Contents

Chapter 1

Classes

1.1 elliptic – elliptic class object

- **Classes**
  - **ECGeneric**
  - **ECoverQ**
  - **ECoverGF**
- **Functions**
  - **EC**

This module using following type:

weierstrassform :
: `weierstrassform` is a list $(a_1, a_2, a_3, a_4, a_6)$ or (a_4, a_6), it represents $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ or $E : y^2 = x^3 + a_4x + a_6$, respectively.

infpoint :
: `infpoint` is the list `[0]`, which represents infinite point on the elliptic curve.

point :
: point is two-dimensional coordinate list [`x`, `y`] or **infpoint**.

1.1.1 †ECGeneric – generic elliptic curve class

Initialize (Constructor)

ECGeneric(coefficient: weierstrassform, basefield: *Field*=None)
→ *ECGeneric*

楕円曲線を作る。

The class is for the definition of elliptic curves over general fields. Instead of using this class directly, we recommend that you call **EC**.
†The class precomputes the following values.

- shorter form: $y^2 = b_2x^3 + b_4x^2 + b_6x + b_8$
- shortest form: $y^2 = x^3 + c_4x + c_6$
- discriminant
- j-invariant

All elements of `coefficient` must be in `basefield`.
See **weierstrassform** for more information about `coefficient`. If discriminant of `self` equals 0, it raises ValueError.

Attributes

basefield :
It expresses the field which each coordinate of all points in `self` is on. (This means not only `self` is defined over `basefield`.)

ch :
It expresses the characteristic of `basefield`.

infpoint :
It expresses infinity point (i.e. [0]).

a1, a2, a3, a4, a6 :
It expresses the coefficients `a1`, `a2`, `a3`, `a4`, `a6`.

b2, b4, b6, b8 :
It expresses the coefficients `b2`, `b4`, `b6`, `b8`.

c4, c6 :
It expresses the coefficients `c4`, `c6`.

disc :
It expresses the discriminant of `self`.

j :

It expresses the j-invariant of self.

coefficient :

It expresses the **weierstrassform** of self.

Methods

1.1.1.1 simple – simplify the curve coefficient

simple(self) → ECGeneric

Return elliptic curve corresponding to the short Weierstrass form of self by changing the coordinates.

1.1.1.2 changeCurve – change the curve by coordinate change

changeCurve(self, `V`: *list*) → ECGeneric

Return elliptic curve corresponding to the curve obtained by some coordinate change $x = u^2x' + r,\ y = u^3y' + su^2x' + t$.

For $u \neq 0$, the coordinate change gives some curve which is **basefield**-isomorphic to self.

V must be a list of the form $[u, r, s, t]$, where u, r, s, t are in **basefield**.

1.1.1.3 changePoint – change coordinate of point on the curve

changePoint(self, `P`: point, `V`: *list*) → point

Return the point corresponding to the point obtained by the coordinate change $x' = (x - r)u^{-2},\ y' = (y - s(x - r) + t)u^{-3}$.

Note that the inverse coordinate change is $x = u^2x' + r,\ y = u^3y' + su^2x' + t$.See **changeCurve**.

V must be a list of the form $[u, r, s, t]$, where u, r, s, t are in **basefield**.u must be non-zero.

1.1.1.4 coordinateY – Y-coordinate from X-coordinate

coordinateY(self, `x`: FieldElement) → FieldElement / *False*

Return Y-coordinate of the point on self whose X-coordinate is `x`.

The output would be one Y-coordinate (if a coordinate is found). If such a Y-coordinate does not exist, it returns False.

1.1.1.5 whetherOn – Check point is on curve

whetherOn(self, `P`: point) → *bool*

Check whether the point `P` is on self or not.

1.1.1.6 add – Point addition on the curve

add(self, `P`: point, `Q`: point) → point

Return the sum of the point `P` and `Q` on self.

1.1.1.7 sub – Point subtraction on the curve

sub(self, `P`: point, `Q`: point) → point

Return the subtraction of the point `P` from `Q` on self.

1.1.1.8 mul – Scalar point multiplication on the curve

mul(self, `k`: *integer*, `P`: point) → point

Return the scalar multiplication of the point `P` by a scalar `k` on self.

1.1.1.9 divPoly – division polynomial

divPoly(self, `m`: *integer*=None) → FieldPolynomial/(*f*: list, *H*: integer)

Return the division polynomial.

If `m` is odd, this method returns the usual division polynomial. If `m` is even, return the quotient of the usual division polynomial by $2y + a_1x + a_3$.
†If `m` is not specified (i.e. `m`=None), then return (`f`, `H`). `H` is the least prime satisfying $\prod_{2 \le l \le H,\ l:prime} l > 4\sqrt{q}$, where q is the order of **basefield**. `f` is the list of k-division polynomials up to $k \le$ `H`. These are used for Schoof's algorithm.

1.1.2 ECoverQ – elliptic curve over rational field

The class is for elliptic curves over the rational field $\mathbb{Q}$ (**RationalField** in nzmath.**rational**).
The class is a subclass of **ECGeneric**.

Initialize (Constructor)

ECoverQ(coefficient: weierstrassform) → ECoverQ

Create elliptic curve over the rational field.

All elements of coefficient must be integer or **Rational**.
See **weierstrassform** for more information about coefficient.

Examples

```
>>> E = elliptic.ECoverQ([ratinal.Rational(1, 2), 3])
>>> print E.disc
-3896/1
>>> print E.j
1728/487
```

Methods

1.1.2.1 point – obtain random point on curve

point(self, limit: *integer*=1000) → point

Return a random point on self.

limit expresses the time of trying to choose points. If failed, raise ValueError. †Because it is difficult to search the rational point over the rational field, it might raise error with high frequency.

Examples

```
>>> print E.changeCurve([1, 2, 3, 4])
y ** 2 + 6/1 * x * y + 8/1 * y = x ** 3 - 3/1 * x ** 2 - 23/2 * x - 4/1
>>> E.divPoly(3)
FieldPolynomial([(0, Rational(-1, 4)), (1, Rational(36, 1)), (2, Rational(3, 1)
), (4, Rational(3, 1))], RationalField())
```

1.1.3 ECoverGF – elliptic curve over finite field

The class is for elliptic curves over a finite field, denoted by $\mathbb{F}_q$ (**FiniteField** and its subclasses in nzmath).
The class is a subclass of **ECGeneric**.

Initialize (Constructor)

ECoverGF(**coefficient: weierstrassform**, **basefield: FiniteField**)
→ **ECoverGF**

Create elliptic curve over a finite field.

All elements of coefficient must be in basefield. basefield should be an instance of **FiniteField**.
See **weierstrassform** for more information about coefficient.

Examples

```
>>> E = elliptic.ECoverGF([2, 5], finitefield.FinitePrimeField(11))
>>> print E.j
7 in F_11
>>> E.whetherOn([8, 4])
True
>>> E.add([3, 4], [9, 9])
[FinitePrimeFieldElement(0, 11), FinitePrimeFieldElement(4, 11)]
>>> E.mul(5, [9, 9])
[FinitePrimeFieldElement(0, 11)]
```

Methods

1.1.3.1 point – find random point on curve

point(self) → point

Return a random point on self.

This method uses a probabilistic algorithm.

1.1.3.2 naive – Frobenius trace by naive method

naive(self) → *integer*

Return Frobenius trace t by a naive method.

†The function counts up the Legendre symbols of all rational points on self. Frobenius trace of the curve is t such that $\#E(\mathbb{F}_q) = q + 1 - t$, where $\#E(\mathbb{F}_q)$ stands for the number of points on self over self.**basefield** $\mathbb{F}_q$.

The characteristic of self.**basefield** must not be 2 nor 3.

1.1.3.3 Shanks_Mestre – Frobenius trace by Shanks and Mestre method

Shanks_Mestre(self) → *integer*

Return Frobenius trace t by Shanks and Mestre method.

†This uses the method proposed by Shanks and Mestre. †See Algorithm 7.5.3 of [1] for more information about the algorithm.
Frobenius trace of the curve is t such that $\#E(\mathbb{F}_q) = q + 1 - t$, where $\#E(\mathbb{F}_q)$ stands for the number of points on self over self.**basefield** $\mathbb{F}_q$.

self.**basefield** must be an instance of **FinitePrimeField**.

1.1.3.4 Schoof – Frobenius trace by Schoof's method

Schoof(self) → *integer*

Return Frobenius trace t by Schoof's method.

†This uses the method proposed by Schoof.

Frobenius trace of the curve is t such that $\#E(\mathbb{F}_q) = q + 1 - t$, where $\#E(\mathbb{F}_q)$ stands for the number of points on `self` over self.**basefield** $\mathbb{F}_q$.

1.1.3.5 trace – Frobenius trace

trace(self, `r`: *integer*=None) → *integer*

Return Frobenius trace t.

Frobenius trace of the curve is t such that $\#E(\mathbb{F}_q) = q + 1 - t$, where $\#E(\mathbb{F}_q)$ stands for the number of points on `self` over self.**basefield** $\mathbb{F}_q$.
If positive r given, it returns $q^r + 1 - \#E(\mathbb{F}_{q^r})$.
†The method selects algorithms by investigating `self`.**ch** when `self`.**basefield** is an instance of **FinitePrimeField**. If `ch`<1000, the method uses **naive**. If $10^4 <$ `ch` $< 10^{30}$, the method uses **Shanks_Mestre**. Otherwise, it uses **Schoof**.

The parameter r must be positive integer.

1.1.3.6 order – order of group of rational points on the curve

order(self, `r`: *integer*=None) → *integer*

Return order $\#E(\mathbb{F}_q) = q + 1 - t$.

If positive r given, this computes $\#E({\mathbb{F}_q}^r)$ instead.
†On the computation of Frobenius trace t, the method calls **trace**.

The parameter r must be positive integer.

1.1.3.7 pointorder – order of point on the curve

pointorder(self, `P`: point, `ord_factor`: *list*=None) → *integer*

Return order of a point `P`.

†The method uses factorization of **order**.
If `ord_factor` is given, computation of factorizing the order of `self` is omitted and it applies `ord_factor` instead.

1.1.3.8 TatePairing – Tate Pairing

TatePairing(self, `m`: *integer*, `P`: point, `Q`: point) → FiniteFieldElement

Return Tate-Lichetenbaum pairing $\langle P,\ Q\rangle_m$.

†The method uses Miller's algorithm.
The image of the Tate pairing is $\mathbb{F}_q^*/\mathbb{F}_q^{*m}$, but the method returns an element of $\mathbb{F}_q$, so the value is not uniquely defined. If uniqueness is needed, use **TatePairing_Extend**.

The point `P` has to be a `m`-torsion point (i.e. mP =[0]). Also, the number `m` must divide **order**.

1.1.3.9 TatePairing_Extend – Tate Pairing with final exponentiation

TatePairing_Extend(self, `m`: *integer*, `P`: point, `Q`: point) → FiniteFieldElement

Return Tate Pairing with final exponentiation, i.e. $\langle P,\ Q\rangle_m{}^{(q-1)/m}$.

†The method calls **TatePairing**.

The point `P` has to be a `m`-torsion point (i.e. mP =[0]). Also the number `m` must divide **order**.
The output is in the group generated by m-th root of unity in $\mathbb{F}_q^*$.

1.1.3.10 WeilPairing – Weil Pairing

WeilPairing(self, `m`: *integer*, `P`: point, `Q`: point) → FiniteFieldElement

Return Weil pairing $e_m(P,\ Q)$.

†The method uses Miller's algorithm.

The points `P` and `Q` has to be a `m`-torsion point (i.e. $mP = mQ$ =[0]). Also, the number `m` must divide **order**.
The output is in the group generated by m-th root of unity in $\mathbb{F}_q^*$.

1.1.3.11 BSGS – point order by Baby-Step and Giant-Step

BSGS(self, `P`: point) → *integer*

Return order of point P by Baby-Step and Giant-Step method.

†See [2] for more information about the algorithm.

1.1.3.12 DLP_BSGS – solve Discrete Logarithm Problem by Baby-Step and Giant-Step

DLP_BSGS(self, `n`: *integer*, `P`: point, `Q`: point) → *m:* integer

Return `m` such that $Q = mP$ by Baby-Step and Giant-Step method.

The points `P` and `Q` has to be a `n`-torsion point (i.e. $nP = nQ =$[0]). Also, the number `n` must divide **order**.
The output `m` is an integer.

1.1.3.13 structure – structure of group of rational points

structure(self) → *structure:* tuple

Return the group structure of self.

The structure of $E(\mathbb{F}_q)$ is represented as $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$. The method uses **WeilPairing**.

The output structure is a tuple of positive two integers (d, n). `d` divides `n`.

1.1.3.14 issupersingular – check supersingular curve

structure(self) → *bool*

Check whether self is a supersingular curve or not.

Examples

```
>>> E=nzmath.elliptic.ECoverGF([2, 5], nzmath.finitefield.FinitePrimeField(11))
>>> E.whetherOn([0, 4])
True
>>> print E.coordinateY(3)
4 in F_11
>>> E.trace()
2
>>> E.order()
10
```

```
>>> E.pointorder([3, 4])
10L
>>> E.TatePairing(10, [3, 4], [9, 9])
FinitePrimeFieldElement(3, 11)
>>> E.DLP_BSGS(10, [3, 4], [9, 9])
6
```

1.1.4 EC(function)

EC(`coefficient`: weierstrassform, `basefield`: Field)
→ ECGeneric

Create an elliptic curve object.

All elements of `coefficient` must be in `basefield`.
`basefield` must be **RationalField** or **FiniteField** or their subclasses. See also **weierstrassform** for `coefficient`.

Bibliography

[1] Richard Crandall and Carl Pomerance. *Prime Numbers*. Springer, 1st. edition, 2001.

[2] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. DISCRETE MATHEMATICS AND ITS APPLICATIONS. CRC Press, 1st. edition, 2003.