

## 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	1
1.4 Hierarchical Structure	2
1.5 Documentation	2
1.6 License	2
<b>2 Deprecated List</b>	<b>3</b>
<b>3 Module Index</b>	<b>5</b>
3.1 Modules	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List	7
<b>5 File Index</b>	<b>9</b>
5.1 File List	9
<b>6 Module Documentation</b>	<b>11</b>
6.1 CMatrix	11
6.1.1 Detailed Description	11
6.1.2 Function Documentation	11
6.1.2.1 apply() [1/2]	12
6.1.2.2 apply() [2/2]	12
6.1.2.3 cast()	12
6.1.2.4 clear()	13
6.1.2.5 copy()	13
6.1.2.6 fill()	13
6.1.2.7 map() [1/2]	13
6.1.2.8 map() [2/2]	15
6.1.2.9 print()	15
6.1.2.10 to_vector()	15
6.2 CMatrixCheck	16
6.2.1 Detailed Description	17
6.2.2 Function Documentation	17
6.2.2.1 all() [1/2]	17
6.2.2.2 all() [2/2]	17
6.2.2.3 any() [1/2]	18
6.2.2.4 any() [2/2]	18
6.2.2.5 check_dim() [1/2]	19
6.2.2.6 check_dim() [2/2]	19
6.2.2.7 check_expected_id() [1/2]	19
6.2.2.8 check_expected_id() [2/2]	20

6.2.2.9 check_valid_col()	20
6.2.2.10 check_valid_col_id()	21
6.2.2.11 check_valid_diag()	21
6.2.2.12 check_valid_row()	21
6.2.2.13 check_valid_row_id()	22
6.2.2.14 check_valid_type()	22
6.2.2.15 is_diag()	22
6.2.2.16 is_empty()	23
6.2.2.17 is_identity()	23
6.2.2.18 is_square()	23
6.2.2.19 is_symetric()	24
6.2.2.20 is_triangular_low()	24
6.2.2.21 is_triangular_up()	24
6.3 CMatrixConstructor	25
6.4 CMatrixGetter	26
6.5 CMatrixManipulation	27
6.5.1 Detailed Description	27
6.5.2 Function Documentation	27
6.5.2.1 find() [1/2]	27
6.5.2.2 find() [2/2]	28
6.5.2.3 find_column() [1/2]	28
6.5.2.4 find_column() [2/2]	29
6.5.2.5 find_row() [1/2]	29
6.5.2.6 find_row() [2/2]	30
6.5.2.7 insert_column()	30
6.5.2.8 insert_row()	31
6.5.2.9 push_col_back()	31
6.5.2.10 push_col_front()	32
6.5.2.11 push_row_back()	32
6.5.2.12 push_row_front()	32
6.5.2.13 remove_column()	33
6.5.2.14 remove_row()	33
6.6 CMatrixOperator	35
6.7 CMatrixSetter	36
6.8 CMatrixStatic	37
6.8.1 Detailed Description	37
6.8.2 Function Documentation	37
6.8.2.1 flatten_vector()	37
6.8.2.2 identity()	37
6.8.2.3 is_matrix()	38
6.8.2.4 randint()	38
6.9 CMatrixStatistics	40

6.9.1 Detailed Description	40
6.9.2 Function Documentation	40
6.9.2.1 max()	40
6.9.2.2 mean()	41
6.9.2.3 median()	41
6.9.2.4 min()	42
6.9.2.5 std()	42
6.9.2.6 sum()	43
<b>7 Class Documentation</b>	<b>45</b>
7.1 cmatrix< T > Class Template Reference	45
7.1.1 Detailed Description	50
7.1.2 Constructor & Destructor Documentation	50
7.1.2.1 cmatrix() [1/5]	50
7.1.2.2 cmatrix() [2/5]	50
7.1.2.3 cmatrix() [3/5]	51
7.1.2.4 cmatrix() [4/5]	51
7.1.2.5 cmatrix() [5/5]	51
7.1.2.6 ~cmatrix()	52
7.1.3 Member Function Documentation	52
7.1.3.1 cell() [1/2]	52
7.1.3.2 cell() [2/2]	53
7.1.3.3 cells() [1/2]	53
7.1.3.4 cells() [2/2]	54
7.1.3.5 columns() [1/2]	54
7.1.3.6 columns() [2/2]	55
7.1.3.7 columns_vec()	55
7.1.3.8 diag()	56
7.1.3.9 dim()	56
7.1.3.10 dim_h()	56
7.1.3.11 dim_v()	56
7.1.3.12 identity()	57
7.1.3.13 operator!=() [1/2]	57
7.1.3.14 operator!=() [2/2]	57
7.1.3.15 operator*() [1/2]	58
7.1.3.16 operator*() [2/2]	58
7.1.3.17 operator*==() [1/2]	58
7.1.3.18 operator*==() [2/2]	59
7.1.3.19 operator+() [1/2]	59
7.1.3.20 operator+() [2/2]	60
7.1.3.21 operator+=() [1/2]	60
7.1.3.22 operator+=() [2/2]	61

