

Namespace LinearAlgebra

Classes

[Identity](#)

This class represents an identity matrix in linear algebra. An identity matrix is a square matrix with ones on the main diagonal and zeros elsewhere.

[Matrix](#)

This class represents a matrix in linear algebra. A matrix is a two-dimensional array of complex numbers.

[Operations](#)

This class contains common linear algebra operations that can be performed on matrices and vectors.

[SparseMatrix](#)

A special representation of a [Matrix](#). A SparseMatrix stores only the non-zero elements to conserve memory and computation.

[Vector](#)

This class represents a vector in linear algebra. A vector is a one-dimensional array of complex numbers.

Class Identity

Namespace: [LinearAlgebra](#)





This class represents an identity matrix in linear algebra. An identity matrix is a square matrix with ones on the main diagonal and zeros elsewhere.

```
public class Identity : Matrix
```

Inheritance

[object](#)  ← [Matrix](#) ← Identity

Inherited Members

[Matrix.rows](#) , [Matrix.cols](#) , [Matrix.elements](#) , [Matrix.GetRow\(int\)](#) , [Matrix.GetColumn\(int\)](#) , [Matrix.this\[int, int\]](#) , [Matrix.Transpose\(\)](#) , [Matrix.TransposeInPlace\(\)](#) , [Matrix.Trace\(\)](#) , [Matrix.Conjugate\(\)](#) , [Matrix.ConjugateInPlace\(\)](#) , [Matrix.ToString\(\)](#) , [Matrix.Equals\(object\)](#) , [Matrix.GetHashCode\(\)](#) , [Matrix.AddInPlace\(Matrix\)](#) , [Matrix.SubtractInPlace\(Matrix\)](#) , [object.Equals\(object, object\)](#)  , [object.GetType\(\)](#)  , [object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

Identity(int)

Initializes a new instance of the [Identity](#) class.

```
public Identity(int size)
```

Parameters

size [int](#) 

The size of the matrix.

See Also

[Matrix](#)

Class Matrix

Namespace: [LinearAlgebra](#)

This class represents a matrix in linear algebra. A matrix is a two-dimensional array of complex numbers.

```
public class Matrix
```





Inheritance

[object](#)  ← Matrix

Derived

[Identity](#)

Inherited Members

[object.Equals\(object, object\)](#)  , [object.GetType\(\)](#)  , [object.MemberwiseClone\(\)](#)  ,
[object.ReferenceEquals\(object, object\)](#) 

Constructors

Matrix(Vector)

Initializes a new instance of the [Matrix](#) class.

```
public Matrix(Vector elements)
```

Parameters

elements [Vector](#)

The elements as a [Vector](#)

Matrix(int, int)

Initializes a new instance of the [Matrix](#) class.

```
public Matrix(int rows, int columns)
```

Parameters

rows [int](#)

The number of rows.

columns [int](#)

The number of columns.

Matrix(int, int, Complex[])

A constructor for a matrix

```
public Matrix(int rows, int cols, Complex[] elements)
```

Parameters

rows [int](#)

cols [int](#)

elements [Complex](#)[]

Exceptions

[ArgumentException](#)

Matrix(Complex[,])

Initializes a new instance of the [Matrix](#) class.

```
public Matrix(Complex[,] elements)
```

Parameters

elements [Complex](#)[,]

The elements of the Matrix.

Exceptions

[ArgumentNullException](#) 

elements

Fields

elements

A [rows](#) x [cols](#) array of Complex numbers


```
public Complex[,] elements
```

Field Value

[Complex](#)  [,]

Properties

this[int, int]

Gets or sets the element with the specified [Complex](#)  number.

```
public Complex this[int i, int j] { get; set; }
```

Parameters

i [int](#) 

The row index.

j [int](#) 

The column index.

Property Value

[Complex](#) 

The [Complex](#).

cols

The number of columns

```
public int cols { get; }
```

Property Value

[int](#)

rows

The number of rows

```
public int rows { get; }
```

Property Value

[int](#)

Methods

AddInPlace(Matrix)

Adds the Matrices in place.

```
public void AddInPlace(Matrix matrixOther)
```

Parameters

matrixOther [Matrix](#)

The matrix to add.

Exceptions

[ArgumentException](#) 

The dimensions of both matrices must match.

Conjugate()

Conjugates this instance.

```
public Matrix Conjugate()
```

Returns

[Matrix](#)

A conjugated [Matrix](#)

ConjugateInPlace()

Conjugates the [Matrix](#) in place.

```
public void ConjugateInPlace()
```

Equals(object?)

