

# Symulacja obliczeń kwantowych

## Model kwantowych bramek logicznych w NumPy

Robert Nowotniak

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej  
Politechnika Łódzka

Sekcja Informatyki Kwantowej, 10 października 2007

- 1 **Python i NumPy** - podstawowe informacje
- 2 Wymagania do obliczeń kwantowych
- 3 Numerical Python - najważniejsze funkcje
- 4 Przykład
- 5 Podsumowanie

- 1 Python i NumPy - podstawowe informacje
- 2 Wymagania do **obliczeń kwantowych**
- 3 Numerical Python - najważniejsze funkcje
- 4 Przykład
- 5 Podsumowanie

# Plan prezentacji

- 1 Python i NumPy - podstawowe informacje
- 2 Wymagania do obliczeń kwantowych
- 3 **Numerical Python** - najważniejsze funkcje
- 4 Przykład
- 5 Podsumowanie

# Plan prezentacji

- 1 Python i NumPy - podstawowe informacje
- 2 Wymagania do obliczeń kwantowych
- 3 Numerical Python - najważniejsze funkcje
- 4 Przykład
- 5 Podsumowanie

## Cechy języka Python:

- 1 Łatwy do nauczenia
- 2 Język imperatywny trzeciej generacji, wysokiego poziomu
- 3 Dobre wsparcie dla programowania zorientowanego obiektowo
- 4 Bogata biblioteka standardowa
- 5 Elementy programowania funkcyjnego (anonimowe funkcje, domknięcia)
- 6 Tutorial dostępny na stronie:  
*<http://docs.python.org/tut/>*

## Cechy języka Python:

- 1 Łatwy do nauczenia
- 2 Język imperatywny trzeciej generacji, wysokiego poziomu
- 3 Dobre wsparcie dla programowania zorientowanego obiektowo
- 4 Bogata biblioteka standardowa
- 5 Elementy programowania funkcyjnego (anonimowe funkcje, domknięcia)
- 6 Tutorial dostępny na stronie:  
*<http://docs.python.org/tut/>*

## Cechy języka Python:

- 1 Łatwy do nauczenia
- 2 Język imperatywny trzeciej generacji, wysokiego poziomu
- 3 Dobre wsparcie dla programowania zorientowanego obiektowo
- 4 Bogata biblioteka standardowa
- 5 Elementy programowania funkcyjnego (anonimowe funkcje, domknięcia)
- 6 Tutorial dostępny na stronie:  
*<http://docs.python.org/tut/>*

## Cechy języka Python:

- 1 Łatwy do nauczenia
- 2 Język imperatywny trzeciej generacji, wysokiego poziomu
- 3 Dobre wsparcie dla programowania **zorientowanego obiektowo**
- 4 Bogata biblioteka standardowa
- 5 Elementy programowania funkcyjnego (anonimowe funkcje, domknięcia)
- 6 Tutorial dostępny na stronie:  
*<http://docs.python.org/tut/>*

## Cechy języka Python:

- 1 Łatwy do nauczenia
- 2 Język imperatywny trzeciej generacji, wysokiego poziomu
- 3 Dobre wsparcie dla programowania zorientowanego obiektowo
- 4 Bogata biblioteka standardowa
- 5 Elementy programowania funkcyjnego (anonimowe funkcje, domknięcia)
- 6 Tutorial dostępny na stronie:  
*<http://docs.python.org/tut/>*

## Cechy języka Python:

- 1 Łatwy do nauczenia
- 2 Język imperatywny trzeciej generacji, wysokiego poziomu
- 3 Dobre wsparcie dla programowania zorientowanego obiektowo
- 4 Bogata biblioteka standardowa
- 5 Elementy programowania funkcyjnego (anonimowe funkcje, domknięcia)
- 6 **Tutorial dostępny na stronie:**  
*<http://docs.python.org/tut/>*

## Biblioteka **Numerical Python** (NumPy)

- Darmowa alternatywa dla Matlaba (do prostych zastosowań)
- Przydatne (dla nas) rzeczy:
  - Wydajne operacje na macierzach  
wyznaczniki, ślady, działania, ...
  - Łatwe obliczenia i eksperymenty numeryczne
  - Algebra liniowa - moduł `numpy.linalg`  
wartości i wektory własne, rozkłady SVD, QR, ...
  - Szybkie przekształcenia ortogonalne (FFT) - moduł `numpy.fft`

## Biblioteka **Numerical Python** (NumPy)

- Darmowa alternatywa dla Matlaba (do prostych zastosowań)
- Przydatne (dla nas) rzeczy:
  - Wydajne **operacje na macierzach**  
wyznaczniki, ślady, działania, ...
  - Łatwe obliczenia i eksperymenty numeryczne
  - Algebra liniowa - moduł `numpy.linalg`  
wartości i wektory własne, rozkłady SVD, QR, ...
  - Szybkie przekształcenia ortogonalne (FFT) - moduł `numpy.fft`

