Package: QuantumOps (via r-universe)

August 27, 2024

Title Performs Common Linear Algebra Operations Used in Quantum Computing and Implements Quantum Algorithms

Version 3.0.1

Date 2020-02-01

Author Salonik Resch

Maintainer Salonik Resch <resc0059@umn.edu>

Description Contains basic structures and operations used frequently in quantum computing. Intended to be a convenient tool to help learn quantum mechanics and algorithms. Can create arbitrarily sized kets and bras and implements quantum gates, inner products, and tensor products. Creates arbitrarily controlled versions of all gates and can simulate complete or partial measurements of kets. Has functionality to convert functions into equivalent quantum gates and model quantum noise. Includes larger applications, such as Steane error correction <DOI:10.1103/physrevlett.77.793>, Quantum Fourier Transform and Shor's algorithm (Shor 1999), Grover's algorithm (1996), Quantum Approximation Optimization Algorithm (QAOA) (Farhi, Goldstone, and Gutmann 2014) <arXiv:1411.4028>, and a variational quantum classifier (Schuld 2018) <arXiv:1804.00633>. Can be used with the gridsynth algorithm <arXiv:1212.6253> to perform decomposition into the Clifford+T set.

Depends R (>= 3.1.0)

License GPL-3

RoxygenNote 5.0.1

NeedsCompilation no

Date/Publication 2020-02-03 09:20:18 UTC

Repository https://resc0059.r-universe.dev

RemoteUrl https://github.com/cran/QuantumOps

RemoteRef HEAD

RemoteSha 35e2a8be5a6bbefbdc53a732eb6145a04dcd9e8e

Contents

---

addmod2 *addmod2*

Description

Takes two integers and adds their bits modulus two. The resulting string of bits represents an integer, that value of which is the output.

Usage

```
addmod2(x,a)
```

Arguments

x	integer
a	integer

Value

Integer resulting from the bit-wise addition of two number modulus 2

Examples

```
addmod2(5,5)
addmod2(1,2)
```

---

adjoint *adjoint*

Description

Finds the adjoint of the input. An input ket will become a bra and input bra will become a ket

Usage

```
adjoint(x)
```

Arguments

x	A ket (column vector), bra (row vecor), or gate (matrix)

Value

The adjoint of x

Examples

```
adjoint(ket(1,5))
```

AmplitudeDamping *AmplitudeDamping*

Description

Implements Amplitude Damping noise model on the input quantum state. Formulas taken from <DOI:10.1103/PhysRevA.90.062320>.

Usage

```
AmplitudeDamping(p,Pad)
```

Arguments

p	Input quantum state, in density matrix format
Pad	The probability of Amplitude Damping. Commonly referred to as gamma in the literature.

Value

The quantum state, after Amplitude Damping has been applied.

Examples

```
AmplitudeDamping(p=convert_ket2DM(ket(1,0)),Pad=0.01)
```

BELL *BELL*

Description

If no argument is supplied, returns the matrix of BELL gate. If ket given as input, applies a BELL gate to the input ket and returns the resulting ket

Usage

```
BELL(...)
```

Arguments

...	No argument, or 4 dimensional (2 qubit) ket (column vector) that is input to the gate

Value

Matix of the BELL gate or ket after a BELL gate is applied

Examples

```
BELL(ket(1,1,1,1))
BELL()
```

`bra` *bra*

Description

Returns a normalized bra (row vector)

Usage

```
bra(...)
```

Arguments

`...`	Variable number of numbers representing the amplitudes of the bra

Value

Row vector containing normalized amplitudes of a bra

Examples

```
bra(1,0,1,2)
```

`CFA` *CFA*

Description

Performs the continued fractions algorithm to find a fraction close to input value

Usage

```
CFA(y,epsilon=1e-2)
```

Arguments

`y`	Value that function attempts to find. Typically comes from measurement of Shor's algorithm
`epsilon`	Acceptable error between value and fraction

Value

Vector containing numerator and denominator of fraction

Examples

```
CFA(285/14)
```

---

checkCases *checkCases*

Description

Takes in a matrix of clauses and determines what percentage of the clauses each possible value satisfies.

Usage

```
checkCases(clauses,colorCode=FALSE)
```

Arguments

clauses	Matrix that specifies the clauses. Each row is a clause. Each row must contain the same number of columns as qubits, the bit length of the clauses. 0 and 1 are values which are added to clause, ignored bits should be set to any other value.
colorCode	Boolean which specifies if data should be returned as list of colors rather than numerical data

Value

Array of numbers or string color names

Examples

```
checkCases( rbind(c(1,-1),c(1,1) ) )
```

---

cntrld *cntrld*

Description

Creates a matrix representing a controlled gate on a system of qubits. The target and control qubits can be located at arbitrarying positions.

Usage

```
cntrld(gate,n,...)
```

Arguments

gate	single qubit gate to create controlled version of
n	Number of qubits in the ket, including target, control, and all other qubits
...	List of qubits. The last qubit in the list is the target. Any others listed before it are control qubits. Can be any number between 1 and n-1 control qubits, where n is the number of qubits in the ket. Qubits are indexed from 0, starting at the most significant qubit

Value

A matrix representing the operation of a controlled qubit gate on any subset of the input ket

Examples

```
cntrld(X(),2,0,1)
cntrld(X(),2,1,0)
cntrld(Y(),4,2,3)
cntrld(X(),8,0,5)
```

CoherentNoise *CoherentNoise*

Description

Implements a model of coherent noise as used in <DOI:10.1038/s41534-018-0106-y>. It rotates each qubit around the z-axis by the specified amount. If randomRoation is set, it will rotate around the X, Y, or Z axis, which is chosen at random. Randomizing provides interesting side effects but is less representative of quantum noise.

Usage

```
CoherentNoise(p,theta,randomRotation=FALSE)
```

Arguments

p	Input quantum state in density matrix format
theta	Angle of rotation to apply
randomRotation	Boolean specifying whether the rotation should be in a semi-random direction

Value

The quantum state in density matrix format after noise has been applied

Examples

```
CoherentNoise( p= convert_ket2DM(ket(1,1,1,1)),theta=0.06*pi)
```

colv *colv*

Description

Returns a column vector

Usage

```
colv(...)
```

Arguments

...	Variable number of numbers representing the values in the column vector

Value

Column vector containing input arguments

Examples

```
colv(1,0,1,2)
```

compareQuantumState *compareQuantumState*

Description

Generates a matrix (quantum oracle) which will flip the last qubit in a quantum state if the qubits at indices in vectors a and b are the same

Usage

```
compareQuantumState(nQubits,a,b)
```

Arguments

nQubits	Number of qubits in a target ket. Should contain at least enough for states a and b and an additional last qubit.
a	Vector of indices of first state to compare in a target ket
b	Vector of indices of second state to compare in a target ket

Value

Matrix of the compareQuantumState oracle

Examples

```
compareQuantumState(5,0:1,2:3)
```

`controlled` *controlled*

Description

Creates a matrix representing a controlled gate on a system of qubits. The target and control qubits can be located at arbitrary positions.

