div(*fv*)

Division algorithm, see [CLO] p64.

fv array of polynomials

return qv, r such that self = sum(fv[i]*qv[i]) + r

All polynomials are required not to be Laurent polynomials.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y = ring('x, y', ZZ)
>>> f = x**3
>>> f0 = x - y**2
>>> f1 = x - y
>>> qv, r = f.div((f0, f1))
>>> qv[0]
x**2 + x*y**2 + y**4
>>> qv[1]
0
>>> r
y**6
```

imul_num(*c*)

multiply inplace the polynomial p by an element in the coefficient ring, provided p is not one of the generators; else multiply not inplace

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y = ring('x, y', ZZ)
>>> p = x + y**2
>>> p1 = p.imul_num(3)
>>> p1
3*x + 3*y**2
>>> p1 is p
True
>>> p = x
>>> p1 = p.imul_num(3)
>>> p1
3*x
>>> p1 is p
False
```

itercoeffs()

Iterator over coefficients of a polynomial.

itermonoms()

Iterator over monomials of a polynomial.

iterterms()

Iterator over terms of a polynomial.

leading_expv()

Leading monomial tuple according to the monomial ordering.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y, z = ring('x, y, z', ZZ)
>>> p = x**4 + x**3*y + x**2*z**2 + z**7
>>> p.leading_expv()
(4, 0, 0)
```

leading_monom()

Leading monomial as a polynomial element.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y = ring('x, y', ZZ)
>>> (3*x*y + y**2).leading_monom()
x*y
```

leading_term()

Leading term as a polynomial element.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y = ring('x, y', ZZ)
>>> (3*x*y + y**2).leading_term()
3*x*y
```

listcoeffs()

Unordered list of polynomial coefficients.

listmonoms()

Unordered list of polynomial monomials.

listterms()

Unordered list of polynomial terms.

monic()

Divides all coefficients by the leading coefficient.

monoms(*order=None*)

Ordered list of polynomial monomials.

Parameters

order : *MonomialOrder* (page 2431) or coercible, optional

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.orderings import lex, grlex
```

```
>>> _, x, y = ring("x, y", ZZ, lex)
>>> f = x*y**7 + 2*x**2*y**3
```

```
>>> f.monoms()
[(2, 3), (1, 7)]
>>> f.monoms(grlex)
[(1, 7), (2, 3)]
```

primitive()

Returns content and a primitive polynomial.

square()

square of a polynomial

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y = ring('x, y', ZZ)
>>> p = x + y**2
>>> p.square()
x**2 + 2*x*y**2 + y**4
```

strip_zero()

Eliminate monomials with zero coefficient.

tail_degree(*x=None*)

The tail degree in x or the main variable.

Note that the degree of 0 is negative infinity (the SymPy object -oo)

tail_degrees()

A tuple containing tail degrees in all variables.

Note that the degree of 0 is negative infinity (the SymPy object -oo)

terms(*order=None*)

Ordered list of polynomial terms.

Parameters

order : *MonomialOrder* (page 2431) or coercible, optional

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.orderings import lex, grlex
```

```
>>> _, x, y = ring("x, y", ZZ, lex)
>>> f = x*y**7 + 2*x**2*y**3
```

```
>>> f.terms()
[((2, 3), 2), ((1, 7), 1)]
>>> f.terms(grlex)
[((1, 7), 1), ((2, 3), 2)]
```


Sparse rational functions

Sparse polynomials are represented as dictionaries.

sympy.polys.fields.**field**(*symbols*, *domain*, *order=LexOrder()*)

Construct new rational function field returning (field, x1, ..., xn).

sympy.polys.fields.**xfield**(*symbols*, *domain*, *order=LexOrder()*)

Construct new rational function field returning (field, (x1, ..., xn)).

sympy.polys.fields.**vfield**(*symbols*, *domain*, *order=LexOrder()*)

Construct new rational function field and inject generators into global namespace.

sympy.polys.fields.**sfield**(*exprs*, **symbols*, ***options*)

Construct a field deriving generators and domain from options and input expressions.

Parameters

exprs : py:class: *.Expr* or sequence of *Expr* (page 947) (sympifiable)

symbols : sequence of *Symbol* (page 976)/*Expr* (page 947)

options : keyword arguments understood by *Options* (page 2642)

Examples

```
>>> from sympy import exp, log, symbols, sfield
```

```
>>> x = symbols("x")
>>> K, f = sfield((x*log(x) + 4*x**2)*exp(1/x + log(x)/3)/x**2)
>>> K
Rational function field in x, exp(1/x), log(x), x**(1/3) over ZZ with␣
↪lex order
>>> f
(4*x**2*(exp(1/x)) + x*(exp(1/x))*(log(x)))/((x**(1/3))**5)
```

class sympy.polys.fields.**FracField**(*symbols*, *domain*, *order=LexOrder()*)

Multivariate distributed rational function field.

class sympy.polys.fields.**FracElement**(*numer*, *denom=None*)

Element of multivariate distributed rational function field.

diff(*x*)

Computes partial derivative in `x`.

Examples

```
>>> from sympy.polys.fields import field
>>> from sympy.polys.domains import ZZ
```

```
>>> _, x, y, z = field("x,y,z", ZZ)
>>> ((x**2 + y)/(z + 1)).diff(x)
2*x/(z + 1)
```

Dense polynomials

class sympy.polys.polyclasses.**DMP**(*rep*, *dom*, *lev=None*, *ring=None*)

Dense Multivariate Polynomials over K.

LC()

Returns the leading coefficient of `f`.

