Yotcopi User Guide 1A – Real Dense Matrix

Jirawat Tangpanitanon[1] and Ping Nang Ma[2]

[1] *Center for Quantum Technologies, National University of Singapore*
[2] *Yotcopi Technologies, National University of Singapore*

Contents

I. BASIC OPERATIONS

The basic matrix operations such as addition, subtraction and multiplication can be done directly using +, - and & (with bracket), respectively. See the following code for illustration:

Example: +, - and &

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
  // insert the following two lines for
  // using short-hand notations for Yotcopi functions
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  // defining 2x2 matrices.
  // auto means "c++ automatic type"
  auto A = m({{0.,1.},{0.,0.}});
  auto B = m({{0.,0.},{1.,0.}});

  auto sum = A + B;
  auto minus = A - B;
  auto product = (A & B);   // need a bracket for a product

  // a matrix can be printed directly using
  // the standard std::cout command
  std::cout << "A = " << "\n" << A;
  std::cout << "B = " << "\n" << B;
  std::cout << "A+B = " << "\n" << sum;
  std::cout << "A-B = " << "\n" << minus;
  std::cout << "(A & B) =" << "\n" << product;

  return 0;
}
```

Output:

```
A =
                0                1
                0                0
B =
                0                0
                1                0
A+B =
                0                1
                1                0
A-B =
                0                1
               -1                0
(A & B) =
                1                0
                0                0
```

II. CONSTRUCTING OBJECTS

A real dense matrix object can be constructed in various ways depending on its arguments.

Example: Constructing a dense matrix

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A1;                          // A1 is an empty matrix
  m A2(2);                       // A2 is a 2x2 zero matrix
  m A3(3,4);                     // A3 is a 3x4 zero matrix
  m A4({1.1,2.2,3.,4.4});        // A4 is a 4x4 matrix with
                                 // diagonal terms defined
                                 // by the input vector
  m A5({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
                                 // A5 is a 2x2 matrix whose
                                 // elements specified by
                                 // the input

  std::cout << "A1 = " << "\n" << A1;
  std::cout << "A2 = " << "\n" << A2;
  std::cout << "A3 = " << "\n" << A3;
  std::cout << "A4 = " << "\n" << A4;
  std::cout << "A5 = " << "\n" << A5;

  return 0;
}
```

Output:

```
A1 =
A2 =
                0                0
                0                0
A3 =
                0                0                0                0
                0                0                0                0
                0                0                0                0
A4 =
              1.1                0                0                0
                0              2.2                0                0
                0                0                3                0
```

```
                0                0                0           4.4
A5 =
                1                2                3
                4                5                6
                7                8                9
```

III. BASIC PROPERTIES

A dense matrix contains a number of properties which can be obtained by using a command `dense_matrix.property_name()`. The following are examples of some of the important properties.

A. Size and shape of the dense matrix

- `length()` returns the number of rows of the matrix.

Example:`length()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(3,4);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.length() = " << A.length();

    return 0;
}
```

Output:

```
A =
    0           0           0           0
    0           0           0           0
    0           0           0           0

A.length() = 3
```

- `width()` returns the number of columns of the matrix.

Example:`width()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(3,4);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.width() = " << A.width();

    return 0;
}
```

Output:

```
A =
     0     0     0     0
     0     0     0     0
     0     0     0     0

A.width() = 4
```

• `size()` returns the number of matrix elements.

Example:size()

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(3,4);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.size() = " << A.size();

    return 0;
}
```

Output:

```
    A =
         0     0     0     0
         0     0     0     0
         0     0     0     0

    A.size() = 12
```

• `shape()` returns a vector. The first and the second elements contain the numbers of rows and columns of the matrix, repectively.

Example:shape()

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(3,4);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.shape() = " << A.shape();

    return 0;
}
```

Output:

```
    A =
         0     0     0     0
         0     0     0     0
         0     0     0     0

    A.shape() =3 4
```

B. Checking matrix properties

- `empty()` returns 1 if the matrix is empty and 0 otherwise.

Example: `empty.()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A1;
    m A2(2);

    std::cout << "A1 = " << "\n" << A1 << "\n";
    std::cout << "A2 = " << "\n" << A2 << "\n";
    std::cout << "A1.empty() = " << A1.empty() << "\n";
    std::cout << "A2.empty() = " << A2.empty();

    return 0;
}
```

Output:

```
    A1 =

    A2 =
            0               0
            0               0

    A1.empty() = 1
    A2.empty() = 0
```

- `is_square_matrix()` returns 1 if the matrix is a square matrix and 0 otherwise.

Example: `.is_square_matrix()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A1(2);
    m A2(2,3);

    std::cout << "A1 = " << "\n" << A1 << "\n";
    std::cout << "A2 = " << "\n" << A2 << "\n";
    std::cout << "A1.is_square_matrix() = "
              << A1.is_square_matrix() << "\n";
    std::cout << "A2.is_square_matrix() = "
              << A2.is_square_matrix();

    return 0;
}
```

Output:

```
A1 =
            0               0
            0               0

A2 =
```

```
                0               0               0
                0               0               0

A1.is_square_matrix() = 1
A2.is_square_matrix() = 0
```

- `is_symmetric_matrix()` returns 1 if the matrix is symmetric and 0 otherwise.

Example: `.is_symmetric_matrix()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A1(2);
    m A2({{0.,1.},{0.,0.}});

    std::cout << "A1 = " << "\n" << A1 << "\n";
    std::cout << "A2 = " << "\n" << A2 << "\n";
    std::cout << "A1.is_symmetric_matrix() = "
              << A1.is_symmetric_matrix() << "\n";
    std::cout << "A2.is_symmetric_matrix() = "
              << A2.is_symmetric_matrix();

    return 0;
}
```

Output:

```
A1 =
                0               0
                0               0

A2 =
                0               1
                0               0

A1.is_symmetric_matrix() = 1
A2.is_symmetric_matrix() = 0
```

- `is_hermitian_matrix()` returns 1 if the matrix is hermitian and 0 otherwise.

Example: `.is_hermitian_matrix()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A1({{0.,1.},{1.,0.}});
    m A2({{0.,1.},{-1.,0.}});

    std::cout << "A1 = " << "\n" << A1 << "\n";
    std::cout << "A2 = " << "\n" << A2 << "\n";
    std::cout << "A1.is_hermitian_matrix() = "
              << A1.is_hermitian_matrix() << "\n";
    std::cout << "A2.is_hermitian_matrix() = "
              << A2.is_hermitian_matrix();

    return 0;
}
```

Output:

```
A1 =
                0                1
                1                0

A2 =
                0                1
               -1                0

A1.is_hermitian_matrix() = 1
A2.is_hermitian_matrix() = 0
```

- `is_diagonal_matrix()` returns 1 if the matrix is diagonal and 0 otherwise.

Example: `.is_diagonal_matrix()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A1({{1.,0.},{0.,1.}});
    m A2({{0.,1.},{1.,0.}});

    std::cout << "A1 = " << "\n" << A1 << "\n";
    std::cout << "A2 = " << "\n" << A2 << "\n";
    std::cout << "A1.is_diagonal_matrix() = "
              << A1.is_diagonal_matrix() << "\n";
    std::cout << "A2.is_diagonal_matrix() = "
              << A2.is_diagonal_matrix();

    return 0;
}
```

Output:

```
A1 =
                1                0
                0                1

A2 =
                0                1
                1                0

A1.is_diagonal_matrix() = 1
A2.is_diagonal_matrix() = 0
```

- `is_diagonal_matrix()` returns 1 if the matrix is an identity matrix and 0 otherwise.

Example: `.is_identity_matrix()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A1({{1.,0.},{0.,1.}});
    m A2({{0.,1.},{1.,0.}});

    std::cout << "A1 = " << "\n" << A1 << "\n";
    std::cout << "A2 = " << "\n" << A2 << "\n";
    std::cout << "A1.is_identity_matrix() = "
              << A1.is_identity_matrix() << "\n";
```

```
    std::cout << "A2.is_identity_matrix() = "
              << A2.is_identity_matrix();

    return 0;
}
```

Output:

```
A1 =
                1               0
                0               1

A2 =
                0               1
                1               0

A1.is_identity_matrix() = 1
A2.is_identity_matrix() = 0
```

C. Extract a vector/ a matrix from the dense matrix

• `row(int n)` returns a vector containing elements of the dense matrix in the row specified by n. The first row is indicated by 0.

