Matrix

Generated by Doxygen 1.8.13

Contents

1 Class Index

1.1 Class List

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

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 const_index_col_iterator< T > Class Template Reference

Template class which implements the column order const_iterator for the Matrix object.

`#include <iterators.h>`

Public Member Functions

- const_index_col_iterator & operator++ ()
- const T & operator* ()
- bool operator== (const const_index_col_iterator &other) const
- bool operator!= (const const_index_col_iterator &other) const
- const_index_col_iterator (const Matrix< T > &m, unsigned r, unsigned c)

Private Attributes

- const Matrix< T > & mat
- unsigned row
- unsigned column

3.1.1 Detailed Description

template<typename T>
class const_index_col_iterator< T >

Template class which implements the column order const_iterator for the Matrix object.

Definition at line 57 of file iterators.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 const_index_col_iterator()

```
template<typename T >
const_index_col_iterator< T >::const_index_col_iterator (
            const Matrix< T > & m,
            unsigned r,
            unsigned c )  [inline]
```

Definition at line 83 of file iterators.h.

```
83                                                                          :
84             mat(m), row(r), column(c) {}
```

3.1.3 Member Function Documentation

3.1.3.1 operator"!=()

```
template<typename T >
bool const_index_col_iterator< T >::operator!= (
            const const_index_col_iterator< T > & other ) const  [inline]
```

Definition at line 79 of file iterators.h.

```
79                                                                      {
80         return row != other.row || column != other.column;
81     }
```

3.1.3.2 operator*()

```
template<typename T >
const T& const_index_col_iterator< T >::operator* ( )  [inline]
```

Definition at line 71 of file iterators.h.

```
71                           {
72         return mat(row, column);
73     }
```

3.1.3.3 operator++()

```
template<typename T >
const_index_col_iterator& const_index_col_iterator< T >::operator++ ( )  [inline]
```

Definition at line 61 of file iterators.h.

```
61                                          {
62         ++row;
63         if (row == mat.getRows()) {
64             row = 0;
65             ++column;
66         }
67
68         return *this;
69     }
```

3.1.3.4 operator==()

```
template<typename T >
bool const_index_col_iterator< T >::operator== (
            const const_index_col_iterator< T > & other ) const  [inline]
```

Definition at line 75 of file iterators.h.

```
75                                                                  {
76         return row == other.row && column == other.column;
77     }
```

3.1.4 Member Data Documentation

3.1.4.1 column

```
template<typename T >
unsigned const_index_col_iterator< T >::column  [private]
```

Definition at line 88 of file iterators.h.

3.1.4.2 mat

```
template<typename T >
const Matrix<T>& const_index_col_iterator< T >::mat  [private]
```

Definition at line 87 of file iterators.h.

3.1.4.3 row

```
template<typename T >
unsigned const_index_col_iterator< T >::row  [private]
```

Definition at line 88 of file iterators.h.

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

- iterators.h

3.2 const_index_row_iterator< T > Class Template Reference

Template class which implements the row order const_iterator for the Matrix object.

```
#include <iterators.h>
```

Public Member Functions

- const_index_row_iterator & operator++ ()
- T & operator* ()
- bool operator== (const const_index_row_iterator &rhs) const
- bool operator!= (const const_index_row_iterator &rhs) const
- const_index_row_iterator (const Matrix< T > &m, unsigned r, unsigned c)

Private Attributes

- const Matrix< T > & mat
- unsigned row
- unsigned col

3.2.1 Detailed Description

template<typename T>
class const_index_row_iterator< T >

Template class which implements the row order const_iterator for the Matrix object.

Definition at line 138 of file iterators.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 const_index_row_iterator()

```
template<typename T >
const_index_row_iterator< T >::const_index_row_iterator (
            const Matrix< T > & m,
            unsigned r,
            unsigned c )  [inline]
```

Definition at line 165 of file iterators.h.

```
165                                                                    :
166             mat(m), row(r), col(c) {}
```

3.2.3 Member Function Documentation

3.2.3.1 operator"!=()

```
template<typename T >
bool const_index_row_iterator< T >::operator!= (
            const const_index_row_iterator< T > & rhs ) const  [inline]
```

Definition at line 160 of file iterators.h.

```
160                                                                 {
161         return row != rhs.row || col != rhs.col;
162     }
```

3.2.3.2 operator∗()

```
template<typename T >
T& const_index_row_iterator< T >::operator* ( )  [inline]
```

Definition at line 153 of file iterators.h.

```
153                     {
154         return mat(row, col);
155     }
```

3.2.3.3 operator++()

```
template<typename T >
const_index_row_iterator& const_index_row_iterator< T >::operator++ ( )  [inline]
```

Definition at line 143 of file iterators.h.

```
143                                         {
144         ++col;
145         if (col == mat.getColumns()) {
146             col = 0;
147             ++row;
148         }
149
150         return *this;
151     }
```

3.2.3.4 operator==()

```
template<typename T >
bool const_index_row_iterator< T >::operator== (
            const const_index_row_iterator< T > & rhs ) const  [inline]
```

Definition at line 157 of file iterators.h.

```
157                                                                 {
158         return row == rhs.row && col == rhs.col;
159     }
```

3.2.4 Member Data Documentation

3.2.4.1 col

```
template<typename T >
unsigned const_index_row_iterator< T >::col  [private]
```

Definition at line 170 of file iterators.h.

3.2.4.2 mat

```
template<typename T >
const Matrix<T>& const_index_row_iterator< T >::mat  [private]
```

Definition at line 169 of file iterators.h.

3.2.4.3 row

```
template<typename T >
unsigned const_index_row_iterator< T >::row  [private]
```

Definition at line 170 of file iterators.h.

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

- iterators.h

3.3 index_col_iterator< T > Class Template Reference

Template class which implements the column order iterator for the Matrix object.

```
#include <iterators.h>
```

Public Member Functions

- index_col_iterator & operator++ ()
- T & operator* ()
- bool operator== (const index_col_iterator &other) const
- bool operator!= (const index_col_iterator &other) const
- index_col_iterator (Matrix< T > &m, unsigned r, unsigned c)

Private Attributes

- Matrix< T > & mat
- unsigned row
- unsigned column

3.3.1 Detailed Description

template<typename T>
class index_col_iterator< T >

Template class which implements the column order iterator for the Matrix object.