TC()

Returns the trailing coefficient of `f`.

abs()

Make all coefficients in `f` positive.

add(*g*)

Add two multivariate polynomials `f` and `g`.

add_ground(*c*)

Add an element of the ground domain to `f`.

all_coeffs()
Returns all coefficients from `f`.

all_monoms()
Returns all monomials from `f`.

all_terms()
Returns all terms from a `f`.

cancel(*g*, *include=True*)
Cancel common factors in a rational function `f/g`.

cauchy_lower_bound()
Computes the Cauchy lower bound on the nonzero roots of `f`.

cauchy_upper_bound()
Computes the Cauchy upper bound on the roots of `f`.

clear_denoms()
Clear denominators, but keep the ground domain.

coeffs(*order=None*)
Returns all non-zero coefficients from `f` in lex order.

cofactors(*g*)
Returns GCD of `f` and `g` and their cofactors.

compose(*g*)
Computes functional composition of `f` and `g`.

content()
Returns GCD of polynomial coefficients.

convert(*dom*)
Convert the ground domain of `f`.

count_complex_roots(*inf=None*, *sup=None*)
Return the number of complex roots of `f` in `[inf, sup]`.

count_real_roots(*inf=None*, *sup=None*)
Return the number of real roots of `f` in `[inf, sup]`.

decompose()
Computes functional decomposition of `f`.

deflate()
Reduce degree of f by mapping x_i^m to y_i.

degree(*j=0*)
Returns the leading degree of `f` in `x_j`.

degree_list()
Returns a list of degrees of `f`.

diff(*m=1*, *j=0*)
Computes the `m`-th order derivative of `f` in `x_j`.

discriminant()
Computes discriminant of f.

div(*g*)
Polynomial division with remainder of f and g.

eject(*dom*, *front=False*)
Eject selected generators into the ground domain.

eval(*a*, *j=0*)
Evaluates f at the given point a in x_j.

exclude()
Remove useless generators from f.

Returns the removed generators and the new excluded f.

Examples

```
>>> from sympy.polys.polyclasses import DMP
>>> from sympy.polys.domains import ZZ
```

```
>>> DMP([[[ZZ(1)]], [[ZZ(1)], [ZZ(2)]]], ZZ).exclude()
([2], DMP([[1], [1, 2]], ZZ, None))
```

exquo(*g*)
Computes polynomial exact quotient of f and g.

exquo_ground(*c*)
Exact quotient of f by a an element of the ground domain.

factor_list()
Returns a list of irreducible factors of f.

factor_list_include()
Returns a list of irreducible factors of f.

classmethod from_dict(*rep*, *lev*, *dom*)
Construct and instance of cls from a dict representation.

classmethod from_list(*rep*, *lev*, *dom*)
Create an instance of cls given a list of native coefficients.

classmethod from_sympy_list(*rep*, *lev*, *dom*)
Create an instance of cls given a list of SymPy coefficients.

gcd(*g*)
Returns polynomial GCD of f and g.

gcdex(*g*)
Extended Euclidean algorithm, if univariate.

gff_list()
Computes greatest factorial factorization of f.

half_gcdex(*g*)
Half extended Euclidean algorithm, if univariate.

homogeneous_order()
Returns the homogeneous order of `f`.

homogenize(*s*)
Return homogeneous polynomial of `f`

inject(*front=False*)
Inject ground domain generators into `f`.

integrate(*m=1*, *j=0*)
Computes the `m`-th order indefinite integral of `f` in `x_j`.

intervals(*all=False*, *eps=None*, *inf=None*, *sup=None*, *fast=False*, *sqf=False*)
Compute isolating intervals for roots of `f`.

invert(*g*)
Invert `f` modulo `g`, if possible.

property is_cyclotomic
Returns `True` if `f` is a cyclotomic polynomial.

property is_ground
Returns `True` if `f` is an element of the ground domain.

property is_homogeneous
Returns `True` if `f` is a homogeneous polynomial.

property is_irreducible
Returns `True` if `f` has no factors over its domain.

property is_linear
Returns `True` if `f` is linear in all its variables.

property is_monic
Returns `True` if the leading coefficient of `f` is one.

property is_monomial
Returns `True` if `f` is zero or has only one term.

property is_one
Returns `True` if `f` is a unit polynomial.

property is_primitive
Returns `True` if the GCD of the coefficients of `f` is one.

property is_quadratic
Returns `True` if `f` is quadratic in all its variables.

property is_sqf
Returns `True` if `f` is a square-free polynomial.

property is_zero
Returns `True` if `f` is a zero polynomial.

l1_norm()
Returns l1 norm of `f`.

l2_norm_squared()
Return squared l2 norm of f.

lcm(*g*)
Returns polynomial LCM of f and g.

lift()
Convert algebraic coefficients to rationals.

max_norm()
Returns maximum norm of f.

mignotte_sep_bound_squared()
Computes the squared Mignotte bound on root separations of f.

monic()
Divides all coefficients by LC(f).

monoms(*order=None*)
Returns all non-zero monomials from f in lex order.

mul(*g*)
Multiply two multivariate polynomials f and g.

mul_ground(*c*)
Multiply f by a an element of the ground domain.

neg()
Negate all coefficients in f.

norm()
Computes Norm(f).

nth(**N*)
Returns the n-th coefficient of f.

pdiv(*g*)
Polynomial pseudo-division of f and g.

per(*rep, dom=None, kill=False, ring=None*)
Create a DMP out of the given representation.

permute(*P*)
Returns a polynomial in $K[x_{P(1)}, ..., x_{P(n)}]$.

Examples

```
>>> from sympy.polys.polyclasses import DMP
>>> from sympy.polys.domains import ZZ
```

```
>>> DMP([[[ZZ(2)], [ZZ(1), ZZ(0)]], [[]]], ZZ).permute([1, 0, 2])
DMP([[[2], []], [[1, 0], []]], ZZ, None)
```

```
>>> DMP([[[ZZ(2)], [ZZ(1), ZZ(0)]], [[]]], ZZ).permute([1, 2, 0])
DMP([[[1], []], [[2, 0], []]], ZZ, None)
```

pexquo(*g*)
Polynomial exact pseudo-quotient of `f` and `g`.

pow(*n*)
Raise `f` to a non-negative power `n`.

pquo(*g*)
Polynomial pseudo-quotient of `f` and `g`.

prem(*g*)
Polynomial pseudo-remainder of `f` and `g`.

primitive()
Returns content and a primitive form of `f`.

quo(*g*)
Computes polynomial quotient of `f` and `g`.

quo_ground(*c*)
Quotient of `f` by a an element of the ground domain.

refine_root(*s*, *t*, *eps=None*, *steps=None*, *fast=False*)
Refine an isolating interval to the given precision.