Equalses the specified object.

```
public override bool Equals(object? obj)
```

Parameters

obj [object](#) 

The object.

Returns

[bool](#)

GetColumn(int)

Gets the column.

```
public Complex[] GetColumn(int columnNumber)
```

Parameters

columnNumber [int](#)

The column index.

Returns

[Complex](#)[]

A specific column of a Matrix as an array of [Complex](#)

Exceptions

[ArgumentOutOfRangeException](#)

columnNumber - Column number is out of bounds.

GetHashCode()

Gets the hash code.

```
public override int GetHashCode()
```

Returns

[int](#)

GetRow(int)

Gets the row.

```
public Complex[] GetRow(int rowNumber)
```

Parameters

rowNumber [int](#)

The row index.

Returns

[Complex](#) []

A specific row of a Matrix as an array of [Complex](#)

Exceptions

[ArgumentOutOfRangeException](#)

rowNumber - Row number is out of bounds.

SubtractInPlace(Matrix)

Subtracts the Matrices in place.

```
public void SubtractInPlace(Matrix matrixOther)
```

Parameters

matrixOther [Matrix](#)

The other matrix.

Exceptions

[ArgumentException](#)

The dimensions of both matrices must match.

ToString()

Converts to string.

```
public override string ToString()
```

Returns

[string](#) 

Trace()

Calculates the trace this instance.

```
public Complex Trace()
```

Returns

[Complex](#) 

The trace of the Matrix as a [Complex](#) .

Exceptions

[InvalidOperationException](#) 

Trace is only defined for square matrices.

Transpose()

Transposes this instance.

```
public Matrix Transpose()
```

Returns

[Matrix](#)

A transposed [Matrix](#)

TransposeInPlace()

Transposes the [Matrix](#) in place.

```
public void TransposeInPlace()
```

Exceptions

[InvalidOperationException](#)[↗](#)

In-place transpose can only be performed on square matrices.

Operators

operator +(Matrix, Matrix)

Implements the operator op_Addition.

```
public static Matrix operator +(Matrix matrix1, Matrix matrix2)
```

Parameters

matrix1 [Matrix](#)

The first matrix.

matrix2 [Matrix](#)

The second matrix.

Returns

[Matrix](#)

The result of the additon.

operator ==(Matrix, Matrix)

Implements the operator op_Equality.

```
public static bool operator ==(Matrix a, Matrix b)
```

Parameters

a [Matrix](#)

The first matrix.

b [Matrix](#)

The second matrix.

Returns

[bool](#)

The result of the operator.

operator !=(Matrix, Matrix)

Implements the operator op_Inequality.

```
public static bool operator !=(Matrix a, Matrix b)
```

Parameters

a [Matrix](#)

The first matrix.

b [Matrix](#)

The second matrix.

Returns

[bool](#)

The result of the operator.

operator *(Matrix, Matrix)

Implements the operator op_Multiply.

```
public static Matrix operator *(Matrix matrix1, Matrix matrix2)
```

Parameters

matrix1 [Matrix](#)

The first matrix.

matrix2 [Matrix](#)

The second matrix.

Returns

[Matrix](#)

The result of the subtraction.

operator *(Matrix, Vector)

Implements the operator op_Multiply.

```
public static Vector operator *(Matrix matrix, Vector vector)
```

Parameters

matrix [Matrix](#)

The matrix.

vector [Vector](#)

The vector.

Returns

[Vector](#)

The result of the multiplication.

operator *(Matrix, Complex)

Implements the operator op_Multiply.

```
public static Matrix operator *(Matrix matrix, Complex scalar)
```

Parameters

matrix [Matrix](#)

The matrix.

scalar [Complex](#)[↗]

The scalar.

Returns

[Matrix](#)

The result of the multiplication.

operator -(Matrix, Matrix)

Implements the operator op_Subtraction.

```
public static Matrix operator -(Matrix matrix1, Matrix matrix2)
```

Parameters

matrix1 [Matrix](#)

The first matrix.

`matrix2` [Matrix](#)

The second matrix.

Returns

[Matrix](#)

The result of the operator.


Class Operations

Namespace: [LinearAlgebra](#)








This class contains common linear algebra operations that can be performed on matrices and vectors.

```
public class Operations
```

Inheritance

[object](#)  ← Operations

Inherited Members

[object.Equals\(object\)](#)  , [object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  , [object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#)  , [object.ToString\(\)](#) 

Methods

Add(Matrix, Matrix)

Adds the instance with the specified matrix.

```
public static Matrix Add(Matrix matrix1, Matrix matrix2)
```

Parameters

matrix1 [Matrix](#)

The first matrix.

matrix2 [Matrix](#)

The second matrix.

