My Project

Generated by Doxygen 1.8.17

Chapter 1

Lin_Alg

Library for Linear Algebra. Work in progress!

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

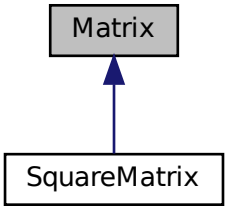
Class Documentation

4.1 Matrix Class Reference

Class implementing a 2D matrix.

`#include <Matrix.h>`

Inheritance diagram for Matrix:



Public Member Functions

- Matrix ()

  Construct a new empty Matrix object.
- Matrix (int m, int n)

  Construct a new Matrix object with dimensions m∗n.
- **Matrix** (std::initializer_list< std::initializer_list< double > > init, bool byColumns=false)
- std::pair< int, int > order () const

  Gives the dimensions of the matrix as the std::pair {num_rows, num_columns}.
- const double & at (int i, int j) const

  Returns a const reference to the (i,j)th element of the matrix.
- double & at (int i, int j)

  Returns a reference to the (i,j)th element of the matrix.

- const Vector & at (int i) const

  Returns a reference to the ith column of the matrix.
- Vector & **at** (int i)
- Matrix operator∗ (const Matrix &m)

  Returns the product of two matrices.
- Matrix transpose (bool modify=false)

  Returns a new matrix which is the transpose of the original matrix.
- Matrix cef (bool modify=false)

  Returns one possible column echelon form of the given matrix.
- Matrix cef (bool modify=false) const

  Returns one possible column echelon form of the given matrix.
- Matrix rcef (bool modify=false)

  Returns the reduced column echelon form of the given matrix.
- Matrix rref (bool modify=false)

  Returns the reduced row echelon form of the given matrix.
- void **augment** (const Matrix &other)
- Matrix GramSchmidt (bool modify=false)

  Runs the Gram-Schmidt algorithm on the columns of a copy of the given matrix.
- void Mj (int j, double c, bool columnOperation=false)

  Multiplies a given row/column at the jth index of the matrix with a nonzero scalar c.
- void Pjk (int j, int k, bool columnOperation=false)

  Swaps the row/column at index j with the row/column at index k.
- void Ejk (int j, int k, double lambda, bool columnOperation=false)

  C_j = C_j+lambda∗C_k (if columnOperation is true), R_j = R_j+lambda∗R_k otherwise.
- void elementaryColumnOperation (const std::string &type, int j, int k, double lambda=0)
- void elementaryRowOperation (const std::string &type, int j, int k, double lambda=0)
- int **rank** () const
- std::pair< Matrix, Matrix > **QR** ()
- std::vector< Vector >::const_iterator **begin** () const
- std::vector< Vector >::const_iterator **end** () const

Protected Member Functions

- Matrix & **cef** (int start_row, int start_col)
- Matrix & **rcef** (int start_row, int start_col)
- int **size** () const

Protected Attributes

- std::vector< Vector > **mat**

4.1.1 Detailed Description

Class implementing a 2D matrix.

Note

All indices start from 0.

An empty matrix has order (0,0).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Matrix() [1/2]

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

Construct a new empty Matrix object.

4.1.2.2 Matrix() [2/2]

```
Matrix::Matrix (
            int m,
            int n )  [inline]
```

Construct a new Matrix object with dimensions m∗n.

Parameters

m	Number of rows in the matrix
n	Number of columns in the matrix

4.1.3 Member Function Documentation

4.1.3.1 at() [1/3]

```
const Vector& Matrix::at (
            int i ) const  [inline]
```

Returns a reference to the ith column of the matrix.

Parameters

i	index of the column

Returns

const std::vector<Vector>&

4.1.3.2 at() [2/3]

```
double& Matrix::at (
            int i,
            int j )  [inline]
```

Returns a reference to the (i,j)th element of the matrix.

Parameters

i	row number of the required element
j	column number of the required element

Returns

const double&

4.1.3.3 at() [3/3]

```
const double& Matrix::at (
            int i,
            int j ) const  [inline]
```

Returns a const reference to the (i,j)th element of the matrix.

Parameters

i	row number of the required element
j	column number of the required element

Returns

const double&

4.1.3.4 cef() [1/2]

```
Matrix Matrix::cef (
            bool modify = false )  [inline]
```

Returns one possible column echelon form of the given matrix.

