Contents

Chapter 1

Classes

1.1 poly.uniutil – univariate utilities

- **Classes**
  - **RingPolynomial**
  - **DomainPolynomial**
  - **UniqueFactorizationDomainPolynomial**
  - **IntegerPolynomial**
  - **FieldPolynomial**
  - **FinitePrimeFieldPolynomial**
  - OrderProvider
  - DivisionProvider
  - PseudoDivisionProvider
  - ContentProvider
  - SubresultantGcdProvider
  - PrimeCharacteristicFunctionsProvider
  - VariableProvider
  - RingElementProvider
- **Functions**
  - **polynomial**

1.1.1 RingPolynomial – polynomial over commutative ring

Initialize (Constructor)

RingPolynomial(`coefficients`: ***terminit***, `coeffring`: ***CommutativeRing***, `**keywords`: ***dict***)
→ ***RingPolynomial object***

Initialize a polynomial over the given commutative ring `coeffring`.

This class inherits from **SortedPolynomial**, **OrderProvider** and **RingElementProvider**.

The type of the `coefficients` is **terminit**. `coeffring` is an instance of descendant of **CommutativeRing**.

Methods

1.1.1.1 getRing

getRing(self) → ***Ring***

Return an object of a subclass of `Ring`, to which the polynomial belongs.
(This method overrides the definition in RingElementProvider)

1.1.1.2 getCoefficientRing

getCoefficientRing(self) → ***Ring***

Return an object of a subclass of `Ring`, to which the all coefficients belong.
(This method overrides the definition in RingElementProvider)

1.1.1.3 shift_degree_to

shift_degree_to(self, degree: ***integer***) → ***polynomial***

Return polynomial whose degree is the given `degree`. More precisely, let $f(X) = a_0 + ... + a_n X^n$, then `f.shift_degree_to(m)` returns:

- zero polynomial, if f is zero polynomial
- $a_{n-m} + ... + a_n X^m$, if $0 \leq m < n$
- $a_0 X^{m-n} + ... + a_n X^m$, if $m \geq n$

(This method is inherited from OrderProvider)

1.1.1.4 split_at

split_at(self, degree: ***integer***) → ***polynomial***

Return tuple of two polynomials, which are split at the given degree. The term of the given degree, if exists, belongs to the lower degree polynomial.
(This method is inherited from OrderProvider)

1.1.2 DomainPolynomial – polynomial over domain

Initialize (Constructor)

DomainPolynomial(coefficients: ***terminit***, coeffring: ***CommutativeRing***, **keywords: ***dict***)
→ ***DomainPolynomial object***

Initialize a polynomial over the given domain `coeffring`.

In addition to the basic polynomial operations, it has pseudo division methods.

This class inherits **RingPolynomial** and **PseudoDivisionProvider**.

The type of the `coefficients` is **terminit**. `coeffring` is an instance of descendant of **CommutativeRing** which satisfies `coeffring.isdomain()`.

Methods

1.1.2.1 pseudo_divmod

pseudo_divmod(`self`, `other`: ***polynomial***) → ***tuple***

Return a tuple `(Q, R)`, where Q, R are polynomials such that:

$$d^{deg(f)-deg(other)+1}f = other \times Q + R,$$

where d is the leading coefficient of `other`.
(This method is inherited from PseudoDivisionProvider)

1.1.2.2 pseudo_floordiv

pseudo_floordiv(`self`, `other`: ***polynomial***) → ***polynomial***

Return a polynomial Q such that:

$$d^{deg(f)-deg(other)+1}f = other \times Q + R,$$

where d is the leading coefficient of `other`.
(This method is inherited from PseudoDivisionProvider)

1.1.2.3 pseudo_mod

pseudo_mod(`self`, `other`: ***polynomial***) → ***polynomial***

Return a polynomial R such that:

$$d^{deg(f)-deg(other)+1}f = other \times Q + R,$$

where d is the leading coefficient of `other`.
(This method is inherited from PseudoDivisionProvider)

1.1.2.4 exact_division

exact_division(`self`, `other`: ***polynomial***) → ***polynomial***

Return quotient of exact division.
(This method is inherited from PseudoDivisionProvider)

1.1.2.5 scalar_exact_division

scalar_exact_division(`self`, `scale`: ***CommutativeRingElement***)
→ ***polynomial***

Return quotient by `scale` which can divide each coefficient exactly.
(This method is inherited from PseudoDivisionProvider)

1.1.2.6 discriminant

discriminant(`self`) → ***CommutativeRingElement***

Return discriminant of the polynomial.

