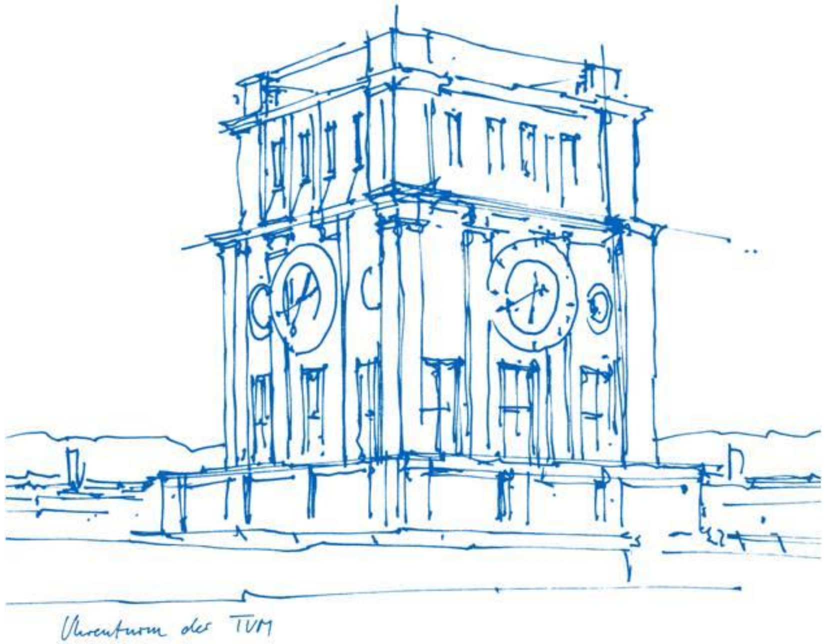
Seminar Advanced Topics of Quantum Computing

Technische Universität München

Munich, 17. November 2023



Circuit Simulation
Statevector simulator with Python

Student Bachelor of Business Informatics

Marina Zhdanova

Munich, 17. November 2023

Content of Seminar Work

1. Quantum circuit model in general
2. Quantum Information
    - Vectors
    - Unitary matrices
3. Quantum circuit basic gates examples
4. Statevector simulation on Python

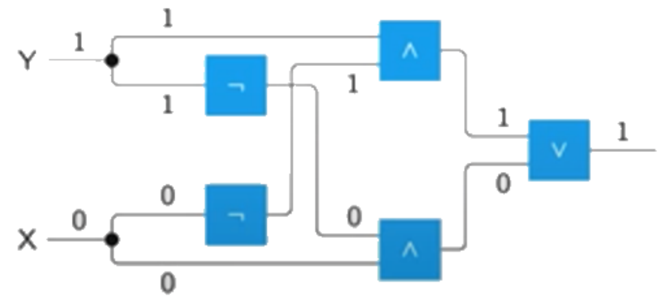
Quantum Circuits Model

Circuit – general model of computation:

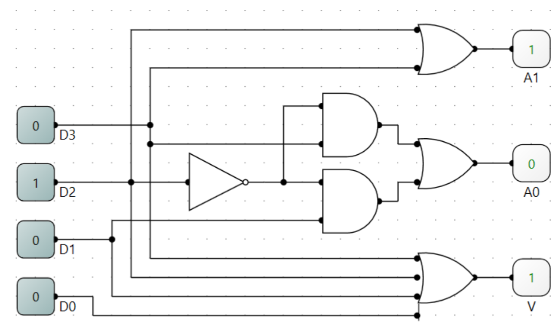
- Wires carry information
- Gates represent operations
- Circuits are always acyclic
- Information flows from left to right

Quantum algorithms are most commonly described by a **Quantum circuit**:

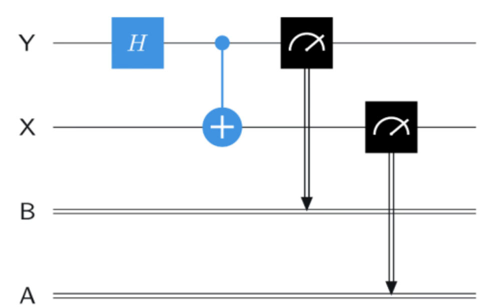
- Wires represent qubits
- Gates represent:
  - unitary operations
  - measurements



Boolean logic



Electrical engineering



Quantum circuit

Quantum Information

- Quantum states are represent by **vectors**,
  Operations are represented by **unitary matrices**
- Standard basis measurements
- Unitary operations (Gates)

$$v = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix}$$ Euclidean norm

$$\|v\| = \sqrt{\sum_{k=1}^{n} |\alpha_k|^2}.$$

Quantum state vectors

A quantum state of a system is represented by a **column vector** (state set Σ), similar to probabilistic states