Definition at line 17 of file iterators.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 index_col_iterator()

```
template<typename T >
index_col_iterator< T >::index_col_iterator (
            Matrix< T > & m,
            unsigned r,
            unsigned c )  [inline]
```

Definition at line 43 of file iterators.h.

```
43                                                          :
44             mat(m), row(r), column(c) {}
```

3.3.3 Member Function Documentation

3.3.3.1 operator"!=()

```
template<typename T >
bool index_col_iterator< T >::operator!= (
            const index_col_iterator< T > & other ) const  [inline]
```

Definition at line 39 of file iterators.h.

```
39                                                          {
40         return row != other.row || column != other.column;
41     }
```

3.3.3.2 operator*()

```
template<typename T >
T& index_col_iterator< T >::operator* ( )  [inline]
```

Definition at line 31 of file iterators.h.

```
31                     {
32         return mat(row, column);
33     }
```

3.3.3.3 operator++()

```
template<typename T >
index_col_iterator& index_col_iterator< T >::operator++ ( )  [inline]
```

Definition at line 21 of file iterators.h.

```
21                                      {
22         ++row;
23         if (row == mat.getRows()) {
24             row = 0;
25             ++column;
26         }
27
28         return *this;
29     }
```

3.3.3.4 operator==()

```
template<typename T >
bool index_col_iterator< T >::operator== (
             const index_col_iterator< T > & other ) const  [inline]
```

Definition at line 35 of file iterators.h.

```
35                                                              {
36         return row == other.row && column == other.column;
37     }
```

3.3.4 Member Data Documentation

3.3.4.1 column

```
template<typename T >
unsigned index_col_iterator< T >::column  [private]
```

Definition at line 48 of file iterators.h.

3.3.4.2 mat

```
template<typename T >
Matrix<T>& index_col_iterator< T >::mat  [private]
```

Definition at line 47 of file iterators.h.

3.3.4.3 row

```
template<typename T >
unsigned index_col_iterator< T >::row  [private]
```

Definition at line 48 of file iterators.h.

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

- iterators.h

3.4 index_row_iterator< T > Class Template Reference

Template class which implements the row order iterator for the Matrix object.

```
#include <iterators.h>
```

Public Member Functions

- index_row_iterator & operator++ ()
- const T & operator* ()
- bool operator== (const index_row_iterator &rhs) const
- bool operator!= (const index_row_iterator &rhs) const
- index_row_iterator (Matrix< T > &m, unsigned r, unsigned c)

Private Attributes

- const Matrix< T > & mat
- unsigned row
- unsigned col

3.4.1 Detailed Description

template<typename T>
class index_row_iterator< T >

Template class which implements the row order iterator for the Matrix object.

Definition at line 97 of file iterators.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 index_row_iterator()

```
template<typename T >
index_row_iterator< T >::index_row_iterator (
            Matrix< T > & m,
            unsigned r,
            unsigned c )  [inline]
```

Definition at line 124 of file iterators.h.

```
124                                                         :
125             mat(m), row(r), col(c) {}
```

3.4.3 Member Function Documentation

3.4.3.1 operator"!=()

```
template<typename T >
bool index_row_iterator< T >::operator!= (
            const index_row_iterator< T > & rhs ) const  [inline]
```

Definition at line 119 of file iterators.h.

```
119                                                        {
120         return row != rhs.row || col != rhs.col;
121     }
```

3.4.3.2 operator∗()

```
template<typename T >
const T& index_row_iterator< T >::operator* ( )  [inline]
```

Definition at line 112 of file iterators.h.

```
112                           {
113         return mat(row, col);
114     }
```

3.4.3.3 operator++()

```
template<typename T >
index_row_iterator& index_row_iterator< T >::operator++ ( )  [inline]
```

Definition at line 102 of file iterators.h.

```
102                                    {
103         ++col;
104         if (col == mat.getColumns()) {
105             col = 0;
106             ++row;
107         }
108
109         return *this;
110     }
```

3.4.3.4 operator==()

```
template<typename T >
bool index_row_iterator< T >::operator== (
            const index_row_iterator< T > & rhs ) const  [inline]
```

Definition at line 116 of file iterators.h.

```
116                                                     {
117         return row == rhs.row && col == rhs.col;
118     }
```

3.4.4 Member Data Documentation

3.4.4.1 col

```
template<typename T >
unsigned index_row_iterator< T >::col  [private]
```

Definition at line 129 of file iterators.h.

3.4.4.2 mat

```
template<typename T >
const Matrix<T>& index_row_iterator< T >::mat  [private]
```

Definition at line 128 of file iterators.h.

3.4.4.3 row

```
template<typename T >
unsigned index_row_iterator< T >::row  [private]
```

Definition at line 129 of file iterators.h.

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

- iterators.h

3.5 Matrix< T > Class Template Reference

A template class that implements a 2D matrix with some matrix operation which return other Matrix objects with the same shared memory.

```
#include <matrix.h>
```

Public Types

- typedef T type
- typedef std::vector< T >::iterator iterator
- typedef std::vector< T >::const_iterator const_iterator
- typedef index_row_iterator< T > row_iterator
- typedef const_index_row_iterator< T > const_row_iterator
- typedef index_col_iterator< T > column_iterator
- typedef const_index_col_iterator< T > const_column_iterator

Public Member Functions

- Matrix ()

  Default Constructor (Must have) Used when creating an Empty matrix(Useful to array constructors)
- Matrix (const unsigned rows, const unsigned columns)

  Optional Constructor Used for creating a matrix of a certain dimension, filled with zero values(Default constructor of type T)
- Matrix (const unsigned rows, const unsigned columns, const type &val)

  Optional Constructor two Used for creating a matrix of a certain dimension, filled with a value val.
- Matrix (const Matrix< type > &other)

  Copy Constructor (MUST HAVE) It creates a deep copy of a given Matrix.
- Matrix (Matrix< T > &&other)

  Move Constructor (MUST HAVE) It "moves" the content of a Matrix into another(Never called in this project thanks to RVO)
- Matrix< type > & operator= (Matrix< type > &&other)

  Move Assignment (MUST HAVE) It "moves" the content of a Matrix rhs into the left side of the assignment(Never called in this thanks to RVO)
- Matrix & operator= (const Matrix< type > &other)

  Assignment Operator (MUST HAVE) It creates a deep copy of a the rhs Matrix.
- void swap (Matrix< T > &other)

  Swap method (MUST HAVE) It swaps method from a Matrix to another.
- type & operator() (unsigned row, unsigned column)