Parameters

modify	if modify is true, then the given matrix is changed to its column echelon form

Returns

Matrix one possible column echelon form of the given matrix

4.1.3.5 cef() [2/2]

```
Matrix Matrix::cef (
            bool modify = false ) const  [inline]
```

Returns one possible column echelon form of the given matrix.

Parameters

modify	throws an exception if modify is true, since a const matrix cannot be modified

Returns

Matrix one possible column echelon form of the given matrix

4.1.3.6 Ejk()

```
void Matrix::Ejk (
            int j,
            int k,
            double lambda,
            bool columnOperation = false )  [inline]
```

C_j = C_j+lambda∗C_k (if columnOperation is true), R_j = R_j+lambda∗R_k otherwise.

Parameters

j	
k	
lambda	
columnOperation	

4.1.3.7 elementaryColumnOperation()

```
void Matrix::elementaryColumnOperation (
            const std::string & type,
            int j,
            int k,
            double lambda = 0 )  [inline]
```

Parameters

type	
j	
k	
lambda	

4.1.3.8 elementaryRowOperation()

```
void Matrix::elementaryRowOperation (
            const std::string & type,
            int j,
            int k,
            double lambda = 0 )  [inline]
```

Parameters

type	
j	
k	
lambda	

4.1.3.9 GramSchmidt()

```
Matrix Matrix::GramSchmidt (
            bool modify = false )
```

Runs the Gram-Schmidt algorithm on the columns of a copy of the given matrix.

Parameters

modify	if true, then the given matrix is modified.

Returns

Matrix

4.1.3.10 Mj()

```
void Matrix::Mj (
            int j,
```

```
double c,
bool columnOperation = false )  [inline]
```

Multiplies a given row/column at the jth index of the matrix with a nonzero scalar c.

Parameters

j	index of the row/column
c	Scalar which is to be multiplied
columnOperation	Multiplies the jth column by c if true, else multiplies the jth row by c.

4.1.3.11 operator∗()

```
Matrix Matrix::operator* (
            const Matrix & m )
```

Returns the product of two matrices.

Returns

Matrix Product of the two matrices

4.1.3.12 order()

```
std::pair<int,int> Matrix::order ( ) const  [inline]
```

Gives the dimensions of the matrix as the std::pair {num_rows, num_columns}.

Returns

std::pair<int,int>

4.1.3.13 Pjk()

```
void Matrix::Pjk (
            int j,
            int k,
            bool columnOperation = false )  [inline]
```

Swaps the row/column at index j with the row/column at index k.

Parameters

j	
k	
columnOperation	If true, then the jth and kth columns are swapped, otherwise rows are swapped.

4.1.3.14 rcef()

```
Matrix Matrix::rcef (
            bool modify = false )  [inline]
```

Returns the reduced column echelon form of the given matrix.

Parameters

modify	if modify is true, then the given matrix is changed to its reduced column echelon form

Returns

Matrix one possible reduced column echelon form of the given matrix

4.1.3.15 rref()

```
Matrix Matrix::rref (
            bool modify = false )  [inline]
```

Returns the reduced row echelon form of the given matrix.

Parameters

modify	if modify is true, then the given matrix is changed to its reduced row echelon form

Returns

Matrix one possible reduced row echelon form of the given matrix

4.1.3.16 transpose()

```
Matrix Matrix::transpose (
            bool modify = false )  [inline]
```

Returns a new matrix which is the transpose of the original matrix.

Parameters

modify	if modify is true, then the given matrix is changed to its transpose

Returns

Matrix transpose of the given matrix

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

- Matrix.h
- Matrix.cpp

4.2 Polynomial Class Reference

Public Member Functions

- **Polynomial** (std::initializer_list< double > i, char x='x')
- **Polynomial** (char x='x')
- **Polynomial** (double d)
- int **degree** () const
- Polynomial **operator+** (const Polynomial p) const
- Polynomial **operator-** (const Polynomial p) const

Protected Attributes

- std::vector< double > **vec**
- char **variable**

Friends

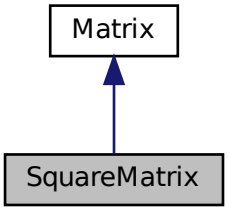
- Polynomial **operator-** (const Polynomial p)
- void **print** (const Polynomial, int, bool)

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

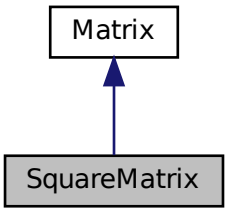
- Polynomial.h

4.3 SquareMatrix Class Reference

Inheritance diagram for SquareMatrix:



Collaboration diagram for SquareMatrix:



Public Member Functions

- **SquareMatrix** (int m, bool Identity=false)
- **SquareMatrix** (std::initializer_list< std::initializer_list< double > > i)
- **SquareMatrix** (const Matrix &m)
- int **order** () const
- double **det** () const
- SquareMatrix **inverse** () const

Additional Inherited Members

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

- squareMatrix.h

4.4 Vector Class Reference

Public Member Functions

- Vector ()

  Construct a new empty Vector object.
- Vector (std::initializer_list< double > init)

  Construct a new Vector object from an initializer list.
- Vector (int n)

  Construct a new Vector object and initialize it to the n-dimensional zero vector.
- ~Vector ()

  Default destructor.
- bool isZero () const

  Check if the vector is a zero vector.
- int size () const

  returns the dimension/length/size of the vector.
- double & operator[ ] (int index)

  access the element at the index-th index of the vector. Throws out_of_range error if the index is invalid.
- const double & operator[ ] (int index) const

  access (read-only) the element at the index-th index of the vector. Throws out_of_range error if the index is invalid.
- double & at (int index)

  access the element at the index-th index of the vector. Throws out_of_range error if the index is invalid.
- const double & at (int index) const

  access (read-only) the element at the index-th index of the vector. Throws out_of_range error if the index is invalid.
- Vector operator+ (const Vector &v) const

  computes the sum of self and Vector v.
- Vector operator- (const Vector &v) const

  computes the difference of self and Vector v.
- Vector operator* (const double &d) const

  computes the product of self and double d.
- Vector operator/ (double d) const

  returns the vector obtained by dividing self by double d. Throws an exception if d is zero.
- const Vector & operator+= (const Vector &v)

  adds Vector v to self. Returns a const reference to self for chaining like so: v2 += (v1 += v);
- const Vector & operator-= (const Vector &v)

  subtracts Vector v from self. Returns a const reference to self for chaining like so: v2 += (v1 -= v);
- const Vector & operator*= (const double &factor)

  *multiplies self by factor. Returns a const reference to self for chaining like so: v2 = (v1 *= 3);*
- const Vector & operator/= (double factor)

  divides self by factor. Returns a const reference to self for chaining like so: v2 = (v1 /= 3);
- double dot (const Vector &v) const

  Computes the dot product of Vectors self and v. Raises invalid_argument error if the dimensions do not match.
- double norm (int k=2) const

  Computes the k-norm of the Vector.
- Vector normalized (bool modify=false, int k=2)

  Normalizes the Vector according to its k-norm. throws invalid_argument exception when the k-norm is 0.
- std::vector< double >::const_iterator **begin** () const
- std::vector< double >::const_iterator **end** () const

Protected Member Functions

- Vector set_component_to_1 (int index, bool modify=false)

  scales the vector such that the element at the index is now 1. Throws an exception if the element at the index is 0.
- void **push_back** (const double &d)

Friends

- class **Matrix**

4.4.1 Constructor & Destructor Documentation

4.4.1.1 Vector() [1/3]

```
Vector::Vector ( )  [inline]
```

Construct a new empty Vector object.

4.4.1.2 Vector() [2/3]

```
Vector::Vector (
            std::initializer_list< double > init )  [inline]
```

Construct a new Vector object from an initializer list.

Parameters

init	The initializer list from where to construct the vector

4.4.1.3 Vector() [3/3]

```
Vector::Vector (
            int n )  [inline]
```

Construct a new Vector object and initialize it to the n-dimensional zero vector.

Parameters

n	the dimension of the vector

4.4.2 Member Function Documentation

4.4.2.1 at() [1/2]

```
double& Vector::at (
            int index )  [inline]
```

access the element at the index-th index of the vector. Throws out_of_range error if the index is invalid.

Parameters

index	index of the required value.

Returns

double&. the element at the required index.

4.4.2.2 at() [2/2]

```
const double& Vector::at (
            int index ) const  [inline]
```

access (read-only) the element at the index-th index of the vector. Throws out_of_range error if the index is invalid.

Parameters

index	index of the required value.

Returns

const double&. A const reference to the element at the required index.

4.4.2.3 dot()

```
double Vector::dot (
            const Vector & v ) const
```

Computes the dot product of Vectors self and v. Raises invalid_argument error if the dimensions do not match.

Parameters

v	The vector to compute the dot product with.

