Matrix

Generated by Doxygen 1.8.13

Contents

1 README

#TODO Allora, la situazione è questa: Noi dobbiamo riuscire a produrre delle "viste" della matrice, ovvero la memoria viene condivisa tra tutte le matrici ma i dati vengono visti dall'utente in modo diverso (matrici, vettori ecc...). Questo non può essere fatto manipolando lo shared_ptd, in quanto mi punta un unico oggetto e non posso usare un raw pointer ad esempio per iterare sugli elementi del vettore (male). L'unico modo per farlo è usare diversi tipi di iteratori per la matrice... In che senso? Mettiamo il caso abbiamo una matrice (4,5). Il metodo transpose mi torna una matrice (5,4) con gli stessi elementi della prima, l'unica cosa che cambia è che tipo di operatore uso per il begin e l'end(invertito rispetto alla matrice normale). Per fare questo però, dobbiamo creare più classi di matrici, in quanto ogni iteratore in base al tipo di matrice avrà comportamenti diversi. In una submatrix ad esempio, begin() sarà un index_row_iterator. In una matrice normale invece uno standard.Spero di essere stato il più chiaro possibile.

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Class Documentation

4.1 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

4.1.1 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.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 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) {}
```

4.1.3 Member Function Documentation

4.1.3.1 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     }
```

4.1.3.2 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     }
```

4.1.3.3 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     }
```

4.1.3.4 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     }
```

4.1.4 Member Data Documentation

4.1.4.1 column

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

Definition at line 88 of file iterators.h.

4.1.4.2 mat

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

Definition at line 87 of file iterators.h.

4.1.4.3 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

4.2 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++ ()
- const 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

4.2.1 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.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 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) {}
```

4.2.3 Member Function Documentation

4.2.3.1 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     }
```

4.2.3.2 operator∗()

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

Definition at line 153 of file iterators.h.

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

4.2.3.3 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     }
```

4.2.3.4 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     }
```

4.2.4 Member Data Documentation

4.2.4.1 col

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

Definition at line 170 of file iterators.h.

4.2.4.2 mat

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

Definition at line 169 of file iterators.h.

4.2.4.3 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

4.3 course Struct Reference

Public Member Functions

- course ()
- course (unsigned int cr, std::string n)
- bool operator> (const course c) const
- unsigned int getCredits () const

Public Attributes

- unsigned int credits
- std::string name

4.3.1 Detailed Description

Definition at line 361 of file main.cpp.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 course() [1/2]

```
course::course ( )  [inline]
```

Definition at line 364 of file main.cpp.

```
364 : credits(0), name("def") {}
```

4.3.2.2 course() [2/2]

```
course::course (
            unsigned int cr,
            std::string n )  [inline]
```

Definition at line 365 of file main.cpp.

```
365 : credits(cr), name(n) {}
```

4.3.3 Member Function Documentation

4.3.3.1 getCredits()

```
unsigned int course::getCredits ( ) const  [inline]
```

Definition at line 369 of file main.cpp.

```
369                                 {
370       return credits;
371     }
```

4.3.3.2 operator>()

```
bool course::operator> (
            const course c ) const  [inline]
```

Definition at line 366 of file main.cpp.

```
366                                         {
367       return ( credits > c.credits );
368     }
```

4.3.4 Member Data Documentation

4.3.4.1 credits

```
unsigned int course::credits
```

Definition at line 362 of file main.cpp.

4.3.4.2 name

```
std::string course::name
```

Definition at line 363 of file main.cpp.

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

- main.cpp

4.4 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

4.4.1 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.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 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) {}
```

4.4.3 Member Function Documentation

4.4.3.1 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     }
```

4.4.3.2 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     }
```

4.4.3.3 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     }
```

4.4.3.4 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     }
```

4.4.4 Member Data Documentation

4.4.4.1 column

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

Definition at line 48 of file iterators.h.

4.4.4.2 mat

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

Definition at line 47 of file iterators.h.

4.4.4.3 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

4.5 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++ ()
- 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

- matrix< T > & mat
- unsigned row
- unsigned col

4.5.1 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.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 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) {}
```

4.5.3 Member Function Documentation

4.5.3.1 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     }
```

4.5.3.2 operator∗()

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

Definition at line 112 of file iterators.h.

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

4.5.3.3 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     }
```

4.5.3.4 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     }
```

4.5.4 Member Data Documentation

4.5.4.1 col

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

Definition at line 129 of file iterators.h.

4.5.4.2 mat

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

Definition at line 128 of file iterators.h.

