Contents

Chapter 1

Classes

1.1 matrix – 行列

- **Classes**
  - **Matrix**
  - **SquareMatrix**
  - **RingMatrix**
  - **RingSquareMatrix**
  - **FieldMatrix**
  - **FieldSquareMatrix**
  - **MatrixRing**
  - **Subspace**
- **Functions**
  - **createMatrix**
  - **identityMatrix**
  - **unitMatrix**
  - **zeroMatrix**

matrix モジュールにもいくつかの例外クラスがある.

MatrixSizeError : 入力された行列のサイズが矛盾していると報告.

VectorsNotIndependent : 列ベクトルが一次独立でないと報告.

NoInverseImage : 逆像が存在しないことを報告.

NoInverse : その行列が可逆でないことを報告.

このモジュールは以下のタイプを使うことができる:

compo : `compo` は以下のどれかの形式でなければならない.

- *[1,2]+[3,4]+[5,6]* のような連結された行のリスト.
- *[[1,2], [3,4], [5,6]]* のような行のリストのリスト.
- *[(1, 3, 5), (2, 4, 6)]* のような列のタプルのリスト.
- *[vector.Vector([1, 3, 5]) , vector.Vector([2, 4, 6])]* のような長さの等しい列ベクトルのリスト.

これらの例はすべて以下の行列を表している:

$$\begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix}$$

1.1.1 Matrix – 行列

Initialize (Constructor)

Matrix(`row`: *integer*, `column`: *integer*, `compo`: *compo*=**0**, `coeff_ring`: *CommutativeRing*=**0**)
→ *Matrix*

新しい行列オブジェクトを作成.

† この作成されたオブジェクトは自動的に自身のクラスを次に述べるクラスのうちの一つに変える: **RingMatrix**, **RingSquareMatrix**, **FieldMatrix**, **FieldSquareMatrix**.

入力すると行列のサイズと係数環を調べ自動的に自身のクラスを決定. `row` と `column` は整数,`coeff_ring` は **Ring** のインスタンスでなければならない. `compo` についての情報は **compo** を参照. `compo` を省略すると, 全て 0 のリストであるとみなされる.
予想される入力と出力のリストは以下の通り:

- Matrix(`row`, `column`, `compo`, `coeff_ring`)
  → 成分は `compo`, 係数環は `coeff_ring` である `row`×`column` の行列.
- Matrix(`row`, `column`, `compo`)
  → 成分が `compo` である `row`×`column` の行列 (係数環は自動的に決定).
- Matrix(`row`, `column`, `coeff_ring`)
  → 係数環が `coeff_ring` である `row`×`column` の行列 (すべての成分は `coeff_ring` 上で 0).
- Matrix(`row`, `column`)
  → 係数環は **Integer**, すべての成分は 0 である `row`×`column` の行列.

Attributes

row : 行列の行数.

column : 行列の列数.

coeff_ring : 行列の係数環.

compo : 行列の成分.

operator	explanation
`M==N`	`M` と `N` が等しいかそうでないか返す.
`M[i, j]`	行列 `M` の `i` 行目 `j` 列目の成分を返す.
`M[i]`	行列 `M` の `i` 列目の列ベクトルを返す.
`M[i, j]=c`	行列 `M` の `i` 行目 `j` 列目の成分を `c` に置き換える.
`M[j]=c`	行列 `M` の `i` 列目の列ベクトルを `c` に置き換える.
`c in M`	成分 `c` が行列 `M` に入っているかどうか返す.
`repr(M)`	行列 `M` の repr 文字列を返す. 文字列は行ベクトルのリストの連結リストを表している.
`str(M)`	行列 `M` の string 文字列を返す.

Operations

Examples

```
>>> A = matrix.Matrix(2, 3, [1,0,0]+[0,0,0])
>>> A.__class__.__name__
'RingMatrix'
>>> B = matrix.Matrix(2, 3, [1,0,0,0,0,0])
>>> A == B
True
>>> B[1, 1] = 0
>>> A != B
True
>>> B == 0
True
>>> A[1, 1]
1
>>> print repr(A)
[1, 0, 0]+[0, 0, 0]
>>> print str(A)
1 0 0
0 0 0
```

Methods

1.1.1.1 map – 各成分に関数を適用

map(`self`, `function`: ***function***) → ***Matrix***

各成分に `function` を適用した行列を返す.

