Contents

Chapter 1

Functions

1.1 equation – solving equations, congruences

In the following descriptions, some type aliases are used.

poly_list :

poly_list is a list `[a0, a1, ..., an]` representing a polynomial coefficients in ascending order, i.e., meaning $a_0 + a_1X + \cdots + a_nX^n$. The type of each `ai` depends on each function (explained in their descriptions).

integer :

`integer` is one of *int*, *long* or **Integer**.

complex :

`complex` includes all number types in the complex field: **integer**, *float*, *complex* of `Python` , **Rational** of `NZMATH` , etc.

1.1.1 e1 – solve equation with degree 1

e1(`f`: **poly_list**) → **complex**

$ax + b = 0$ の値を返す。

`f` は **complex** の linkingoneequationpoly_list `[b, a]` でなければならない。

1.1.2 e1_ZnZ – solve congruent equation modulo n with degree 1

e1_ZnZ(`f`: **poly_list**, `n`: ***integer***) → ***integer***

$ax + b \equiv 0 \pmod{\mathtt{n}}$. の値を返す。

`f` は **integer** の **poly_list** `[b, a]` でなければならない。

1.1.3 e2 – solve equation with degree 2

e2(`f`: **poly_list**) → ***tuple***

$ax^2 + bx + c = 0$ の値を返す。

`f` は **complex** の **poly_list** `[c, b, a]` でなければならない。
結果のタップルは副根も含め二つの根である。

1.1.4 e2_Fp – solve congruent equation modulo p with degree 2

e2_Fp(`f`: **poly_list**, `p`: ***integer***) → ***list***

$ax^2 + bx + c \equiv 0 \pmod{\mathtt{p}}$ の値を返す。

同じ値が返ってきたならば、その値は多重根である。

`f` は integers `[c, b, a]` の **poly_list** でなければならない。さらに、`p` は素数整数。**integer**.

1.1.5 e3 – solve equation with degree 3

e3(`f`: **poly_list**) → ***list***

$ax^3 + bx^2 + cx + d = 0$ の値を返す。

`f` は **complex** の **poly_list** `[d, c, b, a]` でなければならない。
この結果のタップルには重根を含めて三つの根がある。

1.1.6 e3_Fp – solve congruent equation modulo p with degree 3

e3_Fp(`f`: **poly_list**, `p`: ***integer***) → ***list***

$ax^3 + bx^2 + cx + d \equiv 0 \pmod{\mathtt{p}}$ の値を返す。

同じ値が返ってきたならば、その値は多重根である。

`f` は **integer** の **poly_list** `[d, c, b, a]` でなければならない。In addition, `p` は素数整数である。**integer**.

1.1.7 Newton – solve equation using Newton's method

Newton(f: poly_list, initial: complex=1, repeat: *integer*=250)
→ *complex*

$a_n x^n + \cdots + a_1 x + a_0 = 0$ の値を返す。

もしすべての根を得たいのなら **SimMethod** を使うことをお勧めする。
† もし `initial` が実数根を持たない実数ならばこの関数は役に立たない。

`f` は **complex** の **poly_list** でなければならない。
`initial` is an initial approximation **complex** number. `repeat` は根を近似する数である。

1.1.8 SimMethod – find all roots simultaneously

SimMethod(f: poly_list, NewtonInitial: complex=1, repeat: *integer*=250)
→ *list*

$a_n x^n + \cdots + a_1 x + a_0$ の根の一つを返す。

† もしこの方程式が多重根を持っていたら、エラーが返ってくるかもしれない。

`f` は **complex** の **poly_list** でなければならない。
NewtonInitial と `repeat` は近似値を得るため **Newton** を通過するだろう。

1.1.9 root_Fp – solve congruent equation modulo p

root_Fp(f: poly_list, p: *integer*) → *integer*

$a_n x^n + \cdots + a_1 x + a_0 \equiv 0 \pmod{\mathtt{p}}$ の根の人るを返す。

すべての根を得たいのなら **allroots_Fp** を使ってください。

`f` は **integer** の **poly_list** でなければならない。さらに `p` は素数。

根が一つもなければこの関数は何も返さない。

1.1.10 allroots_Fp – solve congruent equation modulo p

allroots_Fp(f: poly_list, p: *integer*) → *integer*

$a_n x^n + \cdots + a_1 x + a_0 \equiv 0 \pmod{\mathtt{p}}$. のすべての根を返す。

f は integer の poly_list でなければならない。さらに p は素数。
根が一つもないときはこの関数はからのリストを返す。

Examples

```
>>> equation.e1([1, 2])
-0.5
>>> equation.e1([1j, 2])
-0.5j
>>> equation.e1_ZnZ([3, 2], 5)
1
>>> equation.e2([-3, 1, 1])
(1.3027756377319946, -2.3027756377319948)
>>> equation.e2_Fp([-3, 1, 1], 13)
[6, 6]
>>> equation.e3([1, 1, 2, 1])
[(-0.12256116687665397-0.74486176661974479j),
(-1.7548776662466921+1.8041124150158794e-16j),
(-0.12256116687665375+0.74486176661974468j)]
>>> equation.e3_Fp([1, 1, 2, 1], 7)
[3]
>>> equation.Newton([-3, 2, 1, 1])
0.84373427789806899
>>> equation.Newton([-3, 2, 1, 1], 2)
0.84373427789806899
>>> equation.Newton([-3, 2, 1, 1], 2, 1000)
0.84373427789806899
>>> equation.SimMethod([-3, 2, 1, 1])
[(0.84373427789806887+0j),
(-0.92186713894903438+1.6449263775999723j),
(-0.92186713894903438-1.6449263775999723j)]
>>> equation.root_Fp([-3, 2, 1, 1], 7)
>>> equation.root_Fp([-3, 2, 1, 1], 11)
9L
>>> equation.allroots_Fp([-3, 2, 1, 1], 7)
[]
```

```
>>> equation.allroots_Fp([-3, 2, 1, 1], 11)
[9L]
>>> equation.allroots_Fp([-3, 2, 1, 1], 13)
[3L, 7L, 2L]
```

Bibliography