## Biblioteka **Numerical Python** (NumPy)

- Darmowa alternatywa dla Matlaba (do prostych zastosowań)
- Przydatne (dla nas) rzeczy:
  - Wydajne operacje na macierzach  
wyznaczniki, ślady, działania, ...
  - Łatwe **obliczenia** i **eksperymenty numeryczne**
  - Algebra liniowa - moduł `numpy.linalg`  
wartości i wektory własne, rozkłady SVD, QR, ...
  - Szybkie przekształcenia ortogonalne (FFT) - moduł `numpy.fft`

## Biblioteka **Numerical Python** (NumPy)

- Darmowa alternatywa dla Matlaba (do prostych zastosowań)
- Przydatne (dla nas) rzeczy:
  - Wydajne operacje na macierzach  
wyznaczniki, ślady, działania, ...
  - Łatwe obliczenia i eksperymenty numeryczne
  - Algebra liniowa - **moduł `numpy.linalg`**  
wartości i wektory własne, rozkłady SVD, QR, ...
  - Szybkie przekształcenia ortogonalne (FFT) - moduł `numpy.fft`

## Biblioteka **Numerical Python** (NumPy)

- Darmowa alternatywa dla Matlaba (do prostych zastosowań)
- Przydatne (dla nas) rzeczy:
  - Wydajne operacje na macierzach  
wyznaczniki, ślady, działania, ...
  - Łatwe obliczenia i eksperymenty numeryczne
  - Algebra liniowa - moduł `numpy.linalg`  
wartości i wektory własne, rozkłady SVD, QR, ...
  - Szybkie przekształcenia ortogonalne (FFT) - **moduł**  
`numpy.fft`

- 1 Python i NumPy - podstawowe informacje
- 2 **Wymagania do obliczeń kwantowych**
- 3 Numerical Python - najważniejsze funkcje
- 4 Przykład
- 5 Podsumowanie

Do symulacji **obliczeń kwantowych** (w modelu bramek kwantowych) potrzebujemy co najmniej następującego zestawu:

- 1 **wektory liczb zespolonych** (macierze kolumnowe) - do reprezentowania kubitów (2-elementowe) i rejestrów kwantowych ( $2^n$ -elementowe) + **odpowiedni zbiór operacji**
- 2 **macierze liczb zespolonych** (bramki kwantowe) - do opisywania zmian stanów układów kwantowych (opisywanych przez operatory unitarne) + **odpowiedni zbiór operacji**
- 3 odpowiednie **struktury łączące powyższe elementy** - do zapisywania układu bramek kwantowych jako sieci macierzy

Do symulacji **obliczeń kwantowych** (w modelu bramek kwantowych) potrzebujemy co najmniej następującego zestawu:

- 1 **wektory liczb zespolonych** (macierze kolumnowe) - do reprezentowania kubitów (2-elementowe) i rejestrów kwantowych ( $2^n$ -elementowe) + **odpowiedni zbiór operacji**
- 2 **macierze liczb zespolonych** (bramki kwantowe) - do opisywania zmian stanów układów kwantowych (opisywanych przez operatory unitarne) + **odpowiedni zbiór operacji**
- 3 odpowiednie **struktury łączące powyższe elementy** - do zapisywania układu bramek kwantowych jako sieci macierzy

Do symulacji **obliczeń kwantowych** (w modelu bramek kwantowych) potrzebujemy co najmniej następującego zestawu:

- 1 **wektory liczb zespolonych** (macierze kolumnowe) - do reprezentowania kubitów (2-elementowe) i rejestrów kwantowych ( $2^n$ -elementowe) + **odpowiedni zbiór operacji**
- 2 **macierze liczb zespolonych** (bramki kwantowe) - do opisywania zmian stanów układów kwantowych (opisywanych przez operatory unitarne) + **odpowiedni zbiór operacji**
- 3 odpowiednie **struktury łączące powyższe elementy** - do zapisywania układu bramek kwantowych jako sieci macierzy

Do symulacji **obliczeń kwantowych** (w modelu bramek kwantowych) potrzebujemy co najmniej następującego zestawu:

- 1 **wektory liczb zespolonych** (macierze kolumnowe) - do reprezentowania kubitów (2-elementowe) i rejestrów kwantowych ( $2^n$ -elementowe) + **odpowiedni zbiór operacji**
- 2 **macierze liczb zespolonych** (bramki kwantowe) - do opisywania zmian stanów układów kwantowych (opisywanych przez operatory unitarne) + **odpowiedni zbiór operacji**
- 3 odpowiednie **struktury łączące powyższe elementy** - do zapisywania układu bramek kwantowych jako sieci macierzy