`eps` should be a rational number.

rem(*g*)
Computes polynomial remainder of `f` and `g`.

resultant(*g*, *includePRS=False*)
Computes resultant of `f` and `g` via PRS.

revert(*n*)
Compute `f**(-1)` mod `x**n`.

shift(*a*)
Efficiently compute Taylor shift `f(x + a)`.

slice(*m*, *n*, *j=0*)
Take a continuous subsequence of terms of `f`.

sqf_list(*all=False*)
Returns a list of square-free factors of `f`.

sqf_list_include(*all=False*)
Returns a list of square-free factors of `f`.

sqf_norm()
Computes square-free norm of `f`.

sqf_part()
Computes square-free part of `f`.

sqr()
Square a multivariate polynomial `f`.

sturm()
Computes the Sturm sequence of `f`.

sub(*g*)
Subtract two multivariate polynomials `f` and `g`.

sub_ground(*c*)
Subtract an element of the ground domain from `f`.

subresultants(*g*)
Computes subresultant PRS sequence of `f` and `g`.

terms(*order=None*)
Returns all non-zero terms from `f` in lex order.

terms_gcd()
Remove GCD of terms from the polynomial `f`.

to_dict(*zero=False*)
Convert `f` to a dict representation with native coefficients.

to_exact()
Make the ground domain exact.

to_field()
Make the ground domain a field.

to_list()
Convert `f` to a list representation with native coefficients.

to_ring()
Make the ground domain a ring.

to_sympy_dict(*zero=False*)
Convert `f` to a dict representation with SymPy coefficients.

to_sympy_list()
Convert `f` to a list representation with SymPy coefficients.

to_tuple()
Convert `f` to a tuple representation with native coefficients.

This is needed for hashing.

total_degree()
Returns the total degree of `f`.

transform(*p*, *q*)
Evaluate functional transformation `q**n * f(p/q)`.

trunc(*p*)
Reduce `f` modulo a constant `p`.

unify(*g*)
Unify representations of two multivariate polynomials.

class sympy.polys.polyclasses.**DMF**(*rep*, *dom*, *lev=None*, *ring=None*)
Dense Multivariate Fractions over K.

add(*g*)
Add two multivariate fractions `f` and `g`.

cancel()
Remove common factors from `f.num` and `f.den`.

denom()
Returns the denominator of `f`.

exquo(*g*)
Computes quotient of fractions `f` and `g`.

frac_unify(*g*)
Unify representations of two multivariate fractions.

half_per(*rep*, *kill=False*)
Create a DMP out of the given representation.

invert(*check=True*)
Computes inverse of a fraction `f`.

property is_one
Returns `True` if `f` is a unit fraction.

property is_zero
Returns `True` if `f` is a zero fraction.

mul(*g*)
Multiply two multivariate fractions `f` and `g`.

neg()
Negate all coefficients in `f`.

numer()
Returns the numerator of `f`.

per(*num*, *den*, *cancel=True*, *kill=False*, *ring=None*)
Create a DMF out of the given representation.

poly_unify(*g*)
Unify a multivariate fraction and a polynomial.

pow(*n*)
Raise `f` to a non-negative power `n`.

quo(*g*)
Computes quotient of fractions `f` and `g`.

sub(*g*)
Subtract two multivariate fractions `f` and `g`.

class sympy.polys.polyclasses.**ANP**(*rep*, *mod*, *dom*)
Dense Algebraic Number Polynomials over a field.

LC()
Returns the leading coefficient of `f`.

TC()
Returns the trailing coefficient of `f`.

property is_ground
Returns `True` if `f` is an element of the ground domain.

property is_one
Returns `True` if `f` is a unit algebraic number.

property is_zero
Returns `True` if `f` is a zero algebraic number.

pow(*n*)
Raise `f` to a non-negative power `n`.

to_dict()
Convert `f` to a dict representation with native coefficients.

to_list()
Convert `f` to a list representation with native coefficients.

to_sympy_dict()
Convert `f` to a dict representation with SymPy coefficients.

to_sympy_list()
Convert `f` to a list representation with SymPy coefficients.

to_tuple()
Convert `f` to a tuple representation with native coefficients.

This is needed for hashing.

unify(*g*)
Unify representations of two algebraic numbers.

Internals of the Polynomial Manipulation Module

The implementation of the polynomials module is structured internally in "levels". There are four levels, called L0, L1, L2 and L3. The levels three and four contain the user-facing functionality and were described in the previous section. This section focuses on levels zero and one.

Level zero provides core polynomial manipulation functionality with C-like, low-level interfaces. Level one wraps this low-level functionality into object oriented structures. These are *not* the classes seen by the user, but rather classes used internally throughout the polys module.

There is one additional complication in the implementation. This comes from the fact that all polynomial manipulations are relative to a *ground domain*. For example, when factoring a polynomial like $x^{10} - 1$, one has to decide what ring the coefficients are supposed to belong to, or less trivially, what coefficients are allowed to appear in the factorization. This choice of coefficients is called a ground domain. Typical choices include the integers $\mathbb{Z}$, the rational numbers $\mathbb{Q}$ or various related rings and fields. But it is perfectly legitimate (although in this case uninteresting) to factorize over polynomial rings such as $k[Y]$, where k is some fixed field.

Thus the polynomial manipulation algorithms (both complicated ones like factoring, and simpler ones like addition or multiplication) have to rely on other code to manipulate the coefficients. In the polynomial manipulation module, such code is encapsulated in so-called "domains". A domain is basically a factory object: it takes various representations of data, and converts them into objects with unified interface. Every object created by a domain has

to implement the arithmetic operations $+$, $-$ and $\times$. Other operations are accessed through the domain, e.g. as in `ZZ.quo(ZZ(4), ZZ(2))`.

Note that there is some amount of *circularity*: the polynomial ring domains use the level one classes, the level one classes use the level zero functions, and level zero functions use domains. It is possible, in principle, but not in the current implementation, to work in rings like $k[X][Y]$. This would create even more layers. For this reason, working in the isomorphic ring $k[X, Y]$ is preferred.

Level Zero

Level zero contains the bulk code of the polynomial manipulation module.

Manipulation of dense, multivariate polynomials

These functions can be used to manipulate polynomials in $K[X_0, \ldots, X_u]$. Functions for manipulating multivariate polynomials in the dense representation have the prefix `dmp_`. Functions which only apply to univariate polynomials (i.e. $u = 0$) have the prefix `dup__`. The ground domain K has to be passed explicitly. For many multivariate polynomial manipulation functions also the level u, i.e. the number of generators minus one, has to be passed. (Note that, in many cases, `dup_` versions of functions are available, which may be slightly more efficient.)