7.1.3.23 operator-() [1/2]	62
7.1.3.24 operator-() [2/2]	62
7.1.3.25 operator-=( ) [1/2]	63
7.1.3.26 operator-=( ) [2/2]	63
7.1.3.27 operator/()	63
7.1.3.28 operator/=( )	64
7.1.3.29 operator<()	64
7.1.3.30 operator<=( )	65
7.1.3.31 operator=( ) [1/2]	65
7.1.3.32 operator=( ) [2/2]	65
7.1.3.33 operator==( ) [1/2]	66
7.1.3.34 operator==( ) [2/2]	66
7.1.3.35 operator>()	67
7.1.3.36 operator>=( )	67
7.1.3.37 operator^()	67
7.1.3.38 operator^=( )	68
7.1.3.39 randint()	68
7.1.3.40 rows() [1/2]	69
7.1.3.41 rows() [2/2]	69
7.1.3.42 rows_vec()	69
7.1.3.43 set_cell()	70
7.1.3.44 set_column()	70
7.1.3.45 set_diag()	71
7.1.3.46 set_row()	71
7.1.3.47 transpose()	73
7.1.3.48 zeros() [1/2]	73
7.1.3.49 zeros() [2/2]	73
7.1.4 Friends And Related Function Documentation	74
7.1.4.1 operator<<	74
<b>8 File Documentation</b>	<b>75</b>
8.1 include/CMatrix.hpp File Reference	75
8.1.1 Detailed Description	75
8.2 readme.md File Reference	76
8.3 src/CMatrix.hpp File Reference	76
8.3.1 Detailed Description	76
8.4 src/CMatrixCheck.hpp File Reference	76
8.4.1 Detailed Description	76
8.5 src/CMatrixConstructor.hpp File Reference	76
8.5.1 Detailed Description	76
8.6 src/CMatrixGetter.hpp File Reference	77
8.6.1 Detailed Description	77

---

8.7 src/CMatrixManipulation.hpp File Reference . . . . .	77
8.7.1 Detailed Description . . . . .	77
8.8 src/CMatrixOperator.hpp File Reference . . . . .	77
8.8.1 Detailed Description . . . . .	77
8.8.2 Function Documentation . . . . .	78
8.8.2.1 operator<<() . . . . .	78
8.9 src/CMatrixSetter.hpp File Reference . . . . .	78
8.9.1 Detailed Description . . . . .	78
8.10 src/CMatrixStatic.hpp File Reference . . . . .	78
8.10.1 Detailed Description . . . . .	78
8.11 src/CMatrixStatistics.hpp File Reference . . . . .	78
8.11.1 Detailed Description . . . . .	78
<b>Index</b>	<b>79</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. [Licence](#)

### 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.

### 1.3 Exemple of Usage

Here's an example of how to use CMatrix:

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

## 1.5 Documentation

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

## 1.6 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 . . . . .	11
CMatrixCheck . . . . .	16
CMatrixConstructor . . . . .	25
CMatrixGetter . . . . .	26
CMatrixManipulation . . . . .	27
CMatrixOperator . . . . .	35
CMatrixSetter . . . . .	36
CMatrixStatic . . . . .	37
CMatrixStatistics . . . . .	40



## 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="#">45</a>
------------------------------------	--	--------------------





## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">include/CMatrix.hpp</a>	File containing the main template class of the 'cmatrix' library . . . . .	75
<a href="#">src/CMatrix.hpp</a>	This file contains the implementation of general methods of the class . . . . .	76
<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 . . . . .	76
<a href="#">src/CMatrixConstructor.hpp</a>	This file contains the implementation of constructors and destructors . . . . .	76
<a href="#">src/CMatrixGetter.hpp</a>	This file contains the implementation of methods to retrieve information from the matrix and get its elements . . . . .	77
<a href="#">src/CMatrixManipulation.hpp</a>	This file contains the implementation of methods to find elements and to perform manipulations on the matrix . . . . .	77
<a href="#">src/CMatrixOperator.hpp</a>	This file contains the implementation of operators . . . . .	77
<a href="#">src/CMatrixSetter.hpp</a>	This file contains the implementation of methods to set values in the matrix . . . . .	78
<a href="#">src/CMatrixStatic.hpp</a>	This file contains the implementation of static methods of the class . . . . .	78
<a href="#">src/CMatrixStatistics.hpp</a>	This file contains the implementation of methods to perform statistical operations on the matrix	78



## Chapter 6

# Module Documentation

### 6.1 CMatrix

#### Functions

- 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, size_t *col=nullptr, size_t *row=nullptr)`  
*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, size_t *col=nullptr, size_t *row=nullptr)` 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.*
- 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.*

#### 6.1.1 Detailed Description

#### 6.1.2 Function Documentation

### 6.1.2.1 `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

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

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

```
template<class T >
void cmatrix< T >::apply (
    const std::function< T(T, size_t *, size_t *)> & f,
    size_t * col = nullptr,
    size_t * row = nullptr )
```

Apply a function to each cell of the matrix.

#### Parameters

<i>f</i>	The function to apply. f(T value, size_t *id_col, size_t *id_row) -> T
<i>col</i>	The pointer to the column index. (default: nullptr)
<i>row</i>	The pointer to the row index. (default: nullptr)

### 6.1.2.3 `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>	If the type T is not convertible to the type U.
------------------------------------	---

## 6.1.2.4 clear()

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

Clear the matrix.

## 6.1.2.5 copy()

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

Copy the matrix.

## Returns

`cmatrix<T>` The copied matrix.

## 6.1.2.6 fill()

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

Fill the matrix with a value.

## Parameters

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

## 6.1.2.7 map() [1/2]

```
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. $f(T \text{ value}) \rightarrow T$
----------	---

## Returns

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

6.1.2.8 `map()` [2/2]

```
template<class T >
cmatrix< T > cmatrix< T >::map (
    const std::function< T(T, size_t *, size_t *)> & f,
    size_t * col = nullptr,
    size_t * row = nullptr ) const
```

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

## Parameters

<i>f</i>	The function to apply. $f(T \text{ value}, \text{size\_t} * \text{id\_col}, \text{size\_t} * \text{id\_row}) \rightarrow T$
<i>col</i>	The pointer to the column index. (default: <code>nullptr</code> )
<i>row</i>	The pointer to the row index. (default: <code>nullptr</code> )

## Returns

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

6.1.2.9 `print()`

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

Print the matrix in the standard output.

6.1.2.10 `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

- 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.*
- void `cmatrix< T >::check_dim (const std::tuple< size_t, size_t > &dim)` const  
*Check if dimensions are equals to the dimensions of the matrix.*
- void `cmatrix< T >::check_dim (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.*



## 6.2.1 Detailed Description

## 6.2.2 Function Documentation

### 6.2.2.1 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

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

#### 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.

### 6.2.2.2 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

<i>val</i>	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.

**6.2.2.3 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.

**6.2.2.4 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.

## 6.2.2.5 check\_dim() [1/2]

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

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.
------------------------------	---

## 6.2.2.6 check\_dim() [2/2]

```
template<class T >
void cmatrix< T >::check_dim (
    const std::tuple< size_t, size_t > & dim ) const
```

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

## Parameters

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

## Exceptions

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

## 6.2.2.7 check\_expected\_id() [1/2]

```
template<class T >
void cmatrix< T >::check_expected_id (
    const size_t & n,
    const size_t & expected ) const
```

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.
------------------------------------	---

**6.2.2.8 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
```

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.
------------------------------------	---

**6.2.2.9 check\_valid\_col()**

```
template<class T >
void cmatrix< T >::check_valid_col (
    const std::vector< T > & col ) const
```

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.