4.5.4.3 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

4.6 matrix< T > Class Template Reference

```
#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).
- const matrix subMatrix (const unsigned start_row, const unsigned start_column, const unsigned end_row, const unsigned end_column) const

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

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

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

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

  diagonal const method (REQUESTED) It returns a "logical" extracted const 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.The vector contents will be deleted in case it goes out of scope (handled by shared_ptr)
- 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

- const_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

Private 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 unsigned rows, const unsigned columns, const unsigned effective_rows, const unsigned effective_columns, const unsigned start_row, const unsigned start_column, const bool diagmatr, const bool from_diag, const bool from_subcovector, const std::shared_ptr< std::vector< type >> pter)

Private Attributes

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

4.6.1 Detailed Description

template<typename T>
class matrix< T >

Definition at line 22 of file matrix.h.

4.6.2 Member Typedef Documentation

4.6.2.1 column_iterator

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

Definition at line 31 of file matrix.h.

4.6.2.2 const_column_iterator

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

Definition at line 32 of file matrix.h.

4.6.2.3 const_iterator

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

Definition at line 26 of file matrix.h.

4.6.2.4 const_row_iterator

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

Definition at line 29 of file matrix.h.

4.6.2.5 iterator

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

Definition at line 25 of file matrix.h.

4.6.2.6 row_iterator

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

Definition at line 28 of file matrix.h.

4.6.2.7 type

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

Definition at line 24 of file matrix.h.

4.6.3 Constructor & Destructor Documentation

4.6.3.1 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 38 of file matrix.h.

```
38 : columns(0), rows(0), start_row(0), start_column(0),
      effective_columns(0), effective_rows(0), transp(false),
      pter(nullptr), diag(false), diagmatr(false), from_diag(false),
      from_subcovector(false){}
```

4.6.3.2 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 44 of file matrix.h.

```
44                                                                  {
45         this->columns = columns;
46         this->rows = rows;
47         effective_rows = rows;
48         effective_columns = columns;
49         start_row = 0;
50         start_column = 0;
51         diagmatr = false;
52         transp = false;
53         diag = false;
54         from_diag = false;
55         from_subcovector = false;
56         pter = std::make_shared<std::vector<T>>(columns * rows);
57         for (type c : *pter)
58             c = type();
59     }
```

4.6.3.3 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 68 of file matrix.h.

```
68                                                                                  {
69          this->columns = columns;
70          this->rows = rows;
71          effective_rows = rows;
72          effective_columns = columns;
73          start_row = 0;
74          start_column = 0;
75          transp = false;
76          diag = false;
77          diagmatr = false;
78          from_diag = false;
79          from_subcovector = false;
80          pter = std::make_shared<std::vector<T>>(columns * rows);
81          for(int i = 0; i <(columns * rows); i++){
82              pter->operator[](i) = val;
83          }
84      }
```

4.6.3.4 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 92 of file matrix.h.

```
92                                      {
93          columns = other.columns;
94          rows = other.rows;
95          effective_rows = other.effective_rows;
96          effective_columns = other.effective_columns;
97          start_row = other.start_row;
98          start_column = other.start_column;
99          transp = false;
```

```
100         diag = false;
101         diagmatr = false;
102         from_diag = false;
103         from_subcovector = false;
104         pter = std::make_shared<std::vector<T>>(columns * rows);
105         for (unsigned r = 0; r < getRows(); r++){
106             for (unsigned c = 0; c < getColumns(); c++)
107                 this->operator()(r, c) = other(r, c);
108         }
109     }
```

4.6.3.5 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 116 of file matrix.h.

```
116                             {
117         columns = other.columns;
118         rows = other.rows;
119         effective_rows = other.effective_rows;
120         effective_columns = other.effective_columns;
121         start_row = other.start_row;
122         start_column = other.start_column;
123         diag = other.diag;
124         from_diag = other.from_diag;
125         diagmatr = other.diagmatr;
126         from_subcovector = other.from_subcovector;
127         pter = other.pter;
128         other.pter = nullptr;
129     }
```

4.6.3.6 ~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.The vector contents will be deleted in case it goes out of scope (handled by shared↩ _ptr)

Definition at line 351 of file matrix.h.

```
351             {
352         columns = 0;
353         rows = 0;
354         start_row = 0;
355         start_column = 0;
356         effective_columns = 0;
357         effective_rows = 0;
358     }
```

4.6.3.7 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], [private]
```

Definition at line 530 of file matrix.h.

```
530
                                {
531         this->rows = rows;
532         this->columns = columns;
533         this->effective_rows = eff_rows;
534         this->effective_columns = eff_columns;
535         this->start_row = start_row;
536         this->start_column = start_column;
537         this->transp = transp;
538         this->diag = diag;
539         this->pter = pter;
540         this->diagmatr = false;
541         this->from_diag = false;
542         this->from_subcovector = false;
543     }
```

