

matOps

Generated by Doxygen 1.9.1

<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>2</b>
2.1 File List	2
<b>3 Class Documentation</b>	<b>3</b>
3.1 Matrix Class Reference	3
3.1.1 Detailed Description	4
3.1.2 Constructor & Destructor Documentation	4
3.1.2.1 Matrix() [1/2]	5
3.1.2.2 Matrix() [2/2]	5
3.1.3 Member Function Documentation	5
3.1.3.1 determinant()	6
3.1.3.2 extractMatrix()	6
3.1.3.3 hStack()	7
3.1.3.4 insertCol() [1/2]	7
3.1.3.5 insertCol() [2/2]	8
3.1.3.6 insertRow() [1/2]	8
3.1.3.7 insertRow() [2/2]	9
3.1.3.8 inverse()	9
3.1.3.9 mean()	9
3.1.3.10 operator>()	10
3.1.3.11 operator*() [1/2]	10
3.1.3.12 operator*() [2/2]	11
3.1.3.13 operator+() [1/2]	11
3.1.3.14 operator+() [2/2]	12
3.1.3.15 operator-() [1/2]	12
3.1.3.16 operator-() [2/2]	13
3.1.3.17 operator/()	13
3.1.3.18 operator==()	14
3.1.3.19 operator^()	14
3.1.3.20 shape()	14
3.1.3.21 sum()	15
3.1.3.22 transpose()	15
3.1.3.23 vStack()	15
3.1.4 Friends And Related Function Documentation	16
3.1.4.1 operator*	16
3.1.4.2 operator+	16
3.1.4.3 operator-	17
3.1.4.4 operator<<	17
<b>4 File Documentation</b>	<b>19</b>

---

4.1 src/matOps.hpp File Reference . . . . .	19
4.1.1 Function Documentation . . . . .	19
4.1.1.1 operator<<() . . . . .	19
4.2 test/testMatrix.cpp File Reference . . . . .	20
4.2.1 Detailed Description . . . . .	21
4.2.2 Macro Definition Documentation . . . . .	21
4.2.2.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN . . . . .	21
4.2.3 Function Documentation . . . . .	21
4.2.3.1 TEST_CASE() [1/22] . . . . .	21
4.2.3.2 TEST_CASE() [2/22] . . . . .	21
4.2.3.3 TEST_CASE() [3/22] . . . . .	21
4.2.3.4 TEST_CASE() [4/22] . . . . .	22
4.2.3.5 TEST_CASE() [5/22] . . . . .	22
4.2.3.6 TEST_CASE() [6/22] . . . . .	22
4.2.3.7 TEST_CASE() [7/22] . . . . .	22
4.2.3.8 TEST_CASE() [8/22] . . . . .	23
4.2.3.9 TEST_CASE() [9/22] . . . . .	23
4.2.3.10 TEST_CASE() [10/22] . . . . .	23
4.2.3.11 TEST_CASE() [11/22] . . . . .	23
4.2.3.12 TEST_CASE() [12/22] . . . . .	23
4.2.3.13 TEST_CASE() [13/22] . . . . .	24
4.2.3.14 TEST_CASE() [14/22] . . . . .	24
4.2.3.15 TEST_CASE() [15/22] . . . . .	24
4.2.3.16 TEST_CASE() [16/22] . . . . .	24
4.2.3.17 TEST_CASE() [17/22] . . . . .	25
4.2.3.18 TEST_CASE() [18/22] . . . . .	25
4.2.3.19 TEST_CASE() [19/22] . . . . .	25
4.2.3.20 TEST_CASE() [20/22] . . . . .	25
4.2.3.21 TEST_CASE() [21/22] . . . . .	25
4.2.3.22 TEST_CASE() [22/22] . . . . .	25

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Matrix</a>	A simple linear algebra library for matrix operations . . . . .	<a href="#">3</a>
------------------------	---	-------------------

## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">matOps.hpp</a> . . . . .	19
test/ <a href="#">testMatrix.cpp</a>	
Unit tests for the <a href="#">Matrix</a> class using doctest . . . . .	20

