

Contents

1	Classes	3
1.1	algfield – Algebraic Number Field	3
1.1.1	NumberField – number field	3
1.1.1.1	getConj – roots of polynomial	5
1.1.1.2	disc – polynomial discriminant	5
1.1.1.3	integer_ring – integer ring	5
1.1.1.4	field_discriminant – discriminant	5
1.1.1.5	basis – standard basis	5
1.1.1.6	signature – signature	6
1.1.1.7	POLRED – polynomial reduction	6
1.1.1.8	isIntBasis – check integral basis	6
1.1.1.9	isGaloisField – check Galois field	6
1.1.1.10	isFieldElement – check field element	6
1.1.1.11	getCharacteristic – characteristic	7
1.1.1.12	createElement – create an element	7
1.1.2	BasicAlgNumber – Algebraic Number Class by standard basis	8
1.1.2.1	inverse – inverse	10
1.1.2.2	getConj – roots of polynomial	10
1.1.2.3	getApprox – approximate conjugates	10
1.1.2.4	getCharPoly – characteristic polynomial	10
1.1.2.5	getRing – the field	10
1.1.2.6	trace – trace	10
1.1.2.7	norm – norm	11
1.1.2.8	isAlgInteger – check (algebraic) integer	11
1.1.2.9	ch_matrix – obtain MatAlgNumber object	11
1.1.3	MatAlgNumber – Algebraic Number Class by matrix representation	12
1.1.3.1	inverse – inverse	14
1.1.3.2	getRing – the field	14
1.1.3.3	trace – trace	14
1.1.3.4	norm – norm	14
1.1.3.5	ch_basic – obtain BasicAlgNumber object	14
1.1.4	changetype(function) – obtain BasicAlgNumber object	16

1.1.5	disc(function) – discriminant	16
1.1.6	fppoly(function) – polynomial over finite prime field	16
1.1.7	qpoly(function) – polynomial over rational field	16
1.1.8	zpoly(function) – polynomial over integer ring	17

Chapter 1

Classes

1.1 `algfield` – Algebraic Number Field

- **Classes**
 - `NumberField`
 - `BasicAlgNumber`
 - `MatAlgNumber`
- **Functions**
 - `changetype`
 - `disc`
 - `fppoly`
 - `qpoly`
 - `zpoly`

1.1.1 `NumberField` – number field

Initialize (Constructor)

`NumberField(f: list, precompute: bool=False) → NumberField`

Create `NumberField` object.

This field defined by the polynomial `f`.
The class inherits `Field`.

`f`, which expresses coefficients of a polynomial, must be a list of integers. `f` should be written in ascending order. `f` must be monic irreducible over rational

field.

If `precompute` is `True`, all solutions of `f` (by `getConj`), the discriminant of `f` (by `disc`), the signature (by `signature`) and the field discriminant of the basis of the integer ring (by `integer_ring`) are precomputed.

Attributes

degree : The (absolute) extension degree of the number field.

polynomial : The defining polynomial of the number field.

Operations

operator	explanation
<code>K * F</code>	Return the composite field of <code>K</code> and <code>F</code> .
<code>K == F</code>	Check whether the equality of <code>K</code> and <code>F</code> .

Examples

```
>>> K = algfield.NumberField([-2, 0, 1])
>>> L = algfield.NumberField([-3, 0, 1])
>>> print K, L
NumberField([-2, 0, 1]) NumberField([-3, 0, 1])
>>> print K * L
NumberField([1L, 0L, -10L, 0L, 1L])
```

Methods

1.1.1.1 `getConj` – roots of polynomial

`getConj(self)` → *list*

Return all (approximate) roots of the `self.polynomial`.

The output is a list of (approximate) complex number.

1.1.1.2 `disc` – polynomial discriminant

`disc(self)` → *integer*

Return the (polynomial) discriminant of the `self.polynomial`.

†The output is not discriminant of the number field itself.

1.1.1.3 `integer_ring` – integer ring

`integer_ring(self)` → **FieldSquareMatrix**

Return a basis of the ring of integers of `self`.

†The function uses **round2**.

1.1.1.4 `field_discriminant` – discriminant

`field_discriminant(self)` → **Rational**

Return the field discriminant of `self`.

†The function uses **round2**.

