

TASK 1

```
def print_matrix(matrix, name, k=1)
```

Этот код предназначен для удобной и структурированной печати матрицы с названием и коэффициентом k , который используется для отображения масштабирования. Разберем каждую часть подробно:

1. Аргументы функции:

- `matrix` — матрица, представляемая как список списков (двумерный массив), где каждый внутренний список — строка матрицы.
 - `name` — строка, представляющая название матрицы (например, AAA, BBB).
 - `k` — коэффициент, умножающий матрицу, по умолчанию равен 1.
-

2. Общий формат вывода: и так понятно, что выводит

3. Вывод имени матрицы и коэффициента:

```
print(f"{name} = {k} * [", end="")
```

- Выводится название матрицы (`name`), коэффициент k , и начинается скобка (`[`) для обозначения начала матрицы.
 - `end=""` предотвращает автоматический переход на новую строку, так что следующая строка начнется сразу после этого.
-

4. Определение ширины для выравнивания чисел:

```
max_len = max(len(f"{elem:6.2f}") for row in matrix for elem in row)
```

- Для корректного отображения всех элементов в столбцах выравнивание чисел производится по максимальной длине элемента.
 - Каждый элемент форматируется в строку с точностью до двух знаков после запятой (`{elem:6.2f}`), что означает:
 - Ширина строки элемента будет минимум 6 символов (включая знаки, точку, и десятичные знаки).
 - `max()` находит максимальную длину среди всех элементов матрицы.
-

5. Перебор строк матрицы и печать:

```
for i, row in enumerate(matrix):
```

- Используется цикл `for` с индексом строки (`i`) и самой строкой (`row`).

5.1 Выравнивание последующих строк:

```
if i > 0:
    print(" " * (len(name) + 8 + (2 if not isinstance(k, int) else 0)),
          end="")
```

- Для строк, кроме первой (`i > 0`), добавляется дополнительный отступ, чтобы выровнять строки под названием матрицы.
- `len(name) + 8` вычисляет длину отступа:
 - `len(name)` — длина имени матрицы.
 - `8` — длина символов = `k * [` (учитывая пробелы и `k`).
 - Если `k` не является целым числом, добавляется ещё два символа отступа (`2 if not isinstance(k, int)`).

5.2 Печать строки матрицы:

```
print("    ".join(f"{elem:{max_len}.2f}" for elem in row), end=" ")
```

- Каждый элемент строки форматируется с точностью до двух знаков и заданной шириной `max_len`.
 - Все элементы объединяются через 3 пробела (`" ".join(...)`), чтобы выровнять их.
-

6. Заккрытие матрицы:

```
if i == len(matrix) - 1:
    print(" ]")
```

- Если это последняя строка матрицы, функция добавляет закрывающую квадратную скобку `]`.
-

7. Добавление пустой строки после матрицы:

```
print()
```

- После завершения печати матрицы добавляется пустая строка для разделения вывода, чтобы было удобнее читать.

```
def print_vector(vector, name, k=1)
```

Этот код представляет функцию для форматированного вывода вектора с указанием его имени и коэффициента масштабирования k . Вот подробное объяснение, как она работает:

1. Аргументы функции

- `vector` — одномерный массив (список), содержащий элементы вектора.
- `name` — строка, представляющая название вектора (например, `vvv`, `uuu`).
- `k` — коэффициент, на который умножается вектор. По умолчанию равен 1.

Пример вызова:

```
print_vector([1, 2.5, 3.14159], "v", k=2)
```

2. Печать заголовка:

```
print(f"{name} = {k} * [", ...)
```

- Формируется начало строки:
 - `{name}` — название вектора (например, `"v"`).
 - `k` — коэффициент, указывающий масштабирование.
 - `[` — открывающая квадратная скобка для обозначения начала вектора.
 - Пример: если `name=v` и `k=2`, вывод начнётся с `v = 2 * [`.
-

3. Форматирование элементов вектора:

```
" ".join(f"{elem:.2f}" for elem in vector)
```