## Chapter 3

# Class Documentation

### 3.1 Matrix Class Reference

A simple linear algebra library for matrix operations.

```
#include <matOps.hpp>
```

#### Public Member Functions

- `std::pair< size_t, size_t > shape () const`  
*Returns the dimensions of the matrix.*
- `Matrix (const std::vector< std::vector< double >> &container)`  
*Constructs a [Matrix](#) from a given 2D vector container.*
- `Matrix (size_t rows, size_t cols, double initialValue)`  
*Constructs a [Matrix](#) with specified dimensions and an initial value.*
- `Matrix operator+ (const Matrix &other) const`  
*Adds two matrices element-wise.*
- `Matrix operator+ (double scalar) const`  
*Adds a scalar value to each element of the matrix. ( $MATRIX + K$ )*
- `Matrix operator- (const Matrix &other) const`  
*Subtracts one matrix from another element-wise.*
- `Matrix operator- (double scalar) const`  
*Subtracts a scalar from each element of the matrix. ( $MATRIX - K$ )*
- `bool operator== (const Matrix &other) const`  
*Compares two matrices for equality.*
- `Matrix operator* (const Matrix &other) const`  
*Multiplies two matrices.*
- `Matrix operator* (double scalar) const`  
*Multiplies each element of the matrix by a scalar. ( $MATRIX * K$ )*
- `Matrix operator/ (double scalar) const`  
*Divides each element of the matrix by a scalar.*
- `Matrix operator^ (double scalar) const`
- `double & operator() (size_t row, size_t col)`  
*Accesses an element of the matrix at a specified row and column.*
- `Matrix transpose () const`

- Transposes the matrix.*
- double `determinant ()` const  
*Computes the determinant of the matrix.*
- `Matrix inverse ()` const  
*Computes the inverse of the matrix.*
- `Matrix insertRow (std::vector< double > row, size_t idx)` const  
*Inserts a new row into the matrix.*
- `Matrix insertRow (double rowVal, size_t idx)` const  
*Inserts a new row filled with a constant value into the matrix.*
- `Matrix insertCol (std::vector< double > col, size_t idx)` const  
*Inserts a new column into the matrix.*
- `Matrix insertCol (double colVal, size_t idx)` const  
*Inserts a new column filled with a constant value into the matrix.*
- `Matrix hStack (const Matrix &other)` const  
*Horizontally concatenates two matrices.*
- `Matrix vStack (const Matrix &other)` const  
*Vertically concatenates two matrices.*
- `Matrix extractMatrix (std::pair< size_t, size_t > rowSlice, std::pair< size_t, size_t > colSlice)` const  
*Extracts a submatrix from the current Matrix.*
- double `sum ()` const  
*Computes the sum of the elements of a vector.*
- double `mean ()` const  
*Computes the mean (average) of the elements of a vector.*

## Friends

- `Matrix operator+` (double scalar, const `Matrix` &other)  
*Adds a scalar to each element of a matrix. ( $K + MATRIX$ )*
- `Matrix operator-` (double scalar, const `Matrix` &other)  
*Subtracts each element of the matrix from a scalar. ( $K - MATRIX$ )*
- `Matrix operator*` (double scalar, const `Matrix` &other)  
*Multiplies a scalar by a matrix. ( $K * MATRIX$ )*
- `std::ostream & operator<<` (std::ostream &os, const `Matrix` &m)  
*Outputs the matrix to an output stream.*

### 3.1.1 Detailed Description

A simple linear algebra library for matrix operations.

This class provides basic matrix operations such as addition, subtraction, multiplication, transposition, determinant calculation, inversion, and row/column insertion.

Example Usage:

```
Matrix A({{1, 2}, {3, 4}});
Matrix B({{5, 6}, {7, 8}});
Matrix C = A + B; // Matrix addition
Matrix D = A * B; // Matrix multiplication
double detA = A.determinant(); // Determinant calculation
```

### 3.1.2 Constructor & Destructor Documentation

**3.1.2.1 Matrix()** [1/2]

```
Matrix::Matrix (
    const std::vector< std::vector< double >> & container ) [inline]
```

Constructs a [Matrix](#) from a given 2D vector container.

