and the names combine the local ordering for the first and second group, e.g.

lexgradlex, 3 for $\{x_1, x_2, x_3, x_4, x_5\}$:
$x_1^{i_1} \dots x_5^{i_5} \gg x_1^{j_1} \dots x_5^{j_5}$
if $(i_1, i_2, i_3) \gg_{lex} (j_1, j_2, j_3)$
or $(i_1, i_2, i_3) = (j_1, j_2, j_3)$
and $(i_4, i_5) \gg_{gradlex} (j_4, j_5)$

Note that in the second place there is no *lex* ordering available; that would not make sense.

3.10.2 Weighted Ordering

The statement

$$torder \quad (vl,\text{weighted, } \{n_1, n_2, n_3 \ldots\}) \ ;$$

establishes a graduated ordering, where the exponents are first multiplied by the given weights. If there are less weight values than variables, the weight 1 is added automatically. If the weighted degree calculation is not decidable, a *lex* comparison follows.

3.10.3 Graded Ordering

The statement

$$torder \quad (vl,\text{graded, } \{n_1, n_2, n_3 \ldots\},order_2) \ ;$$

establishes a graduated ordering, where the exponents are first multiplied by the given weights. If there are less weight values than variables, the weight 1 is added automatically. If the weighted degree calculation is not decidable, the term order $order_2$ specified in the following argument(s) is used. The ordering *graded* is designed primarily for use with the operator *dd_groebner*.

3.10.4 Matrix Ordering

The statement

$$torder \quad (vl,\text{matrix, } m) \ ;$$

where m is a matrix with integer elements and row length which corresponds to the variable number. The exponents of each monomial form a vector; two monomials are compared by multiplying their exponent vectors first with m and comparing the resulting vector lexicographically. E.g. the unit matrix establishes the classical *lex* term order mode, a matrix with a first row of ones followed by the rows of a unit matrix corresponds to the *gradlex* ordering.

The matrix m must have at least as many rows as columns; a non–square matrix contains redundant rows. The matrix must have full rank, and the top non–zero element of each column must be positive.

The generality of the matrix based term order has its price: the computing time spent in the term sorting is significantly higher than with the specialized term orders. To overcome this problem, you can compile a matrix term order ; the compilation reduces the computing time overhead significantly. If you

set the switch *comp* on, any new order matrix is compiled when any operator of the *groebner* package accesses it for the first time. Alternatively you can compile a matrix explicitly

```
torder_compile(<n>,<m>);
```

where $< n >$ is a name (an identifier) and $< m >$ is a term order matrix. *torder_compile* transforms the matrix into a LISP program, which is compiled by the LISP compiler when *comp* is on or when you generate a fast loadable module. Later you can activate the new term order by using the name $< n >$ in a *torder* statement as term ordering mode.

3.11 Gröbner Bases for Graded Homogeneous Systems

For a homogeneous system of polynomials under a term order *graded*, *gradlex*, *revgradlex* or *weighted* a Gröbner Base can be computed with limiting the grade of the intermediate s–polynomials:

dd_groebner ($d1$,$d2$,$\{p_1, p_2, \ldots\}$);

where $d1$ is a non–negative integer and $d2$ is an integer $> d1$ or "infinity". A pair of polynomials is considered only if the grade of the lcm of their head terms is between $d1$ and $d2$. See [**?**] for the mathematical background. For the term orders *graded* or *weighted* the (first) weight vector is used for the grade computation. Otherwise the total degree of a term is used.

4 Ideal Decomposition & Equation System Solving

Based on the elementary Gröbner operations, the *groebner* package offers additional operators, which allow the decomposition of an ideal or of a system of equations down to the individual solutions.

4.1 Solutions Based on Lex Type Gröbner Bases

4.1.1 groesolve: Solution of a Set of Polynomial Equations

The *groesolve* operator incorporates a macro algorithm; lexical Gröbner bases are computed by $groebnerf$ and decomposed into simpler ones by ideal decomposition techniques; if algebraically possible, the problem is reduced to univariate polynomials which are solved by *solve*; if *rounded* is on, numerical approximations are computed for the roots of the univariate polynomials.

$$groesolve(\{exp1, exp2, \ldots, expm\}[, \{var1, var2, \ldots, varn\}]);$$

where $\{exp1, exp2, \ldots, expm\}$ is a list of any number of expressions or equations, $\{var1, var2, \ldots, varn\}$ is an optional list of variables.

The result is a set of subsets. The subsets contain the solutions of the polynomial equations. If there are only finitely many solutions, then each subset is a set of expressions of triangular type $\{exp1, exp2, \ldots, expn\}$, where $exp1$ depends only on $var1$, $exp2$ depends only on $var1$ and $var2$ etc. until $expn$ which depends on $var1, \ldots, varn$. This allows a successive determination of the solution components. If there are infinitely many solutions, some subsets consist in less than n expressions. By considering some of the variables as "free parameters", these subsets are usually again of triangular type.