Returns

[Matrix](#)

Exceptions

[ArgumentException](#)↗

The dimensions of both matrices must match.

Determinant(Matrix)

Calculate determinant of a [Matrix](#)

```
public static double Determinant(Matrix matrix)
```

Parameters

matrix [Matrix](#)

Returns

[double](#)↗

Exceptions

[ArgumentException](#)↗

EuclideanNorm(Vector)

A method to calculate the Euclidean norm of a [Vector](#).

The Euclidean norm refers to the squart root of the sum of the squares of the elements of the [Vector](#).

```
public static double EuclideanNorm(Vector vector)
```


Parameters

vector [Vector](#)

Returns

[double](#)↗

EuclideanNormAsComplex(Vector)

A method to calculate the Euclidean norm of a [Vector](#) as a [Complex](#) 

```
public static Complex EuclideanNormAsComplex(Vector vector)
```

Parameters

vector [Vector](#)

Returns

[Complex](#) 

GenerateIdentityMatrix(int)

Return a [Matrix](#) with all ones along the diagonal

```
public static Matrix GenerateIdentityMatrix(int size)
```

Parameters

size [int](#) 

Returns

[Matrix](#)

InnerProduct(Vector, Vector)

A method to perform the inner product (dot product) of two [Vector](#).

```
public static Complex InnerProduct(Vector vector1, Vector vector2)
```

Parameters

vector1 [Vector](#)

vector2 [Vector](#)

Returns

[Complex](#)[↗](#)

Exceptions

[ArgumentNullException](#)[↗](#)

[ArgumentException](#)[↗](#)

Invert(Matrix)

A method to convert the inverse of a [Matrix](#).

```
public static Matrix Invert(Matrix matrix)
```

Parameters

matrix [Matrix](#)

Returns

[Matrix](#)

Exceptions

[InvalidOperationException](#)[↗](#)

IsEqual(Matrix, Matrix)

Determines whether the specified a is equal.

```
public static bool IsEqual(Matrix a, Matrix b)
```

Parameters

a [Matrix](#)

The first matrix.

b [Matrix](#)

The second matrix.

Returns

[bool](#)

`true` if the specified a is equal; otherwise, `false`.

JoinMatrices(Matrix, Matrix)

A method for joining two [Matrix](#).

```
public static Matrix JoinMatrices(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` [Matrix](#)

`matrix2` [Matrix](#)

Returns

[Matrix](#)

MatrixMultiply(Matrix, Matrix)

Multiplies the matrices concurrently.

```
public static Matrix MatrixMultiply(Matrix a, Matrix b)
```

Parameters

a [Matrix](#)

The first matrix.

b [Matrix](#)

The second matrix.

Returns

[Matrix](#)

Exceptions

[ArgumentException](#) 

The number of columns in the first matrix must match the number of rows in the second matrix.

MatrixVectorMult(Matrix, Vector)

Multiplies the matrix with the vector.

```
public static Vector MatrixVectorMult(Matrix matrix, Vector vector)
```

Parameters

matrix [Matrix](#)

The matrix.

vector [Vector](#)

The vector.

Returns

[Vector](#)

A [Vector](#)

Exceptions

[ArgumentException](#) 

Left Multiplications must have similar dimensions.

Multiply(Matrix, Matrix)

Multiplies the specified matrix with the instance.

```
public static Matrix Multiply(Matrix matrix1, Matrix matrix2)
```

Parameters

matrix1 [Matrix](#)

The first matrix.

matrix2 [Matrix](#)

The second matrix.

Returns

[Matrix](#)

Exceptions

[ArgumentException](#) 

The number of columns in the first matrix must match the number of rows in the second matrix.

Multscaler(Matrix, Complex)

Multiplies a matrix with a scalar.

```
public static Matrix Multscaler(Matrix matrix1, Complex scaler)
```

Parameters

matrix1 [Matrix](#)

The matrix.

scaler [Complex](#)

The scaler.

Returns

[Matrix](#)

OuterProduct(Vector, Vector)

A method to perform the outer product (cross product) of two [Vector](#).

```
public static Matrix OuterProduct(Vector vector1, Vector vector2)
```

Parameters

vector1 [Vector](#)

vector2 [Vector](#)

Returns

[Matrix](#)

Exceptions

[ArgumentNullException](#)

[ArgumentException](#)

Subtract(Matrix, Matrix)

Subtracts the specified matrix from the instance.

