

Contents

1	Classes	2
1.1	poly.univar – 一変数多項式	2
1.1.1	PolynomialInterface – 全ての一変数多項式に対する基底クラス	3
1.1.1.1	differentiate – 正式な微分	4
1.1.1.2	downshift_degree – 多項式の次数を下げる	4
1.1.1.3	upshift_degree – 多項式の次数を上げる	4
1.1.1.4	ring_mul – 環上の乗法	4
1.1.1.5	scalar_mul – スカラーの乗法	4
1.1.1.6	term_mul – 項の乗法	4
1.1.1.7	square – 自身との乗法	5
1.1.2	BasicPolynomial – 多項式の基本的実装	5
1.1.3	SortedPolynomial – ソートされた項を維持する多項式	5
1.1.3.1	degree – 次数	6
1.1.3.2	leading_coefficient – 主係数	6
1.1.3.3	leading_term – 主項	6
1.1.3.4	ring_mul_karatsuba – the leading term	6

Chapter 1

Classes

1.1 poly.univar – 一変数多項式

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

この `poly.univar` は以下の型を使っている:

polynomial :

`polynomial` はこの文脈では **PolynomialInterface** の子孫のインスタンス.

1.1.1 PolynomialInterface – 全ての一変数多項式に対する基底クラス

Initialize (Constructor)

抽象クラスなので例示化はっしません.
このクラスは **FormalSumContainerInterface** から派生される.

Operations

operator	explanation
f * g	乗法 ¹
f ** i	べき乗

Methods

1.1.1.1 differentiate – 正式な微分

`differentiate(self)` → *polynomial*

多項式の正式な微分を返す.

1.1.1.2 downshift_degree – 多項式の次数を下げる

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

次数 `slide` を持つ全ての項を下にシフトして得られた多項式を返す.

最も次数が小さい項が `slide` より小さいとき, 結果は数ガキ的でない多項式だということに注意してください. このような場合でも, このメソッドは例外を起こさない.

†`f.downshift_degree(slide)` は `f.upshift_degree(-slide)` と同等のものであります.

1.1.1.3 upshift_degree – 多項式の次数を上げる

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

次数 `slide` を持つ全ての項を上シフトして得られた多項式を返す.

†`f.upshift_degree(slide)` は `f.term_mul((slide, 1))` と同等のものである.

1.1.1.4 ring_mul – 環上の乗法

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

多項式 `other` との乗法の結果を返す.

1.1.1.5 scalar_mul – スカラーの乗法

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

スカラー `scale` による乗法の結果を返す.

1.1.1.6 term_mul – 項の乗法

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

与えられた `term` の乗法の結果を返す. `term` はタプル (`degree`, `coeff`) として与えられるか, *polynomial* として与えられる.

1.1.1.7 square – 自身との乗法

```
square(self) → polynomial
```

この多項式の平方を返す.

1.1.2 BasicPolynomial – 多項式の基本的実装

基本的な多項式の型. 変数名や環のようなコンセプトはない.

Initialize (Constructor)

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

このクラスは **PolynomialInterface** を継承し実装する.

`coefficients` の型は **terminit** です.

1.1.3 SortedPolynomial – ソートされた項を維持する多項式

Initialize (Constructor)

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

このクラスは **PolynomialInterface** から派生される.

`coefficients` の型は **terminit** です. 任意的に もし係数がすでにソートされた項のリストなら, `_sorted` は `True` になり得る.

Methods

1.1.3.1 degree – 次数

`degree(self)` → *integer*

この多項式の次数を返す。もし零多項式なら、次数は -1 となる。

1.1.3.2 leading_coefficient – 主係数

`leading_coefficient(self)` → *object*

最も次数が高い項の係数を返す。

1.1.3.3 leading_term – 主項

`leading_term(self)` → *tuple*

タプル (degree, coefficient) として主項を返す。

1.1.3.4 †ring_mul_karatsuba – the leading term

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

同じ環上での二つの多項式の乗法。計算は Karatsuba 法によって実行される。
これはだいたい次数が 100 くらいより上のとき早く動くだろう。もしこれをあなた自身で使う必要があるなら、初期設定によりそれはなしです。