Example 8(Intersections of a line with a circle):

$$groesolve(\{x**2 - y**2 - a, p*x + q*y + s\}, \{x, y\});$$

```
                2      2    2             2    2
{{x=(sqrt( - a*p  + a*q  + s )*q - p*s)/(p  - q ),
                  2      2    2             2    2
  y= - (sqrt( - a*p  + a*q  + s )*p - q*s)/(p  - q )},
                  2      2    2             2    2
 {x= - (sqrt( - a*p  + a*q  + s )*q + p*s)/(p  - q ),
                2      2    2             2    2
  y=(sqrt( - a*p  + a*q  + s )*p + q*s)/(p  - q )}}
```

If the system is zero–dimensional (has a number of isolated solutions), the algorithm described in [**?**] is used, if the decomposition leaves a polynomial with mixed leading term. Hillebrand has written the article and Möller was the tutor of this job.

The reordering of the *groesolve* variables is controlled by the REDUCE switch *varopt*. If *varopt* is *on* (which is the default of *varopt*), the variable sequence is optimized (the variables are reordered). If *varopt* is off, the given variable sequence is taken (if no variables are given, the order of the REDUCE system is taken instead). In general, the reordering of the variables makes the Gröbner basis computation significantly faster. A variable dependency, declare by one (or several) *depend* statements, is regarded (if *varopt* is *on*). The switch *groebopt* has no meaning for *groesolve*; it is stored during its processing.

4.1.2 *groepostproc*: Postprocessing of a Gröbner Basis

In many cases, it is difficult to do the general Gröbner processing. If a Gröbner basis with a *lex* ordering is calculated already (e.g., by very individual parameter settings), the solutions can be derived from it by a call to *groepostproc*. *groesolve* is functionally equivalent to a call to $groebnerf$ and subsequent calls to *groepostproc* for each partial basis.

$$groepostproc(\{exp1, exp2, \ldots, expm\}[, \{var1, var2, \ldots, varn\}]);$$

where $\{exp1, exp2, \ldots, expm\}$ is a list of any number of expressions, $\{var1, var2, \ldots, varn\}$ is an optional list of variables. The expressions must be a *lex* Gröbner basis with the given variables; the ordering must be still active.

The result is the same as with *groesolve*.

```
groepostproc({x3**2 + x3 + x2 - 1,
              x2*x3 + x1*x3 + x3 + x1*x2 + x1 + 2,
              x2**2 + 2*x2 - 1,
              x1**2 - 2},{x3,x2,x1});

{{x3= - sqrt(2),

  x2=sqrt(2) - 1,

  x1=sqrt(2)},

 {x3=sqrt(2),
```

```
 x2= - (sqrt(2) + 1),

 x1= - sqrt(2)},

     sqrt(4*sqrt(2) + 9) - 1
{x3=-------------------------,
                2

 x2= - (sqrt(2) + 1),

 x1=sqrt(2)},

      - (sqrt(4*sqrt(2) + 9) + 1)
{x3=-----------------------------,
                  2

 x2= - (sqrt(2) + 1),

 x1=sqrt(2)},

     sqrt( - 4*sqrt(2) + 9) - 1
{x3=--------------------------,
                2

 x2=sqrt(2) - 1,

 x1= - sqrt(2)},

      - (sqrt( - 4*sqrt(2) + 9) + 1)
{x3=---------------------------------,
                   2

 x2=sqrt(2) - 1,

 x1= - sqrt(2)}}
```

4.1.3 Idealquotient: Quotient of an Ideal and an Expression

Let i be an ideal and f be a polynomial in the same variables. Then the algebraic quotient is defined by

$$i : f = \{p \mid p * f \text{ member of } i\} .$$

The ideal quotient $i : f$ contains i and is obviously part of the whole polynomial ring, i.e. contained in $\{1\}$. The case $i : f = \{1\}$ is equivalent to f being a member of i. The other extremal case, $i : f = i$, occurs, when f does not vanish at any general zero of i. The explanation of the notion "general zero" introduced by van der Waerden, however, is beyond the aim of this manual. The operation of *groesolve*/*groepostproc* is based on nested ideal quotient calculations.

If i is given by a basis and f is given as an expression, the quotient can be calculated by

$$idealquotient(\{exp1, \ldots, expm\}, exp);$$

where $\{exp1, exp2, \ldots, expm\}$ is a list of any number of expressions or equations, *exp* is a single expression or equation.

idealquotient calculates the algebraic quotient of the ideal i with the basis $\{exp1, exp2, \ldots, expm\}$ and *exp* with respect to the variables given or extracted. $\{exp1, exp2, \ldots, expm\}$ is not necessarily a Gröbner basis. The result is the Gröbner basis of the quotient.

4.1.4 Saturation: Saturation of an Ideal and an Expression

The *saturation* computes the quotient on an ideal and an arbitrary power of an expression $exp * *n$ with arbitrary n. The call is

$$saturation(\{exp1, \ldots, expm\}, exp);$$

where $\{exp1, exp2, \ldots, expm\}$ is a list of any number of expressions or equations, *exp* is a single expression or equation.

saturation calls *idealquotient* several times, until the result is stable, and returns it.