Example:`row()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

  std::cout << "A = " << "\n" << A << "\n";
  std::cout << "A.row(0) = " << A5.row(0);

  return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

A.row(0) =  1  2  3
```

• `column(int n)` returns a vector containing elements of the dense matrix in the column specified by n. The first row is indicated by 0.

Example:`column()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
```

```
  std::cout << "A = " << "\n" << A << "\n";
  std::cout << "A.column(1) = " << A.column(1);

  return 0;
}
```

Output:

```
A =
            1               2               3
            4               5               6
            7               8               9
A.column(1) = 2   5   8
```

• `rows(std::vector<unsigned int > v)` returns a vector containing elements of the dense matrix in the row specified by n. The first row is indicated by 0.

Example:`rows(std::vector<unsigned int > v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

  std::cout << "A = " << "\n" << A << "\n";
  std::cout << "A.row(0) = " << A5.row(0);

  return 0;
}
```

Output:

```
A =
            1               2               3
            4               5               6
            7               8               9

A.row(0) =  1   2   3
```

• `diagonal()` returns a vector containing diagonal elements of the dense matrix.

Example:`diagonal()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

  std::cout << "A = " << "\n" << A << "\n";
  std::cout << "A.diagonal() = " << A5.diagonal();

  return 0;
}
```

Output:

```
A =
            1               2               3
            4               5               6
            7               8               9
A.diagonal() =  1   5   9
```

• `superdiagonal(int n)` returns a vector containing off-diagonal elements of the dense matrix, specified by an integer `n`. If `n` is not a double value, then it will be rounded down to an integer.

Example:`superdiagonal(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
using namespace yotcopi;
using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

  std::cout << "A = " << "\n" << A << "\n";
  std::cout << "A.superdiagonal(0) = "
            << A.superdiagonal(0) << "\n";
  std::cout << "A.superdiagonal(1) = "
            << A.superdiagonal(1) << "\n";
  std::cout << "A.superdiagonal(-1) = "
            << A.superdiagonal(-1);

  return 0;
}
```

Output:

```
A =
            1               2               3
            4               5               6
            7               8               9
A.superdiagonal(0) =  1   5   9
A.superdiagonal(1) =  2   6
A.superdiagonal(-1) = 4   8
```

There are some more handy functions in this catagory:

• `odd_rows()` returns a new matrix containing odd rows of the original matrix.
• `even_rows()` returns a new matrix containing even rows of the original matrix.
• `odd_columns()` returns a new matrix containing odd columns of the original matrix.
• `even_columns()` returns a new matrix containing even columns of the original matrix.
• `odd_submatrix()` returns a new matrix containing odd submatrices of the original matrix.
• `even_submatrix()` returns a new matrix containing even submatrices of the original matrix.
• `upper_triangular()` returns a new matrix containing an upper triangular part of the original matrix.
• `lower_triangular()` returns a new matrix containing an lower triangular part of the original matrix.

Example:`superdiagonal(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.,4.},{5.,6.,7.,8.},{9.,10.,11.,12.},{13.,14.,15.,16.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.odd_rows() =" << "\n" << A.odd_rows() << "\n";
    std::cout << "A.even_rows() =" << "\n" << A.even_rows() << "\n";
```

```
    std::cout << "A.odd_columns() =" << "\n" << A.odd_columns() << "\n";
    std::cout << "A.even_columns() =" << "\n" << A.even_columns() << "\n";
    std::cout << "A.odd_submatrix() =" << "\n" << A.odd_submatrix() << "\n";
    std::cout << "A.even_submatrix() =" << "\n" << A.even_submatrix() << "\n";
    std::cout << "A.upper_triangular() =" << "\n" << A.upper_triangular() << "\n";
    std::cout << "A.lower_triangular() = " << "\n" << A.lower_triangular() << "\n";

    return 0;
}
```

Output:

```
A =
                1                2                3                4
                5                6                7                8
                9               10               11               12
               13               14               15               16

A.odd_rows() =
                5                6                7                8
               13               14               15               16

A.even_rows() =
                1                2                3                4
                9               10               11               12

A.odd_columns() =
                2                4
                6                8
               10               12
               14               16

A.even_columns() =
                1                3
                5                7
                9               11
               13               15

A.odd_submatrix() =
                6                8
               14               16

A.even_submatrix() =
                1                3
                9               11

A.upper_triangular() =
                1                2                3                4
                0                6                7                8
                0                0               11               12
                0                0                0               16

A.lower_triangular() =
                1                0                0                0
                5                6                0                0
                9               10               11                0
               13               14               15               16
```

IV. USEFUL MATHEMATICAL OPERATIONS

A. operating without modifying the original matrix

In this section, we list useful mathematical expressions. Operations are done by `operation(matrix)`. It gives out a new corresponding matrix without changing the original. In the following we will refer to the original (input) matrix as A.

- `negate(A)`: simply means -A.
- `abs(A)`: absolute values of every elements in A.
- `sq(A)`: squaring every elements in A.
- `cb(A)`: raising every elements in A to the third.
- `sqrt(A)`: square root every elements in A.
- `cbrt(A)`: cube root every elements in A.
- `cos(A)`: cosine of every elements in A.
- `sin(A)`: sine of every elements in A.
- `tan(A)`: tangent of every elements in A.
- `exp(A)`: expoential of every elements in A.
- `log(A)`: logarithm of every elements in A.
- `pow(A,n)`: raising every elements in A to the power n.

Example:operating mathematical operations without modifying the original matrix

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{-1.,-2.,-3.},{4.,5.,6.},{7.,8.,9.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "negate(A) = " << "\n" << negate(A) << "\n";
    std::cout << "abs(A) = " << "\n" << abs(A) << "\n";
    std::cout << "sq(A) = " << "\n" << sq(A) << "\n";
    std::cout << "cb(A) = " << "\n" << cb(A) << "\n";
    std::cout << "sqrt(A) = " << "\n" << sqrt(A) << "\n";
    std::cout << "cbrt(A) = " << "\n" << cbrt(A) << "\n";
    std::cout << "cos(A) = " << "\n" << cos(A) << "\n";
    std::cout << "sin(A) = " << "\n" << sin(A) << "\n";
    std::cout << "tan(A) = " << "\n" << tan(A) << "\n";
    std::cout << "exp(A) = " << "\n" << exp(A) << "\n";
    std::cout << "log(A) = " << "\n" << log(A) << "\n";
    std::cout << "pow(A,2) = " << "\n" << pow(A,2) << "\n";

    std::cout << "\n" << "****************************************" << "\n";
    std::cout << "After operations the original matrix is unchanged" << "\n";
    std::cout << "A = " << "\n" << A << "\n";

    return 0;
}
```

Output:

```
A =
              -1              -2              -3
               4               5               6
               7               8               9

negate(A) =
               1               2               3
              -4              -5              -6
              -7              -8              -9
```

