In [1]:
```
import numpy as np
from qiskit.circuit import ParameterVector, QuantumCircuit
from qiskit.primitives import StatevectorEstimator, StatevectorSampler
from qiskit.quantum_info import SparsePauliOp
from scipy.optimize import minimize
```

In [2]:
```
op = SparsePauliOp("Z")
op.to_matrix()
```

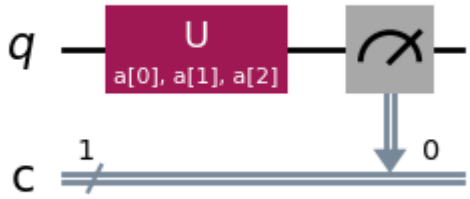
Out[2]:
```
array([[ 1.+0.j,  0.+0.j],
       [ 0.+0.j, -1.+0.j]])
```

In [3]:
```
# compute eigenvalues with numpy
result = np.linalg.eigh(op.to_matrix())
print("Eigenvalues:", result.eigenvalues)
```

```
Eigenvalues: [-1.  1.]
```

In [4]:
```
# define a variational form
param = ParameterVector("a", 3)
qc = QuantumCircuit(1, 1)
qc.u(param[0], param[1], param[2], 0)
qc_estimator = qc.copy()
qc.measure(0, 0)
qc.draw("mpl")
```

Out[4]:



In [5]:
```
sampler = StatevectorSampler()
estimator = StatevectorEstimator()
```

In [6]:
```
# compute counts of bitstrings with random parameter values by Sampler
result = sampler.run([(qc, [1, 2, 3])]).result()
counts = result[0].data.c.get_counts()
counts
```

Out[6]:
```
{'0': 770, '1': 254}
```

In [7]:
```
# compute the expectation value of Z based on the counts
(counts.get("0", 0) - counts.get("1", 0)) / sum(counts.values())
```

Out[7]:
```
0.50390625
```

In [8]:
```
result = estimator.run([(qc_estimator, op, [1, 2, 3])]).result()
result[0].data.evs
```

Out[8]:
```
array(0.54030231)
```

In [9]:
```
# define a cost function to look for the minimum eigenvalue of Z
def cost(x):
    result = sampler.run([(qc, x)]).result()
    counts = result[0].data.c.get_counts()
    expval = (counts.get("0", 0) - counts.get("1", 0)) / sum(counts.values())
    # the following line shows the trajectory of the optimization
    print(expval, counts)
    return expval
```

In [10]:

```
# minimize the cost function with scipy's minimize
min_result = minimize(cost, [0, 0, 0], method="COBYLA", tol=1e-8)
min_result
```

```
1.0 {'0': 1024}
0.521484375 {'0': 779, '1': 245}
0.529296875 {'0': 783, '1': 241}
0.474609375 {'0': 755, '1': 269}
-0.416015625 {'1': 725, '0': 299}
-0.984375 {'1': 1016, '0': 8}
-0.791015625 {'1': 917, '0': 107}
-0.98046875 {'1': 1014, '0': 10}
-0.953125 {'1': 1000, '0': 24}
-0.787109375 {'0': 109, '1': 915}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-0.998046875 {'1': 1023, '0': 1}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-0.998046875 {'1': 1023, '0': 1}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-0.998046875 {'1': 1023, '0': 1}
-0.998046875 {'1': 1023, '0': 1}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-0.998046875 {'1': 1023, '0': 1}
-0.998046875 {'1': 1023, '0': 1}
```

```
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-1.0 {'1': 1024}
-1.0 {'1': 1024}
-0.998046875 {'1': 1023, '0': 1}
-0.998046875 {'1': 1023, '0': 1}
```

Out[10]:
```
 message: Optimization terminated successfully.
 success: True
  status: 1
     fun: -0.998046875
       x: [ 3.109e+00 -2.299e-02  9.282e-01]
    nfev: 56
   maxcv: 0.0
```

In [11]:
```
1 # check counts of bitstrings with the optimal parameters
2 result = sampler.run([(qc, min_result.x)]).result()
3 result[0].data.c.get_counts()
```

