Contents

Chapter 1

Functions

1.1 module – HNF による加群/イデアル

- Classes
  - Submodule
  - Module
  - Ideal
  - Ideal_with_generator

1.1.1 Submodule – 行列表現としての部分加群

Initialize (Constructor)

Submodule(`row`: ***integer***, `column`: ***integer***, `compo`: ***compo***=**0**, `coeff_ring`: ***CommutativeRing***=**0**, `ishnf`: ***True/False***=**None**)
→ ***Submodule***

行列表現で新しい部分加群を作成.

Submodule は **RingMatrix** のサブクラス.
`coeff_ring` は PID(単項イデアル整域) と仮定. その後行列に対応する HNF(Hermite 正規形) を得る.

`ishnf` が True なら入力する行列は HNF と仮定.

Attributes

ishnf もし行列が HNF なら `ishnf` は True, そうでなければ False.

Methods

1.1.1.1 getGenerators – 加群の生成元

getGenerators(`self`) → ***list***

加群 `self` の (現在の) 生成元を返す.

生成元から成るベクトルのリストを返す.

1.1.1.2 isSubmodule – 部分加群かどうか

isSubmodule(`self`, `other`: ***Submodule***) → ***True/False***

部分加群インスタンスが `other` の部分加群なら True, そうでなければ False を返す.

1.1.1.3 isEqual – self と other が同じ加群かどうか

isEqual(`self`, `other`: ***Submodule***) → ***True/False***

部分加群インスタンスが加群として `other` と等しいなら True, そうでなければ False を返す.

.

このメソッドは行列でない加群の等式テストにも使用したほうがよい. 行列の等式テストには単純に `self==other` を使用.

1.1.1.4 isContain – other が self に含まれているかどうか

isContains(`self`, `other`: ***vector.Vector***) → ***True/False***

`other` が `self` に含まれているかどうか返す.

.

もし `other` を `self` の HNF 生成元の一次結合として表したい場合,**represent_element** を使用.

1.1.1.5 toHNF - HNF に変換

toHNF(`self`) → *(None)*

`self` を HNF(Hermite 正規形) に変換し,`ishnf` に True を設定.

HNF は常に `self` の基底を与えるわけではないことに注意.(HNF は冗長なことがある.)

1.1.1.6 sumOfSubmodules - 部分加群の和

sumOfSubmodules(`self`, `other`: *Submodule*) → *Submodule*

二つの部分空間の和である加群を返す.

1.1.1.7 intersectionOfSubmodules - 部分加群の共通部分

intersectionOfSubmodules(`self`, `other`: *Submodule*)
→ *Submodule*

二つの部分空間の共通部分である加群を返す.

1.1.1.8 represent_element – 一次結合として成分を表す

represent_element(`self`, `other`: *vector.Vector*) → *vector.Vector/False*

`other` を HNF 生成元の一次結合として表す.

`other` が `self` に含まれていなければ,False を返す. このメソッドは **toHNF** を呼ぶことに注意.

このメソッドは **Vector** のインスタンスとしての係数を返す.

1.1.1.9 linear_combination – 一次結合を計算

linear_combination(`self`, `coeff`: *list*) → *vector.Vector*

Z 係数 `coeff` が与えられ,(現在) の基底の一次結合に対応するベクトルを返す.

`coeff` はサイズが `self` の列と等しい **RingElement** 上のインスタンスのリスト.

Examples

```
>>> A = module.Submodule(4, 3, [1,2,3]+[4,5,6]+[7,8,9]+[10,11,12])
>>> A.toHNF()
>>> print A
9 1
6 1
3 1
0 1
>>> A.getGenerator
[Vector([9L, 6L, 3L, 0L]), Vector([1L, 1L, 1L, 1L])]
>>> V = vector.Vector([10,7,4,1])
>>> A.represent_element(V)
Vector([1L, 1L])
>>> V == A.linear_combination([1,1])
True
>>> B = module.Submodule(4, 1, [1,2,3,4])
>>> C = module.Submodule(4, 2, [2,-4]+[4,-3]+[6,-2]+[8,-1])
>>> print B.intersectionOfSubmodules(C)
2
4
6
8
```

1.1.2 fromMatrix(class function) - 部分加群を作成

fromMatrix(`cls`, `mat`: ***RingMatrix***, `ishnf`: ***True/False***=**None)**
→ ***Submodule***

クラスが Matrix のサブクラスになり得る行列のインスタンス `mat` から Submodule のインスタンスを作成.