```
abs(A) =
     1     2     3
     4     5     6
     7     8     9

sq(A) =
     1     4     9
    16    25    36
    49    64    81

cb(A) =
    -1    -8   -27
    64   125   216
   343   512   729

sqrt(A) =
   nan   nan   nan
     2 2.23607 2.44949
2.64575 2.82843     3

cbrt(A) =
   nan   nan   nan
1.5874 1.70998 1.81712
1.91293     2 2.08008

cos(A) =
 0.540302 -0.416147 -0.989992
-0.653644 0.283662  0.96017
 0.753902  -0.1455 -0.91113

sin(A) =
-0.841471 -0.909297 -0.14112
-0.756802 -0.958924 -0.279415
 0.656987  0.989358 0.412118

tan(A) =
 -1.55741  2.18504 0.142547
  1.15782 -3.38052 -0.291006
 0.871448 -6.79971 -0.452316

exp(A) =
 0.367879 0.135335 0.0497871
  54.5982   148.413   403.429
  1096.63   2980.96   8103.08

log(A) =
   nan   nan   nan
1.38629 1.60944 1.79176
1.94591 2.07944 2.19722

pow(A,2) =
     1     4     9
    16    25    36
    49    64    81


****************************************
After operations the original matrix is unchanged
A =
    -1    -2    -3
     4     5     6
     7     8     9
```

B. operating while modifying the original matrix

The operations in this section are similar to the previous section except that they return a reference to the orginal matrix. The original matrix is therefore changed after such operations. The available functions are

- `negate_equal()` simply means -A.
- `abs_equal()` absolute values of every elements in A.
- `sq_equal()` squaring every elements in A.
- `cb_equal()` raising every elements in A to the third.
- `sqrt_equal()` square root every elements in A.
- `cbrt_equal()` cube root every elements in A.
- `cos_equal()` cosine of every elements in A.
- `sin_equal()` sine of every elements in A.
- `tan_equal()` tangent of every elements in A.
- `exp_equal()` exponential of every elements in A.
- `log_equal()` logarithm of every elements in A.
- `pow_equal(n)` raising every elements in A to the power n.

Here we show an example of using one of such function: `negate_equal()`

Example:operating mathematical operations while modifying the original matrix

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{-1.,-2.,-3.},{4.,5.,6.},{7.,8.,9.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.negate_equal() = " << "\n" << A.negate_equal() << "\n";
    std::cout << "A = " << "\n" << A << "\n";

    return 0;
}
```

Output:

```
A =
          -1          -2          -3
           4           5           6
           7           8           9

A.negate_equal() =
           1           2           3
          -4          -5          -6
          -7          -8          -9

A =
           1           2           3
          -4          -5          -6
          -7          -8          -9
```

C. more operations

- `multiplies_equal_rows(std::vector<T> v )` multiplies the elements in each row by v.

Example:`multiplies_equal_rows(std::vector<T> v )`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::vector<double> v = {2.,3.};

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.multiplies_equal_rows = " << "\n" << A.multiplies_equal_rows(v) << "\n";

    return 0;
}
```

Output:

```
A =
               1               2
               3               4

A.multiplies_equal_rows =
               2               4
               9              12
```

- `divides_equal_rows(std::vector<T> v )` divides the elements in each row by v.

Example:`divides_equal_rows(std::vector<T> v )`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::vector<double> v = {2.,3.};

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.divides_equal_rows = " << "\n" << A.divides_equal_rows(v) << "\n";

    return 0;
}
```

Output:

```
A =
               1               2
               3               4

A.divides_equal_rows =
             0.5               1
               1         1.33333
```

- `multiplies_equal_columns(std::vector<T> v )` multiplies the elements in each column by v.

Example:`multiplies_equal_columns(std::vector<T> v )`

```cpp
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::vector<double> v = {2.,3.};

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.multiplies_equal_columns = " << "\n" << A.multiplies_equal_columns(v) << "\n
        ";

    return 0;
}
```

Output:

```
A =
                1                2
                3                4

A.multiplies_equal_columns =
                2                6
                6               12
```

- `divides_equal_columns(std::vector<T> v )` divides the elements in each column by v.

Example:`divides_equal_columns(std::vector<T> v )`

```cpp
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::vector<double> v = {2.,3.};

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.divides_equal_columns = " << "\n" << A.divides_equal_columns(v) << "\n";

    return 0;
}
```

Output:

```
A =
                1                2
                3                4

A.divides_equal_columns =
              0.5         0.666667
              1.5          1.33333
```

- `sum_rows()` sums elements in each row and put results in a vector.

Example:`sum_rows()`

```cpp
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
```

```
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.sum_rows() = " << "\n" << A.sum_rows() << "\n";

    return 0;
}
```

Output:

```
A =
                1               2
                3               4

A.sum_rows() =
                3               7
```

- `sum_columns()` sums elements in each column and put results in a vector.

Example:`sum_columns()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.sum_columns() = " << "\n" << A.sum_columns() << "\n";

    return 0;
}
```

Output:

```
A =
                1               2
                3               4

A.sum_columns() =
                4               6
```

- `sum()` sums all elements in the matrix.

Example:`sum()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.sum() = " << A.sum() << "\n";

    return 0;
}
```

Output:

```
A =
                1               2
                3               4

A.sum() = 10
```

V. BASIC INDEXING

There are various ways of accessing matrix elements. The relevant functions are

- `value (unsigned int i, unsigned int j)` returns a value of an element at the i^{th} row and the j^{th} column.
- `value (unsigned int i)` returns a value of the i^{th}element.
- `at (unsigned int i, unsigned int j)` returns a reference of an element at the i^{th} row and the j^{th} column.
- `at (unsigned int i)` returns a reference of the i^{th}element.
- `data (unsigned int i, unsigned int j)` returns a pointer of an element at the i^{th} row and the j^{th} column.
- `data (unsigned int i)` returns a pointer of the i^{th}element.

Example:indexing matrix elements

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    m B({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    m C({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

    // using the function value to get values
    double a1 = A.value(0,1);
    double a2 = A.value(4);

    // printing out
    std::cout << "a1 = " << a1 << "\n";
    std::cout << "a2 = " << a2 << "\n";

    // using the function at to get references
    double& b1 = B.at(0,1);
    double& b2 = B.at(4);

     // printing out
    std::cout << "b1 = " << a1 << "\n";
    std::cout << "b2 = " << a2 << "\n";

    // using the function at to get references
    double c1 = *C.data(0,1);
    double c2 = *C.data(4);

     // printing out
    std::cout << "c1 = " << a1 << "\n";
    std::cout << "c2 = " << a2 << "\n";

    // Now we setting a1, a2, b1, b2 to zero

    a1 = 0; a2 = 0; b1 = 0; b2 = 0; c1 = 0; c2 = 0;

    // See if there is a change in A, B
```

```
    std::cout << "\n\n" << "Now a1=a2=b1=b2=c1=c2=0" << "\n\n";
    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "B = " << "\n" << B << "\n";
    std::cout << "C = " << "\n" << C << "\n";

    return 0;
}
```

Output:

```
a1 = 2
a2 = 5
b1 = 2
b2 = 5
c1 = 2
c2 = 5


Now a1=a2=b1=b2=c1=c2=0

A =
                1               2               3
                4               5               6
                7               8               9

B =
                1               0               3
                4               0               6
                7               8               9

C =
                1               2               3
                4               5               6
                7               8               9
```

There are two more useful functions for accessing matrix elements:

- `front()` returns a reference of the first element in the matrix.
- `back()` returns a reference of the last element in the matrix.

Example:`front()`, `back()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.front() =" << A.front() << "\n";
    std::cout << "A.back() =" << A.back() << "\n";

    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9
```

```
A.front() =1
A.back() =9
```

VI. SPECIAL MATRICES

A. zeros, ones and eye

- `zeros(unsigned int n)` is a zero matrix of size $n \times n$.
- `zeros(unsigned int n, unsigned int m)` is a zero matrix of length n and width m.

