


# Quantum Gate Decomposition for QuTiP

```
In [1]: %matplotlib inline
from IPython.display import Image
import cmath
import scipy
import pprint
from numpy import pi
import itertools
import numpy as np
import matplotlib.pyplot as plt
from qutip import Qobj, average_gate_fidelity, rand_unitary, tensor, identity, s
from qutip_qip.operations import *
from qutip_qip.circuit import QubitCircuit, Gate
from qutip_qip.decompose.decompose_single_qubit_gate import decompose_one_qubit_
from qutip_qip.decompose.decompose_general_qubit_gate import _lastq_target_ancil
from qutip_qip.decompose._utility import check_gate, _binary_sequence, _gray_coc
```

## Decomposing one qubit gates

```
In [2]: num_qubits = 1
input_gate = rand_unitary(2**num_qubits, dims=[[2] * num_qubits] * 2)
qc1 = QubitCircuit(num_qubits, reverse_states=False)
gate_list1 = decompose_general_qubit_gate(input_gate, num_qubits)
# the output of general function is a tuple of a list of gates and number of qut
# ancilla qubits for higher number of qubits
qc1.add_gates(gate_list1[0])
qc1.png
```

Out[2]: 

```
In [3]: qc1.compute_unitary()
```

Out[3]: Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = False

$$\begin{pmatrix} (0.587 + 0.655j) & (-0.470 + 0.081j) \\ (0.268 + 0.394j) & (0.879 - 0.033j) \end{pmatrix}$$

```
In [4]: input_gate
```

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

$$\begin{pmatrix} (0.587 + 0.655j) & (-0.470 + 0.081j) \\ (0.268 + 0.394j) & (0.879 - 0.033j) \end{pmatrix}$$

**\*\*NOTE:\*\***

It is currently not possible to verify the full output of the general method after evaluating the gates in a quantum circuit. This is because the final output is a mix of nested dictionaries and lists which makes it difficult to iterate over the entire output in order to add them to a quantum circuit.

## Decomposing two qubit gates without ancilla qubits

As discussed in the blogpost, the full output of general decomposition function is incorrect. We describe some of the special cases below and don't compare the input of first code block below to the input gate.

```
In [5]: num_qubits = 2
input_gate = rand_unitary(2**num_qubits, dims=[[2] * num_qubits] * 2)
gate_list2 = decompose_general_qubit_gate(input_gate, num_qubits)
```

## Decomposition to all single-qubit gates and CNOT

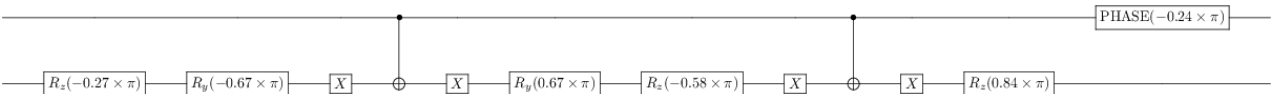
When the array is described with last qubit as target

Because the default output of two-level array decomposition is to output the matrices in reverse, we know the first object in this output describes a two-level array matrix with last qubit as target.

```
In [6]: two_level_arrays_non_expand = _decompose_to_two_level_arrays(input_gate, num_qubits)
print(two_level_arrays_non_expand[0])
```

```
[[2, 3], Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = False
Qobj data =
[[-0.42156275-0.27005623j -0.83521174+0.22754303j]
 [ 0.27601149-0.82047068j  0.24493324+0.43663856j]]]
```

```
In [7]: qc2_1 = QubitCircuit(num_qubits, reverse_states=False)
gate_list2_1 = _two_qubit_last_target(two_level_arrays_non_expand[0][1].full(),
qc2_1.add_gates(gate_list2_1)
qc2_1.png
```

```
Out[7]: 
```

```
In [8]: qc2_1.compute_unitary()
```

```
Out[8]: Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = False
```

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & (-0.422 - 0.270j) & (-0.835 + 0.228j) \\ 0.0 & 0.0 & (0.276 - 0.820j) & (0.245 + 0.437j) \end{pmatrix}$$

```
In [9]:
```

```
_decompose_to_two_level_arrays(input_gate, num_qubits, expand = True)[0]
```

Out[9]: Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = False

$$\begin{pmatrix} 1.000 & -5.551 \times 10^{-17} & 5.551 \times 10^{-17} & -1.665 \times 10^{-16} \\ 0.0 & 1.0 & -1.110 \times 10^{-16} & 1.110 \times 10^{-16} \\ 0.0 & 5.464 \times 10^{-17} & (-0.422 - 0.270j) & (-0.835 + 0.228j) \\ 0.0 & 0.0 & (0.276 - 0.820j) & (0.245 + 0.437j) \end{pmatrix}$$

