Quantum Programming

State Creation

```
>> cd <location>\qit
>> init
MATLAB Quantum Information Toolkit, version 0.10.0

>> a=state('0')
a = +1 |0>

>> state('1')
ans = +1 |1>

>> state([1/sqrt(2) 1/sqrt(2)])
ans = +0.707107 |0> +0.707107 |1>

>> b=state([0.5 0.5 0.5 0.5],[2 2])
b = +0.5 |00> +0.5 |01> +0.5 |10> +0.5 |11>
```

Inner Product

```
>> x=[3 1 2]
>> y=[5 6 4]
>> dot(x,y)
ans =
    29
```

Tensor Product

```
x=[3 1 2]'
x =
     3
     1
     2
y=[5 6 4]'
y =
     5
     6
     4
```

```
kron(x,y)
ans =
    15
    18
    12
     5
     6
     4
    10
    12
     8
```

```
>> tensor(a,b)
ans= +0.5 |000> +0.5 |001> +0.5 |010> +0.5 |011>
```

Outer Product

```
>> x=[3 1 2]'
x =
     3
     1
     2
>> yT=[5 6 4]
yT =
     5     6     4
>> kron(x,yT)
```

```
ans =
    15    18    12
     5     6     4
    10    12     8
```

Hadamard

```
>> a
a = +1 |0>

>> H=qit.H
H =
    0.7071    0.7071
```

```
    0.7071    -0.7071
```

```
>> u_propagate(a,H)
ans = +0.707107 |0> +0.707107 |1>
>> gate.qft(2)
   0.7071 + 0.0000i   0.7071 + 0.0000i
   0.7071 + 0.0000i  -0.7071 + 0.0000i
```

Identity

```
>> I=gate.id(2)
    (1,1)         1
    (2,2)         1
```

X

```
>> gate.mod_inc(-1, 2)
    (2,1)         1
    (1,2)         1
```

Z

```
H * gate.mod_inc(1, 2) * H'
   1.0000 + 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i  -1.0000 + 0.0000i
```

(A)CNOT

```
>> ACNOT=gate.mod_add(2, 2)
    (1,1)          1
    (2,2)          1
    (4,3)          1
    (3,4)          1
```

(B)CNOT

```
>> S=gate.swap(2,2)
    (1,1)          1
    (3,2)          1
    (2,3)          1
    (4,4)          1
>> A*S*A
    (1,1)          1
    (4,2)          1
    (3,3)          1
    (2,4)          1
```

Measurement

```
a=state([0 1 0 0],[2 2])
a = +1 |01>
>> [~, b1, ~]=measure(a, 1)
b1 =
     1
>> cbit=b1-1
cbit =
     0
[~, b2, ~]=measure(a, 2)
b2 =
```

```
     2
>> cbit=b2-1
cbit =
     1
```

Superposition Measurement

```
reg0=state(0.5*[1 1 1 1],[2 2])
reg0 = +0.5 |00> +0.5 |01> +0.5 |10> +0.5 |11>
>> [~,b,reg1]=measure(reg0,1)
b =
     2
reg1 = +0.707107 |10> +0.707107 |11>
>> [~,b,reg1]=measure(reg0,1)
```

```
b =
     1
reg1 = +0.707107 |00> +0.707107 |01>
>> [~,b,reg2]=measure(reg1,1)
b =
     1
re2 = +0.707107 |00> +0.707107 |01>
```

Circuit

```
>> C=tensor(H,I)
   (1,1)        0.7071 + 0.0000i
   (3,1)        0.7071 + 0.0000i
   (2,2)        0.7071 + 0.0000i
   (4,2)        0.7071 + 0.0000i
   (1,3)        0.7071 + 0.0000i
   (3,3)       -0.7071 + 0.0000i
   (2,4)        0.7071 + 0.0000i
   (4,4)       -0.7071 + 0.0000i
>> a=state([0 1 0 0],[2 2])
a = +1 |01>
>> u_propagate(a,C)
```

```
ans = +0.707107 |01> +0.707107 |11>
```