### 6.2.2.10 check\_valid\_col\_id()

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

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

#### Parameters

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

#### Exceptions

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

### 6.2.2.11 check\_valid\_diag()

```
template<class T >
void cmatrix< T >::check_valid_diag (
    const std::vector< T > & diag ) const
```

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

#### Parameters

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

#### Exceptions

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

### 6.2.2.12 check\_valid\_row()

```
template<class T >
void cmatrix< T >::check_valid_row (
    const std::vector< T > & row ) const
```

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

#### Parameters

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

**Exceptions**

<code>std::invalid_argument</code>	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.

**6.2.2.13 check\_valid\_row\_id()**

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

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

**Parameters**

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

**Exceptions**

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

**6.2.2.14 check\_valid\_type()**

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

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.
------------------------------------	-------------------------

**6.2.2.15 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 CMatrixConstructor

## 6.4 CMatrixGetter

## 6.5 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.*

### 6.5.1 Detailed Description

### 6.5.2 Function Documentation

#### 6.5.2.1 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).

**6.5.2.2 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.

**6.5.2.3 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.

**6.5.2.4 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.

**6.5.2.5 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. f(std::vector<T> row) -> bool
----------	---

**Returns**

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

**Note**

The empty matrix always return -1.

### 6.5.2.6 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.

### 6.5.2.7 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.

## 6.5.2.8 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.

## 6.5.2.9 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

<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.

### 6.5.2.10 `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

<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.

### 6.5.2.11 `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.

### 6.5.2.12 `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.

## 6.5.2.13 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.

## 6.5.2.14 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.

## 6.6 CMatrixOperator

## 6.7 CMatrixSetter

## 6.8 CMatrixStatic

### Functions

- static bool `cmatrix< T >::is_matrix` (const std::initializer\_list< std::initializer\_list< 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 std::vector< T > `cmatrix< T >::flatten_vector` (const std::vector< std::vector< T >> &vec)  
*Flatten a nested vector.*
- static `cmatrix< int > cmatrix< T >::randint` (const size\_t &dim\_v, const size\_t &dim\_h, const int &min, const int &max, const int &seed=time(nullptr))  
*Generate a random matrix of integers.*
- static `cmatrix< int > cmatrix< T >::identity` (const size\_t &dim)  
*Generate the identity matrix.*

### 6.8.1 Detailed Description

### 6.8.2 Function Documentation

#### 6.8.2.1 flatten\_vector()

```
template<class T >
std::vector< T > cmatrix< T >::flatten_vector (
    const std::vector< std::vector< T >> & vec ) [static]
```

Flatten a nested vector.

#### Parameters

<code>vec</code>	The nested vector to flatten.
------------------	-------------------------------

#### Returns

std::vector<T> The flattened vector.

#### 6.8.2.2 identity()

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

Generate the identity matrix.

**Parameters**

<i>dim</i>	The number of rows and columns.
------------	---------------------------------

**Returns**

`cmatrix<int>` The identity matrix.

