Contents

Chapter 1

Classes

1.1 poly.univar – univariate polynomial

- **Classes**
  - †**PolynomialInterface**
  - †**BasicPolynomial**
  - **SortedPolynomial**

This poly.univar using following type:

polynomial :
: `polynomial` is an instance of some descendant class of **PolynomialInterface** in this context.

1.1.1 PolynomialInterface – base class for all univariate polynomials

Initialize (Constructor)

Since the interface is an abstract class, do not instantiate.
The class is derived from **FormalSumContainerInterface**.

Operations

operator	explanation
`f * g`	multiplication[1]
`f ** i`	powering

Methods

1.1.1.1 differentiate – formal differentiation

differentiate(`self`) → ***polynomial***

Return the formal differentiation of this polynomial.

1.1.1.2 downshift_degree – decreased degree polynomial

downshift_degree(`self`, `slide`: ***integer***) → ***polynomial***

Return the polynomial obtained by shifting downward all terms with degrees of `slide`.

Be careful that if the least degree term has the degree less than `slide` then the result is not mathematically a polynomial. Even in such a case, the method does not raise an exception.

†`f.downshift_degree(slide)` is equivalent to `f.`**upshift_degree**`(-slide)`.

1.1.1.3 upshift_degree – increased degree polynomial

upshift_degree(`self`, `slide`: ***integer***) → ***polynomial***

Return the polynomial obtained by shifting upward all terms with degrees of `slide`.

†`f.upshift_degree(slide)` is equivalent to `f.term_mul((slide, 1))`.

1.1.1.4 ring_mul – multiplication in the ring

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

Return the result of multiplication with the `other` polynomial.

1.1.1.5 scalar_mul – multiplication with a scalar

scalar_mul(`self`, `scale`: ***scalar***) → ***polynomial***

Return the result of multiplication by scalar `scale`.

1.1.1.6 term_mul – multiplication with a term

term_mul(`self`, `term`: ***term***) → ***polynomial***

Return the result of multiplication with the given `term`. The `term` can be given as a tuple `(degree, coeff)` or as a `polynomial`.

1.1.1.7 square – multiplication with itself

square(`self`) → ***polynomial***

Return the square of this polynomial.

1.1.2 BasicPolynomial – basic implementation of polynomial

Basic polynomial data type. There are no concept such as variable name and ring.

Initialize (Constructor)

BasicPolynomial(`coefficients`: ***terminit***, **`keywords`: ***dict***)
→ ***BasicPolynomial***

This class inherits and implements **PolynomialInterface**.

The type of the `coefficients` is **terminit**.

1.1.3 SortedPolynomial – polynomial keeping terms sorted

Initialize (Constructor)

SortedPolynomial(`coefficients`: ***terminit***, `_sorted`: ***bool***=**False**, **`keywords`: ***dict***)
→ ***SortedPolynomial***

The class is derived from **PolynomialInterface**.

The type of the `coefficients` is **terminit**. Optionally `_sorted` can be `True` if the coefficients is an already sorted list of terms.

Methods

1.1.3.1 degree – degree

degree(`self`) → ***integer***

Return the degree of this polynomial. If the polynomial is the zero polynomial, the degree is -1.

1.1.3.2 leading_coefficient – the leading coefficient

leading_coefficient(`self`) → ***object***

Return the coefficient of highest degree term.

1.1.3.3 leading_term – the leading term

leading_term(`self`) → ***tuple***

Return the leading term as a tuple `(degree, coefficient)`.

1.1.3.4 †ring_mul_karatsuba – the leading term

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

Multiplication of two polynomials in the same ring. Computation is carried out by Karatsuba method.

This may run faster when degree is higher than 100 or so. It is off by default, if you need to use this, do by yourself.