  Operator() (MUST HAVE) This operator is very important, because it's the extractor of the elements of a given matrix.
- const type & operator() (unsigned row, unsigned column) const

  const Operator() (MUST HAVE) Same as operator(), but the elements extracted with this can only be read and a diagonal Matrix(which is always const by the way) must always use this.
- Matrix subMatrix (const unsigned start_row, const unsigned start_column, const unsigned end_row, const unsigned end_column)

  subMatrix method (REQUESTED) It returns a submatrix of the matrix which called the method(using a protected constructor).
- Matrix transpose () const

  transpose method (REQUESTED) It returns a transpose matrix of the matrix which called the method(using a protected constructor).
- Matrix diagonal () const

  diagonal method (REQUESTED) It returns a "logical" extracted vector which corresponds to the diagonal of the calling Matrix.
- const Matrix< type > diagonalMatrix () const

  diagonalMatrix method (REQUESTED) It returns a "logical" const diagonal Matrix from the calling Matrix(which is a vector or covector).
- ~Matrix ()

  Destructor(MUST HAVE) When a Matrix object goes out of scope, this is automatically called, freeing the memory occupied by that same Matrix.
- unsigned getRows () const

getRows method It return the number of effective rows which the current matrix have

- unsigned getColumns () const

  getColumns method It return the number of effective columns which the current matrix have

- iterator begin ()

  begin method Returns the first iterator used to iterate over the whole vector object

- iterator end ()

  end method Returns the last iterator used to iterate over the whole vector object

- const_iterator begin () const

  begin const method Returns the first iterator used to iterate over the whole vector object, but the element that the iterator points to is read-only

- const_iterator end () const

  end const method Returns the last iterator used to iterate over the whole vector object

- row_iterator row_begin (unsigned i)

  row_begin method Returns the first iterator used to iterate over a single row given as a parameter

- row_iterator row_end (unsigned i)

  row_end method Returns the last iterator used to iterate over a single row given as a parameter

- const_row_iterator row_begin (unsigned i) const

  row_begin const method Returns the first iterator used to iterate over a single row given as a parameter, but the element that is pointed by the iterator is immutable

- const_row_iterator row_end (unsigned i) const

  row_end const method Returns the last iterator used to iterate over a single row given as a parameter

- row_iterator row_begin ()

  row_begin method Returns the first iterator used to iterate over the current considered Matrix by rows

- row_iterator row_end ()

  row_end method Returns the last iterator used to iterate over the current considered Matrix by rows

- const_row_iterator row_begin () const

  row_begin const method Returns the first iterator used to iterate over the current considered Matrix by rows, but the element pointed cannot be modified(const)

- const_row_iterator row_end () const

  row_end const method Returns the last iterator used to iterate over the current considered Matrix by rows, but the element pointed cannot be modified(const)

- column_iterator col_begin (unsigned i)

  col_begin method Returns the first iterator used to iterate over a single column

- column_iterator col_end (unsigned i)

  col_end method Returns the last iterator used to iterate over a single column

- const_column_iterator col_begin (unsigned i) const

  col_begin const method Returns the first iterator used to iterate over a single column, whose element pointed cannot be modified

- column_iterator col_end (unsigned i) const

  col_end const method Returns the last iterator used to iterate over a single column

- column_iterator col_begin ()

  col_begin method Returns the first iterator to the first element of current Matrix, used to iterate by column

- column_iterator col_end ()

  col_end method Returns the iterator to the end of current Matrix, used to iterate by column

- const_column_iterator col_begin () const

  col_begin const method Returns the first iterator to the first element of current Matrix, used to iterate by column, that cannot modify the elements

- const_column_iterator col_end () const

  col_end const method Returns the last iterator current Matrix, used to iterate by column

Protected Member Functions

- Matrix (const unsigned rows, const unsigned columns, const unsigned eff_rows, const unsigned eff_columns, const unsigned start_row, const unsigned start_column, const bool transp, const bool diag, const std::shared_ptr< std::vector< type >> pter)
- Matrix (const Matrix< type > &vec, const bool diagmatr, const bool from_diag)

Protected Attributes

- std::shared_ptr< std::vector< type > > pter
- bool transp
- bool diag
- bool diagmatr
- bool from_diag
- unsigned columns
- unsigned rows
- unsigned start_row
- unsigned start_column
- unsigned effective_rows
- unsigned effective_columns
- const type zero = type()

3.5.1 Detailed Description

template<typename T>
class Matrix< T >

A template class that implements a 2D matrix with some matrix operation which return other Matrix objects with the same shared memory.

Definition at line 22 of file matrix.h.

3.5.2 Member Typedef Documentation

3.5.2.1 column_iterator

```
template<typename T>
typedef index_col_iterator<T> Matrix< T >::column_iterator
```

Definition at line 32 of file matrix.h.

3.5.2.2 const_column_iterator

```
template<typename T>
typedef const_index_col_iterator<T> Matrix< T >::const_column_iterator
```

Definition at line 33 of file matrix.h.

3.5.2.3 const_iterator

```
template<typename T>
typedef std::vector<T>::const_iterator Matrix< T >::const_iterator
```

Definition at line 27 of file matrix.h.

3.5.2.4 const_row_iterator

```
template<typename T>
typedef const_index_row_iterator<T> Matrix< T >::const_row_iterator
```

Definition at line 30 of file matrix.h.

3.5.2.5 iterator

```
template<typename T>
typedef std::vector<T>::iterator Matrix< T >::iterator
```

Definition at line 26 of file matrix.h.

3.5.2.6 row_iterator

```
template<typename T>
typedef index_row_iterator<T> Matrix< T >::row_iterator
```

Definition at line 29 of file matrix.h.

3.5.2.7 type

```
template<typename T>
typedef T Matrix< T >::type
```

Definition at line 25 of file matrix.h.

3.5.3 Constructor & Destructor Documentation

3.5.3.1 Matrix() [1/7]

```
template<typename T>
Matrix< T >::Matrix ( )  [inline]
```

Default Constructor (Must have) Used when creating an Empty matrix(Useful to array constructors)

Definition at line 39 of file matrix.h.

```
39 : columns(0), rows(0), start_row(0), start_column(0),
      transp(false), pter(nullptr), diag(false), diagmatr(false),
      from_diag(false){}
```

3.5.3.2 Matrix() [2/7]

```
template<typename T>
Matrix< T >::Matrix (
            const unsigned rows,
            const unsigned columns )  [inline], [explicit]
```

Optional Constructor Used for creating a matrix of a certain dimension, filled with zero values(Default constructor of type T)

Definition at line 45 of file matrix.h.

```
45                                                                  {
46         this->columns = columns;
47         this->rows = rows;
48         effective_rows = rows;
49         effective_columns = columns;
50         start_row = 0;
51         start_column = 0;
52         diagmatr = false;
53         transp = false;
54         diag = false;
55         pter = std::make_shared<std::vector<T>>(columns * rows);
56         for (type c : *pter)
57             c = type();
58     }
```

3.5.3.3 Matrix() [3/7]

```
template<typename T>
Matrix< T >::Matrix (
            const unsigned rows,
            const unsigned columns,
            const type & val )  [inline], [explicit]
```

Optional Constructor two Used for creating a matrix of a certain dimension, filled with a value val.