†map 関数は組み込み関数である map の類似である.

1.1.1.2 reduce – 繰り返し関数を適用

reduce(`self`, `function`: ***function***, `initializer`: ***RingElement***=None)
→ ***RingElement***

左上から右下に `function` を繰り返し適用する. それにより得られる単一の値を返す

†reduce 関数は組み込み関数である reduce の類似である.

1.1.1.3 copy – コピー作成

copy(`self`) → ***Matrix***

`self` のコピーを作成する.

† この関数によって作成された行列は `self` と等しい行列だが, インスタンスとしては等しいわけではない.

1.1.1.4 set – 成分を設定

set(`self`, `compo`: ***compo***) → ***(None)***

compo として `compo` のリストを設定.

`compo` は **compo** の形式でなければならない.

1.1.1.5 setRow – m 行目に行ベクトルを設定

setRow(`self`, `m`: *integer*, `arg`: *list/Vector*) → *(None)*

リストまたは Vector である `arg` を `m` 行目として設定.

`arg` の長さは `self`.**column** と等しくなければならない.

1.1.1.6 setColumn – n 列目に列ベクトルを設定

setColumn(`self`, `n`: *integer*, `arg`: *list/Vector*) → *(None)*

リストまたは Vector である `arg` を `n` 列目として設定.

`arg` の長さは `self`.**row** と等しくなければならない.

1.1.1.7 getRow – i 行目の行ベクトルを返す

getRow(`self`, `i`: *integer*) → *Vector*

`self` の形式で `i` 行目を返す.

この関数は (**Vector** のインスタンスである) 行ベクトルを返す.

1.1.1.8 getColumn – j 列目の列ベクトルを返す

getColumn(`self`, `j`: *integer*) → *Vector*

`self` の形式で `j` 列目を返す.

この関数は (**Vector** のインスタンスである) 列ベクトルを返す.

1.1.1.9 swapRow – 二つの行ベクトルを交換

swapRow(`self`, `m1`: *integer*, `m2`: *integer*) → *(None)*

`self` の `m1` 行目の行ベクトルと `m2` 行目の行ベクトルを交換.

1.1.1.10 swapColumn – 二つの列ベクトルを交換

swapColumn(`self`, `n1`: *integer*, `n2`: *integer*) → *(None)*

`self` の `n1` 列目の列ベクトルと `n2` 列目の列ベクトルを交換.

1.1.1.11 insertRow – 行ベクトルを挿入

insertRow(`self`, `i`: *integer*, `arg`: *list/Vector/Matrix*)
→ *(None)*

行ベクトル `arg` を `i` 行目 `row` に挿入.

`arg` はリスト,Vector または Matrix でなければならない. その長さ (または column) は `self` の列の長さと等しくするべきである.

1.1.1.12 insertColumn – 列ベクトル挿入

insertColumn(`self`, `j`: *integer*, `arg`: *list/Vector/Matrix*)
→ *(None)*

列ベクトル `arg` を `j` 列目 `column` に挿入.

`arg` は,Vector または Matrix のリストでなければならない. その長さ (または row) は `self` の行の長さと等しくするべきである.

1.1.1.13 extendRow – 行ベクトルを伸張

extendRow(`self`, `arg`: *list/Vector/Matrix*) → *(None)*

`self` に行ベクトル `arg` を結合 (垂直方向に).

この関数は `self` の最後の行ベクトルの次に `arg` を結合. つまり extendRow(`arg`) は insertRow(`self.row+1`, `arg`) と同じ.

`arg` は,Vector または Matrix のリストでなければならない. その長さ (または column)`self` の列と等しくするべきである.