**Parameters**

<i>container</i>	A 2D vector of doubles representing the matrix.
------------------	---

**Exceptions**

<i>std::invalid_argument</i>	if the container is empty or row sizes are inconsistent.
------------------------------	--

**Note**

The shape is determined by the size of the container.

**3.1.2.2 Matrix()** [2/2]

```
Matrix::Matrix (
    size_t rows,
    size_t cols,
    double initialValue ) [inline]
```

Constructs a [Matrix](#) with specified dimensions and an initial value.

**Parameters**

<i>rows</i>	Number of rows.
<i>cols</i>	Number of columns.
<i>initialValue</i>	The value to initialize each element.

**Exceptions**

<i>std::invalid_argument</i>	if rows or cols are 0.
------------------------------	------------------------

**Note**

The shape is (rows x cols).

**3.1.3 Member Function Documentation**



### 3.1.3.1 determinant()

```
double Matrix::determinant ( ) const [inline]
```

Computes the determinant of the matrix.

#### Returns

The determinant as a double.

#### Exceptions

<code>std::invalid_argument</code>	if the matrix is not square.
------------------------------------	------------------------------

#### Note

The shape of the matrix remains unchanged.

### 3.1.3.2 extractMatrix()

```
Matrix Matrix::extractMatrix (
    std::pair< size_t, size_t > rowSlice,
    std::pair< size_t, size_t > colSlice ) const [inline]
```

Extracts a submatrix from the current [Matrix](#).

Given a pair of row indices and a pair of column indices, this function creates and returns a new [Matrix](#) containing the submatrix defined by the specified ranges. Both row and column ranges are inclusive, meaning that the elements at both the start and end indices are included in the result.

#### Parameters

<i>rowSlice</i>	A <code>std::pair&lt;size_t, size_t&gt;</code> representing the start and end row indices (inclusive).
<i>colSlice</i>	A <code>std::pair&lt;size_t, size_t&gt;</code> representing the start and end column indices (inclusive).

#### Returns

A new [Matrix](#) object containing the extracted submatrix.

#### Exceptions

<code>std::out_of_range</code>	If any indices are out of bounds or if the slice ranges are invalid.
--------------------------------	--

#### Note

Indices are zero-based (i.e., valid indices range from 0 to `size() - 1`).

### 3.1.3.3 hStack()

```
Matrix Matrix::hStack (
    const Matrix & other ) const [inline]
```

Horizontally concatenates two matrices.

This function creates and returns a new [Matrix](#) by appending the columns of the given matrix to the right of the calling matrix. Both matrices must have the same number of rows.

#### Parameters

<i>other</i>	The <a href="#">Matrix</a> whose columns will be appended to the calling matrix.
--------------	--

#### Returns

A new [Matrix](#) representing the horizontal concatenation of the two matrices.

#### Exceptions

<i>std::invalid_argument</i>	if the two matrices do not have the same number of rows.
------------------------------	--

#### Note

This implementation reserves the necessary capacity before inserting to minimize reallocations.

### 3.1.3.4 insertCol() [1/2]

```
Matrix Matrix::insertCol (
    double colVal,
    size_t idx ) const [inline]
```

Inserts a new column filled with a constant value into the matrix.

#### Parameters

<i>colVal</i>	The constant value to fill the new column.
<i>idx</i>	The index at which to insert the column.

#### Returns

A new [Matrix](#) with the column inserted.

#### Exceptions

<i>std::invalid_argument</i>	if <i>idx</i> is out of range.
------------------------------	--------------------------------

### 3.1.3.5 insertCol() [2/2]

```
Matrix Matrix::insertCol (  
    std::vector< double > col,  
    size_t idx ) const [inline]
```

Inserts a new column into the matrix.

