

## CMatrix

Generated by Doxygen 1.8.17



<b>1 CMatrix: A Powerful C++ Matrix Library</b>	<b>1</b>
1.1 Table of Contents	1
1.2 Installation	1
1.3 Exemple of Usage	2
1.4 Hierarchical Structure	2
1.5 Documentation	2
1.6 Libraries Used	2
1.7 See Also	3
1.8 License	3
<b>2 Deprecated List</b>	<b>5</b>
<b>3 Module Index</b>	<b>7</b>
3.1 Modules	7
<b>4 Class Index</b>	<b>9</b>
4.1 Class List	9
<b>5 File Index</b>	<b>11</b>
5.1 File List	11
<b>6 Module Documentation</b>	<b>13</b>
6.1 CMatrix	13
6.1.1 Detailed Description	14
6.1.2 Function Documentation	14
6.1.2.1 <code>__cast()</code> [1/2]	14
6.1.2.2 <code>__cast()</code> [2/2]	14
6.1.2.3 <code>__to_string()</code> [1/2]	15
6.1.2.4 <code>__to_string()</code> [2/2]	15
6.1.2.5 <code>apply()</code> [1/2]	16
6.1.2.6 <code>apply()</code> [2/2]	16
6.1.2.7 <code>cast()</code>	16
6.1.2.8 <code>clear()</code>	17
6.1.2.9 <code>copy()</code>	17
6.1.2.10 <code>fill()</code>	17
6.1.2.11 <code>map()</code> [1/4]	18
6.1.2.12 <code>map()</code> [2/4]	18
6.1.2.13 <code>map()</code> [3/4]	18
6.1.2.14 <code>map()</code> [4/4]	19
6.1.2.15 <code>print()</code>	19
6.1.2.16 <code>to_float()</code>	20
6.1.2.17 <code>to_int()</code>	20
6.1.2.18 <code>to_string()</code>	20
6.1.2.19 <code>to_vector()</code>	21

6.2 CMatrixCheck	22
6.2.1 Detailed Description	23
6.2.2 Function Documentation	23
6.2.2.1 __check_expected_id() [1/2]	23
6.2.2.2 __check_expected_id() [2/2]	23
6.2.2.3 __check_size() [1/2]	24
6.2.2.4 __check_size() [2/2]	24
6.2.2.5 __check_valid_col()	24
6.2.2.6 __check_valid_col_id()	25
6.2.2.7 __check_valid_diag()	25
6.2.2.8 __check_valid_row()	26
6.2.2.9 __check_valid_row_id()	26
6.2.2.10 __check_valid_type()	26
6.2.2.11 all() [1/2]	27
6.2.2.12 all() [2/2]	27
6.2.2.13 any() [1/2]	28
6.2.2.14 any() [2/2]	28
6.2.2.15 is_diag()	29
6.2.2.16 is_empty()	29
6.2.2.17 is_identity()	29
6.2.2.18 is_square()	29
6.2.2.19 is_symetric()	30
6.2.2.20 is_triangular_low()	30
6.2.2.21 is_triangular_up()	30
6.3 CMatrixGetter	31
6.3.1 Detailed Description	31
6.3.2 Function Documentation	31
6.3.2.1 cell() [1/2]	32
6.3.2.2 cell() [2/2]	32
6.3.2.3 cells() [1/3]	33
6.3.2.4 cells() [2/3]	33
6.3.2.5 cells() [3/3]	33
6.3.2.6 columns() [1/3]	34
6.3.2.7 columns() [2/3]	34
6.3.2.8 columns() [3/3]	35
6.3.2.9 columns_vec()	35
6.3.2.10 diag()	36
6.3.2.11 height()	36
6.3.2.12 rows() [1/3]	36
6.3.2.13 rows() [2/3]	37
6.3.2.14 rows() [3/3]	37
6.3.2.15 rows_vec()	38

6.3.2.16 size()	38
6.3.2.17 slice_columns()	39
6.3.2.18 slice_rows()	39
6.3.2.19 transpose()	40
6.3.2.20 width()	40
6.4 CMatrixManipulation	41
6.4.1 Detailed Description	41
6.4.2 Function Documentation	41
6.4.2.1 concatenate()	41
6.4.2.2 find() [1/2]	42
6.4.2.3 find() [2/2]	42
6.4.2.4 find_column() [1/2]	43
6.4.2.5 find_column() [2/2]	43
6.4.2.6 find_row() [1/2]	44
6.4.2.7 find_row() [2/2]	44
6.4.2.8 insert_column()	45
6.4.2.9 insert_row()	45
6.4.2.10 push_col_back()	46
6.4.2.11 push_col_front()	46
6.4.2.12 push_row_back()	46
6.4.2.13 push_row_front()	47
6.4.2.14 remove_column()	47
6.4.2.15 remove_row()	48
6.5 CMatrixMath	49
6.5.1 Detailed Description	49
6.5.2 Function Documentation	49
6.5.2.1 exp()	49
6.5.2.2 log()	50
6.5.2.3 log10()	50
6.5.2.4 log2()	50
6.5.2.5 matmul()	50
6.5.2.6 matpow()	51
6.5.2.7 near() [1/2]	51
6.5.2.8 near() [2/2]	52
6.5.2.9 nearq() [1/2]	52
6.5.2.10 nearq() [2/2]	53
6.6 CMatrixOperator	54
6.6.1 Detailed Description	55
6.6.2 Function Documentation	55
6.6.2.1 __map_op_arithmetic() [1/2]	55
6.6.2.2 __map_op_arithmetic() [2/2]	56
6.6.2.3 __map_op_comparaison_val()	56

6.6.2.4 operator!=( ) [1/2]	57
6.6.2.5 operator!=( ) [2/2]	57
6.6.2.6 operator*( ) [1/2]	58
6.6.2.7 operator*( ) [2/2]	58
6.6.2.8 operator*=( ) [1/2]	59
6.6.2.9 operator*=( ) [2/2]	59
6.6.2.10 operator+( ) [1/2]	60
6.6.2.11 operator+( ) [2/2]	60
6.6.2.12 operator+=( ) [1/2]	60
6.6.2.13 operator+=( ) [2/2]	61
6.6.2.14 operator-( ) [1/2]	61
6.6.2.15 operator-( ) [2/2]	62
6.6.2.16 operator-=( ) [1/2]	62
6.6.2.17 operator-=( ) [2/2]	63
6.6.2.18 operator/( )	63
6.6.2.19 operator/=( )	64
6.6.2.20 operator<( )	64
6.6.2.21 operator<=( )	64
6.6.2.22 operator=( ) [1/2]	65
6.6.2.23 operator=( ) [2/2]	65
6.6.2.24 operator==( ) [1/2]	66
6.6.2.25 operator==( ) [2/2]	66
6.6.2.26 operator>( )	67
6.6.2.27 operator>=( )	67
6.6.2.28 operator^( )	67
6.6.2.29 operator^( )	68
6.6.3 Friends	68
6.6.3.1 operator*	68
6.6.3.2 operator+	69
6.6.3.3 operator- [1/2]	69
6.6.3.4 operator- [2/2]	70
6.6.3.5 operator<<	70
6.7 CMatrixSetter	71
6.7.1 Detailed Description	71
6.7.2 Function Documentation	71
6.7.2.1 set_cell()	71
6.7.2.2 set_column()	71
6.7.2.3 set_diag()	72
6.7.2.4 set_row()	72
6.8 CMatrixStatic	75
6.8.1 Detailed Description	75
6.8.2 Function Documentation	75

6.8.2.1 identity()	75
6.8.2.2 is_matrix()	75
6.8.2.3 merge()	77
6.8.2.4 randfloat()	77
6.8.2.5 randint()	78
6.8.2.6 zeros()	78
6.9 CMatrixStatistics	79
6.9.1 Detailed Description	79
6.9.2 Function Documentation	79
6.9.2.1 __mean() [1/2]	79
6.9.2.2 __mean() [2/2]	80
6.9.2.3 __std() [1/2]	80
6.9.2.4 __std() [2/2]	81
6.9.2.5 max()	81
6.9.2.6 max_all()	82
6.9.2.7 mean()	82
6.9.2.8 median()	83
6.9.2.9 min()	84
6.9.2.10 min_all()	84
6.9.2.11 std()	85
6.9.2.12 sum()	85
6.9.2.13 sum_all()	86
<b>7 Class Documentation</b>	<b>87</b>
7.1 cmatrix< T > Class Template Reference	87
7.1.1 Detailed Description	94
7.1.2 Constructor & Destructor Documentation	94
7.1.2.1 cmatrix() [1/6]	94
7.1.2.2 cmatrix() [2/6]	94
7.1.2.3 cmatrix() [3/6]	95
7.1.2.4 cmatrix() [4/6]	95
7.1.2.5 cmatrix() [5/6]	96
7.1.2.6 cmatrix() [6/6]	96
7.1.2.7 ~cmatrix()	96
7.1.3 Member Function Documentation	97
7.1.3.1 identity()	97
7.1.3.2 randfloat()	97
7.1.3.3 randint()	97
7.1.3.4 to_float()	97
7.1.3.5 to_int()	97
7.1.3.6 zeros()	98
7.1.4 Member Data Documentation	98

7.1.4.1 matrix	98
<b>8 File Documentation</b>	<b>99</b>
8.1 benchmark.cpp File Reference	99
8.1.1 Function Documentation	99
8.1.1.1 bench()	99
8.1.1.2 BENCHMARK()	99
8.1.1.3 BENCHMARK_MAIN()	100
8.2 include/CMatrix.hpp File Reference	100
8.2.1 Detailed Description	101
8.3 readme.md File Reference	101
8.4 src/CMatrix.hpp File Reference	101
8.4.1 Detailed Description	101
8.5 src/CMatrixCheck.hpp File Reference	102
8.5.1 Detailed Description	102
8.6 src/CMatrixConstructor.hpp File Reference	102
8.6.1 Detailed Description	103
8.7 src/CMatrixGetter.hpp File Reference	103
8.7.1 Detailed Description	104
8.8 src/CMatrixManipulation.hpp File Reference	104
8.8.1 Detailed Description	104
8.9 src/CMatrixMath.hpp File Reference	104
8.9.1 Detailed Description	105
8.10 src/CMatrixOperator.hpp File Reference	105
8.10.1 Detailed Description	106
8.10.2 Function Documentation	106
8.10.2.1 operator*()	106
8.10.2.2 operator+()	106
8.10.2.3 operator-() [1/2]	106
8.10.2.4 operator-() [2/2]	107
8.10.2.5 operator<<()	107
8.11 src/CMatrixSetter.hpp File Reference	107
8.11.1 Detailed Description	107
8.12 src/CMatrixStatic.hpp File Reference	108
8.12.1 Detailed Description	108
8.13 src/CMatrixStatistics.hpp File Reference	108
8.13.1 Detailed Description	109
<b>Index</b>	<b>111</b>



## Chapter 1

# CMatrix: A Powerful C++ Matrix Library

CMatrix is a robust C++ matrix library designed to simplify matrix operations and provide extensive functionalities. This library is tailored for Data Science and Machine Learning projects, offering a versatile toolset for working with matrices.

### 1.1 Table of Contents

1. [Installation](#)
2. [Example of Usage](#)
3. [Hierarchical Structure](#)
4. [Documentation](#)
5. [Libraries Used](#)
6. [See Also](#)
7. [License](#)

### 1.2 Installation

To install the library, follow these steps:

1. Clone the repository using the following command:

```
git clone https://github.com/B-Manitas/CMatrix.git
```

1. Include the `CMatrix.hpp` file in your project.
2. Compile your project with the following flags:

```
-std=c++11 -fopenmp
```

## 1.3 Exemple of Usage

Here's an example of how to use CMatrix:

```
#include "CMatrix.hpp"
int main()
{
    // Create a 2x3 matrix
    cmatrix<int> mat = {{1, 2, 3}, {4, 5, 6}};
    // Create a random 3x2 matrix
    cmatrix<int> rand = cmatrix<int>::randint(3, 2, 0, 10);
    rand.print();
    // Performs a calculation on the matrix
    mat += ((rand * 2) - 1);
    // Print the transpose of the result
    mat.transpose().print();
    return 0;
}
>> "[[18, 9], [5, 22], [20, 13]]"
```

## 1.4 Hierarchical Structure

CMatrix is structured as follows:

Class	Description
include	
<a href="#">CMatrix.hpp</a>	The main template class that can work with any data type except bool.
src	
<a href="#">CMatrix.hpp</a>	General methods of the class.
<a href="#">CMatrixConstructors.hpp</a>	Implementation of class constructors.
<a href="#">CMatrixGetter.hpp</a>	Methods to retrieve information about the matrix and access its elements.
<a href="#">CMatrixSetter.hpp</a>	Methods to set data in the matrix.
<a href="#">CMatrixCheck.hpp</a>	Methods to verify matrix conditions and perform checks before operations to prevent errors.
<a href="#">CMatrixManipulation.hpp</a>	Methods to find elements in the matrix and transform it.
<a href="#">CMatrixOperator.hpp</a>	Implementation of various operators.
<a href="#">CMatrixStatic.hpp</a>	Implementation of static methods of the class.
<a href="#">CMatrixStatistics.hpp</a>	Methods to perform statistical operations on the matrix.
test	
<a href="#">CMatrixTest.hpp</a>	Contains the tests for the class.

## 1.5 Documentation

For detailed information on how to use CMatrix, consult the [documentation](#).

## 1.6 Libraries Used

- [OpenMP](#): An API for parallel programming. (Required for compile CMatrix)
- [GoogleTest](#): A C++ testing framework.
- [GoogleBenchmark](#): A C++ benchmarking framework.
- [Doxygen](#): A documentation generator.

## 1.7 See Also

- `CDataFrame`: A C++ DataFrame library for Data Science and Machine Learning projects.

## 1.8 License

This project is licensed under the MIT License, ensuring its free and open availability to the community.



## Chapter 2

# Deprecated List

Member `cmatrix< T >::columns_vec (const size_t &n) const`

Use `columns` instead.

Member `cmatrix< T >::rows_vec (const size_t &n) const`

Use `rows` instead.



## Chapter 3

# Module Index

### 3.1 Modules

Here is a list of all modules:

CMatrix . . . . .	13
CMatrixCheck . . . . .	22
CMatrixGetter . . . . .	31
CMatrixManipulation . . . . .	41
CMatrixMath . . . . .	49
CMatrixOperator . . . . .	54
CMatrixSetter . . . . .	71
CMatrixStatic . . . . .	75
CMatrixStatistics . . . . .	79





## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">cmatrix&lt; T &gt;</a>	The main template class that can work with any data type except bool . . . . .	<a href="#">87</a>
------------------------------------	--------------------------------------------------------------------------------	--------------------



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">benchmark.cpp</a> . . . . .	99
<a href="#">include/CMatrix.hpp</a>	
File containing the main template class of the 'cmatrix' library . . . . .	100
<a href="#">src/CMatrix.hpp</a>	
This file contains the implementation of general methods of the class . . . . .	101
<a href="#">src/CMatrixCheck.hpp</a>	
This file contains the implementation of methods to verify matrix conditions and perform checks before operations to prevent errors . . . . .	102
<a href="#">src/CMatrixConstructor.hpp</a>	
This file contains the implementation of constructors and destructors . . . . .	102
<a href="#">src/CMatrixGetter.hpp</a>	
This file contains the implementation of methods to retrieve information from the matrix and get its elements . . . . .	103
<a href="#">src/CMatrixManipulation.hpp</a>	
This file contains the implementation of methods to find elements and to perform manipulations on the matrix . . . . .	104
<a href="#">src/CMatrixMath.hpp</a>	
This file contains the implementation of mathematical functions . . . . .	104
<a href="#">src/CMatrixOperator.hpp</a>	
This file contains the implementation of operators . . . . .	105
<a href="#">src/CMatrixSetter.hpp</a>	
This file contains the implementation of methods to set values in the matrix . . . . .	107
<a href="#">src/CMatrixStatic.hpp</a>	
This file contains the implementation of static methods of the class . . . . .	108
<a href="#">src/CMatrixStatistics.hpp</a>	
This file contains the implementation of methods to perform statistical operations on the matrix	108



## Chapter 6

# Module Documentation

### 6.1 CMatrix

#### Functions

- `template<class U >`  
`cmatrix< U > cmatrix< T >::__cast (std::true_type) const`  
*Convert the matrix to a matrix of another type.*
- `template<class U >`  
`cmatrix< U > cmatrix< T >::__cast (std::false_type) const`  
*Convert the matrix to a matrix of another type.*
- `cmatrix< std::string > cmatrix< T >::__to_string (std::true_type) const`  
*Convert the matrix to a string matrix.*
- `cmatrix< std::string > cmatrix< T >::__to_string (std::false_type) const`  
*Convert the matrix to a string matrix.*
- `void cmatrix< T >::print () const`  
*Print the matrix in the standard output.*
- `void cmatrix< T >::clear ()`  
*Clear the matrix.*
- `cmatrix< T > cmatrix< T >::copy () const`  
*Copy the matrix.*
- `void cmatrix< T >::apply (const std::function< T(T, size_t, size_t)> &f)`  
*Apply a function to each cell of the matrix.*
- `void cmatrix< T >::apply (const std::function< T(T)> &f)`  
*Apply a function to each cell of the matrix.*
- `cmatrix< T > cmatrix< T >::map (const std::function< T(T, size_t, size_t)> &f) const`  
*Apply a function to each cell of the matrix and return the result.*
- `template<class U >`  
`cmatrix< U > cmatrix< T >::map (const std::function< U(T, size_t, size_t)> &f) const`  
*Apply a function to each cell of the matrix and return the result.*
- `cmatrix< T > cmatrix< T >::map (const std::function< T(T)> &f) const`  
*Apply a function to each cell of the matrix and return the result.*
- `template<class U >`  
`cmatrix< U > cmatrix< T >::map (const std::function< U(T)> &f) const`  
*Apply a function to each cell of the matrix and return the result.*
- `void cmatrix< T >::fill (const T &val)`

*Fill the matrix with a value.*

- `std::vector< std::vector< T > > cmatrix< T >::to_vector () const`

*Convert the matrix to a vector.*

- `template<class U >`

`cmatrix< U > cmatrix< T >::cast () const`

*Convert the matrix to a matrix of another type.*

- `cmatrix< int > cmatrix< T >::to_int () const`

*Convert the matrix to a matrix of integers.*

- `cmatrix< float > cmatrix< T >::to_float () const`

*Convert the matrix to a matrix of floats.*

- `cmatrix< std::string > cmatrix< T >::to_string () const`

*Convert the matrix to a matrix of strings.*

## 6.1.1 Detailed Description

## 6.1.2 Function Documentation

### 6.1.2.1 `__cast()` [1/2]

```
template<class T >
template<class U >
cmatrix< U > cmatrix< T >::__cast (
    std::false_type ) const [private]
```

Convert the matrix to a matrix of another type.