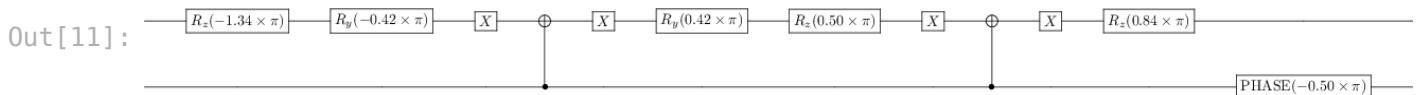
When the array is described with last qubit as target

As shown in this blogpost, the next object in above two-level output describes a two-level gate with control and target qubits flipped.

```
In [10]: two_level_arrays_non_expand[1]
```

Out[10]: [[1, 3],  
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True  
Qobj data =  
[[ 0.79051013-1.38777878e-17j -0.29139905+5.38683883e-01j]  
[-0.29139905-5.38683883e-01j -0.79051013-1.38777878e-17j]]]

```
In [11]: qc2_2 = QubitCircuit(num_qubits, reverse_states=False)
gate_list2_2 = _two_qubit_last_target(two_level_arrays_non_expand[1][1].full(),
qc2_2.add_gates(gate_list2_2)
qc2_2.png
```



```
In [12]: qc2_2.compute_unitary()
```

Out[12]: Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 1.000 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.791 & 0.0 & (-0.291 + 0.539j) \\ 0.0 & 0.0 & 1.000 & 0.0 \\ 0.0 & (-0.291 - 0.539j) & 0.0 & -0.791 \end{pmatrix}$$

```
In [13]: _decompose_to_two_level_arrays(input_gate, num_qubits, expand = True)[1]
```

Out[13]: Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.791 & 0.0 & (-0.291 + 0.539j) \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & (-0.291 - 0.539j) & 0.0 & -0.791 \end{pmatrix}$$

## Decomposing three qubit gates without ancilla qubits

Suppose we want to decompose an arbitrary 3 qubit quantum gate into a decomposition described by single-qubit gates, two-qubit CNOT and 3-qubit Toffoli. The first step is to decompose the input into a composition of two-level gates (function returns the output in reversed order). We describe below how to decompose a couple of these two-level unitary gates.

### Last qubit is target with other two as controls

Again similar to 2-qubit case, we know the first object in two-level array output describes a two-level gate with last qubit as target and first two as controls.

```
In [14]: num_qubits = 3
input_gate = rand_unitary(2**num_qubits, dims=[[2] * num_qubits] * 2)
#gate_list3 = decompose_general_qubit_gate(input_gate, num_qubits)
```

```
In [15]: two_level_arrays_non_expand = _decompose_to_two_level_arrays(input_gate, num_qubits)
print(two_level_arrays_non_expand[0])
```

```
[[6, 7], Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = False
Qobj data =
[[ 0.08921209+0.70272741j -0.43828278+0.55328437j]
 [-0.65764808-0.25634816j  0.27774734+0.65164487j]]]
```

```
In [16]: qc3_1 = QubitCircuit(num_qubits, reverse_states=False)
gate_list3_1 = _threeq_last_target(two_level_arrays_non_expand[0][1].full())
qc3_1.add_gates(gate_list3_1)
qc3_1.png
```

```
Out[16]: 
```

```
In [17]: qc3_1.compute_unitary()
```

```
Out[17]: Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = (8, 8), type = oper, isherm = False
```

$$\begin{pmatrix} 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & (0.089 + 0.703j) & (-0.438 + 0.553j) \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & (-0.658 - 0.256j) & (0.278 + 0.652j) \end{pmatrix}$$

### Comparison of output circuit to input array

```
In [18]: _decompose_to_two_level_arrays(input_gate, num_qubits, expand = True)[0]
```

```
Out[18]: Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = (8, 8), type = oper, isherm = False
```

$$\begin{pmatrix} 1.0 & -1.023 \times 10^{-16} & 6.245 \times 10^{-17} & -7.633 \times 10^{-17} & 6.939 \times 10^{-18} \\ -2.444 \times 10^{-17} & 1.000 & -8.674 \times 10^{-17} & -4.510 \times 10^{-17} & -2.567 \times 10^{-1} \\ -3.074 \times 10^{-18} & -2.614 \times 10^{-17} & 1.0 & 5.551 \times 10^{-17} & -9.021 \times 10^{-1} \\ 9.593 \times 10^{-19} & -1.724 \times 10^{-17} & 3.772 \times 10^{-18} & 1.0 & 9.714 \times 10^{-17} \\ 2.051 \times 10^{-17} & -4.039 \times 10^{-17} & -1.488 \times 10^{-17} & -2.603 \times 10^{-17} & 1.0 \\ -1.123 \times 10^{-17} & 1.353 \times 10^{-19} & -2.870 \times 10^{-17} & 3.852 \times 10^{-18} & 6.045 \times 10^{-18} \\ -5.278 \times 10^{-18} & -2.307 \times 10^{-17} & 4.662 \times 10^{-17} & -1.208 \times 10^{-17} & 1.473 \times 10^{-17} \\ 2.512 \times 10^{-18} & -3.690 \times 10^{-18} & -8.974 \times 10^{-18} & -4.743 \times 10^{-18} & -1.217 \times 10^{-1} \end{pmatrix}$$

