# CMatrix

# 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

## 1.2 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:

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

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# File Index

## 5.1  File List

Here is a list of all files with brief descriptions:

# 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**

| | |
|---|---|
| *f* | 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**

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

| | |
|---|---|
| *U* | The type of the matrix. |

**Returns**

   cmatrix The matrix of another type.

**Exceptions**

| *std::invalid_argument* | 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**

| *val* | 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**

| *f* | The function to apply. f(T value) -> 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**

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

**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**

- 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 &exepectedEnd) 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_dim() [1/2]

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

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

**Parameters**

| *m* | The matrix. |
|-----|-------------|

**Exceptions**

| *std::invalid_argument* | If the dimensions are not equals to the dimensions of the matrix. |
|-------------------------|------------------------------------------------------------------|

#### 6.2.2.2 __check_dim() [2/2]

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

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

**Parameters**

| *dim* | The vertical and horizontal dimensions. |
|-------|------------------------------------------|

**Exceptions**

| *std::invalid_argument* | If the dimensions are not equals to the dimensions of the matrix. |
|-------------------------|------------------------------------------------------------------|

#### 6.2.2.3 __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**

| n | The index to check. |
|---|---|
| expected | The expected index. |

**Exceptions**

| std::invalid_argument | If the index is not the expected index. |
|---|---|

### 6.2.2.4 __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 & exepectedEnd ) const  [private]
```

Check if the index is expected.

**Parameters**

| n | The index to check. |
|---|---|
| expectedBegin | The expected begin index inclusive. |
| exepectedEnd | The expected end index inlusive. |

**Exceptions**

| std::invalid_argument | If the index is not the expected index. |
|---|---|

### 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**

| col | The column to check. |
|---|---|

**Exceptions**

| *std::invalid_argument* | 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**

| *col* | The column index to check. |
|---|---|

**Exceptions**

| *std::invalid_argument* | 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**

| *diag* | The diagonal to check. |
|---|---|

**Exceptions**

| *std::invalid_argument* | 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**

| | |
|---|---|
| *row* | The row to check. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | 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**

| | |
|---|---|
| *row* | The row index to check. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | 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**

| *std::invalid_argument* | 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**

| *f* | 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.

**6.2.2.12 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**

| *val* | 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**

| f | 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**

| val | 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 >::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 >::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.*

- 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.*

- size_t cmatrix< T >::dim_h () const

    *The number of columns of the matrix.*

- size_t cmatrix< T >::dim_v () const

    *The number of rows of the matrix.*

- std::pair< size_t, size_t > cmatrix< T >::dim () 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**

| | |
|---|---|
| *row* | The row of the cell to get. |
| *col* | The column of the cell to get. |

**Returns**

T The reference of the cell.

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | 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**

| | |
|---|---|
| *row* | The row of the cell to get. |
| *col* | The column of the cell to get. |

**Returns**

T The cell.

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the index is out of range. |

### 6.3.2.3 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**

| | |
|---|---|
| *row* | The row of the cell to get. |
| *col* | The column of the cell to get. |

**Returns**

cmatrix<T> The cells of the matrix.

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the index is out of range. |

**6.3.2.4 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**

| | |
|---|---|
| *ids* | The indexes of the cells to get. (row, column) |

**Returns**

cmatrix<T> The cells of the matrix.

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the index is out of range. |

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

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

Get the columns of the matrix.

**Parameters**

| | |
|---|---|
| *ids* | The indexes of the columns to get. |

**Returns**

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

**Exceptions**

| *std::out_of_range* | If the index is out of range. |
|---|---|

**6.3.2.6 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**

| *ids* | The indexes of the columns to get. |
|---|---|

**Returns**

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

**Exceptions**

| *std::out_of_range* | If the index is out of range. |
|---|---|

**6.3.2.7 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**

| *n* | The index of the column to get. |
|---|---|

**Returns**

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

**Exceptions**

| *std::out_of_range* | If the index is out of range. |
| --- | --- |

**[Deprecated](#)** Use `columns` instead.

**6.3.2.8 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.

**6.3.2.9 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.

**6.3.2.10 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.

**6.3.2.11 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.

**6.3.2.12 rows()** [1/2]

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