1.1.1.5 `basis` – standard basis

`basis(self, j: integer)` → **BasicAlgNumber**

Return the j -th basis (over the rational field) of `self`.

Let θ be a solution of `self.polynomial`. Then θ^j is a part of basis of `self`, so

the method returns them. This basis is called “standard basis” or “power basis”.

1.1.1.6 `signature` – signature

`signature(self)` \rightarrow *list*

Return the signature of `self`.

†The method uses Strum’s algorithm.

1.1.1.7 `POLRED` – polynomial reduction

`POLRED(self)` \rightarrow *list*

Return some polynomials defining subfields of `self`.

†“POLRED” means “polynomial reduction”. That is, it finds polynomials whose coefficients are not so large.

1.1.1.8 `isIntBasis` – check integral basis

`isIntBasis(self)` \rightarrow *bool*

Check whether power basis of `self` is also an integral basis of the field.

1.1.1.9 `isGaloisField` – check Galois field

`isGaloisField(self)` \rightarrow *bool*

Check whether the extension `self` over the rational field is Galois.

†As it stands, it only checks the signature.

1.1.1.10 `isFieldElement` – check field element

`isFieldElement(self, A: BasicAlgNumber/MatAlgNumber)`
 \rightarrow *bool*

Check whether `A` is an element of the field `self`.

1.1.1.11 `getCharacteristic` – characteristic

`getCharacteristic(self) → integer`

Return the characteristic of `self`.

It returns always zero. The method is only for ensuring consistency.

1.1.1.12 `createElement` – create an element

`createElement(self, seed: list) → BasicAlgNumber/MatAlgNumber`

Return an element of `self` with `seed`.

`seed` determines the class of returned element.

For example, if `seed` forms as $[[e_1, e_2, \dots, e_n], d]$, then it calls **BasicAlgNumber**.

Examples

```
>>> K = algfield.NumberField([3, 0, 1])
>>> K.getConj()
[-1.7320508075688774j, 1.7320508075688772j]
>>> K.disc()
-12L
>>> print K.integer_ring()
1/1 1/2
0/1 1/2
>>> K.field_discriminant()
Rational(-3, 1)
>>> K.basis(0), K.basis(1)
BasicAlgNumber([[1, 0], 1], [3, 0, 1]) BasicAlgNumber([[0, 1], 1], [3, 0, 1])
>>> K.signature()
(0, 1)
>>> K.POLRED()
[IntegerPolynomial([(0, 4L), (1, -2L), (2, 1L)], IntegerRing()),
IntegerPolynomial([(0, -1L), (1, 1L)], IntegerRing())]
>>> K.isIntBasis()
False
```

1.1.2 BasicAlgNumber – Algebraic Number Class by standard basis

Initialize (Constructor)

```
BasicAlgNumber( valuelist: list, polynomial: list, precompute:
bool=False )
    → BasicAlgNumber
```

Create an algebraic number with standard (power) basis.

`valuelist` = $[[e_1, e_2, \dots, e_n], d]$ means $\frac{1}{d}(e_1 + e_2\theta + e_3\theta^2 + \dots + e_n\theta^{n-1})$, where θ is a solution of the polynomial `polynomial`. Note that $\langle \theta^i \rangle$ is a (standard) basis of the field defining by `polynomial` over the rational field.

e_i, d must be integers. Also, `polynomial` should be list of integers. If `precompute` is True, all solutions of `polynomial` (by `getConj`), approximation values of all conjugates of `self` (by `getApprox`) and a polynomial which is a solution of `self` (by `getCharPoly`) are precomputed.

Attributes

value : The list of numerators (the integer part) and the denominator of `self`.

coeff : The coefficients of numerators (the integer part) of `self`.

denom : The denominator of the algebraic number for standard basis.

degree : The degree of extension of the field over the rational field.

polynomial : The defining polynomial of the field.

field : The number field in which `self` is.

Operations

operator	explanation
<code>a + b</code>	Return the sum of <code>a</code> and <code>b</code> .
<code>a - b</code>	Return the subtraction of <code>a</code> and <code>b</code> .
<code>- a</code>	Return the negation of <code>a</code> .
<code>a * b</code>	Return the product of <code>a</code> and <code>b</code> .
<code>a ** k</code>	Return the <code>k</code> -th power of <code>a</code> .
<code>a / b</code>	Return the quotient of <code>a</code> by <code>b</code> .