Usage

```
controlled(gate,n,cQubits,tQubit)
```

Arguments

`gate`	single qubit gate to create controlled version of
`n`	Number of qubits in the ket, including target, control, and all other qubits
`cQubits`	Vector of qubit indices. There can be between 1 and n-1 control qubits, where n is the number of qubits in the ket. Qubits are indexed from 0, starting at the most significant qubit
`tQubit`	Index of the target qubit. Qubits are indexed from 0, starting at the most significant qubit

Value

A matrix representing the operation of a controlled qubit gate on any subset of the input ket

Examples

```
controlled(X(),n=2,cQubits=0,tQubit=1)
controlled(X(),n=4,cQubits=c(0,1,2),tQubit=3)
```

convert_bin2dec *convert_bin2dec*

Description

Takes a vector of unsigned bits with MSB first and produces integer value

Usage

```
convert_bin2dec(b)
```

Arguments

b	Vector of bits with most significant bits first

Value

Integer value of bits

Examples

```
convert_bin2dec( c(1,0,0) )
```

convert_dec2bin *convert_dec2bin*

Description

Takes an integer and returns an unsigned vector bits representing the same value

Usage

```
convert_dec2bin(x,len=32)
```

Arguments

x	Integer
len	Number of bits to represent integer with. Will crop most significant bits if insufficient length.

Value

Vector of bits with MSB first

Examples

```
convert_dec2bin(10)
convert_dec2bin(10,8)
```

convert_ket2DM *convert_ket2DM*

Description

Converts a ket (pure) description of quantum state and creates a density matrix representation of the same state. Density matrices can represent both pure and mixed states.

Usage

```
convert_ket2DM(v)
```

Arguments

v	An input ket

Value

Density matrix representing same state as input ket

Examples

```
convert_ket2DM( ket(1,0) )
```

CX *CX*

Description

If no argument is supplied, returns the matrix of Controlled-X gate. If ket given as input, applies a Controlled-X gate to the input ket and returns the resulting ket

Usage

```
CX(...)
```

Arguments

...	No argument, or 4 dimensional (2 qubit) ket (column vector) that is input to the gate

Value

Matix of the Controlled-X gate or ket after a Controlled-X gate is applied

Examples

```
CX(ket(1,1,1,1))
CX()
```

CY *CY*

Description

If no argument is supplied, returns the matrix of Controlled-Y gate. If ket given as input, applies a Controlled-Y gate to the input ket and returns the resulting ket

Usage

```
CY(...)
```

Arguments

...	No argument, or 4 dimensional (2 qubit) ket (column vector) that is input to the gate

Value

Matix of the Controlled-Y gate or ket after a Controlled-Y gate is applied

Examples

```
CY(ket(1,1,1,1))
CY()
```

CZ *CZ*

Description

If no argument is supplied, returns the matrix of Controlled-Z gate. If ket given as input, applies a Controlled-Z gate to the input ket and returns the resulting ket

Usage

```
CZ(...)
```

Arguments

...	No argument, or 4 dimensional (2 qubit) ket (column vector) that is input to the gate

Value

Matix of the Controlled-Z gate or ket after a Controlled-Z gate is applied

Examples

```
CZ(ket(1,1,1,1))
CZ()
```

DecomposeGate *DecomposeGate*

Description

Uses the gridsynth algorithm Sellinger 2012 <arXiv:1212.6253>, which is available at https://www.mathstat.dal.ca/~selinger/ to decompose arbitrary gates to the Clifford+T set. For decomposition of controlled 2-qubit gates, circuits from Amy 2013 <DOI:10.1109/TCAD.2013.2244643> are also used.

Usage

```
DecomposeGate(path,g,TwoQubit=FALSE,n=1,tQubit=0,cQubit=1,prec=10)
```

Arguments

path	String of path to folder containing gridsynth binary (not including gridsynths file name). R must have permission to read and write from this folder, and to execute the binary.
g	If a single number, this is the Z-rotation angle to approximate. If a vector of length 3, it is the alpha, beta, and gamma parameters as defined in Schuld 2018 <arXiv:1804.00633>.
TwoQubit	Boolean specifying whether this is a single or controlled 2-qubit gate
n	The total number of qubits in the system. If TwoQubit is TRUE, the returned circuit will have n+1 qubits due to the requirement of an ancilla qubit.
tQubit	The target qubit. If a single qubit gate, the gate is applied to this qubit. If a 2-qubit gate, this is the target qubit.
cQubit	Control qubit if a 2-qubit gate. Value does not matter for single qubit gate.
prec	The binary precision of the approximation, which is passed to the gridsynth binary.

Value

List of cycles which approximates the input gate.

Examples

```
## Not run:
 DecomposeGate(path="./",g=pi/5,TwoQubit=TRUE,n=3,tQubit=0,cQubit=1,prec=3)

## End(Not run)
```

dirac *dirac*

Description

Prints the dirac notation of the input ket

Usage

```
dirac(ket)
```

Arguments

ket	Ket (column vector) to print dirac notation of

Value

String of dirac notation

Examples

```
dirac(ket(1,0,1,0))
```

dist *dist*

Description

Reports the distance between two vectors/kets

Usage

```
dist(a,b)
```

Arguments

a	column vector
b	column vector

Value

Distance between two vectors

Examples

```
dist(ket(1,1,1,1),ket(1,0,0,1))
```

dotmod2 *dotmod2*

Description

Takes two integers and takes the dot product of their binary representations. Output is the value of the dot product, modulus 2

Usage

```
dotmod2(x,a)
```

Arguments

x	integer
a	integer

Value

Binary value resulting from the bit-wise dot product modulus 2

Examples

```
dotmod2(5,5)
dotmod2(1,2)
dotmod2(0,1)
```

exponentialMod *exponentialMod*

Description

Creates a function that raises a number to a power modulus another number. Is a fix for information loss due to extremely large numbers. It takes the modulus for every multiplication

Usage

```
exponentialMod(a,N)
```

Arguments

a	random number that is used as input to Shor's algorithm
N	Number that Shor's algorithm is to factor

Value

A function that takes argument x and returns a^x modulus N

Examples

```
exponentialMod(8,21)
exponentialMod(2,15)
```

`extractMNIST` *extractMNIST*

Description

Opens the MNIST training data and label files (not provided with package) and extracts the images and labels and returns them in a list

Usage

```
extractMNIST(data,labels,s,centercrop=TRUE)
```

Arguments

`data`	String of path to file containing MNIST training images
`labels`	String of path to file containing MNIST training labels
`s`	Number of samples and labels to extract from file
`centercrop`	Boolean indicating whether the images should be centercropped to contain only 256 points

Value

List containing matrix of image data and array of training labels

Examples

```
## Not run:
extractMNIST("train-images.idx3-ubyte","train-labels.idx1-ubyte",2)

## End(Not run)
```

FullAdder *FullAdder*

Description

Provides the quantum operatios for a full-adder with the specified input and output indices. Uses the circuit developed by Cheng and Tseng <DOI:10.1049/el:20020949>. Uses CNOT and TOFFOLI gates, with the TOFFOLI gates being broken down into H, T, and CNOT gates. The SUM (qu)bit gets places where the b operand (qu)bit is.