#### Template Parameters

<i>U</i>	The type of the matrix to convert.
----------	------------------------------------

#### Parameters

<i>false_type</i>	The type of the matrix is not convertible.
-------------------	--------------------------------------------

#### Exceptions

<i>std::invalid_argument</i>	The type of the matrix is not convertible.
------------------------------	--------------------------------------------

### 6.1.2.2 `__cast()` [2/2]

```
template<class T >
template<class U >
```

```

cmatrix< U > cmatrix< T >::__cast (
    std::true_type ) const [private]

```

Convert the matrix to a matrix of another type.

#### Template Parameters

<i>U</i>	The type of the matrix to convert.
----------	------------------------------------

#### Parameters

<i>true_type</i>	The type of the matrix is convertible.
------------------	----------------------------------------

#### Returns

*cmatrix* The converted matrix.

#### 6.1.2.3 \_\_to\_string() [1/2]

```

template<class T >
cmatrix< std::string > cmatrix< T >::__to_string (
    std::false_type ) const [private]

```

Convert the matrix to a string matrix.

#### Parameters

<i>false_type</i>	The type of the matrix is not convertible.
-------------------	--------------------------------------------

#### Exceptions

<i>std::invalid_argument</i>	The type of the matrix is not convertible.
------------------------------	--------------------------------------------

#### 6.1.2.4 \_\_to\_string() [2/2]

```

template<class T >
cmatrix< std::string > cmatrix< T >::__to_string (
    std::true_type ) const [private]

```

Convert the matrix to a string matrix.

#### Parameters

<i>true_type</i>	The type of the matrix is convertible.
------------------	----------------------------------------

**Returns**

`cmatrix<std::string>` The converted matrix.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.1.2.5 apply() [1/2]**

```
template<class T >
void cmatrix< T >::apply (
    const std::function< T(T)> & f )
```

Apply a function to each cell of the matrix.

**Parameters**

<code>f</code>	The function to apply. f(T value) -> T
----------------	----------------------------------------

**Note**

PARALLELIZED METHOD with OpenMP.

**6.1.2.6 apply() [2/2]**

```
template<class T >
void cmatrix< T >::apply (
    const std::function< T(T, size_t, size_t)> & f )
```

Apply a function to each cell of the matrix.

**Parameters**

<code>f</code>	The function to apply. f(T value, size_t id_row, size_t id_col) -> T
----------------	----------------------------------------------------------------------

**6.1.2.7 cast()**

```
template<class T >
template<class U >
cmatrix< U > cmatrix< T >::cast
```

Convert the matrix to a matrix of another type.



## Template Parameters

<i>U</i>	The type of the matrix.
----------	-------------------------

## Returns

`cmatrix` The matrix of another type.

## Exceptions

<code>std::invalid_argument</code>	<u>If the type T is not convertible to the type U.</u>
------------------------------------	--------------------------------------------------------

## 6.1.2.8 clear()

```
template<class T >
void cmatrix< T >::clear
```

Clear the matrix.

## 6.1.2.9 copy()

```
template<class T >
cmatrix< T > cmatrix< T >::copy
```

Copy the matrix.

## Returns

`cmatrix<T>` The copied matrix.

## 6.1.2.10 fill()

```
template<class T >
void cmatrix< T >::fill (
    const T & val )
```

Fill the matrix with a value.

## Parameters

<i>val</i>	The value to fill the matrix.
------------	-------------------------------

**6.1.2.11 map() [1/4]**

```
template<class T >
cmatrix< T > cmatrix< T >::map (
    const std::function< T(T)> & f ) const
```

Apply a function to each cell of the matrix and return the result.

**Parameters**

<i>f</i>	The function to apply. f(T value) -> T
----------	----------------------------------------

**Returns**

cmatrix<T> The result of the function.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.1.2.12 map() [2/4]**

```
template<class T >
cmatrix< T > cmatrix< T >::map (
    const std::function< T(T, size_t, size_t)> & f ) const
```

Apply a function to each cell of the matrix and return the result.

**Parameters**

<i>f</i>	The function to apply. f(T value, size_t id_row, size_t id_col) -> T
----------	----------------------------------------------------------------------

**Returns**

cmatrix<T> The result of the function.

**6.1.2.13 map() [3/4]**

```
template<class T >
template<class U >
cmatrix< U > cmatrix< T >::map (
    const std::function< U(T)> & f ) const
```

Apply a function to each cell of the matrix and return the result.

## Template Parameters

<i>U</i>	The type of the matrix.
----------	-------------------------

## Parameters

<i>f</i>	The function to apply. $f(T \text{ value}) \rightarrow U$
----------	-----------------------------------------------------------

## Returns

`cmatrix` The result of the function.

## Note

PARALLELIZED METHOD with OpenMP.

6.1.2.14 `map()` [4/4]

```
template<class T >
template<class U >
cmatrix< U > cmatrix< T >::map (
    const std::function< U(T, size_t, size_t)> & f ) const
```

Apply a function to each cell of the matrix and return the result.

## Template Parameters

<i>U</i>	The type of the matrix.
----------	-------------------------

## Parameters

<i>f</i>	The function to apply. $f(T \text{ value}, \text{size\_t id\_row}, \text{size\_t id\_col}) \rightarrow U$
----------	-----------------------------------------------------------------------------------------------------------

## Returns

`cmatrix` The result of the function.

6.1.2.15 `print()`

```
template<class T >
void cmatrix< T >::print
```

Print the matrix in the standard output.

### 6.1.2.16 to\_float()

```
template<class T >
cmatrix< float > cmatrix< T >::to_float
```

Convert the matrix to a matrix of floats.

#### Returns

cmatrix<float> The matrix of floats.

#### Exceptions

<i>std::invalid_argument</i>	If the type T is not convertible to the type float.
<i>std::runtime_error</i>	If the value is out of range of the type float.

#### Note

PARALLELIZED METHOD with OpenMP.

### 6.1.2.17 to\_int()

```
template<class T >
cmatrix< int > cmatrix< T >::to_int
```

Convert the matrix to a matrix of integers.

#### Returns

cmatrix<int> The matrix of integers.

#### Exceptions

<i>std::invalid_argument</i>	If the type T is not convertible to the type int.
<i>std::runtime_error</i>	If the value is out of range of the type int.

#### Note

PARALLELIZED METHOD with OpenMP.

### 6.1.2.18 to\_string()

```
template<class T >
cmatrix< std::string > cmatrix< T >::to_string
```

Convert the matrix to a matrix of strings.

**Returns**

`cmatrix<std::string>` The matrix of strings.

**Exceptions**

<code>std::invalid_argument</code>	If the type T is not a primitive type.
------------------------------------	----------------------------------------

**Note**

PARALLELIZED METHOD with OpenMP.

**6.1.2.19 to\_vector()**

```
template<class T >
std::vector< std::vector< T > > cmatrix< T >::to_vector
```

Convert the matrix to a vector.

**Returns**

`std::vector<T>` The vector.

## 6.2 CMatrixCheck

### Functions

- void `cmatrix< T >::__check_size` (const std::tuple< size\_t, size\_t > &size) const  
*Check if dimensions are equals to the dimensions of the matrix.*
- void `cmatrix< T >::__check_size` (const `cmatrix< T >` &m) const  
*Check if dimensions are equals to the dimensions of the matrix.*
- void `cmatrix< T >::__check_valid_row` (const std::vector< T > &row) const  
*Check if the vector is a valid row of the matrix.*
- void `cmatrix< T >::__check_valid_col` (const std::vector< T > &col) const  
*Check if the vector is a valid column of the matrix.*
- void `cmatrix< T >::__check_valid_diag` (const std::vector< T > &diag) const  
*Check if the diagonal is a valid diagonal of the matrix.*
- void `cmatrix< T >::__check_valid_row_id` (const size\_t &n) const  
*Check if the row is a valid row index of the matrix.*
- void `cmatrix< T >::__check_valid_col_id` (const size\_t &n) const  
*Check if the column is a valid column index of the matrix.*
- void `cmatrix< T >::__check_expected_id` (const size\_t &n, const size\_t &expected) const  
*Check if the index is expected.*
- void `cmatrix< T >::__check_expected_id` (const size\_t &n, const size\_t &expectedBegin, const size\_t &expectedEnd) const  
*Check if the index is expected.*
- void `cmatrix< T >::__check_valid_type` () const  
*Check if the type of the matrix is valid. List of types not supported: bool.*
- bool `cmatrix< T >::is_empty` () const  
*Check if the matrix is empty.*
- bool `cmatrix< T >::is_square` () const  
*Check if the matrix is a square matrix.*
- bool `cmatrix< T >::is_diag` () const  
*Check if the matrix is a diagonal matrix.*
- bool `cmatrix< T >::is_identity` () const  
*Check if the matrix is the identity matrix.*
- bool `cmatrix< T >::is_symetric` () const  
*Check if the matrix is a symmetric matrix.*
- bool `cmatrix< T >::is_triangular_up` () const  
*Check if the matrix is an upper triangular matrix.*
- bool `cmatrix< T >::is_triangular_low` () const  
*Check if the matrix is a lower triangular matrix.*
- bool `cmatrix< T >::all` (const std::function< bool(T)> &f) const  
*Check if all the cells of the matrix satisfy a condition.*
- bool `cmatrix< T >::all` (const T &val) const  
*Check if all the cells of the matrix are equal to a value.*
- bool `cmatrix< T >::any` (const std::function< bool(T)> &f) const  
*Check if at least one cell of the matrix satisfy a condition.*
- bool `cmatrix< T >::any` (const T &val) const  
*Check if at least one cell of the matrix is equal to a value.*

## 6.2.1 Detailed Description

## 6.2.2 Function Documentation

### 6.2.2.1 `__check_expected_id()` [1/2]

```
template<class T >
void cmatrix< T >::__check_expected_id (
    const size_t & n,
    const size_t & expected ) const [private]
```

Check if the index is expected.

#### Parameters

<i>n</i>	The index to check.
<i>expected</i>	The expected index.

#### Exceptions

<code>std::invalid_argument</code>	If the index is not the expected index.
------------------------------------	-----------------------------------------

### 6.2.2.2 `__check_expected_id()` [2/2]

```
template<class T >
void cmatrix< T >::__check_expected_id (
    const size_t & n,
    const size_t & expectedBegin,
    const size_t & expectedEnd ) const [private]
```

Check if the index is expected.

#### Parameters

<i>n</i>	The index to check.
<i>expectedBegin</i>	The expected begin index inclusive.
<i>expectedEnd</i>	The expected end index inclusive.

#### Exceptions

<code>std::invalid_argument</code>	If the index is not the expected index.
------------------------------------	-----------------------------------------

**6.2.2.3 \_\_check\_size()** [1/2]

```
template<class T >
void cmatrix< T >::__check_size (
    const cmatrix< T > & m ) const [private]
```

Check if dimensions are equals to the dimensions of the matrix.

**Parameters**

<i>m</i>	The matrix.
----------	-------------

**Exceptions**

<i>std::invalid_argument</i>	If the dimensions are not equals to the dimensions of the matrix.
------------------------------	-------------------------------------------------------------------

**6.2.2.4 \_\_check\_size()** [2/2]

```
template<class T >
void cmatrix< T >::__check_size (
    const std::tuple< size_t, size_t > & size ) const [private]
```

Check if dimensions are equals to the dimensions of the matrix.

**Parameters**

<i>size</i>	The vertical and horizontal dimensions.
-------------	-----------------------------------------

**Exceptions**

<i>std::invalid_argument</i>	If the dimensions are not equals to the dimensions of the matrix.
------------------------------	-------------------------------------------------------------------

**6.2.2.5 \_\_check\_valid\_col()**

```
template<class T >
void cmatrix< T >::__check_valid_col (
    const std::vector< T > & col ) const [private]
```

Check if the vector is a valid column of the matrix.

**Parameters**

<i>col</i>	The column to check.
------------	----------------------



## Exceptions

<code>std::invalid_argument</code>	If the vector is not a valid column of the matrix.
------------------------------------	----------------------------------------------------

## Note

The column must be a vector of the same type of the matrix.

6.2.2.6 `__check_valid_col_id()`

```
template<class T >
void cmatrix< T >::__check_valid_col_id (
    const size_t & n ) const [private]
```

Check if the column is a valid column index of the matrix.

## Parameters

<code>col</code>	The column index to check.
------------------	----------------------------

## Exceptions

<code>std::invalid_argument</code>	If the column is not a valid column index of the matrix.
------------------------------------	----------------------------------------------------------

6.2.2.7 `__check_valid_diag()`

```
template<class T >
void cmatrix< T >::__check_valid_diag (
    const std::vector< T > & diag ) const [private]
```

Check if the diagonal is a valid diagonal of the matrix.

## Parameters

<code>diag</code>	The diagonal to check.
-------------------	------------------------

## Exceptions

<code>std::invalid_argument</code>	If the vector is not a valid diagonal of the matrix.
------------------------------------	------------------------------------------------------

### 6.2.2.8 `__check_valid_row()`

```
template<class T >
void cmatrix< T >::__check_valid_row (
    const std::vector< T > & row ) const [private]
```

Check if the vector is a valid row of the matrix.

#### Parameters

<i>row</i>	The row to check.
------------	-------------------

#### Exceptions

<i>std::invalid_argument</i>	If the vector is not a valid row of the matrix.
------------------------------	-------------------------------------------------

#### Note

The row must be a vector of the same type of the matrix.

### 6.2.2.9 `__check_valid_row_id()`

```
template<class T >
void cmatrix< T >::__check_valid_row_id (
    const size_t & n ) const [private]
```

Check if the row is a valid row index of the matrix.

#### Parameters

<i>row</i>	The row index to check.
------------	-------------------------

#### Exceptions

<i>std::invalid_argument</i>	If the row is not a valid row index of the matrix.
------------------------------	----------------------------------------------------

### 6.2.2.10 `__check_valid_type()`

```
template<class T >
void cmatrix< T >::__check_valid_type [private]
```

Check if the type of the matrix is valid. List of types not supported: bool.

## Exceptions

<code>std::invalid_argument</code>	If the type is invalid.
------------------------------------	-------------------------

**6.2.2.11 all()** [1/2]

```
template<class T >
bool cmatrix< T >::all (
    const std::function< bool(T)> & f ) const
```

Check if all the cells of the matrix satisfy a condition.

## Parameters

<code>f</code>	The condition to satisfy. <code>f(T value) -&gt; bool</code>
----------------	--------------------------------------------------------------

## Returns

true If all the cells satisfy the condition.  
false If at least one cell does not satisfy the condition.

## Note

The empty matrix always return true.

**6.2.2.12 all()** [2/2]

```
template<class T >
bool cmatrix< T >::all (
    const T & val ) const
```

Check if all the cells of the matrix are equal to a value.

## Parameters

<code>val</code>	The value to check.
------------------	---------------------

## Returns

true If all the cells are equal to the value.  
false If at least one cell is not equal to the value.

**Note**

The empty matrix always return true.

**6.2.2.13 any() [1/2]**

```
template<class T >
bool cmatrix< T >::any (
    const std::function< bool(T)> & f ) const
```

Check if at least one cell of the matrix satisfy a condition.

**Parameters**

<i>f</i>	The condition to satisfy. f(T value) -> bool
----------	----------------------------------------------

**Returns**

true If at least one cell satisfy the condition.

false If all the cells does not satisfy the condition.

**Note**

The empty matrix always return false.

**6.2.2.14 any() [2/2]**

```
template<class T >
bool cmatrix< T >::any (
    const T & val ) const
```

Check if at least one cell of the matrix is equal to a value.