#### Parameters

<i>col</i>	A vector representing the new column.
<i>idx</i>	The index at which to insert the column.

#### Returns

A new [Matrix](#) with the column inserted.

#### Exceptions

<i>std::invalid_argument</i>	if the column size is inconsistent or if idx is out of range.
------------------------------	---

### 3.1.3.6 insertRow() [1/2]

```
Matrix Matrix::insertRow (  
    double rowVal,  
    size_t idx ) const [inline]
```

Inserts a new row filled with a constant value into the matrix.

#### Parameters

<i>rowVal</i>	The constant value to fill the new row.
<i>idx</i>	The index at which to insert the row.

#### Returns

A new [Matrix](#) with the row inserted.

#### Exceptions

<i>std::invalid_argument</i>	if idx is out of range.
------------------------------	-------------------------

### 3.1.3.7 insertRow() [2/2]

```
Matrix Matrix::insertRow (
    std::vector< double > row,
    size_t idx ) const [inline]
```

Inserts a new row into the matrix.

#### Parameters

<i>row</i>	A vector representing the new row.
<i>idx</i>	The index at which to insert the row.

#### Returns

A new [Matrix](#) with the row inserted.

#### Exceptions

<i>std::invalid_argument</i>	if the row size is inconsistent or if idx is out of range.
------------------------------	--

### 3.1.3.8 inverse()

```
Matrix Matrix::inverse ( ) const [inline]
```

Computes the inverse of the matrix.

#### Returns

A new [Matrix](#) representing the inverse.

#### Exceptions

<i>std::runtime_error</i>	if the matrix is singular (non-invertible).
---------------------------	---

#### Note

The shape remains unchanged.

### 3.1.3.9 mean()

```
double Matrix::mean ( ) const [inline]
```

Computes the mean (average) of the elements of a vector.

Calculates the mean value for matrices that are considered as vectors. It supports both row vectors (1 x K) and column vectors (K x 1) by dividing the sum of the elements by the number of elements in the vector.

**Returns**

The mean (average) value of the vector elements.

**Exceptions**

<code>std::invalid_argument</code>	If the matrix is not a one-dimensional vector.
------------------------------------	--

**3.1.3.10 operator()()**

```
double& Matrix::operator() (
    size_t row,
    size_t col ) [inline]
```

Accesses an element of the matrix at a specified row and column.

**Parameters**

<i>row</i>	The row index.
<i>col</i>	The column index.

**Returns**

Reference to the value at the specified position. (modifiable)

**Exceptions**

<code>std::out_of_range</code>	if the indices are out of bounds.
--------------------------------	-----------------------------------

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

```
Matrix Matrix::operator* (
    const Matrix & other ) const [inline]
```

Multiplies two matrices.

**Parameters**

<i>other</i>	The <a href="#">Matrix</a> to multiply with.
--------------	--

**Returns**

A new [Matrix](#) resulting from matrix multiplication.

## Exceptions

<code>std::invalid_argument</code>	if the number of columns of the first matrix does not match the number of rows of the second.
------------------------------------	---

## Note

The shape of the result is (nrows of first, ncols of second).

**3.1.3.12 operator\*() [2/2]**

```
Matrix Matrix::operator* (
    double scalar ) const [inline]
```

Multiplies each element of the matrix by a scalar. (MATRIX \* K)

## Parameters

<i>scalar</i>	The scalar value.
---------------	-------------------

## Returns

A new [Matrix](#) with each element multiplied by the scalar.

## Note

The shape remains unchanged.

**3.1.3.13 operator+() [1/2]**

```
Matrix Matrix::operator+ (
    const Matrix & other ) const [inline]
```

Adds two matrices element-wise.

## Parameters

<i>other</i>	The <a href="#">Matrix</a> to add.
--------------	------------------------------------

## Returns

A new [Matrix](#) representing the element-wise sum.

### Exceptions

<code>std::invalid_argument</code>	if the dimensions of the two matrices do not match.
------------------------------------	---

### Note

The shape remains unchanged.