Basic manipulation:

sympy.polys.densebasic.**dmp_LC**(*f*, *K*)

Return leading coefficient of `f`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import poly_LC
```

```
>>> poly_LC([], ZZ)
0
>>> poly_LC([ZZ(1), ZZ(2), ZZ(3)], ZZ)
1
```

sympy.polys.densebasic.**dmp_TC**(*f*, *K*)

Return trailing coefficient of `f`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import poly_TC
```

```
>>> poly_TC([], ZZ)
0
>>> poly_TC([ZZ(1), ZZ(2), ZZ(3)], ZZ)
3
```

sympy.polys.densebasic.**dmp_ground_LC**(*f*, *u*, *K*)

Return the ground leading coefficient.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_ground_LC
```

```
>>> f = ZZ.map([[[1], [2, 3]]])
```

```
>>> dmp_ground_LC(f, 2, ZZ)
1
```

sympy.polys.densebasic.**dmp_ground_TC**(*f*, *u*, *K*)

Return the ground trailing coefficient.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_ground_TC
```

```
>>> f = ZZ.map([[[1], [2, 3]]])
```

```
>>> dmp_ground_TC(f, 2, ZZ)
3
```

sympy.polys.densebasic.**dmp_true_LT**(*f*, *u*, *K*)

Return the leading term `c * x_1**n_1 ... x_k**n_k`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_true_LT
```

```
>>> f = ZZ.map([[4], [2, 0], [3, 0, 0]])
```

```
>>> dmp_true_LT(f, 1, ZZ)
((2, 0), 4)
```

sympy.polys.densebasic.**dmp_degree**(*f*, *u*)

Return the leading degree of `f` in `x_0` in `K[X]`.

Note that the degree of 0 is negative infinity (the SymPy object -oo).

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_degree
```

```
>>> dmp_degree([[[]]], 2)
-oo
```

```
>>> f = ZZ.map([[2], [1, 2, 3]])
```

```
>>> dmp_degree(f, 1)
1
```

sympy.polys.densebasic.**dmp_degree_in**(*f*, *j*, *u*)

Return the leading degree of `f` in `x_j` in `K[X]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_degree_in
```

```
>>> f = ZZ.map([[2], [1, 2, 3]])
```

```
>>> dmp_degree_in(f, 0, 1)
1
>>> dmp_degree_in(f, 1, 1)
2
```

sympy.polys.densebasic.**dmp_degree_list**(*f*, *u*)

Return a list of degrees of `f` in `K[X]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_degree_list
```

```
>>> f = ZZ.map([[1], [1, 2, 3]])
```

```
>>> dmp_degree_list(f, 1)
(1, 2)
```

sympy.polys.densebasic.**dmp_strip**(*f*, *u*)

Remove leading zeros from `f` in `K[X]`.

Examples

```
>>> from sympy.polys.densebasic import dmp_strip
```

```
>>> dmp_strip([[], [0, 1, 2], [1]], 1)
[[0, 1, 2], [1]]
```

sympy.polys.densebasic.**dmp_validate**(*f*, *K=None*)

Return the number of levels in `f` and recursively strip it.

Examples

```
>>> from sympy.polys.densebasic import dmp_validate
```

```
>>> dmp_validate([[], [0, 1, 2], [1]])
([[1, 2], [1]], 1)
```

```
>>> dmp_validate([[1], 1])
Traceback (most recent call last):
...
ValueError: invalid data structure for a multivariate polynomial
```

sympy.polys.densebasic.**dup_reverse**(*f*)

Compute `x**n * f(1/x)`, i.e.: reverse `f` in `K[x]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dup_reverse
```

```
>>> f = ZZ.map([1, 2, 3, 0])
```

```
>>> dup_reverse(f)
[3, 2, 1]
```

sympy.polys.densebasic.**dmp_copy**(*f*, *u*)

Create a new copy of a polynomial `f` in `K[X]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_copy
```

```
>>> f = ZZ.map([[1], [1, 2]])
```

```
>>> dmp_copy(f, 1)
[[1], [1, 2]]
```

sympy.polys.densebasic.**dmp_to_tuple**(*f*, *u*)

Convert f into a nested tuple of tuples.

This is needed for hashing. This is similar to dmp_copy().

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_to_tuple
```

```
>>> f = ZZ.map([[1], [1, 2]])
```

```
>>> dmp_to_tuple(f, 1)
((1,), (1, 2))
```

sympy.polys.densebasic.**dmp_normal**(*f*, *u*, *K*)

Normalize a multivariate polynomial in the given domain.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_normal
```

```
>>> dmp_normal([[], [0, 1.5, 2]], 1, ZZ)
[[1, 2]]
```

sympy.polys.densebasic.**dmp_convert**(*f*, *u*, *K0*, *K1*)

Convert the ground domain of `f` from `K0` to `K1`.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_convert
```

```
>>> R, x = ring("x", ZZ)
```

```
>>> dmp_convert([[R(1)], [R(2)]], 1, R.to_domain(), ZZ)
[[1], [2]]
>>> dmp_convert([[ZZ(1)], [ZZ(2)]], 1, ZZ, R.to_domain())
[[1], [2]]
```

sympy.polys.densebasic.**dmp_from_sympy**(*f*, *u*, *K*)

Convert the ground domain of `f` from SymPy to `K`.

Examples

```
>>> from sympy import S
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_from_sympy
```

```
>>> dmp_from_sympy([[S(1)], [S(2)]], 1, ZZ) == [[ZZ(1)], [ZZ(2)]]
True
```

sympy.polys.densebasic.**dmp_nth**(*f*, *n*, *u*, *K*)

Return the n-th coefficient of `f` in `K[x]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_nth
```

```
>>> f = ZZ.map([[1], [2], [3]])
```

```
>>> dmp_nth(f, 0, 1, ZZ)
[3]
>>> dmp_nth(f, 4, 1, ZZ)
[]
```

sympy.polys.densebasic.**dmp_ground_nth**(*f*, *N*, *u*, *K*)

Return the ground n-th coefficient of `f` in `K[x]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_ground_nth
```

```
>>> f = ZZ.map([[1], [2, 3]])
```

```
>>> dmp_ground_nth(f, (0, 1), 1, ZZ)
2
```

sympy.polys.densebasic.**dmp_zero_p**(*f*, *u*)

Return `True` if `f` is zero in `K[X]`.

Examples

```
>>> from sympy.polys.densebasic import dmp_zero_p
```

```
>>> dmp_zero_p([[[[[[]]]]]], 4)
True
>>> dmp_zero_p([[[[[1]]]]], 4)
False
```

sympy.polys.densebasic.**dmp_zero**(*u*)

Return a multivariate zero.

Examples

