Университет ИТМО
Физико-технический мегафакультет
Физический факультет

Группа M32051 К работе допущен

Студент Пантелеев, Артамонов Работа выполнена

Преподаватель Ефремова Е.А. Отчет принят

Рабочий протокол и отчет по задаче от лектора №2

Задание

Пользователь задает начальное положение на кубита на сфере блоха (углы θ и φ), ось вокруг которой будет происходить вращение единичным вектором направления $\vec{n}$ и угол поворота вокруг этой оси α. Программа должна визуализировать начальное положение, заданное пользователем, ось и движение конца вектора состояния по поверхности сферы Блоха и его конечное состояние на сфере Блоха, с указанием вектора состояния.

Решение

Пользователь либо оставляет по умолчанию, либо вводит в консоли следующие параметры:

1) углы θ и φ (начальное положение на сфере)
2) ось, вокруг которой будет происходить поворот (x, y или z)
3) угол поворота α

Получаем оператор поворота кубита на угол относительно оси вектора
Действуем оператором на начальное состояние для получения конечного состояния, отображаем их на сфере Блоха

$$U(\alpha, n) = \exp(-i\frac{\alpha}{2}(n_x\sigma_x + n_y\sigma_y + n_z\sigma_z))$$

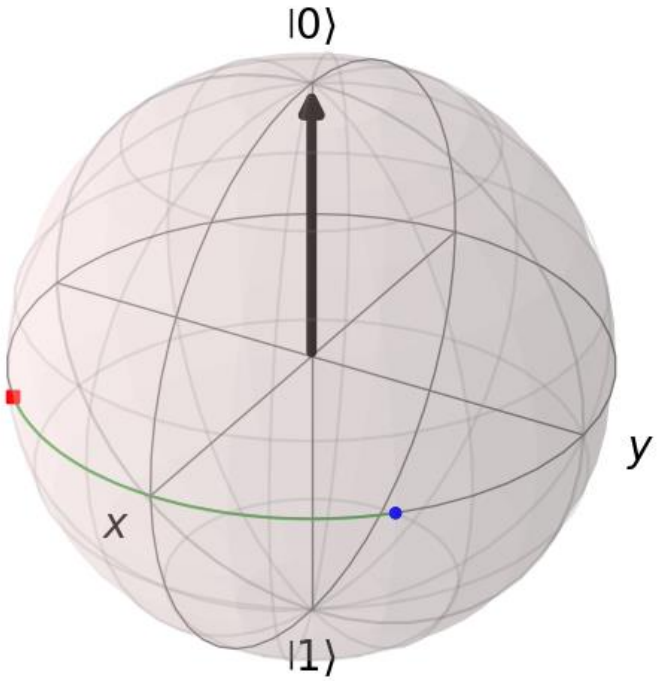
$$|\varphi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

Результат работы программы

```
Числитель для θ:    1
Знаменатель для θ:  2
Числитель для ψ:    1
Знаменатель для ψ:  4
Координатная ось:   Z
Числитель для α:    -1
Знаменатель для α:  2

Начало вектора
θ: 0.5pi
ψ: 0.25pi
Точка: [(0.707+0j), (0.5+0.5j)]
Координаты: (0.707, 0.707, 0.0)

Конец вектора
θ: 0.5pi
ψ: -0.25pi
Точка: [(0.5+0.5j), (0.707+0j)]
Координаты: (0.707, -0.707, 0.0)
```