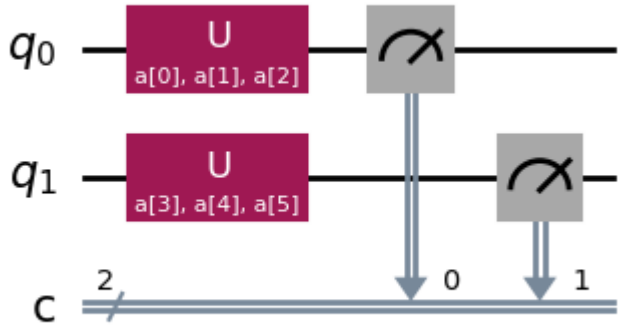
Out[11]:
```
{'1': 1023, '0': 1}
```

In [12]:
```
1 z2 = SparsePauliOp("ZZ")
2 print(z2)
3 print(z2.to_matrix())
```

```
SparsePauliOp(['ZZ'],
              coeffs=[1.+0.j])
[[ 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 -1.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j  1.+0.j]]
```

In [13]:
```
1 # define a variational form
2 param = ParameterVector("a", 6)
3 qc = QuantumCircuit(2, 2)
4 qc.u(param[0], param[1], param[2], 0)
5 qc.u(param[3], param[4], param[5], 1)
6 qc_estimator = qc.copy()
7 qc.measure([0, 1], [0, 1])
8 qc.draw("mpl")
```

Out[13]:



In [14]:
```
sampler = StatevectorSampler()    #STATE OF THE SYSTEM
estimator = StatevectorEstimator() #EXPECTATION VAUE OF AN OPERATOR
```

In [15]:
```
# compute counts of bitstrings with random parameter values by Sampler
result = sampler.run([(qc, [1, 2, 3, 4, 5, 6])]).result()
counts = result[0].data.c.get_counts()
counts
```

Out[15]:
```
{'10': 678, '00': 125, '01': 41, '11': 180}
```

In [16]:
```
# compute the expectation value of ZZ based on the counts
(
    counts.get("00", 0)
    - counts.get("01", 0)
    - counts.get("10", 0)
    + counts.get("11", 0)
) / sum(counts.values())
```

Out[16]:
```
-0.404296875
```

In [17]:
```
# verify the expectation value of ZZ with Estimator
result = estimator.run([(qc_estimator, z2, [1, 2, 3, 4, 5, 6])]).result()
result[0].data.evs
```

Out[17]:
```
array(-0.35316516)
```

In [18]:
```python
# define a cost function to look for the minimum eigenvalue of ZZ
def cost(x):
    result = sampler.run([(qc, x)]).result()
    counts = result[0].data.c.get_counts()
    expval = (
        counts.get("00", 0)
        - counts.get("01", 0)
        - counts.get("10", 0)
        + counts.get("11", 0)
    ) / sum(counts.values())
    print(expval, counts)
    return expval
```

In [19]:
```python
# minimize the cost function with scipy's minimize
min_result = minimize(cost, [0, 0, 0, 0, 0, 0], method="COBYLA", tol=1e-8)
min_result
```

```
1.0 {'00': 1024}
0.53515625 {'00': 786, '01': 238}
0.47265625 {'00': 754, '01': 270}
0.53515625 {'00': 786, '01': 238}
0.314453125 {'00': 624, '10': 175, '01': 176, '11': 49}
0.33984375 {'10': 162, '00': 632, '01': 176, '11': 54}
0.365234375 {'00': 646, '01': 162, '10': 163, '11': 53}
-0.091796875 {'00': 220, '10': 137, '01': 422, '11': 245}
0.154296875 {'11': 583, '01': 413, '10': 20, '00': 8}
0.013671875 {'01': 446, '11': 435, '00': 84, '10': 59}
-0.02734375 {'10': 111, '00': 216, '01': 415, '11': 282}
0.03125 {'00': 199, '11': 329, '01': 278, '10': 218}
-0.09375 {'01': 431, '11': 242, '00': 222, '10': 129}
-0.26953125 {'00': 268, '01': 600, '11': 106, '10': 50}
-0.3203125 {'01': 645, '11': 82, '00': 266, '10': 31}
-0.373046875 {'00': 256, '01': 688, '10': 15, '11': 65}
-0.548828125 {'01': 790, '00': 227, '11': 4, '10': 3}
-0.736328125 {'01': 888, '00': 132, '10': 1, '11': 3}
-0.74609375 {'11': 64, '01': 889, '00': 66, '10': 5}
```

In [20]:
```
# check counts of bitstrings with the optimal parameters
result = sampler.run([(qc, min_result.x)]).result()
result[0].data.c.get_counts()
```

Out[20]:
```
{'01': 1024}
```