```python
import math
import qiskit
import matplotlib
import numpy as np
import time
import copy
from qiskit import IBMQ, BasicAer, Aer
from qiskit import ClassicalRegister, QuantumRegister, QuantumCircuit
from qiskit.providers.ibmq import least_busy
from qiskit.tools.visualization import plot_histogram
from qiskit.visualization import plot_state_city
from qiskit.visualization import plot_bloch_multivector
from qiskit.tools.monitor import job_monitor
from qiskit.providers.jobstatus import JobStatus

from qiskit.quantum_info import state_fidelity
from qiskit.providers.aer import noise

# Tomography functions
from qiskit.ignis.verification.tomography import state_tomography_circuits, StateTomographyFitter
import qiskit.ignis.mitigation.measurement as mc

qiskit.IBMQ.load_accounts()

# get different backends
simulator = qiskit.providers.ibmq.least_busy(qiskit.IBMQ.backends(simulator=True))
least_busy = qiskit.providers.ibmq.least_busy(qiskit.IBMQ.backends(simulator=False))
melbourne = IBMQ.get_backend('ibmq_16_melbourne')

# melbourne noise modeling
gate_times_melbourne = [
        ('u1', None, 0), ('u2', None, 100), ('u3', None, 200),
        ('cx', [1, 0], 678), ('cx', [1, 2], 547), ('cx', [2, 3], 721),
        ('cx', [4, 3], 733), ('cx', [4, 10], 721), ('cx', [5, 4], 800),
        ('cx', [5, 6], 800), ('cx', [5, 9], 895), ('cx', [6, 8], 895),
        ('cx', [7, 8], 640), ('cx', [9, 8], 895), ('cx', [9, 10], 800),
        ('cx', [11, 10], 721), ('cx', [11, 3], 634), ('cx', [12, 2], 773),
        ('cx', [13, 1], 2286), ('cx', [13, 12], 1504), ('cx', [], 800)
    ]
noise_model_melbourne = noise.device.basic_device_noise_model(melbourne.properties(), gate_times=gate_times_melbourne)
basis_gates_melbourne = noise_model_melbourne.basis_gates
coupling_map_melbourne = melbourne.configuration().coupling_map

# helpers

# alternative design of controlled G(p)
def CGalt(circuit, qregister, qbit: int, ctrlbit: int, p: float):
    thetap = t2tp(p2theta(p))
    circuit.u3(-thetap, 0, 0, qregister[qbit])
    circuit.cx(qregister[ctrlbit], qregister[qbit])
    circuit.u3(thetap, 0, 0, qregister[qbit])

# B(p) without considering physical constraints (CNOT not reversible)
def Bdirect(circuit, qregister, qbit: int, ctrlbit: int, p: float):
    CGalt(circuit, qregister, qbit, ctrlbit, p)
    circuit.cx(qregister[qbit], qregister[ctrlbit])

# get theta angle from p for the U3 rotation inside CG(p)
def p2theta(p: float):
    return math.acos(math.sqrt(p)) * 2

# get theta' angle from theta for the U3 rotation inside CGalt(p)
def t2tp(theta:float):
    return math.asin(math.cos(theta / 2))
```

```python
# split a list into wanted_parts smaller lists with same number of elements (+/- 1)
# https://stackoverflow.com/a/752562
def split_list(alist, wanted_parts=1):
    length = len(alist)
    return [ alist[i*length // wanted_parts: (i+1)*length // wanted_parts]
             for i in range(wanted_parts) ]
```