Get the rows of the matrix.

**Parameters**

| | |
|---|---|
| *ids* | The indexes of the rows to get. |

**Returns**

> cmatrix<T> The rows of the matrix.

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the index is out of range. |

**6.3.2.13 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**

| | |
|---|---|
| *ids* | The indexes of the rows to get. |

**Returns**

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

**Exceptions**

| *std::out_of_range* | If the index is out of range. |
| --- | --- |

**6.3.2.14 rows_vec()**

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

Get a row of the matrix.

**Parameters**

| *n* | The index of the row to get. |
| --- | --- |

**Returns**

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

**Exceptions**

| *std::out_of_range* | If the index is out of range. |
| --- | --- |

**Deprecated** Use `rows` instead.

**6.3.2.15 transpose()**

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

Get the transpose of the matrix.

**Returns**

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

## 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.*

### 6.4.1 Detailed Description

### 6.4.2 Function Documentation

#### 6.4.2.1 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**

| | |
|---|---|
| *f* | 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).

**6.4.2.2 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**

| | |
|---|---|
| *val* | 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.

**6.4.2.3 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**

| | |
|---|---|
| *f* | The condition to satisfy. f(std::vector<T> col) -> bool |

**Returns**

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

**Note**

> The empty matrix always return -1.

**6.4.2.4  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**

| | |
|---|---|
| *val* | 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.

**6.4.2.5  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**

| | |
|---|---|
| *f* | 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.

### 6.4.2.6 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**

| val | 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.

### 6.4.2.7 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**

| pos | The index of the row to insert. |
|-----|--------------------------------|
| val | The value to insert. |

**Exceptions**

| std::out_of_range | If the index is out of range. |
|-------------------|-------------------------------|
| std::invalid_argument | If the size of the vector `val` 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.

### 6.4.2.8 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**

| pos | The index of the column to insert. |
|-----|-----|
| val | The value to insert. |

**Exceptions**

| std::out_of_range | If the index is out of range. |
|-----|-----|
| std::invalid_argument | If the size of the vector `val` 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.

### 6.4.2.9 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**

| val | The column to push. |
|-----|-----|

**Exceptions**

| std::invalid_argument | If the size of the vector `val` 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_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**

| | |
|---|---|
| *val* | The column to push. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the size of the vector `val` 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_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**

| | |
|---|---|
| *val* | The row to push. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the size of the vector `val` 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.

### 6.4.2.12 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**

| *val* | The row to push. |
|---|---|

**Exceptions**

| *std::invalid_argument* | If the size of the vector `val` 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   remove_column()**

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

Remove a column of the matrix.

**Parameters**

| *n* | The index of the column to remove. |
|---|---|

**Exceptions**

| *std::out_of_range* | If the index is out of range. |
|---|---|
| *std::invalid_argument* | If the matrix is empty. |

**6.4.2.14   remove_row()**

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

Remove a row of the matrix.

**Parameters**

| *n* | The index of the row to remove. |
|---|---|

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the index is out of range. |
| *std::invalid_argument* | If the matrix is empty. |

## 6.5 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< 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.*
- 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.*
- 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$^\wedge$= (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.*

### 6.5.1 Detailed Description

### 6.5.2 Function Documentation

#### 6.5.2.1 __map_op_arithmetic()

```
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**

| | |
|---|---|
| *f* | The operator to apply. f(T value, T value) -> T |
| *m* | The matrix to apply. |

**Returns**

cmatrix<T> The result of the operator.

### 6.5.2.2 __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**

| | |
|---|---|
| *f* | The comparison operator to apply. f(T value, T value) -> bool |
| *n* | The number to compare. |

**Returns**

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

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

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

The inequality operator.

**Parameters**

| | |
|---|---|
| *m* | 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.

### 6.5.2.4 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**

| | |
|---|---|
| *val* | The value to compare. |

**Returns**

cmatrix<short unsigned int> The matrix of booleans.

### 6.5.2.5 operator∗() [1/2]

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

The multiplication operator.

**Parameters**

| | |
|---|---|
| *m* | The matrix to multiply. |

**Returns**

cmatrix<T> The product of the matrices.

**Note**

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

### 6.5.2.6 operator∗() [2/2]

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

The multiplication operator.

**Parameters**

| | |
|---|---|
| *n* | The value to multiply. |

**Returns**

cmatrix⟨T⟩ The product of the matrices.

