Pomiary_XYZ

May 31, 2020

[1]:
```
%matplotlib inline
# Importowanie standardowej biblioteki Qiskit i konfiguracja konta
from qiskit import QuantumCircuit,ClassicalRegister, QuantumRegister, execute,␣
 ↪Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from math import pi
# Określenie konta IBM Q
provider = IBMQ.load_account()
```

[2]:
```
# wybór symulatora kwantowego ( lub procesora)
backend = Aer.get_backend('qasm_simulator')
n=1 # ustalanie liczby rejestrów kwantowych i klasycznych
seria=2048 # liczba prób w serii pomiarów danego typu
```

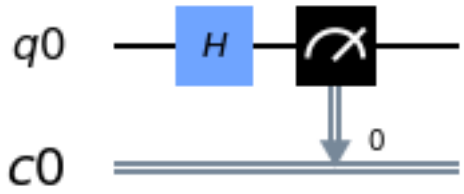
[3]:
```
#Przykład pomiaru typu X ( inaczej sigmaX)
nx =n  # Liczba kubitów i bitów
qx = QuantumRegister(nx) # Rejestr kwantowy
cx = ClassicalRegister(nx) # Rejestr klasyczny

circuitX = QuantumCircuit(qx, cx) # Algorytm kwantowy - kwantowy obwód
circuitX.h(qx[0]) # Operacja X realizowana na kubicie qx[0]
circuitX.measure(qx[0], cx[0]) # Sprawdzenie stanów kubitów - kwantowy pomiar
```

[3]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7fe048f5b190>
```

[4]:
```
circuitX.draw(output='mpl') # Rysowanie obwodu kwantowego pomiaru typu X
```

[4]:

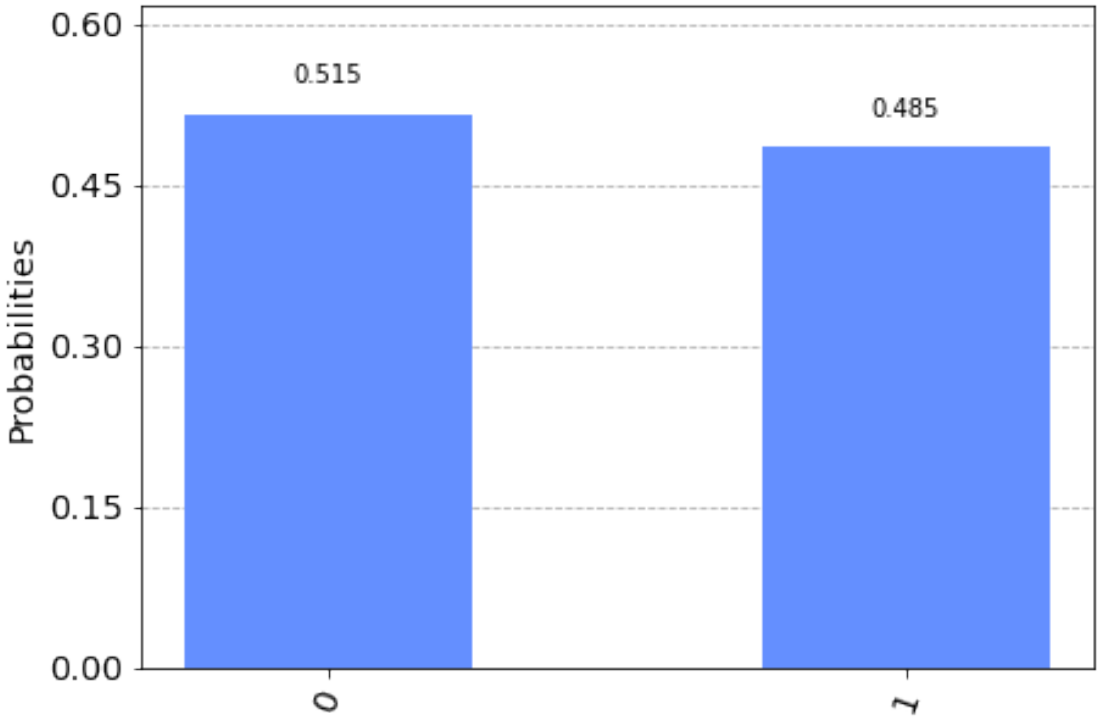


[5]:
```
# Wykonanie obliczeń kwantowych
job_simX = execute(circuitX, backend, shots=seria)
sim_resultX = job_simX.result()
# Liczbowa prezentacja wyników wyników pomiaru typu X
print(sim_resultX.get_counts(circuitX))
```