1.1.1.14 extendColumn – 列ベクトルを伸張

extendColumn(`self`, `arg`: *list/Vector/Matrix*) → *(None)*

`self` に列ベクトル `arg` を結合 (水平方向に).

この関数は self の最後の列ベクトルの次に arg を結合. つまり extendColumn(arg) は (self.column+1, arg) と同じ.

arg は Vector または Matrix のリストでなければならない. その長さ (または row) は self の行と等しくするべきである.

1.1.1.15 deleteRow – 行ベクトルを削除

deleteRow(self, i: *integer*) → *(None)*

i 行目の行ベクトルを削除.

1.1.1.16 deleteColumn – 列ベクトルを削除

deleteColumn(self, j: *integer*) → *(None)*

j 列目の列ベクトルを削除.

1.1.1.17 transpose – 転置行列

transpose(self) → *Matrix*

self の転置行列を返す.

1.1.1.18 getBlock – ブロック行列

getBlock(self, i: *integer*, j: *integer*, row: *integer*, column: *integer*=None)
→ *Matrix*

(i, j) 成分からの row×column 行列を返す.

もし column が省略されたら,column は row と同じ値とみなす.

1.1.1.19 subMatrix – 部分行列

subMatrix(self, I: *integer*, J: *integer*None) → *Matrix*

subMatrix(self, I: *list*, J: *list*=None) → *Matrix*

この関数は二つの意味がある.

- I と J は整数:
  I 列目と J 行目を削除した部分行列を返す.
- I と J はリスト:
  列 I と行 J で指定された self の成分から構成された部分行列を返す.

もし J を省略すると,J は I と同じ値とみなす.

Examples

```
>>> A = matrix.Matrix(2, 3, [1,2,3]+[4,5,6])
>>> A
[1, 2, 3]+[4, 5, 6]
>>> A.map(complex)
[(1+0j), (2+0j), (3+0j)]+[(4+0j), (5+0j), (6+0j)]
>>> A.reduce(max)
6
>>> A.swapRow(1, 2)
>>> A
[4, 5, 6]+[1, 2, 3]
>>> A.extendColumn([-2, -1])
>>> A
[4, 5, 6, -2]+[1, 2, 3, -1]
>>> B = matrix.Matrix(3, 3, [1,2,3]+[4,5,6]+[7,8,9])
>>> B.subMatrix(2, 3)
[1, 2]+[7, 8]
>>> B.subMatrix([2, 3], [1, 2])
[4, 5]+[7, 8]
```

1.1.2 SquareMatrix – 正方行列

Initialize (Constructor)

SquareMatrix(`row`: ***integer***, `column`: ***integer***=**0**, `compo`: ***compo***=**0**, `coeff_ring`: ***CommutativeRing***=**0**)
→ ***SquareMatrix***

新しい正方行列オブジェクトを作成.

SquareMatrix は **Matrix** のサブクラス. † この作成されたオブジェクトは自動的に自身のクラスを次に述べるクラスの内のひとつに変える: **RingMatrix**, **RingSquareMatrix**, **FieldMatrix**, **FieldSquareMatrix**.

入力すると行列のサイズと係数環を調べることにより自動的にそのクラスを決定. `row` と `column` は整数,`coeff_ring` は **Ring** のインスタンスでなければならない. `compo` に関する情報は **compo** を参照. `compo` を省略すると, 全て 0 のリストであるとみなされる.

.
予想される入力と出力のリストは以下の通り:

- Matrix(`row`, `compo`, `coeff_ring`)
  → 成分は `compo`, 係数環は `coeff_ring` の `row` 次正方行列
- Matrix(`row`, `compo`)
  → 成分は `compo` の (係数環は自動的に決定)`row` 次正方行列
- Matrix(`row`, `coeff_ring`)
  → 係数環は `coeff_ring` の (すべての成分は `coeff_ring` 上の 0.)`row` 次正方行列
- Matrix(`row`)
  → (係数環は整数. すべての成分は 0.)`row` 次正方行列

†**Matrix** として初期化できるが, その場合 `column` は `row` と同じでなければならない.