4.2 Operators for Gröbner Bases in all Term Orderings

In some cases where no Gröbner basis with lexical ordering can be calculated, a calculation with a total degree ordering is still possible. Then the Hilbert polynomial gives information about the dimension of the solutions space and for finite sets of solutions univariate polynomials can be calculated. The solutions of the equation system then is contained in the cross product of all solutions of all univariate polynomials.

4.2.1 Hilbertpolynomial: Hilbert Polynomial of an Ideal

This algorithm was contributed by Joachim Hollman, Royal Institute of Technology, Stockholm (private communication).

$$hilbertpolynomial(\{exp1, \ldots, expm\}) ;$$

where $\{exp1, \ldots, expm\}$ is a list of any number of expressions or equations.

hilertpolynomial calculates the Hilbert polynomial of the ideal with basis $\{exp1, \ldots, expm\}$ with respect to the variables given or extracted provided the given term ordering is compatible with the degree, such as the *gradlex*- or *revgradlex*-ordering. The term ordering of the basis must be active and $\{exp1, \ldots, expm\}$ should be a Gröbner basis with respect to this ordering. The Hilbert polynomial gives information about the cardinality of solutions of the system $\{exp1, \ldots, expm\}$: if the Hilbert polynomial is an integer, the system has only a discrete set of solutions and the polynomial is identical with the number of solutions counted with their multiplicities. Otherwise the degree of the Hilbert polynomial is the dimension of the solution space.

If the Hilbert polynomial is not a constant, it is constructed with the variable "x" regardless of whether x is member of $\{var1, \ldots, varn\}$ or not. The value of this polynomial at sufficiently large numbers "x" is the difference of the dimension of the linear vector space of all polynomials of degree $\leq x$ minus the dimension of the subspace of all polynomials of degree $\leq x$ which belong also to the ideal.

Remark: The number of zeros in an ideal and the Hilbert polynomial depend only on the leading terms of the Gröbner basis. So if a subsequent Hilbert calculation is planned, the Gröbner calculation should be performed

with *on gltbasis* and the value of *gltb* (or its elements in a *groebnerf* context) should be given to *hilbertpolynomial*. In this manner, a lot of computing time can be saved in the case of large bases.

5 Calculations "by Hand"

The following operators support explicit calculations with polynomials in a distributive representation at the REDUCE top level. So they allow one to do Gröbner type evaluations stepwise by separate calls. Note that the normal REDUCE arithmetic can be used for arithmetic combinations of monomials and polynomials.

5.1 Representing Polynomials in Distributive Form

gsortp;

where p is a polynomial or a list of polynomials.

If p is a single polynomial, the result is a reordered version of p in the distributive representation according to the variables and the current term order mode; if p is a list, its members are converted into distributive representation and the result is the list sorted by the term ordering of the leading terms; zero polynomials are eliminated from the result.

```
torder({alpha,beta,gamma},lex);

dip := gsort(gamma*(alpha-1)**2*(beta+1)**2);
```

```
            2     2                2
dip := alpha *beta *gamma + 2*alpha *beta*gamma

       2                     2
+ alpha *gamma - 2*alpha*beta *gamma - 4*alpha*beta*gamma

                       2
 - 2*alpha*gamma + beta *gamma + 2*beta*gamma + gamma
```

5.2 Splitting of a Polynomial into Leading Term and Reductum

$$gsplitp;$$

where p is a polynomial.

gsplit converts the polynomial p into distributive representation and splits it into leading monomial and reductum. The result is a list with two elements, the leading monomial and the reductum.

```
gslit dip;

       2     2
 {alpha *beta *gamma,

        2                   2                     2
 2*alpha *beta*gamma + alpha *gamma - 2*alpha*beta *gamma

                     2
 - 4*alpha*beta*gamma - 2*alpha*gamma + beta *gamma


 + 2*beta*gamma + gamma}
```

5.3 Calculation of Buchberger's S-polynomial

$$gspoly(p1, p2);$$

where $p1$ and $p2$ are polynomials.

gspoly calculates the s-polynomial from $p1$ and $p2$;

Example for a complete calculation (taken from DAVENPORT ET AL. [**?**]):

```
torder({x,y,z},lex)$
g1  :=  x**3*y*z - x*z**2;
g2  :=  x*y**2*z - x*y*z;
g3  :=  x**2*y**2 - z;$

% first S-polynomial

g4  :=  gspoly(g2,g3);$

        2        2
 g4 := x *y*z - z

 % next S-polynomial

 p :=  gspoly(g2,g4); $

       2          2
 p := x *y*z - y*z

 % and reducing, here only by g4

 g5  :=  preduce(p,{g4});

             2    2
 g5 :=  - y*z  + z

 % last S-polynomial}

 g6  :=  gspoly(g4,g5);

        2  2    3
 g6 := x *z  - z

 % and the final basis sorted descending

 gsort{g2,g3,g4,g5,g6};

   2  2
 {x *y  - z,
```

```
 2       2
x *y*z - z ,

 2  2    3
x *z  - z ,

   2
x*y *z - x*y*z,

     2    2
 - y*z  + z }
```

References