example-quantum-gates

June 25, 2014

1 QuTiP example: Quantum Gates and their usage

Author: Anubhav Vardhan (anubhavvardhan@gmail.com)

For more information about QuTiP see http://qutip.org

```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: from IPython.display import Image
```

```
In [3]: from qutip import *
```

1.1 Introduction

In quantum computing and specifically the quantum circuit model of computation, a quantum gate (or quantum logic gate) is a basic quantum circuit operating on a small number of qubits. They are the building blocks of quantum circuits, like classical logic gates are for conventional digital circuits.

Unlike many classical logic gates, quantum logic gates are reversible. However, classical computing can be performed using only reversible gates. For example, the reversible Toffoli gate can implement all Boolean functions. This gate has a direct quantum equivalent, showing that quantum circuits can perform all operations performed by classical circuits.

1.1.1 References

http://en.wikipedia.org/wiki/Quantum_gate

1.2 Gates in QuTiP and their representation

1.2.1 Controlled-PHASE

```
In [4]: cphase(pi/2)
```

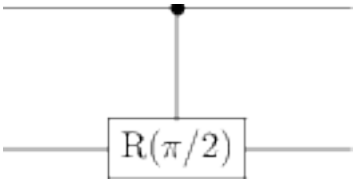
Out[4]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = False

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0j \end{pmatrix}$$

```
In [5]: Image(filename='images/cphase.png')
```

Out[5]:



1.2.2 Rotation about X-axis

In [6]:
```
rx(pi/2)
```

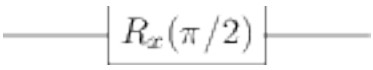
Out[6]:

Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = False

$$\begin{pmatrix} 0.707 & -0.707j \\ -0.707j & 0.707 \end{pmatrix}$$

In [7]:
```
Image(filename='images/rx.png')
```

Out[7]:



1.2.3 Rotation about Y-axis

In [8]:
```
ry(pi/2)
```

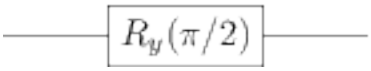
Out[8]:

Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = False

$$\begin{pmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{pmatrix}$$

In [9]:
```
Image(filename='images/ry.png')
```

Out[9]:



1.2.4 Rotation about Z-axis

In [10]:
```
rz(pi/2)
```

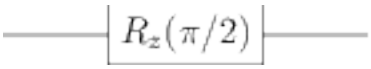
Out[10]:

Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = False

$$\begin{pmatrix} (0.707 - 0.707j) & 0.0 \\ 0.0 & (0.707 + 0.707j) \end{pmatrix}$$

In [11]: Image(filename='images/rz.png')

Out[11]:



1.2.5 CNOT

In [12]: cnot()

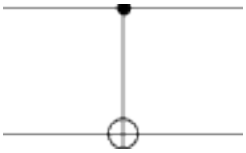
Out[12]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

In [13]: Image(filename='/home/anubhav/Images/cnot.png')

Out[13]:



1.2.6 CSIGN

In [14]: csign()

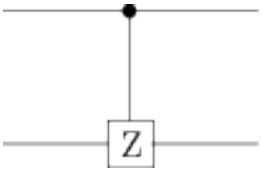
Out[14]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -1.0 \end{pmatrix}$$

In [15]: Image(filename='images/csign.png')

Out[15]:



1.2.7 Berkeley

In [16]:
```
berkeley()
```

Out[16]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = False

$$\begin{pmatrix} 0.924 & 0.0 & 0.0 & 0.383j \\ 0.0 & 0.383 & 0.924j & 0.0 \\ 0.0 & 0.924j & 0.383 & 0.0 \\ 0.383j & 0.0 & 0.0 & 0.924 \end{pmatrix}$$

In [17]:
```
Image(filename='images/berkeley.png')
```

Out[17]:

1.2.8 SWAPalpha

In [18]:
```
swapalpha(pi/2)
```

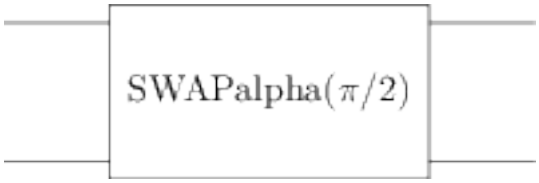
Out[18]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = False

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & (0.610 - 0.488j) & (0.390 + 0.488j) & 0.0 \\ 0.0 & (0.390 + 0.488j) & (0.610 - 0.488j) & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In [19]:
```
Image(filename='images/swapalpha.png')
```