Methods

1.1.2.1 isUpperTriangularMatrix – 上三角行列かどうか

isUpperTriangularMatrix(`self`) → ***True/False***

`self` が上三角行列かどうか返す.

1.1.2.2 isLowerTriangularMatrix – 下三角行列かどうか

isLowerTriangularMatrix(`self`) → ***True/False***

`self` が下三角行列かどうか返す.

1.1.2.3 isDiagonalMatrix – 対角行列かどうか

isDiagonalMatrix(`self`) → ***True/False***

`self` が対角行列かどうか返す.

1.1.2.4 isScalarMatrix – スカラー行列かどうか

isScalarMatrix(`self`) → ***True/False***

`self` がスカラー行列かどうか返す.

1.1.2.5 isSymmetricMatrix – 対称行列かどうか

isSymmetricMatrix(`self`) → ***True/False***

`self` が対称行列かどうか返す.

Examples

```
>>> A = matrix.SquareMatrix(3, [1,2,3]+[0,5,6]+[0,0,9])
>>> A.isUpperTriangularMatrix()
```

```
True
>>> B = matrix.SquareMatrix(3, [1,0,0]+[0,-2,0]+[0,0,7])
>>> B.isDiagonalMatrix()
True
```

1.1.3 RingMatrix – 成分が環に属する行列

RingMatrix(row: *integer*, column: *integer*, compo: *compo*=0, coeff_ring: *CommutativeRing*=0)
→ *RingMatrix*

新しい係数が環に属する行列を作成.

RingMatrix は Matrix のサブクラス. 初期化に関する情報は Matrix を参照.

Operations

operator	explanation
M+N	M と N の行列の和を返す.
M-N	M と N の行列の差を返す.
M*N	M と N の行列の積を返す.N は行列, ベクトルまたはスカラーでなければならない
M % d	M を d で割った余りを返す. d は 0 でない整数でなければならい.
-M	各成分が M の符号を変えた成分である行列を返す.
+M	M のコピーを返す.

Examples

```
>>> A = matrix.Matrix(2, 3, [1,2,3]+[4,5,6])
>>> B = matrix.Matrix(2, 3, [7,8,9]+[0,-1,-2])
>>> A + B
[8, 10, 12]+[4, 4, 4]
>>> A - B
[-6, -6, -6]+[4, 6, 8]
>>> A * B.transpose()
[50, -8]+[122, -17]
>>> -B * vector.Vector([1, -1, 0])
Vector([1, -1])
>>> 2 * A
[2, 4, 6]+[8, 10, 12]
>>> B % 3
[1, 2, 0]+[0, 2, 1]
```

Methods

1.1.3.1 getCoefficientRing – 係数環を返す

getCoefficientRing(`self`) → ***CommutativeRing***

`self` の係数環を返す.

このメソッドは `self` の全ての成分を調べ,`coeff_ring` を正しい係数環に設定.

1.1.3.2 toFieldMatrix – 係数環として体を設定

toFieldMatrix(`self`) → ***(None)***

行列のクラスを係数環が現在の整域の商体になるように FieldMatrix または FieldSquareMatrix に変える.

1.1.3.3 toSubspace – ベクトル空間としてみなす

toSubspace(`self`, `isbasis`: ***True/False***=None) → ***(None)***

行列のクラスを係数環が現在の整域の商体になるように `Subspace` に変える.

1.1.3.4 hermiteNormalForm (HNF) – Hermite 正規形

hermiteNormalForm(`self`) → ***RingMatrix***
HNF(`self`) → ***RingMatrix***

上三角行列である Hermite 正規形 (HNF) を返す.

1.1.3.5 exthermiteNormalForm (extHNF) – 拡張 Hermite 正規形アルゴリズム

exthermiteNormalForm(`self`) → ***(RingSquareMatrix, RingMatrix)***
extHNF(`self`) → ***(RingSquareMatrix, RingMatrix)***

Hermite 正規形 `M` と $\texttt{self} \cdot \texttt{U} = \texttt{M}$ を満たす `U` を返す.

