
Matrix

Release 2.0

CABOS Matthieu

Dec 22, 2021

CONTENTS:

1	Introduction	3
2	Matrix Usage Quick Start	5
2.1	Add function	5
2.2	Sub function	6
2.3	Mul function	7
2.4	Div function	8
2.5	Mod function	8
2.6	Neg function	9
2.7	Str function	10
2.8	Len function	10
2.9	Getitem function	10
2.10	Setitem function	11
2.11	Eq function	12
2.12	Ne function	13
2.13	Le function	13
2.14	Ge function	14
2.15	Lt function	14
2.16	Gt function	15
2.17	Contains function	15
3	API Functions	17
3.1	Matrix generators	17
3.2	Utility	22
3.3	Sequence recognition algorithm	27
4	Matrix	35
4.1	cmatrix.Matrix.ij_2_ind	37
4.2	cmatrix.Matrix.ind_2_ij	37
4.3	cmatrix.Matrix.size_r	37
4.4	cmatrix.Matrix.size_c	37
4.5	cmatrix.Matrix.get_ij	37
4.6	cmatrix.Matrix.set_ij	37
4.7	cmatrix.Matrix.get_ind	37
4.8	cmatrix.Matrix.same_size	38
4.9	cmatrix.Matrix.mult_compatible	38
4.10	cmatrix.Matrix.Print	38
4.11	cmatrix.Matrix.clone	38
4.12	cmatrix.Matrix.xtract_sub_matrix	38
4.13	cmatrix.Matrix.insert_sub_matrix	39

4.14	cmatrix.Matrix.find_sub_matrix	40
4.15	cmatrix.Matrix.op	41
4.16	cmatrix.Matrix.sqrt	43
4.17	cmatrix.Matrix.converter	43
4.18	cmatrix.Matrix.mand	44
4.19	cmatrix.Matrix.mor	45
4.20	cmatrix.Matrix.mnand	46
4.21	cmatrix.Matrix.mnor	47
4.22	cmatrix.Matrix.mxor	48
4.23	cmatrix.Matrix.column	49
4.24	cmatrix.Matrix.line	49
4.25	cmatrix.Matrix.swap_col	50
4.26	cmatrix.Matrix.swap_line	51
4.27	cmatrix.Matrix.mirror_mat	51
4.28	cmatrix.Matrix.nth_diagonal	52
4.29	cmatrix.Matrix.get_coef	54
4.30	cmatrix.Matrix.mul_coef	54
4.31	cmatrix.Matrix.resize	54
4.32	cmatrix.Matrix.accumulate	55
4.33	cmatrix.Matrix.square	56
4.34	cmatrix.Matrix.transpose	57
4.35	cmatrix.Matrix.permut	57
4.36	cmatrix.Matrix.triangle	59
4.37	cmatrix.Matrix.strassen	59
4.38	cmatrix.Matrix.tensorial_product	61
4.39	cmatrix.Matrix.trace	62
4.40	cmatrix.Matrix.convolve	63
4.41	cmatrix.Matrix.convolve_signal	64
4.42	cmatrix.Matrix.AtoB	65
4.43	cmatrix.Matrix.gap	66
4.44	cmatrix.Matrix.map	67
4.45	cmatrix.Matrix.normalize	68
4.46	cmatrix.Matrix.UX_B	68
4.47	cmatrix.Matrix.LX_B	69
4.48	cmatrix.Matrix.replace_zeros	70
4.49	cmatrix.Matrix.LU	71
4.50	cmatrix.Matrix.det	72
4.51	cmatrix.Matrix.orthogonal	73
4.52	cmatrix.Matrix.symmetric	73
4.53	cmatrix.Matrix.hermitian	74
4.54	cmatrix.Matrix.reversal	75
4.55	cmatrix.Matrix.count_zero	75
4.56	cmatrix.Matrix.nilpotent	76
4.57	cmatrix.Matrix.diagonal	77
4.58	cmatrix.Matrix.triangular	77
4.59	cmatrix.Matrix.read_from_file	78
4.60	cmatrix.Matrix.write_in_file	79
4.61	cmatrix.Matrix.pack	80

5 Indices and tables 81

Index 83

Welcome in the CMatrix Module. This module, written with python and Cython, is as fast as a c/c++ Librairie file. It has been thought for Matrix complex computation. This Module have been splitted into three main parts :

- **A Matrix Generators Section** The Matrix Generators section permit to generate automatically standard algebra matrix as :
 - A Laplacian mean Matrix
 - A Gaussian mean Matrix
 - A Mean filter Matrix
 - A Random Permutation Matrix
 - A Unit Matrix
 - A Zeros Matrix
- **A Sequence Recognition Section** This Sequence Recognition Algorithm permit us to find in any list any repeated sequence from **nothing** . This Algorithm is splitted into 10 functions : 1 main algorithm and 9 sub-function. The main algorithm function name is `find_mul_seq`, meaning Find Multiples Sequences function. The 9 sub-functions used to make it works are :
 - `count_seq` : A sequence counter
 - `create_mutants` : A list mutants generator function
 - `cut` : A list cutter function
 - `find_invariant` : An invariant finder function
 - `find_max` : Get the max length list function
 - `find_seq` : A sequence finder function
 - `find_seq_in_list` : An advanced sequence finder function
 - `split_data` : Data splitter function
 - `split_number` : A number splitter function
- **The Main Matrix Engine** The main matrix engine define an new Matrix Object into your Python code. This matrix should be extracted and readed from :
 - Pictures
 - Video
 - Signals
 - Float and Int matrix
 - etc

To read a matrix from an external file, you can use the Matrix Getter py file into the main github repository to extract and use any matrix with this lib. It contains differents kind of function :

- Primary utility functions as Ribbon converter, usual operators, ...
- Middle level functions as Matrix extraction/recognition, usuals algebra operations, ...
- Hard level functions as Strassen algorithm, Convolutions, System resolution, Tensorial operations, ...

The full description of each part is available on the left tree entries. All the functions are documented and a small example will be used as an Usage Guide.

Here you will find the main Scheme to have a global look on this module.

INTRODUCTION

Here is my new version of the matrix library including sequence recognition algorithm as compressor. To setup manually :

- **Install all prerequisites as following :**
 - Get python3.8 from *www.python.org*
 - **pip install sphinx**
 - **pip install numpy**
 - **pip install random**
 - **pip install cython**
 - **pip install sphinxcontrib-napoleon**
 - **pip install sphinx-autoapi**
- **Compile the cmatrix.pyx file using the command : `python setup.py build_ext -inplace`**
- **compile the associated html file using the command: `.make html`**

MATRIX USAGE QUICK START

Here, you will find all the basics operations computed from the redefined python function.

2.1 Add function

Ad hoc polymorphism 'add' function

Parameters	Type	Description
<i>mat</i>	Matrix	The matrix to add

2.1.1 Returns

Matrix The sum of the self matrix and the mat arg matrix

2.1.2 Examples

```
>>> m=rand(3)
>>> n=rand(3)
>>> print(m)
| +5.000 | +1.000 | +6.000 |
| +4.000 | +3.000 | +7.000 |
| +3.000 | +9.000 | +7.000 |
printed
>>> print(n)
| +5.000 | +7.000 | +3.000 |
| +2.000 | +4.000 | +7.000 |
| +9.000 | +5.000 | +2.000 |
printed
>>> print(m+n)
| +10.000 | +8.000 | +9.000 |
| +6.000 | +7.000 | +14.000 |
| +12.000 | +14.000 | +9.000 |
printed
```

2.1.3 See Also

`cmatrix.Matrix.op()`

2.2 Sub function

Ad hoc polymorphism ‘sub’ function

Parameters	Type	Description
<i>mat</i>	Matrix	The matrix to sub

2.2.1 Returns

Matrix The difference between the self matrix and the mat arg matrix

2.2.2 Examples

```
>>> m=rand(3)
>>> n=rand(3)
>>> print(m)
| +8.000 | +3.000 | +6.000 |
| +1.000 | +9.000 | +4.000 |
| +0.000 | +8.000 | +4.000 |
printed
>>> print(n)
| +8.000 | +7.000 | +8.000 |
| +2.000 | +5.000 | +9.000 |
| +0.000 | +9.000 | +7.000 |
printed
>>> print(m-n)
| +0.000 | -4.000 | -2.000 |
| -1.000 | +4.000 | -5.000 |
| +0.000 | -1.000 | -3.000 |
printed
```

2.2.3 See Also

`op()`

2.3 Mul function

Genericity polymorphism ‘mul’ function

Parameters	Type	Description
<i>mat</i>	Matrix	The matrix to multiply
<i>mat</i>	int/double	The multiplicative coefficient

2.3.1 Returns

Matrix

The product between the self matrix and the argument, depending of argument type :

- Matrix mean a matrix product
- integer mean a lambda product
- double mean a lambda product

2.3.2 Examples

```
>>> m=rand(3)
>>> n=rand(3)
>>> print(m)
| +4.000 | +1.000 | +7.000 |
| +2.000 | +9.000 | +7.000 |
| +8.000 | +6.000 | +6.000 |
printed
>>> print(n)
| +2.000 | +1.000 | +5.000 |
| +3.000 | +5.000 | +0.000 |
| +2.000 | +2.000 | +7.000 |
printed
>>> print(m*n)
| +50.000 | +41.000 | +51.000 |
| +22.000 | +48.000 | +56.000 |
| +68.000 | +62.000 | +70.000 |
printed
```

2.3.3 See Also

`cmatrix.Matrix.op()`

2.4 Div function

Genericity polymorphism ‘div’ function

Parameters	Type	Description
<i>value</i>	double	The diviser value
<i>value</i>		int The diviser value

2.4.1 Returns

Matrix The divided matrix

2.4.2 Examples

```
>>> m=rand(4)
>>> print(m)
| +15.000 | +9.000 | +12.000 | +3.000 |
| +9.000 | +14.000 | +8.000 | +4.000 |
| +15.000 | +14.000 | +0.000 | +9.000 |
| +8.000 | +8.000 | +6.000 | +8.000 |
printed
>>> print(m/10)
| +1.500 | +0.900 | +1.200 | +0.300 |
| +0.900 | +1.400 | +0.800 | +0.400 |
| +1.500 | +1.400 | +0.000 | +0.900 |
| +0.800 | +0.800 | +0.600 | +0.800 |
printed
```

2.4.3 See Also

`cmatrix.Matrix.op()`

2.5 Mod function

Genericity polymorphism ‘mod’ function

Parameters	Type	Description
<i>value</i>	double	The modulo value
<i>value</i>		int The modulo value

2.5.1 Returns

Matrix The modulo matrix

2.5.2 Examples

```
>>> m=rand(4)
>>> print(m)
| +12.000 | +5.000 | +10.000 | +5.000 |
| +9.000 | +10.000 | +2.000 | +2.000 |
| +10.000 | +9.000 | +0.000 | +2.000 |
| +13.000 | +9.000 | +8.000 | +16.000 |
printed
>>> print(m%2)
| +0.000 | +1.000 | +0.000 | +1.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +1.000 | +1.000 | +0.000 | +0.000 |
printed
```

2.5.3 See Also

cmatrix.Matrix.op()

2.6 Neg function

Genericity polymorphism ‘neg’ function

2.6.1 Returns

Matrix The opposite Matrix from self

2.6.2 Examples

```
>>> m=rand(3)
>>> print(m)
| +5.000 | +0.000 | +7.000 |
| +9.000 | +4.000 | +8.000 |
| +9.000 | +4.000 | +6.000 |
printed
>>> print(-m)
| -5.000 | -0.000 | -7.000 |
| -9.000 | -4.000 | -8.000 |
| -9.000 | -4.000 | -6.000 |
printed
```

2.6.3 See Also

`cmatrix.Matrix.op()`

2.7 Str function

Genericity polymorphism 'str' function

2.7.1 Examples

```
>>> m=rand(4)
>>> print(m)
| +7.000 | +9.000 | +14.000 | +3.000 |
| +0.000 | +7.000 | +14.000 | +12.000 |
| +2.000 | +12.000 | +11.000 | +7.000 |
| +3.000 | +16.000 | +11.000 | +7.000 |
printed
```

2.7.2 See Also

`cmatrix.Matrix.Print()`

2.8 Len function

Genericity polymorphism 'len' function

2.8.1 Examples

```
>>> m=rand(4)
>>> len(m)
16
```

2.9 Getitem function

Genericity polymorphism 'getitem' function.