Submodule のインスタンスがほしい場合このメソッドを使用.

1.1.3 Module - 数体上の加群

Initialize (Constructor)

Module(`pair_mat_repr`: *list/matrix*, `number_field`: *algfield.NumberField*, `base`: *list/matrix.SquareMatrix*=**None**, `ishnf`: *bool*=**False**)
→ ***Module***

数体上の新しい加群オブジェクトを作成.

加群は有限生成された部分 **Z** 加群. 加群の階数を deg(number_field) と仮定しないことを注意.
加群を,θ が `number_field.`**polynomial** の解となる $\mathbf{Z}[\theta]$ 上の基本的な加群についての生成元として表す.

`pair_mat_repr` は次に示す形式のどれかであるべきである:

- $[M,\ d]$, M のサイズは `number_field` の次数である整数のタプルまたはベクトルのリストであり,d は分母.
- $[M,\ d]$, M のサイズは `number_field` の次数である整数行列, であり d は分母.
- 行の数は `number_field` の次数である有理数行列.

また,`base` は次に示す形式のうちのどれかであるべきである:

- サイズは `number_field` の次数である有理数のタプルまたはベクトルのリスト
- サイズは `number_field` である非特異かつ有理数係数の正方行列

加群は **base** について内部で $\frac{1}{d}M$ と表され, d は **denominator** で M は **mat_repr**.
`ishnf` が True なら,`mat_repr` は HNF であると仮定.

Attributes

mat_repr :サイズが `number_field` の次数である **Submodule** M のインスタンス

denominator :整数 d

base :サイズは `number_field` である正方かつ非特異有理数行列

number_field :加群が定義された数体

operator	explanation
`M==N`	`M` と `N` が加群として等しいかどうか返す.
`c in M`	`M` の要素のどれかが `c` と等しいかどうか返す.
`M+N`	`M` と `N` の加群としての部分集合を返す.
`M*N`	`M` と `N` のイデアル計算としての積を返す.
	`N` は加群またはスカラーでなければならない (**number_field** の要素).
	M と N の共通部分の計算したいときは **intersect** を参照.
`M**c`	イデアルの乗算を基にした `M` の `c` 乗を返す.
`repr(M)`	加群 `M` の repr 文字列を返す.
`str(M)`	加群 `M` の string 文字列を返す.

Operations

Examples

```
>>> F = algfield.NumberField([2,0,1])
>>> M_1 = module.Module([matrix.RingMatrix(2,2,[1,0]+[0,2]), 2], F)
>>> M_2 = module.Module([matrix.RingMatrix(2,2,[2,0]+[0,5]), 3], F)
>>> print M_1
([1, 0]+[0, 2], 2)
 over
([1L, 0L]+[0L, 1L], NumberField([2, 0, 1]))
>>> print M_1 + M_2
([1L, 0L]+[0L, 2L], 6)
 over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
>>> print M_1 * 2
([1L, 0L]+[0L, 2L], 1L)
 over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
>>> print M_1 * M_2
([2L, 0L]+[0L, 1L], 6L)
 over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
>>> print M_1 ** 2
([1L, 0L]+[0L, 2L], 4L)
 over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
```

Methods

1.1.3.1 toHNF - Hermite 正規形 (HNF) に変換

toHNF(`self`) → *(None)*

`self`.**mat_repr** を Hermite 正規形 (HNF) に変換.

1.1.3.2 copy - コピーを作成

copy(`self`) → *Module*

`self` のコピーを作成.

1.1.3.3 intersect - 共通部分を返す

intersect(`self`, `other`: *Module*) → *Module*

`self` と `other` の共通部分を返す.

1.1.3.4 issubmodule - 部分加群かどうか

submodule(`self`, `other`: *Module*) → *True/False*

`self` が `other` の部分加群かどうか返す.

1.1.3.5 issupermodule - 部分加群かどうか

supermodule(`self`, `other`: *Module*) → *True/False*

`other` が `self` の部分加群かどうか返す.

1.1.3.6 represent_element - 一次結合として表す

represent_element(`self`, `other`: *algfield.BasicAlgNumber*)
→ *list/False*

other を self で生成される一次結合として表す. もし other が self に含まれていなかったら,False を返す.

self.mat_repr は HNF であると仮定しているわけではないということに注意.
other が self に含まれていたら出力は整数のリスト.

1.1.3.7 change_base_module - 基底変換

change_base_module(self, other_base: *list/matrix.RingSquareMatrix*) → *Module*

other_base に関連した self と等しい加群を返す.

other_base は base の形式に従う.