**6.5.2.7 operator∗=() [1/2]**

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

The multiplication assignment operator.

**Parameters**

| | |
|---|---|
| *m* | The matrix to multiply. |

**Returns**

cmatrix⟨T⟩& The product of the matrices.

**Note**

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

**6.5.2.8 operator∗=() [2/2]**

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

The multiplication assignment operator.

**Parameters**

| | |
|---|---|
| *n* | The value to multiply. |

**Returns**

cmatrix⟨T⟩& The product of the matrices.

### 6.5.2.9 operator+() [1/2]

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

The addition operator.

**Parameters**

| | |
|---|---|
| *m* | The matrix to add. |

**Returns**

cmatrix<T> The sum of the matrices.

**Note**

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

### 6.5.2.10 operator+() [2/2]

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

The addition operator.

**Parameters**

| | |
|---|---|
| *n* | The value to add. |

**Returns**

cmatrix<T> The sum of the matrices.

### 6.5.2.11 operator+=() [1/2]

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

The addition assignment operator.

**Parameters**

| | |
|---|---|
| *m* | The matrix to add. |

**Returns**

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

**Note**

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

---

**6.5.2.12 operator+=()** [2/2]

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

The addition assignment operator.

**Parameters**

| | |
|---|---|
| *n* | The value to add. |

**Returns**

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

---

**6.5.2.13 operator-()** [1/2]

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

The subtraction operator.

**Parameters**

| | |
|---|---|
| *m* | The matrix to subtract. |

**Returns**

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

---

**Note**

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

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

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

The subtraction operator.

**Parameters**

| *n* | The value to subtract. |

**Returns**

cmatrix<T> The difference of the matrices.

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

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

The subtraction assignment operator.

**Parameters**

| *m* | The matrix to subtract. |

**Returns**

cmatrix<T>& The difference of the matrices.

**Note**

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

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

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

The subtraction assignment operator.

**Parameters**

| | |
|---|---|
| *n* | The value to subtract. |

**Returns**

cmatrix<T>& The difference of the matrices.

**6.5.2.17 operator/()**

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

The division operator.

**Parameters**

| | |
|---|---|
| *n* | The value to divide. |

**Returns**

cmatrix<T> The quotient of the matrices.

**6.5.2.18 operator/=()**

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

The division assignment operator.

**Parameters**

| | |
|---|---|
| *n* | The value to divide. |

**Returns**

cmatrix<T>& The quotient of the matrices.

### 6.5.2.19 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**

| | |
|---|---|
| *val* | The value to compare. |

**Returns**

cmatrix<short unsigned int> The matrix of booleans.

### 6.5.2.20 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**

| | |
|---|---|
| *val* | The value to compare. |

**Returns**

cmatrix<short unsigned int> The matrix of booleans.

### 6.5.2.21 operator=() [1/2]

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

The assignment operator.

**Parameters**

| | |
|---|---|
| *m* | The matrix to copy. |

**Returns**

cmatrix<T>& The copied matrix.

**Note**

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

### 6.5.2.22 operator=() [2/2]

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

The assignment operator.

**Parameters**

| | |
|---|---|
| *m* | The matrix to copy. |

**Returns**

cmatrix<T>& The copied matrix.

**Note**

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

### 6.5.2.23 operator==() [1/2]

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

The equality operator.

**Parameters**

| | |
|---|---|
| *m* | 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.

**6.5.2.24 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**

| | |
|---|---|
| *val* | The value to compare. |

**Returns**

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

**6.5.2.25 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**

| | |
|---|---|
| *val* | The value to compare. |

**Returns**

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

**6.5.2.26 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**

| | |
|---|---|
| *val* | The value to compare. |

**Returns**

cmatrix<short unsigned int> The matrix of booleans.

### 6.5.2.27 operator$^\wedge$()

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

The power operator.

**Parameters**

| | |
|---|---|
| *m* | The power. Must be a positive integer. |

**Returns**

cmatrix<T> The powered matrix.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the matrix is not a square matrix. |

### 6.5.2.28 operator$^\wedge$=()

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

The power assignment operator.

**Parameters**

| | |
|---|---|
| *m* | The power. Must be a positive integer. |

**Returns**

cmatrix<T>& The powered matrix.