Parameters	type	Description
<i>key</i>	tuple	i,j values
<i>key</i>	tuple of tuple	(i,j),(k,l) values (in case of submatrix extraction)

2.9.1 Returns

double The (i,j) item from the self Matrix

Matrix the (i,j) => (k,l) sub matrix extracted from self Matrix

2.9.2 Examples

```
>>> m=rand(4)
>>> print(m)
| +9.000 | +13.000 | +5.000 | +15.000 |
| +12.000 | +6.000 | +5.000 | +4.000 |
| +2.000 | +5.000 | +4.000 | +2.000 |
| +15.000 | +15.000 | +8.000 | +9.000 |
printed
>>> m[0,0]
9.0
>>> print(m[(0,0),(2,2)])
| +9.000 | +13.000 |
| +12.000 | +6.000 |
printed
```

2.9.3 See Also

cmatrix.Matrix.get_ij() *cmatrix.Matrix.xtract_sub_matrix()*

2.10 Setitem function

Genericity polymorphism ‘setitem’ function

Parameters	type	Description
<i>key</i>	tuple	i,j values
<i>value</i>	double	the value to set
<i>value</i>	Matrix	the sub-matrix to set from (i,j) arg

2.10.1 Returns

Matrix The self matrix updated with new value(s)

2.10.2 Examples

```
>>> m=Matrix(4,4)
>>> insert=rand(2)
>>> print(insert)
| +3.000 | +4.000 |
| +3.000 | +1.000 |
printed
>>> m[0,0]=insert
>>> print(m)
| +3.000 | +4.000 | +0.000 | +0.000 |
| +3.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
printed
>>> m[3,3]=10.123456
>>> print(m)
| +3.000 | +4.000 | +0.000 | +0.000 |
| +3.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +10.123 |
printed
```

2.10.3 See Also

`cmatrix.set_ij()` `cmatrix.insert_sub_matrix()`

2.11 Eq function

Genericity polymorphism ‘eq’ function

Parameters	type	Description
<i>mat</i>	Matrix	The matrix to test

2.11.1 Returns

bint Return the test of equality between the self matrix and the mat argument.

2.11.2 Examples

```
>>> m=rand(3)
>>> n=m.clone()
>>> print(m==n)
True
```

2.11.3 See Also

`cmatrix.Matrix.same_size()` `cmatrix.Matrix.size_r()` `cmatrix.Matrix.size_c()` `cmatrix.Matrix.get_ij()`

2.12 Ne function

Genericity polymorphism ‘ne’ function

Parameters	type	Description
<i>mat</i>	Matrix	The matrix to test

2.12.1 Examples

```
>>> m=rand(3)
>>> n=rand(3)
>>> print(m==n)
False
```

2.12.2 Returns

bint Return the test of non-equality between the self matrix and the mat argument.

2.13 Le function

Genericity polymorphism ‘le’ function

Parameters	type	Description
<i>mat</i>	Matrix	The matrix to test

2.13.1 Examples

```
>>> m=rand(4)
>>> n=rand(3)
>>> print(n<=m)
True
>>> print(m<=n)
False
```

2.13.2 Returns

bint Return the test less or equal between the length of the self matrix and the length of mat argument.

2.14 Ge function

Genericity polymorphism 'ge' function

Parameters	type	Description
<i>mat</i>	Matrix	The matrix to test

2.14.1 Returns

bint Return the test greater or equal between the length of the self matrix and the length of mat argument.

2.14.2 Examples

```
>>> m=rand(4)
>>> n=rand(3)
>>> print(m>=n)
True
>>> print(n>=m)
False
```

2.15 Lt function

Genericity polymorphism 'lt' function

Parameters	type	Description
<i>mat</i>	Matrix	The matrix to test

2.15.1 Returns

bint Return the test less than between the length of the self matrix and the length of mat argument.

2.15.2 Examples

```
>>> m=rand(4)
>>> n=rand(3)
>>> print(n<m)
True
>>> print(m<n)
False
```

2.16 Gt function

Genericity polymorphism 'gt' function

Parameters	type	Description
<i>mat</i>	Matrix	The matrix to test

2.16.1 Returns

bint Return the test greater than between the length of the self matrix and the length of mat argument.

2.16.2 Examples

```
>>> m=rand(4)
>>> n=rand(3)
>>> print(m>n)
True
>>> print(n>m)
False
```

2.17 Contains function

Genericity polymorphism 'contains' function

Parameters	type	Description
<i>mat</i>	Matrix	The matrix to test

2.17.1 Returns

bint Return the appartenance test ($x \in A$) wher x i the submatrix mat as arguement and A is the self matrix

2.17.2 Examples

```
>>> m=rand(3)
>>> main=Matrix(5,5)
>>> main[1,1]=m
>>> print(main)
| +0.000 | +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +9.000 | +6.000 | +7.000 | +0.000 |
| +0.000 | +6.000 | +6.000 | +4.000 | +0.000 |
| +0.000 | +1.000 | +3.000 | +1.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 | +0.000 |
printed
>>> print(m)
```

(continues on next page)

(continued from previous page)

```
| +9.000 | +6.000 | +7.000 |  
| +6.000 | +6.000 | +4.000 |  
| +1.000 | +3.000 | +1.000 |  
printed  
>>> print(m in main)  
True  
>>> n=rand(3)  
>>> print(n)  
| +5.000 | +9.000 | +9.000 |  
| +8.000 | +8.000 | +1.000 |  
| +7.000 | +3.000 | +0.000 |  
printed  
>>> print(n in main)  
False
```

2.17.3 See Also

`op()`

API FUNCTIONS

Here, you will find relative documentation about utility functions, containing an algorithm of sequence recognition.

3.1 Matrix generators

Here the matrix generators with automatic memory allocation.

<i>cmatrix.Laplacian_mean</i>	Laplacian mean matrix generator.
<i>cmatrix.gaussian</i>	Generate Gaussian square base matrix.
<i>cmatrix.mean_filter</i>	Generate a mean filter matrix.
<i>cmatrix.rand_perm</i>	Generate a random permutation matrix size x size
<i>cmatrix.unit</i>	Generate the unit matrix.
<i>cmatrix.zeros</i>	Generate the zeros matrix.

3.1.1 cmatrix.Laplacian_mean

`cmatrix.Laplacian_mean()`

Laplacian mean matrix generator. The order has been programmed until 3.

Parameters	type	Description
<i>order</i>	int	the order of the computed laplacian

Returns

Matrix The laplacian mean matrix

See also:

`set_ij()`

`size_r()`

`size_c()`

`get_ij()`

Examples

```
>>> print(Laplacian_mean(3))
| -0.004 | -0.008 | -0.016 | -0.008 | -0.004 |
| -0.008 | -0.016 | -0.031 | -0.016 | -0.008 |
| -0.016 | -0.031 | +0.062 | -0.031 | -0.016 |
| -0.008 | -0.016 | -0.031 | -0.016 | -0.008 |
| -0.004 | -0.008 | -0.016 | -0.008 | -0.004 |
printed
```

3.1.2 cmatrix.gaussian

`cmatrix.gaussian()`

Generate Gaussian square base matrix.

Can be used as a generator of differents gaussian “filter type” matrix using an additionnal function mapped to each value (to convolve self matrix for example).

Two mode are available :

- 0 => binomial distribution
- 1 => regular distribution

In case of regular distribution, you have to instance an odd sized matrix

Parameters	Type	Description
<i>size</i>	int	the size of the square gaussain generated
<i>mode</i>	int	• 1 : regular distribution • 0 : binomial distribution
<i>norm</i>	int	• 1 : normalize the filter mask • 0 : don't normalize

Returns

Matrix The generated gaussian matrix

Examples

```
>>> print(gaussian(3,0,0))
| +1.000 | +2.000 | +1.000 |
| +2.000 | +4.000 | +2.000 |
| +1.000 | +2.000 | +1.000 |
printed
>>> print(gaussian(3,1,0))
| +1.000 | +2.000 | +1.000 |
| +2.000 | +4.000 | +2.000 |
| +1.000 | +2.000 | +1.000 |
printed
>>> print(gaussian(3,0,1))
| +0.062 | +0.125 | +0.062 |
| +0.125 | +0.250 | +0.125 |
| +0.062 | +0.125 | +0.062 |
printed
>>> print(gaussian(3,1,1))
| +0.062 | +0.125 | +0.062 |
| +0.125 | +0.250 | +0.125 |
| +0.062 | +0.125 | +0.062 |
printed
```

3.1.3 cmatrix.mean_filter

`cmatrix.mean_filter()`

Generate a mean filter matrix.

Can be used in two mode :

- 0 => laplacian-like mean first order
- 1 => regular mean matrix

Parameters	Type	Description
<i>size</i>	int	The size of the filter
<i>mode</i>	int	Int value representing two mean mode : • 0 => laplacian-like mean first order • 1 => regular mean matrix
<i>order</i>	int	In case of Laplacian-like mean, define the order of the filter

Returns

Matrix The computed mean filter matrix

See also:

`set_ij()`
`insert_sub_matrix()`
`abs()`

Examples

```
>>> print(mean_filter(3,0))
| -0.125 | -0.125 | -0.125 |
| -0.125 | +1.000 | -0.125 |
| -0.125 | -0.125 | -0.125 |
printed
>>> print(mean_filter(3,0,2))
| -0.021 | -0.042 | -0.021 |
| -0.042 | +0.083 | -0.042 |
| -0.021 | -0.042 | -0.021 |
printed
>>> print(mean_filter(3,0,3))
| +0.062 | +0.062 | +0.062 |
| +0.062 | +0.062 | +0.062 |
| +0.062 | +0.062 | +0.062 |
printed
```

3.1.4 cmatrix.rand_perm

`cmatrix.rand_perm()`

Generate a random permutation matrix size x size

Parameters	Type	Description
<i>size</i>	int	The size of the square matrix

Returns

1/0 Matrix A random-generated permutation matrix

See also:

`set_ij()`

Examples

```
>>> m=rand_perm(4)
>>> print(m)
| +0.000 | +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
printed
```

3.1.5 cmatrix.unit

`cmatrix.unit()`

Generate the unit matrix.

Parameters	Type	Description
<i>size</i>	int	the size of the square matrix

Returns

Matrix The unit matrix

See also:

`set_ij()`

Examples

```
>>> u=unit(4)
>>> print(u)
| +1.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +1.000 |
printed
```

3.1.6 cmatrix.zeros

`cmatrix.zeros()`

Generate the zeros matrix.

Parameters	type	Description
<i>size_r</i>	int	the row size
<i>size_c</i>	int	the columns size

Returns

Matrix The zeros matrix

Examples

```
>>> m=zeros(4,4)
>>> print(m)
| +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
printed
```

3.2 Utility

Here, the utility functions definitions and usage.

<code>cmatrix.abs</code>	Get the absolute value of the given argument.
<code>cmatrix.complement</code>	Return the x complement in the given base (generic algorithm).
<code>cmatrix.complement_at</code>	Utility function computing the complement from x and base.
<code>cmatrix.isalpha</code>	Test if the given arguments is alphanum.
<code>cmatrix.mirror</code>	Local util mirror constructor to Pascal triangle generator.
<code>cmatrix.Pascal_triangle</code>	Pascal triangle generator of dimension dim.
<code>cmatrix.reverse</code>	I think comments are not necessary for this function.
<code>cmatrix.swap</code>	Swap a and b as double values.
<code>cmatrix.switch</code>	Redefined C-like switch.
<code>cmatrix.timeit</code>	Time a function using @timeit syntax.
<code>cmatrix.tri</code>	

3.2.1 cmatrix.abs

`cmatrix.abs()`

Get the absolute value of the given argument.

Parameters	type	Description
<i>x</i>	double	The raw value to treat

Returns

double The absolute value of x

Examples

```
>>> m=-123
>>> n=123
>>> abs(m)
123
>>> abs(n)
123
```

3.2.2 cmatrix.complement

`cmatrix.complement()`

Return the x complement in the given base (generic algorithm).

Parameters	type	Description
<i>x</i>	int	The value to complement
<i>base</i>	int	The numeric base as reference (2 by default)

Returns

double The x complement in given base

See also:

[`split\_number\(\)`](#)

[`complement\_at\(\)`](#)

Examples

```
>>> num=123456
>>> base=10
>>> cp=complement(num,base)
>>> print(cp)
876543
>>> num=11100101
>>> base=2
>>> cp=complement(num,base)
>>> print(cp)
11010
```

3.2.3 cmatrix.complement_at

`cmatrix.complement_at()`

Utility function computing the complement from x and base.