## First qubit is target and other two are controls

Using above output of two-level gate arrays, we need to figure out which of the arrays describes a quantum gate with first qubit as target and other two as control. This is done by iterating over the entire output and checking the targets in gray-code ordering output.

```
In [19]: ind_for_0_target = _find_index_for_firstq_target(two_level_arrays_non_expand, 3)
print("Two-level output index for object controlled on first qubit is", ind_for_0_target)
```

```
Two-level output index for object controlled on first qubit is 11
```

```
In [20]: two_level_arrays_non_expand[ind_for_0_target]
```

```
Out[20]: [[2, 6],
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True
Qobj data =
[[ 0.74509472-1.49393079e-17j -0.59427063-3.02780912e-01j]
 [-0.59427063+3.02780912e-01j -0.74509472-1.49393079e-17j]]]
```

```
In [21]: first_target_qubit_array = two_level_arrays_non_expand[ind_for_0_target]
# first_target_qubit_array gives position of the array of interest in two-level
first_target_qubit_array_index = first_target_qubit_array[0]

# Check if the gray code will add Pauli X gates to make control values = 1
_paulix_for_control_0(_gray_code_gate_info(first_target_qubit_array_index[0], first_target_qubit_array_index[1]))
```

```
Out[21]: [Gate(X, targets=[2], controls=None, classical controls=None, control_value=None),
{'controls =': [1, 2], 'control_value =': ['1', '1'], 'targets =': [0]},
Gate(X, targets=[2], controls=None, classical controls=None, control_value=None)]
```

This shows in addition to switching control and target qubits in `_threeq_last_target` an additional gate has to be added to make the gate controlled on 0 of qubit 2 to controlled on 1. This increases the total number of gates compared to the case when last qubit is target.

```
In [22]: qc3_2 = QubitCircuit(num_qubits, reverse_states=False)
gate_list3_2 = _threeq_first_target(two_level_arrays_non_expand[ind_for_0_target])
qc3_2.add_gates(gate_list3_2)
qc3_2.png

# The output below has 2 extra gates compared to the case when two-level array f
```

```
Out[22]: 
```

### Comparison of output circuit to input array

```
In [23]: qc3_2.compute_unitary()
```

```
Out[23]: Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = (8, 8), type = oper, isherm = True
```

$$\begin{pmatrix} 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.745 & 0.0 & 0.0 & 0.0 & (-0.594 - 0.303j) & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & (-0.594 + 0.303j) & 0.0 & 0.0 & 0.0 & -0.745 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

```
In [24]: _decompose_to_two_level_arrays(input_gate, num_qubits, expand = True)[ind_for_0_
```

```
Out[24]: Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = (8, 8), type = oper, isherm = True
```

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.745 & 0.0 & 0.0 & 0.0 & (-0.594 - 0.303j) & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & (-0.594 + 0.303j) & 0.0 & 0.0 & 0.0 & -0.745 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

## Decomposing four qubit gates with ancilla qubits

**Note :** The output is not correctly figured out because the gate decomposing toffoli gates/targets has an incorrect condition. Instead of total number of qubits, the condition should be number of control qubits > 3.

```
In [25]: num_qubits = 4
```

```
input_gate = rand_unitary(2**num_qubits, dims=[[2] * num_qubits] * 2)
two_level_arrays_non_expand = _decompose_to_two_level_arrays(input_gate, num_qubits)
```

```
In [26]: gate_num_qubits = _lastq_target_ancilla(two_level_arrays_non_expand, num_qubits)
total_num = gate_num_qubits[1]
gate_list = gate_num_qubits[0]
```

```
In [27]: qc1 = QubitCircuit(total_num, reverse_states=False)
qc1.add_gates(gate_list)
qc1.png
```

```
Out[27]: 
```

```
In [28]: qc1.compute_unitary()
```

```
Out[28]: Quantum object: dims = [[2, 2, 2, 2, 2], [2, 2, 2, 2, 2]], shape = (32, 32), type = oper, isherm = False
```

```

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  (-0.359 - 0.781j) ...  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  (-0.359 - 0.781j)  (-0.361 + 0.
0.0  0.0  0.0  0.0      0.0      ...  0.0  (-0.462 + 0.220j)  (-0.617 - 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      0.0      ...  0.0      0.0      0.0

```