Parameters

rows	number of rows of the matrix
columns	number of columns of the matrix
val	value to fill the Matrix

Definition at line 67 of file matrix.h.

```
67                                                                                  {
68          this->columns = columns;
69          this->rows = rows;
70          effective_rows = rows;
71          effective_columns = columns;
72          start_row = 0;
73          start_column = 0;
74          transp = false;
75          diag = false;
76          diagmatr = false;
77          pter = std::make_shared<std::vector<T>>(columns * rows);
78          for (unsigned i = 0; i < (columns * rows); i++)
79              pter->operator[](i) = val;
80      }
```

3.5.3.4 Matrix() [4/7]

```
template<typename T>
Matrix< T >::Matrix (
              const Matrix< type > & other )  [inline]
```

Copy Constructor (MUST HAVE) It creates a deep copy of a given Matrix.

Parameters

other	lvalue reference to a Matrix

Definition at line 88 of file matrix.h.

```
88                                      {
89          columns = other.columns;
90          rows = other.rows;
91          effective_rows = other.effective_rows;
92          effective_columns = other.effective_columns;
93          start_row = other.start_row;
94          start_column = other.start_column;
95          transp = other.transp;
96          diag = other.diag;
97          diagmatr = other.diagmatr;
98          pter = std::make_shared<std::vector<T>>(columns * rows);
99          for (unsigned i = 0; i < (columns * rows); i++)
100              pter->operator[](i) = other.pter->operator[](i);
101          std::cout << "COPY CONSTRUCTOR INVOKED" <<std::endl;
102     }
```

3.5.3.5 Matrix() [5/7]

```
template<typename T>
Matrix< T >::Matrix (
              Matrix< T > && other )  [inline]
```

Move Constructor (MUST HAVE) It "moves" the content of a Matrix into another(Never called in this project thanks to RVO)

Parameters

other	rvalue reference to a Matrix

Definition at line 109 of file matrix.h.

```
109                             {
110         std::cout << "MOVE CONSTUCTOR INVOKED" << std::endl;
111         columns = other.columns;
112         rows = other.rows;
113         effective_rows = other.effective_rows;
114         effective_columns = other.effective_columns;
115         start_row = other.start_row;
116         start_column = other.start_column;
117         diag = other.diag;
118         from_diag = other.from_diag;
119         diagmatr = other.diagmatr;
120         pter = other.pter;  //maybe private method to do this
121         other.pter = nullptr; //same problem as above, but maybe with same class type there is no need
122     }
```

3.5.3.6 ~Matrix()

```
template<typename T>
Matrix< T >::~Matrix ( )  [inline]
```

Destructor(MUST HAVE) When a Matrix object goes out of scope, this is automatically called, freeing the memory occupied by that same Matrix.

Definition at line 293 of file matrix.h.

```
293             {
294         columns = 0;
295         rows = 0;
296         start_row = 0;
297         start_column = 0;
298         effective_columns = 0;
299         effective_rows = 0;
300         std::vector<type>().swap(*pter);
301     }
```

3.5.3.7 Matrix() [6/7]

```
template<typename T>
Matrix< T >::Matrix (
             const unsigned rows,
             const unsigned columns,
             const unsigned eff_rows,
             const unsigned eff_columns,
             const unsigned start_row,
             const unsigned start_column,
             const bool transp,
             const bool diag,
             const std::shared_ptr< std::vector< type >> pter )  [inline], [protected]
```

Definition at line 473 of file matrix.h.

```
473
                                {
474         this->rows = rows;
475         this->columns = columns;
476         this->effective_rows = eff_rows;
477         this->effective_columns = eff_columns;
478         this->start_row = start_row;
479         this->start_column = start_column;
480         this->transp = transp;
481         this->diag = diag;
482         this->pter = pter;
483         this->diagmatr = false;
484         this->from_diag = false;
485     }
```

3.5.3.8 Matrix() [7/7]

```
template<typename T>
Matrix< T >::Matrix (
              const Matrix< type > & vec,
              const bool diagmatr,
              const bool from_diag )  [inline], [protected]
```

Definition at line 488 of file matrix.h.

```
488                                                                                 {
489         this->diagmatr = diagmatr;
490         rows = vec.rows;
491         columns = vec.columns;
492         effective_rows = std::max(vec.effective_rows, vec.
      effective_columns);
493         effective_columns = std::max(vec.effective_rows, vec.
      effective_columns);
494         transp = false;
495         diag = false;
496         start_row = vec.start_row;
497         start_column = vec.start_column;
498         pter = vec.pter;
499         this->from_diag = from_diag;
500     }
```

3.5.4 Member Function Documentation

3.5.4.1 begin() [1/2]

```
template<typename T>
iterator Matrix< T >::begin ( )  [inline]
```

begin method Returns the first iterator used to iterate over the whole vector object

Returns

iterator to the first element contained in the vector

Definition at line 327 of file matrix.h.

```
327 { return pter->begin();}
```

3.5.4.2 begin() [2/2]

```
template<typename T>
const_iterator Matrix< T >::begin ( ) const  [inline]
```

begin const method Returns the first iterator used to iterate over the whole vector object, but the element that the iterator points to is read-only

Returns

const iterator to the first element contained in the vector

Definition at line 341 of file matrix.h.

```
341 { return pter->begin(); }
```

3.5.4.3 col_begin() [1/4]

```
template<typename T>
column_iterator Matrix< T >::col_begin (
            unsigned i )  [inline]
```

col_begin method Returns the first iterator used to iterate over a single column

Parameters

i	column that needs to be iterated

Returns

column_iterator of the first element of the column

Definition at line 416 of file matrix.h.

```
416 { return column_iterator(*this, 0, i); }
```

3.5.4.4 col_begin() [2/4]

```
template<typename T>
const_column_iterator Matrix< T >::col_begin (
            unsigned i ) const  [inline]
```

col_begin const method Returns the first iterator used to iterate over a single column, whose element pointed cannot be modified

Parameters

i	column that needs to be iterated

Returns

const_column_iterator of the first element of the column

Definition at line 432 of file matrix.h.

```
432 { return column_iterator(*this, 0, i); }
```

3.5.4.5 col_begin() [3/4]

```
template<typename T>
column_iterator Matrix< T >::col_begin ( )  [inline]
```

col_begin method Returns the first iterator to the first element of current Matrix, used to iterate by column