4.6.3.8 matrix() [7/7]

```
template<typename T>
matrix< T >::matrix (
              const unsigned rows,
              const unsigned columns,
              const unsigned effective_rows,
              const unsigned effective_columns,
              const unsigned start_row,
              const unsigned start_column,
              const bool diagmatr,
              const bool from_diag,
              const bool from_subcovector,
              const std::shared_ptr< std::vector< type >> pter )  [inline], [private]
```

Definition at line 546 of file matrix.h.

```
546
                                                                                {
547         this->diagmatr = diagmatr;
548         this->rows = rows;
549         this->columns = columns;
550         this->effective_rows = std::max(effective_rows,
      effective_columns);
551         this->effective_columns = std::max(effective_rows,
      effective_columns);
552         this->transp = false;
553         this->diag = false;
554         this->start_row = start_row;
555         this->start_column = start_column;
556         this->pter = pter;
557         this->from_diag = from_diag;
558         this->from_subcovector = from_subcovector;
559
560     }
```

4.6.4 Member Function Documentation

4.6.4.1 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 384 of file matrix.h.

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

4.6.4.2 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 398 of file matrix.h.

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

4.6.4.3 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 473 of file matrix.h.

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

4.6.4.4 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 489 of file matrix.h.

```
489 { return const_column_iterator(*this, 0, i); }
```

4.6.4.5 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 504 of file matrix.h.

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

4.6.4.6 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 518 of file matrix.h.

```
518 {return const_column_iterator(*this, 0, 0); }
```

4.6.4.7 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 481 of file matrix.h.

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

4.6.4.8 col_end() [2/4]

```
template<typename T>
const_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 497 of file matrix.h.

```
497 { return const_column_iterator(*this, 0, i + 1); }
```

4.6.4.9 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 511 of file matrix.h.

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

4.6.4.10 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 525 of file matrix.h.

```
525 {return const_column_iterator(*this, 0, effective_columns); }
```

4.6.4.11 diagonal() [1/2]

```
template<typename T>
matrix matrix< T >::diagonal ( )  [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 310 of file matrix.h.

```
310                     {
311         if(!transp)
312             return matrix<type>(rows, columns, std::min(
      effective_rows, effective_columns), 1, start_row,
      start_column , false, true, pter);
313         else
314             return matrix<type>(columns, rows, std::min(
      effective_rows, effective_columns), 1, start_row,
      start_column , false, true, pter);
315     }
```

4.6.4.12 diagonal() [2/2]

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

diagonal const method (REQUESTED) It returns a "logical" extracted const 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 322 of file matrix.h.

```
322                               {
323         if(!transp)
324             return matrix<type>(rows, columns, std::min(
      effective_rows, effective_columns), 1, start_row,
      start_column , false, true, pter);
325         else
326             return matrix<type>(columns, rows, std::min(
      effective_rows, effective_columns), 1, start_row,
      start_column , false, true, pter);
327     }
```

4.6.4.13 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).

ATTENTION!!!!!!!! This method(alongside every method with const matrix as a return type) is kind of tricky. If the variable that takes the object(the lhs of an assignment or the copy constructed on) is a const matrix<type> then no problem, but if it is a auto type or matrix<type> then the RVO doesen't care about it and it will move THE MATRIX<TYPE> object, which will result in a mutable diagonalMatrix. The fully explanation of the issue will be written on the relation file of our project, please read it.

Returns

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

Definition at line 339 of file matrix.h.

```
339                                             {
340         assert(effective_columns == 1 || effective_rows == 1);
341         if(effective_columns == 1 && columns != 1)
342             return matrix<type>(rows, columns,
      effective_rows, effective_columns, start_row,
      start_column, true, diag, true, pter);
343         else
344             return matrix<type>(rows, columns,
      effective_rows, effective_columns, start_row,
      start_column, true, diag, false, pter);
345     }
```

4.6.4.14 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 391 of file matrix.h.

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

4.6.4.15 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 405 of file matrix.h.

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

4.6.4.16 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 375 of file matrix.h.

```
375                                {
376         return effective_columns;
377     }
```

4.6.4.17 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 366 of file matrix.h.

```
366                             {
367         return effective_rows;
368     }
```

4.6.4.18 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 198 of file matrix.h.

```
198                                                 {
199         if((diag == true) && !transp){
200             assert(column == 0);
201             return pter->operator[]((row + start_row) * (columns) + (row +
      start_column));
202         }
203         else if((diag == true) && (transp == true)){
204             assert(row == 0);
205             return pter->operator[]((column + start_column) * (
      rows) + (column + start_row));
206         }
207         else if((!diag && (transp == true)))
208             return pter->operator[]((column + start_column) * (
      rows) + (row + start_row));
209         else
210             return pter->operator[]((row+start_row) * (columns) + (column +
      start_column));
211       }
```