**Parameters**

<i>val</i>	The value to check.
------------	---------------------

**Returns**

true If at least one cell is equal to the value.

false If all the cells are not equal to the value.

**Note**

The empty matrix always return false.

### 6.2.2.15 is\_diag()

```
template<class T >
bool cmatrix< T >::is_diag
```

Check if the matrix is a diagonal matrix.

#### Returns

true If the matrix is a diagonal matrix.  
false If the matrix is not a diagonal matrix.

### 6.2.2.16 is\_empty()

```
template<class T >
bool cmatrix< T >::is_empty
```

Check if the matrix is empty.

#### Returns

true If the matrix is empty.  
false If the matrix is not empty.

### 6.2.2.17 is\_identity()

```
template<class T >
bool cmatrix< T >::is_identity
```

Check if the matrix is the identity matrix.

#### Returns

true If the matrix is the identity matrix.  
false If the matrix is not the identity matrix.

### 6.2.2.18 is\_square()

```
template<class T >
bool cmatrix< T >::is_square
```

Check if the matrix is a square matrix.

#### Returns

true If the matrix is a square matrix.  
false If the matrix is not a square matrix.

#### 6.2.2.19 is\_symetric()

```
template<class T >
bool cmatrix< T >::is_symetric
```

Check if the matrix is a symmetric matrix.

##### Returns

true If the matrix is a symmetric matrix.  
false If the matrix is not a symmetric matrix.

#### 6.2.2.20 is\_triangular\_low()

```
template<class T >
bool cmatrix< T >::is_triangular_low
```

Check if the matrix is a lower triangular matrix.

##### Returns

true If the matrix is a lower triangular matrix.  
false If the matrix is not a lower triangular matrix.

#### 6.2.2.21 is\_triangular\_up()

```
template<class T >
bool cmatrix< T >::is_triangular_up
```

Check if the matrix is an upper triangular matrix.

##### Returns

true If the matrix is an upper triangular matrix.  
false If the matrix is not an upper triangular matrix.

## 6.3 CMatrixGetter

### Functions

- `std::vector< T > cmatrix< T >::rows_vec (const size_t &n) const`  
*Get a row of the matrix.*
- `std::vector< T > cmatrix< T >::columns_vec (const size_t &n) const`  
*Get a column of the matrix as a flattened vector.*
- `cmatrix< T > cmatrix< T >::rows (const size_t &ids) const`  
*Get the rows of the matrix.*
- `cmatrix< T > cmatrix< T >::rows (const std::initializer_list< size_t > &ids) const`  
*Get the rows of the matrix.*
- `cmatrix< T > cmatrix< T >::rows (const std::vector< size_t > &ids) const`  
*Get the rows of the matrix.*
- `cmatrix< T > cmatrix< T >::columns (const size_t &ids) const`  
*Get the columns of the matrix.*
- `cmatrix< T > cmatrix< T >::columns (const std::initializer_list< size_t > &ids) const`  
*Get the columns of the matrix.*
- `cmatrix< T > cmatrix< T >::columns (const std::vector< size_t > &ids) const`  
*Get the columns of the matrix.*
- `cmatrix< T > cmatrix< T >::cells (const size_t &row, const size_t &col) const`  
*Get the cells of the matrix.*
- `cmatrix< T > cmatrix< T >::cells (const std::initializer_list< std::pair< size_t, size_t >> &ids) const`  
*Get the cells of the matrix.*
- `cmatrix< T > cmatrix< T >::cells (const std::vector< std::pair< size_t, size_t >> &ids) const`  
*Get the cells of the matrix.*
- `T &cmatrix< T >::cell (const size_t &row, const size_t &col)`  
*Get the reference to a cell of the matrix.*
- `T cmatrix< T >::cell (const size_t &row, const size_t &col) const`  
*Get a cell of the matrix.*
- `cmatrix< T > cmatrix< T >::slice_rows (const size_t &start, const size_t &end) const`  
*Get the rows between two indexes.*
- `cmatrix< T > cmatrix< T >::slice_columns (const size_t &start, const size_t &end) const`  
*Get the columns between two indexes.*
- `size_t cmatrix< T >::width () const`  
*The number of columns of the matrix.*
- `size_t cmatrix< T >::height () const`  
*The number of rows of the matrix.*
- `std::pair< size_t, size_t > cmatrix< T >::size () const`  
*The dimensions of the matrix.*
- `cmatrix< T > cmatrix< T >::transpose () const`  
*Get the transpose of the matrix.*
- `std::vector< T > cmatrix< T >::diag () const`  
*Get the diagonal of the matrix.*

#### 6.3.1 Detailed Description

#### 6.3.2 Function Documentation

**6.3.2.1 cell() [1/2]**

```
template<class T >
T & cmatrix< T >::cell (
    const size_t & row,
    const size_t & col )
```

Get the reference to a cell of the matrix.

**Parameters**

<i>row</i>	The row of the cell to get.
<i>col</i>	The column of the cell to get.

**Returns**

T The reference of the cell.

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
--------------------------	-------------------------------

**6.3.2.2 cell() [2/2]**

```
template<class T >
T cmatrix< T >::cell (
    const size_t & row,
    const size_t & col ) const
```

Get a cell of the matrix.

**Parameters**

<i>row</i>	The row of the cell to get.
<i>col</i>	The column of the cell to get.

**Returns**

T The cell.

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
--------------------------	-------------------------------



**6.3.2.3 cells() [1/3]**

```
template<class T >
cmatrix< T > cmatrix< T >::cells (
    const size_t & row,
    const size_t & col ) const
```

Get the cells of the matrix.

**Parameters**

<i>row</i>	The row of the cell to get.
<i>col</i>	The column of the cell to get.

**Returns**

cmatrix<T> The cells of the matrix.

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
--------------------------	-------------------------------

**6.3.2.4 cells() [2/3]**

```
template<class T >
cmatrix< T > cmatrix< T >::cells (
    const std::initializer_list< std::pair< size_t, size_t >> & ids ) const
```

Get the cells of the matrix.

**Parameters**

<i>ids</i>	The indexes of the cells to get. (row, column)
------------	------------------------------------------------

**Returns**

cmatrix<T> The cells of the matrix.

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
--------------------------	-------------------------------

**6.3.2.5 cells() [3/3]**

```
template<class T >
```

```
cmatrix< T > cmatrix< T >::cells (
    const std::vector< std::pair< size_t, size_t >> & ids ) const
```

Get the cells of the matrix.

#### Parameters

<i>ids</i>	The indexes of the cells to get. (row, column)
------------	------------------------------------------------

#### Returns

cmatrix<T> The cells of the matrix.

#### Exceptions

<i>std::out_of_range</i>	If the index is out of range.
--------------------------	-------------------------------

### 6.3.2.6 columns() [1/3]

```
template<class T >
cmatrix< T > cmatrix< T >::columns (
    const size_t & ids ) const
```

Get the columns of the matrix.

#### Parameters

<i>ids</i>	The indexes of the columns to get.
------------	------------------------------------

#### Returns

cmatrix<T> The columns of the matrix.

#### Exceptions

<i>std::out_of_range</i>	If the index is out of range.
--------------------------	-------------------------------

### 6.3.2.7 columns() [2/3]

```
template<class T >
cmatrix< T > cmatrix< T >::columns (
    const std::initializer_list< size_t > & ids ) const
```

Get the columns of the matrix.

## Parameters

<i>ids</i>	The indexes of the columns to get.
------------	------------------------------------

## Returns

`cmatrix<T>` The columns of the matrix.

## Exceptions

<code>std::out_of_range</code>	If the index is out of range.
--------------------------------	-------------------------------

6.3.2.8 `columns()` [3/3]

```
template<class T >
cmatrix< T > cmatrix< T >::columns (
    const std::vector< size_t > & ids ) const
```

Get the columns of the matrix.

## Parameters

<i>ids</i>	The indexes of the columns to get.
------------	------------------------------------

## Returns

`cmatrix<T>` The columns of the matrix.

## Exceptions

<code>std::out_of_range</code>	If the index is out of range.
--------------------------------	-------------------------------

6.3.2.9 `columns_vec()`

```
template<class T >
std::vector< T > cmatrix< T >::columns_vec (
    const size_t & n ) const
```

Get a column of the matrix as a flattened vector.

## Parameters

<i>n</i>	The index of the column to get.
----------	---------------------------------

**Returns**

`std::vector<T>` The column as a flattened vector.

**Exceptions**

<code>std::out_of_range</code>	If the index is out of range.
--------------------------------	-------------------------------

**Deprecated** Use `columns` instead.

**6.3.2.10 diag()**

```
template<class T >
std::vector< T > cmatrix< T >::diag
```

Get the diagonal of the matrix.

**Returns**

`std::vector<T>` The diagonal of the matrix.

**6.3.2.11 height()**

```
template<class T >
size_t cmatrix< T >::height
```

The number of rows of the matrix.

**Returns**

`size_t` The number of rows.

**6.3.2.12 rows() [1/3]**

```
template<class T >
cmatrix< T > cmatrix< T >::rows (
    const size_t & ids ) const
```

Get the rows of the matrix.

## Parameters

<i>ids</i>	The indexes of the rows to get.
------------	---------------------------------

## Returns

`cmatrix<T>` The rows of the matrix.

## Exceptions

<code>std::out_of_range</code>	If the index is out of range.
--------------------------------	-------------------------------

**6.3.2.13 rows()** [2/3]

```
template<class T >
cmatrix< T > cmatrix< T >::rows (
    const std::initializer_list< size_t > & ids ) const
```

Get the rows of the matrix.

## Parameters

<i>ids</i>	The indexes of the rows to get.
------------	---------------------------------

## Returns

`cmatrix<T>` The rows of the matrix.

## Exceptions

<code>std::out_of_range</code>	If the index is out of range.
--------------------------------	-------------------------------

**6.3.2.14 rows()** [3/3]

```
template<class T >
cmatrix< T > cmatrix< T >::rows (
    const std::vector< size_t > & ids ) const
```

Get the rows of the matrix.

## Parameters

<i>ids</i>	The indexes of the rows to get.
------------	---------------------------------

**Returns**

`cmatrix<T>` The rows of the matrix.

**Exceptions**

<code>std::out_of_range</code>	If the index is out of range.
--------------------------------	-------------------------------

**6.3.2.15 rows\_vec()**

```
template<class T >
std::vector< T > cmatrix< T >::rows_vec (
    const size_t & n ) const
```

Get a row of the matrix.

**Parameters**

<code>n</code>	The index of the row to get.
----------------	------------------------------

**Returns**

`std::vector<T>` The row.

**Exceptions**

<code>std::out_of_range</code>	If the index is out of range.
--------------------------------	-------------------------------

**Deprecated** Use `rows` instead.

**6.3.2.16 size()**

```
template<class T >
std::pair< size_t, size_t > cmatrix< T >::size
```

The dimensions of the matrix.

**Returns**

`std::pair<size_t, size_t>` The number of rows and columns.

**6.3.2.17 slice\_columns()**

```
template<class T >
cmatrix< T > cmatrix< T >::slice_columns (
    const size_t & start,
    const size_t & end ) const
```

Get the columns between two indexes.

**Parameters**

<i>start</i>	The start index inclusive.
<i>end</i>	The end index inclusive.

**Returns**

cmatrix<T> The columns between two indexes.

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
<i>std::invalid_argument</i>	If the start index is greater than the end index.

**6.3.2.18 slice\_rows()**

```
template<class T >
cmatrix< T > cmatrix< T >::slice_rows (
    const size_t & start,
    const size_t & end ) const
```

Get the rows between two indexes.

**Parameters**

<i>start</i>	The start index inclusive.
<i>end</i>	The end index inclusive.

**Returns**

cmatrix<T> The rows between two indexes.

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
<i>std::invalid_argument</i>	If the start index is greater than the end index.

### 6.3.2.19 transpose()

```
template<class T >  
cmatrix< T > cmatrix< T >::transpose
```

Get the transpose of the matrix.

#### Returns

cmatrix<T> The transpose of the matrix.

#### Note

PARALLELIZED METHOD with OpenMP.

### 6.3.2.20 width()

```
template<class T >  
size_t cmatrix< T >::width
```

The number of columns of the matrix.

#### Returns

size\_t The number of columns.



## 6.4 CMatrixManipulation

### Functions

- void `cmatrix< T >::insert_row` (const size\_t &pos, const std::vector< T > &val)  
*Insert a column in the matrix.*
- void `cmatrix< T >::insert_column` (const size\_t &pos, const std::vector< T > &val)  
*Insert a row in the matrix.*
- void `cmatrix< T >::push_row_front` (const std::vector< T > &val)  
*Push a row in the front of the matrix.*
- void `cmatrix< T >::push_row_back` (const std::vector< T > &val)  
*Push a row in the back of the matrix.*
- void `cmatrix< T >::push_col_front` (const std::vector< T > &val)  
*Push a column in the front of the matrix.*
- void `cmatrix< T >::push_col_back` (const std::vector< T > &val)  
*Push a column in the back of the matrix.*
- int `cmatrix< T >::find_row` (const std::function< bool(std::vector< T >)> &f) const
- int `cmatrix< T >::find_row` (const std::vector< T > &val) const  
*Find the first row matching the given row.*
- int `cmatrix< T >::find_column` (const std::function< bool(std::vector< T >)> &f) const  
*Find the first column matching the condition.*
- int `cmatrix< T >::find_column` (const std::vector< T > &val) const  
*Find the first column matching the given column.*
- std::tuple< int, int > `cmatrix< T >::find` (const std::function< bool(T)> &f) const  
*Find the first cell matching the condition.*
- std::tuple< int, int > `cmatrix< T >::find` (const T &val) const  
*Find the first cell matching the given cell.*
- void `cmatrix< T >::remove_row` (const size\_t &n)  
*Remove a row of the matrix.*
- void `cmatrix< T >::remove_column` (const size\_t &n)  
*Remove a column of the matrix.*
- void `cmatrix< T >::concatenate` (const `cmatrix< T >` &m, const unsigned int &axis=0)  
*Concatenate a matrix to the matrix.*

### 6.4.1 Detailed Description

### 6.4.2 Function Documentation

#### 6.4.2.1 concatenate()

```
template<class T >
void cmatrix< T >::concatenate (
    const cmatrix< T > & m,
    const unsigned int & axis = 0 )
```

Concatenate a matrix to the matrix.

**Parameters**

<i>m</i>	The matrix to concatenate.
<i>axis</i>	The axis to concatenate. 0 for the rows, 1 for the columns. (default: 0)

**Exceptions**

<i>std::invalid_argument</i>	If the axis is not 0 or 1.
<i>std::invalid_argument</i>	If the dimensions of matrices are not equals.

**6.4.2.2 find() [1/2]**

```
template<class T >
std::tuple< int, int > cmatrix< T >::find (
    const std::function< bool(T)> & f ) const
```

Find the first cell matching the condition.

**Parameters**

<i>f</i>	The condition to satisfy. f(T value) -> bool
----------	----------------------------------------------

**Returns**

std::tuple<int, int> The first index of the cell. (-1, -1) if not found.

**Note**

The empty matrix always return (-1, -1).

**6.4.2.3 find() [2/2]**

```
template<class T >
std::tuple< int, int > cmatrix< T >::find (
    const T & val ) const
```

Find the first cell matching the given cell.

**Parameters**

<i>val</i>	The cell to find.
------------	-------------------

**Returns**

`std::tuple<int, int>` The first index of the cell. (-1, -1) if not found.

**Note**

The cell must be of the same type of the matrix.

**6.4.2.4 find\_column() [1/2]**

```
template<class T >
int cmatrix< T >::find_column (
    const std::function< bool(std::vector< T >)> & f ) const
```

Find the first column matching the condition.

**Parameters**

<i>f</i>	The condition to satisfy. <code>f(std::vector&lt;T&gt; col) -&gt; bool</code>
----------	-------------------------------------------------------------------------------

**Returns**

`int` The first index of the column. -1 if not found.

**Note**

The empty matrix always return -1.

**6.4.2.5 find\_column() [2/2]**

```
template<class T >
int cmatrix< T >::find_column (
    const std::vector< T > & val ) const
```

Find the first column matching the given column.

**Parameters**

<i>val</i>	The column to find.
------------	---------------------

**Returns**

`int` The first index of the row. -1 if not found.

**Note**

The column must be a vector of the same type of the matrix.

**6.4.2.6 find\_row()** [1/2]

```
template<class T >
int cmatrix< T >::find_row (
    const std::function< bool(std::vector< T >)> & f ) const
```

@bried Find the first row matching the condition.

**Parameters**

<i>f</i>	The condition to satisfy. <code>f(std::vector&lt;T&gt; row) -&gt; bool</code>
----------	-------------------------------------------------------------------------------

**Returns**

int The first index of the row. -1 if not found.

**Note**

The empty matrix always return -1.

**6.4.2.7 find\_row()** [2/2]

```
template<class T >
int cmatrix< T >::find_row (
    const std::vector< T > & val ) const
```

Find the first row matching the given row.

**Parameters**

<i>val</i>	The row to find.
------------	------------------

**Returns**

int The first index of the row. -1 if not found.

**Note**

The row must be a vector of the same type of the matrix.

## 6.4.2.8 insert\_column()

```
template<class T >
void cmatrix< T >::insert_column (
    const size_t & pos,
    const std::vector< T > & val )
```

Insert a row in the matrix.

