quantum_calc

13 ноября 2020 г.

1 Моделирование квантового компьютера

2 Квантовые вычисления

2.1 Базовые арифметические операторы

[1]:
```
3*5
```

[1]:
```
15
```

[2]:
```
8/4
```

[2]:
```
2.0
```

2.1.1 Комплексные числа

Комплексное число всегда $c = a + bj$

[2]:
```
import numpy as np
```

[3]:
```
1j*1j
```

[3]:
```
(-1+0j)
```

[4]:
```
z=4+8j
```

[5]:
```
w=5-6j
```

[6]:
```
print("Real of Z:", np.real(z))
```

```
Real of Z: 4.0
```

[7]:
```
print("Imag of Z:", np.imag(z))
```

```
Imag of Z: 8.0
```

[8]:
```
z+w
```

[8]: (9+2j)

2.1.2 Комплексное сопряжение

[9]:
```
np.conj(w)
```

[9]: (5+6j)

[10]:
```
np.conj(z)
```

[10]: (4-8j)

2.1.3 Норма/Модуль/Абсолютное значение

[11]:
```
np.abs(z)
```

[11]: 8.94427190999916

[12]:
```
np.abs(w)
```

[12]: 7.810249675906654

2.1.4 Строки и колонки

[13]:
```
row_vec = np.array([1, 2+2j,3])
```

[14]:
```
row_vec
```

[14]:
```
array([1.+0.j, 2.+2.j, 3.+0.j])
```

[15]:
```
col_vec = np.array([[1],[2+2j],[3]])
```

[16]:
```
col_vec
```

[16]:
```
array([[1.+0.j],
       [2.+2.j],
       [3.+0.j]])
```

Строки в квантовой механике соответствую <A| Колонки соответствуют |B>

2.1.5 Скалярное произведение двух векторов

[17]:
```
A = np.array([[1],[4-5j],[5],[-3]])
```

[18]:
```
A
```

[18]:
```
array([[ 1.+0.j],
       [ 4.-5.j],
       [ 5.+0.j],
       [-3.+0.j]])
```

[19]:
```
B = np.array([1, 5, -4j, -1j])
```

[20]:
```
B
```

[20]:
```
array([ 1.+0.j,  5.+0.j, -0.-4.j, -0.-1.j])
```

<B|A> будет:

[21]:
```
np.dot(B,A)
```

[21]:
```
array([21.-42.j])
```

2.1.6 Матрицы

[22]:
```
M=np.array([[2-1j, -3],[-5j, 2]])
```

[23]:
```
M
```

[23]:
```
array([[ 2.-1.j, -3.+0.j],
       [-0.-5.j,  2.+0.j]])
```

[24]:
```
M=np.matrix([[2-1j, -3],[-5j, 2]])
```

[25]:
```
M
```

[25]:
```
matrix([[ 2.-1.j, -3.+0.j],
        [-0.-5.j,  2.+0.j]])
```

2.1.7 Эрмитово сопряжение

[26]:
```
M.H
```

[26]:
```
matrix([[ 2.+1.j, -0.+5.j],
        [-3.-0.j,  2.-0.j]])
```

2.1.8 Тензорное произведение

[27]:
```
np.kron(M,M)
```

[27]:
```
matrix([[  3. -4.j,  -6. +3.j,  -6. +3.j,   9. -0.j],
        [ -5.-10.j,   4. -2.j,   0.+15.j,  -6. +0.j],
        [ -5.-10.j,   0.+15.j,   4. -2.j,  -6. +0.j],
```

```
[-25. +0.j,   0.-10.j,   0.-10.j,   4. +0.j]])
```