Example:zeros

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    std::cout << "zeros(2) = " << "\n" << zeros(2) << "\n";
    std::cout << "zeros(2,3) = " << "\n" << zeros(2,3) << "\n";

    return 0;
}
```

Output:

```
zeros(2) =
           0           0
           0           0

zeros(2,3) =
           0           0           0
           0           0           0
```

- `ones(unsigned int n)` is a one matrix of size $n \times n$.
- `ones(unsigned int n, unsigned int m)` is a one matrix of length n and width m.

Example:ones

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    std::cout << "ones(2) = " << "\n" << ones(2) << "\n";
    std::cout << "ones(2,3) = " << "\n" << ones(2,3) << "\n";

    return 0;
}
```

Output:

```
ones(2) =
           1           1
           1           1

ones(2,3) =
           1           1           1
           1           1           1
```

- `eye(unsigned int n)` is an identity matrix of size $n \times n$.
- `eye(unsigned int n, double value)` is a diagonal matrix of size $n \times n$. The diagonal elements are `value`'s.
- `eye(unsigned int n, double value, int s, bool b)`: when the argument `s` is specified, the diagonal elements are shifted to a super-diagonal position (See an example below). If `b` is specified as `TRUE`, then those elements are mirrored.

Example:eye

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    std::cout << "eye(3) = " << "\n" << eye(3) << "\n";
    std::cout << "eye(3,2) = " << "\n" << eye(3,2) << "\n";
    std::cout << "eye(3,2,1) = " << "\n" << eye(3,2,1) << "\n";

    return 0;
}
```

Output:

```
eye(3) =
                1                0                0
                0                1                0
                0                0                1

eye(3,2) =
                2                0                0
                0                2                0
                0                0                2

eye(3,2,1) =
                0                2                0
                0                0                2
                0                0                0
```

B. matrices for physicists

- `sigmaPlus()`
- `sigmaMinus()`
- `sigmaX()`
- `laplacian(unsigned int size)`
- `numberOperator(unsigned int size)`
- `annihilationOperator(unsigned int size)`
- `creationOperator(unsigned int size)`
- `commutator(RealMatrix A, RealMatrix B)`

Example:matrices for physicists

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
```

```
    using namespace yotcopi::shortkeys;

    std::cout << "sigmaPlus() =" << "\n" << sigmaPlus() << "\n";
    std::cout << "sigmaMinus() =" << "\n" << sigmaMinus() << "\n";
    std::cout << "sigmaX() =" << "\n" << sigmaX() << "\n";
    std::cout << "laplacian(4) =" << "\n" << laplacian(4) << "\n";
    std::cout << "numberOperator(3) =" << "\n" << numberOperator(3) << "\n";
    std::cout << "annihilationOperator(3) =" << "\n" << annihilationOperator(3) << "\n";
    std::cout << "creationOperator(3) =" << "\n" << creationOperator(3) << "\n";

    return 0;
}
```

Output:

```
sigmaPlus() =
                0                1
                0                0

sigmaMinus() =
                0                0
                1                0

sigmaX() =
                0                1
                1                0

laplacian(4) =
                2               -1                0                0
               -1                2               -1                0
                0               -1                2               -1
                0                0               -1                2

numberOperator(3) =
                0                0                0
                0                1                0
                0                0                2

annihilationOperator(3) =
                0                1                0
                0                0          1.41421
                0                0                0

creationOperator(3) =
                0                0                0
                1                0                0
                0          1.41421                0
```

VII. SET / MODIFYING OPERATIONS

A. Setting matrix elements

- `set(value)` replaces every elements in the matrix with `value`.

Example:`set(value)`

```
#include <yotcopi.hpp>
int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
  std::cout << "A =" << "\n" << A5 << "\n";
  std::cout << "A.set(1) = " << "\n"
```

```
            <<  A.set(1);

  return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

A.set(1) =
                1               1               1
                1               1               1
                1               1               1
```

- `set_diagonal(value)` replaces diagonal elements in the matrix with `value`.

Example:set_diagonal(value)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_diagonal(1.) = " << "\n" <<  A.set_diagonal(1.) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                0               0
                0               0

A.set_diagonal(1.) =
                1               0
                0               1

A =
                1               0
                0               1
```

- `set_diagonal(std::vector v)` replaces diagonal elements in the matrix with elements in the vector `v`.

Example:set_diagonal(std::vector v)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_diagonal({1.,2.}) = " << "\n"
```

```
              << A.set_diagonal({1.,2.}) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
              0             0
              0             0

A.set_diagonal({1.,2.}) =
              1             0
              0             2

A =
              1             0
              0             2
```

• `set_superdiagonal(int n,value)` replaces off-diagonal elements in the matrix, specified by `n`, with `value`. If `n` is a double value, then it will be rounded down to an integer.

Example:`set_superdiagonal(std::vector v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(3);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_superdiagonal(1,1.) = " << "\n"
              << A.set_superdiagonal(1,1.) << "\n";
    std::cout << "A.set_superdiagonal(-1,2.) = " << "\n"
              << A.set_superdiagonal(-1,2.);

    return 0;
}
```

Output:

```
A =
              0             0             0
              0             0             0
              0             0             0

A.set_superdiagonal(1,1.) =
              0             1             0
              0             0             1
              0             0             0

A.set_superdiagonal(-1,2.) =
              0             1             0
              2             0             1
              0             2             0
```

• `set_superdiagonal(int n,std::vector v)` replaces off-diagonal elements in the matrix, specified by `n`, with elements in the vector `v`. If `n` is not a double value, then it will be rounded down to an integer.

Example:`set_superdiagonal(int n,std::vector v)`

```
#include <yotcopi.hpp>
```

```
int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(3);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_superdiagonal(1,{1.,2.}) = " << "\n"
              <<  A.set_superdiagonal(1,{1.,2.}) << "\n";
    std::cout << "A.set_superdiagonal(-1,{3.,4.}) = " << "\n"
              << A.set_superdiagonal(-1,{3.,4.});

    return 0;
}
```

Output:

```
A =
              0              0              0
              0              0              0
              0              0              0

A.set_superdiagonal(1,{1.,2.}) =
              0              1              0
              0              0              2
              0              0              0

A.set_superdiagonal(-1,{3.,4.}) =
              0              1              0
              3              0              2
              0              4              0
```

- `set_superdiagonal(std:vector<int> v,value)` replaces off-diagonal elements in the matrix, specified by the vector `v`, with `value`.

Example:`set_superdiagonal(std:vector<int> v,value)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(3);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_superdiagonal({-1,1},1) = " << "\n"
              <<  A.set_superdiagonal({-1,1},1) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
              0              0              0
              0              0              0
              0              0              0

A.set_superdiagonal({-1,1},1) =
              0              1              0
              1              0              1
              0              1              0
```

```
A =
                0               1               0
                1               0               1
                0               1               0
```

- `set_row(int n,value)` replace elements in a row, specified by `n`, with `value`. The first row is referred to as 0.

Example:set_row(int n,value)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_row(0,1) = " << "\n"
              <<  A.set_row(0,1);

    return 0;
}
```

Output:

```
A =
                0               0
                0               0

A.set_row(0,1) =
                1               1
                0               0
```

- `set_row(int n,std::vector v)` replace elements in a row, specified by `n`, with the vector `v`. The first row is referred to as 0.

Example:set_row(int n,std::vector v)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_row(0,{1.,2.}) = " << "\n"
              << A.set_row(0,{1.,2.});

    return 0;
}
```

Output:

```
A =
                0               0
                0               0

A.set_row(0,{1.,2.}) =
                1               2
                0               0
```

• `set_column(int n,value)` replaces elements in the n^{th} column with a vector `value`. The first column is referred as 0.

Example:set_column(int n,value)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_column(0,1) = " << "\n"
              <<  A.set_column(0,1);

    return 0;
}
```

Output:

```
A =
                0               0
                0               0

A.set_column(0,1) =
                1               0
                1               0
```

• `set_column(int n,std::vector v)` replace elements in a column, specified by `n`, with the vector `v`. The first row is referred to as 0.

Example:set_column(int n,std::vector v)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_column(0,{1.,2.}) = " << "\n"
              << A.set_column(0,{1.,2.});

    return 0;
}
```

Output:

```
A =
                0               0
                0               0

A.set_column(0,{1.,2.}) =
                1               0
                2               0
```

• `set_row(int n,std::vector v)` replaces elements in the n^{th} rows by a vector `v`. The first row is referred to as 0.