```
public static Matrix Subtract(Matrix matrix1, Matrix matrix2)
```

Parameters

matrix1 [Matrix](#)

The first matrix.

`matrix2` [Matrix](#)

The second matrix.

Returns

[Matrix](#)

Exceptions

[ArgumentException](#) 

The dimensions of both matrices must match.

TensorProduct(Matrix, Matrix)

Tensors the product.

```
public static Matrix TensorProduct(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` [Matrix](#)

The matrix1.

`matrix2` [Matrix](#)

The matrix2.

Returns

[Matrix](#)

TensorProductofVectors(Vector, Vector)

A method to tensor two [Vector](#) together.


```
public static Vector TensorProductofVectors(Vector vector1, Vector vector2)
```

Parameters

vector1 [Vector](#)

vector2 [Vector](#)

Returns

[Vector](#)

A [Vector](#).

Class SparseMatrix

Namespace: [LinearAlgebra](#)








A special representation of a [Matrix](#). A SparseMatrix stores only the non-zero elements to conserve memory and computation.

```
public class SparseMatrix
```

Inheritance

[object](#)  ← SparseMatrix

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Constructors

SparseMatrix(int, int)

Constructor to initialize a sparse matrix of a given size

```
public SparseMatrix(int rows, int cols)
```

Parameters

rows [int](#) 

cols [int](#) 

SparseMatrix(Complex[,])

Constructor to initialize a sparse matrix of a given [Matrix](#)

```
public SparseMatrix(Complex[,] matrix)
```

Parameters

`matrix` [Complex](#)[]

Properties

Cols

The number of columns

```
public int Cols { get; }
```

Property Value

[int](#)

this[int, int]

Index the element at row x col

```
public Complex this[int row, int col] { get; set; }
```

Parameters

`row` [int](#)

`col` [int](#)

Property Value

[Complex](#)

Rows

The number of rows

```
public int Rows { get; }
```

Property Value

[int](#)

Methods

FromMatrix(Matrix)

A static method to convert a [Matrix](#) to a [SparseMatrix](#)

```
public static SparseMatrix FromMatrix(Matrix matrix)
```

Parameters

matrix [Matrix](#)

Returns

[SparseMatrix](#)

Identity(int)

Return the identity matrix as a SparseMatrix

```
public static SparseMatrix Identity(int size)
```

Parameters

size [int](#)

Returns

[SparseMatrix](#)

Multiply(SparseMatrix)

A method to perform the matrix multiplication between two [SparseMatrix](#). Note that this is a right multiplication, so it computes $\text{self} * \text{other}$

```
public SparseMatrix Multiply(SparseMatrix other)
```

Parameters

other [SparseMatrix](#)

Returns

[SparseMatrix](#)

Exceptions

[InvalidOperationException](#) 

MultiplyWithVector(Complex[])

A method to perform a $\text{SparseMatrix} * \text{Vector}$ multiplication

```
public Complex[] MultiplyWithVector(Complex[] vector)
```

Parameters

vector [Complex](#)  []

Returns

[Complex](#)  []

Exceptions

[InvalidOperationException](#) 

ParallelTensorProduct(SparseMatrix)

A method to perform the tensor product with another [SparseMatrix](#) that is parallelized to increase performance on large matrices.

```
public SparseMatrix ParallelTensorProduct(SparseMatrix other)
```

Parameters

other [SparseMatrix](#)

Returns

[SparseMatrix](#)

Exceptions

[NullReferenceException](#)[↗](#)

Print()

Method to display the sparse matrix (for debugging purposes)

```
public void Print()
```

TensorProduct(SparseMatrix)

A methdo to perform the tensor product with another [SparseMatrix](#)

```
public SparseMatrix TensorProduct(SparseMatrix other)
```

Parameters

other [SparseMatrix](#)

Returns

[SparseMatrix](#)

Class Vector

Namespace: [LinearAlgebra](#)







This class represents a vector in linear algebra. A vector is a one-dimensional array of complex numbers.

```
public class Vector
```

Inheritance

[object](#)  ← Vector

Inherited Members

[object.Equals\(object\)](#)  , [object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  , [object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

Vector(int)

Initializes a new instance of the [Vector](#) class.

```
public Vector(int rows)
```

Parameters

rows [int](#) 

The number of rows (or number of elements) in the vector.