Examples

```
>>> a = algfield.BasicAlgNumber([[1, 1], 1], [-2, 0, 1])
>>> b = algfield.BasicAlgNumber([[-1, 2], 1], [-2, 0, 1])
>>> print a + b
BasicAlgNumber([[0, 3], 1], [-2, 0, 1])
>>> print a * b
BasicAlgNumber([[3L, 1L], 1], [-2, 0, 1])
>>> print a ** 3
BasicAlgNumber([[7L, 5L], 1], [-2, 0, 1])
>>> a // b
BasicAlgNumber([[5L, 3L], 7L], [-2, 0, 1])
```

Methods

1.1.2.1 `inverse` – `inverse`

`inverse(self) → BasicAlgNumber`

Return the inverse of `self`.

1.1.2.2 `getConj` – roots of polynomial

`getConj(self) → list`

Return all (approximate) roots of `self.polynomial`.

1.1.2.3 `getApprox` – approximate conjugates

`getApprox(self) → list`

Return all (approximate) conjugates of `self`.

1.1.2.4 `getCharPoly` – characteristic polynomial

`getCharPoly(self) → list`

Return the characteristic polynomial of `self`.

†`self` is a solution of the characteristic polynomial.

The output is a list of integers.

1.1.2.5 `getRing` – the field

`getRing(self) → NumberField`

Return the field which `self` belongs to.

1.1.2.6 `trace` – `trace`

`trace(self) → Rational`

Return the trace of `self` in the `self.field` over the rational field.

1.1.2.7 `norm` – `norm`

`norm(self) → Rational`

Return the norm of `self` in the `self.field` over the rational field.

1.1.2.8 `isAlgInteger` – check (algebraic) integer

`isAlgInteger(self) → bool`

Check whether `self` is an (algebraic) integer or not.

1.1.2.9 `ch_matrix` – obtain `MatAlgNumber` object

`ch_matrix(self) → MatAlgNumber`

Return `MatAlgNumber` object corresponding to `self`.

Examples

```
>>> a = algfield.BasicAlgNumber([[1, 1], 1], [-2, 0, 1])
>>> a.inverse()
BasicAlgNumber([[-1L, 1L], 1L], [-2, 0, 1])
>>> a.getConj()
[(1.4142135623730951+0j), (-1.4142135623730951+0j)]
>>> a.getApprox()
[(2.4142135623730949+0j), (-0.41421356237309515+0j)]
>>> a.getCharPoly()
[-1, -2, 1]
>>> a.getRing()
NumberField([-2, 0, 1])
>>> a.trace(), a.norm()
2 -1
>>> a.isAlgInteger()
True
>>> a.ch_matrix()
MatAlgNumber([1, 1]+[2, 1], [-2, 0, 1])
```

1.1.3 MatAlgNumber – Algebraic Number Class by matrix representation

Initialize (Constructor)

```
MatAlgNumber( coefficient: list, polynomial: list )  
→ MatAlgNumber
```

Create an algebraic number represented by a matrix.

“matrix representation” means the matrix A over the rational field such that $(e_1 + e_2\theta + e_3\theta^2 + \dots + e_n\theta^{n-1})(1, \theta, \dots, \theta^{n-1})^T = A(1, \theta, \dots, \theta^{n-1})^T$, where t expresses transpose operation.

coefficient = $[e_1, e_2, \dots, e_n]$ means $e_1 + e_2\theta + e_3\theta^2 + \dots + e_n\theta^{n-1}$, where θ is a solution of the polynomial **polynomial**. Note that $\langle \theta^i \rangle$ is a (standard) basis of the field defining by **polynomial** over the rational field. **coefficient** must be a list of (not only integers) rational numbers. **polynomial** must be a list of integers.

Attributes

coeff : The coefficients of the algebraic number for standard basis.

degree : The degree of extension of the field over the rational field.

matrix : The representation matrix of the algebraic number.

polynomial : The defining polynomial of the field.

field : The number field in which **self** is.

Operations

operator	explanation
a + b	Return the sum of a and b .
a - b	Return the subtraction of a and b .
- a	Return the negation of a .
a * b	Return the product of a and b .
a ** k	Return the k-th power of a .
a / b	Return the quotient of a by b .