Example:set_row(int n,std::vector v)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.set_row(0,{1.,2.}) = " << "\n"
              << A.set_row(0,{1.,2.});

    return 0;
}
```

Output:

```
A =
              0             0
              0             0

A.set_row(0,{1.,2.}) =
              1             2
              0             0
```

B. Modifying matrix structures

• `resize(unsigned_int n)` changes the size of a matrix to $n \times n$. If the original matrix is bigger than $n \times n$, then the matrix is truncated, otherwise the matrix is extended with zero elements.

Example:`resize(unsigned_int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    m B({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "B = " << "\n" << B << "\n";
    std::cout << "A.resize(2) = " << "\n" << A.resize(2) << "\n";
    std::cout << "B.resize(3) = " << "\n" << B.resize(3);

    return 0;
}
```

Output:

```
A =
              1             2             3
              4             5             6
              7             8             9

B =
              1             2
              3             4

A.resize(2) =
              1             2
```

```
                4                5

B.resize(3) =
                1                2                0
                3                4                0
                0                0                0
```

- `push_back_row(std::vector v)` inserts a vector `v` into the last row of the matrix.

Example:`push_back_row(std::vector v)`

```
#include <yotcopi.hpp>
int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
  std::cout << "A =" << "\n" << A5 << "\n";
  std::cout << "A.push_back_row({0.,0.,0.}) = " << "\n"
            <<  A.push_back_row({0.,0.,0.});
  return 0;
}
```

Output:

```
A =
                1                2                3
                4                5                6
                7                8                9

A.push_back_row({0.,0.,0.}) =
                1                2                3
                4                5                6
                7                8                9
                0                0                0
```

- `push_back_column(std::vector v)` inserts a vector `v` into the last column of the matrix.

Example:`push_back_column(std::vector v)`

```
#include <yotcopi.hpp>
int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
  std::cout << "A =" << "\n" << A << "\n";
  std::cout << "A.push_back_column({0.,0.,0.}) = " << "\n"
            <<  A.push_back_row({0.,0.,0.});

  return 0;
}
```

Output:

```
A =
                1                2                3
                4                5                6
                7                8                9

A.push_back_column({0.,0.,0.}) =
                1                2                3                0
                4                5                6                0
                7                8                9                0
```

- `push_front_row(std::vector v)` inserts a vector `v` into the first row of the matrix.

Example:`push_front_row(std::vector v)`

```
#include <yotcopi.hpp>
int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
  std::cout << "A =" << "\n" << A << "\n";
  std::cout << "A.push_front_row({0.,0.,0.}) = " << "\n"
            <<  A.push_back_row({0.,0.,0.});

  return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

A.push_front_row({0.,0.,0.}) =
                0               0               0
                1               2               3
                4               5               6
                7               8               9
```

- `push_front_column(std::vector v)` inserts a vector `v` into the first column of the matrix.

Example:`push_front_column(std::vector v)`

```
#include <yotcopi.hpp>
int main(int argc, char** argv)
{
  using namespace yotcopi;
  using namespace yotcopi::shortkeys;

  m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
  std::cout << "A =" << "\n" << A << "\n";
  std::cout << "A.push_front_column({0.,0.,0.}) = " << "\n"
            <<  A.push_back_column({0.,0.,0.});

  return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

A.push_front_column({0.,0.,0.}) =
                0               1               2               3
                0               4               5               6
                0               7               8               9
```

- `pop_back_row()` removes the last row of the matrix. The original matrix is modified.

Example:`pop_back_row()`

```cpp
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.pop_back_row() = " << "\n"
              << A.pop_back_row() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

A.pop_back_row() =
                1               2               3
                4               5               6

A =
                1               2               3
                4               5               6
```

- `pop_back_column()` removes the last column of the matrix. The original matrix is modified.

Example:`pop_back_column()`

```cpp
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.pop_back_column() = " << "\n"
              << A.pop_back_column() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

A.pop_back_column() =
                1               2
                4               5
                7               8

A =
                1               2
```

```
        4               5
        7               8
```

- `pop_front_row()` removes the first row of the matrix. The original matrix is modified.

Example:`pop_front_column()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.pop_front_row() = " << "\n"
              << A.pop_front_row() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
        1               2               3
        4               5               6
        7               8               9

A.pop_front_row() =
        4               5               6
        7               8               9

A =
        4               5               6
        7               8               9
```

- `pop_front_column()` removes the first column of the matrix. The original matrix is modified.

Example:`pop_front_column()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.pop_front_column() = " << "\n"
              << A.pop_front_column() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
        1               2               3
        4               5               6
        7               8               9
```

```
A.pop_front_column() =
                2               3
                5               6
                8               9

A =
                2               3
                5               6
                8               9
```

- `insert_row(int n, std::vector v)` inserts the vector `v` at the n^{th} row.

Example:insert_row(int n, std::vector v)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.insert_row(1,{1.,2.}) = " << "\n"
              << A.insert_row(1,{1.,2.}) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                0               0
                0               0

A.insert_row(1,{1.,2.}) =
                0               0
                1               2
                0               0

A =
                0               0
                1               2
                0               0
```

- `insert_column(int n, std::vector v)` inserts the vector `v` at the n^{th} column.

Example:insert_column(int n, std::vector v)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A(2);

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.insert_column(1,{1.,2.}) = " << "\n"
              << A.insert_column(1,{1.,2.}) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
         0         0
         0         0

A.insert_column(1,{1.,2.}) =
         0         1         0
         0         2         0

A =
         0         1         0
         0         2         0
```

- `erase_row(int n)` removes the n^{th} row.

Example:erase_row(int n)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.erase_row(1) = " << "\n"
              << A.erase_row(1) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
         1         2
         3         4

A.erase_row(1) =
         1         2

A =
         1         2
```

- `erase_column(int n)` removes a the n^{th} column.

Example:erase_column(int n)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.erase_column(1) = " << "\n"
              << A.erase_column(1) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2
                3               4

A.erase_column(1) =
                1
                3

A =
                1
                3
```

- `repeat_rows(int n)` repeats the whole matrix `n` times along the rows.

Example:repeat_rows(int n)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.repeat_rows(2) = " << "\n"
              << A.repeat_rows(2) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2
                3               4

A.repeat_rows(2) =
                1               2
                3               4
                1               2
                3               4

A =
                1               2
                3               4
                1               2
                3               4
```

- `repeat_columns(int n)` repeats the whole matrix `n` times along the columns.

Example:repeat_columns(int n)

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.repeat_columns(2) = " << "\n"
```

```
            << A.repeat_columns(2) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
            1           2
            3           4

A.repeat_columns(2) =
            1           2           1           2
            3           4           3           4

A =
            1           2           1           2
            3           4           3           4
```

• `lag_row(int n)` pushes matrix elements down by `n` rows. 0's are added to keep the matrix's dimension the same. See the following example for illustration.

Example:`lag_row(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.lag_row(2) = " << "\n"
              << A.lag_row(2) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
            1           2           3
            3           4           5
            6           7           8

A.lag_row(2) =
            0           0           0
            0           0           0
            1           2           3

A =
            0           0           0
            0           0           0
            1           2           3
```

• `lag_column(int n)` pushes matrix elements to the right by `n` column. 0's are added to keep the matrix's dimension the same. See the following example for illustration.

Example:`lag_column(int n)`

```
#include <yotcopi.hpp>
```

```
int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.lag_column(2) = " << "\n"
              << A.lag_column(2) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                3               4               5
                6               7               8

A.lag_column(2) =
                0               0               1
                0               0               3
                0               0               6

A =
                0               0               1
                0               0               3
                0               0               6
```

- `lag(int n1,int n2)` performs `lag_row(n1)` and then `lag_column(n2)`.

Example:`lag(int n1,int n2)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.lag({1,2}) = " << "\n"
              << A.lag({1,2}) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                3               4               5
                6               7               8

A.lag({1,2}) =
                0               0               0
                0               0               1
                0               0               3

A =
                0               0               0
```

```
        0               0               1
        0               0               3
```

- `cyclic_lag_row(int n)` pushs rows of the original matrix down cyclicly by n times.