```
{'1': 994, '0': 1054}
```

[6]:
```
# Graficzna prezentacja wyników pomiaru X
plot_histogram(sim_resultX.get_counts(circuitX))
```

[6]:



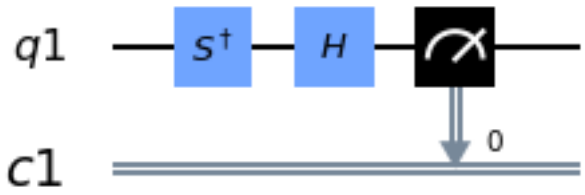
[7]:
```
#Przykład pomiaru typu Y ( inaczej sigmaY)
ny = n  # Liczba kubitów i bitów
qy = QuantumRegister(ny) # Rejestr kwantowy
cy = ClassicalRegister(ny) # Rejestr klasyczny

circuitY = QuantumCircuit(qy, cy) # Algorytm kwantowy - kwantowy obwód
circuitY.sdg(qy[0]) # Operacja X realizowana na kubicie qy[0]
circuitY.h(qy[0]) # Operacja X realizowana na kubicie qy[0]
circuitY.measure(qy[0], cy[0]) # Sprawdzenie stanów kubitów - kwantowy pomiar
```

[7]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7fe04806dbd0>
```

[8]:
```
circuitY.draw(output='mpl') # Rysowanie obwodu pomiaru typu Y
```

[8]:



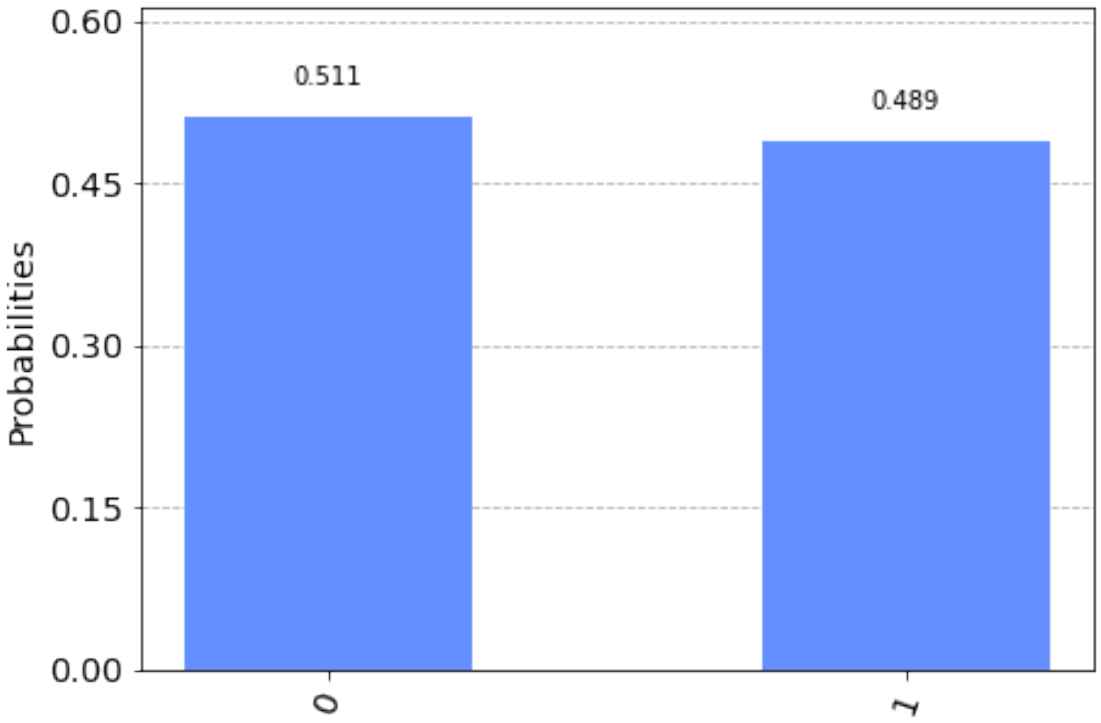
[9]:
```
# Wykonanie obliczeń kwantowych

job_simY = execute(circuitY, backend, shots=seria)
sim_resultY = job_simY.result()
# Liczbowa prezentacja wyników pomoaru typu Y
print(sim_resultY.get_counts(circuitY))
```

```
{'1': 1002, '0': 1046}
```