```
>>> from sympy.polys.densebasic import dmp_zero
```

```
>>> dmp_zero(4)
[[[[[]]]]]
```

sympy.polys.densebasic.**dmp_one_p**(*f*, *u*, *K*)

Return `True` if `f` is one in `K[X]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_one_p
```

```
>>> dmp_one_p([[[ZZ(1)]]], 2, ZZ)
True
```

sympy.polys.densebasic.**dmp_one**(*u*, *K*)

Return a multivariate one over `K`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_one
```

```
>>> dmp_one(2, ZZ)
[[[1]]]
```

sympy.polys.densebasic.**dmp_ground_p**(*f*, *c*, *u*)

Return True if `f` is constant in `K[X]`.

Examples

```
>>> from sympy.polys.densebasic import dmp_ground_p
```

```
>>> dmp_ground_p([[[3]]], 3, 2)
True
>>> dmp_ground_p([[[4]]], None, 2)
True
```

sympy.polys.densebasic.**dmp_ground**(*c*, *u*)

Return a multivariate constant.

Examples

```
>>> from sympy.polys.densebasic import dmp_ground
```

```
>>> dmp_ground(3, 5)
[[[[[[3]]]]]]
>>> dmp_ground(1, -1)
1
```

sympy.polys.densebasic.**dmp_zeros**(*n*, *u*, *K*)

Return a list of multivariate zeros.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_zeros
```

```
>>> dmp_zeros(3, 2, ZZ)
[[[[]]], [[[]]], [[[]]]]
>>> dmp_zeros(3, -1, ZZ)
[0, 0, 0]
```

sympy.polys.densebasic.**dmp_grounds**(*c*, *n*, *u*)

Return a list of multivariate constants.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_grounds
```

```
>>> dmp_grounds(ZZ(4), 3, 2)
[[[4]]], [[[4]]], [[[4]]]]
>>> dmp_grounds(ZZ(4), 3, -1)
[4, 4, 4]
```

sympy.polys.densebasic.**dmp_negative_p**(*f*, *u*, *K*)

Return `True` if `LC(f)` is negative.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_negative_p
```

```
>>> dmp_negative_p([[ZZ(1)], [-ZZ(1)]], 1, ZZ)
False
>>> dmp_negative_p([[-ZZ(1)], [ZZ(1)]], 1, ZZ)
True
```

sympy.polys.densebasic.**dmp_positive_p**(*f*, *u*, *K*)

Return `True` if `LC(f)` is positive.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_positive_p
```

```
>>> dmp_positive_p([[ZZ(1)], [-ZZ(1)]], 1, ZZ)
True
>>> dmp_positive_p([[-ZZ(1)], [ZZ(1)]], 1, ZZ)
False
```

sympy.polys.densebasic.**dmp_from_dict**(*f*, *u*, *K*)

Create a `K[X]` polynomial from a `dict`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_from_dict
```

```
>>> dmp_from_dict({(0, 0): ZZ(3), (0, 1): ZZ(2), (2, 1): ZZ(1)}, 1, ZZ)
[[1, 0], [], [2, 3]]
>>> dmp_from_dict({}, 0, ZZ)
[]
```

sympy.polys.densebasic.**dmp_to_dict**(*f*, *u*, *K=None*, *zero=False*)

Convert a `K[X]` polynomial to a `dict``.

Examples

```
>>> from sympy.polys.densebasic import dmp_to_dict
```

```
>>> dmp_to_dict([[1, 0], [], [2, 3]], 1)
{(0, 0): 3, (0, 1): 2, (2, 1): 1}
>>> dmp_to_dict([], 0)
{}
```

sympy.polys.densebasic.**dmp_swap**(*f*, *i*, *j*, *u*, *K*)

Transform K[..x_i..x_j..] to K[..x_j..x_i..].

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_swap
```

```
>>> f = ZZ.map([[[2], [1, 0]], []])
```

```
>>> dmp_swap(f, 0, 1, 2, ZZ)
[[[2], []], [[1, 0], []]]
>>> dmp_swap(f, 1, 2, 2, ZZ)
[[[1], [2, 0]], [[]]]
>>> dmp_swap(f, 0, 2, 2, ZZ)
[[[1, 0]], [[2, 0], []]]
```

sympy.polys.densebasic.**dmp_permute**(*f*, *P*, *u*, *K*)

Return a polynomial in K[x_{P(1)},..,x_{P(n)}].

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_permute
```

```
>>> f = ZZ.map([[[2], [1, 0]], []])
```

```
>>> dmp_permute(f, [1, 0, 2], 2, ZZ)
[[[2], []], [[1, 0], []]]
>>> dmp_permute(f, [1, 2, 0], 2, ZZ)
[[[1], []], [[2, 0], []]]
```

sympy.polys.densebasic.**dmp_nest**(*f*, *l*, *K*)

Return a multivariate value nested l-levels.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_nest
```

```
>>> dmp_nest([[ZZ(1)]], 2, ZZ)
[[[[1]]]]
```

sympy.polys.densebasic.**dmp_raise**(*f*, *l*, *u*, *K*)

Return a multivariate polynomial raised `l`-levels.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_raise
```

```
>>> f = ZZ.map([[], [1, 2]])
```

```
>>> dmp_raise(f, 2, 1, ZZ)
[[[[]]], [[[1]]], [[[2]]]]
```

sympy.polys.densebasic.**dmp_deflate**(*f*, *u*, *K*)

Map `x_i**m_i` to `y_i` in a polynomial in `K[X]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_deflate
```

```
>>> f = ZZ.map([[1, 0, 0, 2], [], [3, 0, 0, 4]])
```

```
>>> dmp_deflate(f, 1, ZZ)
((2, 3), [[1, 2], [3, 4]])
```

sympy.polys.densebasic.**dmp_multi_deflate**(*polys*, *u*, *K*)

Map `x_i**m_i` to `y_i` in a set of polynomials in `K[X]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_multi_deflate
```

```
>>> f = ZZ.map([[1, 0, 0, 2], [], [3, 0, 0, 4]])
>>> g = ZZ.map([[1, 0, 2], [], [3, 0, 4]])
```

```
>>> dmp_multi_deflate((f, g), 1, ZZ)
((2, 1), ([[1, 0, 0, 2], [3, 0, 0, 4]], [[1, 0, 2], [3, 0, 4]]))
```

sympy.polys.densebasic.**dmp_inflate**(*f*, *M*, *u*, *K*)

Map `y_i` to `x_i**k_i` in a polynomial in `K[X]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_inflate
```