## Parameters

<i>pos</i>	The index of the row to insert.
<i>val</i>	The value to insert.

## Exceptions

<i>std::out_of_range</i>	If the index is out of range.
<i>std::invalid_argument</i>	If the size of the vector <i>val</i> is not equal to the number of columns of the matrix.

## Note

The row must be a vector of the same type of the matrix.  
PARALLELIZED METHOD with OpenMP.

## 6.4.2.9 insert\_row()

```
template<class T >
void cmatrix< T >::insert_row (
    const size_t & pos,
    const std::vector< T > & val )
```

Insert a column in the matrix.

## Parameters

<i>pos</i>	The index of the column to insert.
<i>val</i>	The value to insert.

## Exceptions

<i>std::out_of_range</i>	If the index is out of range.
<i>std::invalid_argument</i>	If the size of the vector <i>val</i> is not equal to the number of rows of the matrix.

## Note

The column must be a vector of the same type of the matrix.

#### 6.4.2.10 `push_col_back()`

```
template<class T >
void cmatrix< T >::push_col_back (
    const std::vector< T > & val )
```

Push a column in the back of the matrix.

##### Parameters

<i>val</i>	The column to push.
------------	---------------------

##### Exceptions

<code>std::invalid_argument</code>	If the size of the vector <i>val</i> is not equal to the number of rows of the matrix.
------------------------------------	----------------------------------------------------------------------------------------

##### Note

The column must be a vector of the same type of the matrix.

#### 6.4.2.11 `push_col_front()`

```
template<class T >
void cmatrix< T >::push_col_front (
    const std::vector< T > & val )
```

Push a column in the front of the matrix.

##### Parameters

<i>val</i>	The column to push.
------------	---------------------

##### Exceptions

<code>std::invalid_argument</code>	If the size of the vector <i>val</i> is not equal to the number of rows of the matrix.
------------------------------------	----------------------------------------------------------------------------------------

##### Note

The column must be a vector of the same type of the matrix.

#### 6.4.2.12 `push_row_back()`

```
template<class T >
void cmatrix< T >::push_row_back (
    const std::vector< T > & val )
```

Push a row in the back of the matrix.

## Parameters

<i>val</i>	The row to push.
------------	------------------

## Exceptions

<i>std::invalid_argument</i>	If the size of the vector <i>val</i> is not equal to the number of columns of the matrix.
------------------------------	-------------------------------------------------------------------------------------------

## Note

The row must be a vector of the same type of the matrix.

**6.4.2.13 push\_row\_front()**

```
template<class T >
void cmatrix< T >::push_row_front (
    const std::vector< T > & val )
```

Push a row in the front of the matrix.

## Parameters

<i>val</i>	The row to push.
------------	------------------

## Exceptions

<i>std::invalid_argument</i>	If the size of the vector <i>val</i> is not equal to the number of columns of the matrix.
------------------------------	-------------------------------------------------------------------------------------------

## Note

The row must be a vector of the same type of the matrix.

**6.4.2.14 remove\_column()**

```
template<class T >
void cmatrix< T >::remove_column (
    const size_t & n )
```

Remove a column of the matrix.

## Parameters

<i>n</i>	The index of the column to remove.
----------	------------------------------------

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
<i>std::invalid_argument</i>	If the matrix is empty.

**6.4.2.15 remove\_row()**

```
template<class T >
void cmatrix< T >::remove_row (
    const size_t & n )
```

Remove a row of the matrix.

**Parameters**

<i>n</i>	The index of the row to remove.
----------	---------------------------------

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
<i>std::invalid_argument</i>	If the matrix is empty.



## 6.5 CMatrixMath

### Functions

- bool `cmatrix< T >::near` (const `cmatrix< T >` &val, const T &tolerance=1e-5) const  
*Test if the matrix is near another matrix.*
- bool `cmatrix< T >::near` (const T &val, const T &tolerance=1e-5) const  
*Test if the matrix is near a value.*
- bool `cmatrix< T >::nearq` (const `cmatrix< T >` &val, const T &tolerance=1e-5) const  
*Test if the matrix is not near another matrix.*
- bool `cmatrix< T >::nearq` (const T &val, const T &tolerance=1e-5) const  
*Test if the matrix is not near a value.*
- `cmatrix< T > cmatrix< T >::matmul` (const `cmatrix< T >` &m) const  
*Get the product with another matrix.*
- `cmatrix< T > cmatrix< T >::matpow` (const unsigned int &n) const  
*Get the power of the matrix.*
- `cmatrix< T > cmatrix< T >::log` () const  
*Get the natural logarithm of the matrix.*
- `cmatrix< T > cmatrix< T >::log2` () const  
*Get the log2 of the matrix.*
- `cmatrix< T > cmatrix< T >::log10` () const  
*Get the log10 of the matrix.*
- `cmatrix< T > cmatrix< T >::exp` () const  
*Get the exponential of the matrix.*

### 6.5.1 Detailed Description

### 6.5.2 Function Documentation

#### 6.5.2.1 exp()

```
template<class T >
cmatrix< T > cmatrix< T >::exp
```

Get the exponential of the matrix.

#### Returns

`cmatrix<T>` The result of the exponential.

#### Note

PARALLELIZED METHOD with OpenMP.

### 6.5.2.2 log()

```
template<class T >
cmatrix< T > cmatrix< T >::log
```

Get the natural logarithm of the matrix.

#### Returns

cmatrix<T> The result of the log.

#### Note

PARALLELIZED METHOD with OpenMP.

### 6.5.2.3 log10()

```
template<class T >
cmatrix< T > cmatrix< T >::log10
```

Get the log10 of the matrix.

#### Returns

cmatrix<T> The result of the log.

#### Note

PARALLELIZED METHOD with OpenMP.

### 6.5.2.4 log2()

```
template<class T >
cmatrix< T > cmatrix< T >::log2
```

Get the log2 of the matrix.

#### Returns

cmatrix<T> The result of the log.

#### Note

PARALLELIZED METHOD with OpenMP.

### 6.5.2.5 matmul()

```
template<class T >
cmatrix< T > cmatrix< T >::matmul (
    const cmatrix< T > & m ) const
```

Get the product with another matrix.

## Parameters

<i>m</i>	The matrix to multiply.
----------	-------------------------

## Returns

`cmatrix<T>` The result of the product.

## Exceptions

<code>std::invalid_argument</code>	If the number of columns of the matrix is not equal to the number of rows of the matrix <i>m</i> .
------------------------------------	----------------------------------------------------------------------------------------------------

## Note

PARALLELIZED METHOD with OpenMP.

6.5.2.6 `matpow()`

```
template<class T >
cmatrix< T > cmatrix< T >::matpow (
    const unsigned int & n ) const
```

Get the power of the matrix.

## Parameters

<i>n</i>	The power.
----------	------------

## Returns

`cmatrix<T>` The result of the power.

## Exceptions

<code>std::invalid_argument</code>	If the matrix is not a square matrix.
------------------------------------	---------------------------------------

## Note

PARALLELIZED METHOD with OpenMP.

6.5.2.7 `near()` [1/2]

```
template<class T >
bool cmatrix< T >::near (
```

```
const cmatrix< T > & val,
const T & tolerance = 1e-5 ) const
```

Test if the matrix is near another matrix.

#### Parameters

<i>val</i>	The matrix to test.
<i>tolerance</i>	The tolerance of the test. (default: 1e-5)

#### Returns

true If the matrix is near the matrix *val*.

false If the matrix is not near the matrix *val*.

### 6.5.2.8 near() [2/2]

```
template<class T >
bool cmatrix< T >::near (
    const T & val,
    const T & tolerance = 1e-5 ) const
```

Test if the matrix is near a value.

#### Parameters

<i>val</i>	The value to test.
<i>tolerance</i>	The tolerance of the test. (default: 1e-5)

#### Returns

true If the matrix is near the value *val*.

false If the matrix is not near the value *val*.

### 6.5.2.9 nearq() [1/2]

```
template<class T >
bool cmatrix< T >::nearq (
    const cmatrix< T > & val,
    const T & tolerance = 1e-5 ) const
```

Test if the matrix is not near another matrix.

#### Parameters

<i>val</i>	The matrix to test.
<i>tolerance</i>	The tolerance of the test. (default: 1e-5)

**Returns**

true If the matrix is not near the matrix `val`.

false If the matrix is near the matrix `val`.

**6.5.2.10 nearq() [2/2]**

```
template<class T >
bool cmatrix< T >::nearq (
    const T & val,
    const T & tolerance = 1e-5 ) const
```

Test if the matrix is not near a value.

**Parameters**

<i>val</i>	The value to test.
<i>tolerance</i>	The tolerance of the test. (default: 1e-5)

**Returns**

true If the matrix is not near the value `val`.

false If the matrix is near the value `val`.

## 6.6 CMatrixOperator

### Functions

- `cmatrix< T > cmatrix< T >::__map_op_arithmetic` (const std::function< T(T, T)> &f, const `cmatrix< T >` &m) const  
*Apply a operator to each cell of the matrix.*
- `cmatrix< T > cmatrix< T >::__map_op_arithmetic` (const std::function< T(T, T)> &f, const T &val) const  
*Apply a operator to each cell of the matrix.*
- `cmatrix< short unsigned int > cmatrix< T >::__map_op_comparaison_val` (const std::function< T(T, T)> &f, const T &n) const  
*Map a comparison operator to each cell of the matrix and return a matrix of boolean.*
- `cmatrix< T > & cmatrix< T >::operator=` (const std::initializer\_list< std::initializer\_list< T >> &m)  
*The assignment operator.*
- `cmatrix< T > & cmatrix< T >::operator=` (const `cmatrix< T >` &m)  
*The assignment operator.*
- `bool cmatrix< T >::operator==` (const `cmatrix< T >` &m) const  
*The equality operator.*
- `bool cmatrix< T >::operator!=` (const `cmatrix< T >` &m) const  
*The inequality operator.*
- `cmatrix< short unsigned int > cmatrix< T >::operator==` (const T &n) const  
*The equality operator comparing the matrix with a value.*
- `cmatrix< short unsigned int > cmatrix< T >::operator!=` (const T &n) const  
*The inequality operator comparing the matrix with a value.*
- `cmatrix< short unsigned int > cmatrix< T >::operator<` (const T &n) const  
*The strictly less than operator comparing the matrix with a value.*
- `cmatrix< short unsigned int > cmatrix< T >::operator<=` (const T &n) const  
*The less than operator comparing the matrix with a value.*
- `cmatrix< short unsigned int > cmatrix< T >::operator>` (const T &n) const  
*The strictly greater than operator comparing the matrix with a value.*
- `cmatrix< short unsigned int > cmatrix< T >::operator>=` (const T &n) const  
*The greater than operator comparing the matrix with a value.*
- `cmatrix< T > cmatrix< T >::operator+` (const `cmatrix< T >` &m) const  
*The addition operator.*
- `cmatrix< T > cmatrix< T >::operator+` (const T &n) const  
*The addition operator.*
- `cmatrix< T > cmatrix< T >::operator-` (const `cmatrix< T >` &m) const  
*The subtraction operator.*
- `cmatrix< T > cmatrix< T >::operator-` (const T &val) const  
*The subtraction operator.*
- `cmatrix< T > cmatrix< T >::operator*` (const `cmatrix< T >` &m) const  
*The multiplication operator element-wise.*
- `cmatrix< T > cmatrix< T >::operator*` (const T &n) const  
*The multiplication operator.*
- `cmatrix< T > cmatrix< T >::operator/` (const T &n) const  
*The division operator.*
- `cmatrix< T > cmatrix< T >::operator^` (const unsigned int &m) const  
*The power operator element-wise.*
- `cmatrix< T > & cmatrix< T >::operator+=` (const `cmatrix< T >` &m)  
*The addition assignment operator.*
- `cmatrix< T > & cmatrix< T >::operator+=` (const T &n)

*The addition assignment operator.*

- `cmatrix< T > & cmatrix< T >::operator+= (const cmatrix< T > &m)`

*The subtraction assignment operator.*

- `cmatrix< T > & cmatrix< T >::operator-= (const T &n)`

*The subtraction assignment operator.*

- `cmatrix< T > & cmatrix< T >::operator*= (const cmatrix< T > &m)`

*The multiplication assignment operator.*

- `cmatrix< T > & cmatrix< T >::operator*= (const T &n)`

*The multiplication assignment operator.*

- `cmatrix< T > & cmatrix< T >::operator/= (const T &n)`

*The division assignment operator.*

- `cmatrix< T > & cmatrix< T >::operator^= (const unsigned int &m)`

*The power assignment operator.*

## Friends

- `template<class U >`  
`std::ostream & cmatrix< T >::operator<< (std::ostream &out, const cmatrix< U > &m)`

*The output operator.*

- `template<class U >`  
`cmatrix< U > cmatrix< T >::operator+ (const U &n, const cmatrix< U > &m)`

*The addition operator.*

- `template<class U >`  
`cmatrix< U > cmatrix< T >::operator- (const U &n, const cmatrix< U > &m)`

*The subtraction operator.*

- `template<class U >`  
`cmatrix< U > cmatrix< T >::operator- (const cmatrix< U > &m)`

*The negation operator.*

- `template<class U >`  
`cmatrix< U > cmatrix< T >::operator* (const U &n, const cmatrix< U > &m)`

*The multiplication operator.*

### 6.6.1 Detailed Description

### 6.6.2 Function Documentation

#### 6.6.2.1 `__map_op_arithmetic()` [1/2]

```
template<class T >
cmatrix< T > cmatrix< T >::__map_op_arithmetic (
    const std::function< T(T, T)> & f,
    const cmatrix< T > & m ) const [private]
```

Apply a operator to each cell of the matrix.

**Parameters**

<i>f</i>	The operator to apply. f(T value, T value) -> T
<i>m</i>	The matrix to apply.

**Returns**

`cmatrix<T>` The result of the operator.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.2.2 \_\_map\_op\_arithmetic() [2/2]**

```
template<class T >
cmatrix< T > cmatrix< T >::__map_op_arithmetic (
    const std::function< T(T, T)> & f,
    const T & val ) const [private]
```

Apply a operator to each cell of the matrix.

**Parameters**

<i>f</i>	The operator to apply. f(T value, T value) -> T
<i>val</i>	The value to apply.

**Returns**

`cmatrix<T>` The result of the operator.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.2.3 \_\_map\_op\_comparaison\_val()**

```
template<class T >
cmatrix< short unsigned int > cmatrix< T >::__map_op_comparaison_val (
    const std::function< T(T, T)> & f,
    const T & n ) const [private]
```

Map a comparison operator to each cell of the matrix and return a matrix of boolean.



## Parameters

<i>f</i>	The comparison operator to apply. f(T value, T value) -> bool
<i>n</i>	The number to compare.

## Returns

cmatrix<short unsigned int> The result of the comparison.

## Note

PARALLELIZED METHOD with OpenMP.

## 6.6.2.4 operator"!=()" [1/2]

```
template<class T >
bool cmatrix< T >::operator!= (
    const cmatrix< T > & m ) const
```

The inequality operator.

## Parameters

<i>m</i>	The matrix to compare.
----------	------------------------

## Returns

true If the matrices are not equal.

false If the matrices are equal.

## Note

The matrix must be of the same type of the matrix.

## 6.6.2.5 operator"!=()" [2/2]

```
template<class T >
cmatrix< short unsigned int > cmatrix< T >::operator!= (
    const T & n ) const
```

The inequality operator comparing the matrix with a value.

**Parameters**

<i>val</i>	The value to compare.
------------	-----------------------

**Returns**

`cmatrix<short unsigned int>` The matrix of booleans.

**6.6.2.6 operator\*() [1/2]**

```
template<class T >
cmatrix< T > cmatrix< T >::operator* (
    const cmatrix< T > & m ) const
```

The multiplication operator element-wise.

**Parameters**

<i>m</i>	The matrix to multiply.
----------	-------------------------

**Returns**

`cmatrix<T>` The product of the matrices.

**Note**

The matrix must be of the same type of the matrix.  
PARALLELIZED METHOD with OpenMP.

**6.6.2.7 operator\*() [2/2]**

```
template<class T >
cmatrix< T > cmatrix< T >::operator* (
    const T & n ) const
```

The multiplication operator.

**Parameters**

<i>n</i>	The value to multiply.
----------	------------------------

**Returns**

`cmatrix<T>` The product of the matrices.

**6.6.2.8 operator\*=( ) [1/2]**

```
template<class T >
cmatrix< T > & cmatrix< T >::operator*= (
    const cmatrix< T > & m )
```

The multiplication assignment operator.

**Parameters**

<i>m</i>	The matrix to multiply.
----------	-------------------------

**Returns**

`cmatrix<T>&` The product of the matrices.

**Note**

The matrix must be of the same type of the matrix.  
PARALLELIZED METHOD with OpenMP.

**6.6.2.9 operator\*=( ) [2/2]**

```
template<class T >
cmatrix< T > & cmatrix< T >::operator*= (
    const T & n )
```

The multiplication assignment operator.

**Parameters**

<i>n</i>	The value to multiply.
----------	------------------------

**Returns**

`cmatrix<T>&` The product of the matrices.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.2.10 operator+()** [1/2]

```
template<class T >
cmatrix< T > cmatrix< T >::operator+ (
    const cmatrix< T > & m ) const
```

The addition operator.