1.1.3.8 index - 加群のサイズ

index(self) → *rational.Rational*

self.base に関する剰余類の位数を返す. $N \subset M$ なら $[M:N]$ を返し,$M \subset N$ のとき $[N:M]^{-1}$ を返す. M は加群 self で N は self.base に対応する加群.

1.1.3.9 smallest_rational - 有理数体上の Z 生成元

smallest_rational(self) → *rational.Rational*

加群 self と有理数体の共通部分の **Z** 生成元を返す.

Examples

```
>>> F = algfield.NumberField([1,0,2])
>>> M_1=module.Module([matrix.RingMatrix(2,2,[1,0]+[0,2]), 2], F)
>>> M_2=module.Module([matrix.RingMatrix(2,2,[2,0]+[0,5]), 3], F)
>>> print M_1.intersect(M_2)
([2L, 0L]+[0L, 5L], 1L)
 over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
 NumberField([2, 0, 1]))
```

```
>>> M_1.represent_element( F.createElement( [[2,4], 1] ) )
[4L, 4L]
>>> print M_1.change_base_module( matrix.FieldSquareMatrix(2, 2, [1,0]+[0,1]) / 2 )
([1L, 0L]+[0L, 2L], 1L)
 over
([Rational(1, 2), Rational(0, 1)]+[Rational(0, 1), Rational(1, 2)],
 NumberField([2, 0, 1]))
>>> M_2.index()
Rational(10, 9)
>>> M_2.smallest_rational()
Rational(2, 3)
```

1.1.4 Ideal - 数体上のイデアル

Initialize (Constructor)

Ideal(`pair_mat_repr`: *list/matrix*, `number_field`: *algfield.NumberField*, `base`: *list/matrix.SquareMatrix*=**None**, `ishnf`: *bool*=**False**)
→ *Ideal*

数体上の新しいイデアルオブジェクトを作成.

イデアルは **Module** のサブクラスです.

Module も初期化を引用.

Methods

1.1.4.1 inverse – 逆元

inverse(`self`) → ***Ideal***

`self` の逆イデアルを返す.

このメソッドは `self`.number_field.integer_ring を呼び出す.

1.1.4.2 issubideal – 部分イデアルかどうか

issubideal(`self`, `other`: ***Ideal***) → ***Ideal***

`self` が `other` の部分イデアルかどうか返す.

1.1.4.3 issuperideal – 部分加群かどうか

issuperideal(`self`, `other`: ***Ideal***) → ***Ideal***

`other` が `self` の部分加群かどうか返す.

1.1.4.4 gcd – 最大公約数

gcd(`self`, `other`: ***Ideal***) → ***Ideal***

`self` と `other` のイデアルとしての最大公約数 (gcd) を返す.

このメソッドは単純に `self`+`other` を実行する.

1.1.4.5 lcm – 最小公倍数

lcm(`self`, `other`: ***Ideal***) → ***Ideal***

イデアルとしての `self` と `other` の最小公倍数 (lcm) を返す.

このメソッドは単純に intersect を呼び出す.

1.1.4.6 norm – ノルム

norm(`self`) → *rational.Rational*

`self` のノルムを返す.

このメソッドは `self`.number_field.integer_ring を呼び出す.

1.1.4.7 isIntegral – 整イデアルかどうか

isIntegral(`self`) → *True*/*False*

`self` が整イデアルかどうか判定.

Examples

```
>>> M = module.Ideal([matrix.RingMatrix(2, 2, [1,0]+[0,2]), 2], F)
>>> print M.inverse()
([-2L, 0L]+[0L, 2L], 1L)
 over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
 NumberField([2, 0, 1]))
>>> print M * M.inverse()
([1L, 0L]+[0L, 1L], 1L)
 over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
 NumberField([2, 0, 1]))
>>> M.norm()
Rational(1, 2)
>>> M.isIntegral()
False
```

1.1.5 Ideal_with_generator - 生成元によるイデアル

Initialize (Constructor)

Ideal_with_generator(generator: *list*) → ***Ideal_with_generator***

生成元により与えられた新しいイデアルを作成.

generator は同じ数体上の生成元を表す **BasicAlgNumber** のインスタンスのリスト.

Attributes

generator : イデアルの生成元

number_field : 生成元が定義された数体

Operations

operator	explanation
M==N	M と N が加群として等しいかどうか返す.
c in M	M のどれかの要素が c と等しいかどうか返す.
M+N	M と N のイデアル生成元としての和を返す.
M*N	M と N のイデアル生成元としての積を返す.
M**c	イデアルの積を基にした M の c 乗を返す.
repr(M)	イデアル M の repr 文字列を返す.
str(M)	イデアル M の str 文字列を返す.