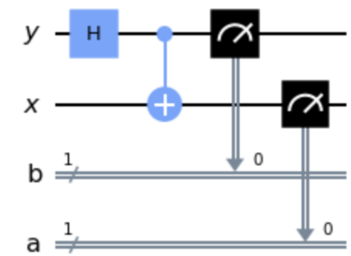
1. The entries of a quantum state vector are **complex numbers**.
2. The sum of the **absolute values squared** of the entries of a quantum state vector is 1.

$$|\psi\rangle = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} \qquad \frac{1}{\sqrt{|\Sigma|}} \sum_{a \in \Sigma} |a\rangle,$$

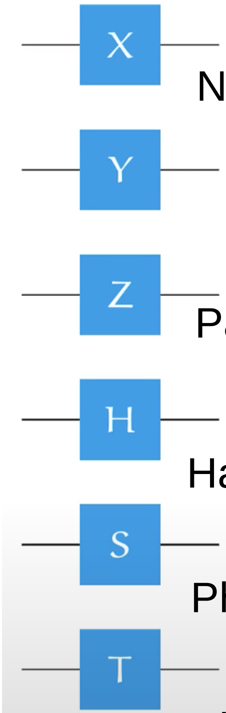
Dirac notation quantum state vector:
number of elements in Σ

Basic Quantum Gates (=Unitary operations on qubits):

- are operations applied to a qubit that changes the quantum state of the qubit for quantum computation to solve the problem



Single-qubit gates:



NOT

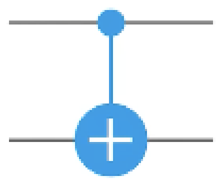
Pauli

Hadamard

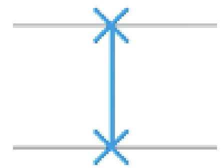
Phase

Phase

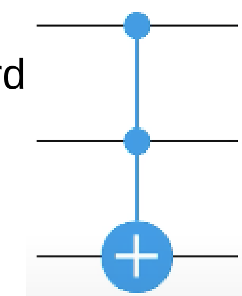
Multi-qubit gates:



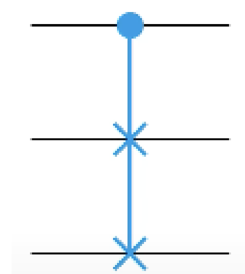
Controlled- NOT



Swap gate



Toffoli gate**



Fredkin gate

**Double controlled-NOT gate (CCX), has two control qubits and one target. It applies a NOT to the target only when both controls are in state |1⟩

Rotation Operators

RZ

The rotation operators are defined as:

$$R_x(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

$$\boxed{R_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}}$$

Parametric gate: the matrix depends on the *parameter*, which should be passed together *with the qubits on which the gate should act*

are generated by exponentiation of the Pauli matrices according to:

$$exp(iAx) = \cos(x)\, I + i\sin(x)\, A$$

where A is one of the three Pauli Matrices.

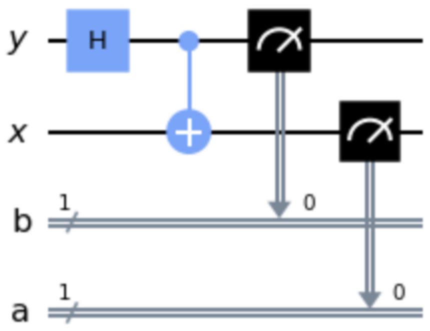
- Rz rotation operator can also be expressed as

$$\begin{pmatrix} e^{i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

which differs from the definition above by a global phase only.

Examples of operation on information using gates

- **Hadamard Gate** perform first operation**, Controlled -NOT** the second
  - Control- Y qubit
  - Target- X qubit