**Parameters**

<i>m</i>	The matrix to add.
----------	--------------------

**Returns**

cmatrix<T> The sum of the matrices.

**Note**

The matrix must be of the same type of the matrix.  
PARALLELIZED METHOD with OpenMP.

**6.6.2.11 operator+()** [2/2]

```
template<class T >
cmatrix< T > cmatrix< T >::operator+ (
    const T & n ) const
```

The addition operator.

**Parameters**

<i>n</i>	The value to add.
----------	-------------------

**Returns**

cmatrix<T> The sum of the matrices.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.2.12 operator+=()** [1/2]

```
template<class T >
cmatrix< T > & cmatrix< T >::operator+= (
    const cmatrix< T > & m )
```

The addition assignment operator.

## Parameters

<i>m</i>	The matrix to add.
----------	--------------------

## Returns

`cmatrix<T>&` The sum of the matrices.

## Note

The matrix must be of the same type of the matrix.

PARALLELIZED METHOD with OpenMP.

6.6.2.13 `operator+=()` [2/2]

```
template<class T >
cmatrix< T > & cmatrix< T >::operator+= (
    const T & n )
```

The addition assignment operator.

## Parameters

<i>n</i>	The value to add.
----------	-------------------

## Returns

`cmatrix<T>&` The sum of the matrices.

## Note

PARALLELIZED METHOD with OpenMP.

6.6.2.14 `operator-()` [1/2]

```
template<class T >
cmatrix< T > cmatrix< T >::operator- (
    const cmatrix< T > & m ) const
```

The subtraction operator.

## Parameters

<i>m</i>	The matrix to subtract.
----------	-------------------------

**Returns**

`cmatrix<T>` The difference of the matrices.

**Note**

PARALLELIZED METHOD with OpenMP.

The matrix must be of the same type of the matrix.

**6.6.2.15 operator-() [2/2]**

```
template<class T >
cmatrix< T > cmatrix< T >::operator- (
    const T & val ) const
```

The subtraction operator.

**Parameters**

<i>n</i>	The value to subtract.
----------	------------------------

**Returns**

`cmatrix<T>` The difference of the matrices.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.2.16 operator-=() [1/2]**

```
template<class T >
cmatrix< T > & cmatrix< T >::operator-= (
    const cmatrix< T > & m )
```

The subtraction assignment operator.

**Parameters**

<i>m</i>	The matrix to subtract.
----------	-------------------------

**Returns**

`cmatrix<T>&` The difference of the matrices.

**Note**

The matrix must be of the same type of the matrix.  
 PARALLELIZED METHOD with OpenMP.

**6.6.2.17 operator-=( ) [2/2]**

```
template<class T >
cmatrix< T > & cmatrix< T >::operator-= (
    const T & n )
```

The subtraction assignment operator.

**Parameters**

<i>n</i>	The value to subtract.
----------	------------------------

**Returns**

cmatrix<T>& The difference of the matrices.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.2.18 operator/( )**

```
template<class T >
cmatrix< T > cmatrix< T >::operator/ (
    const T & n ) const
```

The division operator.

**Parameters**

<i>n</i>	The value to divide.
----------	----------------------

**Returns**

cmatrix<T> The quotient of the matrices.

**Note**

PARALLELIZED METHOD with OpenMP.

### 6.6.2.19 operator/=( )

```
template<class T >
cmatrix< T > & cmatrix< T >::operator/= (
    const T & n )
```

The division assignment operator.

#### Parameters

<i>n</i>	The value to divide.
----------	----------------------

#### Returns

cmatrix<T>& The quotient of the matrices.

#### Note

PARALLELIZED METHOD with OpenMP.

### 6.6.2.20 operator<( )

```
template<class T >
cmatrix< short unsigned int > cmatrix< T >::operator< (
    const T & n ) const
```

The strictly less than operator comparing the matrix with a value.

#### Parameters

<i>val</i>	The value to compare.
------------	-----------------------

#### Returns

cmatrix<short unsigned int> The matrix of booleans.

### 6.6.2.21 operator<=( )

```
template<class T >
cmatrix< short unsigned int > cmatrix< T >::operator<= (
    const T & n ) const
```

The less than operator comparing the matrix with a value.



**Parameters**

<i>val</i>	The value to compare.
------------	-----------------------

**Returns**

`cmatrix<short unsigned int>` The matrix of booleans.

**6.6.2.22 operator=() [1/2]**

```
template<class T >
cmatrix< T > & cmatrix< T >::operator= (
    const cmatrix< T > & m )
```

The assignment operator.

**Parameters**

<i>m</i>	The matrix to copy.
----------	---------------------

**Returns**

`cmatrix<T>&` The copied matrix.

**Note**

The matrix must be of the same type of the matrix.

**6.6.2.23 operator=() [2/2]**

```
template<class T >
cmatrix< T > & cmatrix< T >::operator= (
    const std::initializer_list< std::initializer_list< T >> & m )
```

The assignment operator.

**Parameters**

<i>m</i>	The matrix to copy.
----------	---------------------

**Returns**

`cmatrix<T>&` The copied matrix.

**Note**

The matrix must be of the same type of the matrix.

**6.6.2.24 operator==( ) [1/2]**

```
template<class T >
bool cmatrix< T >::operator== (
    const cmatrix< T > & m ) const
```

The equality operator.

**Parameters**

<i>m</i>	The matrix to compare.
----------	------------------------

**Returns**

true If the matrices are equal.  
false If the matrices are not equal.

**Note**

The matrix must be of the same type of the matrix.

**6.6.2.25 operator==( ) [2/2]**

```
template<class T >
cmatrix< short unsigned int > cmatrix< T >::operator== (
    const T & n ) const
```

The equality operator comparing the matrix with a value.

**Parameters**

<i>val</i>	The value to compare.
------------	-----------------------

**Returns**

cmatrix<short unsigned int> The matrix of booleans.

**6.6.2.26 operator>()**

```
template<class T >
cmatrix< short unsigned int > cmatrix< T >::operator> (
    const T & n ) const
```

The strictly greater than operator comparing the matrix with a value.

**Parameters**

<i>val</i>	The value to compare.
------------	-----------------------

**Returns**

cmatrix<short unsigned int> The matrix of booleans.

**6.6.2.27 operator>=()**

```
template<class T >
cmatrix< short unsigned int > cmatrix< T >::operator>= (
    const T & n ) const
```

The greater than operator comparing the matrix with a value.

**Parameters**

<i>val</i>	The value to compare.
------------	-----------------------

**Returns**

cmatrix<short unsigned int> The matrix of booleans.

**6.6.2.28 operator^()**

```
template<class T >
cmatrix< T > cmatrix< T >::operator^ (
    const unsigned int & m ) const
```

The power operator element-wise.

**Parameters**

<i>m</i>	The power. Must be a positive integer.
----------	----------------------------------------

**Returns**

`cmatrix<T>` The powered matrix.

**Exceptions**

<code>std::invalid_argument</code>	If the matrix is not a square matrix.
------------------------------------	---------------------------------------

**6.6.2.29 operator^=()**

```
template<class T >
cmatrix< T > & cmatrix< T >::operator^= (
    const unsigned int & m )
```

The power assignment operator.

**Parameters**

<code>m</code>	The power. Must be a positive integer.
----------------	----------------------------------------

**Returns**

`cmatrix<T>&` The powered matrix.

**Exceptions**

<code>std::invalid_argument</code>	If the matrix is not a square matrix.
------------------------------------	---------------------------------------

**6.6.3 Friends****6.6.3.1 operator\***

```
template<class T >
template<class U >
cmatrix<U> operator* (
    const U & n,
    const cmatrix< U > & m ) [friend]
```

The multiplication operator.

**Parameters**

<code>n</code>	The value to multiply.
<code>m</code>	The matrix to multiply.

**Returns**

`cmatrix<T>` The product of the matrices.

**6.6.3.2 operator+**

```
template<class T >
template<class U >
cmatrix<U> operator+ (
    const U & n,
    const cmatrix< U > & m ) [friend]
```

The addition operator.

**Parameters**

<i>n</i>	The value to add.
<i>m</i>	The matrix to add.

**Returns**

`cmatrix<T>` The sum of the matrices.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.3.3 operator- [1/2]**

```
template<class T >
template<class U >
cmatrix<U> operator- (
    const cmatrix< U > & m ) [friend]
```

The negation operator.

**Parameters**

<i>m</i>	The matrix to negate.
----------	-----------------------

**Returns**

`cmatrix<T>` The negated matrix.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.3.4 operator- [2/2]**

```
template<class T >
template<class U >
cmatrix<U> operator- (
    const U & n,
    const cmatrix< U > & m ) [friend]
```

The subtraction operator.

**Parameters**

<i>n</i>	The value to subtract.
<i>m</i>	The matrix to subtract.

**Returns**

cmatrix<T> The difference of the matrices.

**Note**

PARALLELIZED METHOD with OpenMP.

**6.6.3.5 operator<<**

```
template<class T >
template<class U >
std::ostream& operator<< (
    std::ostream & out,
    const cmatrix< U > & m ) [friend]
```

The output operator.

**Parameters**

<i>out</i>	The output stream.
<i>m</i>	The matrix to print.

**Returns**

std::ostream& The output stream.

## 6.7 CMatrixSetter

### Functions

- void `cmatrix< T >::set_row` (const size\_t &n, const std::vector< T > &val)  
*Set a row of the matrix.*
- void `cmatrix< T >::set_column` (const size\_t &n, const std::vector< T > &val)  
*Set a column of the matrix.*
- void `cmatrix< T >::set_cell` (const size\_t &row, const size\_t &col, const T &val)  
*Set a cell of the matrix.*
- void `cmatrix< T >::set_diag` (const std::vector< T > &val)  
*Set the diagonal of the matrix.*

### 6.7.1 Detailed Description

### 6.7.2 Function Documentation

#### 6.7.2.1 set\_cell()

```
template<class T >
void cmatrix< T >::set_cell (
    const size_t & row,
    const size_t & col,
    const T & val )
```

Set a cell of the matrix.

#### Parameters

<i>row</i>	The row of the cell to set.
<i>col</i>	The column of the cell to set.
<i>val</i>	The value to set.

#### Exceptions

<code>std::out_of_range</code>	If the index is out of range.
--------------------------------	-------------------------------

#### Note

The cell must be of the same type of the matrix.

#### 6.7.2.2 set\_column()

```
template<class T >
void cmatrix< T >::set_column (
```

```
const size_t & n,
const std::vector< T > & val )
```

Set a column of the matrix.

#### Parameters

<i>n</i>	The index of the column to set.
<i>val</i>	The value to set.

#### Exceptions

<i>std::out_of_range</i>	If the index is out of range.
<i>std::invalid_argument</i>	If the size of the vector <i>val</i> is not equal to the number of rows of the matrix.

#### Note

The column must be a vector of the same type of the matrix.

### 6.7.2.3 set\_diag()

```
template<class T >
void cmatrix< T >::set_diag (
    const std::vector< T > & val )
```

Set the diagonal of the matrix.

#### Parameters

<i>val</i>	The diagonal to set.
------------	----------------------

#### Exceptions

<i>std::invalid_argument</i>	If the size of the vector <i>val</i> is not equal to the minimum of the number of rows and columns of the matrix.
------------------------------	-------------------------------------------------------------------------------------------------------------------

#### Note

The diagonal must be a vector of the same type of the matrix.

### 6.7.2.4 set\_row()

```
template<class T >
void cmatrix< T >::set_row (
```



```
const size_t & n,  
const std::vector< T > & val )
```

Set a row of the matrix.

**Parameters**

<i>n</i>	The index of the row to set.
<i>val</i>	The value to set.

**Exceptions**

<i>std::out_of_range</i>	If the index is out of range.
<i>std::invalid_argument</i>	If the size of the vector <code>val</code> is not equal to the number of columns of the matrix.

**Note**

The row must be a vector of the same type of the matrix.

## 6.8 CMatrixStatic

### Functions

- static bool `cmatrix< T >::is_matrix` (const std::vector< std::vector< T >> &m)  
*Check if a nested vector is a matrix. To be a matrix, all the rows and columns must have the same length.*
- static `cmatrix< int > cmatrix< T >::randint` (const size\_t &height, const size\_t &width, const int &min=0, const int &max=100, const int &seed=time(nullptr))  
*Generate a random matrix of integers.*
- static `cmatrix< float > cmatrix< T >::randfloat` (const size\_t &height, const size\_t &width, const float &min=0, const float &max=1, const int &seed=time(nullptr))  
*Generate a random matrix of floats.*
- static `cmatrix< int > cmatrix< T >::zeros` (const size\_t &width, const size\_t &height)  
*Generate a matrix of zeros.*
- static `cmatrix< int > cmatrix< T >::identity` (const size\_t &size)  
*Generate the identity matrix.*
- static `cmatrix< T > cmatrix< T >::merge` (const `cmatrix< T >` &m1, const `cmatrix< T >` &m2, const unsigned int &axis=0)  
*Merge two matrices.*

### 6.8.1 Detailed Description

### 6.8.2 Function Documentation

#### 6.8.2.1 identity()

```
template<class T >
static cmatrix<int> cmatrix< T >::identity (
    const size_t & size ) [static]
```

Generate the identity matrix.

#### Parameters

<code>size</code>	The number of rows and columns.
-------------------	---------------------------------

#### Returns

`cmatrix<int>` The identity matrix.

#### 6.8.2.2 is\_matrix()

```
template<class T >
bool cmatrix< T >::is_matrix (
    const std::vector< std::vector< T >> & m ) [static]
```

Check if a nested vector is a matrix. To be a matrix, all the rows and columns must have the same length.

## Parameters

<i>m</i>	The nested vector to check.
----------	-----------------------------

## Returns

true If the nested vector is a matrix.  
false If the nested vector is not a matrix.

## 6.8.2.3 merge()

```
template<class T >
cmatrix< T > cmatrix< T >::merge (
    const cmatrix< T > & m1,
    const cmatrix< T > & m2,
    const unsigned int & axis = 0 ) [static]
```

Merge two matrices.

## Parameters

<i>m1</i>	The first matrix.
<i>m2</i>	The second matrix.
<i>axis</i>	The axis to merge. 0 for the rows, 1 for the columns. (default: 0)

## Returns

cmatrix<T> The merged matrix.

## 6.8.2.4 randfloat()

```
template<class T >
static cmatrix<float> cmatrix< T >::randfloat (
    const size_t & height,
    const size_t & width,
    const float & min = 0,
    const float & max = 1,
    const int & seed = time(nullptr) ) [static]
```

Generate a random matrix of floats.

## Parameters

<i>height</i>	The number of rows.
<i>width</i>	The number of columns.
<i>min</i>	The minimum value of the matrix (included). (default: 0)
<i>max</i>	The maximum value of the matrix (included). (default: 1)
<i>seed</i>	The seed of the random generator. (default: time(nullptr))

**Returns**

`cmatrix<float>` The random matrix of floats.

**6.8.2.5 randint()**

```
template<class T >
static cmatrix<int> cmatrix< T >::randint (
    const size_t & height,
    const size_t & width,
    const int & min = 0,
    const int & max = 100,
    const int & seed = time(nullptr) ) [static]
```

Generate a random matrix of integers.

**Parameters**

<i>height</i>	The number of height.
<i>width</i>	The number of columns.
<i>min</i>	The minimum value of the matrix (included). (default: 0)
<i>max</i>	The maximum value of the matrix (included). (default: 100)
<i>seed</i>	The seed of the random generator. (default: <code>time(nullptr)</code> )

**Returns**

`cmatrix<int>` The random matrix of integers.

**6.8.2.6 zeros()**

```
template<class T >
static cmatrix<int> cmatrix< T >::zeros (
    const size_t & width,
    const size_t & height ) [static]
```

Generate a matrix of zeros.

**Parameters**

<i>width</i>	The number of columns.
<i>height</i>	The number of rows.

**Returns**

`cmatrix<int>` The matrix of zeros.

## 6.9 CMatrixStatistics

### Functions

- `cmatrix< float > cmatrix< T >::__mean` (const unsigned int &axis, std::true\_type) const  
*Compute the mean value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is arithmetic.*
- `cmatrix< float > cmatrix< T >::__mean` (const unsigned int &axis, std::false\_type) const  
*Compute the mean value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is not arithmetic.*
- `cmatrix< float > cmatrix< T >::__std` (const unsigned int &axis, std::true\_type) const  
*Compute the std value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is arithmetic.*
- `cmatrix< float > cmatrix< T >::__std` (const unsigned int &axis, std::false\_type) const  
*Compute the std value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is not arithmetic.*
- `cmatrix< T > cmatrix< T >::min` (const unsigned int &axis=0) const  
*Get the minimum value for each row (axis: 0) or column (axis: 1) of the matrix.*
- `T cmatrix< T >::min_all` () const  
*Get the minimum value of all the elements of the matrix.*
- `cmatrix< T > cmatrix< T >::max` (const unsigned int &axis=0) const  
*Get the maximum value for each row (axis: 0) or column (axis: 1) of the matrix.*
- `T cmatrix< T >::max_all` () const  
*Get the maximum value of all the elements of the matrix.*
- `cmatrix< T > cmatrix< T >::sum` (const unsigned int &axis=0, const T &zero=T()) const  
*Get the sum of the matrix for each row (axis: 0) or column (axis: 1) of the matrix.*
- `T cmatrix< T >::sum_all` (const T &zero=T()) const  
*Get the sum of all the elements of the matrix.*
- `cmatrix< float > cmatrix< T >::mean` (const unsigned int &axis=0) const  
*Get the mean value for each row (axis: 0) or column (axis: 1) of the matrix.*
- `cmatrix< float > cmatrix< T >::std` (const unsigned int &axis=0) const  
*Get the standard deviation value for each row (axis: 0) or column (axis: 1) of the matrix.*
- `cmatrix< T > cmatrix< T >::median` (const unsigned int &axis=0) const  
*Get the median value for each row (axis: 0) or column (axis: 1) of the matrix.*

### 6.9.1 Detailed Description

### 6.9.2 Function Documentation

#### 6.9.2.1 \_\_mean() [1/2]

```
template<typename T >
cmatrix< float > cmatrix< T >::__mean (
    const unsigned int & axis,
    std::false_type ) const [private]
```

Compute the mean value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is not arithmetic.