#### 3.1.3.14 `operator+()` [2/2]

```
Matrix Matrix::operator+ (
    double scalar ) const [inline]
```

Adds a scalar value to each element of the matrix. (MATRIX + K)

### Parameters

<i>scalar</i>	A double value to add.
---------------	------------------------

### Returns

A new [Matrix](#) with the scalar added to each element.

### Note

The shape remains unchanged.

#### 3.1.3.15 `operator-()` [1/2]

```
Matrix Matrix::operator- (
    const Matrix & other ) const [inline]
```

Subtracts one matrix from another element-wise.

### Parameters

<i>other</i>	The <a href="#">Matrix</a> to subtract.
--------------	---

### Returns

A new [Matrix](#) representing the element-wise difference.

## Exceptions

<code>std::invalid_argument</code>	if the dimensions of the two matrices do not match.
------------------------------------	---

## Note

The shape remains unchanged.

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

```
Matrix Matrix::operator- (
    double scalar ) const [inline]
```

Subtracts a scalar from each element of the matrix. (MATRIX - K)

## Parameters

<i>scalar</i>	The scalar value to subtract.
---------------	-------------------------------

## Returns

A new [Matrix](#) with each element reduced by the scalar.

## Note

The shape remains unchanged.

**3.1.3.17 operator/()**

```
Matrix Matrix::operator/ (
    double scalar ) const [inline]
```

Divides each element of the matrix by a scalar.

## Parameters

<i>scalar</i>	The scalar value.
---------------	-------------------

## Returns

A new [Matrix](#) with each element divided by the scalar.



### Exceptions

<code>std::runtime_error</code>	if scalar is zero.
---------------------------------	--------------------

### Note

The shape remains unchanged.

### 3.1.3.18 operator==()

```
bool Matrix::operator== (
    const Matrix & other ) const [inline]
```

Compares two matrices for equality.

#### Parameters

<i>other</i>	The <a href="#">Matrix</a> to compare with.
--------------	---

### Returns

True if the matrices are equal (within a tolerance), false otherwise.

### 3.1.3.19 operator^()

```
Matrix Matrix::operator^ (
    double scalar ) const [inline]
```

### 3.1.3.20 shape()

```
std::pair<size_t, size_t> Matrix::shape ( ) const [inline]
```

Returns the dimensions of the matrix.

### Returns

A `std::pair` where first is the number of rows and second is the number of columns.

**3.1.3.21 sum()**

```
double Matrix::sum ( ) const [inline]
```

Computes the sum of the elements of a vector.

Calculates the sum of elements for matrices that are considered as vectors. It supports both column vectors ( $K \times 1$ ) and row vectors ( $1 \times K$ ).

**Returns**

The sum of all elements in the vector.

**Exceptions**

<code>std::invalid_argument</code>	If the matrix is not a one-dimensional vector.
------------------------------------	--

**3.1.3.22 transpose()**

```
Matrix Matrix::transpose ( ) const [inline]
```

Transposes the matrix.

**Returns**

A new matrix of dim (mxn) for a calling matrix of dim (nxm) Example:

```
Matrix A({{1, 2}, {3, 4}});
A = A.transpose();
// A becomes:
// [
//   [1, 3],
//   [2, 4]
// ]
```

**3.1.3.23 vStack()**

```
Matrix Matrix::vStack (
    const Matrix & other ) const [inline]
```

Vertically concatenates two matrices.

This function creates and returns a new [Matrix](#) by appending the rows of the given matrix below the rows of the calling matrix. Both matrices must have the same number of columns.

**Parameters**

<code>other</code>	The <a href="#">Matrix</a> whose rows will be appended to the calling matrix.
--------------------	---

**Returns**

A new [Matrix](#) representing the vertical concatenation of the two matrices.

**Exceptions**

<code>std::invalid_argument</code>	if the two matrices do not have the same number of columns.
------------------------------------	---

**Note**

The function appends each row of the second matrix to the container of the first, and updates the total row count accordingly.