4.6.4.19 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 220 of file matrix.h.

```
220                                                 {
221         if(diagmatr == true){
222             if(row != column)
223                 return zero;
224             else{
```

```
225                 if(from_diag)
226                     return pter->operator[]((row + start_row) * (
    columns) + (row + start_column));
227                 else if(from_subcovector)
228                     return pter->operator[]((row*columns) + (
    start_row * columns + start_column));
229                 else
230                     return pter->operator[](row + (start_row *
    columns + start_column));
231             }
232         }
233         else if((diag == true) && !(transp)){
234             assert(column == 0);
235             return pter->operator[]((row + start_row) * (columns) + (row +
    start_column));
236         }
237         else if((diag == true) && (transp == true))
238             return pter->operator[]((column + start_column) * (
    rows) + column + start_row);
239         else if((!diag && (transp == true)))
240             return pter->operator[]((column + start_column) * (
    rows) + (row + start_row));
241         else
242             return pter->operator[]((row + start_row) * (columns) + column +
    start_column);
243     }
```

4.6.4.20 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 136 of file matrix.h.

```
136                                                 {
137         columns = other.columns;
138         rows = other.rows;
139         effective_rows = other.effective_rows;
140         effective_columns = other.effective_columns;
141         start_row = other.start_row;
142         start_column = other.start_column;
143         transp = other.transp;
144         diag = other.diag;
145         diagmatr = other.diagmatr;
146         from_diag = other.from_diag;
147         from_subcovector = other.from_subcovector;
148         pter = other.pter;  //maybe private method to do this
149         other.pter = nullptr; //same problem as above
150     }
```

4.6.4.21 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 157 of file matrix.h.

```
157                                                 {
158         if (this != &other){
159             matrix<T> tmp(other);
160             this->swap(tmp);
161         }
162         return *this;
163     }
```

4.6.4.22 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 413 of file matrix.h.

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

4.6.4.23 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 429 of file matrix.h.

```
429 { return const_row_iterator(*this, i, 0); }
```

4.6.4.24 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 444 of file matrix.h.

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

4.6.4.25 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 458 of file matrix.h.

```
458 { return const_row_iterator(*this, 0, 0); }
```

4.6.4.26 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 421 of file matrix.h.

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

4.6.4.27 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 437 of file matrix.h.

```
437 { return const_row_iterator(*this, i + 1, 0); }
```

4.6.4.28 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 451 of file matrix.h.

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

4.6.4.29 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 465 of file matrix.h.

```
465 { return const_row_iterator(*this, effective_rows, 0); }
```

4.6.4.30 subMatrix() [1/2]

```
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 254 of file matrix.h.

```
254
                        {
255         const unsigned new_eff_rows = end_row - start_row + 1;
256         const unsigned new_eff_columns = end_column - start_column + 1;
257         return  matrix<type>(rows, columns, new_eff_rows, new_eff_columns, (this->
      start_row + start_row) ,(this->start_column + start_column), transp,
      diag, pter);
258     }
```

4.6.4.31 subMatrix() [2/2]

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

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

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 269 of file matrix.h.

```
269
                                {
270         const unsigned new_eff_rows = end_row - start_row + 1;
271         const unsigned new_eff_columns = end_column - start_column + 1;
272         return matrix<type>(rows, columns, new_eff_rows, new_eff_columns, (this->
      start_row + start_row) ,(this->start_column + start_column), transp,
      diag, pter);
273     }
```

4.6.4.32 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 170 of file matrix.h.

```
170                                 {
171         std::swap(other.pter, this->pter);
172         std::swap(other.columns, this->columns);
173         std::swap(other.rows, this->rows);
174         std::swap(other.effective_columns, this->effective_columns);
```

```
175         std::swap(other.effective_rows, this->effective_rows);
176         std::swap(other.pter, this->pter);
177         std::swap(other.start_column, this->start_column);
178         std::swap(other.start_row, this->start_row);
179         std::swap(other.transp, this->transp);
180         std::swap(other.transp, this->diag);
181         std::swap(other.diagmatr, this->diagmatr);
182         std::swap(other.from_diag, this->from_diag);
183         std::swap(other.from_subcovector, this->from_subcovector);
184     }
```

4.6.4.33 transpose() [1/2]

```
template<typename T>
matrix matrix< T >::transpose ( )  [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 280 of file matrix.h.

```
280                     {
281         const unsigned new_rows = effective_columns;
282         const unsigned new_columns = effective_rows;
283         const unsigned new_start_row = start_column;
284         const unsigned new_start_column = start_row;
285         const bool new_transp = !transp;
286 
287         return matrix<type>(columns, rows, new_rows, new_columns, new_start_row,
      new_start_column, new_transp,diag, pter);
288     }
```