Examples

```
>>> F = algfield.NumberField([2,0,1])
>>> M_1 = module.Ideal_with_generator([
 F.createElement([[1,0], 2]), F.createElement([[0,1], 1])
])
>>> M_2 = module.Ideal_with_generator([
 F.createElement([[2,0], 3]), F.createElement([[0,5], 3])
])
>>> print M_1
[BasicAlgNumber([[1, 0], 2], [2, 0, 1]), BasicAlgNumber([[0, 1], 1], [2, 0, 1])]
>>> print M_1 + M_2
[BasicAlgNumber([[1, 0], 2], [2, 0, 1]), BasicAlgNumber([[0, 1], 1], [2, 0, 1]),
 BasicAlgNumber([[2, 0], 3], [2, 0, 1]), BasicAlgNumber([[0, 5], 3], [2, 0, 1])]
```

```
>>> print M_1 * M_2
[BasicAlgNumber([[1L, 0L], 3L], [2, 0, 1]), BasicAlgNumber([[0L, 5L], 6], [2, 0, 1]),
BasicAlgNumber([[0L, 2L], 3], [2, 0, 1]), BasicAlgNumber([[-10L, 0L], 3], [2, 0, 1])]
>>> print M_1 ** 2
[BasicAlgNumber([[1L, 0L], 4], [2, 0, 1]), BasicAlgNumber([[0L, 1L], 2], [2, 0, 1]),
BasicAlgNumber([[0L, 1L], 2], [2, 0, 1]), BasicAlgNumber([[-2L, 0L], 1], [2, 0, 1])]
```

Methods

1.1.5.1 copy - コピーを作成

copy(`self`) → ***Ideal_with_generator***

`self` のコピーを作成.

1.1.5.2 to_HNFRepresentation - HNF イデアルに変換

to_HNFRepresentation(`self`) → ***Ideal***

`self` をイデアルに対応した HNF(Hermite 正規形) 表現に変換.

1.1.5.3 twoElementRepresentation - 二つの要素で表す

twoElementRepresentation(`self`) → ***Ideal_with_generator***

`self` をイデアルに対応した HNF(Hermite 正規形) 表現に変換.

`self` が素イデアルでなければ, このメソッドは効果がない.

1.1.5.4 smallest_rational - 有理数体上の Z 生成元

smallest_rational(`self`) → ***rational.Rational***

加群 `self` と有理数体の共通部分の **Z** 生成元を返す.

このメソッドは **to_HNFRepresentation** を呼び出す.

1.1.5.5 inverse – 逆元

inverse(`self`) → ***Ideal***

`self` の逆イデアルを返す.

このメソッドは **to_HNFRepresentation** を呼び出す.

1.1.5.6 norm – ノルム

norm(`self`) → *rational.Rational*

`self` のノルムを返す.

このメソッドは **to_HNFRepresentation** を呼び出す.

1.1.5.7 intersect - 共通部分

intersect(`self`, `other`: *Ideal_with_generator*) → *Ideal*

`self` と `other` の共通部分を返す.

このメソッドは **to_HNFRepresentation** を呼び出す.

1.1.5.8 issubideal – 部分イデアルかどうか

issubideal(`self`, `other`: *Ideal_with_generator*) → *Ideal*

`self` が `other` の部分イデアルかどうか返す.

このメソッドは **to_HNFRepresentation** を呼び出す.

1.1.5.9 issuperideal – 部分イデアルかどうか

issuperideal(`self`, `other`: *Ideal_with_generator*) → *Ideal*

このメソッドは **to_HNFRepresentation** を呼び出す.

Examples

```
>>> M = module.Ideal_with_generator([
F.createElement([[2,0], 3]), F.createElement([[0,2], 3]), F.createElement([[1,0], 3])
])
```

```
>>> print M.to_HNFRepresentation()
([2L, 0L, 0L, -4L, 1L, 0L]+[0L, 2L, 2L, 0L, 0L, 1L], 3L)
 over
([1L, 0L]+[0L, 1L], NumberField([2, 0, 1]))
>>> print M.twoElementRepresentation()
[BasicAlgNumber([[1L, 0], 3], [2, 0, 1]), BasicAlgNumber([[3, 2], 3], [2, 0, 1])]
>>> M.norm()
Rational(1, 9)
```