[10]:
```
# Graficzna prezentacja wyników pomiaru typu Y
plot_histogram(sim_resultY.get_counts(circuitY))
```

[10]:



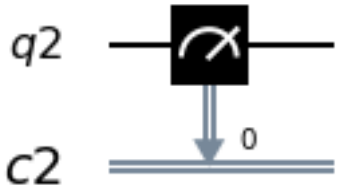
[11]:
```
#Przykład pomiaru typu Z ( inaczej sigmaZ)
nz = n  # Liczba kubitów i bitów
qz = QuantumRegister(nz) # Rejestr kwantowy
cz = ClassicalRegister(nz) # Rejestr klasyczny

circuitZ = QuantumCircuit(qz, cz) # Algorytm kwantowy - kwantowy obwód
circuitZ.measure(qz[0], cz[0]) # Sprawdzenie stanów kubitów - kwantowy pomiar
```

[11]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7fe0488ab550>
```

[12]:
```
circuitZ.draw(output='mpl') # Rysowanie obwodu pomiaru typu Z
```

[12]:



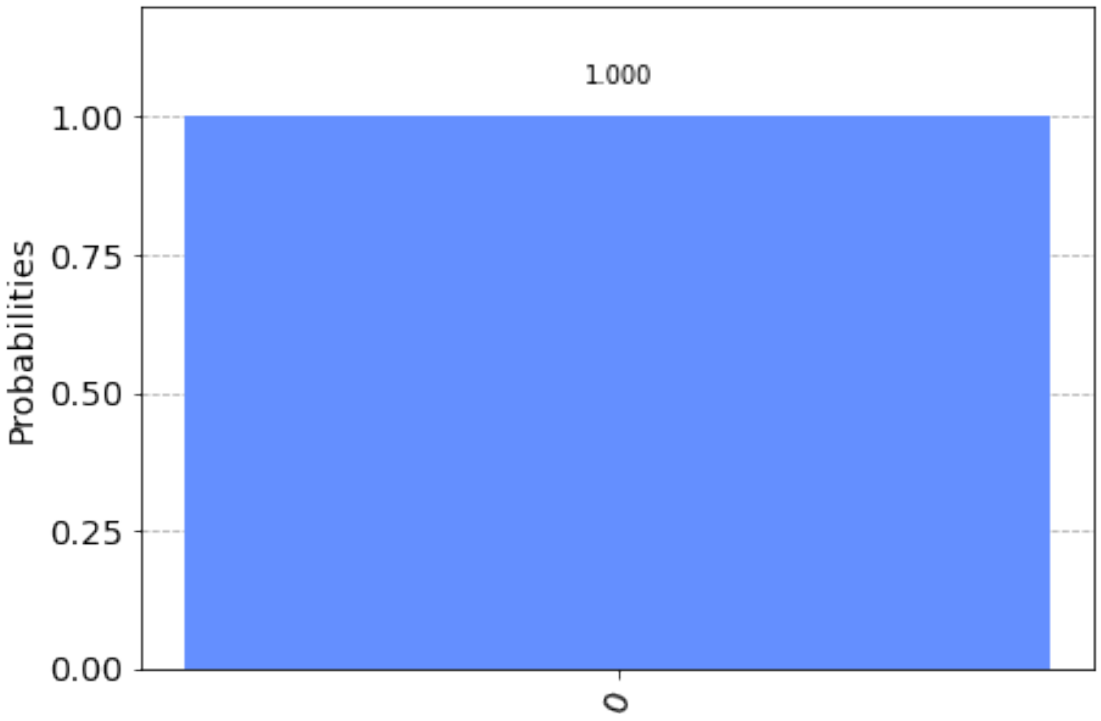
[13]:
```
# Wykonanie obliczeń kwantowych

job_simZ = execute(circuitZ, backend, shots=seria)
sim_resultZ = job_simZ.result()
# Liczbowa prezentacja wyników pomiaru typu Z
print(sim_resultZ.get_counts(circuitZ))
```