Parameters	type	Description
<i>x</i>	int	the value to base-complement
<i>base</i>	int	the numeric base as reference (2 by default)

Returns

int the 1st level complement (on one digit)

Examples

```
>>> num=7
>>> base=10
>>> cp=complement_at(num,base)
>>> print(cp)
2
>>> num=2
>>> base=3
>>> cp=complement_at(num,base)
>>> print(cp)
0
```

3.2.4 cmatrix.isalpha

`cmatrix.isalpha()`

Test if the given arguments is alphanum.

Returns

Bool The tested condition

3.2.5 cmatrix.mirror

`cmatrix.mirror()`

Local util mirror constructor to Pascal triangle generator.

Parameters	Type	Description
<i>array</i>	int list	The list containing the first half of the binomial coefficients suit.
<i>dim</i>	int	The dimension of the binomial coefficients suit.

Returns

list Pascal triangle mirror line from an half size one

Examples

```
>>> mirror([1,3],4)
[1, 3, 3, 1]
```

3.2.6 cmatrix.Pascal_triangle

`cmatrix.Pascal_triangle()`

Pascal triangle generator of dimension dim.

Parameters	Type	Description
<i>dim</i>	int	Dimension of the triangle

Returns

int list Binomial coefficients computed.

See also:

mirror()

Examples

```
>>> for i in range(0,10):
...     print(Pascal_triangle(i))
...
[1]
[1, 1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
```

3.2.7 cmatrix.reverse

cmatrix.reverse()

I think comments are not necessary for this function.

3.2.8 cmatrix.swap

cmatrix.swap()

Swap a and b as double values.

Parameters	type	Description
<i>a</i>	double	value to swap
<i>b</i>	double	value to swap

Returns

tuple the swapped values

Examples

```
>>> from cmatrix import *
>>> a=123
>>> b=456
>>> swap(a,b)
(456.0, 123.0)
```

3.2.9 cmatrix.switch

`cmatrix.switch()`

Redefined C-like switch. Could also be used as an inverse switch where *x* is a value and *arg* the differents variables to test.

Parameters	Type	Description
<i>x</i>	string	the variable name to switch
<i>arg</i>	string/generic	the differents states to switch, defined mod 2 : “string”,function,”string”,function,...

Returns

Generic The function or action corresponding to the switch entry.

Examples

```
>>> test1=False
>>> test2=False
>>> test3=True
>>> test4=False
>>> val=switch(True,
...    test1,1,
...    test2,2,
...    test3,3,
...    test4,4)
>>> print(val)
3
>>> val=switch(test3,
...    True,3,
...    False,4)
>>> print(val)
3
```

3.2.10 cmatrix.timeit

`cmatrix.timeit()`

Time a function using @timeit syntax.

Parameters	Type	Description
<i>func</i>	Function	The function to be stopwatch

Returns

Executed decorated function with originals arguments

Examples

```
>>> from matrix import *
>>> @timeit
... def isalpha(char):
...     return (char in ['0','1','2','3','4','5','6','7','8','9'])
...
>>> isalpha('9')
<function isalpha at 0x0337B6F0> executed 10 000 times in 0.011322021484375 = 1.
↪1322021484375e-06 /s.
True
```

3.2.11 cmatrix.tri

`cmatrix.tri()`

3.3 Sequence recognition algorithm

Here, the sequence recognition algorithm. The algorithm is working in three steps:

- From the original data as raw list, I create mutants list containing all the subset of the main list
- I consider each mutant as a claiming sequence
- I verify and count (if verified) the number of repetition

<code>cmatrix.count_seq</code>	Count the number of sequences repetition in the given data segment as list.
<code>cmatrix.create_mutants</code>	Create the 'mutants lists' from the original one.
<code>cmatrix.cut</code>	Cut the liste from the pivot (as double argument).
<code>cmatrix.find_invariant</code>	Find the invariant part since the beggining until the sequence repetitions start.
<code>cmatrix.find_max</code>	Search into the list of list containing all the claiming sequences the longest one.
<code>cmatrix.find_mul_seq</code>	The main sequence finder algorithm.
<code>cmatrix.find_seq</code>	Search a sequence into the parameter list.

continues on next page

Table 3 – continued from previous page

<code>cmatrix.find_seq_in_list</code>	Test if the given sequence have been found in the raw data (given as list)
<code>cmatrix.split_data</code>	A data splitter to treat a large amount of data.
<code>cmatrix.split_number</code>	Split a number by heigth in decimal base.

3.3.1 cmatrix.count_seq

`cmatrix.count_seq()`

Count the number of sequences repetition in the given data segment as list.

Parameters	type	Description
<i>liste</i>	list	The liste containing the given sequence
<i>seq</i>	list	The sequence as list

Returns

int The number of repetition of the sequence in da list

Examples

```
>>> l=[1,2,3,7,3,4,7,3,4,7,3,4]
>>> seq=[7,3,4]
>>> count_seq(l,seq)
3
```

3.3.2 cmatrix.create_mutants

`cmatrix.create_mutants()`

Create the ‘mutants lists’ from the original one.

Two mode are available :

- Mutants created from the begin
- Mutants created from the end

Parameters	type	Description
<i>liste</i>	list	The original liste to generate mutants
<i>mode</i>	int	The mode define the way to generate mutants : – 0 => The mutants lists are cutted since the be-gining – 1 => The mutants lists are cutted since the end

Returns

list list A list of list containing all the mutants generated

Examples

```
>>> l=[1,2,3,4,5,6,7]
>>> create_mutants(l)
[[2, 3, 4, 5, 6, 7], [3, 4, 5, 6, 7], [4, 5, 6, 7], [5, 6, 7], [6, 7], [7]]
```

3.3.3 cmatrix.cut**cmatrix.cut()**

Cut the liste from the pivot (as double argument).

Parameters	type	Description
<i>liste</i>	list	The liste to cut
<i>piv</i>	double	The pivot value as cutter delimiter

Returns

list The cutted list from the pivot value

Examples

```
>>> l=[1,2,3,4,5,6]
>>> piv=3
>>> cut(l,piv)
[3, 4, 5, 6]
```

3.3.4 cmatrix.find_invariant**cmatrix.find_invariant()**

Find the invariant part since the beggining until the sequence repetitions start.

Parameters	type	Description
<i>liste</i>	list	the raw data as list
<i>seq</i>	list	the sequence found in the data list

Returns

list A list containing the unsequenced begining of liste

Examples

```
>>> n=[1,2,6,4,5,6,4,5,6,4,5,8,3,9]
>>> seq=[6,4,5]
>>> find_invariant(n,seq)
[1, 2]
```

3.3.5 cmatrix.find_max

`cmatrix.find_max()`

Search into the list of list containing all the claiming sequences the longest one.

Parameters	type	Description
<i>liste</i>	list	A list of list to treat

Returns

list A list containing the longest list found

Examples

```
>>> l=[1,2,3,5,6,3,5,6,3,5,6]
>>> m=create_mutants(l)
>>> print(m)
[[2, 3, 5, 6, 3, 5, 6, 3, 5, 6], [3, 5, 6, 3, 5, 6, 3, 5, 6], [5, 6, 3, 5, 6, 3, 5, ↵
↵6], [6, 3, 5, 6, 3, 5, 6], [3, 5, 6, 3, 5, 6], [5, 6, 3, 5, 6], [6, 3, 5, 6], [3, ↵
↵5, 6], [5, 6], [6]]
>>> find_max(m)
[2, 3, 5, 6, 3, 5, 6, 3, 5, 6]
```

3.3.6 cmatrix.find_mul_seq

`cmatrix.find_mul_seq()`

The main sequence finder algorithm. Return the sequence (if founded) with number of repetition. First thought as data compressor, it should be used as ARN/ADN sequence analyzer :) The algorithm use the various micro-function written above.

Parameters	type	Description
<i>liste</i>	list	The raw data to treat as list

Returns

tuple A tuple containing the sequence as list and the number of repetition as int (if found)

See also:

[`create\_mutants\(\)`](#)

[`find\_seq\_in\_list\(\)`](#)

`count_seq()`
`suppr_double()`
`find_max()`

Examples

```
>>> l=[1,2,3,7,3,4,7,3,4,7,3,4]
>>> m=[4,2,1,4,2,1,4,2,1,5,7,9]
>>> n=[1,2,6,4,5,6,4,5,6,4,5,8,3,9]
>>> find_mul_seq(l)
([7, 3, 4], 3)
>>> find_mul_seq(m)
([4, 2, 1], 3)
>>> find_mul_seq(n)
([6, 4, 5], 3)
```

3.3.7 cmatrix.find_seq

`cmatrix.find_seq()`

Search a sequence into the parameter list. This is the first naïve step of the algorithm.

Parameters	type	Description
<i>liste</i>	list	The raw data to treat as list

Returns

list A list containing the sequence repeated from the beginning to the end if found

See also:

`find_seq_in_list()`

Examples

```
>>> find_seq([1,2,3,1,2,3,1,2,3])
[1, 2, 3]
```

3.3.8 cmatrix.find_seq_in_list

`cmatrix.find_seq_in_list()`

Test if the given sequence have been found in the raw data (given as list)

Parameters	type	Description
<i>seq</i>	list	The sequence to search in liste
<i>liste</i>	list	Raw data as a list

Returns

bint

- True => The sequence has been found
- False => The sequence hasn't been found

Examples

```
>>> seq=[3,7,5]
>>> l=[1,2,3,4,5,3,7,5,3,7,5,3,7,5]
>>> find_seq_in_list(seq,l)
True
>>> seq=[9,8,7]
>>> l=[1,2,3,4,5,6,7]
>>> find_seq_in_list(seq,l)
False
```

3.3.9 cmatrix.split_data**cmatrix.split_data()**

A data splitter to treat a large amount of data. Must be improved to treat efficiently a large amount of data.

Parameters	type	Description
<i>data</i>	list	The raw data as list
<i>size_r</i>	int	The size_row arguments (considering data as a Matrix)

Returns

list of list The splitted data into equal size parts

3.3.10 cmatrix.split_number**cmatrix.split_number()**

Split a number by heigth in decimal base.

Returns

list The splitted number as a list

Examples

```
>>> num=654321
>>> base=10
>>> split=split_number(num,base)
>>> print(split)
[6, 5, 4, 3, 2, 1]
```


MATRIX

Here the main matrix class engine definition.

<i>cmatrix.Matrix.ij_2_ind</i>	Convert (i,j) values to raw ribbon index.
<i>cmatrix.Matrix.ind_2_ij</i>	Convert ribbon index to (i,j) values.
<i>cmatrix.Matrix.size_r</i>	Getters : size of rows
<i>cmatrix.Matrix.size_c</i>	Getters : size of columns
<i>cmatrix.Matrix.get_ij</i>	Getters : get the (i,j) value
<i>cmatrix.Matrix.set_ij</i>	Setters : set the (i,j) value
<i>cmatrix.Matrix.get_ind</i>	Getters : get the index value from raw data vector
<i>cmatrix.Matrix.same_size</i>	Assertion : same size
<i>cmatrix.Matrix.mult_compatible</i>	Assertion : multiplication compatibility
<i>cmatrix.Matrix.Print</i>	Screen Printer
<i>cmatrix.Matrix.clone</i>	Object Cloner.
<i>cmatrix.Matrix.xtract_sub_matrix</i>	Extract the [from_ij] [to_ij] matrix in the self object (if exists)
<i>cmatrix.Matrix.insert_sub_matrix</i>	Insert the given sub-matrix in the self matrix if and only if to_insert is smaller.
<i>cmatrix.Matrix.find_sub_matrix</i>	Search the sub)matrix in the self matrix.
<i>cmatrix.Matrix.op</i>	Standard operators factorization
<i>cmatrix.Matrix.sqrt</i>	Compute the self sqrt matrix Not Fixed : Do Not Use
<i>cmatrix.Matrix.converter</i>	Convert the self matrix values into a string base converted values.
<i>cmatrix.Matrix.mand</i>	Realize a big and between self matrix and mat
<i>cmatrix.Matrix.mor</i>	Realize a big or between self matrix and mat
<i>cmatrix.Matrix.mnand</i>	Realize a big nand between self matrix and mat
<i>cmatrix.Matrix.mnor</i>	Realize a big nor between self matrix and mat
<i>cmatrix.Matrix.mxor</i>	Realize a big xor between self matrix and mat
<i>cmatrix.Matrix.column</i>	Xtract the ind column of the self Matrix object.
<i>cmatrix.Matrix.line</i>	Extract the ind line of the self Matrix object.
<i>cmatrix.Matrix.swap_col</i>	Swap the two given columns of the self matrix object from the given a and b index
<i>cmatrix.Matrix.swap_line</i>	Swap the two given lines of the self matrix object from the given a and b index
<i>cmatrix.Matrix.mirror_mat</i>	Create the mirror matrix from the self matrix object from the given mode argument
<i>cmatrix.Matrix.nth_diagonal</i>	Get the nth diagonal from the current matrix and given n as a list.
<i>cmatrix.Matrix.get_coef</i>	Return the coefficient to mean a filter mask