Example:`cyclic_lag_row(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.cyclic_lag_row(2) = " << "\n" << A.cyclic_lag_row(2) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
        1               2               3
        3               4               5
        6               7               8

A.cyclic_lag_row(2) =
        3               4               5
        6               7               8
        1               2               3

A =
        3               4               5
        6               7               8
        1               2               3
```

- `cyclic_lag_column(int n)` pushes columns of the original matrix to the right cyclicly by n times.

Example:`cyclic_lag_column(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.cyclic_lag_column(2) = "
              << "\n" << A.cyclic_lag_column(2) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
        1               2               3
        3               4               5
```

```
                6               7               8

A.cyclic_lag_column(2) =
                2               3               1
                4               5               3
                7               8               6

A =
                2               3               1
                4               5               3
                7               8               6
```

- `cyclic_lag(int n1,int n2)` performs `cyclic_lag_row(int n1)` and then `cyclic_lag_column(n2)`

Example:`cyclic_lag(int n1,int n2)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.cyclic_lag({1,2}) = "
              << "\n" << A.cyclic_lag({1,2}) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                3               4               5
                6               7               8

A.cyclic_lag({1,2}) =
                7               8               6
                2               3               1
                4               5               3

A =
                7               8               6
                2               3               1
                4               5               3
```

- `reverse_row()` reverses the row's orders in the original matrix.

Example:`reverse_row()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.reverse_row() = "
              << "\n" << A.reverse_row() << "\n";
    std::cout << "A = " << "\n" << A;
```

```
    return 0;
}
```

Output:

```
A =
                1               2               3
                3               4               5
                6               7               8

A.reverse_row() =
                6               7               8
                3               4               5
                1               2               3

A =
                6               7               8
                3               4               5
                1               2               3
```

- `reverse_column()` reverses the column's orders in the original matrix.

Example:`reverse_column()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.reverse_column() = "
              << "\n" << A.reverse_column() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                3               4               5
                6               7               8

A.reverse_column() =
                3               2               1
                5               4               3
                8               7               6

A =
                3               2               1
                5               4               3
                8               7               6
```

- `reverse()` reverses the row's and column's orders in the original matrix.

Example:`reverse()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
```

```
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.reverse() = "
              << "\n" << A.reverse() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
              1              2              3
              3              4              5
              6              7              8

A.reverse() =
              8              2              6
              5              4              3
              3              7              1

A =
              8              2              6
              5              4              3
              3              7              1
```

- `reflect_row()` reverses the row's orders in the original matrix.

Example:`reflect_row()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.reflect_row() = "
              << "\n" << A.reflect_row() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
              1              2              3
              3              4              5
              6              7              8

A.reflect_row() =
              6              7              8
              3              4              5
              1              2              3

A =
              6              7              8
              3              4              5
              1              2              3
```

- `reflect_column()` reverses the column's orders in the original matrix.

Example:`reflect_column()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.reflect_column() = "
              << "\n" << A.reflect_column() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
              1              2              3
              3              4              5
              6              7              8

A.reflect_column() =
              3              2              1
              5              4              3
              8              7              6

A =
              3              2              1
              5              4              3
              8              7              6
```

- `reflect()` reverses the row's and the column's orders in the original matrix.

Example:`reflect()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.reflect() = "
              << "\n" << A.reflect() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
              1              2              3
              3              4              5
              6              7              8

A.reflect() =
```

```
                8               2               6
                5               4               3
                3               7               1

A =
                8               2               6
                5               4               3
                3               7               1
```

- `mirror_row()` reverses the row's orders in the original matrix.

Example:`mirror_row()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.mirror_row() = "
              << "\n" << A.mirror_row() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                3               4               5
                6               7               8

A.mirror_row() =
                6               7               8
                3               4               5
                1               2               3

A =
                6               7               8
                3               4               5
                1               2               3
```

- `mirror_column()` reverses the column's orders in the original matrix.

Example:`mirror_column()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.mirror_column() = "
              << "\n" << A.mirror_column() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                3               4               5
                6               7               8

A.mirror_column() =
                3               2               1
                5               4               3
                8               7               6

A =
                3               2               1
                5               4               3
                8               7               6
```

- `mirror()` reverses the row's and the column's orders in the original matrix.

Example:`mirror()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{3.,4.,5.},{6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.reflect() = "
              << "\n" << A.reflect() << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                3               4               5
                6               7               8

A.mirror() =
                8               2               6
                5               4               3
                3               7               1

A =
                8               2               6
                5               4               3
                3               7               1
```

- `move_rows_from(int n)` gives a matrix containing rows from the n^{th} row to the end. The original matrix is left with the remaining rows.

Example:`move_rows_from(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.},{5.,6.},{7.,8.}});
```

```
    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.move_rows_from(2) = "
              << "\n" << A.move_rows_from(2) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2
                3               4
                5               6
                7               8

A.move_rows_from(2) =
                5               6
                7               8

A =
                1               2
                3               4
```

• `move_columns_from(int n)` gives a new matrix containing columns from the n^{th} column to the end. The original matrix is left with the remaining columns.

Example:`move_columns_from(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.,4.},{5.,6.,7.,8.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "A.move_columns_from(2) = "
              << "\n" << A.move_columns_from(2) << "\n";
    std::cout << "A = " << "\n" << A;

    return 0;
}
```

Output:

```
A =
                1               2               3               4
                5               6               7               8

A.move_columns_from(2) =
                3               4
                7               8

A =
                1               2
                5               6
```

VIII. TRANSFORM OPERATIONS

• `resize_copy(int n)` gives a new matrix of size $n \times n$. The output matrix has the same elements as the orginal matrix. The original matrix is left unchanged.

Example:`resize_copy(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.resize_copy(2);
    std::cout << "B = " << "\n" << B << "\n";

    B = 0;
    std::cout << "B is now changed to zero, while A is unchanged" << "\n";
    std::cout << "A = " << "\n" << A;
    std::cout << "B = " << "\n" << B;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                1               2
                4               5

B is now changed to zero, while A is unchanged
A =
                1               2               3
                4               5               6
                7               8               9
B =
                0               0
                0               0
```

• `push_back_row_copy(std::vector v)` gives a new matrix. The output matrix is the same as the original matrix with an extra row `v` attached to the end. The original matrix is left unchanged.

Example:`push_back_row_copy(std::vector v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.push_back_row_copy({6.,6.});
    std::cout << "B = " << "\n" << B << "\n";

    B = 0;
    std::cout << "B is now changed to zero, while A is unchanged" << "\n";
```

```
    std::cout << "A = " << "\n" << A;
    std::cout << "B = " << "\n" << B;
    return 0;
}
```

Output:

```
A =
                1               2
                3               4

B =
                1               2
                3               4
                6               6

B is now changed to zero, while A is unchanged
A =
                1               2
                3               4
B =
                0               0
                0               0
                0               0
```

• `push_back_column_copy (std::vector v)` gives a new matrix. The output matrix is the same as the original matrix with an extra column `v` attached to the right end. The original matrix is left unchanged.

Example:`push_back_column_copy(std::vector v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.push_back_column_copy({6.,6.});
    std::cout << "B = " << "\n" << B << "\n";

    B = 0;
    std::cout << "B is now changed to zero, while A is unchanged" << "\n";
    std::cout << "A = " << "\n" << A;
    std::cout << "B = " << "\n" << B;
    return 0;
}
```

Output:

```
A =
                1               2
                3               4

B =
                1               2               6
                3               4               6

B is now changed to zero, while A is unchanged
A =
                1               2
                3               4
B =
                0               0               0
                0               0               0
```

• `push_front_row_copy(std::vector v)` gives a new matrix. The output matrix is the same as the original matrix with an extra row `v` attached to the top. The original matrix is left unchanged.

Example:`push_front_row_copy(std::vector v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.push_front_row_copy({6.,6.});
    std::cout << "B = " << "\n" << B << "\n";

    B = 0;
    std::cout << "B is now changed to zero, while A is unchanged" << "\n";
    std::cout << "A = " << "\n" << A;
    std::cout << "B = " << "\n" << B;
    return 0;
}
```

Output:

```
A =
                1               2
                3               4

B =
                6               6
                1               2
                3               4

B is now changed to zero, while A is unchanged
A =
                1               2
                3               4
B =
                0               0
                0               0
                0               0
```

• `push_front_column_copy(std::vector v)` gives a new matrix. The output matrix is the same as the original matrix with an extra column `v` attached to the front. The original matrix is left unchanged.