Niezbędne operacje na wektorach (**kubity i rejestry kwantowe**):

- 1 sprzężenie hermitowskie
- 2 normalizacja
- 3 pomiar kwantowy (redukcja wektora stanu) - dwa rodzaje:
  - pomiar całego rejestru
  - pomiar **niektórych** kubitów z rejestru

Niezbędne operacje na wektorach (**kubity i rejestry kwantowe**):

- 1 sprzężenie hermitowskie
- 2 normalizacja
- 3 pomiar kwantowy (redukcja wektora stanu) - dwa rodzaje:
  - pomiar całego rejestru
  - pomiar **niektórych** kubitów z rejestru

Niezbędne operacje na wektorach (**kubity i rejestry kwantowe**):

- 1 sprzężenie hermitowskie
- 2 normalizacja
- 3 pomiar kwantowy (redukcja wektora stanu) - dwa rodzaje:
  - pomiar całego rejestru
  - pomiar **niektórych** kubitów z rejestru

Niezbędne operacje na wektorach (**kubity i rejestry kwantowe**):

- 1 sprzężenie hermitowskie
- 2 normalizacja
- 3 pomiar kwantowy (redukcja wektora stanu) - dwa rodzaje:
  - pomiar całego rejestru
  - pomiar **niektórych** kubitów z rejestru

## Niezbędne operacje na macierzach (**bramki kwantowe**):

- 1 iloczyn tensorowy macierzy
- 2 mnożenie macierzy (zwykłe)
- 3 sprzężenie hermitowskie
- 4 mnożenie przez wektor (działanie na stan w rejestrze kwantowym)
- 5 dodatkowo przydatne:
  - sprawdzenie, czy macierz jest unitarna
  - obliczanie wyznacznika, odwrotności

Niezbędne operacje na macierzach (**bramki kwantowe**):

- 1 **iloczyn tensorowy** macierzy
- 2 mnożenie macierzy (zwykłe)
- 3 sprzężenie hermitowskie
- 4 mnożenie przez wektor (działanie na stan w rejestrze kwantowym)
- 5 dodatkowo przydatne:
  - sprawdzenie, czy macierz jest unitarna
  - obliczanie wyznacznika, odwrotności

Niezbędne operacje na macierzach (**bramki kwantowe**):

- 1 iloczyn tensorowy macierzy
- 2 mnożenie macierzy (zwykłe)
- 3 sprzężenie hermitowskie
- 4 mnożenie przez wektor (działanie na stan w rejestrze kwantowym)
- 5 dodatkowo przydatne:
  - sprawdzenie, czy macierz jest unitarna
  - obliczanie wyznacznika, odwrotności

# Plan prezentacji

- 1 Python i NumPy - podstawowe informacje
- 2 Wymagania do obliczeń kwantowych
- 3 **Numerical Python - najważniejsze funkcje**
- 4 Przykład
- 5 Podsumowanie

Podstawowe typy danych w Numerical Python:

1 macierze **matrix**

```
macierz1 = matrix([[1, 2],  
                  [3, 4]])
```

2 tablice **array**

```
tablica1 = array([[5, 6],  
                 [7, 8]])
```

3 **matrix** i **array** różnią się sposobem działania operatora \* (mnożenie).

## Uwaga

- 1 Obiekty klasy **matrix** można mnożyć za pomocą przeciążonego dla nich operatora `*`.
- 2 Tablice klasy **array** należy mnożyć za pomocą funkcji `dot()`.
- 3 **Różnica:** operator `*` dla obiektów *array* oznacza mnożenie "*element po elemencie*".

## Uwaga

- 1 Obiekty klasy **matrix** można mnożyć za pomocą przeciążonego dla nich operatora `*`.
- 2 Tablice klasy **array** należy mnożyć za pomocą funkcji `dot()`.
- 3 **Różnica:** operator `*` dla obiektów *array* oznacza mnożenie "*element po elemencie*".

**Przykład:** Definicje kilku podstawowych bramek kwantowych (Controlled-Not, Hadamard (dla jednego kubitu), obrót amplitudy o  $\pi/2$ ):

```
1 CNot = array([
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 1],
    [0, 0, 1, 0]])

2 H1 = sqrt(2) / 2 * array([
    [1, 1],
    [1,-1]])

3 PhasePi2 = array([[ 1,          0          ],
                    [ 0, exp(1j * pi/2) ]])
```