この関数は RingSquareMatrix のインスタンスである U と RingMatrix のインスタンスである M のタプルである (U, M) を返す,.

1.1.3.6 kernelAsModule – $\mathbb{Z}$ 加群としての核

kernelAsModule(self) → ***RingMatrix***

$\mathbb{Z}$ 加群としての核を返す.

この関数と kernel の違いは値として返されたそれぞれの値は整数であるということ.

Examples

```
>>> A = matrix.Matrix(3, 4, [1,2,3,4,5,6,7,8,9,-1,-2,-3])
>>> print A.hermiteNormalForm()
0 36 29 28
0  0  1  0
0  0  0  1
>>> U, M = A.hermiteNormalForm()
>>> A * U == M
True
>>> B = matrix.Matrix(1, 2, [2, 1])
>>> print B.kernelAsModule()
1
-2
```

1.1.4 RingSquareMatrix – 各成分が環に属する正方行列

RingSquareMatrix(`row`: *integer*, `column`: *integer*=0, `compo`: *compo*=0, `coeff_ring`: *CommutativeRing*=0)
→ *RingMatrix*

係数環が環に属する新しい正方行列を作成.

RingSquareMatrix**RingMatrix** と **SquareMatrix** のサブクラス. 初期化に関する情報は SquareMatrix を参照.

Operations

operator	explanation
`M**c`	行列 `M` の `c` 乗を返す.

Examples

```
>>> A = matrix.RingSquareMatrix(3, [1,2,3]+[4,5,6]+[7,8,9])
>>> A ** 2
[30L, 36L, 42L]+[66L, 81L, 96L]+[102L, 126L, 150L]
```

Methods

1.1.4.1 getRing – 行列の環を返す

getRing(`self`) → ***MatrixRing***

`self` の所属する MatrixRing を返す.

1.1.4.2 isOrthogonalMatrix – 直交行列かどうか

isOrthogonalMatrix(`self`) → ***True/False***

`self` が直交行列かどうか返す.

1.1.4.3 isAlternatingMatrix (isAntiSymmetricMatrix, isSkewSymmetricMatrix) – 交代行列かどうか

isAlternatingMatrix(`self`) → ***True/False***

`self` が交代行列かどうか返す.

1.1.4.4 isSingular – 特異行列かどうか

isSingular(`self`) → ***True/False***

`self` が特異行列かどうか返す.

この関数は `self` が 0 かどうか明らかにする. 正則行列が自動的に逆行列を持つわけではないということに注意; 逆行列が存在するかどうかの性質は係数環に依存する.

1.1.4.5 trace – トレース

trace(`self`) → ***RingElement***

`self` のトレースを返す.

1.1.4.6 determinant – 行列式

determinant(`self`) → ***RingElement***

`self` の行列式を返す.

1.1.4.7 cofactor – 余因子

cofactor(`self`, `i`: ***integer***, `j`: ***integer***) → ***RingElement***

(`i`, `j`) の余因子を返す.

1.1.4.8 commutator – 交換子

commutator(`self`, `N`: ***RingSquareMatrix element***) → ***RingSquareMatrix***

`self` と `N` の交換子を返す.

$[M,\ N]$ と表記される `M` と `N` の交換子は $[\mathtt{M},\ \mathtt{N}] = \mathtt{MN} - \mathtt{NM}$ と定義される.

1.1.4.9 characteristicMatrix – 特性行列

characteristicMatrix(`self`) → ***RingSquareMatrix***

`self` の特性行列を返す.

1.1.4.10 adjugateMatrix – 随伴行列

adjugateMatrix(`self`) → ***RingSquareMatrix***

`self` の随伴行列を返す.

`M` に対する随伴行列は単位行列 E に対し $\mathtt{MN} = \mathtt{NM} = (\det \mathtt{M})E$ となる行列 `N`.

1.1.4.11 cofactorMatrix (cofactors) – 余因子行列

cofactorMatrix(`self`) → ***RingSquareMatrix***
cofactors(`self`) → ***RingSquareMatrix***

self の余因子行列を返す.

M に対する余因子行列は M の $(i,\ j)$ 成分が $(i,\ j)$ 余因子である行列. 余因子行列は随伴行列の転置と同じ.

1.1.4.12 smithNormalForm (SNF, elementary_divisor) – Smith 正規形 (SNF)

smithNormalForm(self) → ***RingSquareMatrix***
SNF(self) → ***RingSquareMatrix***
elementary_divisor(self) → ***RingSquareMatrix***

self に対する Smith 正規形 (SNF) の対角成分のリストを返す.

この関数は self が非特異行列であることを仮定している.

1.1.4.13 extsmithNormalForm (extSNF) – Smith 正規形 (SNF)

extsmithNormalForm(self) → ***(RingSquareMatrix, RingSquareMatrix, RingSquareMatrix)***
extSNF(self) → ***RingSquareMatrix, RingSquareMatrix, RingSquareMatrix)***

self に対する Smith 正規形である M と U · self · V = M を満たす U,V を返す,.

Examples

```
>>> A = matrix.RingSquareMatrix(3, [3,-5,8]+[-9,2,7]+[6,1,-4])
>>> A.trace()
1L
>>> A.determinant()
-243L
>>> B = matrix.RingSquareMatrix(3, [87,38,80]+[13,6,12]+[65,28,60])
>>> U, V, M = B.extsmithNormalForm()
>>> U * B * V == M
True
>>> print M
4 0 0
0 2 0
0 0 1
>>> B.smithNormalForm()
[4L, 2L, 1L]
```

1.1.5 FieldMatrix – 各成分が体に属する行列

FieldMatrix(row: *integer*, column: *integer*, compo: *compo*=0, coeff_ring: *CommutativeRing*=0)
→ *RingMatrix*

係数環が体に属する新しい行列を作成.

FieldMatrix は **RingMatrix** のサブクラス. 初期化に関する情報は **Matrix** を参照.

Operations

operator	explanation
M/d	M を d で割った商を返す.d はスカラー.
M//d	M を d で割った商を返す.d はスカラー.

Examples

```
>>> A = matrix.FieldMatrix(3, 3, [1,2,3,4,5,6,7,8,9])
>>> A / 210
1/210 1/105 1/70
2/105  1/42 1/35
1/30  4/105 3/70
```

Methods

1.1.5.1 kernel – 核

kernel(`self`) → ***FieldMatrix***

`self` の核を返す.

出力は列ベクトルが核の基底となっている行列.
この関数は核が存在しなければ None を返す.

1.1.5.2 image – 像

image(`self`) → ***FieldMatrix***

`self` の像を返す.

出力は列ベクトルが像の基底となっている行列.
この関数は核が存在しなければ None を返す.

1.1.5.3 rank – 階数

rank(`self`) → ***integer***

`self` の階数を返す.

1.1.5.4 inverseImage – 逆像:一次方程式の基底解

inverseImage(`self`, `V`: ***Vector/RingMatrix***) → ***RingMatrix***

`self` による `V` の逆像を返す.

この関数は `self` $\cdot X =$ `V` と等しい一次式の一つの解を返す.

1.1.5.5 solve – 一次方程式の解

solve(`self`, `B`: ***Vector/RingMatrix***) → ***(RingMatrix, RingMatrix)***

`self` $\cdot X =$ `B` を解く.

この関数は特殊解 `sol` と `self` の核を行列として返す. もし特殊解のみ得たい

ときは **inverseImage** を使用.

1.1.5.6 columnEchelonForm – 列階段行列

columnEchelonForm(`self`) → ***RingMatrix***

列被約階段行列を返す.

Examples

```
>>> A = matrix.FieldMatrix(2, 3, [1,2,3]+[4,5,6])
>>> print A.kernel
 1/1
-2/1
   1
>>> print A.image()
1 2
4 5
>>> C = matrix.FieldMatrix(4, 3, [1,2,3]+[4,5,6]+[7,8,9]+[-1,-2,-3])
>>> D = matrix.FieldMatrix(4, 2, [1,0]+[7,6]+[13,12]+[-1,0])
>>> print C.inverseImage(D)
 3/1  4/1
-1/1 -2/1
 0/1  0/1
>>> sol, ker = C.solve(D)
>>> C * (sol + ker[0]) == D
True
>>> AA = matrix.FieldMatrix(3, 3, [1,2,3]+[4,5,6]+[7,8,9])
>>> print AA.columnEchelonForm()
0/1 2/1 -1/1
0/1 1/1  0/1
0/1 0/1  1/1
```

1.1.6 FieldSquareMatrix – 各成分が体に属する正方行列

FieldSquareMatrix(`row`: *integer*, `column`: *integer*=0, `compo`: *compo*=0, `coeff_ring`: *CommutativeRing*=0)
→ *FieldSquareMatrix*

係数環が体に属する新しい正方行列を返す.

FieldSquareMatrix は **FieldMatrix** と **SquareMatrix** のサブクラスです. †**determinant** 関数はオーバーライドされていて,**determinant** とは異なるアルゴリズムを用いている; この関数は **triangulate** から呼ばれる. 初期化に関する情報は **SquareMatrix** を参照.

Methods

1.1.6.1 triangulate - 行基本変形による三角化

triangulate(`self`) → ***FieldSquareMatrix***

行基本変形によって得られる上三角行列を返す.

1.1.6.2 inverse - 逆行列

inverse(`self` `V`: ***Vector/RingMatrix***=None) → ***FieldSquareMatrix***

`self` の逆行列を返す. `V` が与えられたら $\texttt{self}^{-1}V$ を返す.

† もし逆行列が存在しなければ **NoInverse** を返す.

1.1.6.3 hessenbergForm - Hessenberg 行列

hessenbergForm(`self`) → ***FieldSquareMatrix***

`self` の Hessenberg 行列を返す.

1.1.6.4 LUDecomposition - LU 分解

LUDecomposition(`self`) → ***(FieldSquareMatrix, FieldSquareMatrix)***

`self` == `LU` を満たす下三角行列 `L` と上三角行列 `U` を返す.

1.1.7 †MatrixRing – 行列の環

MatrixRing(`size`: *integer*, `scalars`: *CommutativeRing*)
→ *MatrixRing*

`size` と係数環 `scalars` を与えられた新しい行列の環を作成.

MatrixRing は **Ring** のサブクラス.

Methods

1.1.7.1 unitMatrix - 単位行列

unitMatrix(`self`) → ***RingSquareMatrix***

単位行列を返す.

1.1.7.2 zeroMatrix - 零行列

zeroMatrix(`self`) → ***RingSquareMatrix***

零行列を返す.

Examples

```
>>> M = matrix.MatrixRing(3, rational.theIntegerRing)
>>> print M
M_3(Z)
>>> M.unitMatrix()
[1L, 0L, 0L]+[0L, 1L, 0L]+[0L, 0L, 1L]
>>> M.zero
[0L, 0L, 0L]+[0L, 0L, 0L]+[0L, 0L, 0L]
```

1.1.7.3 getInstance(class function) - キャッシュされたインスタンスを返す

getInstance(`cls`, `size`: ***integer***, `scalars`: ***CommutativeRing***)
→ ***RingSquareMatrix***

与えられた `size` とスカラーの環に対する MatrixRing のインスタンスを返す.

初期化の代わりにこのメソッドを使うメリットは, 効率のためメソッドによって作成されたインスタンスがキャッシュされ再利用されることにある.

Examples

```
>>> print MatrixRing.getInstance(3, rational.theIntegerRing)
M_3(Z)
```

1.1.8 Subspace – 有限次元ベクトル空間の部分空間

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

いくつかの有限次元ベクトル空間の新しい部分空間を作成.

Subspace は **FieldMatrix** のサブクラス.
初期化に関する情報は **Matrix** を参照. 部分空間は `self` の列ベクトルに張られる空間を示す.

`isbasis` が True なら, 列ベクトルは一次独立と仮定.