time

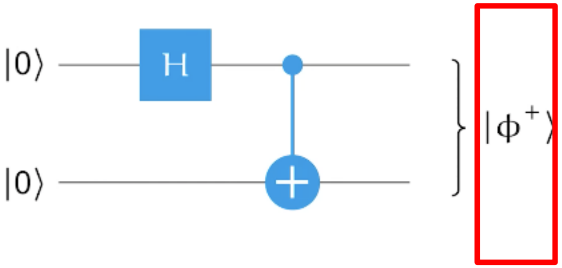
multiplication

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \end{pmatrix}.$$

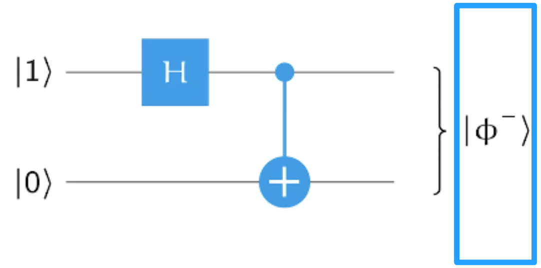
1. Controlled-NOT 2. Hadamard

Explanation of calculations

Run qubit at ket |0>, |0>

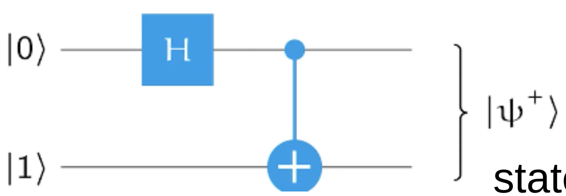


Run qubit at ket |1>, |0>



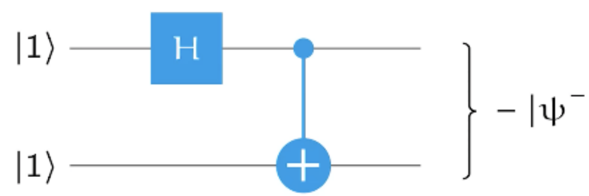
$$U|00\rangle = |\phi^+\rangle$$
$$U|01\rangle = |\phi^-\rangle$$
$$U|10\rangle = |\psi^+\rangle$$
$$U|11\rangle = -|\psi^-\rangle$$

Run qubit at ket |0>, |1>



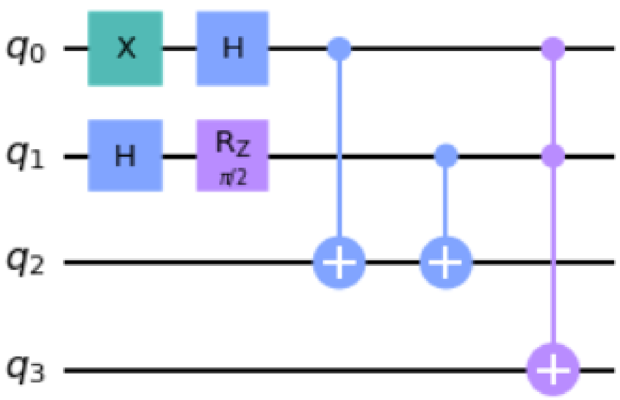
state

Run qubit at ket |1>, |1>



$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \end{pmatrix}$$

Statevector simulator with Python




```python
import numpy as np
import matplotlib.pyplot as plt
```

```python
q = QuantumCircuit(4)
q.x(0) # NOT gate on Q0
q.h(0) # Hadamard gate on Q0
q.h(1) # Hadamard gate on Q1
q.rz(np.pi/2, 1) # RZ gate on Q1
q.cx(0, 2) # Controlled NOT gate Q0 -> Q2
q.cx(1, 2) # Controlled NOT gate Q1 -> Q2
q.ccx(0, 1, 3) # Tofolli gate Q0 & Q1 -> Q3
print("State vector:\n", q.get_statevector())
print("Measurements:\n", q.measure(5))
q.draw_measures()
```

```
State vector:
 [ 0.35355339-0.35355339j  0.        +0.j          0.        +0.j
  0.        +0.j          0.        +0.j          0.        +0.j
  0.35355339+0.35355339j  0.        +0.j          0.        +0.j
  0.        +0.j         -0.35355339+0.35355339j  0.        +0.j
  0.        +0.j         -0.35355339-0.35355339j  0.        +0.j
  0.        +0.j        ]
```

I built Python statevector simulator use Qiskit as a reference to understand, what it is doing under the hood

https://github.com/MarinaZuzu/Test_-Quantum-Computing-Seminar/tree/main

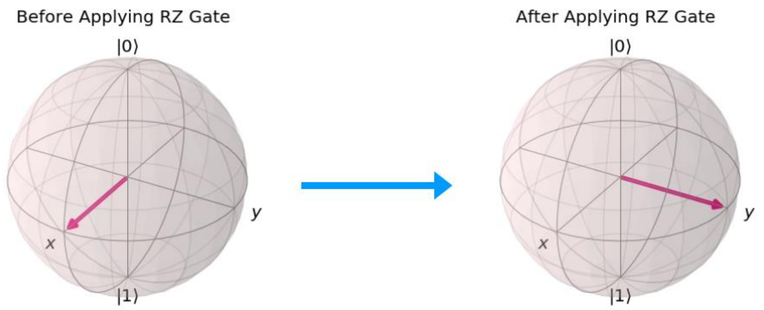
Class QuantumCircuit

```python
# Class QuantumCircuit
class QuantumCircuit:
    def __init__(self, qubits):
        if qubits < 1:
            raise Exception("qubits should be more than zero")
        self.qubits = qubits
        self.measures = {}

        # Init state vector
        self.state_vector = np.zeros((2**qubits), dtype=complex)
        # Start from state "0000"
```