continues on next page

Table 1 – continued from previous page

<i>cmatrix.Matrix.mul_coef</i>	
<i>cmatrix.Matrix.resize</i>	Resize the self matrix from given float ratio.
<i>cmatrix.Matrix.accumulate</i>	Sum accumulator function ruled by mode argument.
<i>cmatrix.Matrix.square</i>	Test the square propriety of the matrix.
<i>cmatrix.Matrix.transpose</i>	Transpose the self Matrix.
<i>cmatrix.Matrix.permut</i>	Do the permutation of the self matrix with the given signed permutation.
<i>cmatrix.Matrix.triangle</i>	Get the triangle (respectively up or down) of the self matrix
<i>cmatrix.Matrix.strassen</i>	Compute the multiply using the Strassen algorithm to get faster multiplication(not necessary on small matrix).
<i>cmatrix.Matrix.tensorial_product</i>	Compute the tensorial product between self Matrix and the mat argument.
<i>cmatrix.Matrix.trace</i>	Trace the current matrix
<i>cmatrix.Matrix.convolve</i>	Convolve filtering data matrix using the given mask.
<i>cmatrix.Matrix.convolve_signal</i>	Convolve self object (as a 1D signal) with given convolution mask (also as 1D signal)
<i>cmatrix.Matrix.AtoB</i>	Compute the meaned matrix between self and given mat.
<i>cmatrix.Matrix.gap</i>	Compute the absolute gap between self matrix and given mat.
<i>cmatrix.Matrix.map</i>	Map a function (as parameter) to each element of the self matrix
<i>cmatrix.Matrix.normalize</i>	Normalize the self matrix.
<i>cmatrix.Matrix.UX_B</i>	Compute the $UX=B$ system wher U is an upper matrix and B the result matrix.
<i>cmatrix.Matrix.LX_B</i>	Compute the $LX=B$ system wher L is a lower matrix and B the result matrix.
<i>cmatrix.Matrix.replace_zeros</i>	Utility function to replace the null values with 1.
<i>cmatrix.Matrix.LU</i>	Realize a LU decomposition from the self Matrix.
<i>cmatrix.Matrix.det</i>	Compute the associated determinant of the self matrix.
<i>cmatrix.Matrix.orthogonal</i>	Boolean test to verify orthogonality
<i>cmatrix.Matrix.symmetric</i>	Boolean test to verify symmetry
<i>cmatrix.Matrix.hermitian</i>	Boolean test to verify if the matrix is hermitian
<i>cmatrix.Matrix.reversal</i>	Boolean test to verify the reversability
<i>cmatrix.Matrix.count_zero</i>	Count the number of zeros in the self Matrix
<i>cmatrix.Matrix.nilpotent</i>	Boolean test to verify the nilpotence.
<i>cmatrix.Matrix.diagonal</i>	Test if the matrix have a diagonal profile.
<i>cmatrix.Matrix.triangular</i>	Test if the matrix have a triangulare profile.
<i>cmatrix.Matrix.read_from_file</i>	Read and convert a matrix from extern file.
<i>cmatrix.Matrix.write_in_file</i>	Write the current object to file (re-openable in any other Matrix object instance)
<i>cmatrix.Matrix.pack</i>	Pack the current matrix to 64bit regular float.

4.1 cmatrix.Matrix.ij_2_ind

Matrix.ij_2_ind()

Convert (i,j) values to raw ribbon index.

4.2 cmatrix.Matrix.ind_2_ij

Matrix.ind_2_ij()

Convert ribbon index to (i,j) values.

4.3 cmatrix.Matrix.size_r

Matrix.size_r()

Getters : size of rows

4.4 cmatrix.Matrix.size_c

Matrix.size_c()

Getters : size of columns

4.5 cmatrix.Matrix.get_ij

Matrix.get_ij()

Getters : get the (i,j) value

4.6 cmatrix.Matrix.set_ij

Matrix.set_ij()

Setters : set the (i,j) value

4.7 cmatrix.Matrix.get_ind

Matrix.get_ind()

Getters : get the index value from raw data vector

4.8 cmatrix.Matrix.same_size

`Matrix.same_size()`

Assertion : same size

4.9 cmatrix.Matrix.mult_compatible

`Matrix.mult_compatible()`

Assertion : multiplication compatibility

4.10 cmatrix.Matrix.Print

`Matrix.Print()`

Screen Printer

4.11 cmatrix.Matrix.clone

`Matrix.clone()`

Object Cloner.

4.12 cmatrix.Matrix.xtract_sub_matrix

`Matrix.xtract_sub_matrix()`

Extract the [from_ij] [to_ij] matrix in the self object (if exists)

Parameters	Type	Description
<i>from_ij</i>	int tuple	tuple containing top left up i,j corner
<i>to_ij</i>	int tuple	tuple containing top right down i,j corner

Returns

Matrix The extracted sub matrix

See also:

[`set\_ij\(\)`](#)

[`get\_ij\(\)`](#)

Examples

```
>>> mat=gaussian(5,0,1)
>>> xtract=mat.xtract_sub_matrix([1,1],[4,4])
>>> Print(mat)
| +0.004 | +0.016 | +0.023 | +0.016 | +0.004 |
| +0.016 | +0.062 | +0.094 | +0.062 | +0.016 |
| +0.023 | +0.094 | +0.141 | +0.094 | +0.023 |
| +0.016 | +0.062 | +0.094 | +0.062 | +0.016 |
| +0.004 | +0.016 | +0.023 | +0.016 | +0.004 |
printed
>>> Print(xtract)
| +0.062 | +0.094 | +0.062 |
| +0.094 | +0.141 | +0.094 |
| +0.062 | +0.094 | +0.062 |
printed
```

4.13 cmatrix.Matrix.insert_sub_matrix

Matrix.insert_sub_matrix()

Insert the given sub-matrix in the self matrix if and only if to_insert is smaller.

Parameters	Type	Description
<i>to_ij</i>	double tuple	The 2D coor representing the inserting point in the self matrix
<i>to_insert</i>	Matrix	The atrix to insert

See also:

size_r()

size_c()

get_ij()

Examples

```
>>> mat2=zeros(5,5)
>>> mat2.insert_sub_matrix([1,1],xtract)
>>> Print(mat2)
| +0.000 | +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.062 | +0.094 | +0.062 | +0.000 |
| +0.000 | +0.094 | +0.141 | +0.094 | +0.000 |
| +0.000 | +0.062 | +0.094 | +0.062 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 | +0.000 |
printed
```

4.14 cmatrix.Matrix.find_sub_matrix

`Matrix.find_sub_matrix()`

Search the sub)matrix in the self matrix. The given sub must be smaller than the self matrix.

Parameters	Type	Description
<i>sub</i>	Matrix	the matrix to search

Returns

Boolean

- True mean the sub)matrix is in the main matrix
- False mean the sub)matrix isn't contained in the main matrix

See also:

[`get\_ij\(\)`](#)

[`size\_r\(\)`](#)

[`size\_c\(\)`](#)

Examples

```
>>> mat=gaussian(5,0,0)
>>> Print(mat)
| +1.000 | +4.000 | +6.000 | +4.000 | +1.000 |
| +4.000 | +16.000 | +24.000 | +16.000 | +4.000 |
| +6.000 | +24.000 | +36.000 | +24.000 | +6.000 |
| +4.000 | +16.000 | +24.000 | +16.000 | +4.000 |
| +1.000 | +4.000 | +6.000 | +4.000 | +1.000 |
printed
>>> xtract=mat.xtract_sub_matrix([1,1],[4,4])
>>> Print(xtract)
| +16.000 | +24.000 | +16.000 |
| +24.000 | +36.000 | +24.000 |
| +16.000 | +24.000 | +16.000 |
printed
>>> mat2=unit(5)
>>> Print(mat2)
| +1.000 | +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +1.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +1.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 | +1.000 |
printed
>>> mat.find_sub_matrix(xtract)
True
>>> mat2.find_sub_matrix(xtract)
False
```

4.15 cmatrix.Matrix.op

Matrix.op()

Standard operators factorization

Eventually complete the description with details as :

The given operator as a string is interpreted and the corresponding computing is returned

Parameters	Type	Description
<i>mat2</i>	Matrix	The matrix second term
<i>op</i>	String	the operation to realize between : • - : add • - : sub • - : multiply • & : logic and • : logic or • ~ : logic nand • ⌘ : logic nor • ^ : logic xor

See also:

size_r()

size_c()

same_size()

get_ij()

set_ij()

mult_compatible()

transpose()

Examples

```
>>> # Decimal Operator Factorisation
>>> t=gaussian(5,1,0)
>>> Print(t) # initializing
| +1.000 | +2.000 | +4.000 | +2.000 | +1.000 |
| +2.000 | +4.000 | +8.000 | +4.000 | +2.000 |
| +4.000 | +8.000 | +16.000 | +8.000 | +4.000 |
| +2.000 | +4.000 | +8.000 | +4.000 | +2.000 |
| +1.000 | +2.000 | +4.000 | +2.000 | +1.000 |
printed
>>> u=unit(5)*100
>>> Print(u) # initializing
| +100.000 | +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +100.000 | +0.000 | +0.000 | +0.000 |
```

(continues on next page)

(continued from previous page)

```

| +0.000 | +0.000 | +100.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +100.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 | +100.000 |
printed
>>> Print(t.op(u,'+')) # applying operator
| +101.000 | +2.000 | +4.000 | +2.000 | +1.000 |
| +2.000 | +104.000 | +8.000 | +4.000 | +2.000 |
| +4.000 | +8.000 | +116.000 | +8.000 | +4.000 |
| +2.000 | +4.000 | +8.000 | +104.000 | +2.000 |
| +1.000 | +2.000 | +4.000 | +2.000 | +101.000 |
printed
>>> Print(t.op(u,'-')) # applying operator
| -99.000 | +2.000 | +4.000 | +2.000 | +1.000 |
| +2.000 | -96.000 | +8.000 | +4.000 | +2.000 |
| +4.000 | +8.000 | -84.000 | +8.000 | +4.000 |
| +2.000 | +4.000 | +8.000 | -96.000 | +2.000 |
| +1.000 | +2.000 | +4.000 | +2.000 | -99.000 |
printed
>>> Print(t.op(u,'*')) # applying operator
| +100.000 | +200.000 | +400.000 | +200.000 | +100.000 |
| +200.000 | +400.000 | +800.000 | +400.000 | +200.000 |
| +400.000 | +800.000 | +1600.000 | +800.000 | +400.000 |
| +200.000 | +400.000 | +800.000 | +400.000 | +200.000 |
| +100.000 | +200.000 | +400.000 | +200.000 | +100.000 |
printed
>>> # Binary Operator Factorisation
>>> t=rand_perm(3)
>>> Print(t)
| +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 |
printed
>>> u=unit(3)
>>> Print(u)
| +1.000 | +0.000 | +0.000 |
| +0.000 | +1.000 | +0.000 |
| +0.000 | +0.000 | +1.000 |
printed
>>> # Logical And
>>> Print(t.op(u,'&'))
| +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 |
printed
>>> # Logical Or
>>> Print(t.op(u,'|'))
| +1.000 | +0.000 | +0.000 |
| +0.000 | +1.000 | +1.000 |
| +0.000 | +1.000 | +1.000 |
printed
>>> # Logical Nand
>>> Print(t.op(u,'~'))

```

(continues on next page)

(continued from previous page)

```
| +0.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 |
printed
>>> # Logical Nor
>>> Print(t.op(u, '⌘'))
| +0.000 | +1.000 | +1.000 |
| +1.000 | +0.000 | +0.000 |
| +1.000 | +0.000 | +0.000 |
printed
>>> # Logical Xor
>>> Print(t.op(u, '⌘'))
| +0.000 | +0.000 | +0.000 |
| +0.000 | +1.000 | +1.000 |
| +0.000 | +1.000 | +1.000 |
printed
```

4.16 cmatrix.Matrix.sqrt

Matrix.sqrt()
Compute the self sqrt matrix Not Fixed : Do Not Use

Parameters	type	Description
precision	int	Precision parameter

Returns
Matrix The sqrt matrix from self matrix

See also:

- square()
- clone()
- size_r()
- inverse()

4.17 cmatrix.Matrix.converter

Matrix.converter()
Convert the self matrix values into a string base converted values.