2.2 Кубиты, Сфера Блоха

pip install qiskit

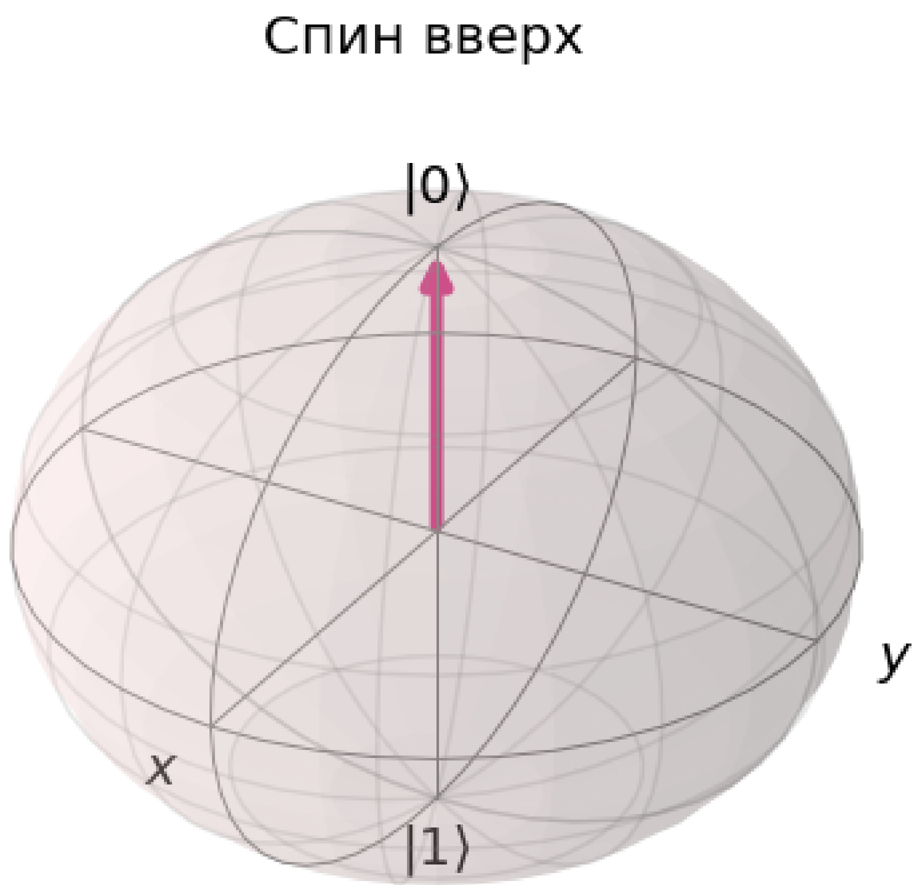
[2]:
```python
import qiskit
```

[3]:
```python
qiskit.__qiskit_version__
```

[3]:
```
{'qiskit-terra': '0.15.2',
 'qiskit-aer': '0.6.1',
 'qiskit-ignis': '0.4.0',
 'qiskit-ibmq-provider': '0.9.0',
 'qiskit-aqua': '0.7.5',
 'qiskit': '0.21.0'}
```

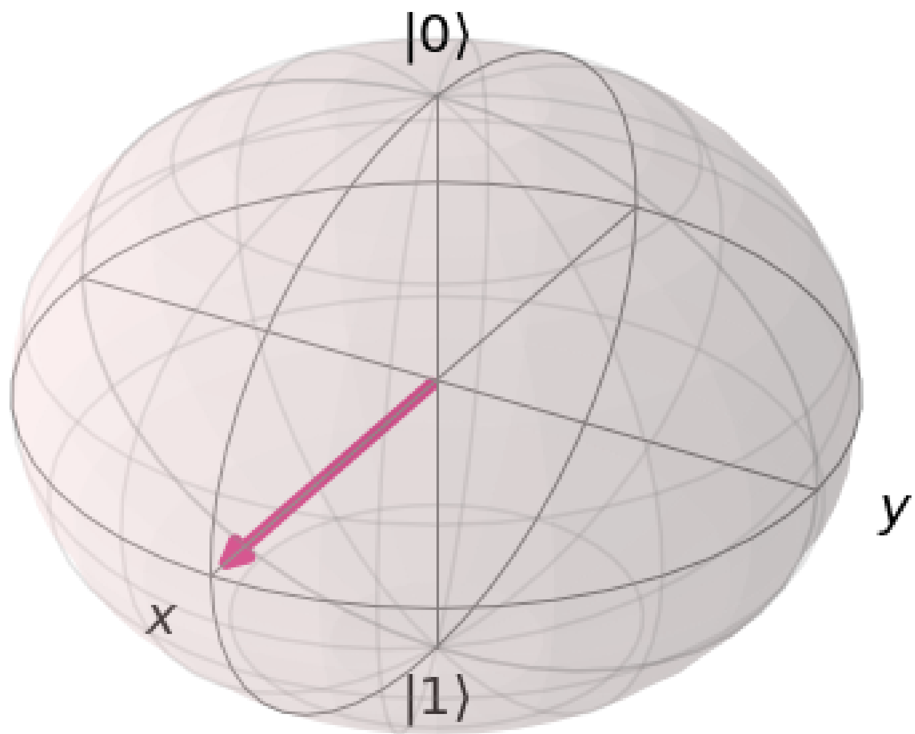
[4]:
```python
from qiskit import *
from qiskit.visualization import plot_bloch_vector
plot_bloch_vector([0,0,1], title = 'Спин вверх')
```