1.1.2.7 to_field_polynomial

to_field_polynomial(`self`) → ***FieldPolenomial***

Return a `FieldPolynomial` object obtained by embedding the polynomial ring over the domain D to over the quotient field of D.

1.1.3 UniqueFactorizationDomainPolynomial – polynomial over UFD

Initialize (Constructor)

UniqueFactorizationDomainPolynomial(`coefficients`: ***terminit***, `coeffring`: ***CommutativeRing***, `**keywords`: ***dict***)
→ ***UniqueFactorizationDomainPolynomial object***

Initialize a polynomial over the given UFD `coeffring`.

This class inherits from **DomainPolynomial**, **SubresultantGcdProvider** and **ContentProvider**.

The type of the `coefficients` is **terminit**. `coeffring` is an instance of descendant of **CommutativeRing** which satisfies `coeffring.isufd()`.

1.1.3.1 content

content(`self`) → ***CommutativeRingElement***

Return content of the polynomial.
(This method is inherited from ContentProvider)

1.1.3.2 primitive_part

primitive_part(`self`) → ***UniqueFactorizationDomainPolynomial***

Return the primitive part of the polynomial.
(This method is inherited from ContentProvider)

1.1.3.3 subresultant_gcd

subresultant_gcd(`self`, `other`: ***polynomial***) → ***UniqueFactorizationDomainPolynomial***

Return the greatest common divisor of given polynomials. They must be in the polynomial ring and its coefficient ring must be a UFD.
(This method is inherited from SubresultantGcdProvider)
Reference: [1]Algorithm 3.3.1

1.1.3.4 subresultant_extgcd

subresultant_extgcd(`self`, `other`: ***polynomial***) → ***tuple***

Return `(A, B, P)` s.t. $A \times self + B \times other = P$, where P is the greatest common divisor of given polynomials. They must be in the polynomial ring and its coefficient ring must be a UFD.
Reference: [2]p.18
(This method is inherited from SubresultantGcdProvider)

1.1.3.5 resultant

resultant(`self`, `other`: ***polynomial***) → ***polynomial***

Return the resultant of `self` and `other`.
(This method is inherited from SubresultantGcdProvider)

1.1.4 IntegerPolynomial – polynomial over ring of rational integers

Initialize (Constructor)

IntegerPolynomial(`coefficients`: ***terminit***, `coeffring`: ***CommutativeRing***, `**keywords`: ***dict***)
→ ***IntegerPolynomial object***

Initialize a polynomial over the given commutative ring `coeffring`.

This class is required because special initialization must be done for built-in int/long.

This class inherits from **UniqueFactorizationDomainPolynomial**.

The type of the `coefficients` is **terminit**. `coeffring` is an instance of **IntegerRing**. You have to give the rational integer ring, though it seems redundant.

1.1.5 FieldPolynomial – polynomial over field

Initialize (Constructor)

FieldPolynomial(`coefficients`: ***terminit***, `coeffring`: ***Field***, **`keywords`: ***dict***)
→ ***FieldPolynomial object***

Initialize a polynomial over the given field `coeffring`.

Since the polynomial ring over field is a Euclidean domain, it provides divisions.

This class inherits from **RingPolynomial**, **DivisionProvider** and **ContentProvider**.

The type of the `coefficients` is **terminit**. `coeffring` is an instance of descendant of **Field**.

Operations

operator	explanation
`f // g`	quotient of floor division
`f % g`	remainder
`divmod(f, g)`	quotient and remainder
`f / g`	division in rational function field

Methods

1.1.5.1 content

content(self) → ***FieldElement***

Return content of the polynomial.
(This method is inherited from ContentProvider)

1.1.5.2 primitive_part

primitive_part(self) → ***polynomial***

Return the primitive part of the polynomial.
(This method is inherited from ContentProvider)

1.1.5.3 mod

mod(self, dividend: ***polynomial***) → ***polynomial***

Return *dividend* mod *self*.
(This method is inherited from DivisionProvider)

1.1.5.4 scalar_exact_division

scalar_exact_division(self, scale: ***FieldElement***)
→ ***polynomial***

Return quotient by `scale` which can divide each coefficient exactly.
(This method is inherited from DivisionProvider)

1.1.5.5 gcd

gcd(self, other: ***polynomial***) → ***polynomial***

Return a greatest common divisor of self and other.