Returns

column_iterator of the first element of the current Matrix

Definition at line 447 of file matrix.h.

```
447 {return column_iterator(*this, 0, 0); }
```

3.5.4.6 col_begin() [4/4]

```
template<typename T>
const_column_iterator Matrix< T >::col_begin ( ) const  [inline]
```

col_begin const method Returns the first iterator to the first element of current Matrix, used to iterate by column, that cannot modify the elements

Returns

const_column_iterator of the first element of the current Matrix

Definition at line 461 of file matrix.h.

```
461 {return column_iterator(*this, 0, 0); }
```

3.5.4.7 col_end() [1/4]

```
template<typename T>
column_iterator Matrix< T >::col_end (
            unsigned i )  [inline]
```

col_end method Returns the last iterator used to iterate over a single column

Parameters

i	column that needs to be iterated

Returns

column_iterator representing the logic end of the column

Definition at line 424 of file matrix.h.

```
424 { return column_iterator(*this, 0, i + 1); }
```

3.5.4.8 col_end() [2/4]

```
template<typename T>
column_iterator Matrix< T >::col_end (
            unsigned i ) const  [inline]
```

col_end const method Returns the last iterator used to iterate over a single column

Parameters

i	column that needs to be iterated

Returns

const_column_iterator of the logic end of the column

Definition at line 440 of file matrix.h.

```
440 { return column_iterator(*this, 0, i + 1); }
```

3.5.4.9 col_end() [3/4]

```
template<typename T>
column_iterator Matrix< T >::col_end ( )  [inline]
```

col_end method Returns the iterator to the end of current Matrix, used to iterate by column

Returns

column_iterator of the logic end of the current Matrix

Definition at line 454 of file matrix.h.

```
454 {return column_iterator(*this, 0, effective_columns); }
```

3.5.4.10 col_end() [4/4]

```
template<typename T>
const_column_iterator Matrix< T >::col_end ( ) const  [inline]
```

col_end const method Returns the last iterator current Matrix, used to iterate by column

Returns

const_column_iterator of logic end of the current Matrix

Definition at line 468 of file matrix.h.

```
468 {return column_iterator(*this, 0, effective_columns); }
```

3.5.4.11 diagonal()

```
template<typename T>
Matrix Matrix< T >::diagonal ( ) const  [inline]
```

diagonal method (REQUESTED) It returns a "logical" extracted vector which corresponds to the diagonal of the calling Matrix.

Returns

a Matrix which is a logical built diagonal vector of the starting matrix

Definition at line 272 of file matrix.h.

```
272                             {
273         if(!transp)
274             return Matrix<type>(rows, columns, std::min(
      effective_rows, effective_columns), 1, start_row,
      start_column , false, true, pter);
275         else
276             return Matrix<type>(columns, rows, std::min(
      effective_rows, effective_columns), 1, start_row,
      start_column , false, true, pter);
277     }
```

3.5.4.12 diagonalMatrix()

```
template<typename T>
const Matrix<type> Matrix< T >::diagonalMatrix ( ) const  [inline]
```

diagonalMatrix method (REQUESTED) It returns a "logical" const diagonal Matrix from the calling Matrix(which is a vector or covector).

Returns

a Matrix which is a logical built diagonal matrix of the starting vector/covector

Definition at line 284 of file matrix.h.

```
284                                             {
285         assert(effective_columns == 1 || effective_rows == 1);
286         return Matrix<type>(*this, true, diag);
287     }
```

3.5.4.13 end() [1/2]

```
template<typename T>
iterator Matrix< T >::end ( )  [inline]
```

end method Returns the last iterator used to iterate over the whole vector object

Returns

iterator that represent the end(logic) of the vector

Definition at line 334 of file matrix.h.

```
334 { return pter->end(); }
```

3.5.4.14 end() [2/2]

```
template<typename T>
const_iterator Matrix< T >::end ( ) const  [inline]
```

end const method Returns the last iterator used to iterate over the whole vector object

Returns

const iterator to the end(logic) of the vector

Definition at line 348 of file matrix.h.

```
348 { return pter->end(); }
```

3.5.4.15 getColumns()

```
template<typename T>
unsigned Matrix< T >::getColumns ( ) const  [inline]
```

getColumns method It return the number of effective columns which the current matrix have

Returns

effective columns of the matrix

Definition at line 318 of file matrix.h.

```
318                                {
319         return effective_columns;
320     }
```

3.5.4.16 getRows()

```
template<typename T>
unsigned Matrix< T >::getRows ( ) const  [inline]
```

getRows method It return the number of effective rows which the current matrix have

Returns

effective rows of the matrix

Definition at line 309 of file matrix.h.

```
309                             {
310         return effective_rows;
311     }
```

3.5.4.17 operator()() [1/2]

```
template<typename T>
type& Matrix< T >::operator() (
             unsigned row,
             unsigned column )  [inline]
```

Operator() (MUST HAVE) This operator is very important, because it's the extractor of the elements of a given matrix.

Any methods that wants to access a matrix element must use this. Depending on the "type" of matrix we want to access its elements from (given by some flags), this operator will behave differently(diagmatrix no because is always constant) The elements taken with this method can be read and overwritten

Parameters

row	row of the element that needs to be taken
column	column of the element that needs to be taken

Returns

lvalue reference of the retrieved element

Definition at line 191 of file matrix.h.

```
191                                                 {
192         if((diag == true) && !transp){
193             assert(column == 0);
194             return pter->operator[]((row + start_row) * (columns) + (row +
      start_column));
195         }
196         else if((diag == true) && (transp == true)){
197             assert(row == 0);
198             return pter->operator[]((column + start_column) * (
      rows) + (column + start_row));
199         }
200         else if((!diag && (transp == true)))
```

```
201             return pter->operator[]((column + start_column) * (
    rows) + (row + start_row));
202         else
203             return pter->operator[]((row+start_row) * (columns) + (column +
    start_column));
204     }
```

3.5.4.18 operator()() [2/2]

```
template<typename T>
const type& Matrix< T >::operator() (
            unsigned row,
            unsigned column ) const  [inline]
```

const Operator() (MUST HAVE) Same as operator(), but the elements extracted with this can only be read and a diagonal Matrix(which is always const by the way) must always use this.

Parameters

row	row of the element that needs to be taken
column	column of the element that needs to be taken

Returns

const lvalue reference of the retrieved element

Definition at line 213 of file matrix.h.

```
213                                                                 {
214         if(diagmatr == true){
215             if(row != column)
216                 return zero;
217             else{
218                 if(from_diag)
219                     return pter->operator[]((row + start_row) * (
    columns) + (row + start_column));
220                 else
221                     return pter->operator[](row + (start_row *
    columns + start_column));
222             }
223         }
224         else if((diag == true) && !(transp)){
225             assert(column == 0);
226             return pter->operator[]((row + start_row) * (columns) + (row +
    start_column));
227         }
228         else if((diag == true) && (transp == true)){
229             return pter->operator[]((column + start_column) * (
    rows) + column + start_row);
230         }
231         else if((!diag && (transp == true)))
232             return pter->operator[]((column + start_column) * (
    rows) + (row + start_row));
233         else
234             return pter->operator[]((row + start_row) * (columns) + column +
    start_column);
235     }
```

3.5.4.19 operator=() [1/2]

```
template<typename T>
Matrix<type>& Matrix< T >::operator= (
            Matrix< type > && other )  [inline]
```

Move Assignment (MUST HAVE) It "moves" the content of a Matrix rhs into the left side of the assignment(Never called in this thanks to RVO)

Parameters

other	rvalue reference to the rhs Matrix

Definition at line 129 of file matrix.h.

```
129                                                 {
130         std::cout<<"MOVE ASSIGNMENT INVOKED" << std::endl;
131         columns = other.columns;
132         rows = other.rows;
133         effective_rows = other.effective_rows;
134         effective_columns = other.effective_columns;
135         start_row = other.start_row;
136         start_column = other.start_column;
137         transp = other.transp;
138         diag = other.diag;
139         diagmatr = other.diagmatr;
140         from_diag = other.from_diag;
141         pter = other.pter;  //maybe private method to do this
142         other.pter = nullptr; //same problem as above
143     }
```

3.5.4.20 operator=() [2/2]

```
template<typename T>
Matrix& Matrix< T >::operator= (
            const Matrix< type > & other )  [inline]
```

Assignment Operator (MUST HAVE) It creates a deep copy of a the rhs Matrix.

Parameters

other	lvalue reference to the rhs Matrix

Definition at line 150 of file matrix.h.

```
150                                                 {
151         std::cout<<"NORMAL ASSIGNMENT INVOKED" << std::endl;
152         if (this != &other){
153             Matrix<T> tmp(other);
154             this->swap(tmp);
155         }
156         return *this;
157     }
```

3.5.4.21 row_begin() [1/4]

```
template<typename T>
row_iterator Matrix< T >::row_begin (
            unsigned i )  [inline]
```

row_begin method Returns the first iterator used to iterate over a single row given as a parameter

Parameters

i	row which the iteration will be perfomed on

Returns

iterator to the first element contained in the vector

Definition at line 356 of file matrix.h.

```
356 { return row_iterator(*this, i, 0); }
```

3.5.4.22 row_begin() [2/4]

```
template<typename T>
const_row_iterator Matrix< T >::row_begin (
            unsigned i ) const  [inline]
```

row_begin const method Returns the first iterator used to iterate over a single row given as a parameter, but the element that is pointed by the iterator is immutable

Parameters

i	row which the iteration will be perfomed on

Returns

iterator to the first element contained in the vector

Definition at line 372 of file matrix.h.

```
372 { return row_iterator(*this, i, 0); }
```

3.5.4.23 row_begin() [3/4]

```
template<typename T>
row_iterator Matrix< T >::row_begin ( )  [inline]
```

row_begin method Returns the first iterator used to iterate over the current considered Matrix by rows

Returns

iterator to the first element of the current Matrix

Definition at line 387 of file matrix.h.

```
387 { return row_iterator(*this, 0, 0); }
```

3.5.4.24 row_begin() [4/4]

```
template<typename T>
const_row_iterator Matrix< T >::row_begin ( ) const  [inline]
```

row_begin const method Returns the first iterator used to iterate over the current considered Matrix by rows, but the element pointed cannot be modified(const)

Returns

const_row_iterator to the first element of the current Matrix

Definition at line 401 of file matrix.h.

```
401 { return row_iterator(*this, 0, 0); }
```

3.5.4.25 row_end() [1/4]

```
template<typename T>
row_iterator Matrix< T >::row_end (
            unsigned i )  [inline]
```

row_end method Returns the last iterator used to iterate over a single row given as a parameter

Parameters

i	row which the iteration will be perfomed on

Returns

iterator that represent the end(logic) of the row

Definition at line 364 of file matrix.h.

```
364 { return row_iterator(*this, i + 1, 0); }
```

3.5.4.26 row_end() [2/4]

```
template<typename T>
const_row_iterator Matrix< T >::row_end (
            unsigned i ) const  [inline]
```

row_end const method Returns the last iterator used to iterate over a single row given as a parameter

Parameters

i	row which the iteration will be perfomed on

Returns

iterator to the end(logic) of the row

Definition at line 380 of file matrix.h.

```
380 { return row_iterator(*this, i + 1, 0); }
```

3.5.4.27 row_end() [3/4]

```
template<typename T>
row_iterator Matrix< T >::row_end ( )  [inline]
```

row_end method Returns the last iterator used to iterate over the current considered Matrix by rows

Returns

iterator to the end(logic) of the current Matrix

Definition at line 394 of file matrix.h.

```
394 { return row_iterator(*this, effective_rows, 0); }
```

3.5.4.28 row_end() [4/4]

```
template<typename T>
const_row_iterator Matrix< T >::row_end ( ) const  [inline]
```

row_end const method Returns the last iterator used to iterate over the current considered Matrix by rows, but the element pointed cannot be modified(const)

Returns

const_row_iterator that represent the logic end of the current Matrix

Definition at line 408 of file matrix.h.

```
408 { return row_iterator(*this, effective_rows, 0); }
```

3.5.4.29 subMatrix()

```
template<typename T>
Matrix Matrix< T >::subMatrix (
            const unsigned start_row,
            const unsigned start_column,
            const unsigned end_row,
            const unsigned end_column )  [inline]
```

subMatrix method (REQUESTED) It returns a submatrix of the matrix which called the method(using a protected constructor).

Parameters

start_row	index of the row from which the submatrix starts
start_column	index of the column from which the submatrix starts
end_row	index of the row to which the submatrix ends
end_column	index of the column to which the submatrix ends

Returns

a Matrix which is a logical subMatrix of the calling one

Definition at line 246 of file matrix.h.

```
246
                    {
247         const unsigned new_eff_rows = end_row - start_row + 1;
248         const unsigned new_eff_columns = end_column - start_column + 1;
249         return Matrix<type>(rows, columns, new_eff_rows, new_eff_columns, (this->
      start_row + start_row) ,(this->start_column + start_column), transp,
      diag, pter);
250     }
```

3.5.4.30 swap()

```
template<typename T>
void Matrix< T >::swap (
            Matrix< T > & other )  [inline]
```

Swap method (MUST HAVE) It swaps method from a Matrix to another.