**Exceptions**

| *std::invalid_argument* | If the matrix is not a square matrix. |
| --- | --- |

### 6.5.3 Friends

#### 6.5.3.1 operator∗

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

The multiplication operator.

**Parameters**

| *n* | The value to multiply. |
| --- | --- |
| *m* | The matrix to multiply. |

**Returns**

cmatrix<T> The product of the matrices.

#### 6.5.3.2 operator+

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

The addition operator.

**Parameters**

| *n* | The value to add. |
| --- | --- |
| *m* | The matrix to add. |

**Returns**

cmatrix<T> The sum of the matrices.

### 6.5.3.3 operator- [1/2]

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

The negation operator.

**Parameters**

| | |
|---|---|
| *m* | The matrix to negate. |

**Returns**

cmatrix<T> The negated matrix.

### 6.5.3.4 operator- [2/2]

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

The subtraction operator.

**Parameters**

| | |
|---|---|
| *n* | The value to subtract. |
| *m* | The matrix to subtract. |

**Returns**

cmatrix<T> The difference of the matrices.

### 6.5.3.5 operator<<

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

The output operator.

**Parameters**

| | |
|---|---|
| *out* | The output stream. |
| *m* | The matrix to print. |

**Returns**

std::ostream& The output stream.

## 6.6 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.*

### 6.6.1 Detailed Description

### 6.6.2 Function Documentation

#### 6.6.2.1 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**

| row | The row of the cell to set. |
|-----|-----------------------------|
| col | The column of the cell to set. |
| val | The value to set. |

**Exceptions**

| std::out_of_range | If the index is out of range. |
|-------------------|-------------------------------|

**Note**

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

#### 6.6.2.2 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**

| | |
|---|---|
| *n* | The index of the column to set. |
| *val* | The value to set. |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the index is out of range. |
| *std::invalid_argument* | If the size of the vector `val` 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.

### 6.6.2.3 set_diag()

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

Set the diagonal of the matrix.

**Parameters**

| | |
|---|---|
| *val* | The diagonal to set. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the size of the vector `val` 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.

### 6.6.2.4 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**

| | |
|---|---|
| *n* | The index of the row to set. |
| *val* | The value to set. |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | If the index is out of range. |
| *std::invalid_argument* | If the size of the vector `val` 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.

## 6.7 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.*

### 6.7.1 Detailed Description

### 6.7.2 Function Documentation

#### 6.7.2.1 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**

| | |
|---|---|
| *vec* | The nested vector to flatten. |

**Returns**

std::vector<T> The flattened vector.

#### 6.7.2.2 identity()

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

Generate the identity matrix.

**Parameters**

| | |
|---|---|
| *dim* | The number of rows and columns. |

**Returns**

cmatrix<int> The identity matrix.

### 6.7.2.3 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**

| | |
|---|---|
| *m* | The nested vector to check. |

**Returns**

true If the nested vector is a matrix.

false If the nested vector is not a matrix.

### 6.7.2.4 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**

| | |
|---|---|
| *dim↩ _h* | The number of rows. |
| *dim↩ _v* | The number of columns. |
| *min* | The minimum value of the matrix. |
| *max* | The maximum value of the matrix. |
| *seed* | The seed of the random generator. (default: time(nullptr)) |

**Returns**

cmatrix<int> The random matrix of integers.

## 6.8 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.*

- 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.*

### 6.8.1 Detailed Description

### 6.8.2 Function Documentation

#### 6.8.2.1 __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**

| | |
|---|---|
| *axis* | The axis to get the mean value. 0 for the rows, 1 for the columns. (default: 0) |
| *false_type* | The type of the matrix is not arithmetic. |

**Exceptions**

| *std::invalid_argument* | If the matrix is not arithmetic. |
| --- | --- |

### 6.8.2.2 __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**

| *axis* | The axis to get the mean value. 0 for the rows, 1 for the columns. (default: 0) |
| --- | --- |
| *true_type* | The type of the matrix is arithmetic. |

**Returns**

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

**Exceptions**

| *std::invalid_argument* | If the axis is not 0 or 1. |
| --- | --- |

### 6.8.2.3 __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**

| *axis* | The axis to get the std value. 0 for the rows, 1 for the columns. (default: 0) |
| --- | --- |
| *false_type* | The type of the matrix is not arithmetic. |