Vector(Complex[])

Initializes a new instance of the [Vector](#) class.

```
public Vector(Complex[] elements)
```

Parameters

`elements` [Complex](#)[]

The elements of the Vector.

Fields

cols

The number of columns in the vector.

```
public int cols
```

Field Value

[int](#)

elements

The elements of the vector.

```
public Complex[] elements
```

Field Value

[Complex](#)[]

rows

The number of rows in the vector.

```
public int rows
```

Field Value

[int](#)

Properties

ToMatrix

Converts to matrix.

```
public Matrix ToMatrix { get; }
```

Property Value

[Matrix](#)

To [Matrix](#).

Methods

Conjugate()

Conjugates this [Vector](#) instance.

```
public Vector Conjugate()
```

Returns

[Vector](#)

A conjugated [Vector](#)

ConjugateInPlace()

Conjugates the [Vector](#) in place. This method modifies the original [Vector](#).

```
public void ConjugateInPlace()
```

GetState()

Gets the state of the [Vector](#).

```
public Complex[] GetState()
```

Returns

[Complex](#)[]

An array of [Complex](#) of the elements.

IsApproximatelyEqual(Vector, double)

Determines whether [is approximately equal] [the specified other].

```
public bool IsApproximatelyEqual(Vector other, double tolerance = 1E-10)
```

Parameters

other [Vector](#)

The other [Vector](#).

tolerance [double](#)

The tolerance. Default tolerance is 1e-10

Returns

[bool](#)

true if [is approximately equal] [the specified other]; otherwise, **false**.

IsColVector()

Determines whether [is col vector].

```
public bool IsColVector()
```

Returns

[bool](#)

`true` if [is col vector]; otherwise, `false`.

IsRowVector()

Determines whether [is row vector].

```
public bool IsRowVector()
```

Returns

[bool](#)

`true` if [is row vector]; otherwise, `false`.

ToString()

Converts to string.

```
public override string ToString()
```

Returns

[string](#)

A [string](#) that represents this instance.

Transpose(Vector)

Transposes the specified vector.

```
public static Vector Transpose(Vector vector)
```

Parameters

`vector` [Vector](#)

The vector.

Returns

[Vector](#)

A transposed [Vector](#)

TransposeInPlace()

Transposes the [Vector](#) in place. This method modifies the original [Vector](#).

```
public void TransposeInPlace()
```

Namespace QuantumCircuits

Classes

[CX](#)

Constructs an Controlled Not (CX) Gate

[CircuitExecution](#)

This class is responsible for executing a quantum circuit

[Gate](#)

Constructor for a Quantum Gate

[H](#)

Constructs a Hadamard Gate

[NOP](#)

Constructs a NOP Gate which acts as an empty space

[QuantumCircuitBuilder](#)

This class is used to build a quantum circuit by adding gates to the circuit lines.

[RX](#)

Constructs a Rotate X Gate (RX)

[RY](#)

Constructs a Rotate Y Gate (RY)

[RZ](#)

Constructs a Rotate Z Gate (RZ)

[SWAP](#)

Constructs a SWAP Gate (SWP)

[T](#)

Constructs a T Gate

[Toff](#)

Constructs a Toffoli Gate (Toff/TOF)

[X](#)

Constructs an X Gate

[Y](#)

Constructs a Y Gate

[Z](#)

Constructs an Z Gate

Enums

[GateTypes](#)

An [Enum](#) of the supported Gate Types

Class CX

Namespace: [QuantumCircuits](#)







Constructs an Controlled Not (CX) Gate

```
public class CX : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← CX

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

CX(int, int)

Constructs an Controlled Not (CX) Gate

```
public CX(int control, int target)
```

Parameters

control [int](#) 

target [int](#) 

Class CircuitExecution

Namespace: [QuantumCircuits](#)

This class is responsible for executing a quantum circuit

```
public class CircuitExecution
```

Inheritance

[object](#) ← CircuitExecution

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#)

Constructors

CircuitExecution(QuantumCircuitBuilder)

Initializes a new instance of the [CircuitExecution](#) class.

```
public CircuitExecution(QuantumCircuitBuilder inputcircuit)
```

Parameters

inputcircuit [QuantumCircuitBuilder](#)

The inputcircuit.

Properties

QbitCount

Gets the qbit count.

```
public int QbitCount { get; }
```


Property Value

[int](#)

The qbit count.

StateVector

Gets the state vector.

```
public Complex[] StateVector { get; }
```

Property Value

[Complex](#) []

The state vector.

Methods

CNOTCreation(int, int, int)

Creates a tensored CNOT operator matrix.

```
public SparseMatrix CNOTCreation(int gatesize, int controlbit, int targetbit)
```

Parameters

gatesize [int](#)

The size of the gate.

controlbit [int](#)

The control qubit.

targetbit [int](#)

The target qubit.

Returns

[SparseMatrix](#)

A [SparseMatrix](#) representation of the tensored CNOT operator matrix

ExecuteCircuit()

Executes the quantum circuit.

```
public Vector ExecuteCircuit()
```

Returns

[Vector](#)

The result statevector as a [Vector](#)

GetStateProbabilities()

Calculates the probability distribution for the entire system.

```
public double[] GetStateProbabilities()
```

Returns

[double](#)[↗][]

An array of doubles representing the probability of each basis state.

MeasureAllQubits()

Measures all qubits.

```
public byte[] MeasureAllQubits()
```

Returns

[byte](#)[]

An array of [byte](#) of bits for the measured state.

PrintBitstrings(int)

Prints simulated measurement bitstring(s) to the console.

```
public void PrintBitstrings(int iterations = 1)
```

Parameters

iterations [int](#)

The number of simulations to perform.

PrintHistogram(int)

Prints a sideways histogram based on the probabilities of each basis state alongside their respective probability.

```
public void PrintHistogram(int bars = 100)
```

Parameters

bars [int](#)

SimulateHistogram(int, int)

Prints a sideways histogram of simulated measurement results, normalized to a specified number of bars.

```
public void SimulateHistogram(int iterations = 1000, int bars = 100)
```

Parameters

iterations [int](#)

The number of simulations to perform.

bars [int](#)

The total number of hyphens to display in the histogram.

SimulateMeasurements(int)

Simulates measurements on the entire quantum system.

```
public List<string> SimulateMeasurements(int iterations = 1)
```

Parameters

iterations [int](#)

The number of simulations to perform.

Returns

[List](#) <[string](#)>

A list of bitstrings representing the measurement outcomes.

SwapCreation(int, int, int)

Creates a tensored SWAP operator matrix.

```
public SparseMatrix SwapCreation(int gatesize, int target1, int target2)
```

Parameters

gatesize [int](#)

The size of the gate.

target1 [int](#)

The first target qubit.

target2 [int](#)

The second target qubit.

Returns

[SparseMatrix](#)

A [SparseMatrix](#) representation of the tensored SWAP operator matrix

ToString()

Converts to string.

```
public override string ToString()
```

Returns

[string](#)

A [string](#) that represents this instance.

ToffoliCreation(int, int, int, int)

Creates a tensored Toffoli operator matrix.

```
public SparseMatrix ToffoliCreation(int gatesize, int controlbit1, int controlbit2,  
int targetbit)
```

Parameters

gatesize [int](#)

The size of the gate.

controlbit1 [int](#)

The first control qubit.

controlbit2 [int](#)

The second control qubit.

`targetbit` [int](#)

The target qubit.

Returns

[SparseMatrix](#)

A [SparseMatrix](#) representation of the tensored Tofolli operator matrix

Class Gate

Namespace: [QuantumCircuits](#)

Constructor for a Quantum Gate

```
public class Gate
```







Inheritance

[object](#)  ← Gate

Derived

[CX](#), [H](#), [NOP](#), [RX](#), [RY](#), [RZ](#), [SWAP](#), [T](#), [Toff](#), [X](#), [Y](#), [Z](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) 

Constructors

Gate(GateTypes, SparseMatrix, int[], int[])

Constructor for a Quantum Gate

```
public Gate(GateTypes type, SparseMatrix operation, int[] controls, int[] targets)
```

Parameters

type [GateTypes](#)

The [GateTypes](#) of the gate

operation [SparseMatrix](#)

A [SparseMatrix](#) of the gate's operator matrix

controls [int](#)  []

The control qubits as an array of qubits (as indexes)

`targets` [int\[\]](#)

The target qubits as an array of qubits (as indexes)

Properties

Controls

The control qubits

```
public int[] Controls { get; protected set; }
```

Property Value

[int\[\]](#)

Operation

The operator matrix of the gate

```
public SparseMatrix Operation { get; }
```

Property Value

[SparseMatrix](#)

Targets

The target qubits

```
public int[] Targets { get; protected set; }
```

Property Value

[int\[\]](#)

Type

The type of gate. See [GateTypes](#)

```
public GateTypes Type { get; }
```

Property Value

[GateTypes](#)

Methods

ToString()

Returns the [string](#)[↗] representation of the Gate