4.6.4.34 transpose() [2/2]

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

transpose const method (REQUESTED) It returns a const 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 295 of file matrix.h.

```
295                                 {
296         const unsigned new_rows = effective_columns;
297         const unsigned new_columns = effective_rows;
298         const unsigned new_start_row = start_column;
299         const unsigned new_start_column = start_row;
300         const bool new_transp = !transp;
301 
302         return matrix<type>(columns, rows, new_rows, new_columns, new_start_row,
      new_start_column, new_transp,diag, pter);
303     }
```

4.6.5 Member Data Documentation

4.6.5.1 columns

```
template<typename T>
unsigned matrix< T >::columns  [private]
```

Definition at line 565 of file matrix.h.

4.6.5.2 diag

```
template<typename T>
bool matrix< T >::diag  [private]
```

Definition at line 564 of file matrix.h.

4.6.5.3 diagmatr

```
template<typename T>
bool matrix< T >::diagmatr  [private]
```

Definition at line 564 of file matrix.h.

4.6.5.4 effective_columns

```
template<typename T>
unsigned matrix< T >::effective_columns  [private]
```

Definition at line 566 of file matrix.h.

4.6.5.5 effective_rows

```
template<typename T>
unsigned matrix< T >::effective_rows  [private]
```

Definition at line 566 of file matrix.h.

4.6.5.6 from_diag

```
template<typename T>
bool matrix< T >::from_diag  [private]
```

Definition at line 564 of file matrix.h.

4.6.5.7 from_subcovector

```
template<typename T>
bool matrix< T >::from_subcovector  [private]
```

Definition at line 564 of file matrix.h.

4.6.5.8 pter

```
template<typename T>
std::shared_ptr<std::vector<type> > matrix< T >::pter  [private]
```

Definition at line 563 of file matrix.h.

4.6.5.9 rows

```
template<typename T>
unsigned matrix< T >::rows  [private]
```

Definition at line 565 of file matrix.h.

4.6.5.10 start_column

```
template<typename T>
unsigned matrix< T >::start_column  [private]
```

Definition at line 566 of file matrix.h.

4.6.5.11 start_row

```
template<typename T>
unsigned matrix< T >::start_row  [private]
```

Definition at line 566 of file matrix.h.

4.6.5.12 transp

```
template<typename T>
bool matrix< T >::transp  [private]
```

Definition at line 564 of file matrix.h.

4.6.5.13 zero

```
template<typename T>
const type matrix< T >::zero = type()  [private]
```

Definition at line 567 of file matrix.h.

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

- matrix.h

4.7 Matrix Class 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>
```

4.7.1 Detailed Description

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

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

- matrix.h

5 File Documentation

5.1 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.

5.1.1 Detailed Description

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

5.2 main.cpp File Reference

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

Classes

- struct course

Functions

- void test_fondamental_methods ()
- void test_transpose ()
- void test_subMatrix ()
- void test_diagonal ()
- void test_diagonalmatrix ()
- void test_deepcopy ()
- void test_iterators ()
- void test_library_usage ()
- std::ostream & operator<< (std::ostream &os, const course c)
- void test_custom_type ()
- int main ()

5.2.1 Function Documentation

5.2.1.1 main()

```
int main ( )
```

Definition at line 411 of file main.cpp.

```
411           {
412
413     test_fondamental_methods();
414
415     test_deepcopy();
416
417     test_subMatrix();
418
419     test_transpose();
420
421     test_diagonal();
422
423     test_diagonalmatrix();
424
425     test_iterators();
426
427     test_library_usage();
428
429     test_custom_type();
430
431 }
```

5.2.1.2 operator<<()

```
std::ostream& operator<< (
            std::ostream & os,
            const course c )
```

Definition at line 376 of file main.cpp.

```
376                                                     {
377     os << c.name << " ; " << c.credits;
378     return os;
379 }
```

5.2.1.3 test_custom_type()

```
void test_custom_type ( )
```

Definition at line 381 of file main.cpp.

```
381                         {
382     std::cout << "***CUSTOM TYPE TEST***\n\n";
383
384     std::cout << "***We will create a matrix containg objects of type course***\n\n";
385
386     matrix<course> A(4, 5);
387
388     std::cout << "Empty 4x5 matrix\n" << A;
389
390     course def(6, "Advanced algorithm 2");
391     matrix<course> B(3, 4, def);
392
393     std::cout << "3x4 matrix with fixed value\n" << B;
394
395     std::cout << "Operations test on last matrix\n\n\n";
396     std::cout << "Transpose\n";
397     std::cout << B.transpose();
398
399     std::cout << "Submatrix 2x2\n";
400     std::cout << B.subMatrix(0,0,1,1);
401
402     std::cout << "Diagonal\n";
403     std::cout << B.diagonal();
404
405     std::cout << "Diagonal matrix of diagonal\n";
406     std::cout << B.diagonal().diagonalMatrix();
407
408 }
```

5.2.1.4 test_deepcopy()

```
void test_deepcopy ( )
```

Definition at line 134 of file main.cpp.