**6.8.2.3 is\_matrix()**

```
template<class T >
bool cmatrix< T >::is_matrix (
    const std::initializer_list< std::initializer_list< 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.

**6.8.2.4 randint()**

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

Generate a random matrix of integers.

**Parameters**

<i>dim<sub>h</sub></i>	The number of rows.
<i>dim<sub>v</sub></i>	The number of columns.
<i>min</i>	The minimum value of the matrix.
<i>max</i>	The maximum value of the matrix.
<i>seed</i>	The seed of the random generator. (default: time(nullptr))

**Returns**

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

## 6.9 CMatrixStatistics

### Functions

- `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.
- `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.
- `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.
- `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 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 `>`.



### 6.9.2.2 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.

### 6.9.2.3 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.

**6.9.2.4 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 `<`.

**6.9.2.5 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

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

## Note

The matrix must be of arithmetic type.

## 6.9.2.6 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.
------------------------------------	----------------------------



## 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` ()  
*Construct a new cmatrix object.*
- `cmatrix` (const size\_t &dim\_v, const size\_t &dim\_h)  
*Construct a new cmatrix object.*
- `cmatrix` (const size\_t &dim\_v, const size\_t &dim\_h, 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 > `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 > `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.*
- `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.*
- `size_t dim_h` () const  
*The number of columns of the matrix.*
- `size_t dim_v` () const  
*The number of rows of the matrix.*
- `std::pair< size_t, size_t > dim` () 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.*

- `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.*
- `void check_dim (const std::tuple< size_t, size_t > &dim) const`  
*Check if dimensions are equals to the dimensions of the matrix.*
- `void check_dim (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< T > min (const unsigned int &axis=0) const`  
*Get the minimum value for each row (axis: 0) or column (axis: 1) 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.*
- `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.*
- `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.*

  - 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, size\_t \*col=nullptr, size\_t \*row=nullptr)
- 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, size\_t \*col=nullptr, size\_t \*row=nullptr) 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.*

  - 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< 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.*
- `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.*
- `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 > randint` (const size\_t &dim\_v, const size\_t &dim\_h, const int &min, const int &max, const int &seed)
- `cmatrix< int > zeros` (const size\_t &dim\_h, const size\_t &dim\_v)
- `cmatrix< int > identity` (const size\_t &dim)

## Static Public Member Functions

- static bool `is_matrix` (const std::initializer\_list< std::initializer\_list< 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 std::vector< T > `flatten_vector` (const std::vector< std::vector< T >> &vec)  
*Flatten a nested vector.*
- static `cmatrix< int > randint` (const size\_t &dim\_v, const size\_t &dim\_h, const int &min, const int &max, const int &seed=time(nullptr))  
*Generate a random matrix of integers.*
- static `cmatrix< int > zeros` (const size\_t &dim\_h, const size\_t &dim\_v)  
*Generate a matrix of zeros.*
- static `cmatrix< int > identity` (const size\_t &dim)  
*Generate the identity matrix.*

## Friends

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

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

### 7.1.2 Constructor & Destructor Documentation

#### 7.1.2.1 cmatrix() [1/5]

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

Construct a new cmatrix object.

#### Parameters

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

#### Exceptions

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

#### 7.1.2.2 cmatrix() [2/5]

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

Construct a new cmatrix object.

#### Exceptions

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

**7.1.2.3 `cmatrix()`** [3/5]

```
template<class T >
cmatrix< T >::cmatrix (
    const size_t & dim_v,
    const size_t & dim_h )
```

Construct a new `cmatrix` object.

**Parameters**

<i>dim_v</i>	The number of rows.
<i>dim_h</i>	The number of columns.

**Exceptions**

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

**7.1.2.4 `cmatrix()`** [4/5]

```
template<class T >
cmatrix< T >::cmatrix (
    const size_t & dim_v,
    const size_t & dim_h,
    const T & val )
```

Construct a new `cmatrix` object.

**Parameters**

<i>dim_v</i>	The number of rows.
<i>dim_h</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> .
------------------------------------	------------------------------------

**7.1.2.5 `cmatrix()`** [5/5]

```
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 bool.
------------------------------------	----------------------

#### 7.1.2.6 ~cmatrix()

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

### 7.1.3 Member Function Documentation

#### 7.1.3.1 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

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

7.1.3.2 `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

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

7.1.3.3 `cells()` [1/2]

```
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**

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

**7.1.3.4 cells() [2/2]**

```
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**

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

**7.1.3.5 columns() [1/2]**

```
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**

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

### 7.1.3.6 `columns()` [2/2]

```
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

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

### 7.1.3.7 `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

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

**Deprecated** Use `columns` instead.

#### 7.1.3.8 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.

#### 7.1.3.9 dim()

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

The dimensions of the matrix.

##### Returns

std::pair<size\_t, size\_t> The number of rows and columns.

#### 7.1.3.10 dim\_h()

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

The number of columns of the matrix.

##### Returns

size\_t The number of columns.

#### 7.1.3.11 dim\_v()

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

The number of rows of the matrix.

##### Returns

size\_t The number of rows.



### 7.1.3.12 `identity()`

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

### 7.1.3.13 `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.

### 7.1.3.14 `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.

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

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

The multiplication 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.