```
public override string ToString()
```

Returns

[string](#)[↗]

Enum GateTypes

Namespace: [QuantumCircuits](#)

An [Enum](#) of the supported Gate Types

```
public enum GateTypes
```

Fields

CXC = 6

Controlled Not Gate Control

CXT = 5

Controlled Not Gate Target

HGT = 4

Hadamard Gate

NOP = 14

No operation gate used to fill gaps

RXT = 11

Rx Gate

RYT = 12

Ry Gate

RZT = 13

Rz Gate

SWP = 7

Swap Gate target

SWT = 8

Swap Gate target

TGT = 3

T Gate

TOC = 10

Toffoli Gate controls

TOF = 9

Toffoli Gate

XGT = 0

Not Gate

YGT = 1

Y Gate

ZGT = 2

Z Gate

Class H

Namespace: [QuantumCircuits](#)







Constructs a Hadamard Gate

```
public class H : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← H

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

H(int)

Constructs a Hadamard Gate

```
public H(int target)
```

Parameters

target [int](#) 

Class NOP

Namespace: [QuantumCircuits](#)







Constructs a NOP Gate which acts as an empty space

```
public class NOP : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← NOP

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

NOP(int, GateTypes)

Constructs a NOP Gate which acts as an empty space

```
public NOP(int target, GateTypes type)
```

Parameters

target [int](#) 

type [GateTypes](#)

Class QuantumCircuitBuilder

Namespace: [QuantumCircuits](#)

This class is used to build a quantum circuit by adding gates to the circuit lines.

```
public class QuantumCircuitBuilder
```

Inheritance

[object](#) ← QuantumCircuitBuilder

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#)

Constructors

QuantumCircuitBuilder(int, int)

Initializes a new instance of the [QuantumCircuitBuilder](#) class.

```
public QuantumCircuitBuilder(int numQuantumLines, int numClassicalLines)
```

Parameters

numQuantumLines [int](#)

The number quantum lines.

numClassicalLines [int](#)

The number classical lines.

Fields

classicalLines

The classical lines represent the series of classical gates that are applied to a classical bit.

```
public List<Gate>[] classicalLines
```

Field Value

[List](#) <[Gate](#)> []

quantumLines

The quantum lines represent the series of quantum gates that are applied to a qubit.

```
public List<Gate>[] quantumLines
```

Field Value

[List](#) <[Gate](#)> []

Methods

AddGateCX(int, int)

Adds the gate CX (Controlled Not).

```
public void AddGateCX(int control, int target)
```

Parameters

control [int](#)

The control qubit.

target [int](#)

The target qubit.

Exceptions

[ArgumentException](#)

target and control cannot be the same value or target or control outside of circuit bounds

AddGateH(int)

Adds the gate H.

```
public void AddGateH(int target)
```

Parameters

target [int](#)

The target qubit

Exceptions

[ArgumentException](#)

target outside of circuit bounds

AddGateRX(int, double)

Adds the gate RX.

```
public void AddGateRX(int target, double theta)
```

Parameters

target [int](#)

The target qubit

theta [double](#)

Angle of rotation in radians

Exceptions

[ArgumentException](#)

target outside of circuit bounds

AddGateRY(int, double)

Adds the gate RY.

```
public void AddGateRY(int target, double theta)
```

Parameters

target [int](#)

The target qubit

theta [double](#)

Angle of rotation in radians

Exceptions

[ArgumentException](#)

target outside of circuit bounds

AddGateRZ(int, double)

Adds the gate RZ.

```
public void AddGateRZ(int target, double theta)
```

Parameters

target [int](#)

The target qubit

theta [double](#)

Angle of rotation in radians

Exceptions

[ArgumentException](#) 

target outside of circuit bounds

AddGateSWP(int, int)

Adds the gate SWP.

```
public void AddGateSWP(int target1, int target2)
```

Parameters

target1 [int](#) 

The first target qubit.

target2 [int](#) 

The second target qubit.

Exceptions

[ArgumentException](#) 

target and control cannot be the same value or target or control outside of circuit bounds

AddGateT(int)

Adds the gate T.

```
public void AddGateT(int target)
```

Parameters

target [int](#) 

The target qubit

Exceptions

[ArgumentException](#)

target outside of circuit bounds

AddGateTOF(int, int, int)

Adds the gate TOF.