**Parameters**

<i>axis</i>	The axis to get the mean value. 0 for the rows, 1 for the columns. (default: 0)
<i>false_type</i>	The type of the matrix is not arithmetic.

**Exceptions**

<i>std::invalid_argument</i>	If the matrix is not arithmetic.
------------------------------	----------------------------------

**6.9.2.2 `__mean()` [2/2]**

```
template<typename T >
cmatrix< float > cmatrix< T >::__mean (
    const unsigned int & axis,
    std::true_type ) const [private]
```

Compute the mean value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is arithmetic.

**Parameters**

<i>axis</i>	The axis to get the mean value. 0 for the rows, 1 for the columns. (default: 0)
<i>true_type</i>	The type of the matrix is arithmetic.

**Returns**

`cmatrix<float>` The mean value for each row or column of the matrix.

**Exceptions**

<i>std::invalid_argument</i>	If the axis is not 0 or 1.
------------------------------	----------------------------

**6.9.2.3 `__std()` [1/2]**

```
template<class T >
cmatrix< float > cmatrix< T >::__std (
    const unsigned int & axis,
    std::false_type ) const [private]
```

Compute the std value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is not arithmetic.



## Parameters

<i>axis</i>	The axis to get the std value. 0 for the rows, 1 for the columns. (default: 0)
<i>false_type</i>	The type of the matrix is not arithmetic.

## Exceptions

<i>std::invalid_argument</i>	If the matrix is not arithmetic.
------------------------------	----------------------------------

## 6.9.2.4 \_\_std() [2/2]

```
template<class T >
cmatrix< float > cmatrix< T >::__std (
    const unsigned int & axis,
    std::true_type ) const [private]
```

Compute the std value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is arithmetic.

## Parameters

<i>axis</i>	The axis to get the std value. 0 for the rows, 1 for the columns. (default: 0)
<i>true_type</i>	The type of the matrix is arithmetic.

## Returns

cmatrix<float> The std value for each row or column of the matrix.

## Exceptions

<i>std::invalid_argument</i>	If the axis is not 0 or 1.
------------------------------	----------------------------

## Note

PARALLELIZED METHOD with OpenMP.

## 6.9.2.5 max()

```
template<class T >
cmatrix< T > cmatrix< T >::max (
    const unsigned int & axis = 0 ) const
```

Get the maximum value for each row (axis: 0) or column (axis: 1) of the matrix.

**Parameters**

<i>axis</i>	The axis to get the maximum value. 0 for the rows, 1 for the columns. (default: 0)
-------------	------------------------------------------------------------------------------------

**Returns**

`cmatrix<T>` The maximum value for each row or column of the matrix.

**Exceptions**

<code>std::invalid_argument</code>	If the axis is not 0 or 1.
------------------------------------	----------------------------

**Note**

The type of the matrix must implement the operator `>`.  
 PARALLELIZED METHOD with OpenMP.

**6.9.2.6 max\_all()**

```
template<class T >
T cmatrix< T >::max_all
```

Get the maximum value of all the elements of the matrix.

**Returns**

`T` The maximum value of all the elements of the matrix.

**Exceptions**

<code>std::invalid_argument</code>	If the matrix is empty.
------------------------------------	-------------------------

**Note**

The type of the matrix must implement the operator `>`.

**6.9.2.7 mean()**

```
template<typename T >
cmatrix< float > cmatrix< T >::mean (
    const unsigned int & axis = 0 ) const
```

Get the mean value for each row (axis: 0) or column (axis: 1) of the matrix.

## Parameters

<i>axis</i>	The axis to get the mean value. 0 for the rows, 1 for the columns. (default: 0)
-------------	---------------------------------------------------------------------------------

## Returns

`cmatrix<float>` The mean value for each row or column of the matrix.

## Exceptions

<i>std::invalid_argument</i>	If the axis is not 0 or 1.
<i>std::invalid_argument</i>	If the matrix is not arithmetic.

## Note

The matrix must be of arithmetic type.

## 6.9.2.8 median()

```
template<class T >
cmatrix< T > cmatrix< T >::median (
    const unsigned int & axis = 0 ) const
```

Get the median value for each row (axis: 0) or column (axis: 1) of the matrix.

## Parameters

<i>axis</i>	The axis to get the median value. 0 for the rows, 1 for the columns. (default: 0)
-------------	-----------------------------------------------------------------------------------

## Returns

`cmatrix<T>` The median value of the matrix for each row or column of the matrix.

## Exceptions

<i>std::invalid_argument</i>	If the axis is not 0 or 1.
------------------------------	----------------------------

## Note

The matrix must implement the operator `<`.

If the number of elements is even, the median is the smallest value of the two middle values.

PARALLELIZED METHOD with OpenMP.

### 6.9.2.9 min()

```
template<class T >
cmatrix< T > cmatrix< T >::min (
    const unsigned int & axis = 0 ) const
```

Get the minimum value for each row (axis: 0) or column (axis: 1) of the matrix.

#### Parameters

<i>axis</i>	The axis to get the minimum value. 0 for the rows, 1 for the columns. (default: 0)
-------------	------------------------------------------------------------------------------------

#### Returns

cmatrix<T> The minimum value for each row or column of the matrix.

#### Exceptions

<i>std::invalid_argument</i>	If the axis is not 0 or 1.
------------------------------	----------------------------

#### Note

The type of the matrix must implement the operator <.  
PARALLELIZED METHOD with OpenMP.

### 6.9.2.10 min\_all()

```
template<class T >
T cmatrix< T >::min_all
```

Get the minimum value of all the elements of the matrix.

#### Returns

T The minimum value of all the elements of the matrix.

#### Exceptions

<i>std::invalid_argument</i>	If the matrix is empty.
------------------------------	-------------------------

#### Note

The type of the matrix must implement the operator <.

**6.9.2.11 std()**

```
template<class T >
cmatrix< float > cmatrix< T >::std (
    const unsigned int & axis = 0 ) const
```

Get the standard deviation value for each row (axis: 0) or column (axis: 1) of the matrix.

**Parameters**

<i>axis</i>	The axis to get the standard deviation. 0 for the rows, 1 for the columns. (default: 0)
-------------	-----------------------------------------------------------------------------------------

**Returns**

cmatrix<float> The standard deviation for each row or column of the matrix.

**Exceptions**

<i>std::invalid_argument</i>	If the axis is not 0 or 1.
<i>std::invalid_argument</i>	If the matrix is not arithmetic.
<i>std::invalid_argument</i>	If the number of elements is less than 2 for the axis.

**Note**

The matrix must be of arithmetic type.

PARALLELIZED METHOD with OpenMP.

**6.9.2.12 sum()**

```
template<class T >
cmatrix< T > cmatrix< T >::sum (
    const unsigned int & axis = 0,
    const T & zero = T() ) const
```

Get the sum of the matrix for each row (axis: 0) or column (axis: 1) of the matrix.

**Parameters**

<i>axis</i>	The axis to get the sum. 0 for the rows, 1 for the columns. (default: 0)
<i>zero</i>	The zero value of the sum. (default: the value of the default constructor of the type T)

**Returns**

cmatrix<T> The sum of the matrix.

### Exceptions

<code>std::invalid_argument</code>	If the axis is not 0 or 1.
------------------------------------	----------------------------

### Note

PARALLELIZED METHOD with OpenMP.

### 6.9.2.13 `sum_all()`

```
template<class T >
T cmatrix< T >::sum_all (
    const T & zero = T() ) const
```

Get the sum of all the elements of the matrix.

### Parameters

<i>zero</i>	The zero value of the sum. (default: the value of the default constructor of the type T)
-------------	------------------------------------------------------------------------------------------

### Returns

T The sum of all the elements of the matrix.

## Chapter 7

# Class Documentation

### 7.1 `cmatrix< T >` Class Template Reference

The main template class that can work with any data type except bool.

```
#include <CMatrix.hpp>
```

#### Public Member Functions

- `cmatrix` (const std::initializer\_list< std::initializer\_list< T >> &m)  
*Construct a new cmatrix object.*
- `cmatrix` (const std::vector< std::vector< T >> &m)  
*Construct a new cmatrix object.*
- `cmatrix` ()  
*Construct a new cmatrix object.*
- `cmatrix` (const size\_t &height, const size\_t &width)  
*Construct a new cmatrix object.*
- `cmatrix` (const size\_t &height, const size\_t &width, const T &val)  
*Construct a new cmatrix object.*
- template<class U >  
  `cmatrix` (const `cmatrix`< U > &m)  
  *Cast a matrix to another type.*
- `~cmatrix` ()
- std::vector< T > `rows_vec` (const size\_t &n) const  
  *Get a row of the matrix.*
- std::vector< T > `columns_vec` (const size\_t &n) const  
  *Get a column of the matrix as a flattened vector.*
- `cmatrix`< T > `rows` (const size\_t &ids) const  
  *Get the rows of the matrix.*
- `cmatrix`< T > `rows` (const std::initializer\_list< size\_t > &ids) const  
  *Get the rows of the matrix.*
- `cmatrix`< T > `rows` (const std::vector< size\_t > &ids) const  
  *Get the rows of the matrix.*
- `cmatrix`< T > `columns` (const size\_t &ids) const  
  *Get the columns of the matrix.*

- `cmatrix< T > columns` (const std::initializer\_list< size\_t > &ids) const  
*Get the columns of the matrix.*
- `cmatrix< T > columns` (const std::vector< size\_t > &ids) const  
*Get the columns of the matrix.*
- `cmatrix< T > cells` (const size\_t &row, const size\_t &col) const  
*Get the cells of the matrix.*
- `cmatrix< T > cells` (const std::initializer\_list< std::pair< size\_t, size\_t >> &ids) const  
*Get the cells of the matrix.*
- `cmatrix< T > cells` (const std::vector< std::pair< size\_t, size\_t >> &ids) const  
*Get the cells of the matrix.*
- `T & cell` (const size\_t &row, const size\_t &col)  
*Get the reference to a cell of the matrix.*
- `T cell` (const size\_t &row, const size\_t &col) const  
*Get a cell of the matrix.*
- `cmatrix< T > slice_rows` (const size\_t &start, const size\_t &end) const  
*Get the rows between two indexes.*
- `cmatrix< T > slice_columns` (const size\_t &start, const size\_t &end) const  
*Get the columns between two indexes.*
- `size_t width` () const  
*The number of columns of the matrix.*
- `size_t height` () const  
*The number of rows of the matrix.*
- `std::pair< size_t, size_t > size` () const  
*The dimensions of the matrix.*
- `cmatrix< T > transpose` () const  
*Get the transpose of the matrix.*
- `std::vector< T > diag` () const  
*Get the diagonal of the matrix.*
- `void set_row` (const size\_t &n, const std::vector< T > &val)  
*Set a row of the matrix.*
- `void set_column` (const size\_t &n, const std::vector< T > &val)  
*Set a column of the matrix.*
- `void set_cell` (const size\_t &row, const size\_t &col, const T &val)  
*Set a cell of the matrix.*
- `void set_diag` (const std::vector< T > &val)  
*Set the diagonal of the matrix.*
- `void insert_row` (const size\_t &pos, const std::vector< T > &val)  
*Insert a column in the matrix.*
- `void insert_column` (const size\_t &pos, const std::vector< T > &val)  
*Insert a row in the matrix.*
- `void push_row_front` (const std::vector< T > &val)  
*Push a row in the front of the matrix.*
- `void push_row_back` (const std::vector< T > &val)  
*Push a row in the back of the matrix.*
- `void push_col_front` (const std::vector< T > &val)  
*Push a column in the front of the matrix.*
- `void push_col_back` (const std::vector< T > &val)  
*Push a column in the back of the matrix.*
- `int find_row` (const std::function< bool(std::vector< T >) > &f) const
- `int find_row` (const std::vector< T > &val) const  
*Find the first row matching the given row.*



- `int find_column (const std::function< bool(std::vector< T >)> &f) const`  
*Find the first column matching the condition.*
- `int find_column (const std::vector< T > &val) const`  
*Find the first column matching the given column.*
- `std::tuple< int, int > find (const std::function< bool(T)> &f) const`  
*Find the first cell matching the condition.*
- `std::tuple< int, int > find (const T &val) const`  
*Find the first cell matching the given cell.*
- `void remove_row (const size_t &n)`  
*Remove a row of the matrix.*
- `void remove_column (const size_t &n)`  
*Remove a column of the matrix.*
- `void concatenate (const cmatrix< T > &m, const unsigned int &axis=0)`  
*Concatenate a matrix to the matrix.*
- `bool is_empty () const`  
*Check if the matrix is empty.*
- `bool is_square () const`  
*Check if the matrix is a square matrix.*
- `bool is_diag () const`  
*Check if the matrix is a diagonal matrix.*
- `bool is_identity () const`  
*Check if the matrix is the identity matrix.*
- `bool is_symetric () const`  
*Check if the matrix is a symmetric matrix.*
- `bool is_triangular_up () const`  
*Check if the matrix is an upper triangular matrix.*
- `bool is_triangular_low () const`  
*Check if the matrix is a lower triangular matrix.*
- `bool all (const std::function< bool(T)> &f) const`  
*Check if all the cells of the matrix satisfy a condition.*
- `bool all (const T &val) const`  
*Check if all the cells of the matrix are equal to a value.*
- `bool any (const std::function< bool(T)> &f) const`  
*Check if at least one cell of the matrix satisfy a condition.*
- `bool any (const T &val) const`  
*Check if at least one cell of the matrix is equal to a value.*
- `cmatrix< T > min (const unsigned int &axis=0) const`  
*Get the minimum value for each row (axis: 0) or column (axis: 1) of the matrix.*
- `T min_all () const`  
*Get the minimum value of all the elements of the matrix.*
- `cmatrix< T > max (const unsigned int &axis=0) const`  
*Get the maximum value for each row (axis: 0) or column (axis: 1) of the matrix.*
- `T max_all () const`  
*Get the maximum value of all the elements of the matrix.*
- `cmatrix< T > sum (const unsigned int &axis=0, const T &zero=T()) const`  
*Get the sum of the matrix for each row (axis: 0) or column (axis: 1) of the matrix.*
- `T sum_all (const T &zero=T()) const`  
*Get the sum of all the elements of the matrix.*
- `cmatrix< float > mean (const unsigned int &axis=0) const`  
*Get the mean value for each row (axis: 0) or column (axis: 1) of the matrix.*
- `cmatrix< float > std (const unsigned int &axis=0) const`

- Get the standard deviation value for each row (axis: 0) or column (axis: 1) of the matrix.*

  - `cmatrix< T > median` (const unsigned int &axis=0) const
- Get the median value for each row (axis: 0) or column (axis: 1) of the matrix.*

  - `bool near` (const `cmatrix< T >` &val, const T &tolerance=1e-5) const

*Test if the matrix is near another matrix.*

  - `bool near` (const T &val, const T &tolerance=1e-5) const

*Test if the matrix is near a value.*

  - `bool nearq` (const `cmatrix< T >` &val, const T &tolerance=1e-5) const

*Test if the matrix is not near another matrix.*

  - `bool nearq` (const T &val, const T &tolerance=1e-5) const

*Test if the matrix is not near a value.*
- `cmatrix< T > matmul` (const `cmatrix< T >` &m) const

*Get the product with another matrix.*
- `cmatrix< T > matpow` (const unsigned int &n) const

*Get the power of the matrix.*
- `cmatrix< T > log` () const

*Get the natural logarithm of the matrix.*
- `cmatrix< T > log2` () const

*Get the log2 of the matrix.*
- `cmatrix< T > log10` () const

*Get the log10 of the matrix.*
- `cmatrix< T > exp` () const

*Get the exponential of the matrix.*
- `void print` () const

*Print the matrix in the standard output.*
- `void clear` ()

*Clear the matrix.*
- `cmatrix< T > copy` () const

*Copy the matrix.*
- `void apply` (const std::function< T(T, size\_t, size\_t)> &f)

*Apply a function to each cell of the matrix.*
- `void apply` (const std::function< T(T)> &f)

*Apply a function to each cell of the matrix.*
- `cmatrix< T > map` (const std::function< T(T, size\_t, size\_t)> &f) const