```
{'0': 2048}
```

[14]:
```
# Graficzna prezentacja wyników pomiaru typu Z
plot_histogram(sim_resultZ.get_counts(circuitZ))
```

[14]:



[15]:
```
#Przykład przygotowania stanu kubitu za pomocą bramek kwantowych U1 i U3

np = 1  # Liczba kubitów i bitów
theta= pi/4 # kąt theta
fi=pi/2 # kąt fi
qp = QuantumRegister(np) # Rejestr kwantowy
cp = ClassicalRegister(np) # Rejestr klasyczny
circuitP = QuantumCircuit(qp, cp) # Algorytm kwantowy - kwantowy obwód
circuitP.u3(theta,0,0,qp[0]) # Operacja X realizowana na kubicie qp[0]
circuitP.u1(fi,qp[0]) # Operacja X realizowana na kubicie qp[0]
```

```
circuitP.measure(qp[0], cp[0]) # Sprawdzenie stanów kubitów - kwantowy pomiar␣
↪typu Z
```

[15]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7fe043ebdbd0>
```

[16]:
```
circuitP.draw(output='mpl') # Rysowanie obwodu kwantowego służącego do␣
↪przygotowania stanu kubitu
```

[16]:

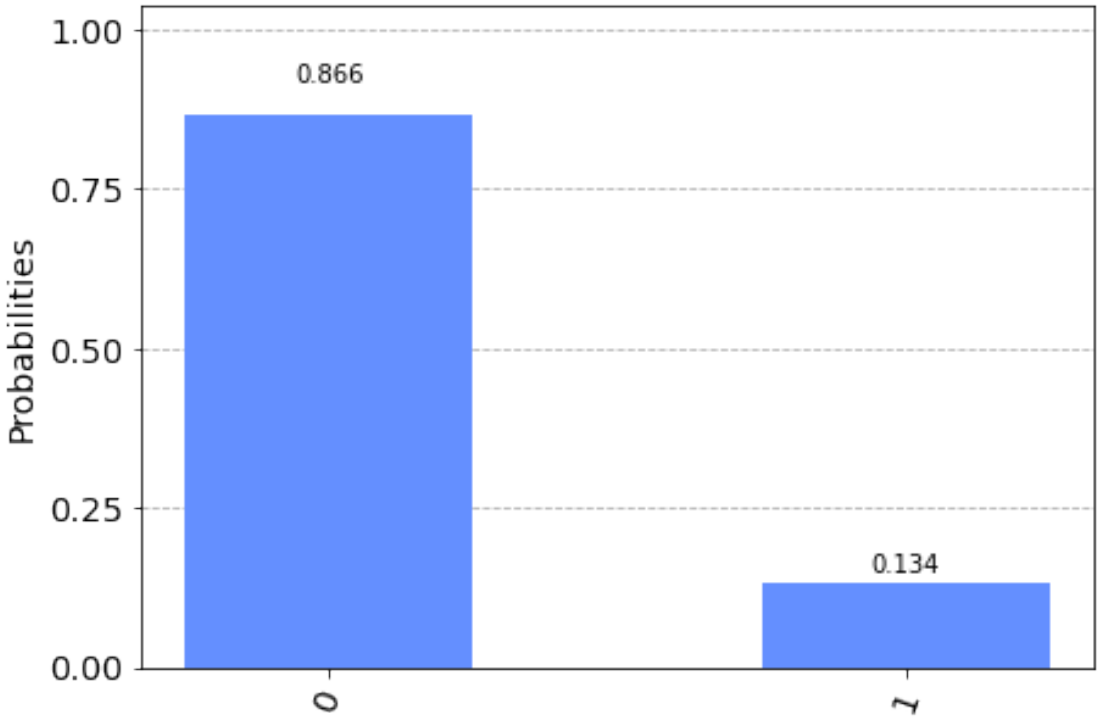
[17]:
```
# Wykonaj obliczeń kwantowych

job_simP = execute(circuitP, backend, shots=seria)
sim_resultP = job_simP.result()
# Liczbowa prezentacja wyników
print(sim_resultP.get_counts(circuitP))
```

```
{'1': 275, '0': 1773}
```

[18]:
```
# Graficzna prezentacja wyników
plot_histogram(sim_resultP.get_counts(circuitP))
```

[18]:


1.00
0.75
0.50
0.25
0.00
Probabilities
0.866
0.134
0
1

[ ]: