matOps

Generated by Doxygen 1.9.1

1 Class Index 1

1.1 Class List . . . 1

2 File Index 2

2.1 File List . . . 2

3 Class Documentation 3

3.1 Matrix Class Reference . . . 3

3.1.1 Detailed Description . . . 5

3.1.2 Constructor & Destructor Documentation . . . 5

3.1.2.1 Matrix() . . . 5

3.1.3 Member Function Documentation . . . 5

3.1.3.1 constValMatrix() . . . 5

3.1.3.2 determinant() . . . 6

3.1.3.3 extractMatrix() . . . 6

3.1.3.4 hStack() . . . 7

3.1.3.5 identity() . . . 8

3.1.3.6 insertCol() [1/2] . . . 8

3.1.3.7 insertCol() [2/2] . . . 8

3.1.3.8 insertRow() [1/2] . . . 9

3.1.3.9 insertRow() [2/2] . . . 9

3.1.3.10 inverse() . . . 11

3.1.3.11 mean() . . . 11

3.1.3.12 operator"!=() . . . 12

3.1.3.13 operator()() . . . 12

3.1.3.14 operator*() [1/2] . . . 13

3.1.3.15 operator*() [2/2] . . . 13

3.1.3.16 operator+() [1/2] . . . 14

3.1.3.17 operator+() [2/2] . . . 14

3.1.3.18 operator-() [1/2] . . . 15

3.1.3.19 operator-() [2/2] . . . 15

3.1.3.20 operator/() . . . 16

3.1.3.21 operator==() . . . 17

3.1.3.22 operator^() . . . 17

3.1.3.23 shape() . . . 17

3.1.3.24 shuffleRows() [1/2] . . . 18

3.1.3.25 shuffleRows() [2/2] . . . 18

3.1.3.26 sum() [1/2] . . . 18

3.1.3.27 sum() [2/2] . . . 18

3.1.3.28 toVector() . . . 19

3.1.3.29 transpose() . . . 19

3.1.3.30 vStack() . . . 20

3.1.4 Friends And Related Function Documentation . . . 20

3.1.4.1 operator∗ . . . 20
3.1.4.2 operator+ . . . 21
3.1.4.3 operator- . . . 21
3.1.4.4 operator<< . . . 22

4 File Documentation **23**
4.1 src/matOps.hpp File Reference . . . 23
4.1.1 Macro Definition Documentation . . . 23
4.1.1.1 EPS . . . 23
4.1.1.2 OPENMP_THRESHOLD . . . 24
4.1.2 Function Documentation . . . 24
4.1.2.1 operator<<() . . . 24
4.2 test/testMatrix.cpp File Reference . . . 24
4.2.1 Detailed Description . . . 25
4.2.2 Macro Definition Documentation . . . 25
4.2.2.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN . . . 25
4.2.3 Function Documentation . . . 25
4.2.3.1 TEST_CASE() [1/31] . . . 26
4.2.3.2 TEST_CASE() [2/31] . . . 26
4.2.3.3 TEST_CASE() [3/31] . . . 26
4.2.3.4 TEST_CASE() [4/31] . . . 26
4.2.3.5 TEST_CASE() [5/31] . . . 26
4.2.3.6 TEST_CASE() [6/31] . . . 26
4.2.3.7 TEST_CASE() [7/31] . . . 27
4.2.3.8 TEST_CASE() [8/31] . . . 27
4.2.3.9 TEST_CASE() [9/31] . . . 27
4.2.3.10 TEST_CASE() [10/31] . . . 27
4.2.3.11 TEST_CASE() [11/31] . . . 27
4.2.3.12 TEST_CASE() [12/31] . . . 28
4.2.3.13 TEST_CASE() [13/31] . . . 28
4.2.3.14 TEST_CASE() [14/31] . . . 28
4.2.3.15 TEST_CASE() [15/31] . . . 28
4.2.3.16 TEST_CASE() [16/31] . . . 28
4.2.3.17 TEST_CASE() [17/31] . . . 29
4.2.3.18 TEST_CASE() [18/31] . . . 29
4.2.3.19 TEST_CASE() [19/31] . . . 29
4.2.3.20 TEST_CASE() [20/31] . . . 29
4.2.3.21 TEST_CASE() [21/31] . . . 30
4.2.3.22 TEST_CASE() [22/31] . . . 30
4.2.3.23 TEST_CASE() [23/31] . . . 30
4.2.3.24 TEST_CASE() [24/31] . . . 30
4.2.3.25 TEST_CASE() [25/31] . . . 30

4.2.3.26 TEST_CASE() [26/31] 31
4.2.3.27 TEST_CASE() [27/31] 31
4.2.3.28 TEST_CASE() [28/31] 31
4.2.3.29 TEST_CASE() [29/31] 31
4.2.3.30 TEST_CASE() [30/31] 31
4.2.3.31 TEST_CASE() [31/31] 31

Chapter 1

Class Index

1.1 Class List

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

Matrix
A simple linear algebra library for matrix operations 3

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

src/matOps.hpp 23
test/testMatrix.cpp
Unit tests for the Matrix class using doctest 24

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.
- std::vector< std::vector< double > > toVector () const

  Returns a copy of the matrix data.
- 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.
- bool operator!= (const Matrix &other) const

  Compares two matrices for inequality.
- 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.
- void shuffleRows ()

  Shuffles the rows of the matrix using a random seed.
- void shuffleRows (size_t random_state)

  Shuffles the rows of the matrix using a specified seed.
- 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 sum (double power) const

  Computes the sum of all elements raised to a specified power.
- double mean () const

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

Static Public Member Functions

- static Matrix identity (size_t dim)

  Constructs an Identity Matrix of specified dimensions.
- static Matrix constValMatrix (size_t rows, size_t cols, double val)

  Creates a matrix with constant values.

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()

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

Constructs a Matrix from a given 2D vector container.

Parameters

container	A 2D vector of doubles representing the matrix.

Exceptions

std::invalid_argument	if the container is empty or row sizes are inconsistent.

Note

The shape is determined by the size of the container.

3.1.3 Member Function Documentation

3.1.3.1 constValMatrix()

```
static Matrix Matrix::constValMatrix (
            size_t rows,
            size_t cols,
            double val )  [inline], [static]
```

Creates a matrix with constant values.

Function constructs and returns a Matrix object with the specified dimensions, initializing every element to the given constant value.

Example usage:

```
// Create a 3x3 matrix filled with 5.0
Matrix A = Matrix::constValMatrix(3, 3, 5.0);
// Expected Output:
//  5.0  5.0  5.0
//  5.0  5.0  5.0
//  5.0  5.0  5.0
```

Parameters

rows	The number of rows in the matrix.
cols	The number of columns in the matrix.
val	The constant value to initialize each element of the matrix.

Returns

A Matrix object of dimensions (rows x cols) where each element is set to `val`.

Exceptions

std::invalid_argument	if either `rows` or `cols` is zero.

3.1.3.2 determinant()

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

Computes the determinant of the matrix.

Returns

The determinant as a double.

Exceptions

std::invalid_argument	if the matrix is not square.

Note

The shape of the matrix remains unchanged.

3.1.3.3 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 [start, end).

Parameters

rowSlice	A std::pair<size_t, size_t> representing the start and end row indices.
colSlice	A std::pair<size_t, size_t> representing the start and end column indices.

Returns

A new Matrix object containing the extracted submatrix.

Exceptions

std::out_of_range	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).

3.1.3.4 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

other	The Matrix whose columns will be appended to the calling matrix.

Returns

A new Matrix representing the horizontal concatenation of the two matrices.

Exceptions

std::invalid_argument	if the two matrices do not have the same number of rows.

Note

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

3.1.3.5 identity()

```
static Matrix Matrix::identity (
            size_t dim )  [inline], [static]
```

Constructs an Identity Matrix of specified dimensions.

```
Matrix A = Matrix::identity(3);
A = [
     [1, 0, 0]
     [0, 1, 0]
     [0, 0, 1]
]
```

Parameters

dim	Dimensions of matrix (dim x dim).

3.1.3.6 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

colVal	The constant value to fill the new column.
idx	The index at which to insert the column.

Returns

A new Matrix with the column inserted.

Exceptions

std::invalid_argument	if idx is out of range.

3.1.3.7 insertCol() [2/2]

```
Matrix Matrix::insertCol (
```

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

Inserts a new column into the matrix.

Parameters

col	A vector representing the new column.
idx	The index at which to insert the column.

Returns

A new Matrix with the column inserted.

Exceptions

std::invalid_argument	if the column size is inconsistent or if idx is out of range.

3.1.3.8 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

rowVal	The constant value to fill the new row.
idx	The index at which to insert the row.

Returns

A new Matrix with the row inserted.

Exceptions

std::invalid_argument	if idx is out of range.

3.1.3.9 insertRow() [2/2]

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

Inserts a new row into the matrix.

Parameters

row	A vector representing the new row.
idx	The index at which to insert the row.

Returns

A new Matrix with the row inserted.

Exceptions

std::invalid_argument	if the row size is inconsistent or if idx is out of range.

3.1.3.10 inverse()

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

Computes the inverse of the matrix.

Returns

A new Matrix representing the inverse.

Exceptions

std::runtime_error	if the matrix is singular (non-invertible).

Note

The shape remains unchanged.

3.1.3.11 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

std::invalid_argument	If the matrix is not a one-dimensional vector.

3.1.3.12 operator"!=()

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

Compares two matrices for inequality.

This operator checks if two matrices are not equal by comparing their dimensions and individual elements. Two matrices are considered not equal if:

- Their dimensions differ, or
- At least one pair of corresponding elements differs by more than EPS.

Parameters

other	The matrix to compare with.

Returns

true if the matrices differ by at least one element more than EPS; false otherwise.

3.1.3.13 operator()()

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

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

Parameters

row	The row index.
col	The column index.

Returns

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

Exceptions

std::out_of_range	if the indices are out of bounds.

3.1.3.14 operator∗() [1/2]

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

Multiplies two matrices.

Parameters

other	The Matrix to multiply with.

Returns

A new Matrix resulting from matrix multiplication.

Exceptions

std::invalid_argument	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).

3.1.3.15 operator∗() [2/2]

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

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

Parameters

scalar	The scalar value.

Returns

A new Matrix with each element multiplied by the scalar.

Note

The shape remains unchanged.

3.1.3.16 operator+() [1/2]

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

Adds two matrices element-wise.

Parameters

other	The Matrix to add.

Returns

A new Matrix representing the element-wise sum.

Exceptions

std::invalid_argument	if the dimensions of the two matrices do not match.

Note

The shape remains unchanged.

3.1.3.17 operator+() [2/2]

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

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

Parameters

scalar	A double value to add.

Returns

A new Matrix with the scalar added to each element.

Note

The shape remains unchanged.

3.1.3.18 operator-() [1/2]

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

Subtracts one matrix from another element-wise.

Parameters

other	The Matrix to subtract.

Returns

A new Matrix representing the element-wise difference.

Exceptions

std::invalid_argument	if the dimensions of the two matrices do not match.

Note

The shape remains unchanged.

3.1.3.19 operator-() [2/2]

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

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

Parameters

scalar	The scalar value to subtract.

Returns

A new Matrix with each element reduced by the scalar.

Note

The shape remains unchanged.

3.1.3.20 operator/()

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

Divides each element of the matrix by a scalar.

Parameters

scalar	The scalar value.

Returns

A new Matrix with each element divided by the scalar.

Exceptions

std::runtime_error	if scalar is zero.

Note

The shape remains unchanged.

3.1.3.21 operator==()

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

Compares two matrices for equality.

Parameters

other	The Matrix to compare with.

Returns

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

3.1.3.22 operator^()

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

3.1.3.23 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.

3.1.3.24 shuffleRows() [1/2]

```
void Matrix::shuffleRows ( )  [inline]
```

Shuffles the rows of the matrix using a random seed.

This function uses a random seed generated by std::random_device to initialize the random number generator, ensuring that the shuffle is non-deterministic.

3.1.3.25 shuffleRows() [2/2]

```
void Matrix::shuffleRows (
            size_t random_state )  [inline]
```

Shuffles the rows of the matrix using a specified seed.

This function initializes the random number generator with the provided seed (random_state), allowing for reproducible shuffling.

Parameters

random_state	The seed value used to initialize the random number generator.

3.1.3.26 sum() [1/2]

```
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 x 1) and row vectors (1 x K).

Returns

The sum of all elements in the vector.

Exceptions

std::invalid_argument	If the matrix is not a one-dimensional vector.

3.1.3.27 sum() [2/2]

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

Computes the sum of all elements raised to a specified power.

This function calculates the sum of each element in the matrix after raising it to the given `power`. The function only works for matrices that are either a row matrix (1 x K) or a column matrix (K x 1). If the matrix has any other dimensions, an std::invalid_argument exception is thrown.

Additionally, if an element is zero and the `power` is less than or equal to zero, the function will throw an std↩ ::runtime_error to indicate a division by zero scenario, since 0 raised to a non-positive power is undefined.

Parameters

power	The exponent to which each element in the matrix is raised.

Returns

The sum of all elements raised to the specified power.

Exceptions

std::invalid_argument	if the matrix dimensions are not (1, K) or (K, 1).
std::runtime_error	if any element is zero and `power` is less than or equal to zero.

3.1.3.28 toVector()

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

Returns a copy of the matrix data.

This function returns a new 2D vector containing the matrix elements. Modifications to the returned vector do not affect the original matrix.

Returns

A copy of the 2D vector representing the matrix.

3.1.3.29 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]
// ]
```

3.1.3.30 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

other	The Matrix whose rows will be appended to the calling matrix.

Returns

A new Matrix representing the vertical concatenation of the two matrices.