Usage

```
FullAdder(n=4,cin=0,a=1,b=2,cout=3)
```

Arguments

n	Number of qubits in input quantum state
cin	index of the carry in (qu)bit
a	Index of the first operand (qu)bit
b	Index of the second operand (qu)bit
cout	Index where the output carry (qu)bit will be placed

Value

A list with elements containing the quantum operations (matrices) for the full adder in each cycle.

Examples

```
FullAdder(n=4,cin=0,a=1,b=2,cout=3)
```

G *G*

Description

Creates quantum gate defined by 4 angles as demonstrated by Barenco (1995). If no argument is supplied, returns the matrix of G gate. If ket given as input, applies an G gate to the input ket and returns the resulting ket

Usage

```
G(a,b,g,p=0,...)
```

Arguments

a	First angle
b	second angle
g	third angle
p	global phase
...	No argument, or ket (column vector) that is input to the gate

Value

Matrix of the G gate or ket after an G gate is applied

Examples

```
G(0,0,0,0,ket(1,0))
G(1,1,1)
```

---

gcd *gcd*

Description

Finds the gcd

Usage

```
gcd(x,y)
```

Arguments

x	First argument
y	Second argument

Value

The greated common divisor of x and y

Examples

```
gcd(7,3)
gcd(10,4)
```

GroverDiffusion *GroverDiffusion*

Description

If integer is input, returns the matrix of Grover Diffusion operation on the integer number of qubits. If ket given as input, applies a Grover Diffusion operation to the input ket and returns the resulting ket

Usage

```
GroverDiffusion(input)
```

Arguments

input	Either integer specifying size of operation (in number of qubits it is applied to) or input ket to apply Grover Diffusion to

Value

Either the matrix of the Grover Diffusion gate of the specified size or ket after a Grover Diffusion operation is applied

Examples

```
GroverDiffusion(ket(1,1,1,1,1,1,1,1))
GroverDiffusion(3)
```

GroverOracle *GroverOracle*

Description

If integer is input, returns the matrix of GroverOracle operation on the integer number of qubits. If ket given as input, applies a GroverOracle operation to the input ket and returns the resulting ket

Usage

```
GroverOracle(w,input)
```

Arguments

w	Integer specifying the state to search for, between 0 and 2^n-1 where n is the number of qubits
input	Either integer specifying size of operation (in number of qubits it is applied to) or input ket to apply GroverOracle to

Value

Either the matrix of the GroverOracle gate of the specified size or ket after a GroverOracle operation is applied

Examples

```
GroverOracle(0,ket(1,1,1,1,1,1,1,1))
GroverOracle(0,3)
```

`GroversAlgorithm` *GroversAlgorithm*

Description

Applies Grover's search algorithm to a uniform ket to simulate a quantum search

Usage

```
GroversAlgorithm(n,w,iterations=n,printOutput=FALSE,plotOutput=FALSE,tag="")
```

Arguments

`n`	Number of qubits in the problem, not counting the extra ancillary qubit
`w`	Integer specifying the state to search for, between 0 and 2^n-1 where n is the number of qubits
`iterations`	Number of iterations to apply the oracle and diffusion, optimal is approximately n
`printOutput`	Boolean specifying if the measurement probabilities should be printed as search progresses
`plotOutput`	Boolean specifying if the output probabilities should be plotted to a graph
`tag`	String which is attached to output file name if plotOutput is TRUE

Value

Ket after a Grover search has been applied to it

Examples

```
GroversAlgorithm(7,0,14)
GroversAlgorithm(7,0,14,printOutput=TRUE)
```

H *H*

Description

If no argument is supplied, returns the matrix of H gate. If ket given as input, applies an H gate to the input ket and returns the resulting ket

Usage

```
H(...)
```

Arguments

...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the H gate or ket after a Hgate is applied

Examples

```
H(ket(1,0))
H()
```

hermitian *hermitian*

Description

Determines whether an operation (matrix) is hermitian by comparing it to its adjoint

Usage

```
hermitian(m)
```

Arguments

m	gate operation (gate) that is to be checked

Value

boolean indicating whether matrix is hermitian or not

Examples

```
hermitian(matrix(c(0,1,1,0),nrow=2))
```

I *I*

Description

If no argument is supplied, returns the matrix of I gate. If ket given as input, applies an I gate to the input ket and returns the resulting ket

Usage

```
I(...)
```

Arguments

...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the I gate or ket after an I gate is applied

Examples

```
I(ket(1,0))
I()
```

inner *inner*

Description

Finds the inner product of two kets, <w|v>. w and v can be the same

Usage

```
inner(w,v)
```

Arguments

w	ket (column vector) that is the left side of the innter product, converted to a bra before the dot product
v	ket (column vector) that is the right side of the inner product

Value

Value of the inner product

Examples

```
inner(ket(1,0),ket(1,1))
```

intket *intket*

Description

Returns a ket (column vector) that has the encoded value of the specified integers. Implements what is commonly known as basis encoding. Does not simulate the state creation.

Usage

```
intket(x,n,amplitudes=rep(1,length(x)))
```

Arguments

x	Integer, or vector of integers, specifying the integer encoded state(s) of the ket
n	Integer specifying the number of qubits in the ket
amplitudes	Integer, or vector of integers, specifying the amplitudes for corresponding basis in x. Must be same length as x. Only relative values matter as the ket will be normalized. Default is for all states to have same amplitude.

Value

Column vector containing normalized amplitudes of a ket

Examples

```
intket(0,1)
intket(3,2)
intket(4,3)
intket( c(0,1), 4)
intket( c(0,2), 4 , c(1,2) )
```

ket *ket*

Description

Returns a normalized ket (column vector)

Usage

```
ket(...)
```

Arguments

...	Variable number of numbers representing the amplitudes of the ket

Value

Column vector containing normalized amplitudes of a ket

Examples

```
ket(1,0,1,2)
```

`many` *many*

Description

Takes as input a gate and generates the matrix for that gate being applied to multiple qubits by creating a tensor product of the matrix. If a ket is supplied, the matrix will be applied to the ket

Usage

```
many(gate,n,...)
```

Arguments

`gate`	Single qubit gate to apply
`n`	Number of qubits that the gate will be applied to
`...`	Either no argument or a ket that the gates will be applied to

Value

The matrix representing the application of many gates or a ket after the gates have been applied

Examples

```
many(H(),4)
many(X(),2,ket(1,0,0,0))
```

measure *measure*

Description

Probabilistically measures the input ket. By default measures all qubits, but if a list of integers is supplied it will measure only those qubits. Returns a list containing the state of the ket after measurement along with integer value of the state that was measured. Additionally, returns a vector of the measured binary values, if a list of qubits to measure was specified.