[4]:



[31]:
```
plot_bloch_vector([1,0,0])
```

[31]:



2.3 Базисные состояния

[32]:
```
ket_zero = np.array([[1],[0]])
ket_one = np.array([[0],[1]])
```

[33]:
```
np.kron(ket_zero, ket_zero)
```

[33]:
```
array([[1],
       [0],
       [0],
       [0]])
```

[34]:
```
np.kron(ket_zero, ket_one)
```

[34]:
```
array([[0],
       [1],
```

```
        [0],
        [0]])
```

[35]:
```
np.kron(ket_one, ket_zero)
```

[35]:
```
array([[0],
       [0],
       [1],
       [0]])
```

[36]:
```
np.kron(ket_one, ket_one)
```

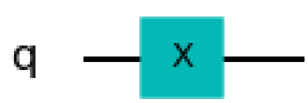
[36]:
```
array([[0],
       [0],
       [0],
       [1]])
```

2.4 Квантовые операторы

2.4.1 X на |0>

[6]:
```
qc = QuantumCircuit(1)
qc.x(0)
qc.draw('mpl')
```

[6]:



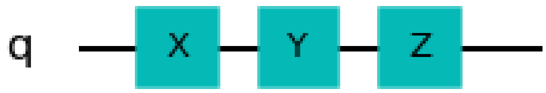
[7]:
```
backend = Aer.get_backend('statevector_simulator')
out = execute(qc, backend).result().get_statevector()
print(out)
```

```
[0.+0.j 1.+0.j]
```

2.5 Z and Y операторы

[8]:
```
qc.y(0)
qc.z(0)
qc.draw('mpl')
```

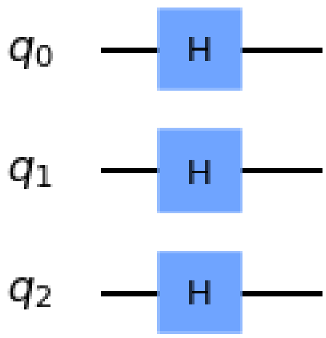
[8]:



2.6 Оператор Адамара

[9]:
```
qc = QuantumCircuit(3)
for qubit in range(3):
    qc.h(qubit)
qc.draw('mpl')
```

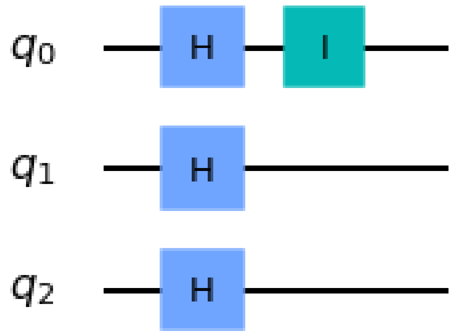
[9]:



2.7 Единичный оператор

[10]:
```
qc.i(0)
qc.draw('mpl')
```

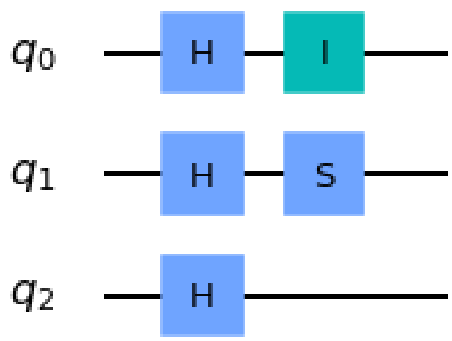
[10]:



2.8 S - Оператор

[11]:
```
qc.s(1)
qc.draw('mpl')
```

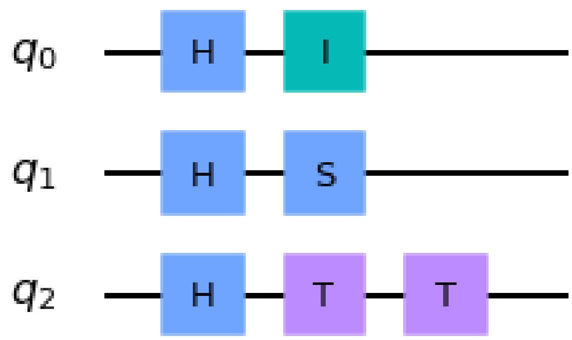
[11]:



2.9 T - Оператор

[13]:
```
qc.t(2)
qc.draw('mpl')
```

[13]:



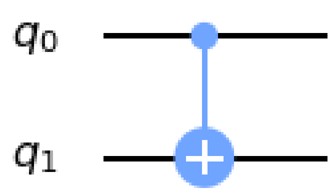
2.10 U - Оператор

...