Examples

```
>>> a = algfield.MatAlgNumber([1, 2], [-2, 0, 1])
>>> b = algfield.MatAlgNumber([-2, 3], [-2, 0, 1])
>>> print a + b
MatAlgNumber([-1, 5]+[10, -1], [-2, 0, 1])
>>> print a * b
MatAlgNumber([10, -1]+[-2, 10], [-2, 0, 1])
>>> print a ** 3
MatAlgNumber([25L, 22L]+[44L, 25L], [-2, 0, 1])
>>> print a / b
MatAlgNumber([Rational(1, 1), Rational(1, 2)]+
[Rational(1, 1), Rational(1, 1)], [-2, 0, 1])
```

Methods

1.1.3.1 inverse – inverse

`inverse(self)` → *MatAlgNumber*

Return the inverse of `self`.

1.1.3.2 getRing – the field

`getRing(self)` → *NumberField*

Return the field which `self` belongs to.

1.1.3.3 trace – trace

`trace(self)` → *Rational*

Return the trace of `self` in the `self.field` over the rational field.

1.1.3.4 norm – norm

`norm(self)` → *Rational*

Return the norm of `self` in the `self.field` over the rational field.

1.1.3.5 ch_basic – obtain BasicAlgNumber object

`ch_basic(self)` → *BasicAlgNumber*

Return **BasicAlgNumber** object corresponding to `self`.

Examples

```
>>> a = algfield.MatAlgNumber([1, -1, 1], [-3, 1, 2, 1])
>>> a.inverse()
MatAlgNumber([Rational(2, 3), Rational(4, 9), Rational(1, 9)]+
[Rational(1, 3), Rational(5, 9), Rational(2, 9)]+
[Rational(2, 3), Rational(1, 9), Rational(1, 9)], [-3, 1, 2, 1])
>>> a.trace()
Rational(7, 1)
```

```
>>> a.norm()
Rational(27, 1)
>>> a.getRing()
NumberField([-3, 1, 2, 1])
>>> a.ch_basic()
BasicAlgNumber([[1, -1, 1], 1], [-3, 1, 2, 1])
```

1.1.4 `changetype(function)` – obtain `BasicAlgNumber` object

`changetype(a: integer, polynomial: list=[0, 1]) → BasicAlgNumber`

`changetype(a: Rational, polynomial: list=[0, 1]) → BasicAlgNumber`

`changetype(polynomial: list) → BasicAlgNumber`

Return a `BasicAlgNumber` object corresponding to `a`.

If `a` is an integer or an instance of `Rational`, the function returns `BasicAlgNumber` object whose field is defined by `polynomial`. If `a` is a list, the function returns `BasicAlgNumber` corresponding to a solution of `a`, considering `a` as the polynomial.

The input parameter `a` must be an integer, `Rational` or a list of integers.

1.1.5 `disc(function)` – discriminant

`disc(A: list) → Rational`

Return the discriminant of a_i , where $A = [a_1, a_2, \dots, a_n]$.

a_i must be an instance of `BasicAlgNumber` or `MatAlgNumber` defined over a same number field.

1.1.6 `fppoly(function)` – polynomial over finite prime field

`fppoly(coeffs: list, p: integer) → FinitePrimeFieldPolynomial`

Return the polynomial whose coefficients `coeffs` are defined over the prime field \mathbb{Z}_p .

`coeffs` should be a list of integers or of instances of `FinitePrimeFieldElement`.

1.1.7 `qpoly(function)` – polynomial over rational field

`qpoly(coeffs: list) → FieldPolynomial`

Return the polynomial whose coefficients `coeffs` are defined over the rational

field.

`coeffs` must be a list of integers or instances of **Rational**.

1.1.8 `zpoly(function)` – polynomial over integer ring

`zpoly(coeffs: list) → IntegerPolynomial`

Return the polynomial whose coefficients `coeffs` are defined over the (rational) integer ring.

`coeffs` must be a list of integers.

Examples

```
>>> a = algfield.changetype(3, [-2, 0, 1])
>>> b = algfield.BasicAlgNumber([[1, 2], 1], [-2, 0, 1])
>>> A = [a, b]
>>> algfield.disc(A)
288L
```