```
>>> f = ZZ.map([[1, 2], [3, 4]])
```

```
>>> dmp_inflate(f, (2, 3), 1, ZZ)
[[1, 0, 0, 2], [], [3, 0, 0, 4]]
```

sympy.polys.densebasic.**dmp_exclude**(*f*, *u*, *K*)

Exclude useless levels from `f`.

Return the levels excluded, the new excluded `f`, and the new `u`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_exclude
```

```
>>> f = ZZ.map([[[1]], [[1], [2]]])
```

```
>>> dmp_exclude(f, 2, ZZ)
([2], [[1], [1, 2]], 1)
```

sympy.polys.densebasic.**dmp_include**(*f*, *J*, *u*, *K*)

Include useless levels in `f`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_include
```

```
>>> f = ZZ.map([[1], [1, 2]])
```

```
>>> dmp_include(f, [2], 1, ZZ)
[[[1]], [[1], [2]]]
```

sympy.polys.densebasic.**dmp_inject**(*f*, *u*, *K*, *front=False*)

Convert `f` from `K[X][Y]` to `K[X,Y]`.

Examples

```
>>> from sympy.polys.rings import ring
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_inject
```

```
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> dmp_inject([R(1), x + 2], 0, R.to_domain())
([[[1]], [[1], [2]]], 2)
>>> dmp_inject([R(1), x + 2], 0, R.to_domain(), front=True)
([[[1]], [[1, 2]]], 2)
```

sympy.polys.densebasic.**dmp_eject**(*f*, *u*, *K*, *front=False*)

Convert `f` from `K[X,Y]` to `K[X][Y]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_eject
```

```
>>> dmp_eject([[[1]], [[1], [2]]], 2, ZZ['x', 'y'])
[1, x + 2]
```

sympy.polys.densebasic.**dmp_terms_gcd**(*f*, *u*, *K*)

Remove GCD of terms from `f` in `K[X]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_terms_gcd
```

```
>>> f = ZZ.map([[1, 0], [1, 0, 0], [], []])
```

```
>>> dmp_terms_gcd(f, 1, ZZ)
((2, 1), [[1], [1, 0]])
```

sympy.polys.densebasic.**dmp_list_terms**(*f*, *u*, *K*, *order=None*)

List all non-zero terms from `f` in the given order `order`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_list_terms
```

```
>>> f = ZZ.map([[1, 1], [2, 3]])
```

```
>>> dmp_list_terms(f, 1, ZZ)
[((1, 1), 1), ((1, 0), 1), ((0, 1), 2), ((0, 0), 3)]
>>> dmp_list_terms(f, 1, ZZ, order='grevlex')
[((1, 1), 1), ((1, 0), 1), ((0, 1), 2), ((0, 0), 3)]
```

sympy.polys.densebasic.**dmp_apply_pairs**(*f*, *g*, *h*, *args*, *u*, *K*)

Apply `h` to pairs of coefficients of `f` and `g`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dmp_apply_pairs
```

```
>>> h = lambda x, y, z: 2*x + y - z
```

```
>>> dmp_apply_pairs([[1], [2, 3]], [[3], [2, 1]], h, (1,), 1, ZZ)
[[4], [5, 6]]
```

sympy.polys.densebasic.**dmp_slice**(*f*, *m*, *n*, *u*, *K*)

Take a continuous subsequence of terms of `f` in `K[X]`.

sympy.polys.densebasic.**dup_random**(*n*, *a*, *b*, *K*)

Return a polynomial of degree `n` with coefficients in `[a, b]`.

Examples

```
>>> from sympy.polys.domains import ZZ
>>> from sympy.polys.densebasic import dup_random
```

```
>>> dup_random(3, -10, 10, ZZ)
[-2, -8, 9, -4]
```

Arithmetic operations:

sympy.polys.densearith.**dmp_add_term**(*f*, *c*, *i*, *u*, *K*)

Add `c(x_2..x_u)*x_0**i` to `f` in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_add_term(x*y + 1, 2, 2)
2*x**2 + x*y + 1
```

sympy.polys.densearith.**dmp_sub_term**(*f*, *c*, *i*, *u*, *K*)

Subtract `c(x_2..x_u)*x_0**i` from `f` in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_sub_term(2*x**2 + x*y + 1, 2, 2)
x*y + 1
```

sympy.polys.densearith.**dmp_mul_term**(*f*, *c*, *i*, *u*, *K*)

Multiply `f` by `c(x_2..x_u)*x_0**i` in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_mul_term(x**2*y + x, 3*y, 2)
3*x**4*y**2 + 3*x**3*y
```

sympy.polys.densearith.**dmp_add_ground**(*f*, *c*, *u*, *K*)

Add an element of the ground domain to `f`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_add_ground(x**3 + 2*x**2 + 3*x + 4, ZZ(4))
x**3 + 2*x**2 + 3*x + 8
```

sympy.polys.densearith.**dmp_sub_ground**(*f*, *c*, *u*, *K*)

Subtract an element of the ground domain from `f`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_sub_ground(x**3 + 2*x**2 + 3*x + 4, ZZ(4))
x**3 + 2*x**2 + 3*x
```

sympy.polys.densearith.**dmp_mul_ground**(*f*, *c*, *u*, *K*)

Multiply `f` by a constant value in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_mul_ground(2*x + 2*y, ZZ(3))
6*x + 6*y
```

sympy.polys.densearith.**dmp_quo_ground**(*f*, *c*, *u*, *K*)

Quotient by a constant in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ, QQ
```

```
>>> R, x,y = ring("x,y", ZZ)
>>> R.dmp_quo_ground(2*x**2*y + 3*x, ZZ(2))
x**2*y + x
```

```
>>> R, x,y = ring("x,y", QQ)
>>> R.dmp_quo_ground(2*x**2*y + 3*x, QQ(2))
x**2*y + 3/2*x
```

sympy.polys.densearith.**dmp_exquo_ground**(*f*, *c*, *u*, *K*)

Exact quotient by a constant in `K[X]`.

Examples

```
>>> from sympy.polys import ring, QQ
>>> R, x,y = ring("x,y", QQ)
```

```
>>> R.dmp_exquo_ground(x**2*y + 2*x, QQ(2))
1/2*x**2*y + x
```

sympy.polys.densearith.**dup_lshift**(*f*, *n*, *K*)

Efficiently multiply `f` by `x**n` in `K[x]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x = ring("x", ZZ)
```

```
>>> R.dup_lshift(x**2 + 1, 2)
x**4 + x**2
```

sympy.polys.densearith.**dup_rshift**(*f*, *n*, *K*)

Efficiently divide `f` by `x**n` in `K[x]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x = ring("x", ZZ)
```

```
>>> R.dup_rshift(x**4 + x**2, 2)
x**2 + 1
>>> R.dup_rshift(x**4 + x**2 + 2, 2)
x**2 + 1
```

sympy.polys.densearith.**dmp_abs**(*f*, *u*, *K*)

Make all coefficients positive in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_abs(x**2*y - x)
x**2*y + x
```

sympy.polys.densearith.**dmp_neg**(*f*, *u*, *K*)

Negate a polynomial in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_neg(x**2*y - x)
-x**2*y + x
```

sympy.polys.densearith.**dmp_add**(*f*, *g*, *u*, *K*)

Add dense polynomials in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_add(x**2 + y, x**2*y + x)
x**2*y + x**2 + x + y
```

sympy.polys.densearith.**dmp_sub**(*f*, *g*, *u*, *K*)

Subtract dense polynomials in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_sub(x**2 + y, x**2*y + x)
-x**2*y + x**2 - x + y
```

sympy.polys.densearith.**dmp_add_mul**(*f*, *g*, *h*, *u*, *K*)

Returns `f + g*h` where `f, g, h` are in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_add_mul(x**2 + y, x, x + 2)
2*x**2 + 2*x + y
```

sympy.polys.densearith.**dmp_sub_mul**(*f*, *g*, *h*, *u*, *K*)

Returns `f - g*h` where `f, g, h` are in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_sub_mul(x**2 + y, x, x + 2)
-2*x + y
```

sympy.polys.densearith.**dmp_mul**(*f*, *g*, *u*, *K*)

Multiply dense polynomials in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_mul(x*y + 1, x)
x**2*y + x
```

sympy.polys.densearith.**dmp_sqr**(*f*, *u*, *K*)

Square dense polynomials in K[X].

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_sqr(x**2 + x*y + y**2)
x**4 + 2*x**3*y + 3*x**2*y**2 + 2*x*y**3 + y**4
```

sympy.polys.densearith.**dmp_pow**(*f*, *n*, *u*, *K*)

Raise f to the n-th power in K[X].

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_pow(x*y + 1, 3)
x**3*y**3 + 3*x**2*y**2 + 3*x*y + 1
```

sympy.polys.densearith.**dmp_pdiv**(*f*, *g*, *u*, *K*)

Polynomial pseudo-division in K[X].

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_pdiv(x**2 + x*y, 2*x + 2)
(2*x + 2*y - 2, -4*y + 4)
```

sympy.polys.densearith.**dmp_prem**(*f*, *g*, *u*, *K*)

Polynomial pseudo-remainder in K[X].

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_prem(x**2 + x*y, 2*x + 2)
-4*y + 4
```

sympy.polys.densearith.**dmp_pquo**(*f*, *g*, *u*, *K*)

Polynomial exact pseudo-quotient in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = x**2 + x*y
>>> g = 2*x + 2*y
>>> h = 2*x + 2
```

```
>>> R.dmp_pquo(f, g)
2*x
```

```
>>> R.dmp_pquo(f, h)
2*x + 2*y - 2
```

sympy.polys.densearith.**dmp_pexquo**(*f*, *g*, *u*, *K*)

Polynomial pseudo-quotient in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = x**2 + x*y
>>> g = 2*x + 2*y
>>> h = 2*x + 2
```

```
>>> R.dmp_pexquo(f, g)
2*x
```

```
>>> R.dmp_pexquo(f, h)
Traceback (most recent call last):
...
ExactQuotientFailed: [[2], [2]] does not divide [[1], [1, 0], []]
```

sympy.polys.densearith.**dmp_rr_div**(*f*, *g*, *u*, *K*)

Multivariate division with remainder over a ring.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_rr_div(x**2 + x*y, 2*x + 2)
(0, x**2 + x*y)
```

sympy.polys.densearith.**dmp_ff_div**(*f*, *g*, *u*, *K*)

Polynomial division with remainder over a field.

Examples

```
>>> from sympy.polys import ring, QQ
>>> R, x,y = ring("x,y", QQ)
```

```
>>> R.dmp_ff_div(x**2 + x*y, 2*x + 2)
(1/2*x + 1/2*y - 1/2, -y + 1)
```

sympy.polys.densearith.**dmp_div**(*f*, *g*, *u*, *K*)

Polynomial division with remainder in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ, QQ
```

```
>>> R, x,y = ring("x,y", ZZ)
>>> R.dmp_div(x**2 + x*y, 2*x + 2)
(0, x**2 + x*y)
```

```
>>> R, x,y = ring("x,y", QQ)
>>> R.dmp_div(x**2 + x*y, 2*x + 2)
(1/2*x + 1/2*y - 1/2, -y + 1)
```

sympy.polys.densearith.**dmp_rem**(*f*, *g*, *u*, *K*)

Returns polynomial remainder in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ, QQ
```

```
>>> R, x,y = ring("x,y", ZZ)
>>> R.dmp_rem(x**2 + x*y, 2*x + 2)
x**2 + x*y
```

```
>>> R, x,y = ring("x,y", QQ)
>>> R.dmp_rem(x**2 + x*y, 2*x + 2)
-y + 1
```

sympy.polys.densearith.**dmp_quo**(*f*, *g*, *u*, *K*)

Returns exact polynomial quotient in K[X].

Examples

```
>>> from sympy.polys import ring, ZZ, QQ
```

```
>>> R, x,y = ring("x,y", ZZ)
>>> R.dmp_quo(x**2 + x*y, 2*x + 2)
0
```

```
>>> R, x,y = ring("x,y", QQ)
>>> R.dmp_quo(x**2 + x*y, 2*x + 2)
1/2*x + 1/2*y - 1/2
```

sympy.polys.densearith.**dmp_exquo**(*f*, *g*, *u*, *K*)

Returns polynomial quotient in K[X].

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = x**2 + x*y
>>> g = x + y
>>> h = 2*x + 2
```

```
>>> R.dmp_exquo(f, g)
x
```

```
>>> R.dmp_exquo(f, h)
Traceback (most recent call last):
...
ExactQuotientFailed: [[2], [2]] does not divide [[1], [1, 0], []]
```

sympy.polys.densearith.**dmp_max_norm**(*f*, *u*, *K*)

Returns maximum norm of a polynomial in K[X].

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_max_norm(2*x*y - x - 3)
3
```

sympy.polys.densearith.**dmp_l1_norm**(*f*, *u*, *K*)

Returns l1 norm of a polynomial in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_l1_norm(2*x*y - x - 3)
6
```

sympy.polys.densearith.**dmp_expand**(*polys*, *u*, *K*)

Multiply together several polynomials in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_expand([x**2 + y**2, x + 1])
x**3 + x**2 + x*y**2 + y**2
```

Further tools:

sympy.polys.densetools.**dmp_integrate**(*f*, *m*, *u*, *K*)

Computes the indefinite integral of `f` in `x_0` in `K[X]`.