*Apply a function to each cell of the matrix and return the result.*
- `template<class U >`  
`cmatrix< U > map` (const std::function< U(T, size\_t, size\_t)> &f) const

*Apply a function to each cell of the matrix and return the result.*
- `cmatrix< T > map` (const std::function< T(T)> &f) const

*Apply a function to each cell of the matrix and return the result.*
- `template<class U >`  
`cmatrix< U > map` (const std::function< U(T)> &f) const

*Apply a function to each cell of the matrix and return the result.*
- `void fill` (const T &val)

*Fill the matrix with a value.*
- `std::vector< std::vector< T > >` `to_vector` () const

*Convert the matrix to a vector.*
- `template<class U >`  
`cmatrix< U > cast` () const

*Convert the matrix to a matrix of another type.*
- `cmatrix< int >` `to_int` () const

- Convert the matrix to a matrix of integers.*

  - `cmatrix< float > to_float () const`
- Convert the matrix to a matrix of floats.*

  - `cmatrix< std::string > to_string () const`
- Convert the matrix to a matrix of strings.*

  - `cmatrix< T > & operator= (const std::initializer_list< std::initializer_list< T >> &m)`
- The assignment operator.*

  - `cmatrix< T > & operator= (const cmatrix< T > &m)`
- The assignment operator.*

  - `bool operator== (const cmatrix< T > &m) const`
- The equality operator.*

  - `bool operator!= (const cmatrix< T > &m) const`
- The inequality operator.*

  - `cmatrix< short unsigned int > operator== (const T &n) const`
- The equality operator comparing the matrix with a value.*

  - `cmatrix< short unsigned int > operator!= (const T &n) const`
- The inequality operator comparing the matrix with a value.*

  - `cmatrix< short unsigned int > operator< (const T &n) const`
- The strictly less than operator comparing the matrix with a value.*

  - `cmatrix< short unsigned int > operator<= (const T &n) const`
- The less than operator comparing the matrix with a value.*

  - `cmatrix< short unsigned int > operator> (const T &n) const`
- The strictly greater than operator comparing the matrix with a value.*

  - `cmatrix< short unsigned int > operator>= (const T &n) const`
- The greater than operator comparing the matrix with a value.*

  - `cmatrix< T > operator+ (const cmatrix< T > &m) const`
- The addition operator.*

  - `cmatrix< T > operator+ (const T &n) const`
- The addition operator.*

  - `cmatrix< T > operator- (const cmatrix< T > &m) const`
- The subtraction operator.*

  - `cmatrix< T > operator- (const T &val) const`
- The subtraction operator.*

  - `cmatrix< T > operator* (const cmatrix< T > &m) const`
- The multiplication operator element-wise.*

  - `cmatrix< T > operator* (const T &n) const`
- The multiplication operator.*

  - `cmatrix< T > operator/ (const T &n) const`
- The division operator.*

  - `cmatrix< T > operator^ (const unsigned int &m) const`
- The power operator element-wise.*

  - `cmatrix< T > & operator+= (const cmatrix< T > &m)`
- The addition assignment operator.*

  - `cmatrix< T > & operator+= (const T &n)`
- The addition assignment operator.*

  - `cmatrix< T > & operator-= (const cmatrix< T > &m)`
- The subtraction assignment operator.*

  - `cmatrix< T > & operator-= (const T &n)`
- The subtraction assignment operator.*

  - `cmatrix< T > & operator*= (const cmatrix< T > &m)`
- The multiplication assignment operator.*

- `cmatrix< T > & operator*=(const T &n)`  
*The multiplication assignment operator.*
- `cmatrix< T > & operator/=(const T &n)`  
*The division assignment operator.*
- `cmatrix< T > & operator^=(const unsigned int &m)`  
*The power assignment operator.*
- `cmatrix< int > to_int () const`
- `cmatrix< float > to_float () const`
- `cmatrix< int > randint (const size_t &height, const size_t &width, const int &min, const int &max, const int &seed)`
- `cmatrix< float > randfloat (const size_t &height, const size_t &width, const float &min, const float &max, const int &seed)`
- `cmatrix< int > zeros (const size_t &width, const size_t &height)`
- `cmatrix< int > identity (const size_t &size)`

## Static Public Member Functions

- static bool `is_matrix (const std::vector< std::vector< T >> &m)`  
*Check if a nested vector is a matrix. To be a matrix, all the rows and columns must have the same length.*
- static `cmatrix< int > randint (const size_t &height, const size_t &width, const int &min=0, const int &max=100, const int &seed=time(nullptr))`  
*Generate a random matrix of integers.*
- static `cmatrix< float > randfloat (const size_t &height, const size_t &width, const float &min=0, const float &max=1, const int &seed=time(nullptr))`  
*Generate a random matrix of floats.*
- static `cmatrix< int > zeros (const size_t &width, const size_t &height)`  
*Generate a matrix of zeros.*
- static `cmatrix< int > identity (const size_t &size)`  
*Generate the identity matrix.*
- static `cmatrix< T > merge (const cmatrix< T > &m1, const cmatrix< T > &m2, const unsigned int &axis=0)`  
*Merge two matrices.*

## Private Member Functions

- void `__check_size (const std::tuple< size_t, size_t > &size) const`  
*Check if dimensions are equals to the dimensions of the matrix.*
- void `__check_size (const cmatrix< T > &m) const`  
*Check if dimensions are equals to the dimensions of the matrix.*
- void `__check_valid_row (const std::vector< T > &row) const`  
*Check if the vector is a valid row of the matrix.*
- void `__check_valid_col (const std::vector< T > &col) const`  
*Check if the vector is a valid column of the matrix.*
- void `__check_valid_diag (const std::vector< T > &diag) const`  
*Check if the diagonal is a valid diagonal of the matrix.*
- void `__check_valid_row_id (const size_t &n) const`  
*Check if the row is a valid row index of the matrix.*
- void `__check_valid_col_id (const size_t &n) const`  
*Check if the column is a valid column index of the matrix.*
- void `__check_expected_id (const size_t &n, const size_t &expected) const`  
*Check if the index is expected.*

- `void __check_expected_id` (const size\_t &n, const size\_t &expectedBegin, const size\_t &expectedEnd) const  
*Check if the index is expected.*
- `void __check_valid_type` () const  
*Check if the type of the matrix is valid. List of types not supported: bool.*
- `cmatrix< float > __mean` (const unsigned int &axis, std::true\_type) const  
*Compute the mean value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is arithmetic.*
- `cmatrix< float > __mean` (const unsigned int &axis, std::false\_type) const  
*Compute the mean value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is not arithmetic.*
- `cmatrix< float > __std` (const unsigned int &axis, std::true\_type) const  
*Compute the std value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is arithmetic.*
- `cmatrix< float > __std` (const unsigned int &axis, std::false\_type) const  
*Compute the std value for each row (axis: 0) or column (axis: 1) of the matrix. This method is used when the type of the matrix is not arithmetic.*
- `cmatrix< T > __map_op_arithmetic` (const std::function< T(T, T)> &f, const `cmatrix< T >` &m) const  
*Apply a operator to each cell of the matrix.*
- `cmatrix< T > __map_op_arithmetic` (const std::function< T(T, T)> &f, const T &val) const  
*Apply a operator to each cell of the matrix.*
- `cmatrix< short unsigned int > __map_op_comparaison_val` (const std::function< T(T, T)> &f, const T &n) const  
*Map a comparison operator to each cell of the matrix and return a matrix of boolean.*
- `template<class U >`  
`cmatrix< U > __cast` (std::true\_type) const  
*Convert the matrix to a matrix of another type.*
- `template<class U >`  
`cmatrix< U > __cast` (std::false\_type) const  
*Convert the matrix to a matrix of another type.*
- `cmatrix< std::string > __to_string` (std::true\_type) const  
*Convert the matrix to a string matrix.*
- `cmatrix< std::string > __to_string` (std::false\_type) const  
*Convert the matrix to a string matrix.*

## Private Attributes

- `std::vector< std::vector< T > >` `matrix` = `std::vector<std::vector<T>>()`

## Friends

- `template<class U >`  
`std::ostream & operator<<` (std::ostream &out, const `cmatrix< U >` &m)  
*The output operator.*
- `template<class U >`  
`cmatrix< U > operator+` (const U &n, const `cmatrix< U >` &m)  
*The addition operator.*
- `template<class U >`  
`cmatrix< U > operator-` (const U &n, const `cmatrix< U >` &m)  
*The subtraction operator.*

- `template<class U >`  
`cmatrix< U > operator-` (const `cmatrix< U >` &m)  
*The negation operator.*
- `template<class U >`  
`cmatrix< U > operator*` (const U &n, const `cmatrix< U >` &m)  
*The multiplication operator.*

### 7.1.1 Detailed Description

```
template<class T>
class cmatrix< T >
```

The main template class that can work with any data type except bool.

#### Template Parameters

<code>T</code>	The type of elements in the cmatrix.
----------------	--------------------------------------

### 7.1.2 Constructor & Destructor Documentation

#### 7.1.2.1 `cmatrix()` [1/6]

```
template<class T >
cmatrix< T >::cmatrix (
    const std::initializer_list< std::initializer_list< T >> & m )
```

Construct a new cmatrix object.

#### Parameters

<code>m</code>	The matrix to copy.
----------------	---------------------

#### Exceptions

<code>std::invalid_argument</code>	If the initializer list is not a matrix.
<code>std::invalid_argument</code>	If the type is bool.

#### 7.1.2.2 `cmatrix()` [2/6]

```
template<class T >
cmatrix< T >::cmatrix (
    const std::vector< std::vector< T >> & m )
```

Construct a new `cmatrix` object.

#### Parameters

<i>m</i>	The vector matrix.
----------	--------------------

#### Exceptions

<code>std::invalid_argument</code>	If the vector is not a matrix.
<code>std::invalid_argument</code>	If the type is <code>bool</code> .

### 7.1.2.3 `cmatrix()` [3/6]

```
template<class T >
cmatrix< T >::cmatrix
```

Construct a new `cmatrix` object.

#### Exceptions

<code>std::invalid_argument</code>	If the type is <code>bool</code> .
------------------------------------	------------------------------------

### 7.1.2.4 `cmatrix()` [4/6]

```
template<class T >
cmatrix< T >::cmatrix (
    const size_t & height,
    const size_t & width )
```

Construct a new `cmatrix` object.

#### Parameters

<i>height</i>	The number of rows.
<i>width</i>	The number of columns.

#### Exceptions

<code>std::invalid_argument</code>	If the type is <code>bool</code> .
------------------------------------	------------------------------------

### 7.1.2.5 `cmatrix()` [5/6]

```
template<class T >
cmatrix< T >::cmatrix (
    const size_t & height,
    const size_t & width,
    const T & val )
```

Construct a new `cmatrix` object.

#### Parameters

<i>height</i>	The number of rows.
<i>width</i>	The number of columns.
<i>val</i>	The value to fill the matrix.

#### Exceptions

<code>std::invalid_argument</code>	If the type is <code>bool</code> .
------------------------------------	------------------------------------

### 7.1.2.6 `cmatrix()` [6/6]

```
template<class T >
template<class U >
cmatrix< T >::cmatrix (
    const cmatrix< U > & m )
```

Cast a matrix to another type.

#### Parameters

<i>m</i>	The matrix to copy.
----------	---------------------

#### Template Parameters

<i>U</i>	The type of the matrix to copy.
----------	---------------------------------

#### Exceptions

<code>std::invalid_argument</code>	If the type is <code>bool</code> .
------------------------------------	------------------------------------

### 7.1.2.7 `~cmatrix()`

```
template<class T >
cmatrix< T >::~cmatrix
```



### 7.1.3 Member Function Documentation

#### 7.1.3.1 `identity()`

```
cmatrix< int > cmatrix< int >::identity (
    const size_t & size )
```

#### 7.1.3.2 `randfloat()`

```
cmatrix< float > cmatrix< float >::randfloat (
    const size_t & height,
    const size_t & width,
    const float & min,
    const float & max,
    const int & seed )
```

#### 7.1.3.3 `randint()`

```
cmatrix< int > cmatrix< int >::randint (
    const size_t & height,
    const size_t & width,
    const int & min,
    const int & max,
    const int & seed )
```

#### 7.1.3.4 `to_float()`

```
cmatrix< float > cmatrix< std::string >::to_float ( ) const
```

#### 7.1.3.5 `to_int()`

```
cmatrix< int > cmatrix< std::string >::to_int ( ) const
```

### 7.1.3.6 zeros()

```
cmatrix< int > cmatrix< int >::zeros (
    const size_t & width,
    const size_t & height )
```

## 7.1.4 Member Data Documentation

### 7.1.4.1 matrix

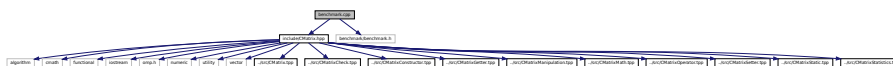
```
template<class T >
std::vector<std::vector<T> > cmatrix< T >::matrix = std::vector<std::vector<T>>>() [private]
```

The documentation for this class was generated from the following files:

- [include/CMatrix.hpp](#)
- [src/CMatrix.hpp](#)
- [src/CMatrixCheck.hpp](#)
- [src/CMatrixConstructor.hpp](#)
- [src/CMatrixGetter.hpp](#)
- [src/CMatrixManipulation.hpp](#)
- [src/CMatrixMath.hpp](#)
- [src/CMatrixOperator.hpp](#)
- [src/CMatrixSetter.hpp](#)
- [src/CMatrixStatic.hpp](#)
- [src/CMatrixStatistics.hpp](#)

## File Documentation

```
#include "include/CMatrix.hpp"
#include <benchmark/benchmark.h>
Include dependency graph for benchmark.cpp:
```



- static void **bench** (benchmark::State &state)
- **BENCHMARK** (**bench**) -> Unit(benchmark::kMillisecond)
- **BENCHMARK MAIN** ()

#### 8.1.1.1 bench()

```
static void bench (
    benchmark::State & state ) [static]
```

```
BENCHMARK (
    bench ) -> Unit(benchmark::kMillisecond)
```

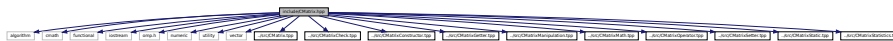
### 8.1.1.3 BENCHMARK\_MAIN()

```
BENCHMARK_MAIN ( )
```

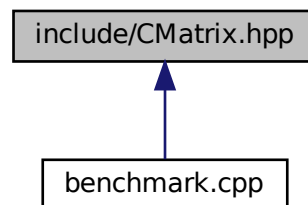
## 8.2 include/CMatrix.hpp File Reference