2.11 C-Not Оператор

[17]:
```
qc = QuantumCircuit(2)
qc.cx(0,1)
qc.draw('mpl')
```

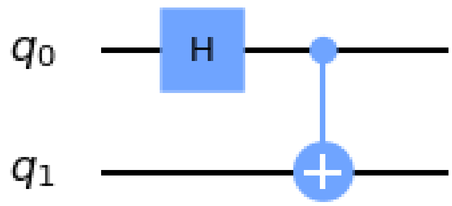
[17]:



2.12 Запутанность

[18]:
```
qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0,1)
qc.draw('mpl')
```

[18]:



[19]:
```
final_state = execute(qc,backend).result().get_statevector()
print(final_state)
```

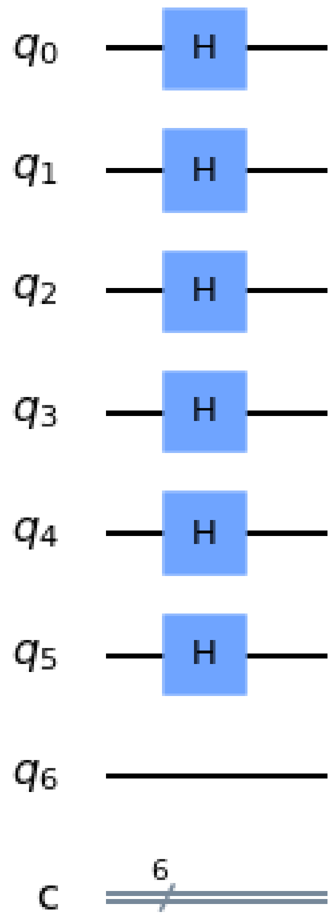
```
[0.70710678+0.j 0.        +0.j 0.        +0.j 0.70710678+0.j]
```

2.13 Берштейн-Вазирани Алгоритм

[ ]:
```
s = 101011
```

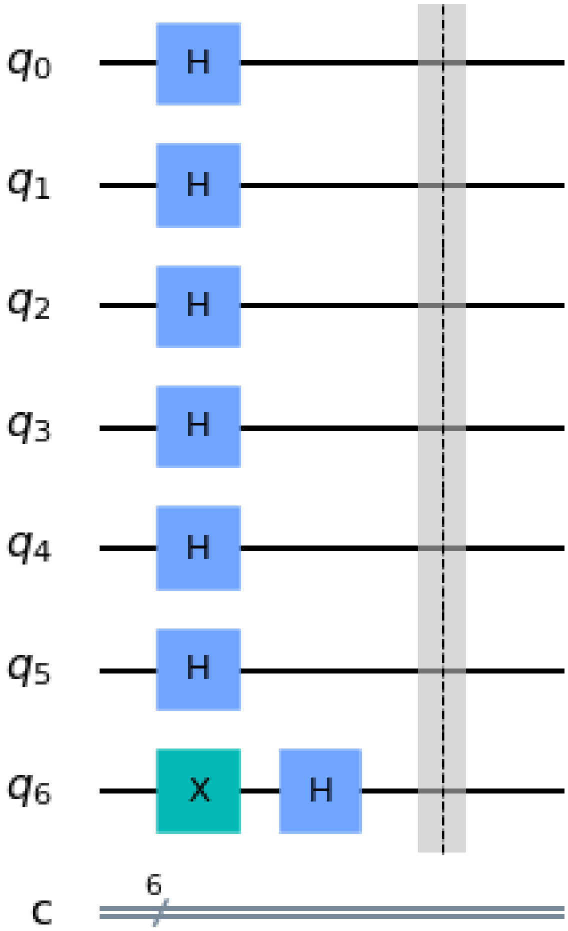
[20]:
```
qc = QuantumCircuit(6+1, 6)
qc.h([0,1,2,3,4,5])
qc.draw('mpl')
```

[20]:



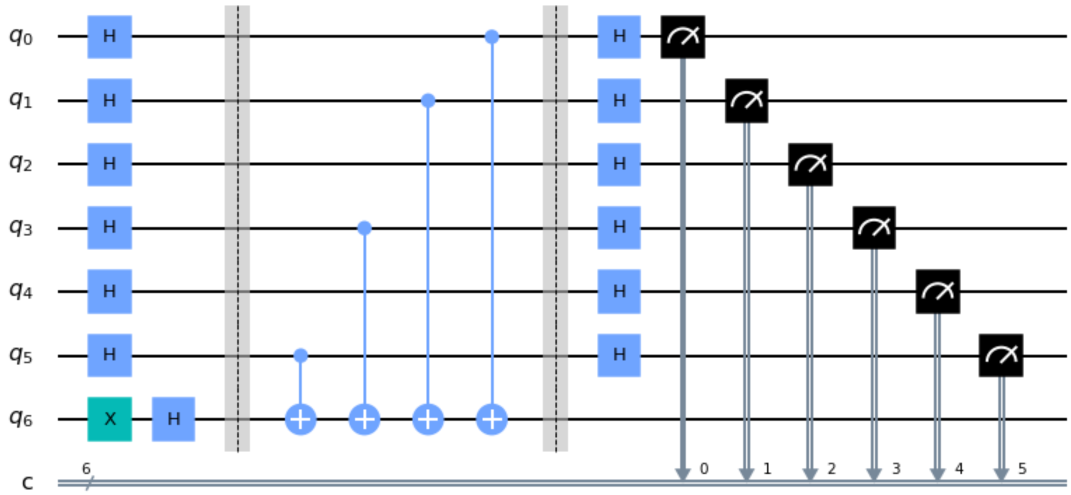
[21]:
```
qc.x(6)
qc.h(6)
qc.barrier()
qc.draw('mpl')
```

[21]:



[22]:
```
qc.cx(5,6)
qc.cx(3,6)
qc.cx(1,6)
qc.cx(0,6)
qc.barrier()
qc.h([0,1,2,3,4,5])
qc.measure([0,1,2,3,4,5],[0,1,2,3,4,5])
qc.draw('mpl')
```

[22]:

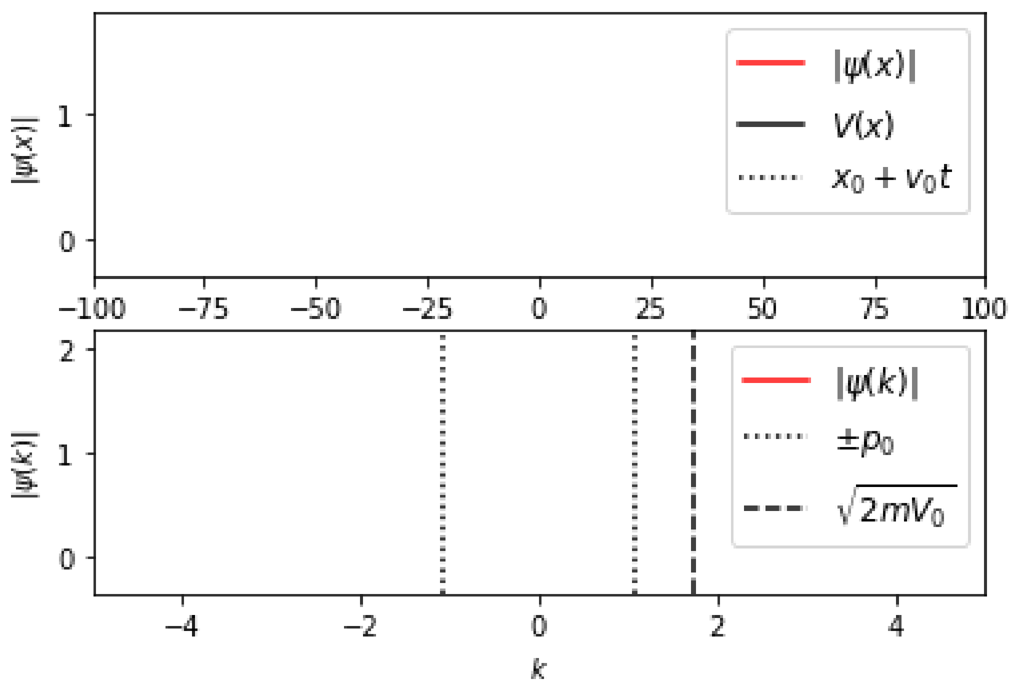


[24]:
```
simulator= Aer.get_backend('qasm_simulator')
result = execute(qc,backend=simulator, shots = 1).result()
counts=result.get_counts()
print(counts)
```

```
{'101011': 1}
```

[27]:
```

```



```
[ ]:
```