Parameters	Type	Description
<i>mode</i>	int	int value representing the function profile: • 0 => binary conversion • 1 => octal conversion • 2 => hexadecimal conversion

See also:

`size_r()`

`size_c()`

`get_ij()`

Examples

```

>>> mat=gaussian(3,0,0)
>>> mat.converter(0)
>>> Print(mat)
| 0b1      | | 0b10      | | 0b1      |
| 0b10     | | 0b100     | | 0b10     |
| 0b1      | | 0b10      | | 0b1      |
printed
>>> mat=gaussian(3,0,0)
>>> mat.converter(1)
>>> Print(mat)
| 0o1      | | 0o2      | | 0o1      |
| 0o2      | | 0o4      | | 0o2      |
| 0o1      | | 0o2      | | 0o1      |
printed
>>> mat=gaussian(3,0,0)
>>> mat.converter(2)
>>> Print(mat)
| 0x1      | | 0x2      | | 0x1      |
| 0x2      | | 0x4      | | 0x2      |
| 0x1      | | 0x2      | | 0x1      |
printed

```

4.18 cmatrix.Matrix.mand

`Matrix.mand()`

Realize a big and between self matrix and mat

Parameters	type	Description
<i>mat</i>	Matrix*	the matrix operand

Returns

Matrix The matrix obtained from a big and between self matrix and given matrix argument

See also:

[op\(\)](#)

Examples

```
>>> m=Matrix(4,4,1)
>>> n=rand_perm(4)
>>> print(m)
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
printed
>>> print(n)
| +0.000 | +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
printed
>>> print(m.mand(n))
| +0.000 | +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
printed
```

4.19 cmatrix.Matrix.mor

Matrix.mor()

Realize a big or between self matrix and mat

Parameters	type	Description
<i>mat</i>	Matrix*	the matrix operand

Returns

Matrix The matrix obtained from a big or between self matrix and given matrix argument

See also:

[op\(\)](#)

Examples

```
>>> m=Matrix(4,4,1)
>>> n=rand_perm(4)
>>> print(m)
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
printed
>>> print(n)
| +0.000 | +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
printed
>>> print(m.mor(n))
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
printed
```

4.20 cmatrix.Matrix.mnand

Matrix.mnand()

Realize a big nand between self matrix and mat

Parameters	type	Description
<i>mat</i>	Matrix*	the matrix operand

Returns

Matrix The matrix obtained from a big nand between self matrix and given matrix argument

See also:

op()

Examples

```
>>> m=Matrix(4,4,1)
>>> n=rand_perm(4)
>>> print(m)
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
printed
```

(continues on next page)

(continued from previous page)

```

>>> print(n)
| +0.000 | +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
printed
>>> print(m.mnand(n))
| +1.000 | +1.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +1.000 | +1.000 |
printed

```

4.21 cmatrix.Matrix.mnor

Matrix.mnor()

Realize a big nor between self matrix and mat

Parameters	type	Description
<i>mat</i>	Matrix*	the matrix operand

Returns

Matrix The matrix obtained from a big nor between self matrix and given matrix argument

See also:

[*op\(\)*](#)

Examples

```

>>> m=Matrix(4,4,1)
>>> n=rand_perm(4)
>>> print(m)
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
printed
>>> print(n)
| +0.000 | +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
printed
>>> print(m.mnor(n))
| +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |

```

(continues on next page)

(continued from previous page)

```
| +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
printed
```

4.22 cmatrix.Matrix.mxor

Matrix.mxor()

Realize a big xor between self matrix and mat

Parameters	type	Description
<i>mat</i>	Matrix*	the matrix operand

Returns

Matrix The matrix obtained from a big xor between self matrix and given matrix argument

See also:

[*op\(\)*](#)

Examples

```
>>> m=Matrix(4,4,1)
>>> n=rand_perm(4)
>>> print(m)
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
printed
>>> print(n)
| +0.000 | +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
printed
>>> print(m.mxor(n))
| +1.000 | +1.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +1.000 | +1.000 |
printed
```

4.23 cmatrix.Matrix.column

Matrix.column()

Xtract the ind column of the self Matrix object.

Parameters	type	Description
<i>ind</i>	int	the indice of the column to xtract

Returns

List The list containig all the matrix column values

Examples

```
>>> mat=unit(4)
>>> mat.column(0)
[1, 0, 0, 0]
>>> mat.column(1)
[0, 1, 0, 0]
>>> mat.column(2)
[0, 0, 1, 0]
>>> mat.column(3)
[0, 0, 0, 1]
```

4.24 cmatrix.Matrix.line

Matrix.line()

Extract the ind line of the self Matrix object.

Parameters	type	Description
<i>ind</i>	int	the indice of the line to xtract

Returns

List The list containig all the matrix line values

See also:

get_ij()

Examples

```
>>> mat=unit(4)
>>> mat.line(0)
[1, 0, 0, 0]
>>> mat.line(1)
[0, 1, 0, 0]
>>> mat.line(2)
[0, 0, 1, 0]
>>> mat.line(3)
[0, 0, 0, 1]
```

4.25 cmatrix.Matrix.swap_col

Matrix.**swap_col()**

Swap the two given columns of the self matrix object from the given a and b index

Parameters	Type	Description
<i>a</i>	int	the first column index
<i>b</i>	int	the second column index

Returns

Matrix The matrix with a-th and b-th column swapped

See also:

[*size_r\(\)*](#)

[*get_ij\(\)*](#)

[*set_ij\(\)*](#)

Examples

```
>>> m=rand(4)
>>> print(m)
| +4.000 | +14.000 | +10.000 | +16.000 |
| +7.000 | +9.000 | +13.000 | +9.000 |
| +3.000 | +7.000 | +7.000 | +6.000 |
| +3.000 | +2.000 | +9.000 | +16.000 |
printed
>>> print(m.swap_col(0,2))
| +10.000 | +14.000 | +4.000 | +16.000 |
| +13.000 | +9.000 | +7.000 | +9.000 |
| +7.000 | +7.000 | +3.000 | +6.000 |
| +9.000 | +2.000 | +3.000 | +16.000 |
printed
```

4.26 cmatrix.Matrix.swap_line

Matrix.swap_line()

Swap the two given lines of the self matrix object from the given a and b index

Parameters	Type	Description
<i>a</i>	int	the first line index
<i>b</i>	int	the second line index

Returns

Matrix The matrix with a-th and b-th line swaped

See also:

[*size_r\(\)*](#)

[*get_ij\(\)*](#)

[*set_ij\(\)*](#)

Examples

```
>>> m=rand(4)
>>> print(m)
| +13.000 | +5.000 | +8.000 | +1.000 |
| +13.000 | +13.000 | +6.000 | +5.000 |
| +16.000 | +0.000 | +0.000 | +0.000 |
| +10.000 | +2.000 | +16.000 | +13.000 |
printed
>>> print(m.swap_line(0,2))
| +16.000 | +0.000 | +0.000 | +0.000 |
| +13.000 | +13.000 | +6.000 | +5.000 |
| +13.000 | +5.000 | +8.000 | +1.000 |
| +10.000 | +2.000 | +16.000 | +13.000 |
printed
```

4.27 cmatrix.Matrix.mirror_mat

Matrix.mirror_mat()

Create the mirror matrix from the self matrix object from the given mode argument

Parameters	Type	Description
<i>mode</i>	int	mode define the profile of algorithm between : • 0 => vertical mirror • 1 => horizontal mirror

Returns**Matrix** The self mirror matrix

See also:

`size_r()``size_c()``clone()``transpose()``swap_col()`**Examples**

```

>>> m=rand(4)
>>> print(m)
| +5.000 | +14.000 | +8.000 | +6.000 |
| +14.000 | +11.000 | +4.000 | +0.000 |
| +2.000 | +14.000 | +12.000 | +0.000 |
| +16.000 | +0.000 | +11.000 | +4.000 |
printed
>>> print(m.mirror_mat(0))
| +6.000 | +8.000 | +14.000 | +5.000 |
| +0.000 | +4.000 | +11.000 | +14.000 |
| +0.000 | +12.000 | +14.000 | +2.000 |
| +4.000 | +11.000 | +0.000 | +16.000 |
printed
>>> print(m.mirror_mat(1))
| +16.000 | +0.000 | +11.000 | +4.000 |
| +2.000 | +14.000 | +12.000 | +0.000 |
| +14.000 | +11.000 | +4.000 | +0.000 |
| +5.000 | +14.000 | +8.000 | +6.000 |
printed

```

4.28 cmatrix.Matrix.nth_diagonal

Matrix.nth_diagonal()

Get the nth diagonal from the current matrix and given n as a list.

Parameters	Type	Description
<i>n</i>	int	the diagonal index to xtract
<i>mode</i>	int	mode define the profile of algorithm between : • 0 => ascendant • 1 => descendent
<i>rec</i>	boolean	Recursion manager

Returns

List the list containing the expected n-th diagonal

See also:

`size_r()`

`nth_diagonal()`

`mirror_mat()`

`get_ij()`

`reverse()`

Examples

```
>>> m=rand(4)
>>> print(m)
| +13.000 | +6.000 | +9.000 | +9.000 |
| +12.000 | +16.000 | +2.000 | +13.000 |
| +3.000 | +8.000 | +5.000 | +2.000 |
| +12.000 | +11.000 | +11.000 | +9.000 |
printed
>>> for i in range(0,7):
...     print(m.nth_diagonal(i,0))
...
[13.0]
[6.0, 12.0]
[9.0, 16.0, 3.0]
[9.0, 2.0, 8.0, 12.0]
[13.0, 5.0, 11.0]
[2.0, 11.0]
[9.0]
>>> for i in range(0,7):
...     print(m.nth_diagonal(i,1))
...
[12.0]
[3.0, 11.0]
[12.0, 8.0, 11.0]
[13.0, 16.0, 5.0, 9.0]
[6.0, 2.0, 2.0]
[9.0, 13.0]
[9.0]
```

4.29 cmatrix.Matrix.get_coef

Matrix.get_coef()

Return the coefficient to mean a filter mask

Returns

float The computed coef.

See also:

[*size_r\(\)*](#)

[*size_c\(\)*](#)

[*get_ij\(\)*](#)

Examples

```
>>> mat=gaussian(3,0,0)
>>> Print(mat)
| +1.000 | +2.000 | +1.000 |
| +2.000 | +4.000 | +2.000 |
| +1.000 | +2.000 | +1.000 |
printed
>>> mat.get_coef()
0.0625
```

4.30 cmatrix.Matrix.mul_coef

Matrix.mul_coef()

4.31 cmatrix.Matrix.resize

Matrix.resize()

Resize the self matrix from given float ratio. A ratio of 0.5 mean the half matrix as a result

Parameters	Type	Description
<i>ratio</i>	float	The ratio of resizing

Returns

Matrix The proportionnaly resized matrix.

See also:

[*size_r\(\)*](#)

[*size_c\(\)*](#)

`set_ij()``get_ij()`

Examples

```

>>> from matrix import *
>>> t=rand(8)
>>> Print(t)
| +40.000 | +17.000 | +61.000 | +56.000 | +60.000 | +4.000 | +52.000 | +55.000 |
| +27.000 | +12.000 | +27.000 | +29.000 | +9.000 | +22.000 | +50.000 | +57.000 |
| +45.000 | +32.000 | +24.000 | +40.000 | +29.000 | +16.000 | +10.000 | +12.000 |
| +13.000 | +28.000 | +27.000 | +46.000 | +1.000 | +57.000 | +34.000 | +25.000 |
| +63.000 | +22.000 | +39.000 | +55.000 | +2.000 | +22.000 | +26.000 | +15.000 |
| +1.000 | +11.000 | +25.000 | +3.000 | +15.000 | +43.000 | +14.000 | +6.000 |
| +54.000 | +42.000 | +59.000 | +53.000 | +23.000 | +35.000 | +54.000 | +13.000 |
| +44.000 | +38.000 | +31.000 | +64.000 | +47.000 | +32.000 | +5.000 | +44.000 |
printed
>>> Print(t.resize(0.5))
| +40.000 | +61.000 | +60.000 | +52.000 |
| +45.000 | +24.000 | +29.000 | +10.000 |
| +63.000 | +39.000 | +2.000 | +26.000 |
| +54.000 | +59.000 | +23.000 | +54.000 |
printed

```

4.32 cmatrix.Matrix.accumulate

Matrix.accumulate()

Sum accumulator function ruled by mode argument. Realize the cumulated sum depending on direction.