File containing the main template class of the 'cmatrix' library.

```
#include <algorithm>
#include <cmath>
#include <functional>
#include <iostream>
#include <omp.h>
#include <numeric>
#include <utility>
#include <vector>
#include "../src/CMatrix.hpp"
#include "../src/CMatrixCheck.hpp"
#include "../src/CMatrixConstructor.hpp"
#include "../src/CMatrixGetter.hpp"
#include "../src/CMatrixManipulation.hpp"
#include "../src/CMatrixMath.hpp"
#include "../src/CMatrixOperator.hpp"
#include "../src/CMatrixSetter.hpp"
#include "../src/CMatrixStatic.hpp"
#include "../src/CMatrixStatistics.hpp"
Include dependency graph for CMatrix.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class `cmatrix< T >`

*The main template class that can work with any data type except bool.*

### 8.2.1 Detailed Description

File containing the main template class of the 'cmatrix' library.

#### Author

Manitas Bahri <https://github.com/b-manitas>

#### Date

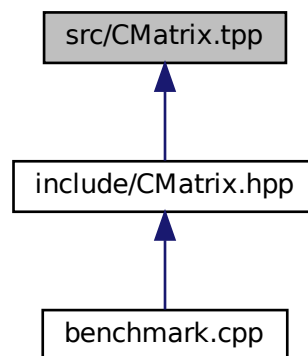
2023 @license MIT License

## 8.3 readme.md File Reference

## 8.4 src/CMatrix.hpp File Reference

This file contains the implementation of general methods of the class.

This graph shows which files directly or indirectly include this file:



### 8.4.1 Detailed Description

This file contains the implementation of general methods of the class.

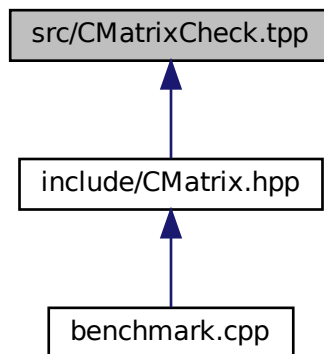
#### See also

[cmatrix](#)

## 8.5 src/CMatrixCheck.hpp File Reference

This file contains the implementation of methods to verify matrix conditions and perform checks before operations to prevent errors.

This graph shows which files directly or indirectly include this file:



### 8.5.1 Detailed Description

This file contains the implementation of methods to verify matrix conditions and perform checks before operations to prevent errors.

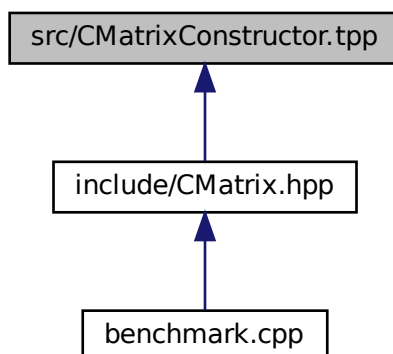
See also

[cmatrix](#)

## 8.6 src/CMatrixConstructor.hpp File Reference

This file contains the implementation of constructors and destructors.

This graph shows which files directly or indirectly include this file:



### 8.6.1 Detailed Description

This file contains the implementation of constructors and destructors.

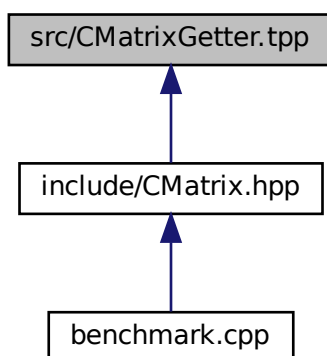
See also

[cmatrix](#)

## 8.7 src/CMatrixGetter.hpp File Reference

This file contains the implementation of methods to retrieve information from the matrix and get its elements.

This graph shows which files directly or indirectly include this file:



### 8.7.1 Detailed Description

This file contains the implementation of methods to retrieve information from the matrix and get its elements.

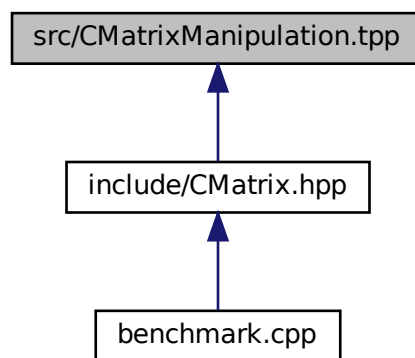
See also

[cmatrix](#)

## 8.8 src/CMatrixManipulation.tpp File Reference

This file contains the implementation of methods to find elements and to perform manipulations on the matrix.

This graph shows which files directly or indirectly include this file:



### 8.8.1 Detailed Description

This file contains the implementation of methods to find elements and to perform manipulations on the matrix.

See also

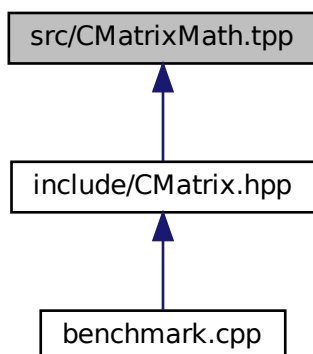
[cmatrix](#)

## 8.9 src/CMatrixMath.tpp File Reference

This file contains the implementation of mathematical functions.



This graph shows which files directly or indirectly include this file:



### 8.9.1 Detailed Description

This file contains the implementation of mathematical functions.

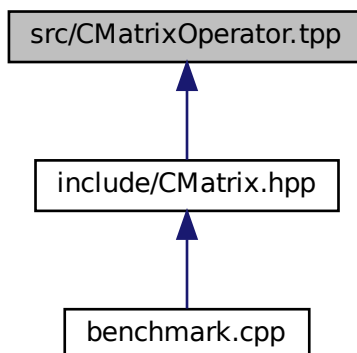
See also

[cmatrix](#)

## 8.10 src/CMatrixOperator.tpp File Reference

This file contains the implementation of operators.

This graph shows which files directly or indirectly include this file:



## Functions

- `template<class T >`  
`cmatrix< T > operator+ (const T &n, const cmatrix< T > &m)`
- `template<class T >`  
`cmatrix< T > operator- (const T &n, const cmatrix< T > &m)`
- `template<class T >`  
`cmatrix< T > operator- (const cmatrix< T > &m)`
- `template<class T >`  
`cmatrix< T > operator* (const T &n, const cmatrix< T > &m)`
- `template<class T >`  
`std::ostream & operator<< (std::ostream &out, const cmatrix< T > &m)`

### 8.10.1 Detailed Description

This file contains the implementation of operators.

See also

[cmatrix](#)

### 8.10.2 Function Documentation

#### 8.10.2.1 operator\*()

```
template<class T >
cmatrix<T> operator* (
    const T & n,
    const cmatrix< T > & m )
```

#### 8.10.2.2 operator+()

```
template<class T >
cmatrix<T> operator+ (
    const T & n,
    const cmatrix< T > & m )
```

#### 8.10.2.3 operator-() [1/2]

```
template<class T >
cmatrix<T> operator- (
    const cmatrix< T > & m )
```

#### 8.10.2.4 operator-() [2/2]

```
template<class T >
cmatrix<T> operator- (
    const T & n,
    const cmatrix< T > & m )
```

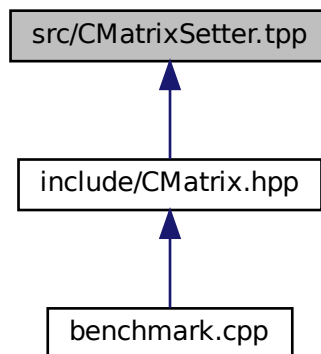
#### 8.10.2.5 operator<<()

```
template<class T >
std::ostream& operator<< (
    std::ostream & out,
    const cmatrix< T > & m )
```

## 8.11 src/CMatrixSetter.hpp File Reference

This file contains the implementation of methods to set values in the matrix.

This graph shows which files directly or indirectly include this file:



### 8.11.1 Detailed Description

This file contains the implementation of methods to set values in the matrix.

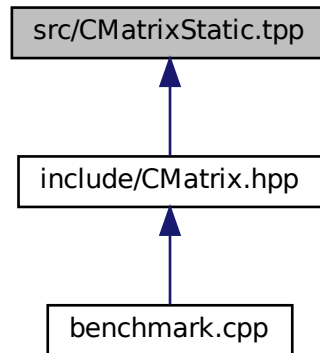
See also

[cmatrix](#)

## 8.12 src/CMatrixStatic.hpp File Reference

This file contains the implementation of static methods of the class.

This graph shows which files directly or indirectly include this file:



### 8.12.1 Detailed Description

This file contains the implementation of static methods of the class.

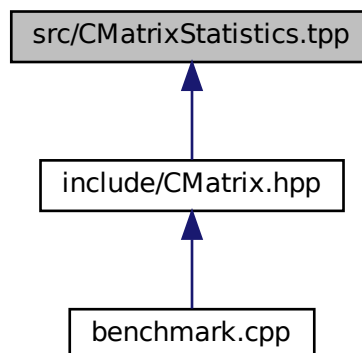
See also

[cmatrix](#)

## 8.13 src/CMatrixStatistics.hpp File Reference

This file contains the implementation of methods to perform statistical operations on the matrix.

This graph shows which files directly or indirectly include this file:



### 8.13.1 Detailed Description

This file contains the implementation of methods to perform statistical operations on the matrix.

See also

[cmatrix](#)



# Index

- `__cast`
    - `CMatrix`, 14
  - `__check_expected_id`
    - `CMatrixCheck`, 23
  - `__check_size`
    - `CMatrixCheck`, 23, 24
  - `__check_valid_col`
    - `CMatrixCheck`, 24
  - `__check_valid_col_id`
    - `CMatrixCheck`, 25
  - `__check_valid_diag`
    - `CMatrixCheck`, 25
  - `__check_valid_row`
    - `CMatrixCheck`, 25
  - `__check_valid_row_id`
    - `CMatrixCheck`, 26
  - `__check_valid_type`
    - `CMatrixCheck`, 26
  - `__map_op_arithmetic`
    - `CMatrixOperator`, 55, 56
  - `__map_op_comparaison_val`
    - `CMatrixOperator`, 56
  - `__mean`
    - `CMatrixStatistics`, 79, 80
  - `__std`
    - `CMatrixStatistics`, 80, 81
  - `__to_string`
    - `CMatrix`, 15
- `~cmatrix`
  - `cmatrix< T >`, 96
- `all`
  - `CMatrixCheck`, 27
- `any`
  - `CMatrixCheck`, 28
- `apply`
  - `CMatrix`, 16
- `bench`
  - `benchmark.cpp`, 99
- `BENCHMARK`
  - `benchmark.cpp`, 99
- `benchmark.cpp`, 99
  - `bench`, 99
  - `BENCHMARK`, 99
  - `BENCHMARK_MAIN`, 99
- `BENCHMARK_MAIN`
  - `benchmark.cpp`, 99
- `cast`
  - `CMatrix`, 16
- `cell`
  - `CMatrixGetter`, 31, 32
- `cells`
  - `CMatrixGetter`, 32, 33
- `clear`
  - `CMatrix`, 17
- `CMatrix`, 13
  - `__cast`, 14
  - `__to_string`, 15
  - `apply`, 16
  - `cast`, 16
  - `clear`, 17
  - `copy`, 17
  - `fill`, 17
  - `map`, 18, 19
  - `print`, 19
  - `to_float`, 19
  - `to_int`, 20
  - `to_string`, 20
  - `to_vector`, 21
- `cmatrix`
  - `cmatrix< T >`, 94–96
- `cmatrix< T >`, 87
  - `~cmatrix`, 96
  - `cmatrix`, 94–96
  - `identity`, 97
  - `matrix`, 98
  - `randfloat`, 97
  - `randint`, 97
  - `to_float`, 97
  - `to_int`, 97
  - `zeros`, 97
- `CMatrixCheck`, 22
  - `__check_expected_id`, 23
  - `__check_size`, 23, 24
  - `__check_valid_col`, 24
  - `__check_valid_col_id`, 25
  - `__check_valid_diag`, 25
  - `__check_valid_row`, 25
  - `__check_valid_row_id`, 26
  - `__check_valid_type`, 26
  - `all`, 27
  - `any`, 28
  - `is_diag`, 28
  - `is_empty`, 29
  - `is_identity`, 29
  - `is_square`, 29
  - `is_symetric`, 29

- is\_triangular\_low, 30
- is\_triangular\_up, 30
- CMatrixGetter, 31
  - cell, 31, 32
  - cells, 32, 33
  - columns, 34, 35
  - columns\_vec, 35
  - diag, 36
  - height, 36
  - rows, 36, 37
  - rows\_vec, 38
  - size, 38
  - slice\_columns, 38
  - slice\_rows, 39
  - transpose, 39
  - width, 40
- CMatrixManipulation, 41
  - concatenate, 41
  - find, 42
  - find\_column, 43
  - find\_row, 44
  - insert\_column, 44
  - insert\_row, 45
  - push\_col\_back, 45
  - push\_col\_front, 46
  - push\_row\_back, 46
  - push\_row\_front, 47
  - remove\_column, 47
  - remove\_row, 48
- CMatrixMath, 49
  - exp, 49
  - log, 49
  - log10, 50
  - log2, 50
  - matmul, 50
  - matpow, 51
  - near, 51, 52
  - nearq, 52, 53
- CMatrixOperator, 54
  - \_\_map\_op\_arithmetic, 55, 56
  - \_\_map\_op\_comparaison\_val, 56
  - operator!=, 57
  - operator<, 64
  - operator<<, 70
  - operator<=, 64
  - operator>, 66
  - operator>=, 67
  - operator\*, 58, 68
  - operator\*=:, 59
  - operator^, 67
  - operator^=, 68
  - operator+, 59, 60, 69
  - operator+=, 60, 61
  - operator-, 61, 62, 69, 70
  - operator-=, 62, 63
  - operator/, 63
  - operator/=:, 63
  - operator=, 65
  - operator==, 66
- CMatrixOperator.hpp
  - operator<<, 107
  - operator\*, 106
  - operator+, 106
  - operator-, 106
- CMatrixSetter, 71
  - set\_cell, 71
  - set\_column, 71
  - set\_diag, 72
  - set\_row, 72
- CMatrixStatic, 75
  - identity, 75
  - is\_matrix, 75
  - merge, 77
  - randfloat, 77
  - randint, 78
  - zeros, 78
- CMatrixStatistics, 79
  - \_\_mean, 79, 80
  - \_\_std, 80, 81
  - max, 81
  - max\_all, 82
  - mean, 82
  - median, 83
  - min, 83
  - min\_all, 84
  - std, 84
  - sum, 85
  - sum\_all, 86
- columns
  - CMatrixGetter, 34, 35
- columns\_vec
  - CMatrixGetter, 35
- concatenate
  - CMatrixManipulation, 41
- copy
  - CMatrix, 17
- diag
  - CMatrixGetter, 36
- exp
  - CMatrixMath, 49
- fill
  - CMatrix, 17
- find
  - CMatrixManipulation, 42
- find\_column
  - CMatrixManipulation, 43
- find\_row
  - CMatrixManipulation, 44
- height
  - CMatrixGetter, 36
- identity
  - cmatrix< T >, 97



- CMatrixStatic, 75
- include/CMatrix.hpp, 100
- insert\_column
  - CMatrixManipulation, 44
- insert\_row
  - CMatrixManipulation, 45
- is\_diag
  - CMatrixCheck, 28
- is\_empty
  - CMatrixCheck, 29
- is\_identity
  - CMatrixCheck, 29
- is\_matrix
  - CMatrixStatic, 75
- is\_square
  - CMatrixCheck, 29
- is\_symetric
  - CMatrixCheck, 29
- is\_triangular\_low
  - CMatrixCheck, 30
- is\_triangular\_up
  - CMatrixCheck, 30
- log
  - CMatrixMath, 49
- log10
  - CMatrixMath, 50
- log2
  - CMatrixMath, 50
- map
  - CMatrix, 18, 19
- matmul
  - CMatrixMath, 50
- matpow
  - CMatrixMath, 51
- matrix
  - cmatrix< T >, 98
- max
  - CMatrixStatistics, 81
- max\_all
  - CMatrixStatistics, 82
- mean
  - CMatrixStatistics, 82
- median
  - CMatrixStatistics, 83
- merge
  - CMatrixStatic, 77
- min
  - CMatrixStatistics, 83
- min\_all
  - CMatrixStatistics, 84
- near
  - CMatrixMath, 51, 52
- nearq
  - CMatrixMath, 52, 53
- operator!=
  - CMatrixOperator, 57
- operator<
  - CMatrixOperator, 64
- operator<<
  - CMatrixOperator, 70
  - CMatrixOperator.tpp, 107
- operator<=
  - CMatrixOperator, 64
- operator>
  - CMatrixOperator, 66
- operator>=
  - CMatrixOperator, 67
- operator\*
  - CMatrixOperator, 58, 68
  - CMatrixOperator.tpp, 106
- operator\*=
  - CMatrixOperator, 59
- operator^
  - CMatrixOperator, 67
- operator^=
  - CMatrixOperator, 68
- operator+
  - CMatrixOperator, 59, 60, 69
  - CMatrixOperator.tpp, 106
- operator+=
  - CMatrixOperator, 60, 61
- operator-
  - CMatrixOperator, 61, 62, 69, 70
  - CMatrixOperator.tpp, 106
- operator-=
  - CMatrixOperator, 62, 63
- operator/
  - CMatrixOperator, 63
- operator/=
  - CMatrixOperator, 63
- operator=
  - CMatrixOperator, 65
- operator==
  - CMatrixOperator, 66
- print
  - CMatrix, 19
- push\_col\_back
  - CMatrixManipulation, 45
- push\_col\_front
  - CMatrixManipulation, 46
- push\_row\_back
  - CMatrixManipulation, 46
- push\_row\_front
  - CMatrixManipulation, 47
- randfloat
  - cmatrix< T >, 97
  - CMatrixStatic, 77
- randint
  - cmatrix< T >, 97
  - CMatrixStatic, 78
- readme.md, 101
- remove\_column

- CMatrixManipulation, [47](#)
- remove\_row
  - CMatrixManipulation, [48](#)
- rows
  - CMatrixGetter, [36](#), [37](#)
- rows\_vec
  - CMatrixGetter, [38](#)
- set\_cell
  - CMatrixSetter, [71](#)
- set\_column
  - CMatrixSetter, [71](#)
- set\_diag
  - CMatrixSetter, [72](#)
- set\_row
  - CMatrixSetter, [72](#)
- size
  - CMatrixGetter, [38](#)
- slice\_columns
  - CMatrixGetter, [38](#)
- slice\_rows
  - CMatrixGetter, [39](#)
- src/CMatrix.hpp, [101](#)
- src/CMatrixCheck.hpp, [102](#)
- src/CMatrixConstructor.hpp, [102](#)
- src/CMatrixGetter.hpp, [103](#)
- src/CMatrixManipulation.hpp, [104](#)
- src/CMatrixMath.hpp, [104](#)
- src/CMatrixOperator.hpp, [105](#)
- src/CMatrixSetter.hpp, [107](#)
- src/CMatrixStatic.hpp, [108](#)
- src/CMatrixStatistics.hpp, [108](#)
- std
  - CMatrixStatistics, [84](#)
- sum
  - CMatrixStatistics, [85](#)
- sum\_all
  - CMatrixStatistics, [86](#)
- to\_float
  - CMatrix, [19](#)
  - cmatrix< T >, [97](#)
- to\_int
  - CMatrix, [20](#)
  - cmatrix< T >, [97](#)
- to\_string
  - CMatrix, [20](#)
- to\_vector
  - CMatrix, [21](#)
- transpose
  - CMatrixGetter, [39](#)
- width
  - CMatrixGetter, [40](#)
- zeros
  - cmatrix< T >, [97](#)
  - CMatrixStatic, [78](#)