**Exceptions**

| *std::invalid_argument* | If the matrix is not arithmetic. |
|---|---|

### 6.8.2.4 __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**

| *axis* | The axis to get the std value. 0 for the rows, 1 for the columns. (default: 0) |
|---|---|
| *true_type* | The type of the matrix is arithmetic. |

**Returns**

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

**Exceptions**

| *std::invalid_argument* | If the axis is not 0 or 1. |
|---|---|

### 6.8.2.5 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**

| *axis* | 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**

| *std::invalid_argument* | If the axis is not 0 or 1. |
| --- | --- |

**Note**

The type of the matrix must implement the operator >.

**6.8.2.6 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**

| *axis* | 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**

| *std::invalid_argument* | If the axis is not 0 or 1. |
| --- | --- |
| *std::invalid_argument* | If the matrix is not arithmetic. |

**Note**

The matrix must be of arithmetic type.

**6.8.2.7 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**

| *axis* | 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**

| *std::invalid_argument* | 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.

**6.8.2.8   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**

| *axis* | 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**

| *std::invalid_argument* | If the axis is not 0 or 1. |
|---|---|

**Note**

The type of the matrix must implement the operator <.

**6.8.2.9   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**

| *axis* | 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**

| *std::invalid_argument* | If the axis is not 0 or 1. |
|---|---|
| *std::invalid_argument* | If the matrix is not arithmetic. |
| *std::invalid_argument* | If the number of elements is less than 2 for the axis. |

**Note**

The matrix must be of arithmetic type.

**6.8.2.10  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**

| *axis* | The axis to get the sum. 0 for the rows, 1 for the columns. (default: 0) |
|---|---|
| *zero* | 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**

| *std::invalid_argument* | 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.*
- 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.*

## Private Member Functions

- 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 &exepectedEnd) 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< 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.*

## 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**

| | |
|---|---|
| T | 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**

| | |
|---|---|
| *m* | The matrix to copy. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the initializer list is not a matrix. |
| *std::invalid_argument* | If the type is bool. |

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

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

Construct a new cmatrix object.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | 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**

| | |
|---|---|
| *dim↩_v* | The number of rows. |
| *dim↩_h* | The number of columns. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the type is bool. |

### 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**

| | |
|---|---|
| *dim↩_v* | The number of rows. |
| *dim↩_h* | The number of columns. |
| *val* | The value to fill the matrix. |

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the type is bool. |

### 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**

| | |
|---|---|
| *m* | The matrix to copy. |

**Template Parameters**

| $U$ | The type of the matrix to copy. |
|-----|--------------------------------|

**Exceptions**

| *std::invalid_argument* | 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 identity()

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

### 7.1.3.2 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 )
```

### 7.1.3.3 zeros() [1/2]

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

### 7.1.3.4 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**

| *dim↩_h* | The number of columns. |
|---|---|
| *dim↩_v* | The number of rows. |

**Returns**

cmatrix⟨int⟩ The matrix of zeros.

### 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.tpp
- src/CMatrixCheck.tpp
- src/CMatrixConstructor.tpp
- src/CMatrixGetter.tpp
- src/CMatrixManipulation.tpp
- src/CMatrixOperator.tpp
- src/CMatrixSetter.tpp
- src/CMatrixStatic.tpp
- src/CMatrixStatistics.tpp

# 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.tpp"
#include "CMatrixCheck.tpp"
#include "CMatrixConstructor.tpp"
#include "CMatrixGetter.tpp"
#include "CMatrixManipulation.tpp"
#include "CMatrixOperator.tpp"
#include "CMatrixSetter.tpp"
#include "CMatrixStatic.tpp"
#include "CMatrixStatistics.tpp"
```
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.tpp 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.tpp 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.tpp 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.tpp 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.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:

### 8.7.1 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.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)

### 8.8.1  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 >
cmatrix<T> operator* (
            const T & n,
            const cmatrix< T > & m )
```

#### 8.8.2.2  operator+()

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

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

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

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

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

**8.8.2.5 operator$<<$()**

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

## 8.9 src/CMatrixSetter.tpp 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.tpp 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.tpp 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