Parameters	Type	Description
<i>mode</i>	int	Switch the mode between : • 0 => vectorized horizontal • 1 => vectorized vertical • 2 => matrix vertical • 3 => matrix horizontal

Returns

Matrix The accumulated matrix

See also:

`size_r()``size_c()``get_ij()``set_ij()`

Examples

```
>>> from matrix import *
>>> t=rand(4)
>>> Print(t)
| +2.000 | +3.000 | +3.000 | +2.000 |
| +5.000 | +6.000 | +0.000 | +15.000 |
| +10.000 | +6.000 | +1.000 | +1.000 |
| +14.000 | +10.000 | +9.000 | +16.000 |
printed
>>> print(t.accumulate(0).data)
[10, 26, 18, 49]
>>> print(t.accumulate(1).data)
[31, 25, 13, 34]
>>> Print(t.accumulate(2))
| +2.000 | +3.000 | +3.000 | +2.000 |
| +7.000 | +9.000 | +3.000 | +17.000 |
| +17.000 | +15.000 | +4.000 | +18.000 |
| +31.000 | +25.000 | +13.000 | +34.000 |
printed
>>> Print(t.accumulate(3))
| +2.000 | +5.000 | +8.000 | +10.000 |
| +5.000 | +11.000 | +11.000 | +26.000 |
| +10.000 | +16.000 | +17.000 | +18.000 |
| +14.000 | +24.000 | +33.000 | +49.000 |
printed
```

4.33 cmatrix.Matrix.square

Matrix.square()

Test the square propriety of the matrix.

Returns

int 1 mean square matrix 0 mean not square matrix

Examples

```
>>> mat.get_coef()
0.0625
>>> mat=unit(3)
>>> mat.square()
True
>>> mat=Matrix(1,3,0.3)
>>> mat.square()
False
```

4.34 cmatrix.Matrix.transpose

Matrix.transpose()

Transpose the self Matrix.

Returns

Matrix The transposed matrix

See also:

size_c()

size_r()

set_ij()

get_ij()

Examples

```
>>> t=Matrix(3,3,0)
>>> for i in range(0,3):
...     for j in range(0,3):
...         t.set_ij(i,j,i*j+i)
...
>>> Print(t)
| +0.000 | +0.000 | +0.000 |
| +1.000 | +2.000 | +3.000 |
| +2.000 | +4.000 | +6.000 |
printed
>>> tr=t.transpose()
>>> Print(tr)
| +0.000 | +1.000 | +2.000 |
| +0.000 | +2.000 | +4.000 |
| +0.000 | +3.000 | +6.000 |
printed
```

4.35 cmatrix.Matrix.permut

Matrix.permut()

Do the permutation of the self matrix with the given signed permutation.

Parameters	type	Description
<i>Permut</i>	int list	list representing the signed permutation ([2,0,1] mean [a,b,c] become [c,a,b])
<i>mode</i>	int	Algorithm mode : • 0 mean vertical permutation • 1 mean horizontal permutation

Returns

Matrix The permuted matrix from the given signed permutation (as vector)

See also:

size_r()

size_c()

square()

transpose()

Examples

```

>>> m=rand(4)
>>> print(m)
| +13.000 | +14.000 | +12.000 | +0.000 |
| +7.000 | +7.000 | +13.000 | +10.000 |
| +10.000 | +4.000 | +2.000 | +16.000 |
| +6.000 | +12.000 | +7.000 | +11.000 |
printed
>>> permut=[1,0,3,2]
>>> print(m.permut(permut,0))
| +14.000 | +7.000 | +4.000 | +12.000 |
| +13.000 | +7.000 | +10.000 | +6.000 |
| +0.000 | +10.000 | +16.000 | +11.000 |
| +12.000 | +13.000 | +2.000 | +7.000 |
printed
>>> print(m.permut(permut,1))
| +7.000 | +13.000 | +6.000 | +10.000 |
| +7.000 | +14.000 | +12.000 | +4.000 |
| +13.000 | +12.000 | +7.000 | +2.000 |
| +10.000 | +0.000 | +11.000 | +16.000 |
printed

```

4.36 cmatrix.Matrix.triangle

Matrix.triangle()

Get the triangle (respectively up or down) of the self matrix

Parameters	type	Description
<i>mode</i>	int	Mode mean : 1 sup, 0 inf

Returns

Matrix The triangle matrix from self matrix

See also:

[*clone\(\)*](#)

[*set_ij\(\)*](#)

Examples

```
>>> t=gaussian(5,0,0)
>>> t.triangle(0) # inferior profile
>>> Print(t)
| +1.000 | +0.000 | +0.000 | +0.000 | +0.000 |
| +4.000 | +16.000 | +0.000 | +0.000 | +0.000 |
| +6.000 | +24.000 | +36.000 | +0.000 | +0.000 |
| +4.000 | +16.000 | +24.000 | +16.000 | +0.000 |
| +1.000 | +4.000 | +6.000 | +4.000 | +1.000 |
printed
>>> t=gaussian(5,0,0)
>>> t.triangle(1) # superior profile
>>> Print(t)
| +0.000 | +4.000 | +6.000 | +4.000 | +1.000 |
| +0.000 | +0.000 | +24.000 | +16.000 | +4.000 |
| +0.000 | +0.000 | +0.000 | +24.000 | +6.000 |
| +0.000 | +0.000 | +0.000 | +0.000 | +4.000 |
| +0.000 | +0.000 | +0.000 | +0.000 | +0.000 |
printed
```

4.37 cmatrix.Matrix.strassen

Matrix.strassen()

Compute the multiply using the Strassen algorithm to get faster multiplication(not necessary on small matrix).

Strassen algorithm is evaluate in 6 steps :

- Split the self matrix into 4 equal-length submatrix
- Split the second term matrix into 4 equal-length submatrix
- Allocating memory for temporary computing
- Apply the Strassen multiplication rules (M matrix)

- Compute the C matrix from M Matrix
- Rebuild the final matrix with temporary matrix

Parameters	type	Description
<i>m2</i>	Matrix	The second member of the multiplication

Returns

Matrix The computed multiplication.

See also:

`square()`

`size_r()`

`xtract_sub_matrix()`

`set_ij()`

`get_ij()`

Examples

```
>>> m=rand(4)
>>> u=unit(4)
>>> print(m)
| +14.000 | +12.000 | +12.000 | +1.000 |
| +7.000 | +16.000 | +8.000 | +8.000 |
| +9.000 | +6.000 | +8.000 | +2.000 |
| +11.000 | +1.000 | +8.000 | +7.000 |
printed
>>> print(u)
| +1.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +1.000 |
printed
>>> print(m.strassen(u))
| +14.000 | +12.000 | +12.000 | +1.000 |
| +7.000 | +16.000 | +8.000 | +8.000 |
| +9.000 | +6.000 | +8.000 | +2.000 |
| +11.000 | +1.000 | +8.000 | +7.000 |
printed
```

4.38 cmatrix.Matrix.tensorial_product

`Matrix.tensorial_product()`

Compute the tensorial product between self Matrix and the mat argument.

Parameters	type	Description
<i>mat</i>	Matrix	The second operand matrix

Returns

Matrix The double sized matrix result of the tenso product

See also:

`size_r()`

`size_c()`

`set_ij()`

`get_ij()`

`transpose()`

Examples

```
>>> #Matrix Tensor Profile
>>> m=rand(4)
>>> tensor=Matrix(2,2)
>>> tensor[0,0]=1
>>> tensor[0,1]=1
>>> tensor[1,0]=2
>>> tensor[1,1]=2
>>> print(m)
| +14.000 | +5.000 | +4.000 | +2.000 |
| +10.000 | +0.000 | +9.000 | +3.000 |
| +14.000 | +1.000 | +3.000 | +13.000 |
| +5.000 | +0.000 | +3.000 | +3.000 |
printed
>>> print(tensor)
| +1.000 | +1.000 |
| +2.000 | +2.000 |
printed
>>> print(m.tensorial_product(tensor))
| +14.000 | +5.000 | +4.000 | +2.000 | +14.000 | +5.000 | +4.000 | +2.000 |
| +20.000 | +0.000 | +18.000 | +6.000 | +20.000 | +0.000 | +18.000 | +6.000 |
| +14.000 | +1.000 | +3.000 | +13.000 | +14.000 | +1.000 | +3.000 | +13.000 |
| +10.000 | +0.000 | +6.000 | +6.000 | +10.000 | +0.000 | +6.000 | +6.000 |
| +14.000 | +5.000 | +4.000 | +2.000 | +14.000 | +5.000 | +4.000 | +2.000 |
| +20.000 | +0.000 | +18.000 | +6.000 | +20.000 | +0.000 | +18.000 | +6.000 |
| +14.000 | +1.000 | +3.000 | +13.000 | +14.000 | +1.000 | +3.000 | +13.000 |
| +10.000 | +0.000 | +6.000 | +6.000 | +10.000 | +0.000 | +6.000 | +6.000 |
printed
```

(continues on next page)

(continued from previous page)

```

>>> #Vector Tensor Profile
>>> m=rand(4)
>>> print(m)
| +8.000 | +2.000 | +15.000 | +16.000 |
| +2.000 | +9.000 | +6.000 | +13.000 |
| +7.000 | +5.000 | +3.000 | +7.000 |
| +7.000 | +15.000 | +5.000 | +0.000 |
printed
>>> tensor=Matrix(2,1)
>>> tensor[0,0]=1
>>> tensor[1,0]=2
>>> print(m.tensorial_product(tensor))
| +8.000 | +2.000 | +15.000 | +16.000 |
| +16.000 | +4.000 | +30.000 | +32.000 |
| +8.000 | +2.000 | +15.000 | +16.000 |
| +16.000 | +4.000 | +30.000 | +32.000 |
| +2.000 | +9.000 | +6.000 | +13.000 |
| +4.000 | +18.000 | +12.000 | +26.000 |
| +2.000 | +9.000 | +6.000 | +13.000 |
| +4.000 | +18.000 | +12.000 | +26.000 |
printed

```

4.39 cmatrix.Matrix.trace

Matrix.trace()

Trace the current matrix

Returns

double The computed trace of the matrix

See also:

[*size_r\(\)*](#)

[*get_ij\(\)*](#)

Examples

```

>>> m=rand(4)
>>> print(m)
| +0.000 | +6.000 | +8.000 | +4.000 |
| +8.000 | +11.000 | +13.000 | +0.000 |
| +15.000 | +10.000 | +15.000 | +4.000 |
| +11.000 | +2.000 | +1.000 | +10.000 |
printed
>>> print(m.trace())
36.0

```

4.40 cmatrix.Matrix.convolve

`Matrix.convolve()`

Convolve filtering data matrix using the given mask.

Parameters	Type	Description
<i>mask</i>	Matrix	The convolution mask as a matrix

Returns

Matrix The convolution product result between self matrix and the given mask

See also:

[`size\_r\(\)`](#)

[`size\_c\(\)`](#)

[`insert\_sub\_matrix\(\)`](#)

Examples

```
>>> mask=gaussian(3,0,1)
>>> print(mask)
| +0.062 | +0.125 | +0.062 |
| +0.125 | +0.250 | +0.125 |
| +0.062 | +0.125 | +0.062 |
printed
>>> m=rand(5)
>>> print(m)
| +11.000 | +11.000 | +11.000 | +24.000 | +4.000 |
| +2.000 | +17.000 | +2.000 | +11.000 | +17.000 |
| +1.000 | +19.000 | +22.000 | +14.000 | +14.000 |
| +7.000 | +20.000 | +18.000 | +17.000 | +15.000 |
| +19.000 | +1.000 | +1.000 | +5.000 | +18.000 |
printed
>>> print(m.convolve(mask))
| +9.625 | +15.625 | +14.438 | +11.562 | +6.750 |
| +10.625 | +15.875 | +14.938 | +11.188 | +6.875 |
| +11.688 | +14.062 | +14.188 | +10.500 | +8.000 |
| +9.000 | +9.750 | +10.188 | +7.875 | +6.688 |
| +11.312 | +10.438 | +8.938 | +5.438 | +6.938 |
printed
```

4.41 cmatrix.Matrix.convolve_signal

`Matrix.convolve_signal()`

Convolve self object (as a 1D signal) with given convolution mask (also as 1D signal)

Parameters	Type	Description
<i>convolution_mask</i>	Matrix	1D Matrix representing the convolution mask
<i>mode</i>	Int	Define the algorithm profile between • 0 => full convolution algorithm • 1 => convolution algorithm preservng time

Returns

Matrix The 1D matrix containing the convolved signal.