Example:`push_front_column_copy(std::vector v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.push_front_column_copy({6.,6.});
    std::cout << "B = " << "\n" << B << "\n";

    B = 0;
    std::cout << "B is now changed to zero, while A is unchanged" << "\n";
    std::cout << "A = " << "\n" << A;
    std::cout << "B = " << "\n" << B;
    return 0;
```

```
}
```

Output:

```
A =
                1               2
                3               4

B =
                6               1               2
                6               3               4

B is now changed to zero, while A is unchanged
A =
                1               2
                3               4
B =
                0               0               0
                0               0               0
```

• `pop_back_row_copy()` gives a new matrix. The output matrix is the same as the original matrix with the last row excluded. The original matrix is left unchanged.

Example:`pop_back_row_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.pop_back_row_copy();
    std::cout << "B = " << "\n" << B << "\n";

    B = 0;
    std::cout << "B is now changed to zero, while A is unchanged" << "\n";
    std::cout << "A = " << "\n" << A;
    std::cout << "B = " << "\n" << B;
    return 0;

    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                1               2               3
                4               5               6
B is now changed to zero, while A is unchanged
A =
                1               2               3
                4               5               6
                7               8               9
B =
                0               0               0
                0               0               0
```

• `pop_back_column_copy()` gives a new matrix. The output matrix is the same as the original matrix with the last column excluded. The original matrix is left unchanged.

Example:`pop_back_column_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.pop_back_column_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = "<< "\n"  << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                1               2
                4               5
                7               8
A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `pop_front_row_copy()` gives a new matrix. The output matrix is the same as the original matrix with the first row excluded. The original matrix is left unchanged.

Example:`pop_front_row_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.pop_front_row_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = "<< "\n"  << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                4               5               6
                7               8               9

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `pop_front_column_copy()` gives a new matrix. The output matrix is the same as the original matrix with the first column excluded. The original matrix is left unchanged.

Example:`pop_front_column_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.pop_front_column_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = "<< "\n"  << A;

    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                2               3
                5               6
                8               9

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `insert_row_copy(unsigned_int n,std::vector v)` gives a new matrix. The output matrix is the same as the original matrix with a vector v inserted at the row n. The original matrix is left unchanged.

Example:`insert_row_copy(int_n,std::vector v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
```

```
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.insert_row_copy(1,{0.,0.});
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2
                3               4

B =
                1               2
                0               0
                3               4

A is unchanged:
A =
                1               2
                3               4
```

• `insert_column_copy(unsigned_int n,std::vector v)` gives a new matrix. The output matrix is the same as the original matrix with a vector v inserted at the column n. The original matrix is left unchanged.

Example:`insert_column_copy(unsigned_int n,std::vector v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.insert_column_copy(1,{0.,0.});
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2
                3               4

B =
                1               0               2
                3               0               4

A is unchanged:
A =
                1               2
                3               4
```

• `erase_row_copy(unsigned_int n)` gives a new matrix. The output matrix is the same as the original matrix with the n^{th} row erased. The original matrix is left unchanged.

Example:`erase_row_copy(unsigned_int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.erase_row_copy(0);
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2
                3               4

B =
                3               4

A is unchanged:
A =
                1               2
                3               4
```

• `erase_column_copy(unsigned_int n)` gives a new matrix. The output matrix is the same as the original matrix with the n^{th}column erased. The original matrix is left unchanged.

Example:`erase_column_copy(unsigned_int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.erase_column_copy(0);
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2
                3               4

B =
```

```
                2
                4

A is unchanged:
A =
                1               2
                3               4
```

• `repeat_rows_copy(unsigned_int n)` gives a new matrix. The output matrix is the same as the original matrix but repeated n times vertically. The original matrix is left unchanged.

Example:`repeat_rows_copy(unsigned_int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.repeat_rows_copy(2);
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2
                3               4

B =
                1               2
                3               4
                1               2
                3               4

A is unchanged:
A =
                1               2
                3               4
```

• `repeat_columns_copy(unsigned_int n)` gives a new matrix. The output matrix is the same as the original matrix but repeated n times horizontally. The original matrix is left unchanged.

Example:`repeat_columns_copy(unsigned_int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.},{3.,4.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.repeat_columns_copy(2);
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
```

```
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2

                3               4

B =
                1               2               1               2
                3               4               3               4

A is unchanged:
A =
                1               2
                3               4
```

• `swap_rows _copy(unsigned_int n1, unsigned_int n2)` gives a new matrix. The output matrix is the same as the original matrix with the rows $n1$ and $n2$ swapped. The original matrix is left unchanged.

Example:`swap_rows _copy(unsigned_int n1, unsigned_int n2)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.swap_rows_copy(1,2);
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                1               2               3
                7               8               9
                4               5               6

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `swap_columns _copy(unsigned_int n1, unsigned_int n2)` gives a new matrix. The output matrix is the same as the original matrix with the columns $n1$ and $n2$ swapped. The original matrix is left unchanged.

Example:`swap_columns _copy(unsigned_int n1, unsigned_int n2)`

```
#include <yotcopi.hpp>
```

```
int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.swap_columns_copy(1,2);
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                1               3               2
                4               6               5
                7               9               8

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `lag_row_copy(int n)` gives a new matrix. The output matrix has the same dimension same as the original matrix but its elements being shifted vertically by n rows. The original matrix is left unchanged. See the following example for illustration.

Example:`repeat_row_copy(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.lag_row_copy(1);
    std::cout << "B = " << "\n" << B << "\n";

    m C = A.lag_row_copy(-1);
    std::cout << "C = " << "\n" << C << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
```

```
                7               8               9

B =
                0               0               0
                1               2               3
                4               5               6

C =
                4               5               6
                7               8               9
                0               0               0

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `lag_column_copy(int n)` gives a new matrix. The output matrix has the same dimension same as the original matrix but its elements being shifted horizontally by n rows. The original matrix is left unchanged. See the following example for illustration.

Example:`repeat_columns_copy(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.lag_column_copy(1);
    std::cout << "B = " << "\n" << B << "\n";

    m C = A.lag_column_copy(-1);
    std::cout << "C = " << "\n" << C << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                0               1               2
                0               4               5
                0               7               8

C =
                2               3               0
                5               6               0
                8               9               0

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `lag_copy(std::vector<int> v)` gives a new matrix. The output matrix has the same dimension same as the original matrix but its elements being shifted vertically by $v(0)$ rows and horizontally by $v(1)$ columns. The original matrix is left unchanged. See the following example for illustration.

Example:`lag_copy(std::vector<int> v)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.lag_copy({1,2});
    std::cout << "B = " << "\n" << B << "\n";

    m C = A.lag_copy({-1,-2});
    std::cout << "C = " << "\n" << C << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                0               0               0
                0               0               1
                0               0               4

C =
                6               0               0
                9               0               0
                0               0               0

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `cyclic_lag_row_copy(int n)` gives a new matrix. The output matrix is the same as the original matrix with the rows permuted by n times. The original matrix is left unchanged.

Example:`cyclic_lag_row_copy(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.cyclic_lag_row_copy(1);
    std::cout << "B = " << "\n" << B << "\n";
```

```
    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                7               8               9
                1               2               3
                4               5               6

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `cyclic_lag_column_copy(int n)` gives a new matrix. The output matrix is the same as the original matrix with the columns permuted by n times. The original matrix is left unchanged.

Example:`cyclic_lag_column_copy(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.cyclic_lag_column_copy(1);
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                3               1               2
                6               4               5
                9               7               8

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `cyclic_lag_copy(std::vector<int> v)` gives a new matrix. The output matrix is the same as the original matrix with the rows permuted by $v(0)$ times and the columns permuted by $v(1)$. The original matrix is left unchanged.

Example:`cyclic_lag_copy(int n)`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.cyclic_lag_copy({1,2});
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                8               9               7
                2               3               1
                5               6               4

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `reverse_row_copy()` gives a new matrix

Example:`reverse_row_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.reverse_row_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9
```

```
B =
                7               8               9
                4               5               6
                1               2               3

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `reverse_row_copy()` gives a new matrix with reversed rows.

Example:reverse_row_copy()

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.reverse_row_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                7               8               9
                4               5               6
                1               2               3

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `reverse_column_copy()` gives a new matrix with the reversed column.

Example:reverse_column_copy()

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.reverse_column_copy();
    std::cout << "B = " << "\n" << B << "\n";
```

```
    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                3               2               1
                6               5               4
                9               8               7

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `reverse_copy()` gives a new matrix with the reversed rows and the reversed columns.

Example:`reverse_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.reverse_column_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                9               8               7
                6               5               4
                3               2               1

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `reflect_row_copy()` gives a new matrix with the reversed rows.

Example:`reflect_row_copy()`