Examples

```
>>> from sympy.polys import ring, QQ
>>> R, x,y = ring("x,y", QQ)
```

```
>>> R.dmp_integrate(x + 2*y, 1)
1/2*x**2 + 2*x*y
>>> R.dmp_integrate(x + 2*y, 2)
1/6*x**3 + x**2*y
```

sympy.polys.densetools.**dmp_integrate_in**(*f*, *m*, *j*, *u*, *K*)

Computes the indefinite integral of `f` in `x_j` in `K[X]`.

Examples

```
>>> from sympy.polys import ring, QQ
>>> R, x,y = ring("x,y", QQ)
```

```
>>> R.dmp_integrate_in(x + 2*y, 1, 0)
1/2*x**2 + 2*x*y
>>> R.dmp_integrate_in(x + 2*y, 1, 1)
x*y + y**2
```

sympy.polys.densetools.**dmp_diff**(*f*, *m*, *u*, *K*)

m-th order derivative in `x_0` of a polynomial in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = x*y**2 + 2*x*y + 3*x + 2*y**2 + 3*y + 1
```

```
>>> R.dmp_diff(f, 1)
y**2 + 2*y + 3
>>> R.dmp_diff(f, 2)
0
```

sympy.polys.densetools.**dmp_diff_in**(*f*, *m*, *j*, *u*, *K*)

m-th order derivative in `x_j` of a polynomial in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = x*y**2 + 2*x*y + 3*x + 2*y**2 + 3*y + 1
```

```
>>> R.dmp_diff_in(f, 1, 0)
y**2 + 2*y + 3
>>> R.dmp_diff_in(f, 1, 1)
2*x*y + 2*x + 4*y + 3
```

sympy.polys.densetools.**dmp_eval**(*f*, *a*, *u*, *K*)

Evaluate a polynomial at `x_0 = a` in `K[X]` using the Horner scheme.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> R.dmp_eval(2*x*y + 3*x + y + 2, 2)
5*y + 8
```

sympy.polys.densetools.**dmp_eval_in**(*f*, *a*, *j*, *u*, *K*)

Evaluate a polynomial at `x_j = a` in `K[X]` using the Horner scheme.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = 2*x*y + 3*x + y + 2
```

```
>>> R.dmp_eval_in(f, 2, 0)
5*y + 8
>>> R.dmp_eval_in(f, 2, 1)
7*x + 4
```

sympy.polys.densetools.**dmp_eval_tail**(*f*, *A*, *u*, *K*)

Evaluate a polynomial at `x_j = a_j, ...` in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = 2*x*y + 3*x + y + 2
```

```
>>> R.dmp_eval_tail(f, [2])
7*x + 4
>>> R.dmp_eval_tail(f, [2, 2])
18
```

sympy.polys.densetools.**dmp_diff_eval_in**(*f*, *m*, *a*, *j*, *u*, *K*)

Differentiate and evaluate a polynomial in `x_j` at `a` in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = x*y**2 + 2*x*y + 3*x + 2*y**2 + 3*y + 1
```

```
>>> R.dmp_diff_eval_in(f, 1, 2, 0)
y**2 + 2*y + 3
>>> R.dmp_diff_eval_in(f, 1, 2, 1)
6*x + 11
```

sympy.polys.densetools.**dmp_trunc**(*f*, *p*, *u*, *K*)

Reduce a `K[X]` polynomial modulo a polynomial `p` in `K[Y]`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = 3*x**2*y + 8*x**2 + 5*x*y + 6*x + 2*y + 3
>>> g = (y - 1).drop(x)
```

```
>>> R.dmp_trunc(f, g)
11*x**2 + 11*x + 5
```

sympy.polys.densetools.**dmp_ground_trunc**(*f*, *p*, *u*, *K*)

Reduce a `K[X]` polynomial modulo a constant `p` in `K`.

Examples

```
>>> from sympy.polys import ring, ZZ
>>> R, x,y = ring("x,y", ZZ)
```

```
>>> f = 3*x**2*y + 8*x**2 + 5*x*y + 6*x + 2*y + 3
```

```
>>> R.dmp_ground_trunc(f, ZZ(3))
-x**2 - x*y - y
```

sympy.polys.densetools.**dup_monic**(*f*, *K*)

Divide all coefficients by `LC(f)` in `K[x]`.

Examples

```
>>> from sympy.polys import ring, ZZ, QQ
```

```
>>> R, x = ring("x", ZZ)
>>> R.dup_monic(3*x**2 + 6*x + 9)
x**2 + 2*x + 3
```

```
>>> R, x = ring("x", QQ)
>>> R.dup_monic(3*x**2 + 4*x + 2)
x**2 + 4/3*x + 2/3
```

sympy.polys.densetools.**dmp_ground_monic**(*f*, *u*, *K*)

Divide all coefficients by `LC(f)` in `K[X]`.

Examples

```
>>> from sympy.polys import ring, ZZ, QQ
```

```
>>> R, x,y = ring("x,y", ZZ)
>>> f = 3*x**2*y + 6*x**2 + 3*x*y + 9*y + 3
```

```
>>> R.dmp_ground_monic(f)
x**2*y + 2*x**2 + x*y + 3*y + 1
```

```
>>> R, x,y = ring("x,y", QQ)
>>> f = 3*x**2*y + 8*x**2 + 5*x*y + 6*x + 2*y + 3
```

```
>>> R.dmp_ground_monic(f)
x**2*y + 8/3*x**2 + 5/3*x*y + 2*x + 2/3*y + 1
```

sympy.polys.densetools.**dup_content**(*f*, *K*)

Compute the GCD of coefficients of `f` in `K[x]`.

Examples

```
>>> from sympy.polys import ring, ZZ, QQ
```

```
>>> R, x = ring("x", ZZ)
>>> f = 6*x**2 + 8*x + 12
```

```
>>> R.dup_content(f)
2
```

```
>>> R, x = ring("x", QQ)
>>> f = 6*x**2 + 8*x + 12
```