**3.1.4 Friends And Related Function Documentation****3.1.4.1 operator\***

```
Matrix operator* (
    double scalar,
    const Matrix & other ) [friend]
```

Multiplies a scalar by a matrix. ( $K * MATRIX$ )

**Parameters**

<i>scalar</i>	The scalar value.
<i>other</i>	The <a href="#">Matrix</a> to multiply.

**Returns**

A new [Matrix](#) with each element multiplied by the scalar.

**Note**

The shape remains unchanged.

**3.1.4.2 operator+**

```
Matrix operator+ (
    double scalar,
    const Matrix & other ) [friend]
```

Adds a scalar to each element of a matrix. ( $K + MATRIX$ )

## Parameters

<i>scalar</i>	The scalar value.
<i>other</i>	The <a href="#">Matrix</a> to add the scalar to.

## Returns

A new [Matrix](#) with the result.

## Note

The shape remains unchanged.

**3.1.4.3 operator-**

```
Matrix operator- (
    double scalar,
    const Matrix & other ) [friend]
```

Subtracts each element of the matrix from a scalar. (K - MATRIX)

## Parameters

<i>scalar</i>	The scalar value.
<i>other</i>	The <a href="#">Matrix</a> whose elements are subtracted from the scalar.

## Returns

A new [Matrix](#) with the result.

## Note

The shape remains unchanged.

**3.1.4.4 operator<<**

```
std::ostream& operator<< (
    std::ostream & os,
    const Matrix & m ) [friend]
```

Outputs the matrix to an output stream.

## Parameters

<i>os</i>	The output stream.
<i>m</i>	The <a href="#">Matrix</a> to output.

**Returns**

A reference to the output stream.

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

- [src/matOps.hpp](#)

## Chapter 4

# File Documentation

### 4.1 src/matOps.hpp File Reference

```
#include <iostream>
#include <vector>
#include <cmath>
```

Include dependency graph for matOps.hpp: This graph shows which files directly or indirectly include this file:

#### Classes

- class [Matrix](#)  
*A simple linear algebra library for matrix operations.*

#### Functions

- `std::ostream & operator<< (std::ostream &os, const std::pair< size_t, size_t > &shape)`

#### 4.1.1 Function Documentation

##### 4.1.1.1 `operator<<()`

```
std::ostream& operator<< (
    std::ostream & os,
    const std::pair< size_t, size_t > & shape ) [inline]
```

## 4.2 test/testMatrix.cpp File Reference

Unit tests for the [Matrix](#) class using doctest.

```
#include "doctest.h"
#include "../src/matOps.hpp"
#include <vector>
#include <stdexcept>
#include <cmath>
Include dependency graph for testMatrix.cpp:
```