Exceptions

std::invalid_argument	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

scalar	The scalar value.
other	The Matrix 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

scalar	The scalar value.
other	The Matrix 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

scalar	The scalar value.
other	The Matrix 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

os	The output stream.
m	The Matrix 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 <random>
#include <algorithm>
```

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.

Macros

- #define EPS 1e-12
- #define OPENMP_THRESHOLD 10000

Functions

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

4.1.1 Macro Definition Documentation

4.1.1.1 EPS

```
#define EPS 1e-12
```

4.1.1.2 OPENMP_THRESHOLD

```
#define OPENMP_THRESHOLD 10000
```

4.1.2 Function Documentation

4.1.2.1 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 using exclusive indices.
- TEST_CASE ("Equality Tolerance Test")

  Tests for equality comparisons with tolerance.
- TEST_CASE ("Matrix Inequality Operator")
- 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 <= 0")
- TEST_CASE ("Matrix shuffleRows methods")
- TEST_CASE ("Matrix::constValMatrix creates a constant matrix")
- TEST_CASE ("Matrix::constValMatrix throws for zero dimensions")
- TEST_CASE ("Row matrix: valid sum calculation with positive power")
- TEST_CASE ("Column matrix: valid sum calculation with positive power")
- TEST_CASE ("Invalid matrix dimensions should throw invalid_argument")
- TEST_CASE ("Row matrix: zero element with non-positive power throws runtime_error")
- TEST_CASE ("Column matrix: zero element with non-positive power throws runtime_error")

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() [1/31]

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

Tests for 3x3 matrix inversion and verifying the identity.

4.2.3.2 TEST_CASE() [2/31]

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

Tests for chained mixed arithmetic operations.

4.2.3.3 TEST_CASE() [3/31]

```
TEST_CASE (
            "Column matrix:  valid sum calculation with positive power"  )
```

4.2.3.4 TEST_CASE() [4/31]

```
TEST_CASE (
            "Column matrix:  zero element with non-positive power throws runtime_error"  )
```

4.2.3.5 TEST_CASE() [5/31]

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

4.2.3.6 TEST_CASE() [6/31]

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

Tests for copy construction and assignment operator.

4.2.3.7 TEST_CASE() [7/31]

```
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.

4.2.3.8 TEST_CASE() [8/31]

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

Tests for equality comparisons with tolerance.

4.2.3.9 TEST_CASE() [9/31]

```
TEST_CASE (
            "Invalid matrix dimensions should throw invalid_argument"  )
```

4.2.3.10 TEST_CASE() [10/31]

```
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.

4.2.3.11 TEST_CASE() [11/31]

```
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.

4.2.3.12 TEST_CASE() [12/31]

```
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.

4.2.3.13 TEST_CASE() [13/31]

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

4.2.3.14 TEST_CASE() [14/31]

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

4.2.3.15 TEST_CASE() [15/31]

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

4.2.3.16 TEST_CASE() [16/31]

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

4.2.3.17 TEST_CASE() [17/31]

```
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.

4.2.3.18 TEST_CASE() [18/31]

```
TEST_CASE (
            "Matrix Inequality Operator"  )
```

4.2.3.19 TEST_CASE() [19/31]

```
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.

4.2.3.20 TEST_CASE() [20/31]

```
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.

4.2.3.21 TEST_CASE() [21/31]

```
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.

4.2.3.22 TEST_CASE() [22/31]

```
TEST_CASE (
             "Matrix shuffleRows methods"  )
```

4.2.3.23 TEST_CASE() [23/31]

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

Tests for Matrix submatrix extraction functionality using exclusive indices.

4.2.3.24 TEST_CASE() [24/31]

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

Tests for Matrix transposition.

4.2.3.25 TEST_CASE() [25/31]

```
TEST_CASE (
             "Matrix::constValMatrix creates a constant matrix"  )
```

4.2.3.26 TEST_CASE() [26/31]

```
TEST_CASE (
            "Matrix::constValMatrix throws for zero dimensions"  )
```

4.2.3.27 TEST_CASE() [27/31]

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

4.2.3.28 TEST_CASE() [28/31]

```
TEST_CASE (
            "Row matrix:  valid sum calculation with positive power"  )
```

4.2.3.29 TEST_CASE() [29/31]

```
TEST_CASE (
            "Row matrix:  zero element with non-positive power throws runtime_error"  )
```

4.2.3.30 TEST_CASE() [30/31]

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

4.2.3.31 TEST_CASE() [31/31]

```
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. Using row indices [1,2) and column indices [0,3).

Extract the third column as a 3x1 matrix. Using row indices [0,3) and column indices [2,3).