See also:

`size_c()`

`set_ij()`

`get_ij()`

`size_r()`

`nth_diagonal()`

Examples

```
>>> m=Matrix(1,8)
>>> import random as r
>>> for i in range(0,8):
...     m[0,i]=r.randint(0,50)
...
>>> mask=Matrix(1,3)
>>> for i in range(0,3):
...     mask[0,i]=r.randint(0,10)
...
>>> print(m)
| +25.000 | +8.000 | +41.000 | +20.000 | +28.000 | +23.000 | +17.000 | +45.000 |
printed
>>> print(mask)
| +1.000 | +5.000 | +9.000 |
printed
>>> print(m.convolve_signal(mask))
| +25.000 | +133.000 | +306.000 | +297.000 | +497.000 | +343.000 | +384.000 | +337.
↪000 | +378.000 | +405.000 | +0.000 | +0.000 | +0.000 | +0.000 |
printed
```

4.42 cmatrix.Matrix.AtoB

Matrix.AtoB()

Compute the meaned matrix between self and given mat. Can be used to generate mutants matrix derivative/interpolated from self and mat.

Parameters	Type	Description
<i>mat</i>	Matrix	The matrix to mean with self
<i>ratio</i>	float	The balace ratio between matrix Must be in [0,1]

Returns

Matrix The linear interpolated matrix computed from the mat parameter and the ratio The ratio determine the multiplicative coefficient.

See also:

[*same_size\(\)*](#)

[*size_r\(\)*](#)

[*size_c\(\)*](#)

[*set_ij\(\)*](#)

[*get_ij\(\)*](#)

Examples

```
>>> Print(t)
| +1.000 | +4.000 | +6.000 | +4.000 | +1.000 |
| +4.000 | +16.000 | +24.000 | +16.000 | +4.000 |
| +6.000 | +24.000 | +36.000 | +24.000 | +6.000 |
| +4.000 | +16.000 | +24.000 | +16.000 | +4.000 |
| +1.000 | +4.000 | +6.000 | +4.000 | +1.000 |
printed
>>> Print(u)
| +1.000 | +0.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +1.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +1.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +0.000 | +1.000 |
printed
>>> v=t.AtoB(u) # 0.5 ration as default
>>> Print(v)
| +1.000 | +2.000 | +3.000 | +2.000 | +0.500 |
| +2.000 | +8.500 | +12.000 | +8.000 | +2.000 |
| +3.000 | +12.000 | +18.500 | +12.000 | +3.000 |
| +2.000 | +8.000 | +12.000 | +8.500 | +2.000 |
| +0.500 | +2.000 | +3.000 | +2.000 | +1.000 |
```

(continues on next page)

(continued from previous page)

```

printed
>>> v=t.AtoB(u,0.1) # Personalize ratio as the last arg
>>> Print(v)
| +1.000 | +0.400 | +0.600 | +0.400 | +0.100 |
| +0.400 | +2.500 | +2.400 | +1.600 | +0.400 |
| +0.600 | +2.400 | +4.500 | +2.400 | +0.600 |
| +0.400 | +1.600 | +2.400 | +2.500 | +0.400 |
| +0.100 | +0.400 | +0.600 | +0.400 | +1.000 |
printed

```

4.43 cmatrix.Matrix.gap

Matrix.gap()

Compute the absolute gap between self matrix and given mat. Can be used to compare probabilities matrix.

Parameters	Type	Description
<i>mat</i>	Matrix	The matrix to gap with self
<i>mode</i>	int	Get the absolute or relative gap • 0 mean absolute • 1 mean relative

Returns

Matrix The absolute gap matrix between self matrix and mat.

See also:

same_size()

size_r()

size_c()

normalize()

Examples

```

>>> m=rand(3)
>>> n=rand(3)
>>> print(m)
| +9.000 | +3.000 | +8.000 |
| +8.000 | +3.000 | +3.000 |
| +3.000 | +5.000 | +3.000 |
printed
>>> print(n)
| +7.000 | +8.000 | +6.000 |
| +4.000 | +0.000 | +2.000 |

```

(continues on next page)

(continued from previous page)

```

| +3.000 | +7.000 | +6.000 |
printed
>>> print(m.gap(n,0))
| +2.000 | +5.000 | +2.000 |
| +4.000 | +3.000 | +1.000 |
| +0.000 | +2.000 | +3.000 |
printed
>>> print(m.gap(n,1))
| +0.091 | +0.227 | +0.091 |
| +0.182 | +0.136 | +0.045 |
| +0.000 | +0.091 | +0.136 |
printed

```

4.44 cmatrix.Matrix.map

Matrix.map()

Map a function (as parameter) to each element of the self matrix

See also:

size_r()

size_c()

set_ij()

get_ij()

Examples

```

>>> def Coef(x):
...     return x*10
...
>>> m=rand(4)
>>> print(m)
| +14.000 | +9.000 | +5.000 | +2.000 |
| +0.000 | +7.000 | +5.000 | +10.000 |
| +2.000 | +10.000 | +0.000 | +2.000 |
| +10.000 | +6.000 | +8.000 | +3.000 |
printed
>>> m.map(Coef)
>>> print(m)
| +140.000 | +90.000 | +50.000 | +20.000 |
| +0.000 | +70.000 | +50.000 | +100.000 |
| +20.000 | +100.000 | +0.000 | +20.000 |
| +100.000 | +60.000 | +80.000 | +30.000 |
printed

```

4.45 cmatrix.Matrix.normalize

Matrix.normalize()

Normalize the self matrix. (the sum of each element is equal to 1)

Returns

Matrix The normalized matrix

See also:

[`clone\(\)`](#)

[`get\_coef\(\)`](#)

[`map\(\)`](#)

Examples

```
>>> t=gaussian(5,0,0)
>>> Print(t)
| +1.000 | +4.000 | +6.000 | +4.000 | +1.000 |
| +4.000 | +16.000 | +24.000 | +16.000 | +4.000 |
| +6.000 | +24.000 | +36.000 | +24.000 | +6.000 |
| +4.000 | +16.000 | +24.000 | +16.000 | +4.000 |
| +1.000 | +4.000 | +6.000 | +4.000 | +1.000 |
printed
>>> t.normalize()
>>> Print(t)
| +0.004 | +0.016 | +0.023 | +0.016 | +0.004 |
| +0.016 | +0.062 | +0.094 | +0.062 | +0.016 |
| +0.023 | +0.094 | +0.141 | +0.094 | +0.023 |
| +0.016 | +0.062 | +0.094 | +0.062 | +0.016 |
| +0.004 | +0.016 | +0.023 | +0.016 | +0.004 |
printed
>>> sum=0
>>> for i in range(0,5):
...     for j in range(0,5):
...         sum+=t[[i,j]]
...
>>> Print(sum)
1.0
```

4.46 cmatrix.Matrix.UX_B

Matrix.UX_B()

Compute the $UX=B$ system wher U is an upper matrix and B the result matrix. The system is solved by gauss.

Parameters	type	Description
U	Matrix	The upper matrix of the system
B	Matrix	The result matrix of the system

Returns

Matrix 1D Matrix representing the solution of the Upper X system

See also:

`square()`

`size_r()`

`size_c()`

`zeros()`

Examples

```
>>> m=Matrix(3,3)
>>> m[0,0]=1
>>> m[0,1]=3
>>> m[0,2]=9
>>> m[1,1]=2
>>> m[1,2]=4
>>> m[2,2]=1
>>> b=Matrix(3,1)
>>> b[0,0]=1
>>> b[1,0]=2
>>> b[2,0]=-1
>>> print(m)
| +1.000 | +3.000 | +9.000 |
| +0.000 | +2.000 | +4.000 |
| +0.000 | +0.000 | +1.000 |
printed
>>> print(b)
| +1.000 |
| +2.000 |
| -1.000 |
printed
>>> print(m.UX_B(m,b))
| +1.000 |
| +3.000 |
| -1.000 |
printed
```

4.47 cmatrix.Matrix.LX_B

Matrix.LX_B()

Compute the $LX=B$ system wher L is a lower matrix and B the result matrix. The system is solved by gauss.

Parameters	type	Description
<i>L</i>	Matrix	The lower matrix of the system
<i>B</i>	Matrix	The result matrix of the system

Returns

Matrix 1D Matrix representing the solution of the Lower X system

See also:

`square()`

`size_r()`

`size_c()`

`zeros()`

Examples

```
>>> m[0,0]=1
>>> m[1,0]=4
>>> m[1,1]=2
>>> m[2,0]=1
>>> m[2,1]=3
>>> m[2,1]=3
>>> m[2,2]=9
>>> b=Matrix(3,1)
>>> b[0,0]=-1
>>> b[1,0]=3
>>> b[2,0]=1
>>> print(m)
| +1.000 | +0.000 | +0.000 |
| +4.000 | +2.000 | +0.000 |
| +1.000 | +3.000 | +9.000 |
printed
>>> print(b)
| -1.000 |
| +3.000 |
| +1.000 |
printed
>>> print(m.LX_B(m,b))
| -1.000 |
| +3.500 |
| -0.944 |
printed
```

4.48 cmatrix.Matrix.replace_zeros

Matrix.replace_zeros()

Utility function to replace the null values with 1. It is used for LU computing genericity.

Returns

Matrix The non-zeros matrix

Examples

```
>>> m=rand_perm(4)
>>> print(m)
| +0.000 | +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
printed
>>> m.replace_zeros()
>>> print(m)
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
| +1.000 | +1.000 | +1.000 | +1.000 |
printed
```

4.49 cmatrix.Matrix.LU

Matrix.LU()

Realize a LU decomposition from the self Matrix. If the self matrix contain some 0, they are automatically replaced with 1

Returns

tuple Matrix A tuple containing the Lower and Upper matrix computed. To restore the original self matrix, compute $U*L$

See also:

count_zero()

replace_zeros()

square()

size_r()

get_ij()

Examples

```
>>> m=rand(6)
>>> print(m)
| +19.000 | +8.000 | +18.000 | +6.000 | +4.000 | +30.000 |
| +1.000 | +28.000 | +12.000 | +32.000 | +33.000 | +35.000 |
| +0.000 | +7.000 | +16.000 | +35.000 | +10.000 | +28.000 |
| +25.000 | +11.000 | +16.000 | +17.000 | +30.000 | +30.000 |
| +27.000 | +4.000 | +18.000 | +6.000 | +2.000 | +31.000 |
| +6.000 | +34.000 | +16.000 | +11.000 | +9.000 | +22.000 |
printed
>>> L,U=m.LU()
>>> print(L)
```

(continues on next page)

(continued from previous page)

```

| +1.000 | +0.000 | +0.000 | +0.000 | +0.000 | +0.000 |
| +0.053 | +1.000 | +0.000 | +0.000 | +0.000 | +0.000 |
| +0.053 | +0.239 | +1.000 | +0.000 | +0.000 | +0.000 |
| +1.316 | +0.017 | -0.634 | +1.000 | +0.000 | +0.000 |
| +1.421 | -0.267 | -0.373 | +0.623 | +1.000 | +0.000 |
| +0.316 | +1.141 | -0.185 | -0.855 | +0.756 | +1.000 |
printed
>>> print(U)
| +19.000 | +8.000 | +18.000 | +6.000 | +4.000 | +30.000 |
| +0.000 | +27.579 | +11.053 | +31.684 | +32.789 | +33.421 |
| +0.000 | +0.000 | +12.416 | +27.126 | +1.968 | +18.448 |
| +0.000 | +0.000 | +0.000 | +25.764 | +25.421 | +1.652 |
| +0.000 | +0.000 | +0.000 | +0.000 | -10.023 | +3.142 |
| +0.000 | +0.000 | +0.000 | +0.000 | +0.000 | -23.164 |
printed
>>> print(U*L)
| +19.000 | +8.000 | +18.000 | +6.000 | +4.000 | +30.000 |
| +1.000 | +28.000 | +12.000 | +32.000 | +33.000 | +35.000 |
| +1.000 | +7.000 | +16.000 | +35.000 | +10.000 | +28.000 |
| +25.000 | +11.000 | +16.000 | +17.000 | +30.000 | +30.000 |
| +27.000 | +4.000 | +18.000 | +6.000 | +2.000 | +31.000 |
| +6.000 | +34.000 | +16.000 | +11.000 | +9.000 | +22.000 |

```

4.50 cmatrix.Matrix.det

Matrix.det()

Compute the associated determinant of the self matrix. To realize it, I decompose the self Matrix into from LU method and compute the associated determinant independantly. Teproduct of the Lower and Upper determinant give the det of the self matrix.