Parametric gate (RZ) simulation

- The gate matrix depends on the parameter
- Parameter (phi) passed together with the qubits
- RZ-gate is a single-qubit rotation about the Z axis

$$RZ(\lambda) = \exp\left(-i\frac{\lambda}{2}Z\right) = \begin{pmatrix} e^{-i\frac{\lambda}{2}} & 0 \\ 0 & e^{i\frac{\lambda}{2}} \end{pmatrix}$$




```
def rz(self, phi, q0):

    if not self.__check_param(q0):
        raise Exception("wrong parameter")

    # Init RZ gate matrix
    RZ = np.array([[np.exp(-0.5j*phi), 0], [0, np.exp(0.5j*phi)]], dtype=complex) # rz(phi, 0)
    print("\nRZ gate base matrix:\n", RZ)

    # Adjust RZ gate matrix for circuit size
    I = np.eye(2**(self.qubits - 1))
    RZI = np.kron(RZ, I)
    print("\nRZ gate matrix for quibit 0:\n", RZI)

    # reorder to qbit q0
    perm = self.__get_perm(q0)
    G = self.__reorder_gate(RZI, perm)
    print("\nRZ gate matrix for quibit {}:\n".format(q0), G)

    # calculate new state vector
    self.state_vector = np.matmul(G, self.state_vector)
```

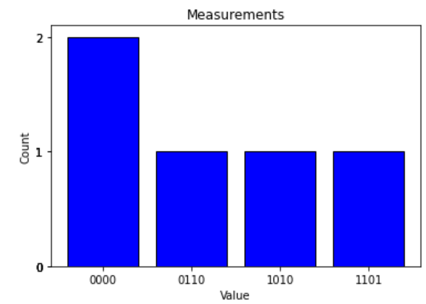
Measurement of qubit’s states simulation

- Sampling from the statevector
- Measure all qubits
- Result of a single measurement as a bitstring

```
def measure(self, shots):

    # calculate probability distribution
    distribution = abs(self.state_vector)**2
    print("Distribution:\n", distribution)

    # perform measures with a random sample function
    self.measures = {}
    for i in range(shots):
        sample = np.random.choice(len(distribution), p=distribution)
        key = f"{sample:0{self.qubits}b}" # single measurement as a bitstring
        if key in self.measures:
            self.measures[key] += 1
        else:
            self.measures[key] = 1

    # sort the dictionary
    self.measures = dict(sorted(self.measures.items()))
    return self.measures
```

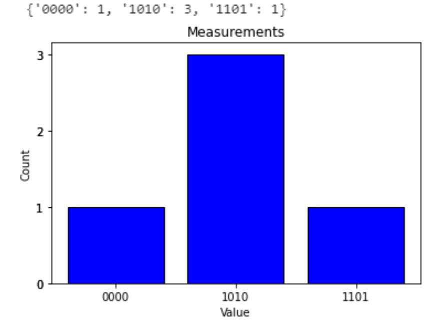
```
print("State vector:\n", q.get_statevector())
print("Measurements:\n", q.measure(5))
q.draw_measures()
```

```
State vector:
 [ 0.35355339-0.35355339j  0.        +0.j          0.        +0.j
  0.        +0.j          0.        +0.j          0.        +0.j
  0.35355339+0.35355339j  0.        +0.j          0.        +0.j
  0.        +0.j         -0.35355339+0.35355339j  0.        +0.j
  0.        +0.j         -0.35355339-0.35355339j  0.        +0.j
  0.        +0.j        ]
Distribution:
 [0.25 0.   0.   0.   0.   0.   0.25 0.   0.   0.   0.25 0.   0.   0.25
 0.   0.  ]
Measurements:
 {'0000': 2, '0110': 1, '1010': 1, '1101': 1}
```




```
print("State vector:\n", q.get_statevector())
print("Measurements:\n", q.measure(5))
q.draw_measures()
```

```
State vector:
 [ 0.35355339-0.35355339j  0.        +0.j          0.        +0.j
  0.        +0.j          0.        +0.j          0.        +0.j
  0.35355339+0.35355339j  0.        +0.j          0.        +0.j
  0.        +0.j         -0.35355339+0.35355339j  0.        +0.j
  0.        +0.j         -0.35355339-0.35355339j  0.        +0.j
  0.        +0.j        ]
Distribution:
 [0.25 0.   0.   0.   0.   0.   0.25 0.   0.   0.   0.25 0.   0.   0.25
 0.   0.  ]
Measurements:
 {'0000': 1, '1010': 3, '1101': 1}
```



Thank you!

Literature

1. https://www.quantum-inspire.com/kbase/what-is-a-quantum-algorithm
2. https://learning.quantum-computing.ibm.com/tutorial/composer-user-guide
3. https://colab.research.google.com/github/MarinaZuzu/Test_-Quantum-Computing-Seminar/blob/main/sim-wrapped-inclass%20(2).ipynb#scrollTo=f3d7774f-1f7a-416d-a57a-857e84c7c0c5