# NumPy - kluczowe funkcje

## Podstawowe funkcje:

- `identity(n)` , `eye(n)` - macierz jednostkowa  $n \times n$
- `dot(m1, m2)` - iloczyn skalarny, mnożenie macierzowe  $m1 \cdot m2$
- `transpose(m)` - transpozycja macierzy  $m^T$
- `shape(m1)` - zwraca rozmiar macierzy  $(w, k)$
- `kron(m1, m2)` - iloczyn tensorowy (Kroneckera)  $m1 \otimes m2$
- `vdot(v1, v2)` - iloczyn  $\langle v1 | v2 \rangle$
- `set_printoptions(...)` - opcje wyświetlania l. zmiennoprzecinkowych
- `M1.H` - sprzężenie hermitowskie macierzy  $M^\dagger$
- `M1[n]` -  $n$ -ty wiersz macierzy  $M1$
- `M1[:, n]` -  $n$ -ta kolumna macierzy  $M1$

- 1 Python i NumPy - podstawowe informacje
- 2 Wymagania do obliczeń kwantowych
- 3 Numerical Python - najważniejsze funkcje
- 4 **Przykład**
- 5 Podsumowanie

**Przykład** (`qcirc1.py`): Generowanie splątanej trójki kubitów:

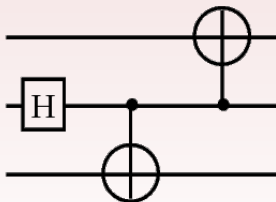
$$|\phi\rangle = \frac{\sqrt{2}}{2}|000\rangle + \frac{\sqrt{2}}{2}|111\rangle$$

# Symulacja układu kwantowego

**Przykład** (`qcircl.py`): Generowanie splątanej trójki kubitów:

$$|\phi\rangle = \frac{\sqrt{2}}{2}|000\rangle + \frac{\sqrt{2}}{2}|111\rangle$$

Taki stan może być wytworzony ze stanu bazowego  $|000\rangle$  za pomocą poniższego układu bramek elementarnych (Hadamard + dwukrotnie CNot):



# Symulacja układu kwantowego

Stan początkowy  $|000\rangle$  rejestru kwantowego (input):

```
input = transpose(array([[1,0,0,0,0,0,0,0]]))
```

# Symulacja układu kwantowego

Stan początkowy  $|000\rangle$  rejestru kwantowego (input):

```
input = transpose(array([[1,0,0,0,0,0,0,0]]))
```

Macierze ( $8 \times 8$ ) dla trzech poszczególnych etapów obliczeń:

```
stage1 = kron(kron(I, h), I)
```

```
stage2 = kron(I, CNot)
```

```
stage3 = kron(CNot2, I)
```

# Symulacja układu kwantowego

Stan początkowy  $|000\rangle$  rejestru kwantowego (`input`):

```
input = transpose(array([[1, 0, 0, 0, 0, 0, 0, 0]]))
```

Macierze ( $8 \times 8$ ) dla trzech poszczególnych etapów obliczeń:

```
stage1 = kron(kron(I, h), I)
```

```
stage2 = kron(I, CNot)
```

```
stage3 = kron(CNot2, I)
```

Obliczenie macierzy całego układu (`circuit`) i wartości na wyjściu:

```
circuit = dot(dot(stage3, stage2), stage1)
```

```
output = dot(circuit, input)
```

## Rezultat działania programu

Output quantum register state:

```
[[ 0.70710678]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.70710678]]
```

Zatem rezultatem działania programu było otrzymanie wektora stanu:

$$\begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{\sqrt{2}}{2} \end{bmatrix} = \frac{\sqrt{2}}{2} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \frac{\sqrt{2}}{2} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{\sqrt{2}}{2} |000\rangle + \frac{\sqrt{2}}{2} |111\rangle$$

- 1 Python i NumPy - podstawowe informacje
- 2 Wymagania do obliczeń kwantowych
- 3 Numerical Python - najważniejsze funkcje
- 4 Przykład
- 5 **Podsumowanie**

W przedstawiony sposób za pomocą języka Python i biblioteki NumPy mogą być przeprowadzane dowolne obliczenia kwantowe.

Wystarczające do tego są bowiem:

- 1 Macierze unitarne opisujące bramki - i operacje na nich
- 2 Łączenie bramek elementarnych:
  - równoległe - iloczyn tensorowy (`kron`)
  - szeregowo - iloczyn macierzy poszczególnych etapów
- 3 Operacja pomiaru stanu (należy dodatkowo zaimplementować)
- 4 Iloczyn skalarny macierzy całego układu przez wektor stanu wejściowego

- 1 Tutorial do języka Python:  
*<http://docs.python.org/tut/>*
- 2 Dokumentacja biblioteki NumPy  
*<http://numpy.scipy.org/numpy.pdf>*
- 3 Numpy Example List With Doc - przykłady do NumPy:  
*[http://www.scipy.org/Numpy\\_Example\\_List\\_With\\_Doc](http://www.scipy.org/Numpy_Example_List_With_Doc)*

Dziękuję.  
Pytania lub wątpliwości?