Helper

```
classdef helper < handle
    % static properties
    properties (Constant)
        % Gates
        ACNOT=gate.mod_add(2, 2)
        BCNOT=helper.ACNOT*helper.SWAP*helper.ACNOT
        H=gate.qft(2)
        I=gate.id(2)
        SWAP=gate.swap(2,2)
        X=gate.mod_inc(-1, 2)
        Y=1/i*helper.Z*helper.X
        Z=helper.H*gate.mod_inc(1, 2)*helper.H'
    end
end
```

Conditional Gate

```
>> reg0=tensor(state('0'),state('1'))
reg0 = +1 |01>
>> C=tensor(H,I)
   (1,1)       0.7071 + 0.0000i
   (3,1)       0.7071 + 0.0000i
   (2,2)       0.7071 + 0.0000i
   (4,2)       0.7071 + 0.0000i
   (1,3)       0.7071 + 0.0000i
   (3,3)      -0.7071 + 0.0000i
   (2,4)       0.7071 + 0.0000i
   (4,4)      -0.7071 + 0.0000i
>> reg1=u_propagate(reg0,C)
reg1 = +0.707107 |01> +0.707107 |11>
>> [~,b,reg2]=measure(reg1,1)
b =
     2
```

```
reg2 = +1 |11>
>> cbit=b-1
cbit =
     1
```

```
>> B=tensor(helper.I,helper.X)
   (2,1)         1
   (1,2)         1
   (4,3)         1
   (3,4)         1
>> A=tensor(helper.I,helper.I)
   (1,1)         1
   (2,2)         1
   (3,3)         1
   (4,4)         1
>> u_propagate(reg2,cbit*B+double(~cbit)*A)
ans = +1 |10>
>> cbit=0
cbit =
     0
>> u_propagate(reg2,cbit*B+double(~cbit)*A)
ans = -1 |11>
```

Tensor Power

```
classdef helper < handle
    % static properties
    properties (Constant)
        ...
    end
    methods (Static)
        function [ P ] = power(G, n)
            % return the nth tensor power of gate G (n>=0)
            if (n==0)
                P=lmap([1], {[ 1 1 ]});
            else
                P=G;
                for i=2:n
                    P=tensor(P,G);
                end
            end
        end
    end
end
```

```
>> helper.power(helper.H,0)

ans = 1

>> helper.power(helper.H,1)
```

```
   0.7071 + 0.0000i   0.7071 + 0.0000i
   0.7071 + 0.0000i  -0.7071 + 0.0000
>> helper.power(helper.H,2)
  0.5000 + 0.0000i   0.5000 + 0.0000i   0.5000 + 0.0000i   0.5000 + 0.0000i
  0.5000 + 0.0000i  -0.5000 + 0.0000i   0.5000 + 0.0000i  -0.5000 + 0.0000i
  0.5000 + 0.0000i   0.5000 + 0.0000i  -0.5000 + 0.0000i  -0.5000 + 0.0000i
  0.5000 + 0.0000i  -0.5000 + 0.0000i  -0.5000 + 0.0000i   0.5000 - 0.0000i
```

In and Out Quregisters Init

```
>> ketx=state('00')
ketx = +1 |00>
>> kety=state('00')
kety = +1 |00>

>> tensor(helper.power(helper.H,2),...
   helper.power(helper.I,2))*...
   tensor(ketx,kety)
ans = +0.5 |0000> +0.5 |0100> +0.5 |1000> +0.5 |1100>
```

Deutsch's Algo

```
function y = mydeutsch(f)
%! f = bininary function, an array, e.g., [1 0]
    % H gate creation
    H=helper.H;
    % Identity gate creation
    I=helper.I;
    % define the U gate
    U=[ not(f(1)) f(1) 0 0 ;
        f(1) not(f(1)) 0 0 ;
        0 0 not(f(2)) f(2);
        0 0 f(2) not(f(2)) ];
    % convert to a "lmap"
    Uf=lmap(U, {[ 2 2 ], [ 2 2 ]});
    % Build the circuit
    C=tensor(H,I)*Uf*tensor(H,H);
    % Propagate input state through the circuit
    reg=u_propagate(state('01'),C);
    % Measure the first qubit
    [~, b, ~]=measure(reg, 1);
    % Return the result
    y=b-1; % conversion to 0s and 1s
```

```
>> mydeutsch([0 0])          ans =     0
```

```
>> mydeutsch([0 1])          ans =     1
>> mydeutsch([1 0])          ans =     1
>> mydeutsch([1 1])          ans =     0
```

Reordering

```
>> r=state('00100')
|00100>

>> reorder(r,[3 4])
|00010>

>> reorder(r,[3 5])
|00001>
```

Sum Circuit – Solution 1

```
CNOT=helper.ACNOT;
H=helper.H;
I=helper.I;
SWAP=helper.SWAP;

% input
in=u_propagate(state('000'),tensor(H,H,I));
display(in);

% apply 1st CNOT
s0=u_propagate(in,tensor(I,CNOT));

% reorder qubits 1 & 2
s1=reorder(s0,[2 1 3]);

% apply 2nd CNOT
s2=u_propagate(s1,tensor(I,CNOT));

% reorder qubits 1 & 2
out=reorder(s2,[2 1 3]);
display(out);
```

```
in =
   +0.5 |000> +0.5 |010> +0.5 |100> +0.5 |110>
out =
   +0.5 |000> +0.5 |011> +0.5 |101> +0.5 |110>
```

Sum Circuit – Solution 2

```
% gate that reorders qubits 1 and 2
R=tensor(SWAP,I);
C=tensor(I,CNOT)*R*tensor(I,CNOT)*R;

% input
in=u_propagate(state('000'),tensor(H,H,I));
display(in);

% apply gate
out=u_propagate(in,C);
display(out);
```

```
in =

   +0.5 |000> +0.5 |010> +0.5 |100> +0.5 |110>

out =

   +0.5 |000> +0.5 |011> +0.5 |101> +0.5 |110>
```

Sum Circuit – Solution 3

```
function [ G ] = R(k,n)
    % ret. a n by n gate swapping qubits k & k+1
    % assert: 0<k<n
    if (k+1<n)
        G=tensor(...
            helper.SWAP,...
            helper.power(helper.I,n-(k+1)));
    else
        G=helper.SWAP;
    end
    if (k>1)
        G=tensor(helper.power(helper.I,k-1),G);
    end
end


%
% gate that reorders qubits 1 and 2
R=helper.R(1,3);
C=tensor(I,CNOT)*R*tensor(I,CNOT)*R;

% input
```

```
in=u_propagate(state('000'),tensor(H,H,I));
display(in);

% apply gate
out=u_propagate(in,C);
display(out);
```

Block SWAP (BSWAP)

```
function [ G ] = BSWAP(k,l,n)
   % assert: 0<k,l<n
   G=helper.power(helper.I,n);
   if (k<l)
      for m=k:l-1
         G=helper.R(m,n)*G;
      end
   elseif (k>l)
      for m=k-1:-1:l
         G=helper.R(m,n)*G;
      end
   end
end
```

```
>> s=state('110010');
```

```
>> B=helper.BSWAP(1,5,6);
>> u_propagate(s,B)
ans = +1 |100110>
>> A=helper.BSWAP(5,1,6);
>> u_propagate(t,A)
ans = +1 |110010>
```

Bell-EPR Pair (ebit) - Production

```
classdef helper < handle
    % static properties
    properties (Constant)
    ...
        % Bell-EPR production
        E=helper.ACNOT*tensor(helper.H,helper.I);
        % Bell-EPR measurement preperation
    ...
    function [ S ] = ebit(in)
        % return a Bell-EPR state
        % in=input state, e.g., |00>
        S=helper.E*in;
    end
```

```
>> helper.ebit(state('00'))
ans = +0.707107 |00> +0.707107 |11>
>> helper.ebit(state('01'))
ans = +0.707107 |01> +0.707107 |10>
```

```
>> helper.ebit(state('10'))
ans = +0.707107 |00> -0.707107 |11>
>> helper.ebit(state('11'))
ans = +0.707107 |01> -0.707107 |10>
```

Bell-EPR Pair (ebit) – Measurement Preparation

```
classdef helper < handle
    % static properties
    properties (Constant)
    ...
    prep=tensor(helper.H,helper.I)*helper.ACNOT
```

```
>> helper.prep*helper.ebit(state('00'))
ans = +1 |00>

>> helper.prep*helper.ebit(state('01'))
ans = +1 |01>

>> helper.prep*helper.ebit(state('10'))
ans = +1 |10>
```

```
>> helper.prep*helper.ebit(state('11'))
ans = +1 |11>
```

Teleportation

```
% qubit to teleport
phi=state('0');

% ebit creation
psi=helper.ebit(state('00'));

% combine with phi
r1=tensor(phi,psi);

% measurement preparation
r2=u_propagate(r1,tensor(helper.prep,helper.I));

% measurement
[~, b1, r3]=measure(r2, 1); b1=b1-1;
[~, b2, r4]=measure(r3, 2); b2=b2-1;

% conditional X gate
I3=helper.power(helper.I,3);
IIX=tensor(helper.I,helper.I,helper.X);
r5=u_propagate(r4,b2*IIX+double(~b2)*I3);

% conditional Z
IIZ=tensor(helper.I,helper.I,helper.Z);
```

```
r6=u_propagate(r5,b1*IIZ+double(~b1)*I3);
```

Teleportation

phi = *(teleported qubit)*

+1 |0>

psi = *(ebit)*

+0.707107 |00> +0.707107 |11>

r1 = *(phi and psi together)*

+0.707107 |000> +0.707107 |011>

r2 = *(after prep)*

+0.5 |000> +0.5 |011> +0.5 |100> +0.5 |111>

r4 = *(after measurement)*

+1 |111>

r5 = *(after conditional X gate)*

+1 |110>

r6 = *(after conditional Z gate)*

+1 |110>

Teleportation

phi = *(teleported qubit)*

+1 |0>

r2 = *(after prep)*

+0.5 |000> +0.5 |011> +0.5 |100> +0.5 |111>

phi =

+1 |1>

r2 =

+0.5 |001> +0.5 |010> -0.5 |101> -0.5 |110>

Teleportation

```
phi=fix_phase(u_propagate(state('0',2), rand_SU(2)))
```

phi = *(teleported qubit)*

+0.220078 |0> +(-0.12515-0.96742i) |1>

psi = *(ebit)*

+0.707107 |00> +0.707107 |11>

r1 = *(phi and psi together)*

+0.155618 |000>

+0.155618 |011>

+(-0.088491-0.68407i) |100>

+(-0.088491-0.68407i) |111>

r2 = *(after prep)*

+0.110039 |000>

+(-0.062573-0.48371i) |001>

+(-0.062573-0.48371i) |010>

+0.110039 |011> +0.110039 |100>

+(0.062573+0.48371i) |101>

+(0.062573+0.48371i) |110>

+0.110039 |111>

r4 = *(after measurement)*

+0.220078 |000> +(-0.12515-0.96742i) |001>

r5 = *(after conditional X gate)*

+0.220078 |000> +(-0.12515-0.96742i) |001>

r6 = *(after conditional Z gate)*

+0.220078 |000> +(-0.12515-0.96742i) |001>

Error Detection

```
%%% code word with single bit error
%
codeword=state('010')
%
%%% Error detection
%
in=tensor(codeword,state('00'))
%%% 1st CNOT
temp1=reorder(in,[1 4 3 2 5]); % reorder 2 & 4
temp2=u_propagate(temp1,tensor(CNOT,I,I,I));
in=reorder(temp2,[1 4 3 2 5]); % reorder 2 & 4
%%% 2nd CNOT
temp1=reorder(in,[1 2 4 3 5]); % reorder 3 & 4
temp2=u_propagate(temp1,tensor(I,CNOT,I,I));
in=reorder(temp2,[1 2 4 3 5]); % reorder 3 & 4
%%% 3rd CNOT
temp1=reorder(in,[1 2 5 4 3]); % reorder 3 & 5
temp2=u_propagate(temp1,tensor(I,CNOT,I,I));
in=reorder(temp2,[1 2 5 4 3]); % reorder 3 & 5
%%% 4th CNOT
temp1=reorder(in,[1 2 3 5 4]); % reorder 4 & 5
temp2=u_propagate(temp1,tensor(I,I,CNOT,I));
in=reorder(temp2,[1 2 3 5 4]); % reorder 4 & 5
display(in);
% Measure the 4th qubit
[~, b1, in]=measure(in, 4);b1=b1-1;
display(b1);
% Measure the 5th qubit
[~, b2, in]=measure(in, 5);b2=b2-1;
display(b2);
```

```
>> errordetection

codeword = +1 |010>

in = +1 |01000>

in = +1 |01011>

b1 = 1

b2 = 1
```