- Каждый элемент вектора форматируется в строку с точностью до двух знаков после запятой (`{elem:.2f}`):
 - `.2f` означает, что элемент будет представлен как число с двумя знаками после запятой (или точки, в зависимости от локали).
 - Если число имеет больше десятичных знаков, оно округляется. Например:
 - `3.14159` → `3.14`
 - `2.5` → `2.50`
 - Форматированные элементы объединяются через два пробела (`" ".join(...)`):
 - Это создаёт аккуратный, читаемый вид, где элементы отделены равномерно.
 - Пример: для `vector = [1, 2.5, 3.14159]` результатом будет строка `"1.00 2.50 3.14"`.
-

4. Закрывающая часть строки:

"] \n "

- После элементов вектора добавляется закрывающая квадратная скобка] и символ новой строки (\n), чтобы отделить вывод от следующих строк.
-

5. Общий результат вызова функции

Объединяя все части, функция печатает строку вида:

`v = 2 * [1.00 2.50 3.14]`

где:

- `v` — имя вектора.
- `2` — коэффициент масштабирования.
- `[1.00 2.50 3.14]` — содержимое вектора в заданном формате.

```
def matrix_multiplication(A, B, k_A, k_B)
```

Тут простая формула умножения матрицы на матрицу. Сначала заполняем её нулями, потом применяем формулу:

$$C_{ij} = \sum_{n=1}^k A_{in} * B_{nj}$$

Ну и ещё коэффициенты k_A и k_B используются и всё

```
def matrix_vector_multiplication(A, v, k_A=1, k_v=1)
```

Шаги такие же, как и в прошлом пункте. Просто оставлю здесь формулу

$$AB = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_{1n} \end{pmatrix} = \begin{pmatrix} a_{11} \times b_1 + a_{12} \times b_2 + \dots + a_{1n} \times b_n \\ a_{21} \times b_1 + a_{22} \times b_2 + \dots + a_{2n} \times b_n \\ \dots & \dots & \dots & \dots \\ a_{m1} \times b_1 + a_{m2} \times b_2 + \dots + a_{mn} \times b_n \end{pmatrix}$$
$$= \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_{1m} \end{pmatrix}$$

```
def tensor_product_matrixes(A, B, k_A=1, k_B=1)
```

Этот код реализует вычисление тензорного произведения (или произведения Кронекера) двух матриц A и B с учётом масштабирующих коэффициентов k_A и k_B . Ниже приведено подробное объяснение.

1. Аргументы функции:

- A — первая матрица, заданная в виде двумерного списка (список списков).
- B — вторая матрица, также заданная как двумерный список.
- k_A — коэффициент масштабирования элементов матрицы A . По умолчанию равен 1.
- k_B — коэффициент масштабирования элементов матрицы B . По умолчанию равен 1.

2. Идея тензорного произведения:

Тензорное произведение двух матриц A размером $m \times n$ и B размером $p \times q$ — это новая матрица C размером $(m \cdot p) \times (n \cdot q)$. Каждый элемент C вычисляется как произведение элементов A и B с учётом их индексов.

Элемент $C[i \cdot p + j][k \cdot q + l]$ задаётся формулой:

$$C[i \cdot p + j][k \cdot q + l] = k_A \cdot A[i][k] \cdot k_B \cdot B[j][l]$$

3. Структура функции:

3.1 Инициализация результирующей матрицы:

```
result = []
```

- Создаётся пустой список для хранения строк результирующей матрицы.

3.2 Два внешних цикла для строк:

```
for i in range(len(A)): # Перебираются строки матрицы A
    for j in range(len(B)): # Перебираются строки матрицы B
```

- Внешний цикл проходит по строкам A (индекс i).
- Вложенный цикл проходит по строкам B (индекс j).
- Для каждой строки $A[i]$ и $B[j]$ будет сформирована часть результирующей строки.

3.3 Инициализация строки:

```
row = []
```

- Для каждой пары строк из $A[i]$ и $B[j]$ создаётся пустая строка результирующей матрицы.