Out[19]:



1.2.9 FREDKIN

In [20]:
```
fredkin()
```

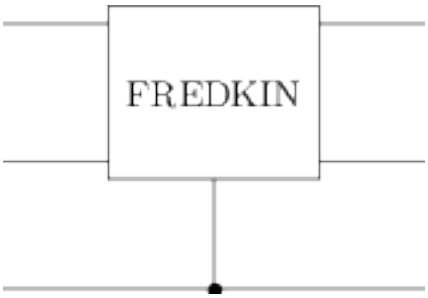
Out[20]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In [21]:
```
Image(filename='images/fredkin.png')
```

Out[21]:



1.2.10 TOFFOLI

In [22]:
```
toffoli()
```

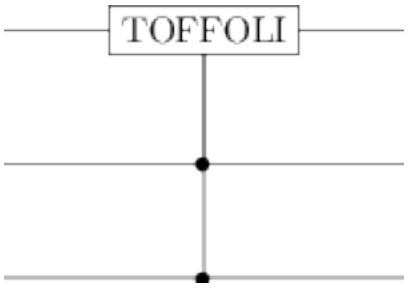
Out[22]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

In [23]:
```
Image(filename='images/toffoli.png')
```

Out[23]:



1.2.11 SWAP

In [24]:
```
swap()
```

Out[24]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In [25]:
```
Image(filename='images/swap.png')
```

Out[25]:

1.2.12 ISWAP

In [26]:
```
iswap()
```

Out[26]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = False

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0j & 0.0 \\ 0.0 & 1.0j & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In [27]:
```
Image(filename='images/iswap.png')
```

Out[27]:

1.2.13 SQRTiSWAP

In [28]:
```
sqrtiswap()
```

Out[28]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = False

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.707 & 0.707j & 0.0 \\ 0.0 & 0.707j & 0.707 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In [29]:
```
Image(filename='images/sqrtiswap.png')
```

Out[29]:

1.2.14 SQRTSWAP

In [30]:
```
sqrtswap()
```

Out[30]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = False

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & (0.500+0.500j) & (0.500-0.500j) & 0.0 \\ 0.0 & (0.500-0.500j) & (0.500+0.500j) & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In [31]:
```
Image(filename='images/sqrtswap.png')
```

Out[31]:

1.2.15 SQRTNOT

In [32]:
```
sqrtnot()
```

Out[32]:

Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = False

$$\begin{pmatrix} (0.500+0.500j) & (0.500-0.500j) \\ (0.500-0.500j) & (0.500+0.500j) \end{pmatrix}$$

In [33]:
```
Image(filename='images/sqrtnot.png')
```

Out[33]:

1.2.16 HADAMARD

In [34]: `snot()`

Out[34]:

Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = True

$$\begin{pmatrix} 0.707 & 0.707 \\ 0.707 & -0.707 \end{pmatrix}$$

In [35]: `Image(filename='images/snot.png')`

Out[35]:

1.2.17 PHASEGATE

In [36]: `phasegate(pi/2)`

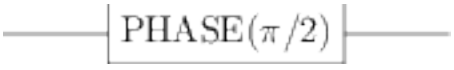
Out[36]:

Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = False

$$\begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0j \end{pmatrix}$$

In [37]: `Image(filename='images/phasegate.png')`

Out[37]:



1.2.18 GLOBALPHASE

In [38]: `globalphase(pi/2)`

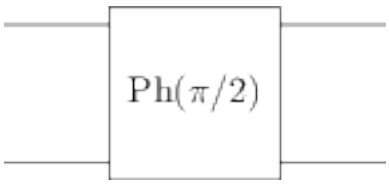
Out[38]:

Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = False

$$\begin{pmatrix} 1.0j & 0.0 \\ 0.0 & 1.0j \end{pmatrix}$$

In [39]: `Image(filename='images/globalphase.png')`

Out[39]:



1.2.19 Expanding gates to larger qubit registers

The example above show how to generate matrice representations of the gates implemented in QuTiP, in their minimal qubit requirements. If the same gates is to be represented in a qubit register of size N, the optional keywork argument `N` can be specified when calling the gate function. For example, to generate the matrix for the CNOT gate for a $N = 3$ bit register:

In [40]:
```
cnot(N=3)
```

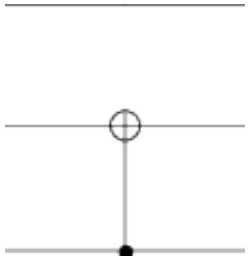
Out[40]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \end{pmatrix}$$