Parameters

other	lvalue reference to a Matrix

Definition at line 164 of file matrix.h.

```
164                             {
165         std::swap(other.pter, this->pter);
166         std::swap(other.columns, this->columns);
167         std::swap(other.rows, this->rows);
168         std::swap(other.effective_columns, this->effective_columns);
```

```
169         std::swap(other.effective_rows, this->effective_rows);
170         std::swap(other.pter, this->pter);
171         std::swap(other.start_column, this->start_column);
172         std::swap(other.start_row, this->start_row);
173         std::swap(other.transp, this->transp);
174         std::swap(other.transp, this->diag);
175         std::swap(other.diagmatr, this->diagmatr);
176         std::swap(other.from_diag, this->from_diag);
177     }
```

3.5.4.31 transpose()

```
template<typename T>
Matrix Matrix< T >::transpose ( ) const  [inline]
```

transpose method (REQUESTED) It returns a transpose matrix of the matrix which called the method(using a protected constructor).

Returns

a Matrix which is a logical tranpose Matrix of the calling one

Definition at line 257 of file matrix.h.

```
257                             {
258         const unsigned new_rows = effective_columns;
259         const unsigned new_columns = effective_rows;
260         const unsigned new_start_row = start_column;
261         const unsigned new_start_column = start_row;
262         const bool new_transp = !transp;
263 
264         return Matrix<type>(columns, rows, new_rows, new_columns, new_start_row,
      new_start_column, new_transp,diag, pter);
265     }
```

3.5.5 Member Data Documentation

3.5.5.1 columns

```
template<typename T>
unsigned Matrix< T >::columns  [protected]
```

Definition at line 505 of file matrix.h.

3.5.5.2 diag

```
template<typename T>
bool Matrix< T >::diag  [protected]
```

Definition at line 504 of file matrix.h.

3.5.5.3 diagmatr

```
template<typename T>
bool Matrix< T >::diagmatr  [protected]
```

Definition at line 504 of file matrix.h.

3.5.5.4 effective_columns

```
template<typename T>
unsigned Matrix< T >::effective_columns  [protected]
```

Definition at line 506 of file matrix.h.

3.5.5.5 effective_rows

```
template<typename T>
unsigned Matrix< T >::effective_rows  [protected]
```

Definition at line 506 of file matrix.h.

3.5.5.6 from_diag

```
template<typename T>
bool Matrix< T >::from_diag  [protected]
```

Definition at line 504 of file matrix.h.

3.5.5.7 pter

```
template<typename T>
std::shared_ptr<std::vector<type> > Matrix< T >::pter  [protected]
```

Definition at line 503 of file matrix.h.

3.5.5.8 rows

```
template<typename T>
unsigned Matrix< T >::rows  [protected]
```

Definition at line 505 of file matrix.h.

3.5.5.9 start_column

```
template<typename T>
unsigned Matrix< T >::start_column  [protected]
```

Definition at line 506 of file matrix.h.

3.5.5.10 start_row

```
template<typename T>
unsigned Matrix< T >::start_row  [protected]
```

Definition at line 506 of file matrix.h.

3.5.5.11 transp

```
template<typename T>
bool Matrix< T >::transp  [protected]
```

Definition at line 504 of file matrix.h.

3.5.5.12 zero

```
template<typename T>
const type Matrix< T >::zero = type()  [protected]
```

Definition at line 507 of file matrix.h.

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

- matrix.h

4 File Documentation

4.1 iterators.h File Reference

Declaration and definition of the iterators needed to iterate in any given order(row or column) over a Matrix object.

```
#include "matrix_forward.h"
```

Classes

- class index_col_iterator< T >

  Template class which implements the column order iterator for the Matrix object.
- class const_index_col_iterator< T >

  Template class which implements the column order const_iterator for the Matrix object.
- class index_row_iterator< T >

  Template class which implements the row order iterator for the Matrix object.
- class const_index_row_iterator< T >

  Template class which implements the row order const_iterator for the Matrix object.

4.1.1 Detailed Description

Declaration and definition of the iterators needed to iterate in any given order(row or column) over a Matrix object.

4.2 main.cpp File Reference

```
#include "matrix.h"
```

Functions

- int main ()

