k

March 8, 2023

0.0.1 Import library

[ ]:
```
import numpy as np
import qiskit
from qiskit import IBMQ

# Loading your IBM Quantum account(s)
provider = IBMQ.load_account()
```

```
ibmqfactory.load_account:WARNING:2023-03-08 09:21:11,626: Credentials are
already in use. The existing account in the session will be replaced.
```

[ ]:
```
%matplotlib inline
```

0.1 Basic Quantum Circuit 1

[ ]:
```
qc = qiskit.QuantumCircuit(2,2)
qc.h(0)          # Hadamard Gate
qc.cx(0,1)       # Controled Not
# qc.h(0)
qc.measure(range(2), range(2))  # Measurement on classical bit
qc.draw()
```

```
---------------------------------------------------------------------------
MissingOptionalLibraryError               Traceback (most recent call last)
Cell In[7], line 6
      4 # qc.h(0)
      5 qc.measure(range(2), range(2))  # Measurement on classical bit
----> 6 qc.draw()

File ~/Program1/QuantumLearning/.qenv/lib/python3.10/site-packages/qiskit/
  ↪circuit/quantumcircuit.py:1896, in QuantumCircuit.draw(self, output, scale,␣
  ↪filename, style, interactive, plot_barriers, reverse_bits, justify,␣
  ↪vertical_compression, idle_wires, with_layout, fold, ax, initial_state,␣
  ↪cregbundle, wire_order)
   1893 # pylint: disable=cyclic-import
   1894 from qiskit.visualization import circuit_drawer
-> 1896 return circuit_drawer(
   1897     self,
   1898     scale=scale,
   1899     filename=filename,
   1900     style=style,
   1901     output=output,
   1902     interactive=interactive,
   1903     plot_barriers=plot_barriers,
   1904     reverse_bits=reverse_bits,
   1905     justify=justify,
   1906     vertical_compression=vertical_compression,
   1907     idle_wires=idle_wires,
   1908     with_layout=with_layout,
   1909     fold=fold,
   1910     ax=ax,
   1911     initial_state=initial_state,
   1912     cregbundle=cregbundle,
   1913     wire_order=wire_order,
   1914 )

File ~/Program1/QuantumLearning/.qenv/lib/python3.10/site-packages/qiskit/visualization/circuit/circuit_visualization.py:274, in circuit_drawer(circuit scale, filename, style, output, interactive, plot_barriers, reverse_bits, justify, vertical_compression, idle_wires, with_layout, fold, ax, initial_state, cregbundle, wire_order)
    259     return _generate_latex_source(
    260         circuit,
    261         filename=filename,
   (…)
    271         wire_order=wire_order,
    272     )
    273 elif output == "mpl":
--> 274     image = _matplotlib_circuit_drawer(
    275         circuit,
    276         scale=scale,
    277         filename=filename,
    278         style=style,
    279         plot_barriers=plot_barriers,
    280         reverse_bits=reverse_bits,
    281         justify=justify,
    282         idle_wires=idle_wires,
    283         with_layout=with_layout,
    284         fold=fold,
    285         ax=ax,
    286         initial_state=initial_state,
    287         cregbundle=cregbundle,
    288         wire_order=wire_order,
    289     )
    290 else:
    291     raise VisualizationError(
    292         "Invalid output type %s selected. The only valid choices "
    293         "are text, latex, latex_source, and mpl" % output
    294     )

File ~/Program1/QuantumLearning/.qenv/lib/python3.10/site-packages/qiskit/
 ↪visualization/circuit/circuit_visualization.py:653, in␣
 ↪_matplotlib_circuit_drawer(circuit, scale, filename, style, plot_barriers,␣
 ↪reverse_bits, justify, idle_wires, with_layout, fold, ax, initial_state,␣
 ↪cregbundle, wire_order)
    650 if fold is None:
    651     fold = 25
--> 653 qcd = _matplotlib.MatplotlibDrawer(
    654     qubits,
    655     clbits,
    656     nodes,
    657     scale=scale,
    658     style=style,
    659     reverse_bits=reverse_bits,
    660     plot_barriers=plot_barriers,
    661     layout=None,
    662     fold=fold,
    663     ax=ax,
    664     initial_state=initial_state,
    665     cregbundle=cregbundle,
    666     global_phase=None,
    667     calibrations=None,
    668     qregs=None,
    669     cregs=None,
    670     with_layout=with_layout,
    671     circuit=circuit,
    672 )
    673 return qcd.draw(filename)

File ~/Program1/QuantumLearning/.qenv/lib/python3.10/site-packages/qiskit/utils
 ↪classtools.py:111, in _WrappedMethod.__get__.<locals>.out(*args, **kwargs)
    108 @functools.wraps(method)
    109 def out(*args, **kwargs):
    110     for callback in self._before:
--> 111         callback.__get__(obj, objtype)(*args, **kwargs)
    112     retval = method(*args, **kwargs)
    113     for callback in self._after:

File ~/Program1/QuantumLearning/.qenv/lib/python3.10/site-packages/qiskit/utils
 ↪lazy_tester.py:39, in _RequireNow.__call__(self, *_args, **_kwargs)
     38 def __call__(self, *_args, **_kwargs):
---> 39     self._tester.require_now(self._feature)
```

```
File ~/Program1/QuantumLearning/.qenv/lib/python3.10/site-packages/qiskit/utils
  ↪lazy_tester.py:223, in LazyDependencyManager.require_now(self, feature)
    221 if self:
    222     return
--> 223 raise MissingOptionalLibraryError(
    224     libname=self._name, name=feature, pip_install=self._install,␣
  ↪msg=self._msg
    225 )

MissingOptionalLibraryError: "The 'pylatexenc' library is required to use␣
  ↪'MatplotlibDrawer'. You can install it with 'pip install pylatexenc'."
```

[ ]:
```
backend_simulator = Aer.get_backend('qasm_simulator')
job_simulator = execute(qc, backend_simulator, shots=1024)
```

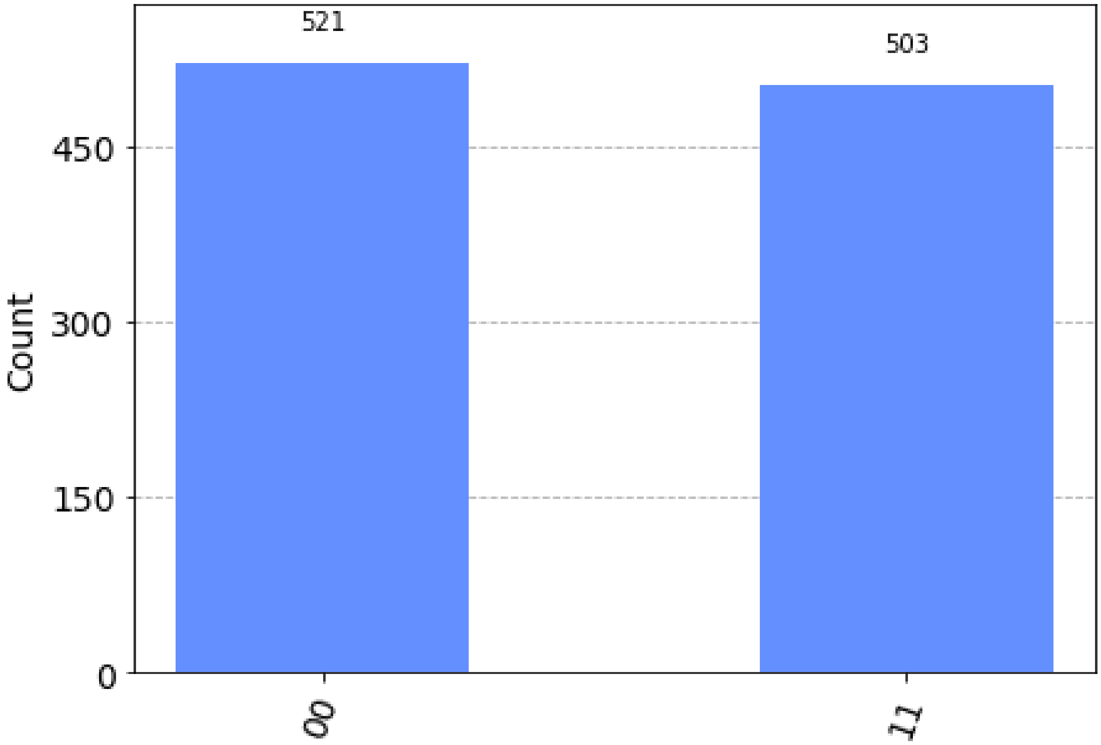
[ ]:
```
result_simulator = job_simulator.result()
```

[ ]:
```
count = result_simulator.get_counts(qc)
count
```

[ ]:
```
{'11': 503, '00': 521}
```

[ ]:
```
plot_histogram(count)
```

[ ]:



0.1.1 Execute on real Hardware

```
[ ]: provider.backends()
```

```
[ ]: [<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQBackend('ibmq_lima') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQBackend('ibmq_belem') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQBackend('ibmq_quito') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQSimulator('simulator_statevector') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQSimulator('simulator_mps') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQSimulator('simulator_extended_stabilizer') from IBMQ(hub='ibm-q',
     group='open', project='main')>,
      <IBMQSimulator('simulator_stabilizer') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQBackend('ibm_nairobi') from IBMQ(hub='ibm-q', group='open',
     project='main')>,
      <IBMQBackend('ibm_oslo') from IBMQ(hub='ibm-q', group='open', project='main')>]
```

```
[ ]: backend = provider.get_backend('ibmq_belem')
```

```
[ ]: # job_simulator = execute(qc, backend, shots=1024)
     # result_simulator = job_simulator.result()
```

```
[ ]: # count = result_simulator.get_counts(qc)
     # count
```

0.1.2 Quantum State Visualization

```
[ ]: from qiskit.visualization import plot_state_qsphere, plot_bloch_multivector
```

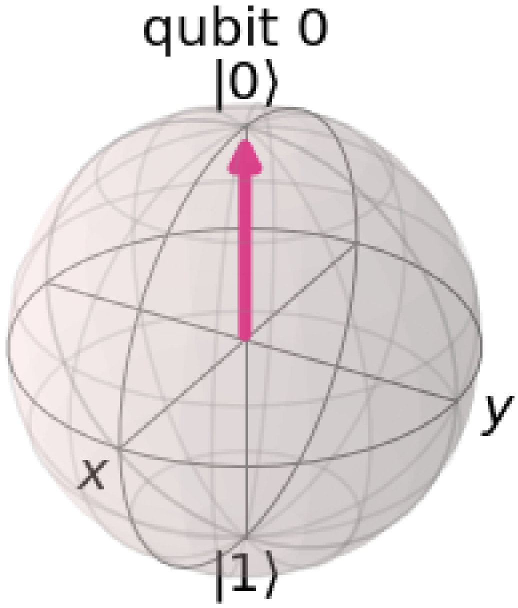
```
[ ]: qc=qiskit.QuantumCircuit(1)
     statevector_simulator = qiskit.Aer.get_backend("statevector_simulator")
     result = qiskit.execute(qc,statevector_simulator).result()
     statevector_result = result.get_statevector(qc)
     statevector_result
```

```
Statevector([1.+0.j, 0.+0.j],
            dims=(2,))
```

[ ]:

```
plot_bloch_multivector(statevector_result)
```

[ ]:



# Parameterized quantum circuits

[ ]:

```
from qiskit.circuit.library import ZZFeatureMap
qc_zz = ZZFeatureMap(3, reps=1, insert_barriers=True)
decomposed_circuit = qc_zz.decompose()
```

0.2 ZZFeatureMapCircuit Expression

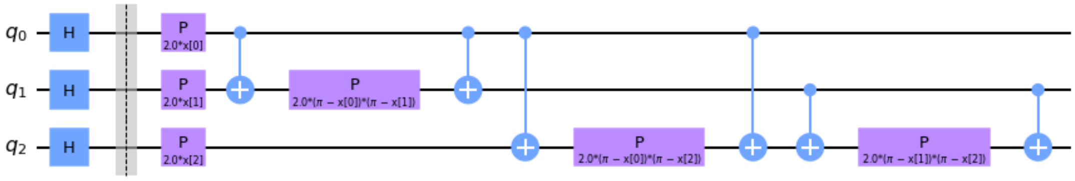
$$u_{\Phi(x)} = \prod_{d} U_{\Phi(x)} H^{\otimes n}, U_{\Phi(x)} = \exp\left(i \sum_{S\subseteq[n]} \phi_s(x) \prod_{k\in S} P_i \right)$$

$$\phi_s : x \mapsto \begin{cases} x_i & \textbf{if } s = \{i\} \\ (\pi - x_i)(\pi - x_j) & \textbf{if } s = \{i, j\} \end{cases}$$

[ ]:

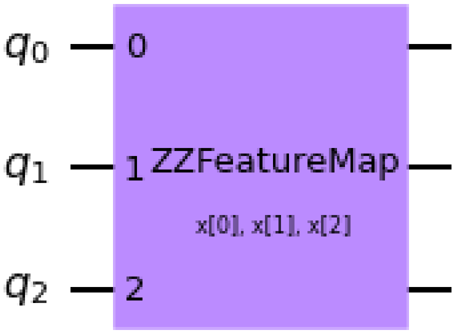
```
decomposed_circuit.draw()
```

[ ]:


q_0
q_1
q_2
H
H
H
P
2.0*x[0]
P
2.0*x[1]
P
2.0*x[2]
P
2.0*(π − x[0])*(π − x[1])
P
2.0*(π − x[0])*(π − x[2])
P
2.0*(π − x[1])*(π − x[2])

```
[ ]: qc_zz.draw()
```

[ ]:


q_0
q_1
q_2
0
1
2
ZZFeatureMap
x[0], x[1], x[2]

```
[ ]:
```