```cpp
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.reflect_row_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                7               8               9
                4               5               6
                1               2               3

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `reflect_column_copy()` gives a new matrix with the reversed columns.

Example:`reflect_column_copy()`

```cpp
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.reflect_column_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
```

```
                3               2               1
                6               5               4
                9               8               7

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `reflect_copy()` gives a new matrix with the reversed rows and the reversed columns.

Example:`reflect_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.reflect_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                9               8               7
                6               5               4
                3               2               1

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `mirror_row _copy ()` gives a new matrix with the reversed rows.

Example:`mirror_row_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.mirror_row_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
```

```
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                7               8               9
                4               5               6
                1               2               3

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `mirror_column_copy()` gives a new matrix with the reversed columns.

Example:`mirror_column_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.mirror_column_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                3               2               1
                6               5               4
                9               8               7

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

- `mirror_copy()` gives a new matrix with the reversed rows and the reversed columns.

Example:`mirror_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.mirror_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                9               8               7
                6               5               4
                3               2               1

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `transpose_copy()` gives a new matrix which is the transpose of the original matrix. The original matrix is left unchanged.

Example:`transpose_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.transpose_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
```

```
                1               4               7
                2               5               8
                3               6               9

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

• `conjugate_copy()` gives a new matrix which is the conjugate of the original matrix. The original matrix is left unchanged.

Example:`transpose_copy()`

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}});
    std::cout << "A = " << "\n" << A << "\n";

    m B = A.conjugate_copy();
    std::cout << "B = " << "\n" << B << "\n";

    std::cout <<"A is unchanged:" << "\n";
    std::cout <<"A = " << "\n" << A;
    return 0;
}
```

Output:

```
A =
                1               2               3
                4               5               6
                7               8               9

B =
                1               4               7
                2               5               8
                3               6               9

A is unchanged:
A =
                1               2               3
                4               5               6
                7               8               9
```

IX. MATRIX COMPARISON

A. comparing the whole matrices

The functions in this section return 1 if "every elements" in the matrix satify a given condition and 0 otherwise. The available functions are the following.

- `is_equal_to(x)`: If x is a number, then the function checks whether every elements in the matrix are x. If x is a matrix, then the function checks whether two matrices are the same.
- `is_not_equal_to(x)`: This function is the same as `is_equal_to(x)` but returns 1 if they are not equal.

- `is_equal_to_within(x,e)`: This function is the same as `is_equal_to(x)` but still returns 1 if the differences are within e.
- `is_not _equal _within(x,e)`: This function is the same as `is_equal_to(x)` but returns 0 if the differences are within e.

Example:comparing the matrices

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,1.},{1.,1.}});
    m B({{1.1,1.2},{1.3,1.4}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "B = " << "\n" << B << "\n\n";

    std::cout << "A.is_equal_to(1.) = " << A.is_equal_to(1.) << "\n";
    std::cout << "A.is_not_equal_to(1.) = " << A.is_not_equal_to(1.) << "\n\n";

    std::cout << "A.is_equal_to(B) = " << A.is_equal_to(B) << "\n";
    std::cout << "A.is_not_equal_to(B) = " << A.is_not_equal_to(B) << "\n\n";

    double e=1.;
    std::cout << "B.is_equal_to_within(1.,1.) = " << B.is_equal_to_within(1.,e) << "\n";
    std::cout << "B.is_not_equal_to_within(1.,1.) = " << B.is_not_equal_to_within(1.,e) << "\n\n
        ";

    std::cout << "B.is_equal_to_within(A,1.) = " << B.is_equal_to_within(A,e) << "\n";
    std::cout << "B.is_not_equal_to_within(A,1.) = " << B.is_not_equal_to_within(A,e) << "\n\n";
    return 0;
}
```

Output:

```
A =
                1                1
                1                1

B =
              1.1              1.2
              1.3              1.4


A.is_equal_to(1.) = 1
A.is_not_equal_to(1.) = 0

A.is_equal_to(B) = 0
A.is_not_equal_to(B) = 1

B.is_equal_to_within(1.,1.) = 1
B.is_not_equal_to_within(1.,1.) = 0

B.is_equal_to_within(A,1.) = 1
B.is_not_equal_to_within(A,1.) = 0
```

B. comparing elements in the matrices

The operators in this section return a bool matrix, i.e. its elements are either 1 if such element satify a given condition or 0 otherwise. The available operators are the following.

- == equal to
- ! = not equal to
- <= less than or equal to
- >= more than or equal to
- < less than
- > more than

Example:comparing elements in the matrices

```
#include <yotcopi.hpp>

int main(int argc, char** argv)
{
    using namespace yotcopi;
    using namespace yotcopi::shortkeys;

    m A({{1.,1.},{1.,1.}});
    m B({{1.,2.},{3.,4.}});

    std::cout << "A = " << "\n" << A << "\n";
    std::cout << "B = " << "\n" << B << "\n\n";

    std::cout << "A == 1 : " << "\n" << (A==1) << "\n";
    std::cout << "A == B : " << "\n" <<  (A==B) << "\n";

    std::cout << "A != 1 : " << "\n" << (A!=1) << "\n";
    std::cout << "A != B : " << "\n" <<  (A!=B) << "\n";

    std::cout << "A <= 1 : " << "\n" << (A<=1) << "\n";
    std::cout << "A <= B : " << "\n" <<  (A<=B) << "\n";

    std::cout << "A >= 1 : " << "\n" << (A>=1) << "\n";
    std::cout << "A >= B : " << "\n" <<  (A>=B) << "\n";

    std::cout << "A < 1 : " << "\n" << (A<1) << "\n";
    std::cout << "A < B : " << "\n" <<  (A<B) << "\n";

    std::cout << "A > 1 : " << "\n" << (A>1) << "\n";
    std::cout << "A > B : " << "\n" <<  (A>B) << "\n";

    return 0;
}
```

Output:

```
A =
                1               1
                1               1

B =
                1               2
                3               4


A == 1 :
                1               1
                1               1

A == B :
                1               0
                0               0

A != 1 :
```

```
                0               0
                0               0

A != B  :
                0               1
                1               1

A <= 1  :
                1               1
                1               1

A <= B  :
                1               1
                1               1

A >= 1  :
                1               1
                1               1

A >= B  :
                1               0
                0               0

A < 1  :
                0               0
                0               0

A < B  :
                0               1
                1               1

A > 1  :
                0               0
                0               0

A > B  :
                0               0
                0               0
```