**7.1.3.16 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.

**7.1.3.17 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.

**7.1.3.18 `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.

**7.1.3.19 `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.

**7.1.3.20 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.

**7.1.3.21 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.

### 7.1.3.22 `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.

**7.1.3.23 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**

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

**7.1.3.24 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.

### 7.1.3.25 `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.

### 7.1.3.26 `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.

### 7.1.3.27 `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.

**7.1.3.28 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.

**7.1.3.29 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.



**7.1.3.30 `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.

**7.1.3.31 `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.

**7.1.3.32 `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.

**7.1.3.33 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.

**7.1.3.34 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.

**7.1.3.35 `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.

**7.1.3.36 `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.

**7.1.3.37 `operator^()`**

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

The power operator.

**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.
------------------------------------	---------------------------------------

**7.1.3.38 operator^=()**

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

The power assignment operator.

**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.
------------------------------------	---------------------------------------

**7.1.3.39 randint()**

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

**7.1.3.40** `rows()` [1/2]

```
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.
--------------------------------	-------------------------------

**7.1.3.41** `rows()` [2/2]

```
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.
--------------------------------	-------------------------------

**7.1.3.42** `rows_vec()`

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

Get a row of the matrix.

#### Parameters

<i>n</i>	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.

### 7.1.3.43 `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.

### 7.1.3.44 `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.

#### 7.1.3.45 `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.

#### 7.1.3.46 `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

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

7.1.3.47 `transpose()`

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

Get the transpose of the matrix.

## Returns

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

7.1.3.48 `zeros()` [1/2]

```
cmatrix< int > cmatrix< int >::zeros (
    const size_t & dim_h,
    const size_t & dim_v )
```

7.1.3.49 `zeros()` [2/2]

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

Generate a matrix of zeros.

**Parameters**

<i>dim</i> ↔ <i>_h</i>	The number of columns.
<i>dim</i> ↔ <i>_v</i>	The number of rows.

**Returns**

`cmatrix<int>` The matrix of zeros.

**7.1.4 Friends And Related Function Documentation****7.1.4.1 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.

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/CMatrixOperator.hpp](#)
- [src/CMatrixSetter.hpp](#)
- [src/CMatrixStatic.hpp](#)
- [src/CMatrixStatistics.hpp](#)

## Chapter 8

# File Documentation

### 8.1 include/CMatrix.hpp File Reference

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

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

#### Classes

- class [cmatrix< T >](#)

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

#### 8.1.1 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

## 8.2 readme.md File Reference

## 8.3 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:

### 8.3.1 Detailed Description

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

See also

[cmatrix](#)

## 8.4 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:

### 8.4.1 Detailed Description

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

See also

[cmatrix](#)

## 8.5 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:

### 8.5.1 Detailed Description

This file contains the implementation of constructors and destructors.

See also

[cmatrix](#)

## 8.6 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:

### 8.6.1 Detailed Description

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

See also

[cmatrix](#)

## 8.7 src/CMatrixManipulation.hpp 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:

### 8.7.1 Detailed Description

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

See also

[cmatrix](#)

## 8.8 src/CMatrixOperator.hpp File Reference

This file contains the implementation of operators.

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

### Functions

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

### 8.8.1 Detailed Description

This file contains the implementation of operators.

See also

[cmatrix](#)

## 8.8.2 Function Documentation

### 8.8.2.1 `operator<<()`

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

## 8.9 `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:

### 8.9.1 Detailed Description

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

See also

[cmatrix](#)

## 8.10 `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:

### 8.10.1 Detailed Description

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

See also

[cmatrix](#)

## 8.11 `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:

### 8.11.1 Detailed Description

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

See also

[cmatrix](#)