Returns

double The compute determinant

See also:

LU()

size_r()

get_ij()

Examples

```

>>> m=rand(3)
>>> print(m)
| +3.000 | +9.000 | +3.000 |
| +5.000 | +3.000 | +5.000 |
| +8.000 | +8.000 | +2.000 |
printed

```

(continues on next page)

(continued from previous page)

```
>>> print(m.det())
215.99999999999994
```

4.51 cmatrix.Matrix.orthogonal

Matrix.orthogonal()

Boolean test to verify orthogonality

Returns

bint Test the orthogonality of the self matrix.

See also:

transpose()

size_r()

size_c()

Examples

```
>>> m=rand_perm(3)
>>> print(m)
| +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 |
printed
>>> print(m.orthogonal())
True
```

4.52 cmatrix.Matrix.symmetric

Matrix.symmetric()

Boolean test to verify symmetry

Returns

bint Test the symmetry of the self matrix.

See also:

size_c()

size_r()

get_ij()

Examples

```
>>> m=Matrix(3,3)
>>> m[0,0]=2
>>> m[0,1]=7
>>> m[0,2]=3
>>> m[1,0]=7
>>> m[1,1]=9
>>> m[1,2]=4
>>> m[2,0]=3
>>> m[2,1]=4
>>> m[2,2]=7
>>> print(m)
| +2.000 | +7.000 | +3.000 |
| +7.000 | +9.000 | +4.000 |
| +3.000 | +4.000 | +7.000 |
printed
>>> print(m.symmetric())
True
```

4.53 cmatrix.Matrix.hermitian

Matrix.hermitian()

Boolean test to verify if the matrix is hermitian

Returns

bint Test if the matrix is hermitian.

See also:

size_r()

size_c()

transpose()

set_ij()

get_ij()

Examples

```
>>> m=Matrix(3,3)
>>> m[0,0]=1
>>> m[1,2]=1
>>> m[2,1]=1
>>> print(m)
| +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 |
| +0.000 | +1.000 | +0.000 |
printed
```

(continues on next page)

(continued from previous page)

```
>>> print(m.hermitian())
True
```

4.54 cmatrix.Matrix.reversal

Matrix.reversal()

Boolean test to verify the reversability

Returns

bint Test the reversality of the self matrix.

See also:

[*square\(\)*](#)

[*det\(\)*](#)

Examples

```
>>> m=rand(3)
>>> print(m)
| +0.000 | +0.000 | +8.000 |
| +2.000 | +7.000 | +8.000 |
| +8.000 | +7.000 | +2.000 |
printed
>>> print(m.reversal())
True
```

4.55 cmatrix.Matrix.count_zero

Matrix.count_zero()

Count the number of zeros in the self Matrix

Returns

int Count number of zeros contains in the self matrix.

See also:

[*size_r\(\)*](#)

[*size_c\(\)*](#)

[*get_ij\(\)*](#)

Examples

```
>>> m=rand_perm(4)
>>> print(m)
| +0.000 | +1.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +1.000 | +0.000 |
| +1.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +1.000 |
printed
>>> print(m.count_zero())
12
```

4.56 cmatrix.Matrix.nilpotent

Matrix.nilpotent()

Boolean test to verify the nilpotence.

Returns

bint Test the nilpotence of the self matrix.

See also:

count_zero()

size_r()

size_c()

Examples

```
>>> m=rand(4)
>>> m=m.triangle(1)
>>> print(m)
| +0.000 | +12.000 | +16.000 | +11.000 |
| +0.000 | +0.000 | +13.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +4.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
printed
>>> print(m.nilpotent())
True
```

4.57 cmatrix.Matrix.diagonal

`Matrix.diagonal()`

Test if the matrix have a diagonal profile.

Returns

bint Test if the self matrix is diagonal.

Examples

```
>>> m=rand(4)
>>> for i in range(0,4):
...     for j in range(0,4):
...         if(i!=j):
...             m[i,j]=0
...
>>> print(m)
| +4.000 | +0.000 | +0.000 | +0.000 |
| +0.000 | +12.000 | +0.000 | +0.000 |
| +0.000 | +0.000 | +13.000 | +0.000 |
| +0.000 | +0.000 | +0.000 | +6.000 |
printed
>>> print(m.diagonal())
True
```

4.58 cmatrix.Matrix.triangular

`Matrix.triangular()`

Test if the matrix have a triangulare profile.

Returns

int

Test if the self matrix is triangular.

In case of, returns code are given :

- 0 => no triangular
- 1 => inf triangular
- 2 => sup triangular

Examples

```
>>> m=rand(4)
>>> n=m.triangle(0)
>>> o=m.triangle(1)
>>> print(n)
| +7.000 | +0.000 | +0.000 | +0.000 |
| +15.000 | +10.000 | +0.000 | +0.000 |
| +5.000 | +9.000 | +13.000 | +0.000 |
| +15.000 | +9.000 | +5.000 | +10.000 |
printed
>>> print(o)
| +0.000 | +12.000 | +0.000 | +10.000 |
| +0.000 | +0.000 | +2.000 | +1.000 |
| +0.000 | +0.000 | +0.000 | +9.000 |
| +0.000 | +0.000 | +0.000 | +0.000 |
printed
>>> print(m.triangular())
0
>>> print(n.triangular())
1
>>> print(o.triangular())
2
```

4.59 cmatrix.Matrix.read_from_file

Matrix.read_from_file()

Read and convert a matrix from extern file. The file format must respect the writing convention (See write_in_file function).

Parameters	Type	Description
<i>pathname</i>	string	The file name to open

Returns

bint

- True => file readed
- False => Error occurred

See also:

size_r()

size_c()

ind_2_ij()

set_ij()

Examples

```
>>> # Rebuild the matrix since test file, generated from write_in_file func
>>> m=Matrix(4,4)
>>> m.read_from_file("./test")
True
>>> print(m)
| +16.000 | +10.000 | +6.000 | +2.000 |
| +9.000 | +10.000 | +3.000 | +2.000 |
| +1.000 | +1.000 | +5.000 | +16.000 |
| +11.000 | +15.000 | +13.000 | +6.000 |
printed
```

4.60 cmatrix.Matrix.write_in_file

Matrix.write_in_file()

Write the current object to file (re-openable in any other Matrix object instance)

Parameters	Type	Description
<i>pathname</i>	String	The name of the saved file

Returns

bint

- True => file saved
- False => error occurred

See also:

size_r()

size_c()

get_ij()

pack()

Examples

```
>>> m=rand(4)
>>> print(m)
| +16.000 | +10.000 | +6.000 | +2.000 |
| +9.000 | +10.000 | +3.000 | +2.000 |
| +1.000 | +1.000 | +5.000 | +16.000 |
| +11.000 | +15.000 | +13.000 | +6.000 |
printed
>>> m.write_in_file("test")
True
```

4.61 cmatrix.Matrix.pack

Matrix.pack()

Pack the current matrix to 64bit regular float.

See also:

write_in_file()

read_from_file()

Examples

```
>>> m=rand(4)
>>> m.pack()
```

INDICES AND TABLES

- genindex
- modindex
- search

A

abs() (in module *cmatrix*), 22
 accumulate() (*cmatrix.Matrix* method), 55
 AtoB() (*cmatrix.Matrix* method), 65

C

clone() (*cmatrix.Matrix* method), 38
 column() (*cmatrix.Matrix* method), 49
 complement() (in module *cmatrix*), 23
 complement_at() (in module *cmatrix*), 23
 converter() (*cmatrix.Matrix* method), 43
 convolve() (*cmatrix.Matrix* method), 63
 convolve_signal() (*cmatrix.Matrix* method), 64
 count_seq() (in module *cmatrix*), 28
 count_zero() (*cmatrix.Matrix* method), 75
 create_mutants() (in module *cmatrix*), 28
 cut() (in module *cmatrix*), 29

D

det() (*cmatrix.Matrix* method), 72
 diagonal() (*cmatrix.Matrix* method), 77

F

find_invariant() (in module *cmatrix*), 29
 find_max() (in module *cmatrix*), 30
 find_mul_seq() (in module *cmatrix*), 30
 find_seq() (in module *cmatrix*), 31
 find_seq_in_list() (in module *cmatrix*), 31
 find_sub_matrix() (*cmatrix.Matrix* method), 40

G

gap() (*cmatrix.Matrix* method), 66
 gaussian() (in module *cmatrix*), 18
 get_coef() (*cmatrix.Matrix* method), 54
 get_ij() (*cmatrix.Matrix* method), 37
 get_ind() (*cmatrix.Matrix* method), 37

H

hermitian() (*cmatrix.Matrix* method), 74

I

ij_2_ind() (*cmatrix.Matrix* method), 37

ind_2_ij() (*cmatrix.Matrix* method), 37
 insert_sub_matrix() (*cmatrix.Matrix* method), 39
 isalpha() (in module *cmatrix*), 24

L

Laplacian_mean() (in module *cmatrix*), 17
 line() (*cmatrix.Matrix* method), 49
 LU() (*cmatrix.Matrix* method), 71
 LX_B() (*cmatrix.Matrix* method), 69

M

mand() (*cmatrix.Matrix* method), 44
 map() (*cmatrix.Matrix* method), 67
 mean_filter() (in module *cmatrix*), 19
 mirror() (in module *cmatrix*), 24
 mirror_mat() (*cmatrix.Matrix* method), 51
 mnand() (*cmatrix.Matrix* method), 46
 mnor() (*cmatrix.Matrix* method), 47
 mor() (*cmatrix.Matrix* method), 45
 mul_coef() (*cmatrix.Matrix* method), 54
 mult_compatible() (*cmatrix.Matrix* method), 38
 mxor() (*cmatrix.Matrix* method), 48

N

nilpotent() (*cmatrix.Matrix* method), 76
 normalize() (*cmatrix.Matrix* method), 68
 nth_diagonal() (*cmatrix.Matrix* method), 52

O

op() (*cmatrix.Matrix* method), 41
 orthogonal() (*cmatrix.Matrix* method), 73

P

pack() (*cmatrix.Matrix* method), 80
 Pascal_triangle() (in module *cmatrix*), 24
 permut() (*cmatrix.Matrix* method), 57
 Print() (*cmatrix.Matrix* method), 38

R

rand_perm() (in module *cmatrix*), 20
 read_from_file() (*cmatrix.Matrix* method), 78

`replace_zeros()` (*cmatrix.Matrix method*), 70
`resize()` (*cmatrix.Matrix method*), 54
`reversal()` (*cmatrix.Matrix method*), 75
`reverse()` (*in module cmatrix*), 25

S

`same_size()` (*cmatrix.Matrix method*), 38
`set_ij()` (*cmatrix.Matrix method*), 37
`size_c()` (*cmatrix.Matrix method*), 37
`size_r()` (*cmatrix.Matrix method*), 37
`split_data()` (*in module cmatrix*), 32
`split_number()` (*in module cmatrix*), 32
`sqr()` (*cmatrix.Matrix method*), 43
`square()` (*cmatrix.Matrix method*), 56
`strassen()` (*cmatrix.Matrix method*), 59
`swap()` (*in module cmatrix*), 25
`swap_col()` (*cmatrix.Matrix method*), 50
`swap_line()` (*cmatrix.Matrix method*), 51
`switch()` (*in module cmatrix*), 26
`symmetric()` (*cmatrix.Matrix method*), 73

T

`tensorial_product()` (*cmatrix.Matrix method*), 61
`timeit()` (*in module cmatrix*), 27
`trace()` (*cmatrix.Matrix method*), 62
`transpose()` (*cmatrix.Matrix method*), 57
`tri()` (*in module cmatrix*), 27
`triangle()` (*cmatrix.Matrix method*), 59
`triangular()` (*cmatrix.Matrix method*), 77

U

`unit()` (*in module cmatrix*), 21
`UX_B()` (*cmatrix.Matrix method*), 68

W

`write_in_file()` (*cmatrix.Matrix method*), 79

X

`xtract_sub_matrix()` (*cmatrix.Matrix method*), 38

Z

`zeros()` (*in module cmatrix*), 21