Quantum Computing

Lab 1. Quantum Circuits

Frank C Langbein

[1]:
```
from qutip import *
import numpy as np
```

Quirk

- Start quirk (local) or remote (remote)
- Familiarise yourself with the top toolbox and understand how to construct circuits
  - View the tutorial
- Create a circuit to generate the following single qubit states
  - $(|0\rangle + |1\rangle)/\sqrt{2}$
  - $(|0\rangle - i|1\rangle)/\sqrt{2}$
  - $(|0\rangle + i|1\rangle)/\sqrt{2}$
  - $|1\rangle$
- Create a circuit to produce a GHZ (Greenberger Horne Zeilinger) state: $(|000\rangle + |111\rangle)/\sqrt{2}$
  - Note, GHZ states for more quibits are equivalently defined to this 3-qubit state

Solution

- Single Qubit State circuit
- GHZ state circuit
  - Just add more CNOTs in the same manner for GHZ states with more qubits

CNOT with Hadamard Gates

- Consider the Hadamard with CNOT circuit
- Which two qubit gate is equivalnet to this circuit? Proof your answer by calculating the circuit matrix (with qutip or manually) and construct the equivalent circuit in quirk.

Solution

[2]:
```
# Python code to calculate circuit operator

# Individual gates
H = snot()
CNOT = tensor( qeye(2), ket("0")*bra("0") ) + tensor( sigmax(), ket("1")*bra("1") )

# Full circuit
CIRCUIT = tensor(H,H) * CNOT * tensor(H,H)
```

```
print(CIRCUIT.full())
```

```
[[1.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 1.+0.j]
 [0.+0.j 0.+0.j 1.+0.j 0.+0.j]]
```

```
/usr/lib/python3/dist-packages/ipykernel_launcher.py:4: DeprecationWarning: Importing
functions/classes of the qip submodule directly from the namespace qutip is deprecated.
Please import them from the submodule instead, e.g.
from qutip.qip.operations import cnot
from qutip.qip.circuit import QubitCircuit

  after removing the cwd from sys.path.
```

This is a CNOT gate with the first qubit being the target and the second qubit the control:

equivalent quirk circuit

This demonstrates the equivalence of these circuits:

demonstration of equivalent quirk circuit

Circuit Equivalence

Show that in this circuit from the lecture, the gate on the first three qubits is identical to the gate on the last three qubits (using qutip or manually).

Create a similar circuit equivalence for a controlled Y gate.

Solution

[3]:
```
# Python code to compare the circuits

# CCNOT circuit for first three quibits:
# (|00><00| + |01><01| + |10><10|) x I + |11><11| x X
CCNOT = tensor (ket("00")*bra("00")+ket("01")*bra("01")+ket("10")*bra("10"), qeye(2)) \
        + tensor(ket("11")*bra("11"), sigmax())

print("CCNOT = "); print(CCNOT.full())

# First Controlled X^1/2 gate: I x [(0><0| x I) + (|1><1|) x X^{1/2})]
G1 = tensor(qeye(2), \
            tensor(ket("0")*bra("0"), qeye(2)) \
            + tensor(ket("1")*bra("1"), sigmax().sqrtm()))

# Controlled NOT gate (used twiced): [(0><0| x I) + (|1><1|) x X)] x I
G2 = tensor(tensor(ket("0")*bra("0"), qeye(2)) \
            + tensor(ket("1")*bra("1"), sigmax()), \
            qeye(2))

# Calulcate X^{-1/2} via numpy and generate qutip gate from it
X2I = Qobj(np.linalg.inv(sigmax().sqrtm().full()))
# Controlled X^{-1/2} gate: I x [ (|0><0| x I) + (|1><1| * X^{-1/2}) ]
```

```
G3 = tensor(qeye(2), \
           tensor(ket("0")*bra("0"), qeye(2)) \
           + tensor(ket("1")*bra("1"), X2I) )

# Second controlled X^1/2 gate: |0><0| x I x I + |1><1| x I x X^{1/2}
G4 = tensor(ket("0")*bra("0"), tensor(qeye(2), qeye(2))) + \
     tensor(ket("1")*bra("1"), tensor(qeye(2), sigmax().sqrtm()) )

# Full circuit for last three qubits:
CIRCUIT = G4 * G2 * G3 * G2 * G1

print("CIRCUIT = "); print(CIRCUIT.full())

# Compare the two circuit operators
# If A and B are unitary and B is the inverse of A, then
#    A * B^\dagger = I
# The trace of a matrix is unique and 8 for the identiy of a 3 qubit system. So
#   tr(A * B^\dagger)/8 = 1
# iff A and B are identical
print("Fidelity =", (CIRCUIT * CCNOT.dag()).tr() / 8)
```

```
CCNOT =
[[1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j]]
CIRCUIT =
[[1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j]]
Fidelity = 1.0
```

The approach taking the square roots of the operators is universal to get a CC-Operator gate.

Circuit for CCY