# Index

- `~cmatrix`
  - `cmatrix< T >`, [52](#)
- `all`
  - `CMatrixCheck`, [17](#)
- `any`
  - `CMatrixCheck`, [18](#)
- `apply`
  - `CMatrix`, [11](#), [12](#)
- `cast`
  - `CMatrix`, [12](#)
- `cell`
  - `cmatrix< T >`, [52](#), [53](#)
- `cells`
  - `cmatrix< T >`, [53](#), [54](#)
- `check_dim`
  - `CMatrixCheck`, [18](#), [19](#)
- `check_expected_id`
  - `CMatrixCheck`, [19](#), [20](#)
- `check_valid_col`
  - `CMatrixCheck`, [20](#)
- `check_valid_col_id`
  - `CMatrixCheck`, [20](#)
- `check_valid_diag`
  - `CMatrixCheck`, [21](#)
- `check_valid_row`
  - `CMatrixCheck`, [21](#)
- `check_valid_row_id`
  - `CMatrixCheck`, [22](#)
- `check_valid_type`
  - `CMatrixCheck`, [22](#)
- `clear`
  - `CMatrix`, [13](#)
- `CMatrix`, [11](#)
  - `apply`, [11](#), [12](#)
  - `cast`, [12](#)
  - `clear`, [13](#)
  - `copy`, [13](#)
  - `fill`, [13](#)
  - `map`, [13](#), [15](#)
  - `print`, [15](#)
  - `to_vector`, [15](#)
- `cmatrix`
  - `cmatrix< T >`, [50](#), [51](#)
- `cmatrix< T >`, [45](#)
  - `~cmatrix`, [52](#)
  - `cell`, [52](#), [53](#)
  - `cells`, [53](#), [54](#)
  - `cmatrix`, [50](#), [51](#)
  - `columns`, [54](#), [55](#)
  - `columns_vec`, [55](#)
  - `diag`, [55](#)
  - `dim`, [56](#)
  - `dim_h`, [56](#)
  - `dim_v`, [56](#)
  - `identity`, [56](#)
  - `operator!=`, [57](#)
  - `operator<`, [64](#)
  - `operator<<`, [74](#)
  - `operator<=`, [64](#)
  - `operator>`, [67](#)
  - `operator>=`, [67](#)
  - `operator*`, [57](#), [58](#)
  - `operator*=  
operator^`, [58](#), [59](#)
  - `operator^`, [67](#)
  - `operator^=`, [68](#)
  - `operator+`, [59](#), [60](#)
  - `operator+=`, [60](#)
  - `operator-`, [62](#)
  - `operator-=`, [62](#), [63](#)
  - `operator/`, [63](#)
  - `operator/=`, [64](#)
  - `operator=`, [65](#)
  - `operator==`, [66](#)
  - `randint`, [68](#)
  - `rows`, [68](#), [69](#)
  - `rows_vec`, [69](#)
  - `set_cell`, [70](#)
  - `set_column`, [70](#)
  - `set_diag`, [71](#)
  - `set_row`, [71](#)
  - `transpose`, [73](#)
  - `zeros`, [73](#)
- `CMatrixCheck`, [16](#)
  - `all`, [17](#)
  - `any`, [18](#)
  - `check_dim`, [18](#), [19](#)
  - `check_expected_id`, [19](#), [20](#)
  - `check_valid_col`, [20](#)
  - `check_valid_col_id`, [20](#)
  - `check_valid_diag`, [21](#)
  - `check_valid_row`, [21](#)
  - `check_valid_row_id`, [22](#)
  - `check_valid_type`, [22](#)
  - `is_diag`, [22](#)
  - `is_empty`, [23](#)
  - `is_identity`, [23](#)
  - `is_square`, [23](#)