Attributes

isbasis 列ベクトルが一次独立の性質を表す. もし各ベクトルが空間の基底を成せば `isbasis` は True, そうでなければ False.

Methods

1.1.8.1 issubspace - 部分空間かどうか

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

もし `other` の部分空間であれば True, そうでなければ False を返す.

1.1.8.2 toBasis - 基底を選択

toBasis(`self`) → ***(None)***

列ベクトルが基底を成すように `self` を書き直し, その `isbasis` を True にする.

この関数は `isbasis` がすでに True ならばなにもしない.

1.1.8.3 supplementBasis - 最大階数にする

supplementBasis(`self`) → ***Subspace***

`self` に基底を補完したことによる最大階数行列を返す.

1.1.8.4 sumOfSubspaces - 部分空間の和

sumOfSubspaces(`self`, `other`: ***Subspace***) → ***Subspace***

二つの部分空間の和集合の基底を成す列の行列を返す.

1.1.8.5 intersectionOfSubspaces - 部分空間の共通部分

intersectionOfSubspaces(`self`, `other`: ***Subspace***) → ***Subspace***

二つの部分空間の共通部分の基底を成す列の行列を返す.

Examples

```
>>> A = matrix.Subspace(4, 3, [1,2,3]+[4,5,6]+[7,8,9]+[10,11,12])
>>> A.toBasis()
>>> print A
 1  2
 4  5
 7  8
10 11
>>> B = matrix.Subspace(3, 2, [1,2]+[3,4]+[5,7])
>>> print B.supplementBasis()
1 2 0
3 4 0
5 7 1
>>> C = matrix.Subspace(4, 1, [1,2,3,4])
>>> D = matrix.Subspace(4, 2, [2,-4]+[4,-3]+[6,-2]+[8,-1])
>>> print C.intersectionOfSubspaces(D)
-2/1
-4/1
-6/1
-8/1
```

1.1.8.6 fromMatrix(class function) - 部分空間を作成

fromMatrix(`cls`, `mat`: ***FieldMatrix***, `isbasis`: ***True/False***=**None**)
→ ***Subspace***

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

Subspace のインスタンスがほしい場合はこのメソッドを使用.

1.1.9 createMatrix[function] – インスタンスを作成

createMatrix(row: *integer*, column: *integer*=0, compo: *compo*=0, coeff_ring: *CommutativeRing*=None)
→ RingMatrix

RingMatrix, RingSquareMatrix, FieldMatrix または FieldSquareMatrix のインスタンスを作成.

入力すると行列のサイズと係数環を調べることにより自動的に自身のクラスを決定. 初期化に関する情報は Matrix または SquareMatrix を参照.

1.1.10 identityMatrix(unitMatrix)[function] – 単位行列

identityMatrix(size: *integer*, coeff: *CommutativeRing/CommutativeRingElement*=None)
→ RingMatrix

unitMatrix(size: *integer*, coeff: *CommutativeRing/CommutativeRingElement*=None)
→ RingMatrix

size 次元の単位行列を返す.

coeff により, 整数上だけでなく coeff により決定された係数環上でも行列を作成することができる.

coeff は Ring のインスタンス, または積に関する単位元でなければならない.

1.1.11 zeroMatrix[function] – 零行列

zeroMatrix(row: *integer*, column: *0*=, coeff: *CommutativeRing/CommutativeRingElement*=None)
→ RingMatrix

row × column 零行列を返す.

coeff により, 整数上だけでなく coeff により決定された係数環上でも行列を作成することができる.

coeff は Ring のインスタンス, または和に関する単位元でなければならない. column を省略したら,column は row と同じで設定される.

Examples

```
>>> M = matrix.createMatrix(3, [1,2,3]+[4,5,6]+[7,8,9])
>>> print M
1 2 3
4 5 6
7 8 9
>>> O = matrix.zeroMatrix(2, 3, imaginary.ComplexField())
>>> print O
0 + 0j 0 + 0j 0 + 0j
0 + 0j 0 + 0j 0 + 0j
```

Bibliography