### Macros

- `#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN`

### Functions

- [TEST\\_CASE](#) ("Matrix Construction and Shape")  
*Tests for [Matrix](#) construction and shape reporting.*
- [TEST\\_CASE](#) ("Matrix Addition and Subtraction")  
*Tests for [Matrix](#) addition and subtraction operations.*
- [TEST\\_CASE](#) ("Matrix Multiplication and Division")  
*Tests for [Matrix](#) multiplication and division.*
- [TEST\\_CASE](#) ("Element Access Operator()")  
*Tests for element access using operator().*
- [TEST\\_CASE](#) ("Matrix Transpose")  
*Tests for [Matrix](#) transposition.*
- [TEST\\_CASE](#) ("Matrix Determinant")  
*Tests for [Matrix](#) determinant computation.*
- [TEST\\_CASE](#) ("Matrix Inverse")  
*Tests for [Matrix](#) inversion.*
- [TEST\\_CASE](#) ("Matrix Row and Column Insertion")  
*Tests for [Matrix](#) row and column insertion methods.*
- [TEST\\_CASE](#) ("Matrix Horizontal and Vertical Stacking")  
*Tests for [Matrix](#) horizontal and vertical stacking.*
- [TEST\\_CASE](#) ("Matrix Submatrix Extraction")  
*Tests for [Matrix](#) submatrix extraction functionality.*
- [TEST\\_CASE](#) ("Equality Tolerance Test")  
*Tests for equality comparisons with tolerance.*
- [TEST\\_CASE](#) ("Copy Constructor and Assignment Operator")  
*Tests for copy construction and assignment operator.*
- [TEST\\_CASE](#) ("3x3 [Matrix](#) Inversion and Identity Check")  
*Tests for 3x3 matrix inversion and verifying the identity.*
- [TEST\\_CASE](#) ("Single Row and Single Column Submatrix Extraction")  
*Tests for submatrix extraction of a single row and a single column.*
- [TEST\\_CASE](#) ("Chained Mixed Operations")  
*Tests for chained mixed arithmetic operations.*
- [TEST\\_CASE](#) ("Row vector: sum and mean")
- [TEST\\_CASE](#) ("Column vector: sum and mean")
- [TEST\\_CASE](#) ("Non-vector matrix: sum throws invalid\_argument")
- [TEST\\_CASE](#) ("Matrix exponentiation with scalar 1 returns the same matrix")
- [TEST\\_CASE](#) ("Matrix exponentiation with scalar 2 returns element-wise square")
- [TEST\\_CASE](#) ("Matrix exponentiation with scalar 0 returns element-wise 1 (nonzero elements)")
- [TEST\\_CASE](#) ("Matrix exponentiation when an element is 0 and exponent is  $\leq 0$ ")

### 4.2.1 Detailed Description

Unit tests for the [Matrix](#) class using doctest.

### 4.2.2 Macro Definition Documentation

#### 4.2.2.1 DOCTEST\_CONFIG\_IMPLEMENT\_WITH\_MAIN

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

### 4.2.3 Function Documentation

#### 4.2.3.1 TEST\_CASE() [1/22]

```
TEST_CASE (
    "3x3 Matrix Inversion and Identity Check" )
```

Tests for 3x3 matrix inversion and verifying the identity.

#### 4.2.3.2 TEST\_CASE() [2/22]

```
TEST_CASE (
    "Chained Mixed Operations" )
```

Tests for chained mixed arithmetic operations.

#### 4.2.3.3 TEST\_CASE() [3/22]

```
TEST_CASE (
    "Column vector:  sum and mean" )
```



#### 4.2.3.4 TEST\_CASE() [4/22]

```
TEST_CASE (
    "Copy Constructor and Assignment Operator" )
```

Tests for copy construction and assignment operator.

#### 4.2.3.5 TEST\_CASE() [5/22]

```
TEST_CASE (
    "Element Access Operator()" )
```

Tests for element access using operator().

Verify correct element access.

Modification via operator(). Tests that modifying an element via the accessor works correctly.

Out-of-range access. Checks that accessing an element outside the matrix bounds throws an exception.

#### 4.2.3.6 TEST\_CASE() [6/22]

```
TEST_CASE (
    "Equality Tolerance Test" )
```

Tests for equality comparisons with tolerance.

#### 4.2.3.7 TEST\_CASE() [7/22]

```
TEST_CASE (
    "Matrix Addition and Subtraction" )
```

Tests for [Matrix](#) addition and subtraction operations.

[Matrix](#) addition (matrix + matrix). Verifies that element-wise addition produces the expected result.

[Matrix](#) addition with scalar (matrix + k and k + matrix). Verifies both left and right scalar addition.

[Matrix](#) subtraction (matrix - matrix). Checks that subtracting one matrix from another yields the correct result.

[Matrix](#) subtraction with scalar (matrix - k and k - matrix). Verifies subtraction when a scalar is involved.

#### 4.2.3.8 TEST\_CASE() [8/22]

```
TEST_CASE (
    "Matrix Construction and Shape" )
```

Tests for [Matrix](#) construction and shape reporting.

Valid construction with a 2x2 matrix. Checks that the matrix correctly reports its dimensions.

Invalid construction with an empty container. Expects an `invalid_argument` exception.

Invalid construction with inconsistent row sizes. Expects an `invalid_argument` exception.