```
134                     {
135     matrix<int> A(4,5,6),B(A);
136     std::cout<< "***TEST DEEP COPY*** \n\n" << B << std::endl;
137 
138     matrix<int> C(B.transpose()); //This thing shall not deep copy the object (RVO but it's
        theoretically move constructor)
139     std::cout<< "matrix copy costructor on transp method \n" << C << std::endl;
140     std::cout<< "Changing 1,1 element, the first two matrixies does not have to share memory, the last two
        does \n";
141     C(1,1) = 0;
142 
143     matrix<int> D = B;
144     std::cout<< "Printing matrixes \n" << A << std::endl << B << std::endl << C;
145 
146     std::cout<< "Deep copying a Diagonal Matrix" << std::endl;
147     std::cout<< "Start matrix\n" << A;
148     const matrix<int> E = A.diagonal().diagonalMatrix();
149     std::cout<< "\nDiagonal Matrix\n" << E;
150     matrix<int> F = E;
151     std::cout<< "\nDeep copy matrix\n" << F;
152     F(0,0) = 1;
153     std::cout<< "\nMatrixes after a modification on the deep copied one\n" << F << std::endl << E;
154 }
```

5.2.1.5 test_diagonal()

```
void test_diagonal ( )
```

Definition at line 85 of file main.cpp.

```
85                     {
86 
87     std::cout << "***DIAGONAL TEST***\n\n";
88 
89     matrix<int> A(4,5);
90 
91     for(int r = 0; r != 4; r++){
92         for(int c = 0; c != 5; c++){
93             A(r, c) = r + c;
94         }
95     }
96 
97     std::cout << "matrix A\n" << A << std::endl;
98 
99     auto B = A.diagonal();
100 
101     std::cout << "diagonal of A\n" << B << std::endl;
102 
103     matrix<int> C(5,4);
104 
105     for(int r = 0; r != 5; r++){
106         for(int c = 0; c != 4; c++){
107             C(r, c) = r + c;
108         }
109     }
110 
111     std::cout << "matrix C\n" << C << std::endl;
112 
113     auto D = C.diagonal();
114 
115     std::cout << "diagonal of C\n" << D << std::endl;
116 
117 }
```

5.2.1.6 test_diagonalmatrix()

```
void test_diagonalmatrix ( )
```

Definition at line 120 of file main.cpp.

```
120                             {
121     std::cout << "***TEST DIAGONALMATRIX***\n\n";
122     matrix<int> A(1,6,6);
123
124     std::cout << "matrix A\n" << A << std::endl;
125
126     const matrix<int> B = A.diagonalMatrix().transpose();
127
128     std::cout << "diagonal matrix of A\n" << B << std::endl;
129
130
131 }
```

5.2.1.7 test_fondamental_methods()

```
void test_fondamental_methods ( )
```

Definition at line 6 of file main.cpp.

```
6                                {
7     std::cout << "***TEST FONDAMENTAL METHODS***\n\n";
8
9     //testing the constructor with 0 rows, 0 colunms filled with
10     //default values
11     matrix<int> A(0,0);
12     std::cout << "matrix of 0 elements\n" << A << std::endl;
13
14     //generic matrices with default values
15     matrix<int> B(3,4);
16     std::cout << "matrix (3,4) with default (int) values\n" << B << std::endl;
17
18     //generic matrix with strings as elements
19     matrix<std::string> C(2,2);
20     std::cout << "matrix with strings as elements (default values "")\n" << C << std::endl;
21
22     //matrix with strings as elements and "prova" as default value
23     matrix<std::string> D(4,3, "prova");
24     std::cout << "matrix with strings as elements and \"prova as default value\"\n" << D << std::endl;
25
26     //copy constructor (results in a deep copy of the matrix)
27     matrix<std::string> E(D);
28     std::cout << "copy constructor of the previous matrix\n" << E << std::endl;
29
30     //assignment operator test
31     matrix<std::string> F = E;
32     std::cout << "assignment operator\n" << F << std::endl;
33
34     //test operator ()
35     F(1,1) = "(1,1)";
36     F(2,2) = "(2,2)";
37
38     std::cout << "modified positions (1,1) and (2,2)\n" << F << std::endl;
39 }
```

5.2.1.8 test_iterators()

```
void test_iterators ( )
```

Definition at line 157 of file main.cpp.