Returns

double. The computed dot product

4.4.2.4 isZero()

```
bool Vector::isZero ( ) const  [inline]
```

Check if the vector is a zero vector.

Returns

true if the vector is zero

false otherwise

4.4.2.5 norm()

```
double Vector::norm (
            int k = 2 ) const
```

Computes the k-norm of the Vector.

Parameters

k.	The norm required. Defaults to 2.

Returns

double. The computed norm.

4.4.2.6 normalized()

```
Vector Vector::normalized (
            bool modify = false,
            int k = 2 )
```

Normalizes the Vector according to its k-norm. throws invalid_argument exception when the k-norm is 0.

Parameters

modify	if true, normalizes self itself. Otherwise returns a new normalized Vector.
k	The type of norm required.

Returns

Vector. Either self(if modify is true) or a new Vector. In each case the returned Vector is normalized.

4.4.2.7 operator∗()

```
Vector Vector::operator* (
            const double & d ) const
```

computes the product of self and double d.

Parameters

v	The multiplying factor

Returns

Vector. The result of multiplication.

4.4.2.8 operator∗=()

```
const Vector & Vector::operator*= (
            const double & factor )
```

multiplies self by factor. Returns a const reference to self for chaining like so: v2 = (v1 ∗= 3);

Parameters

v	The double that is to be multiplied to self

Returns

const Vector&.

4.4.2.9 operator+()

```
Vector Vector::operator+ (
            const Vector & v ) const
```

computes the sum of self and Vector v.

Parameters

v	The Vector to add

Returns

Vector. The result of addition.

4.4.2.10 operator+=()

```
const Vector & Vector::operator+= (
            const Vector & v )
```

adds Vector v to self. Returns a const reference to self for chaining like so: v2 += (v1 += v);

Parameters

v	The vector that is to be added to self

Returns

const Vector&.

4.4.2.11 operator-()

```
Vector Vector::operator- (
            const Vector & v ) const
```

computes the difference of self and Vector v.

Parameters

v	The Vector to subtract

Returns

Vector. The result of subtraction.

4.4.2.12 operator-=()

```
const Vector & Vector::operator-= (
            const Vector & v )
```

subtracts Vector v from self. Returns a const reference to self for chaining like so: v2 += (v1 -= v);

Parameters

v	The vector that is to be subtracted from self

Returns

const Vector&.

4.4.2.13 operator/()

```
Vector Vector::operator/ (
            double d ) const
```

returns the vector obtained by dividing self by double d. Throws an exception if d is zero.

Parameters

v	The (nonzero)factor to divide by

Returns

Vector. The result of division.

4.4.2.14 operator/=()

```
const Vector & Vector::operator/= (
            double factor )
```

divides self by factor. Returns a const reference to self for chaining like so: v2 = (v1 /= 3);

Parameters

v	The double by which self is to be divided

Returns

const Vector&.

4.4.2.15 operator[]() [1/2]

```
double& Vector::operator[] (
            int index )  [inline]
```

access the element at the index-th index of the vector. Throws out_of_range error if the index is invalid.

Parameters

index	index of the required value.

Returns

double&. the element at the required index.

4.4.2.16 operator[]() [2/2]

```
const double& Vector::operator[] (
            int index ) const  [inline]
```

access (read-only) the element at the index-th index of the vector. Throws out_of_range error if the index is invalid.

Parameters

index	index of the required value.

Returns

const double&. A non-modifiable reference to the element at the required index.

4.4.2.17 set_component_to_1()

```
Vector Vector::set_component_to_1 (
            int index,
            bool modify = false )  [protected]
```

scales the vector such that the element at the index is now 1. Throws an exception if the element at the index is 0.

Parameters

index	The index to set to 1
modify	modifies the vector itself to be the scaled version if true. Returns a copy of the scaled object otherwise.

Returns

Vector

4.4.2.18 size()

```
int Vector::size ( ) const  [inline]
```

returns the dimension/length/size of the vector.

Note

see also len and dim.

Returns

int. The dimension of the vector

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

- Vector.h
- Vector.cpp

Chapter 5

Example Documentation

5.1 Matrix

Construct a new Matrix object from the given initializer list({{1,2},{3,4},{5,6}}) creates a 3∗2 matrix when byColumns is false and a 2∗3 matrix when byColumns is true.

Parameters

init	Initializer list used to create the matrix.
byColumns	True if the initializer list contains columns of the matrix, and false otherwise.

Index