```
public void AddGateTOF(int control1, int control2, int target)
```

Parameters

control1 [int](#)

The first control qubit.

control2 [int](#)

The second control qubit.

target [int](#)

The target qubit.

Exceptions

[ArgumentException](#)

target or controls can be the same value or target or control outside of circuit bounds

AddGateX(int)

Adds the gate X.

```
public void AddGateX(int target)
```

Parameters

target [int](#)

The target qubit

Exceptions

[ArgumentException](#)

target outside of circuit bounds

AddGateY(int)

Adds the gate Y.

```
public void AddGateY(int target)
```

Parameters

target [int](#)

The target qubit

Exceptions

[ArgumentException](#)

target outside of circuit bounds

AddGateZ(int)

Adds the gate Z.

```
public void AddGateZ(int target)
```

Parameters

target [int](#)

The target qubit

Exceptions

[ArgumentException](#) 

target outside of circuit bounds

GetBoxedAsciiSymbol(Gate)

Helper method for circuit ToString. Makes an array of strings to represent a gate.

```
public string[] GetBoxedAsciiSymbol(Gate gate)
```

Parameters

gate [Gate](#)

The [Gate](#) to generate ASCII symbols for.

Returns

[string](#)  []

An array of [string](#)  array representing the ASCII symbol of the gate.

ToString()

Converts to string.

```
public override string ToString()
```

Returns

[string](#) 

A [string](#)  that represents this instance.

Class RX

Namespace: [QuantumCircuits](#)







Constructs a Rotate X Gate (RX)

```
public class RX : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← RX

Inherited Members

[Gate.Type](#), [Gate.Operation](#), [Gate.Controls](#), [Gate.Targets](#), [Gate.ToString\(\)](#), [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) 

Constructors

RX(int, double)

Constructs a Rotate X Gate (RX)

```
public RX(int target, double theta)
```

Parameters

target [int](#) 

theta [double](#) 

Class RY

Namespace: [QuantumCircuits](#)







Constructs a Rotate Y Gate (RY)

```
public class RY : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← RY

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

RY(int, double)

Constructs a Rotate Y Gate (RY)

```
public RY(int target, double theta)
```

Parameters

target [int](#) 

theta [double](#) 

Class RZ

Namespace: [QuantumCircuits](#)







Constructs a Rotate Z Gate (RZ)

```
public class RZ : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← RZ

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

RZ(int, double)

Constructs a Rotate Z Gate (RZ)

```
public RZ(int target, double theta)
```

Parameters

target [int](#) 

theta [double](#) 

Class SWAP

Namespace: [QuantumCircuits](#)







Constructs a SWAP Gate (SWP)

```
public class SWAP : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← SWAP

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

SWAP(int, int)

Constructs a SWAP Gate (SWP)

```
public SWAP(int target1, int target2)
```

Parameters

target1 [int](#) 

target2 [int](#) 

Class T

Namespace: [QuantumCircuits](#)







Constructs a T Gate

```
public class T : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← T

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

T(int)

Constructs a T Gate

```
public T(int target)
```

Parameters

target [int](#) 

Class Toff

Namespace: [QuantumCircuits](#)







Constructs a Toffoli Gate (Toff/TOF)

```
public class Toff : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← Toff

Inherited Members

[Gate.Type](#), [Gate.Operation](#), [Gate.Controls](#), [Gate.Targets](#), [Gate.ToString\(\)](#), [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) 


Constructors


Toff(int, int, int)

Constructs a Toffoli Gate (Toff/TOF)

```
public Toff(int control1, int control2, int target)
```

Parameters

control1 [int](#) 

control2 [int](#) 

target [int](#) 

Class X

Namespace: [QuantumCircuits](#)







Constructs an X Gate

```
public class X : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← X

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

X(int)

Constructs an X Gate

```
public X(int target)
```

Parameters

target [int](#) 

Class Y

Namespace: [QuantumCircuits](#)







Constructs a Y Gate

```
public class Y : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← Y

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

Y(int)

Constructs a Y Gate

```
public Y(int target)
```

Parameters

target [int](#) 

Class Z

Namespace: [QuantumCircuits](#)







Constructs an Z Gate

```
public class Z : Gate
```

Inheritance

[object](#)  ← [Gate](#) ← Z

Inherited Members

[Gate.Type](#) , [Gate.Operation](#) , [Gate.Controls](#) , [Gate.Targets](#) , [Gate.ToString\(\)](#) , [object.Equals\(object\)](#)  ,
[object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  ,
[object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#) 

Constructors

Z(int)

Constructs an Z Gate

```
public Z(int target)
```

Parameters

target [int](#) 