```
157                     {
158     std::cout << "***TEST ITERATORS***\n\n";
159 
160     matrix<int> A(4,5);
161 
162     for(int r = 0; r != 4; r++){
163         for(int c = 0; c != 5; c++){
164             A(r, c) = r + c;
165         }
166     }
167 
168     std::cout << "Matrix A\n" << A << std::endl;
169 
170     //here we iterate the matrix by row and by colunm, on the whole matrix or selecting the
171     //starting row/column and the ending row/column
172 
173     std::cout << "Matrix iterated with the row iterator\n";
174 
175     for(auto iter = A.row_begin(); iter != A.row_end(); ++iter) {
176         std::cout << *iter << " ";
177     }
178 
179     std::cout << "\n";
180 
181     std::cout << "\nMatrix iterated with the col iterator\n";
182 
183     for(auto iter = A.col_begin(); iter != A.col_end(); ++iter) {
184         std::cout << *iter << " ";
185     }
186 
187 
188     std::cout << "\n";
189 
190     std::cout << "\nMatrix iterated with the row iterator, only the first row\n";
191 
192     for(auto iter = A.row_begin(0); iter != A.row_end(0); ++iter) {
193         std::cout << *iter << " ";
194     }
195 
196     std::cout << "\n";
197 
198     std::cout << "\nMatrix iterated with the row iterator, from row 2 to 3\n";
199 
200     for(auto iter = A.row_begin(1); iter != A.row_end(2); ++iter) {
201         std::cout << *iter << " ";
202     }
203 
204     std::cout << "\n";
205 
206     std::cout << "\nMatrix iterated with the col iterator, only the last column\n";
207 
208     for(auto iter = A.col_begin(4); iter != A.col_end(4); ++iter) {
209         std::cout << *iter << " ";
210     }
211 
212     std::cout << "\n";
213 
214     std::cout << "\nMatrix iterated with the col iterator, from row 1 to 2\n";
215 
216     for(auto iter = A.col_begin(0); iter != A.col_end(1); ++iter) {
217         std::cout << *iter << " ";
218     }
219 
220     std::cout << "\n\n";
221 
222 
223     //testing the const iterators (called automatically on a constant matrix)
224     matrix<int> B(4,4);
225 
226     for(auto iter = B.row_begin(); iter != B.row_end(); ++iter) {
227         *iter = rand() % 50;
228     }
229 
230     const auto C = B;
231 
232     std::cout << "Matrix B\n" << B << std::endl;
233 
```

```
234     std::cout << "Matrix iterated with the const row iterator" << std::endl;
235
236     for(auto iter = C.row_begin(); iter != C.row_end(); ++iter) {
237         std::cout << *iter << " ";
238     }
239
240     std::cout << "\n\n";
241
242
243     std::cout << "Matrix iterated with the const column iterator" << std::endl;
244
245     for(auto iter = C.col_begin(); iter != C.col_end(); ++iter) {
246         std::cout << *iter << " ";
247     }
248
249     std::cout << "\n\n";
250
251
252     //the same tests as before but on a constant matrix
253
254     std::cout << "Matrix iterated with the row iterator, only the first column\n";
255
256     for(auto iter = C.row_begin(0); iter != C.row_end(0); ++iter) {
257         std::cout << *iter << " ";
258     }
259
260     std::cout << "\n\n";
261
262     std::cout << "Matrix iterated with the row iterator, from column 2 to 3\n";
263
264     for(auto iter = C.row_begin(1); iter != C.row_end(2); ++iter) {
265         std::cout << *iter << " ";
266     }
267
268     std::cout << "\n\n";
269
270     std::cout << "Matrix iterated with the col iterator, only the last column\n";
271
272     for(auto iter = C.col_begin(3); iter != C.col_end(3); ++iter) {
273         std::cout << *iter << " ";
274     }
275
276     std::cout << "\n\n";
277
278
279     std::cout << "Matrix iterated with the col iterator, from colunm 1 to 2\n";
280
281     for(auto iter = C.col_begin(0); iter != C.col_end(1); ++iter) {
282         std::cout << *iter << " ";
283     }
284
285     std::cout << "\n";
286
287 }
```

5.2.1.9 test_library_usage()

```
void test_library_usage ( )
```

Definition at line 290 of file main.cpp.