Returned polynomial is always monic.
(This method is inherited from DivisionProvider)

1.1.5.6 extgcd

extgcd(self, other: ***polynomial***) → ***tuple***

Return a tuple `(u, v, d)`; they are the greatest common divisor d of two polynomials `self` and `other` and u, v such that

$$d = self \times u + other \times v$$

See **extgcd**.
(This method is inherited from DivisionProvider)

1.1.6 FinitePrimeFieldPolynomial – polynomial over finite prime field

Initialize (Constructor)

FinitePrimeFieldPolynomial(`coefficients`: ***terminit***, `coeffring`: ***FinitePrimeField***, **`keywords`: ***dict***)
→ ***FinitePrimeFieldPolynomial object***

Initialize a polynomial over the given commutative ring `coeffring`.

This class inherits from **FieldPolynomial** and **PrimeCharacteristicFunctionsProvider**.

The type of the `coefficients` is **terminit**. `coeffring` is an instance of descendant of **FinitePrimeField**.

Methods

1.1.6.1 mod_pow – powering with modulus

mod_pow(`self`, `polynom`: ***polynomial***, `index`: ***integer***) → ***polynomial***

Return $polynom^{index} \bmod self$.

Note that **`self`** is the modulus.
(This method is inherited from PrimeCharacteristicFunctionsProvider)

1.1.6.2 pthroot

pthroot(`self`) → ***polynomial***

Return a polynomial obtained by sending X^p to X, where p is the characteristic. If the polynomial does not consist of p-th powered terms only, result is nonsense.
(This method is inherited from PrimeCharacteristicFunctionsProvider)

1.1.6.3 squarefree_decomposition

squarefree_decomposition(`self`) → ***dict***

Return the square free decomposition of the polynomial.

The return value is a dict whose keys are integers and values are corresponding powered factors. For example, If

Examples

```
>>> A = A1 * A2**2
>>> A.squarefree_decomposition()
{1: A1, 2: A2}.
```

(This method is inherited from PrimeCharacteristicFunctionsProvider)

1.1.6.4 distinct_degree_decomposition

distinct_degree_decomposition(`self`) → ***dict***

Return the distinct degree factorization of the polynomial.

The return value is a dict whose keys are integers and values are corresponding product of factors of the degree. For example, if $A = A1 \times A2$, and all irreducible factors of $A1$ having degree 1 and all irreducible factors of $A2$ having degree 2, then the result is: `{1: A1, 2: A2}`.

The given polynomial must be square free, and its coefficient ring must be a finite field.
(This method is inherited from PrimeCharacteristicFunctionsProvider)

1.1.6.5 split_same_degrees

split_same_degrees(`self`, `degree`: *)* → ***list***

Return the irreducible factors of the polynomial.

The polynomial must be a product of irreducible factors of the given degree.
(This method is inherited from PrimeCharacteristicFunctionsProvider)

1.1.6.6 factor

factor(`self`) → ***list***

Factor the polynomial.

The returned value is a list of tuples whose first component is a factor and second component is its multiplicity.
(This method is inherited from PrimeCharacteristicFunctionsProvider)

1.1.6.7 isirreducible

isirreducible(`self`) → ***bool***

If the polynomial is irreducible return `True`, otherwise `False`.
(This method is inherited from PrimeCharacteristicFunctionsProvider)

1.1.7 polynomial – factory function for various polynomials

polynomial(`coefficients`: ***terminit***, `coeffring`: ***CommutativeRing***) → ***polynomial***

Return a polynomial.

†One can override the way to choose a polynomial type from a coefficient ring, by setting:
`special_ring_table[coeffring_type] = polynomial_type`
before the function call.

Bibliography

[1] Henri Cohen. *A Course in Computational Algebraic Number Theory*. GTM138. Springer, 1st. edition, 1993.

[2] Kida Yuuji. 代数体の整数基底と素数の素イデアル分解 (japanese). http://www.rkmath.rikkyo.ac.jp/~kida/intbasis.pdf.