4.2.1 Function Documentation

4.2.1.1 main()

```
int main ( )
```

Definition at line 4 of file main.cpp.

```
4           {
5      Matrix<int> A(4, 5), B(A);
6      std::cout << A;
7      std::cout << "COPYYYYYYYY" << std::endl;
8      std::cout << B;
9      Matrix<int> C(4,5,6);
10      std::cout << "GUARDA MAMMA COL SEI DIOCANEEEEEEE" << std::endl;
11      std::cout << C;
12      Matrix<int> D(C);
13      std::cout << "COPYYYYYYYYO ANCHE QUELLA COL 6, DIO VENTISEI" << std::endl;
14      std::cout << D;
15
16
17      std::cout << "\n\n\n";
18      std::cout << "PROVA ITERATORE COLONNA: " << std::endl;
19
20      for(int r = 0; r != 4; r++){
21          for(int c = 0; c != 5; c++){
22              D(r, c) = r + c;
23          }
24      }
25      std::cout << D;
26
```

```
27      //iterazione sulla matrice per colonna
28      int i = 0;
29      int col = 1;
30      for(auto iter = D.col_begin(0); iter != D.col_end(4); ++iter){
31          if(i == 4 || i == 0){
32              std::cout << "\nColonna " << col << ": ";
33              i=0;
34              col++;
35          }
36          std::cout << *iter << " ";
37          i++;
38      }
39
40      std::cout << "\n";
41
42      auto iter1 = D.col_begin(1);
43      auto iter2 = D.col_begin(2);
44
45      for(int i = 0; i < 4; i++){
46          ++iter1;
47      }
48
49      if(iter1 == iter2){
50          std::cout << "equivalence between iterators test passed" << std::endl;
51      }
52
53      std::cout << "\n\n\n";
54
55      i= 0;
56      int rowa = 1;
57      //prova iteratore riga
58      std::cout << "PROVA ITERATORE RIGA: " << std::endl;
59      for(auto iter = D.row_begin(0); iter != D.row_end(3); ++iter){
60          if(i == 5 || i == 0){
61              std::cout << "\nRiga " << rowa << ": ";
62              i=0;
63              rowa++;
64          }
65          std::cout << *iter << " " ;
66          i++;
67
68      }
69
70      std::cout << "\nMATRICE CON OSTREAM \n"<< D;
71
72      auto S = D.subMatrix(2,1,3,4);
73      std::cout << "\n\n" << S;
74
75
76      std::cout << "\n\n\n" << std::endl;
77
78      std::cout << D << std::endl;
79
80      std::cout << "\n\n\n" << std::endl;
81
82      auto E = D.transpose();
83      std::cout << E << std::endl;
84
85      auto F = E.transpose();
86      std::cout << "\n\n\n" << std::endl;
87      std::cout << F << std::endl;
88
89      auto G = F.subMatrix(1,1,3,3);
90      std::cout << "\n\n\n" << std::endl;
91
92      std::cout << G;
93
94      std::cout << "matrice dritta\n";
95
96      Matrix<int> FF (4,5);
97
98      for(int r = 0; r != 4; r++){
99          for(int c = 0; c != 5; c++){
100             FF(r, c) = r + c;
101         }
102     }
103     std::cout << FF << std::endl;
104
105     std::cout << "trans"<< std::endl;
106
107     auto LL =  FF.transpose() ;
108
109     std::cout << LL << std::endl;
110
111     std::cout << " sub\n" << std::endl;
112
113     auto GG = LL.subMatrix(0,1,3,3);
```

```
114
115     std::cout << GG << "\n";
116
117     auto DI = GG.transpose();
118
119     std::cout << "trasposta\n" <<DI << std::endl;
120
121     auto DG = GG.diagonal();
122
123     std::cout <<"Diagonalley\n" << DG;
124
125     auto DGT = DG.transpose();
126     std::cout <<"Diagonalley al contrario\n" << DGT;
127
128     auto DG2 = LL.diagonal();
129     std::cout << LL;
130     std::cout << "Diagonally su transposta \n" << DG2;
131
132     auto DG3 = DG2.transpose();
133     std::cout<< "DIO SMANDRAPPINO CANE " << DG3 << std::endl;
134
135     auto DG6 = DG2.diagonalMatrix();
136     std::cout << DG6;
137
138     Matrix<int> VE1(5,1,3), VE2(1,5,3);
139     std::cout << "Vettore\n" << VE1;
140     auto DGMV = VE1.diagonalMatrix();
141     std::cout << "Covettore\n" << VE2;
142     auto DGMV2 = VE2.diagonalMatrix();
143     std::cout << "DIAGMATRIX Vettore\n "<<DGMV;
144     std::cout << "DIAGMATRIX Covettore\n" << DGMV2;
145     std::cout << "SOTTOMATRICE DI PARTENZA\n" << GG;
146     auto SMD = GG.subMatrix(0, 0, 3, 0);
147     std::cout << "SOTTOMATRICE VETTORE\n" << SMD;
148     auto DMVS = SMD.diagonalMatrix();
149     std::cout << "DIAGONALMATRIX DI UN VETTORE SOTTOMATRICE\n" <<DMVS;
150     std::cout   << "Check transpose method" <<   std::endl;
151     auto R = C.transpose();
152     std::cout << R;
153     C(1,2) = 5;
154     std::cout<<"POSIZIONE 1,2 MATRICE NORMALE = 5" << std::endl;
155     std::cout<<"STAMPO MATRICE NORMALE" << std::endl;
156     std::cout << C;
157     std::cout<<"STAMPO MATRICE TRASPOSTA" << std::endl;
158     std::cout << R;
159     R(1,2) = 3;
160     std::cout<<"POSIZIONE 1,2 MATRICE TRASPOSTA = 3" << std::endl;
161     std::cout<<"STAMPO MATRICE NORMALE" << std::endl;
162     std::cout << C;
163     std::cout<<"STAMPO MATRICE TRASPOSTA" << std::endl;
164     std::cout << R;
165     auto H = R.transpose();
166     std::cout<<"STAMPO MATRICE TRASPOSTA TRANSPOSTA DIO CANTASTICO" << std::endl;
167     std::cout << H;
168     std::cout<<"TEST SOTTOMATRICE" << std::endl;
169     auto SC = C.subMatrix(1,1,3,4);
170     std::cout<< "MATRICE NORMALE" <<std::endl;
171     std::cout << C;
172     std::cout<< "MATRICE SUB" << std::endl;
173     std::cout << SC;
174     auto GS = SC.transpose();
175     std::cout<< "MATRICE TRANSPOSTA" << std::endl;
176     std::cout << GS;
177     std::cout<< "PROVO A CAMBIARE UN ELEMENTO, DOVREBBE MODIFICARE TUTTE LE MATRICI" << std::endl;
178     GS(0, 0) = 0;
179     std::cout<<"TUTTE LE MATRICI DOVREBBERO AVERE UNO ZERO" << std::endl;
180     std::cout << GS;
181     std::cout << SC;
182     std::cout << C;
183     std::cout << "NON CENE DI NEGRI IN ITALIA CON MATRICI FUNXIONANTI" << std::endl;
184
185 }
```

4.3 matrix.h File Reference

Library of a 2d matrix with methods like requested in the assignment.

```
#include "iterators.h"
#include <ostream>
#include <vector>
```

```
#include <iterator>
#include <memory>
#include <iostream>
#include <cassert>
```

Classes

- class Matrix< T >

  A template class that implements a 2D matrix with some matrix operation which return other Matrix objects with the same shared memory.

Functions

- template<typename T >
  std::ostream & operator<< (std::ostream &os, const Matrix< T > &ma)

  Overload of stream operator that permits printing a Matrix object.

4.3.1 Detailed Description

Library of a 2d matrix with methods like requested in the assignment.

4.3.2 Function Documentation

4.3.2.1 operator<<()

```
template<typename T >
std::ostream& operator<< (
            std::ostream & os,
            const Matrix< T > & ma )
```

Overload of stream operator that permits printing a Matrix object.

Parameters

os	output stream
ma	Matrix to stamp

Returns

lvalue reference to output stream

Definition at line 519 of file matrix.h.

```
519                                                     {
520     for (unsigned r = 0; r < ma.getRows(); r++){
```

```
521         for (unsigned c = 0; c < ma.getColumns(); c++){
522             os << "[" << ma(r, c) << "] ";
523         }
524         os << std::endl;
525     }
526     return os;
527 }
```

4.4 matrix_forward.h File Reference

Forward declaration needed for using file iterator.h.

Classes

- class Matrix< T >

  A template class that implements a 2D matrix with some matrix operation which return other Matrix objects with the same shared memory.

4.4.1 Detailed Description

Forward declaration needed for using file iterator.h.

Index