Usage

```
measure(...,l2r=FALSE)
```

Arguments

`...`	The input ket to measure. Optionally followed by integers specifying which qubits of the ket to measure. Qubits indexed from 0 from right to left
`l2r`	Boolean which specifices if indexing should be performed from left to right. Is FALSE by default to maintain backwards compatibility, however all other functions index from left to right.

Value

A list with the first item a column vector containing normalized amplitudes of the measured ket and the second item the integer value of the state which was measured. If a list of qubits to measure was specified as an argument, there is a 3rd item in the list which is a vector of the binary measured values.

Examples

```
measure(ket(1,0),l2r=TRUE)
measure(ket(1,2,2,1),0,l2r=TRUE)
measure(ket(1,2,3,4,5,6,7,8),0,l2r=TRUE)
measure(ket(1,2,3,4,5,6,7,8),0,1,l2r=TRUE)
measure(ket(1,2,3,4,5,6,7,8),0,1,2,l2r=TRUE)
```

mm *mm*

Description

Returns a matrix containing the specified elements. Values are input column-wise. Used for convenient shorthand creation of matrices

Usage

```
mm(...)
```

Arguments

...	Variable number of numbers representing the values in the matrix

Value

Matrix containing the values of the inputs

Examples

```
mm(1,0,1,2)
```

---

nBitAddition *nBitAddition*

Description

Strings together output from FullAdder function to create multi-(qu)bit addition. It assumes the input operands are laid out as in <DOI:10.1049/el:20020949>. From left to right (top to bottom) the order is C0, a1, b1, C1, a2, b2, C2, ... bn-1, Cn. There must be 3n+1 qubits in order to perform n-(qu)bit addition.

Usage

```
nBitAddition(n)
```

Arguments

n	Length of input (qu)bit strings.

Value

A list containing the quantum circuit (each elemented is one cycle of the circuit) performing n-bit addition.

Examples

```
## Not run:
 nBitAddition(2)

## End(Not run)
```

norm *norm*

Description

Finds the norm of input column vector by taking the inner product with itself

Usage

```
norm(v)
```

Arguments

v	kcolumn vector

Value

Norm of the input column vector

Examples

```
norm(ket(1,0))
```

opDM *opDM*

Description

Applies a quantum operation to a density matrix

Usage

```
opDM(V,G)
```

Arguments

V	Input density matrix
G	Quantum operation to apply to density matrix

Value

A density matrix which has been modofied by the input quantum operation

Examples

```
opDM( V=convert_ket2DM(ket(1,0)) , G=X()  )
```

PauliNoise *PauliNoise*

Description

Applies stochastic Pauli noise to an input quantum state. If only e is set, it is equally distributed to X, Y, and Z error which is an isotropic Pauli noise model. Otherwise, levels can be set seperately for each.

Usage

```
PauliNoise(p,e=ex+ey+ez,ex=e/3,ey=e/3,ez=e/3)
```

Arguments

p	Input quantum state, in density matrix format
e	Total amount of noise to apply the state, is the sum of ex, ey, and ez
ex	Amount of X noise to apply to the state
ey	Amount of Y noise to apply to the state
ez	Amount of Z noise to apply to the state

Value

The quantum state in density matrix format, after Pauli noise has been applied to it

Examples

```
PauliNoise( p=convert_ket2DM(ket(1,0)) , e=0.01  )
```

PauliOperators *PauliOperators*

Description

Generates random Pauli operators (tensor products of random I,X,Y,or Z gates applied to each qubit) that can be applied to register of n qubits. Used with Randomized Compiling, where random Pauli gates are applied to each qubit.

Usage

```
PauliOperators(n,m=4^n,unique=TRUE)
```

Arguments

n	Size of the Pauli operators to generate, should be equal to the number of target qubits
m	Number of different Pauli operators to generate
unique	Boolean indicating if each Pauli operator generated should be unqiue. Must be false is m > 4^n

Value

A list of m Pauli operators of size n

Examples

```
PauliOperators( n=2,m=2,unique=FALSE)
```

PhaseDamping *PhaseDamping*

Description

Implements Phase Damping noise model on the input quantum state. Formulas taken from <DOI:10.1103/PhysRevA.90.0623

Usage

```
PhaseDamping(p,Ppd)
```

Arguments

p	Input quantum state, in density matrix format
Ppd	The probability of phase Damping.

Value

The quantum state, after Phase Damping has been applied.

Examples

```
PhaseDamping(p=convert_ket2DM(ket(1,0)),Ppd=0.01)
```

plotprobs *plotprobs*

Description

Plots the probabilities of each of the amplitudes of ket in a barplot

Usage

```
plotprobs(v,color=rep("Blue",length(v)),customLegend=FALSE,lgNm="",lgCl="")
```

Arguments

v	ket that is to be plotted
color	Text, possibly an array, specifying the colors of the bars
customLegend	Boolean specifying if a custom legend should be inserted
lgNm	Vector of legend names
lgCl	Vector of legend colors

Value

A plot

Examples

```
plotprobs(ket(1,0,1,0),color=c("Red","Blue","Red","Blue"))
```

pp *pp*

Description

Prints a pasted string containing all arguments. Short hand for print(paste(...))

Usage

```
pp(...)
```

Arguments

...	Variable number of inputs to be printed

Value

Prints string

Examples

```
pp("Value is",1,0,1,2)
```

---

probs *probs*

Description

Returns a column vector containing the probabilities of measuring the system in each state

Usage

```
probs(ket)
```

Arguments

ket	ket (column vector) that is input to the gate

Value

Column vector containing probabilities

Examples

```
probs(ket(1,1))
```

QAOA *QAOA*

Description

Implements a clause-based version of Quantum Approximation Optimization Algorithm (Farhi, Goldstone, and Gutmann 2014) <arXiv:1411.4028>. Takes as input a set of clauses and performs Controlled-Phase and Rx gates to perform optimization. See "An Introduction to Quantum Optimization Approximation Algorithm" (Wang and Abdullah 2018) for explanation.

Usage

```
QAOA(clauses,p=1,gamma=pi/p,beta=pi/(2*p),displayProgress=FALSE,byCycle=FALSE)
```

Arguments

clauses	Matrix that specifies the clauses. Each row is a clause. Each row must contain the same number of columns as qubits, the bit length of the clauses. 0 and 1 are values which are added to clause, ignored bits should be set to any other value.
p	Number of iterations that algorithm will run. Each iteration applies U(C,g) and U(B,b)
gamma	Angle for U(C,g), currently the same for all iterations. Should be between 0 and 2*pi
beta	Angle for U(B,b), currently the same for all iterations. Should be between 0 and pi
displayProgress	Boolean which specifies if progress should be shown. If TRUE, a bar plot is continually updated showing the amplitudes
byCycle	Boolean which specifies if function should return the circuit. If TRUE, rather than performing the algorithm it will generate and return the equivalent circuit.

Value

Ket after algorithm is applied

Examples

```
QAOA(rbind(c(0,0),c(0,1)))
```

QAOA_example *QAOA_example*

Description

Runs an example of QAOA

Usage

```
QAOA_example(case=1)
```

Arguments

case	Integer specifying case to demonstrate. Currently only two, 1 (small) and 2 (medium)

Value

No value

Examples

```
 ## Not run:
 QAOA_example(1)
 QAOA_example(2)

## End(Not run)
```

QAOA_maxcut *QAOA_maxcut*

Description

Takes a connection matrix as input and converts it to a set of clauses, then runs the Quantum Approximation Optimization Algorithm (Farhi, Goldstone, and Gutmann 2014) <arXiv:1411.4028>.

Usage

```
QAOA_maxcut(connectionMatrix,p=1,gamma=pi/p,beta=pi/(2*p),displayProgress=FALSE)
```

Arguments

connectionMatrix	Matrix that specifies the edges between nodes. Rows are source nodes and columns are destination nodes. Value of 0 means no edge, value of 1 means edge. If edge is undirected, an edge should be specified going both directions.
p	Number of iterations that algorithm will run. Each iteration applies U(C,g) and U(B,b)
gamma	Angle for U(C,g), currently the same for all iterations. Should be between 0 and 2*pi
beta	Angle for U(B,b), currently the same for all iterations. Should be between 0 and pi
displayProgress	Boolean which specifies if progress should be shown. If TRUE, a bar plot is continually updated showing the amplitudes

Value

Ket after algorithm is applied

Examples

```
QAOA_maxcut(randomConnectionMatrix(4,2),p=5)
```

QFT *QFT*

Description

If integer is input, returns the matrix of QFT operation on the integer number of qubits. If ket given as input, applies a QFT operation to the input ket and returns the resulting ket. If byCycle is TRUE, it generated the circuit for the QFT and returns a list of the cycles.

Usage

```
QFT(input,byCycle=FALSE,swaps=TRUE,CliffordT=FALSE,prec=10,path="./")
```

Arguments

input	Either integer specifying size of operation (in number of qubits it is applied to) or input ket to apply QFT to
byCycle	Boolean which specifies whether the circuit should be generated or not. If TRUE, rather than returning the matrix or performing the algorithm, the function will generate and return the equivalent circuit.
swaps	Boolean which specifies if the the SWAP gates required at the end of the QFT should be inserted. May not be necessary if qubit reordering is acceptable. Only valid if byCycle is TRUE.
CliffordT	Boolean which specifies if the generated circuit should be decomposed into the Clifford+T set. Only valid if byCycle is TRUE.
prec	The precision of the decomposition into the Clifford+T set. Only valid if byCycle and CliffordT are both TRUE.
path	Path from current working directory to the gridsynth binary. Only used if CliffordT is set to TRUE. The gridsynth binary is not contained in QuantumOps but available from https://www.mathstat.dal.ca/~selinger/newsynth/

Value

If the input is an integer, the matrix of the QFT gate of the specified size. If the input is a ket, the ket after a QFT operation is applied. If byCycle is TRUE, a list of the cycles of the algorithm.

Examples

```
QFT(ket(1,0))
QFT(ket(1,0,0,1))
QFT(3)
```

QuantumClassifier *QuantumClassifier*

Description

Quantum classifier which was proposed by Maria Schuld (2018). Consists of code blocks which have parallel single qubit quantum gates followed by controlled qubit gates. Takes as input samples and a corresponding list of labels indicating the correct output value of each sample. Will update the parameters of the gates in order to correctly identify the samples.

Usage

```
QuantumClassifier( n=8,B=2,r=c(1,3),
   data=NULL,labels=NULL,digit=0,
   eta=1,decay=1,bsc=1,t=20,tag="",pl=TRUE,train=TRUE,
   validT=FALSE,vdata=NULL,vlabels=NULL,
   pretrained=FALSE,alpha=NULL,beta=NULL,gamma=NULL,bias=NULL,
          writeParameters=FALSE,outputPath=NULL )
```

Arguments

n	Number of qubits that this the classifier will use.
B	Number of blocks in the circuit. A block consists of n single qubit gates applied to each of the qubits in parallel and then n/(gcd,r) controlled qubit gates to perform a maximal entanglement. r is the specified range of the controlled gates
r	Vector containing the range for each block. length(r) should be equal to B. r of 1 means controlled gates are performed on adjacent qubits
data	matrix containing input training data. Rows are individual samples. The number of columns should be equal to 2^n
labels	Vector containing labels of digits. Length must be the same as the number of rows in data
digit	Individual output to identify. The network will attempt to differentiate between inputs that are labelled as digit (in the labels vector) and inputs that are labelled as any other number
eta	learning rate for parameter updates
decay	Multiplier for learning rate after each training iteration. If set to less than 1, the learning rate decays in time
bsc	Scaler for the learning rate of the bias. Setting to a low value will result in other parameters updating faster than the bias
t	Number of training iterations to perform. Total runs is equal to this value multiplied by the number of samples provided
tag	String to attach to name of output files
pl	Boolean indicating whether training output should be plotted

train	Boolean specifying if network should trained on training data, only false if passing in pretrained model
validT	Boolean specifying if the network should be tested on validation data while being trained
vdata	Validation data, necessary if validT is set to true in which case network is tested on this data while being trained. Can be set to same as data.
vlabels	Validation labels, necessary if validT is set to true in which case network is tested on vdata while being trained. Can be set to same as labels.
pretrained	Boolean specifying if a pretrained model is being passed in. If so, alpha, beta, and gamme will be set to intputs, rather than randomized.
alpha	alpha values for gates if pretrained is set to TRUE, should be a vector of length equal to the number of gates in the circuit.
beta	beta values for gates if pretrained is set to TRUE, should be a vector of length equal to the number of gates in the circuit.
gamma	gamma values for gates if pretrained is set to TRUE, should be a vector of length equal to the number of gates in the circuit.
bias	Bias applied to the output of the circuit.
writeParameters	Boolean specifying whether function should write the parameters as it trains. Useful when training takes a long time.
outputPath	String which specifies path to write output parameters to if writeParameters is TRUE. Must have write priveleges in this directory. The function will create two directories inside outputPath, named 0 and 1, and will alternate output to each folder. This prevents corruption of output if interrupted.

Value

List containing a list of the 33 gates and the matrix representing the entire circuit of the trained classifier

Examples

```
## Not run:
QuantumClassifier(n=8,B=2,r=c(1,3),
   matrix(sample(256,replace=TRUE),nrow=1),
array(1),0,1,1,.001,1,"test")

## End(Not run)
```

QuantumMNIST256Classifier

QuantumMNIST256Classifier

Description

Quantum classifier which was proposed by Maria Schuld (2018). Consists of 33 quantum gates with a depth of 19. Takes as input samples with dimensions of 256 and a corresponding list of labels indicating the correct output value of each sample. Will update the parameters of the gates in order to correctly identify one of the digits specified.

Usage

```
QuantumMNIST256Classifier(
 data=NULL,labels=NULL,digit=0,
 eta=1,decay=1,bsc=1,t=20,tag="",pl=TRUE,train=TRUE,
 validT=FALSE,vdata=NULL,vlabels=NULL,
 pretrained=FALSE,alpha=NULL,beta=NULL,gamma=NULL)
```

Arguments

data	matrix containing input training data. Rows are individual samples. There must be 256 columns
labels	Vector containing labels of digits. Length must be the same as the number of rows in data
digit	Individual digit (0-9) to identify
eta	learning rate for parameter updates
decay	Multiplier for learning rate after each training iteration. If set to less than 1, the learning rate decays in time
bsc	Scaler for the learning rate of the bias. Setting to a low value will result in other parameters updating faster than the bias
t	Number of training iterations to perform. Total runs is equal to this value multiplied by the number of samples provided
tag	String to attach to name of output files
pl	Boolean indicating whether training output should be plotted
train	Boolean specifying if network should trained on training data, only false if passing in pretrained model
validT	Boolean specifying if the network should be tested on validation data while being trained
vdata	Validation data, necessary if validT is set to true in which case network is tested on this data while being trained. Can be set to same as data.
vlabels	Validation labels, necessary if validT is set to true in which case network is tested on vdata while being trained. Can be set to same as labels.

pretrained	Boolean specifying if a pretrained model is being passed in. If so, alpha, beta, and gamme will be set to intputs, rather than randomized.
alpha	alpha values for gates if pretrained is set to TRUE, should be a vector of length 33
beta	beta values for gates if pretrained is set to TRUE, should be a vector of length 33
gamma	gamma values for gates if pretrained is set to TRUE, should be a vector of length 33

Value

List containing a list of the 33 gates and the matrix representing the entire circuit of the trained classifier

Examples

```
 ## Not run:
QuantumMNIST256Classifier(matrix(sample(256,replace=TRUE),nrow=1),array(1),0,1,1,.001,1,"test")

## End(Not run)
```

R *R*

Description

If no second argument is supplied, returns the matrix of an R phase gate of the specified radians. If ket given as second argument, applies the R gate to the input ket and returns the resulting ket. Is equivalent to the more recently added Rz function.

Usage

```
R(theta,...)
```

Arguments

theta	Radians to phase rotate the ket
...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the R gate or ket after an R gate is applied

Examples

```
R(pi,ket(1,0))
R(pi)
```

randomConnectionMatrix *randomConnectionMatrix*

Description

Generates a connection matrix for a random undirected graph. Intended for input to QAOA_maxcut.

Usage

```
randomConnectionMatrix(nNodes,nEdges)
```

Arguments

nNodes	Number of nodes in generated graph
nEdges	Number of undirected edges in generated graph

Value

Connection Matrix specifying the edges of an undirected graph. Rows are source nodes, columns are destination nodes.

Examples

```
randomConnectionMatrix(5,3)
```

RandomizeCompile *RandomizeCompile*

Description

Implements Randomized Compiling as described by Wallman and Emerson <DOI:10.1103/PhysRevA.94.052325>. Takes as input a list of easy cycles and a list of hard cycles. In this context, a cycle is the application of one operation to a register of qubits. Inserts randomizing Pauli gates after easy cycles, and corrective operations before the next easy cycle. The randomizations are then combined with the easy cycles. The first and last cycles are easy, with all other cycles alternating betweene asy and hard. Hence, the number of easy cycles should be one more than the number of hard cycles. Easy cycles (C) can be left unset, in which case Idle cycles will be inserted to enabled the randomizations.

Usage

```
RandomizeCompile( C=rep( list(
    repeatTensor(I(),log( dim(G[[1]])[1],base=2))),
      length(G)+1)
    ,G,combine=TRUE)
```

Arguments

C	List of easy cycles
G	List of hard cycles
combine	Boolean specifying if the output should be combined into one list or left seperate

Value

If combine is TRUE, a list of cycles that are now Randomly Compiled. If combine is FALSE, a list of two lists, the first being the Randomly Compiled easy cycles and the second the hard cycles.

Examples

```
RandomizeCompile( G=list( CX(), CX()))
RandomizeCompile( G=list( controlled(gate=Z(),n=3,cQubits=0,tQubit=1) ,
  single(gate=H(),n=3,t=1) ))
```

---

ranket *ranket*

Description

Generates a random ket by selecting random polar coordinates (theta,phi) for each. Approach taken from <DOI:10.1103/PhysRevA.95.062338>.

Usage

```
ranket(n)
```

Arguments

n	Number of qubits in generated get

Value

A ket with a randomized state

Examples

```
ranket(4)
```

reduceMeasure *reduceMeasure*

Description

Probabilistically measures the input ket and reduces the size of ket by removing the measured qubits. By default measures all qubits, but if a list of integers is supplied it will measure only those qubits. Returns a list containing the state of the ket after measurement along with integer value of the state that was measured. Additionally, returns a vector of the measured binary value if a list of qubits to measure was specified.

Usage

```
reduceMeasure(...,l2r=FALSE)
```

Arguments

...	The input ket to measure. Optionally followed by integers specifying which qubits of the ket to measure. Qubits indexed from 0 from right to left
l2r	Boolean which specifices if indexing should be performed from left to right. Is FALSE by default to maintain backwards compatibility, however all other functions index from left to right.

Value

A list with the first item a column vector containing normalized amplitudes of the measured ket and the second item the integer value of the state which was measured. If a list of qubits to measure was specified as an argument, there is a 3rd item in the list which is a vector of the binary measured

Examples

```
reduceMeasure(ket(1,0),l2r=TRUE)
reduceMeasure(ket(1,2,2,1),0,l2r=TRUE)
reduceMeasure(ket(1,2,3,4,5,6,7,8),0,l2r=TRUE)
reduceMeasure(ket(1,2,3,4,5,6,7,8),0,1,l2r=TRUE)
reduceMeasure(ket(1,2,3,4,5,6,7,8),0,1,2,l2r=TRUE)
```

repeatTensor *repeatTensor*

Description

Repeatedly tensors the input with itself

Usage

```
repeatTensor(g,n)
```

Arguments

g	Object, typically a gate, that is to be tensored with itself
n	Number of times to tensor g with itself

Value

The input g tensored by itself n times

Examples

```
repeatTensor(X(),2)
repeatTensor(X(),2)
```

Rx *Rx*

Description

If no second argument is supplied, returns the matrix of an Rx rotation gate of the specified radians. If ket given as second argument, applies the Rx gate to the input ket and returns the resulting ket.

Usage

```
Rx(theta,...)
```

Arguments

theta	Radians to phase rotate the ket around the x-axis
...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the Rx gate or ket after an Rz gate is applied

Examples

```
Rx(pi,ket(1,0))
Rx(pi)
```

Ry *Ry*

Description

If no second argument is supplied, returns the matrix of an Ry rotation gate of the specified radians. If ket given as second argument, applies the Ry gate to the input ket and returns the resulting ket.

Usage

```
Ry(theta,...)
```

Arguments

theta	Radians to phase rotate the ket around the y-axis
...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the Rz gate or ket after an Rz gate is applied

Examples

```
Ry(pi,ket(1,0))
Ry(pi)
```

Rz *Rz*

Description

If no second argument is supplied, returns the matrix of an Rz rotation gate of the specified radians. If ket given as second argument, applies the Rz gate to the input ket and returns the resulting ket.

Usage

```
Rz(theta,...)
```

Arguments

theta	Radians to phase rotate the ket around the z-axis
...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the Rz gate or ket after an Rz gate is applied

Examples

```
Rz(pi,ket(1,0))
Rz(pi)
```

S *S*

Description

If no argument is supplied, returns the matrix of S gate. If ket given as input, applies an S gate to the input ket and returns the resulting ket

Usage

```
S(...)
```

Arguments

...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the S gate or ket after an S gate is applied

Examples

```
S(ket(1,1))
S()
```

Shor *Shor*

Description

Implements Shor's algorithm by applying the quantum oracle, performing a QFT, measuring the output, and using continued fractions algorithm to find period. Period is then used with Euclidean algorithm to check if factors are legitimate prime factors. Is probabilistic and may fail. Factors 15 with ease and 21 occassionally.

Usage

```
Shor(N,trials=150,random=FALSE)
```

Arguments

N	Number that Shor's algorithm is to factor
trials	Number of times to attempt before giving up
random	Boolean which determines whether seed is random or not

Value

Vector containing prime factors

Examples

```
Shor(15,trials=2)
```

---

single *single*

Description

Takes as input a gate and generates the matrix for that gate being applied to a single qubit in a ket by creating a tensor product of the matrix with Identity matrices. If a ket is supplied, the matrix will be applied to the ket

Usage

```
single(gate,n,t,...)
```

Arguments

gate	Single qubit gate to apply
n	Number of qubits that are in the target ket
t	Target qubit that the gate will be applied to, other qubits are unmodified. Indexed from 0.
...	Either no argument or a ket that the gate will be applied to

Value

The matrix representing the application of a single gate to one of the qubits in a ket or a ket after the gate has been applied

Examples

```
single(H(),4,1)
single(H(),2,1,ket(1,0,0,0))
single(X(),2,0,ket(1,0,0,0))
```

singleSWAP *singleSWAP*

Description

Implements the SWAP gate between two qubits, which can be in a larger ket. If no argument is supplied, returns the matrix of the gate. If ket given as input, applies the gate to the input ket and returns the resulting ket. In its default configuration it is the same as standard SWAP.

Usage

```
singleSWAP(nQubits=2,a=0,b=1,...)
```

Arguments

nQubits	Number of qubits in target ket
a	Index of first qubit to swap, indexed from 0
b	Index of second qubit to swap, indexed from 0
...	No argument, or ket (column vector) that is input to the gate

Value

Matrix of the singleSWAP gate or ket after an singleSWAP gate is applied

Examples

```
singleSWAP(2,0,1, ket(1,2,3,4) )
singleSWAP(4,0,3, intket( c(1,5),4,c(1,2)) )
```

Steane *Steane*

Description

Takes an unencoded single qubit ket and converts it to a 7-qubit Steane encoded ket

Usage

```
Steane(v)
```

Arguments

v	Single qubit ket to Steane encode

Value

Steane encoded ket containing 7 qubits

Examples

```
Steane(ket(1,0))
Steane(ket(0,1))
Steane(ket(1,1))
```

SteaneCorrect *SteaneCorrect*

Description

Performs Steane error correction on an encoded qubit. Useful explanation provided by Devitt <DOI:10.1088/0034-4885/76/7/076001>

Usage

```
SteaneCorrect(v)
```

Arguments

v	Steane encoded qubit ket

Value

Steane encoded ket after error correction has been performed

Examples

```
## Not run:
SteaneCorrect(Steane(ket(1,0)))
SteaneCorrect(Steane(ket(0,1)))
SteaneCorrect(Steane(ket(1,1)))
SteaneCorrect(single(X(),n=7,t=2,Steane(ket(1,0))))

## End(Not run)
```

SWAP *SWAP*

Description

If no argument is supplied, returns the matrix of SWAP gate. If ket given as input, applies an SWAP gate to the input ket and returns the resulting ket

Usage

```
SWAP(...)
```

Arguments

...	No argument, or ket (column vector) that is input to the gate

Value

Matrix of the SWAP gate or ket after an SWAP gate is applied

Examples

```
SWAP(ket(0,1,0,0))
SWAP()
```

swapTest *swapTest*

Description

Encodes absolute square of inner product of two states, |<a|b>|^2, into an ancilliary qubit. It swaps the states of |a> and |b> conditioned on the ancilla which results in a state where the probability of measuring the ancilla qubit in the 0 state is equal to 1/2 - 1/2*(|<a|b>|^2). The ancilla qubit is inserted before qubit index 0, as the most significant qubit. Full explanation can be found in "Supervised Learning with Quantum Computers" <DOI:10.1007/978-3-319-96424-9>.

Usage

```
swapTest(v,a,b)
```

Arguments

v	Ket (column vector) that swap test is applied to. Should be a tensor product of two quantum state.
a	Vector of indices of |a> within v
b	Vector of indices of |b> within v

Value

Ket which contains the modified input ket, v, along with a leading ancillary qubit.

Examples

```
swapTest(intket(3,4),a=0:1,b=2:3)
swapTest(intket(5,4),a=0:1,b=2:3)
```

`SynthesizeCircuit` *SynthesizeCircuit*

Description

Converts the list form of a quantum circuit into a matrix representation. If the input is a single list, this function multiplies each entry. If each entry is a 4x4 unitary matrix, this function will multiply all, starting with the first, and generate a single 4x4 unitary matrix. If the input is a list of lists, this function will perform the same operation but interleave each list. The lists can be of different lengths.

Usage

```
SynthesizeCircuit(l)
```

Arguments

l	list containing the quantum operators of each cycle. The quantum operators should be unitary matrices which act on a number of qubits. Each entry in l should be of the same dimension. Optionally, l can be a of such lists, in which case each list will be interleaved.

Value

A matrix representing the result of applying each operation listed in l

Examples

```
## Not run:
 SynthesizeCircuit( list( tensor(X(),X()) , tensor(Y(),X()),
   tensor(I(),X()) , tensor(Z(),Z())  ))

## End(Not run)
## Not run:
 SynthesizeCircuit( list(  list(  tensor(X(),X()) , tensor(I(),X()) ),
   list( tensor(Y(),X()) , tensor(Z(),Z()) ) ) )

## End(Not run)
```

T *T*

Description

If no argument is supplied, returns the matrix of T gate. If ket given as input, applies a T gate to the input ket and returns the resulting ket

Usage

```
T(...)
```

Arguments

...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the T gate or ket after an T gate is applied

Examples

```
T(ket(1,1))
T()
```

teleport *teleport*

Description

Shows the steps of teleporting a single qubit

Usage

```
teleport(v)
```

Arguments

v	Ket (column vector) to teleport

Value

String describing teleportation process

Examples

```
teleport(ket(2,1))
```

tensor *tensor*

Description

Takes all arguments and combines them as a tensor product. Can be used to create a unified vector that represents multiple qubits or to create higher dimensional gates. If the inputs are two n-dimensional kets, the output is a 2-n dimensional ket representing the combined system.

Usage

```
tensor(...)
```

Arguments

...	kets (column vectosr) or gates (matrices) to take tensor product of

Value

The tensor product of all supplied arguments

Examples

```
tensor(ket(1,0),ket(0,1),ket(1,0),ket(1,0))
tensor(ket(1,1,1,1),ket(1,0,0,1))
tensor(X(),I())
tensor(H(),H(),H())
```

testGate *testGate*

Description

Takes a given quantum gate and tests it with computational basis states as input. Can test a subset of possible inputs if specified, otherwise it tests all possible inputs. Useful for testing user defined gates.

Usage

```
testGate(g,inputs=0:(dim(g)[1]-1) )
```

Arguments

g	Matrix that represents a quantum gate (operation)
inputs	Vector of indices of computational basis states to test. Default is that all computational basis states will be tested

Value

None

Examples

```
testGate(CX())
testGate(CX(),inputs=c(0,1))
```

TOFFOLI *TOFFOLI*

Description

If no argument is supplied, returns the matrix of TOFFOLI gate. If ket given as input, applies a TOFFOLI gate to the input ket and returns the resulting ket. If byCycle is TRUE, it generates the cycles which implement the TOFFOLI gate with standard gates.

Usage

```
TOFFOLI(...,byCycle=FALSE,n=3,cQubits=c(0,1),tQubit=2)
```

Arguments

...	No argument, or an 8 dimensional (3 qubit) ket (column vector) that is input to the gate
byCycle	Boolean specifying if the circuit should be generated. If TRUE, rather than returning a matrix or performing a TOFFOLI gate, it returns a list of cycles of standard gates which implements the TOFFOLI gate.
n	Number of qubits in the system.
cQubits	Vector of control qubit indices. Indexed from 0.
tQubit	Index of target qubit. Indexed from 0.

Value

Matix of the TOFFOLI gate or ket after a TOFFOLI gate is applied. If byCycle is TRUE, a list of cycles implementing the TOFFOLI.

Examples

```
TOFFOLI(ket(1,1,1,1,0,1,0,1))
TOFFOLI()
```

U *U*

Description

Takes as input a list of gates (in matrix form) and creates the tensor product, forming a higher dimensional gate. If the last argument is a ket, the gate is applied to the ket and the ket returned. If last argument is another gate, it returns the tensor product of all gates

Usage

```
U(...)
```

Arguments

...	List of quantum gates in matrix form, optionally the last argument is ket (column vector) that is input to the gate

Value

Matix of the gate that is the tensor product of all input gates, or the ket which is the result of the gate applied to the input ket

Examples

```
U(X(),ket(1,0))
U(H(),H(),ket(1,0,0,0))
U(I(),X(),ket(1,0,1,0))
U(I(),X())
```

Uf *Uf*

Description

Generates an operator (oracle) of specified size that implements the function that is passed to it. Assumes there are n qubits in data register and m qubits in target register.

Usage

```
Uf(fun,n,m)
```

Arguments

fun	Function of an n-bit argument that produces an m-bit result
n	Integer that specifies the number of qubits in the data register, same as number of bits to function
m	Integer that specifies the number of qubits in the target register, same as number of bits as output of function

Value

Matrix of the operator (oracle) which implements the specified function)

Examples

```
Uf(function(x){x - floor(x/2)*2},1,1)
Uf(function(x){0},2,2)
Uf(function(x){1},2,2)
Uf(function(x){x - floor(x/4)*4},2,2)
Uf(function(x){x^3},3,4)
```

---

unitary *unitary*

Description

Determines whether an operation (matrix) is unitary by comparing its inverse to its adjoint

Usage

```
unitary(m,epsilon=1e-13)
```

Arguments

m	gate operation (gate) that is to be checked
epsilon	Amount of error to tolerate. Accounts for numerical precision on practical computing systems

Value

boolean indicating whether matrix is unitary or not

Examples

```
unitary(mm(0,1,1,0))
unitary(mm(0,1,1,0),1e-15)
```

X *X*

Description

If no argument is supplied, returns the matrix of X gate. If ket given as input, applies an X gate to the input ket and returns the resulting ket

Usage

```
X(...)
```

Arguments

...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the X gate or ket after an X gate is applied

Examples

```
X(ket(1,0))
X()
```

Y *Y*

Description

If no argument is supplied, returns the matrix of the Y gate. If ket given as input, applies a Y gate to the input ket and return the resulting ket

Usage

```
Y(...)
```

Arguments

...	No argument, or ket (column vector) that is input to the gate

Value

Matrix of the Y gate or ket after a Y gate is applied

Examples

```
Y(ket(1,0))
Y()
```

Z Z

Description

If no argument is supplied, returns the matrix of Z gate. If ket given as input, applies a Z gate to the input ket and returns the resulting ket

Usage

```
Z(...)
```

Arguments

...	No argument, or ket (column vector) that is input to the gate

Value

Matix of the Z gate or ket after a Z gate is applied

Examples

```
Z(ket(1,0))
Z()
```

Index