- is\_symetric, [23](#)
- is\_triangular\_low, [24](#)
- is\_triangular\_up, [24](#)
- CMatrixConstructor, [25](#)
- CMatrixGetter, [26](#)
- CMatrixManipulation, [27](#)
  - find, [27, 28](#)
  - find\_column, [28, 29](#)
  - find\_row, [29](#)
  - insert\_column, [30](#)
  - insert\_row, [30](#)
  - push\_col\_back, [31](#)
  - push\_col\_front, [31](#)
  - push\_row\_back, [32](#)
  - push\_row\_front, [32](#)
  - remove\_column, [33](#)
  - remove\_row, [33](#)
- CMatrixOperator, [35](#)
- CMatrixOperator.hpp
  - operator<<, [78](#)
- CMatrixSetter, [36](#)
- CMatrixStatic, [37](#)
  - flatten\_vector, [37](#)
  - identity, [37](#)
  - is\_matrix, [38](#)
  - randint, [38](#)
- CMatrixStatistics, [40](#)
  - max, [40](#)
  - mean, [40](#)
  - median, [41](#)
  - min, [42](#)
  - std, [42](#)
  - sum, [43](#)
- columns
  - cmatrix< T >, [54, 55](#)
- columns\_vec
  - cmatrix< T >, [55](#)
- copy
  - CMatrix, [13](#)
- diag
  - cmatrix< T >, [55](#)
- dim
  - cmatrix< T >, [56](#)
- dim\_h
  - cmatrix< T >, [56](#)
- dim\_v
  - cmatrix< T >, [56](#)
- fill
  - CMatrix, [13](#)
- find
  - CMatrixManipulation, [27, 28](#)
- find\_column
  - CMatrixManipulation, [28, 29](#)
- find\_row
  - CMatrixManipulation, [29](#)
- flatten\_vector
  - CMatrixStatic, [37](#)
- identity
  - cmatrix< T >, [56](#)
  - CMatrixStatic, [37](#)
- include/CMatrix.hpp, [75](#)
- insert\_column
  - CMatrixManipulation, [30](#)
- insert\_row
  - CMatrixManipulation, [30](#)
- is\_diag
  - CMatrixCheck, [22](#)
- is\_empty
  - CMatrixCheck, [23](#)
- is\_identity
  - CMatrixCheck, [23](#)
- is\_matrix
  - CMatrixStatic, [38](#)
- is\_square
  - CMatrixCheck, [23](#)
- is\_symetric
  - CMatrixCheck, [23](#)
- is\_triangular\_low
  - CMatrixCheck, [24](#)
- is\_triangular\_up
  - CMatrixCheck, [24](#)
- map
  - CMatrix, [13, 15](#)
- max
  - CMatrixStatistics, [40](#)
- mean
  - CMatrixStatistics, [40](#)
- median
  - CMatrixStatistics, [41](#)
- min
  - CMatrixStatistics, [42](#)
- operator!=
  - cmatrix< T >, [57](#)
- operator<
  - cmatrix< T >, [64](#)
- operator<<
  - cmatrix< T >, [74](#)
  - CMatrixOperator.hpp, [78](#)
- operator<=
  - cmatrix< T >, [64](#)
- operator>
  - cmatrix< T >, [67](#)
- operator>=
  - cmatrix< T >, [67](#)
- operator\*
  - cmatrix< T >, [57, 58](#)
- operator\*=
  - cmatrix< T >, [58, 59](#)
- operator^
  - cmatrix< T >, [67](#)
- operator^=
  - cmatrix< T >, [68](#)
- operator+
  - cmatrix< T >, [59, 60](#)



- operator+=
  - cmatrix< T >, 60
- operator-
  - cmatrix< T >, 62
- operator-=
  - cmatrix< T >, 62, 63
- operator/
  - cmatrix< T >, 63
- operator/=
  - cmatrix< T >, 64
- operator=
  - cmatrix< T >, 65
- operator==
  - cmatrix< T >, 66
- print
  - CMatrix, 15
- push\_col\_back
  - CMatrixManipulation, 31
- push\_col\_front
  - CMatrixManipulation, 31
- push\_row\_back
  - CMatrixManipulation, 32
- push\_row\_front
  - CMatrixManipulation, 32
- randint
  - cmatrix< T >, 68
  - CMatrixStatic, 38
- readme.md, 76
- remove\_column
  - CMatrixManipulation, 33
- remove\_row
  - CMatrixManipulation, 33
- rows
  - cmatrix< T >, 68, 69
- rows\_vec
  - cmatrix< T >, 69
- set\_cell
  - cmatrix< T >, 70
- set\_column
  - cmatrix< T >, 70
- set\_diag
  - cmatrix< T >, 71
- set\_row
  - cmatrix< T >, 71
- src/CMatrix.tpp, 76
- src/CMatrixCheck.tpp, 76
- src/CMatrixConstructor.tpp, 76
- src/CMatrixGetter.tpp, 77
- src/CMatrixManipulation.tpp, 77
- src/CMatrixOperator.tpp, 77
- src/CMatrixSetter.tpp, 78
- src/CMatrixStatic.tpp, 78
- src/CMatrixStatistics.tpp, 78
- std
  - CMatrixStatistics, 42
- sum
  - CMatrixStatistics, 43
- to\_vector
  - CMatrix, 15
- transpose
  - cmatrix< T >, 73
- zeros
  - cmatrix< T >, 73