

Contents

1	Classes	4
1.1	matrix – matrices	4
1.1.1	Matrix – matrices	6
1.1.1.1	map – apply function to elements	8
1.1.1.2	reduce – reduce elements iteratively	8
1.1.1.3	copy – create a copy	8
1.1.1.4	set – set compo	8
1.1.1.5	setRow – set m-th row vector	9
1.1.1.6	setColumn – set n-th column vector	9
1.1.1.7	getRow – get i-th row vector	9
1.1.1.8	getColumn – get j-th column vector	9
1.1.1.9	swapRow – swap two row vectors	9
1.1.1.10	swapColumn – swap two column vectors	10
1.1.1.11	insertRow – insert row vectors	10
1.1.1.12	insertColumn – insert column vectors	10
1.1.1.13	extendRow – extend row vectors	10
1.1.1.14	extendColumn – extend column vectors	10
1.1.1.15	deleteRow – delete row vector	11
1.1.1.16	deleteColumn – delete column vector	11
1.1.1.17	transpose – transpose matrix	11
1.1.1.18	getBlock – block matrix	11
1.1.1.19	subMatrix – submatrix	11
1.1.2	SquareMatrix – square matrices	13
1.1.2.1	isUpperTriangularMatrix – check upper triangular	14
1.1.2.2	isLowerTriangularMatrix – check lower triangular	14
1.1.2.3	isDiagonalMatrix – check diagonal matrix	14
1.1.2.4	isScalarMatrix – check scalar matrix	14
1.1.2.5	isSymmetricMatrix – check symmetric matrix	14
1.1.3	RingMatrix – matrix whose elements belong ring	16
1.1.3.1	getCoefficientRing – get coefficient ring	17
1.1.3.2	toFieldMatrix – set field as coefficient ring	17
1.1.3.3	toSubspace – regard as vector space	17
1.1.3.4	hermiteNormalForm (HNF) – Hermite Normal Form	17

1.1.3.5	exthermiteNormalForm (extHNF) – extended Her- mite Normal Form algorithm	17
1.1.3.6	kernelAsModule – kernel as \mathbb{Z} -module	18
1.1.4	RingSquareMatrix – square matrix whose elements belong ring	19
1.1.4.1	getRing – get matrix ring	20
1.1.4.2	isOrthogonalMatrix – check orthogonal matrix	20
1.1.4.3	isAlternatingMatrix (isAntiSymmetricMatrix, isSkewSym- metricMatrix) – check alternating matrix	20
1.1.4.4	isSingular – check singular matrix	20
1.1.4.5	trace – trace	20
1.1.4.6	determinant – determinant	21
1.1.4.7	cofactor – cofactor	21
1.1.4.8	commutator – commutator	21
1.1.4.9	characteristicMatrix – characteristic matrix	21
1.1.4.10	adjugateMatrix – adjugate matrix	21
1.1.4.11	cofactorMatrix (cofactors) – cofactor matrix	22
1.1.4.12	smithNormalForm (SNF, elementary_divisor) – Smith Normal Form (SNF)	22
1.1.4.13	extsmithNormalForm (extSNF) – Smith Normal Form (SNF)	22
1.1.5	FieldMatrix – matrix whose elements belong field	24
1.1.5.1	kernel – kernel	25
1.1.5.2	image – image	25
1.1.5.3	rank – rank	25
1.1.5.4	inverseImage – inverse image: base solution of linear system	25
1.1.5.5	solve – solve linear system	25
1.1.5.6	columnEchelonForm – column echelon form	26
1.1.6	FieldSquareMatrix – square matrix whose elements be- long field	27
1.1.6.1	triangulate - triangulate by elementary row op- eration	28
1.1.6.2	inverse - inverse matrix	28
1.1.6.3	hessenbergForm - hessenberg form	28
1.1.6.4	LUdecomposition - LU decomposition	28
1.1.7	†MatrixRing – ring of matrices	29
1.1.7.1	unitMatrix - unit matrix	30
1.1.7.2	zeroMatrix - zero matrix	30
1.1.7.3	getInstance(class function) - get cached instance	31
1.1.8	Subspace – subspace of finite dimensional vector space	32
1.1.8.1	issubspace - check subspace of self	33
1.1.8.2	toBasis - select basis	33
1.1.8.3	supplementBasis - to full rank	33
1.1.8.4	sumOfSubspaces - sum as subspace	33
1.1.8.5	intersectionOfSubspaces - intersection as subspace	33

1.1.8.6	fromMatrix(class function) - create subspace . .	35
1.1.9	createMatrix[function] - create an instance	36
1.1.10	identityMatrix(unitMatrix)[function] - unit matrix	36
1.1.11	zeroMatrix[function] - zero matrix	36

Chapter 1

Classes

1.1 matrix – matrices

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

The module matrix has also some exception classes.

MatrixSizeError : Report contradicting given input to the matrix size.

VectorsNotIndependent : Report column vectors are not independent.

NoInverseImage : Report any inverse image does not exist.

NoInverse : Report the matrix is not invertible.

This module using following type:

compo : compo must be one of these forms below.

- concatenated row lists, such as $[1,2]+[3,4]+[5,6]$.
- list of row lists, such as $[[1,2], [3,4], [5,6]]$.
- list of column tuples, such as $[(1, 3, 5), (2, 4, 6)]$.
- list of vectors whose dimension equals column, such as $vector.Vector([1, 3, 5]), vector.Vector([2, 4, 6])$.

The examples above represent the same matrix form as follows:

1	2
3	4
5	6

1.1.1 Matrix – matrices

Initialize (Constructor)

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

Create new matrices object.

†This constructor automatically changes the class to one of the following class: **RingMatrix**, **RingSquareMatrix**, **FieldMatrix**, **FieldSquareMatrix**.

Your input determines the class automatically by examining the matrix size and the coefficient ring. **row** and **column** must be integer, and **coeff_ring** must be an instance of **Ring**. Refer to **compo** for information about **compo**. If you abbreviate **compo**, it will be deemed to all zero list.

The list of expected inputs and outputs is as following:

- Matrix(**row**, **column**, **compo**, **coeff_ring**)
→ the **row**×**column** matrix whose elements are **compo** and coefficient ring is **coeff_ring**
- Matrix(**row**, **column**, **compo**)
→ the **row**×**column** matrix whose elements are **compo** (The coefficient ring is automatically determined.)
- Matrix(**row**, **column**, **coeff_ring**)
→ the **row**×**column** matrix whose coefficient ring is **coeff_ring** (All elements are 0 in **coeff_ring**.)
- Matrix(**row**, **column**)
→ the **row**×**column** matrix (The coefficient matrix is **Integer**. All elements are 0.)

Attribute

row : The row size of the matrix.

column : The column size of the matrix.

coeff_ring : The coefficient ring of the matrix.

compo : The elements of the matrix.

Operations

operator	explanation
<code>M==N</code>	Return whether M and N are equal or not.
<code>M[i, j]</code>	Return the coefficient of i-th row, j-th column term of matrix M.
<code>M[i]</code>	Return the vector of i-th column term of matrix M.
<code>M[i, j]=c</code>	Replace the coefficient of i-th row, j-th column term of matrix M by c.
<code>M[j]=c</code>	Replace the vector of i-th column term of matrix M by vector c.
<code>c in M</code>	Check whether some element of M equals c.
<code>repr(M)</code>	Return the repr string of the matrix M. string represents list concatenated row vector lists.
<code>str(M)</code>	Return the str string of the matrix M.

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 – apply function to elements

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

Return the matrix whose elements is applied `function` to.

†The function `map` is an analogy of built-in function `map`.

1.1.1.2 reduce – reduce elements iteratively

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

Apply `function` from upper-left to lower-right, so as to reduce the iterable to a single value.

†The function `map` is an analogy of built-in function `reduce`.

1.1.1.3 copy – create a copy

copy(self) → *Matrix*

create a copy of `self`.

†The matrix generated by the function is same matrix to `self`, but not same as a instance.

1.1.1.4 set – set compo

set(self, compo: *compo*) → (*None*)

Substitute the list `compo` for `compo`.

`compo` must be the form of `compo`.

1.1.1.5 setRow – set m-th row vector

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

Substitute the list/Vector `arg` as m-th row.

The length of `arg` must be same to `self.column`.

1.1.1.6 setColumn – set n-th column vector

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

Substitute the list/Vector `arg` as n-th column.

The length of `arg` must be same to `self.row`.

1.1.1.7 getRow – get i-th row vector

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

Return i-th row in form of `self`.

The function returns a row vector (an instance of `Vector`).

1.1.1.8 getColumn – get j-th column vector

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

Return j-th column in form of `self`.

1.1.1.9 swapRow – swap two row vectors

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

Swap `self`'s m1-th row vector and m2-th row one.

1.1.1.10 swapColumn – swap two column vectors

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

Swap `self`'s `n1`-th column vector and `n2`-th column one.

1.1.1.11 insertRow – insert row vectors

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

Insert row vectors `arg` to `i`-th row.

`arg` must be list, **Vector** or **Matrix**. The length (or **column**) of it should be same to the column of `self`.

1.1.1.12 insertColumn – insert column vectors

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

Insert column vectors `arg` to `j`-th column.

`arg` must be list, **Vector** or **Matrix**. The length (or **row**) of it should be same to the row of `self`.

1.1.1.13 extendRow – extend row vectors

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

Join `self` with row vectors `arg` (in vertical way).

The function combines `self` with the last row vector of `self`. That is, `extendRow(arg)` is same to `insertRow(self.row+1, arg)`.

`arg` must be list, **Vector** or **Matrix**. The length (or **column**) of it should be same to the column of `self`.

1.1.1.14 extendColumn – extend column vectors

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

Join `self` with column vectors `arg` (in horizontal way).

The function combines `self` with the last column vector of `self`. That is,

`extendColumn(arg)` is same to `insertColumn(self.column+1, arg)`.

`arg` must be list, **Vector** or **Matrix**. The length (or **row**) of it should be same to the row of `self`.

1.1.1.15 deleteRow – delete row vector

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

Delete i-th row vector.

1.1.1.16 deleteColumn – delete column vector

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

Delete j-th column vector.

1.1.1.17 transpose – transpose matrix

`transpose(self) → Matrix`

Return the transpose of `self`.

1.1.1.18 getBlock – block matrix

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

Return the `row`×`column` block matrix from the (i, j)-element.

If `column` is omitted, `column` is considered as same value to `row`.

1.1.1.19 subMatrix – submatrix

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

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

The function has a twofold significance.

- I and J are integer:
Return submatrix deleted I-th row and J-th column.
- I and J are list:
Return the submatrix composed of elements from `self` assigned by rows I and columns J, respectively.

If J is omitted, J is considered as same value to I.

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 – square matrices

Initialize (Constructor)

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

Create new square matrices object.

SquareMatrix is subclass of **Matrix**. †This constructor automatically changes the class to one of the following class: **RingMatrix**, **RingSquareMatrix**, **FieldMatrix**, **FieldSquareMatrix**.

Your input determines the class automatically by examining the matrix size and the coefficient ring. **row** and **column** must be integer, and **coeff_ring** must be an instance of **Ring**. Refer to **compo** for information about **compo**. If you abbreviate **compo**, it will be deemed to all zero list.

The list of expected inputs and outputs is as following:

- Matrix(**row**, **compo**, **coeff_ring**)
→ the **row** square matrix whose elements are **compo** and coefficient ring is **coeff_ring**
- Matrix(**row**, **compo**)
→ the **row** square matrix whose elements are **compo** (coefficient ring is automatically determined)
- Matrix(**row**, **coeff_ring**)
→ the **row** square matrix whose coefficient ring is **coeff_ring** (All elements are 0 in **coeff_ring**.)
- Matrix(**row**)
→ the **row** square matrix (The coefficient ring is Integer. All elements are 0.)

†We can initialize as **Matrix**, but **column** must be same to **row** in the case.

Methods

1.1.2.1 isUpperTriangularMatrix – check upper triangular

isUpperTriangularMatrix(self) → *True/False*

Check whether `self` is upper triangular matrix or not.

1.1.2.2 isLowerTriangularMatrix – check lower triangular

isLowerTriangularMatrix(self) → *True/False*

Check whether `self` is lower triangular matrix or not.

1.1.2.3 isDiagonalMatrix – check diagonal matrix

isDiagonalMatrix(self) → *True/False*

Check whether `self` is diagonal matrix or not.

1.1.2.4 isScalarMatrix – check scalar matrix

isScalarMatrix(self) → *True/False*

Check whether `self` is scalar matrix or not.

1.1.2.5 isSymmetricMatrix – check symmetric matrix

isSymmetricMatrix(self) → *True/False*

Check whether `self` is symmetric matrix or not.

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 – matrix whose elements belong ring

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

Create matrix whose coefficient ring belongs ring.

RingMatrix is subclass of **Matrix**. See Matrix for getting information about the initialization.

Operations

operator	explanation
M+N	Return the sum of matrices M and N.
M-N	Return the difference of matrices M and N.
M*N	Return the product of M and N. N must be matrix, vector or scalar
M % d	Return M modulo d. d must be nonzero integer.
-M	Return the matrix whose coefficients have inverted signs of M.
+M	Return the copy of M.

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` – get coefficient ring

`getCoefficientRing(self)` → *CommutativeRing*

Return the coefficient ring of `self`.

This method checks all elements of `self` and set `coeff_ring` to the valid coefficient ring.

1.1.3.2 `toFieldMatrix` – set field as coefficient ring

`toFieldMatrix(self)` → (*None*)

Change the class of the matrix to **FieldMatrix** or **FieldSquareMatrix**, where the coefficient ring will be the quotient field of the current domain.

1.1.3.3 `toSubspace` – regard as vector space

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

Change the class of the matrix to `Subspace`, where the coefficient ring will be the quotient field of the current domain.

1.1.3.4 `hermiteNormalForm` (HNF) – Hermite Normal Form

`hermiteNormalForm(self)` → *RingMatrix*

`HNF(self)` → *RingMatrix*

Return upper triangular hermite normal form (HNF).

1.1.3.5 `exthermiteNormalForm` (extHNF) – extended Hermite Normal Form algorithm

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

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

Return hermite normal form M and U satisfied $\text{self}U = M$.

The function returns tuple (U, M) , where U is an instance of **RingSquareMatrix** and M is an instance of **RingMatrix**.

1.1.3.6 kernelAsModule – kernel as \mathbb{Z} -module

kernelAsModule(self) \rightarrow *RingMatrix*

Return kernel as \mathbb{Z} -module.

The difference between the function and **kernel** is that each elements of the returned value are integer.

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 – square matrix whose elements belong ring

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

Create square matrix whose coefficient ring belongs ring.

RingSquareMatrix is subclass of **RingMatrix** and **SquareMatrix**. See SquareMatrix for getting information about the initialization.

Operations

operator	explanation
M**c	Return the c-th power of matrices M.

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` – get matrix ring

`getRing(self)` → *MatrixRing*

Return the **MatrixRing** belonged to by `self`.

1.1.4.2 `isOrthogonalMatrix` – check orthogonal matrix

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

Check whether `self` is orthogonal matrix or not.

1.1.4.3 `isAlternatingMatrix` (`isAntiSymmetricMatrix`, `isSkewSymmetricMatrix`) – check alternating matrix

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

Check whether `self` is alternating matrix or not.

1.1.4.4 `isSingular` – check singular matrix

`isSingular(self)` → *True/False*

Check whether `self` is singular matrix or not.

The function determines whether determinant of `self` is 0. Note that the non-singular matrix does not automatically mean invertible matrix; the nature that the matrix is invertible depends on its coefficient ring.

1.1.4.5 `trace` – trace

`trace(self)` → *RingElement*

Return the trace of `self`.

1.1.4.6 determinant – determinant

determinant(self) → *RingElement*

Return the determinant of self.

1.1.4.7 cofactor – cofactor

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

Return the (i, j)-cofactor.

1.1.4.8 commutator – commutator

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

Return the commutator for self and N.

The commutator for M and N, which is denoted as $[M, N]$, is defined as $[M, N] = MN - NM$.

1.1.4.9 characteristicMatrix – characteristic matrix

characteristicMatrix(self) → *RingSquareMatrix*

Return the characteristic matrix of self.

1.1.4.10 adjugateMatrix – adjugate matrix

adjugateMatrix(self) → *RingSquareMatrix*

Return the adjugate matrix of self.

The adjugate matrix for M is the matrix N such that $MN = NM = (\det M)E$, where E is the identity matrix.

1.1.4.11 cofactorMatrix (cofactors) – cofactor matrix

cofactorMatrix(self) → *RingSquareMatrix*

cofactors(self) → *RingSquareMatrix*

Return the cofactor matrix of **self**.

The cofactor matrix for **M** is the matrix whose (i, j) element is (i, j) -cofactor of **M**. The cofactor matrix is same to transpose of the adjugate matrix.

1.1.4.12 smithNormalForm (SNF, elementary_divisor) – Smith Normal Form (SNF)

smithNormalForm(self) → *RingSquareMatrix*

SNF(self) → *RingSquareMatrix*

elementary_divisor(self) → *RingSquareMatrix*

Return the list of diagonal elements of the Smith Normal Form (SNF) for **self**.

The function assumes that **self** is non-singular.

1.1.4.13 extsmithNormalForm (extSNF) – Smith Normal Form (SNF)

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

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

Return the Smith normal form **M** for **self** and **U, V** satisfied **UselfV = M**.

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 – matrix whose elements belong field

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

Create matrix whose coefficient ring belongs field.

FieldMatrix is subclass of **RingMatrix**. See **Matrix** for getting information about the initialization.

Operations

operator	explanation
M/d	Return the division of M by d.d must be scalar.
M//d	Return the division of M by d.d must be scalar.

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 – kernel

kernel(self) → *FieldMatrix*

Return the kernel of **self**.

The output is the matrix whose column vectors form basis of the kernel.
The function returns None if the kernel do not exist.

1.1.5.2 image – image

image(self) → *FieldMatrix*

Return the image of **self**.

The output is the matrix whose column vectors form basis of the image.
The function returns None if the kernel do not exist.

1.1.5.3 rank – rank

rank(self) → *integer*

Return the rank of **self**.

1.1.5.4 inverseImage – inverse image: base solution of linear system

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

Return an inverse image of V by **self**.

The function returns one solution of the linear equation **self**X = V.

1.1.5.5 solve – solve linear system

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

Solve **self**X = B.

The function returns a particular solution **sol** and the kernel of **self** as a

matrix. If you only have to obtain the particular solution, use **inverseImage**.

1.1.5.6 columnEchelonForm – column echelon form

columnEchelonForm(self) → *RingMatrix*

Return the column reduced echelon form.

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 – square matrix whose elements belong field

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

Create square matrix whose coefficient ring belongs field.

FieldSquareMatrix is subclass of **FieldMatrix** and **SquareMatrix**.

†The function **RingSquareMatrix**determinant is overridden and use different algorithm from one used in **RingSquareMatrix**determinant;the function calls **FieldSquareMatrix**triangulate. See **SquareMatrix** for getting information about the initialization.

Methods

1.1.6.1 triangulate - triangulate by elementary row operation

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

Return an upper triangulated matrix obtained by elementary row operations.

1.1.6.2 inverse - inverse matrix

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

Return the inverse of `self`. If `V` is given, then return `self(-1)V`.

‡If the matrix is not invertible, then raise **NoInverse**.

1.1.6.3 hessenbergForm - hessenberg form

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

Return the hessenberg form of `self`.

1.1.6.4 LUdecomposition - LU decomposition

```
LUdecomposition(self)  
→ (FieldSquareMatrix, FieldSquareMatrix)
```

Return the lower triangular matrix `L` and the upper triangular matrix `U` such that `self == LU`.

1.1.7 †MatrixRing – ring of matrices

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

Create a ring of matrices with given **size** and coefficient ring **scalars**.

MatrixRing is subclass of **Ring**.

Methods

1.1.7.1 unitMatrix - unit matrix

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

Return the unit matrix.

1.1.7.2 zeroMatrix - zero matrix

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

Return the zero matrix.

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) - get cached instance

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

Return an instance of MatrixRing of given size and ring of scalars.

The merit of using the method instead of the constructor is that the instances created by the method are cached and reused for efficiency.

Examples

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

1.1.8 Subspace – subspace of finite dimensional vector space

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

Create subspace of some finite dimensional vector space over a field.

Subspace is subclass of **FieldMatrix**.

See **Matrix** for getting information about the initialization. The subspace expresses the space generated by column vectors of **self**.

If **isbasis** is True, we assume that column vectors are linearly independent.

Attribute

isbasis The attribute indicates the linear independence of column vectors, i.e., if they form a basis of the space then **isbasis** should be True, otherwise False.

Methods

1.1.8.1 issubspace - check subspace of self

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

Return True if the subspace instance is a subspace of the `other`, or False otherwise.

1.1.8.2 toBasis - select basis

```
toBasis(self)  
→ (None)
```

Rewrite `self` so that its column vectors form a basis, and set True to its `isbasis`.

The function does nothing if `isbasis` is already True.

1.1.8.3 supplementBasis - to full rank

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

Return full rank matrix by supplementing bases for `self`.

1.1.8.4 sumOfSubspaces - sum as subspace

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

Return a matrix whose columns form a basis for sum of two subspaces.

1.1.8.5 intersectionOfSubspaces - intersection as subspace

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

Return a matrix whose columns form a basis for intersection of two subspaces.

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) - create subspace

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

Create a Subspace instance from a matrix instance `mat`, whose class can be any of subclasses of `Matrix`.

Please use this method if you want a Subspace instance for sure.

1.1.9 createMatrix[function] – create an instance

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

Create an instance of **RingMatrix**, **RingSquareMatrix**, **FieldMatrix** or **FieldSquareMatrix**.

Your input determines the class automatically by examining the matrix size and the coefficient ring. See **Matrix** or **SquareMatrix** for getting information about the initialization.

1.1.10 identityMatrix(unitMatrix)[function] – unit matrix

```
identityMatrix(size: integer, coeff: CommutativeR-  
ing/CommutativeRingElement=None)  
→ RingMatrix
```

```
unitMatrix(size: integer, coeff: CommutativeR-  
ing/CommutativeRingElement=None)  
→ RingMatrix
```

Return size-dimensional unit matrix.

`coeff` enables us to create matrix not only in integer but in coefficient ring which is determined by `coeff`.

`coeff` must be an instance of **Ring** or a multiplicative unit (one).

1.1.11 zeroMatrix[function] – zero matrix

```
zeroMatrix(row: integer, column: 0=, coeff: CommutativeR-  
ing/CommutativeRingElement=None)  
→ RingMatrix
```

Return `row` × `column` zero matrix.

`coeff` enables us to create matrix not only in integer but in coefficient ring which is determined by `coeff`.

`coeff` must be an instance of **Ring** or an additive unit (zero). If `column` is abbreviated, `column` is set same to `row`.

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