Исходный код

```
import numpy as np
import qutip
from cmath import phase
from math import cos, sin, acos, e, pi
from scipy.linalg import expm


def pauli_x():
    return np.array([[0, 1], [1, 0]])


def pauli_y():
    return np.array([[0, -1j], [1j, 0]])


def pauli_z():
    return np.array([[1, 0], [0, -1]])


def round_complex(x, precision=10):
    """
    Округляет комплексное число.

    :param x: комплексное число
    :param precision: точность округления
    :return: округленное комплексное число
    """
    return round(x.real, precision) + 1j * round(x.imag,
precision)


def vector_point_to_bloch_angles(vector_points):
    """
    Преобразует точки вектора в углы на сфере Блоха.

    :param vector_points: точки вектора
    :return: углы theta и phi на сфере Блоха
    """
    return 2 * acos(round(abs(vector_points[0]), 12)),
phase(vector_points[1]) - phase(vector_points[0])


def bloch_angles_to_vector_point(theta, phi):
    """
    Преобразует углы на сфере Блоха в точки вектора.

    :param theta: угол theta
```

```
    :param phi: угол phi
    :return: точки вектора
    """
    return np.array([cos(theta / 2), sin(theta / 2) * e ** (1j
* phi)])


def point_coords_by_angles(theta, phi):
    """
    Вычисляет координаты (x, y, z) на сфере Блоха из углов
theta и phi.

    :param theta: угол theta
    :param phi: угол phi
    :return: массив координат (x, y, z)
    """
    x = cos(phi) * sin(theta)
    y = sin(phi) * sin(theta)
    z = cos(theta)
    return np.array([x, y, z])


def point_on_sphere(i, j, r0, R, phi):
    """
    Вычисляет координаты точки на сфере с центром в r0 и
радиусом R.

    :param i: координата i
    :param j: координата j
    :param r0: центр сферы
    :param R: радиус сферы
    :param phi: угол phi
    :return: координаты точки на сфере
    """
    return r0 + R * cos(phi) * i + R * sin(phi) * j


def normalize_vector(v):
    """
    Нормализует вектор.

    :param v: вектор
    :return: нормализованный вектор
    """
    return v / np.linalg.norm(v)


def rotation_trace_points(n, A, B, alpha, total_points=50):
    """
```

```
    Вычисляет точки вращения вокруг оси n между точками A и B.

    :param n: ось вращения
    :param A: начальная точка
    :param B: конечная точка
    :param alpha: угол вращения
    :param total_points: общее количество точек
    :return: список точек вращения
    """
    R = np.linalg.norm((B - A)) / (2 * sin(abs(alpha) / 2))
    j = normalize_vector(np.array(B) - np.array(A))
    if alpha > 0:
        i = np.cross(j, n)
        r0 = B - R * (cos(alpha / 2) * i + sin(alpha / 2) * j)
    else:
        i = np.cross(n, j)
        r0 = A - R * (cos(alpha / 2) * i + sin(alpha / 2) * j)

    angles = np.linspace(-alpha / 2, alpha / 2, total_points)
    points = [point_on_sphere(i, j, r0, R, phi) for phi in
angles]

    return points


def format_angle_for_view(x):
    """
    Форматирует угол для отображения.

    :param x: угол
    :return: отформатированное значение угла
    """
    x = round(x / pi, 10)
    angle = ""
    if x != 1:
        angle += str(x)
    if x != 0:
        angle += "pi"
    return angle


def format_vector_point(vector_state):
    """
    Форматирует состояние вектора для отображения.

    :param vector_state: состояние вектора
    :return: отформатированное состояние вектора
    """
    x1 = round_complex(vector_state[0], 3)
```

```
    x2 = round_complex(vector_state[1], 3)
    return f'[{x1}, {x2}]'


def format_sphere_coords(x, y, z):
    """
    Форматирует координаты (x, y, z) для отображения.

    :param x: координата x
    :param y: координата y
    :param z: координата z
    :return: отформатированные координаты (x, y, z)
    """
    return f'({round(x, 3)}, {round(y, 3)}, {round(z, 3)})'


def get_start_values(input_mode=False):
    if not input_mode:
        print("Числитель для θ:\t1")
        print("Знаменатель для θ:\t2")
        print("Числитель для φ:\t1")
        print("Знаменатель для φ:\t4")
        print("Координатная ось:\tZ")
        print("Числитель для α:\t-1")
        print("Знаменатель для α:\t2")
        return pi / 2, pi / 4, np.array([0, 0, 1]), -1 * pi / 2

    numerator_theta = int(input("Введите числитель для θ:\t"))
    denominator_theta = int(input("Введите знаменатель для
θ:\t"))
    start_theta = pi * numerator_theta / denominator_theta

    numerator_phi = int(input("Введите числитель для φ:\t"))
    denominator_phi = int(input("Введите знаменатель для
φ:\t"))
    start_phi = pi * numerator_phi / denominator_phi

    chosen_axis = int(input("Выберите координатную ось: 0 - x,
1 - y, 2 - z:\t")) % 3
    array = [0, 0, 0]
    array[chosen_axis] = 1
    axis = np.array(array)

    numerator_alpha = int(input("Введите числитель для α:\t"))
    denominator_alpha = int(input("Введите знаменатель для
α:\t"))
    alpha = normalize_vector(pi * numerator_alpha /
denominator_alpha)
```

```
    return start_theta, start_phi, axis, alpha


start_theta, start_phi, axis, alpha =
get_start_values(input_mode=False)
nx, ny, nz = axis[0], axis[1], axis[2]
U = expm(-1j * alpha / 2 * (nx * pauli_x() + ny * pauli_y() +
nz * pauli_z()))

start_vector_point = bloch_angles_to_vector_point(start_theta,
start_phi)
end_vector_point = np.dot(U, start_vector_point)

theta, phi = vector_point_to_bloch_angles(start_vector_point)
start_point = point_coords_by_angles(theta, phi)

print()
print('Начало вектора')
print(f'θ: {format_angle_for_view(theta)}')
print(f'φ: {format_angle_for_view(phi)}')
print(f'Точка: {format_vector_point(start_vector_point)}')
print(f'Координаты: {format_sphere_coords(*start_point)}')

theta, phi = vector_point_to_bloch_angles(end_vector_point)
end_point = point_coords_by_angles(theta, phi)

print()
print('Конец вектора')
print(f'θ: {format_angle_for_view(theta)}')
print(f'φ: {format_angle_for_view(phi)}')
print(f'Точка: {format_vector_point(end_vector_point)}')
print(f'Координаты: {format_sphere_coords(*end_point)}')

b = qutip.Bloch()
b.vector_color = ['#000000']
b.vector_width = 5
b.point_color = ['b', 'r', '#00FF00']
b.add_points(start_point)
b.add_points(end_point)
b.add_vectors(axis)
if not np.array_equal(start_point, end_point):
    rotation_points = rotation_trace_points(axis, start_point,
end_point, alpha)
    b.add_points(list(zip(*rotation_points)), meth='l')
b.show()
```

Исходный код опубликован на GitHub и может быть скачан и запущен:
https://github.com/SmartOven/Python/blob/master/Physics/LecTask2/main.py