In [41]:
```
Image(filename='images/cnot310.png')
```

Out[41]:



Furthermore, the control and target qubits (when applicable) can also be similarly specified using keyword arguments `control` and `target` (or in some cases `controls` or `targets`):

In [42]:
```
cnot(N=3, control=2, target=0)
```

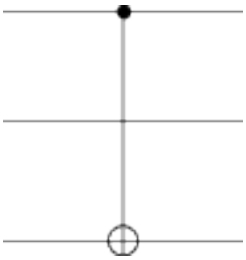
Out[42]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

```
In [43]: Image(filename='images/cnot302.png')
```

Out[43]:



1.3 Setup of a Qubit Circuit

The gates implemented in QuTiP can be used to build any qubit circuit using the class QubitCircuit. The output can be obtained in the form of a unitary matrix or a latex representation.

In the following example, we take a SWAP gate. It is known that a swap gate is equivalent to three CNOT gates applied in the given format.

```
In [44]: N = 2
         qc0 = QubitCircuit(N)
         qc0.add_gate("SWAP", [0, 1], None)
         qc0.png
```

Out[44]:


```
In [45]: U_list0 = qc0.unitary_matrix()
         U0 = gate_sequence_product(U_list0)
         U0
```

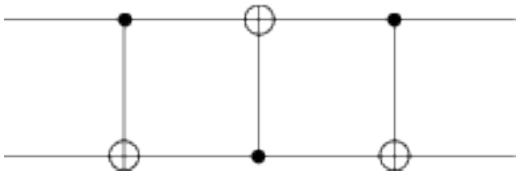
Out[45]:

Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In [46]:
```
qc1 = QubitCircuit(N)
qc1.add_gate("CNOT", 0, 1)
qc1.add_gate("CNOT", 1, 0)
qc1.add_gate("CNOT", 0, 1)
qc1.png
```

Out[46]:



In [47]:
```
U_list1 = qc1.unitary_matrix()
U1 = gate_sequence_product(U_list1)
U1
```

Out[47]:

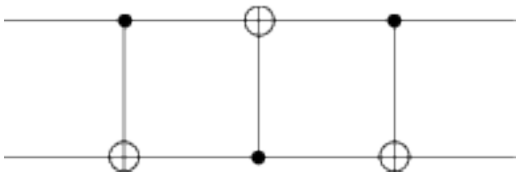
Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In place of manually converting the SWAP gate to CNOTs, it can be automatically converted using an inbuilt function in QubitCircuit

In [48]:
```
qc2 = qc0.resolve_gates("CNOT")
qc2.png
```

Out[48]:



In [49]:
```
U_list2 = qc2.unitary_matrix()
U2 = gate_sequence_product(U_list2)
U2
```

Out[49]:

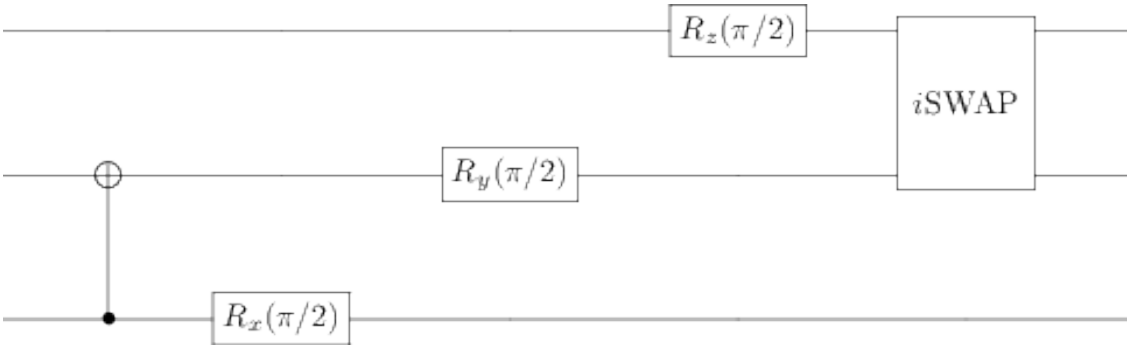
Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

1.4 Example of basis transformation

In [50]:
```
qc3 = QubitCircuit(3)
qc3.add_gate("CNOT", 1, 0)
qc3.add_gate("RX", 0, None, pi/2, r"\pi/2")
qc3.add_gate("RY", 1, None, pi/2, r"\pi/2")
qc3.add_gate("RZ", 2, None, pi/2, r"\pi/2")
qc3.add_gate("ISWAP", [1, 2])
qc3.png
```

Out[50]:



In [51]:
```
U3 = gate_sequence_product(qc3.unitary_matrix())
U3
```

Out[51]:

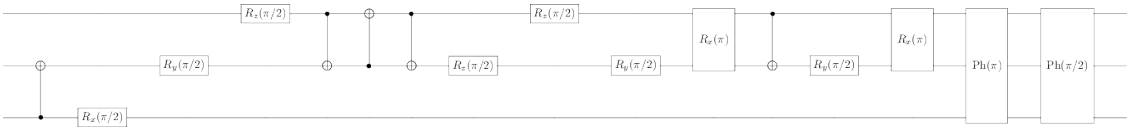
Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = False

$$\begin{pmatrix} (0.354-0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\ (0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & \\ 0.0 & (-0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (-0.354-0.354j) & \\ 0.0 & (0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & \\ (-0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & \\ (0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\ 0.0 & (0.354+0.354j) & 0.0 & (-0.354-0.354j) & 0.0 & (0.354-0.354j) & \\ 0.0 & (0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) & \end{pmatrix}$$

1.4.1 The transformation can either be only in terms of 2-qubit gates:

In [52]:
```
qc4 = qc3.resolve_gates("CNOT")
qc4.png
```

Out[52]:



In [53]:
```
U4 = gate_sequence_product(qc4.unitary_matrix())
U4
```

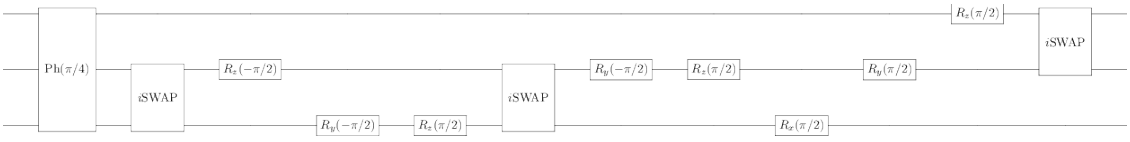
Out[53]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = False

$$\begin{pmatrix}
(0.354-0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\
(0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & \\
0.0 & (-0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (-0.354-0.354j) & \\
0.0 & (0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & \\
(-0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & \\
(0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\
0.0 & (0.354+0.354j) & 0.0 & (-0.354-0.354j) & 0.0 & (0.354-0.354j) & \\
0.0 & (0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) &
\end{pmatrix}$$

In [54]:
```
qc5 = qc3.resolve_gates("ISWAP")
qc5.png
```

Out[54]:



In [55]:
```
U5 = gate_sequence_product(qc5.unitary_matrix())
U5
```

Out[55]:

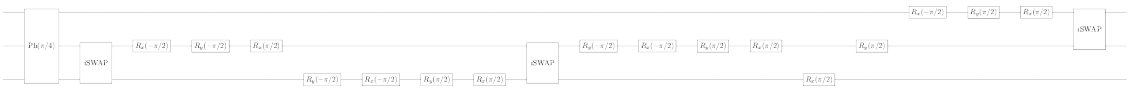
Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = False

$$\begin{pmatrix}
(0.354-0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\
(0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & \\
0.0 & (-0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (-0.354-0.354j) & \\
0.0 & (0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & \\
(-0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & \\
(0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\
0.0 & (0.354+0.354j) & 0.0 & (-0.354-0.354j) & 0.0 & (0.354-0.354j) & \\
0.0 & (0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) &
\end{pmatrix}$$

1.4.2 Or the transformation can be in terms of any 2 single qubit rotation gates along with the 2-qubit gate.

In [56]:
```
qc6 = qc3.resolve_gates(["ISWAP", "RX", "RY"])
qc6.png
```

Out[56]:



In [57]:
```
U6 = gate_sequence_product(qc6.unitary_matrix())
U6
```

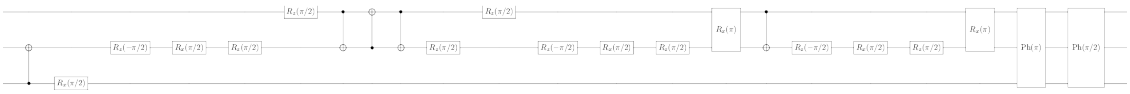
Out[57]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = False

$$\begin{pmatrix} (0.354-0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\ (0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & \\ 0.0 & (-0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (-0.354-0.354j) & \\ 0.0 & (0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & \\ (-0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & \\ (0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\ 0.0 & (0.354+0.354j) & 0.0 & (-0.354-0.354j) & 0.0 & (0.354-0.354j) & \\ 0.0 & (0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) & \end{pmatrix}$$

In [58]:
```
qc7 = qc3.resolve_gates(["CNOT", "RZ", "RX"])
qc7.png
```

Out[58]:



In [59]:
```
U7 = gate_sequence_product(qc7.unitary_matrix())
U7
```

Out[59]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = False

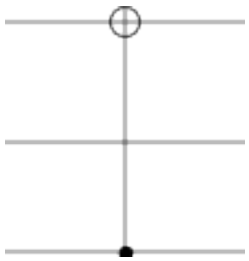
$$\begin{pmatrix} (0.354-0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\ (0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & \\ 0.0 & (-0.354+0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (-0.354-0.354j) & \\ 0.0 & (0.354+0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (0.354-0.354j) & \\ (-0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & (-0.354+0.354j) & 0.0 & \\ (0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) & 0.0 & \\ 0.0 & (0.354+0.354j) & 0.0 & (-0.354-0.354j) & 0.0 & (0.354-0.354j) & \\ 0.0 & (0.354-0.354j) & 0.0 & (0.354-0.354j) & 0.0 & (0.354+0.354j) & \end{pmatrix}$$

1.5 Resolving non-adjacent interactions

Interactions between non-adjacent qubits can be resolved by QubitCircuit to a series of adjacent interactions, which is useful for systems such as spin chain models.

In [60]:
```
qc8 = QubitCircuit(3)
qc8.add_gate("CNOT", 2, 0)
qc8.png
```

Out[60]:



In [61]:
```
U8 = gate_sequence_product(qc8.unitary_matrix())
U8
```

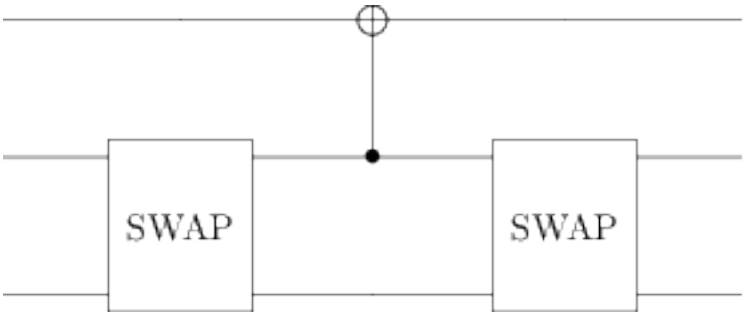
Out[61]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

In [62]:
```
qc9 = qc8.adjacent_gates()
qc9.png
```

Out[62]:



In [63]:
```
U9 = gate_sequence_product(qc9.unitary_matrix())
U9
```

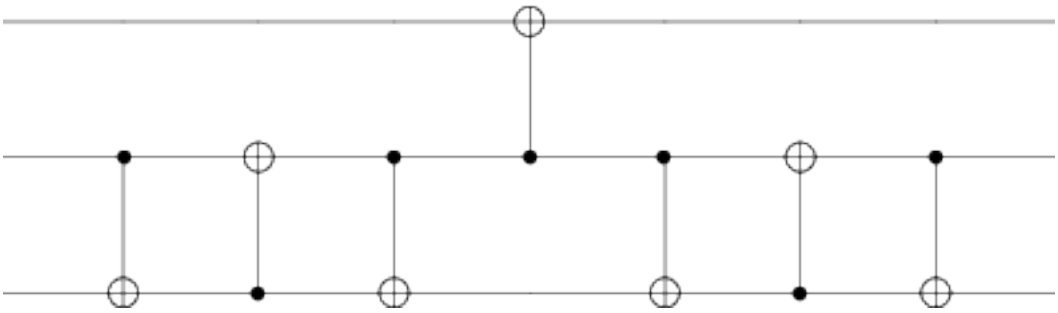
Out[63]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

In [64]:
```
qc10 = qc9.resolve_gates("CNOT")
qc10.png
```

Out[64]:



In [65]:
```
U10 = gate_sequence_product(qc10.unitary_matrix())
U10
```

Out[65]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

1.6 Software versions

In [66]:
```
from qutip.ipynbtools import version_table
version_table()
```

Out[66]: <IPython.core.display.HTML at 0x7f1c5e1a4f98>