#### 4.2.3.9 TEST\_CASE() [9/22]

```
TEST_CASE (
    "Matrix Determinant" )
```

Tests for [Matrix](#) determinant computation.

Determinant for a 1x1 matrix.

Determinant for a 2x2 matrix.

Determinant for a 3x3 matrix.

Non-square matrix determinant. Expects an exception because the determinant is only defined for square matrices.

#### 4.2.3.10 TEST\_CASE() [10/22]

```
TEST_CASE (
    "Matrix exponentiation when an element is 0 and exponent is <= 0" )
```

#### 4.2.3.11 TEST\_CASE() [11/22]

```
TEST_CASE (
    "Matrix exponentiation with scalar 0 returns element-wise 1 (nonzero elements)"
)
```

#### 4.2.3.12 TEST\_CASE() [12/22]

```
TEST_CASE (
    "Matrix exponentiation with scalar 1 returns the same matrix" )
```

#### 4.2.3.13 TEST\_CASE() [13/22]

```
TEST_CASE (
    "Matrix exponentiation with scalar 2 returns element-wise square" )
```

#### 4.2.3.14 TEST\_CASE() [14/22]

```
TEST_CASE (
    "Matrix Horizontal and Vertical Stacking" )
```

Tests for [Matrix](#) horizontal and vertical stacking.

Horizontal stacking. Verifies that concatenating matrices side-by-side produces the expected result.

Vertical stacking. Verifies that concatenating matrices top-to-bottom produces the expected result.

Stacking with mismatched dimensions. Ensures that attempting to stack matrices with incompatible dimensions throws an exception.

#### 4.2.3.15 TEST\_CASE() [15/22]

```
TEST_CASE (
    "Matrix Inverse" )
```

Tests for [Matrix](#) inversion.

Inversion of a 2x2 invertible matrix. Checks that the computed inverse matches the expected result.

Inversion of a singular matrix. Verifies that attempting to invert a non-invertible matrix throws an exception.

#### 4.2.3.16 TEST\_CASE() [16/22]

```
TEST_CASE (
    "Matrix Multiplication and Division" )
```

Tests for [Matrix](#) multiplication and division.

[Matrix](#) multiplication (matrix \* matrix). Verifies that the product of two matrices is as expected.

Scalar multiplication (matrix \* k and k \* matrix). Checks that scalar multiplication scales each element appropriately.

[Matrix](#) division by scalar. Tests division by a scalar and expects an exception when dividing by zero.

**4.2.3.17 TEST\_CASE() [17/22]**

```
TEST_CASE (
    "Matrix Row and Column Insertion" )
```

Tests for [Matrix](#) row and column insertion methods.

Insert row with a provided vector.

Insert row filled with a scalar value.

Insert column with a provided vector.

Insert column filled with a scalar value.

**4.2.3.18 TEST\_CASE() [18/22]**

```
TEST_CASE (
    "Matrix Submatrix Extraction" )
```

Tests for [Matrix](#) submatrix extraction functionality.

Valid submatrix extraction. Extracts a submatrix (rows 0-1, columns 1-2) and compares with the expected result.

Invalid extraction due to row indices out of bounds.

Invalid extraction when start index is greater than end index (rows).

Invalid extraction when start index is greater than end index (columns).

**4.2.3.19 TEST\_CASE() [19/22]**

```
TEST_CASE (
    "Matrix Transpose" )
```

Tests for [Matrix](#) transposition.

**4.2.3.20 TEST\_CASE() [20/22]**

```
TEST_CASE (
    "Non-vector matrix: sum throws invalid_argument" )
```

**4.2.3.21 TEST\_CASE() [21/22]**

```
TEST_CASE (
    "Row vector: sum and mean" )
```

**4.2.3.22 TEST\_CASE() [22/22]**

```
TEST_CASE (
    "Single Row and Single Column Submatrix Extraction" )
```

Tests for submatrix extraction of a single row and a single column.

Extract the second row as a 1x3 matrix.

Extract the third column as a 3x1 matrix.