```
290                         {
291     std::cout << "***TEST LIBRARY USAGE***\n\n";
292
293     matrix<int>A (3,6);
294
295     for(auto iter = A.row_begin(); iter != A.row_end(); ++iter) {
296         *iter = rand() % 50;
297     }
298
299     std::cout << "matrix A\n" << A << std::endl;
300
301     auto B = A.transpose();
302
303     std::cout << "transpose of A (matrix B)\n" << B << std::endl;
304
305     auto C = B.subMatrix(0,0,0,2);
```

```
306
307     std::cout << "submatrix of B (taking the first row, creating vector C)\n" << C << std::endl;
308
309     auto D = C.transpose();
310
311     std::cout << "transposing C (creating vector D)\n" << D << std::endl;
312
313     auto E = D.diagonalMatrix();
314
315     std::cout << "creating the diagonalmatrix starting from D (matrix E)(submatrix covector)\n" << E <<
    std::endl;
316
317     matrix<int>F(4,1,7);
318
319     std::cout << "creating diagonal matrix from standard covector" << std::endl;
320
321     auto O = F.diagonalMatrix();
322
323     std::cout << O;
324
325     std::cout << "creating diagonal matrix from standard vector" << std::endl;
326
327     matrix<int>G(1,4,7);
328
329     std::cout << G.diagonalMatrix();
330
331     std::cout << "creating diagonal matrix from submatrix vector" << std::endl;
332     matrix<int> J = A.subMatrix(0,2,0,5);
333
334     std::cout << "starting vector" << std::endl;
335     std::cout << J;
336     std::cout << "\n" << std::endl;
337     std::cout << J.diagonalMatrix();
338
339
340     std::cout << "creating diagonal matrix from diagonal vector" << std::endl;
341     std::cout << "starting matrix" << std::endl;
342     std::cout << A;
343     std::cout << "starting vector" << std::endl;
344     std::cout << A.diagonal();
345     std::cout << "result matrix" << std::endl;
346     std::cout << A.diagonal().diagonalMatrix();
347
348     std::cout << "Using operators on temporary objects" << std::endl;
349     std::cout << "Transpose of a diagonal vector of a submatrix\n";
350     std::cout << "Starting matrix\n" << A;
351     std::cout << "Result vector \n" << A.subMatrix(0,0,2,2).diagonal().transpose();
352
353     std::cout << "Now the (0, 0) element of a temporary view of the matrix will be changed\n";
354     A.diagonal().transpose().subMatrix(0, 0, 0, 0)(0,0) = 0;
355     std::cout << "Showing some views of the matrix, there should be a 0 somewhere\n";
356     std::cout << A << std::endl << C << std::endl << B << std::endl;
357
358 }
```

5.2.1.10 test_subMatrix()

```
void test_subMatrix ( )
```

Definition at line 62 of file main.cpp.

```
62                      {
63     matrix<int> A(4,5);
64
65     for(int r = 0; r != 4; r++){
66         for(int c = 0; c != 5; c++){
67             A(r, c) = r + c;
68         }
69     }
70
71     std::cout << "matrix with assigned values (we effetuate all the submatrices on this matrix)\n" << A <<
    std::endl;
72
73
74     matrix<int> B = A.subMatrix(1,1,2,2);
75     std::cout << "sub starting from (1,1) and ending in (2,2)\n" << B << std::endl;
76
```

```
77     matrix<int> C = A.subMatrix(0,0,3,4);
78     std::cout << "submatrix starting form the begin of the matrix and ending to the end of the matrix\n" <<
       C << std::endl;
79
80     matrix<int> D = A.subMatrix(0,1,1,3);
81     std::cout << "submatrix starting from (0,1) and ending in (1,3)\n" << D << std::endl;
82 }
```

5.2.1.11 test_transpose()

```
void test_transpose ( )
```

Definition at line 42 of file main.cpp.

```
42                      {
43     std::cout << "***TEST TRANSPOSE METHOD***\n\n";
44
45     matrix<int> A(4,5);
46
47     for(int r = 0; r != 4; r++){
48         for(int c = 0; c != 5; c++){
49             A(r, c) = r + c;
50         }
51     }
52
53     std::cout << "matrix A\n" << A << "\n\n";
54
55     auto B = A.transpose(); //doesn't use copy constructor thanks to rvo
56
57     std::cout << "matrix B\n" << B << "\n\n";
58
59 }
```

5.3 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 >

Functions

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

  Overload of stream operator that permits printing a matrix object.

5.3.1 Detailed Description

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

5.3.2 Function Documentation

5.3.2.1 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 579 of file matrix.h.

```
580 {
581     for (unsigned r = 0; r < ma.getRows(); r++){
582         for (unsigned c = 0; c < ma.getColumns(); c++){
583             os << "[" << ma(r, c) << "] ";
584         }
585         os << std::endl;
586     }
587     return os;
588 }
```

5.4 matrix_forward.h File Reference

Forward declaration needed for using file iterator.h.

Classes

- class matrix< T >

5.4.1 Detailed Description

Forward declaration needed for using file iterator.h.

5.5 README.md File Reference

Index