3.4 Два внутренних цикла для столбцов:

```
for k in range(len(A[0])): # Перебираются столбцы матрицы A
    for l in range(len(B[0])): # Перебираются столбцы матрицы B
```

- Внутренние циклы отвечают за формирование элементов результирующей строки.
- k и l — индексы столбцов матриц A и B соответственно.

3.5 Вычисление элемента:

```
row.append(k_A * A[i][k] * k_B * B[j][l])
```

- Каждый элемент результирующей строки вычисляется как произведение:
 $k_A \cdot A[i][k] \cdot k_B \cdot B[j][l]$
- Этот элемент добавляется в текущую строку `row`.

3.6 Добавление строки в результирующую матрицу:

```
result.append(row)
```

- После завершения внутреннего цикла (перебора k и l) строка `row` добавляется в матрицу `result`.

3.7 Возврат результата:

```
return result
```

- Возвращается сформированная матрица C , представляющая тензорное произведение матриц A и B .

```
def tensor_product_vectors(v, w, k_v=1, k_w=1)
```

Этот код реализует **тензорное произведение двух векторов** v и w с учётом коэффициентов масштабирования k_v и k_w . Тензорное произведение векторов — это операция, создающая новый вектор, где каждый элемент является произведением элементов исходных векторов.

1. Аргументы функции:

- v — первый вектор (список чисел).
 - w — второй вектор (список чисел).
 - k_v — коэффициент масштабирования для элементов вектора v . По умолчанию равен 1.
 - k_w — коэффициент масштабирования для элементов вектора w . По умолчанию равен 1.
-

2. Идея тензорного произведения:

Тензорное произведение двух векторов v размером n и w размером m создаёт новый вектор $result$ длиной $n \cdot m$, где каждый элемент вычисляется по формуле:

$$result[i \cdot m + j] = k_v \cdot v[i] \cdot k_w \cdot w[j]$$

То есть, берётся каждый элемент из v , умножается на каждый элемент из w , и результат добавляется в новый вектор.

3. Структура функции:

3.1 Инициализация результирующего вектора:

```
result = []
```

- Создаётся пустой список `result`, который будет хранить результаты произведений.


3.2 Два вложенных цикла:

```
for i in range(len(v)): # Перебор элементов вектора v
    for j in range(len(w)): # Перебор элементов вектора w
```

- Внешний цикл проходит по каждому элементу $v[i]$
- Внутренний цикл проходит по каждому элементу $w[j]$
- Для каждой пары $v[i]$ и $w[j]$ вычисляется их произведение с учётом коэффициентов k_v и k_w .

3.3 Вычисление элемента тензорного произведения:

python

 Копировать код

```
result.append(k_v * v[i] * k_w * w[j])
```

- Каждый элемент результирующего вектора вычисляется по формуле:

$$\text{Элемент} = k_v \cdot v[i] \cdot k_w \cdot w[j]$$

- Результат добавляется в конец списка `result` с помощью `append()`.

4. Возврат результата:

```
return result
```

- После завершения всех циклов функция возвращает список `result`, который содержит элементы тензорного произведения.

```
def vector_to_bracket_notation(v, k=1)
```

Этот код преобразует вектор v в **скобочную нотацию** (bracket notation), часто используемую в квантовой механике и квантовых вычислениях, с учётом масштабирующего коэффициента k .

Давайте разберём этот код подробно.

1. Что делает функция?

Функция принимает два аргумента:

- v — вектор (список), элементы которого интерпретируются как амплитуды квантового состояния.
- k — коэффициент масштабирования для элементов вектора. По умолчанию равен 1.

Функция преобразует вектор v в строку, представляющую квантовое состояние в скобочной нотации (например, $0.71|0\rangle + 0.71|1\rangle$ для двухкубитной системы).

2. Структура функции

2.1 Обработка случая для двух состояний:

```
if len(v) == 2:  
    return f"{k * v[0]:.2f}|0> + {k * v[1]:.2f}|1>"
```

Если вектор v имеет длину 2 (то есть описывает одно кубитное состояние):

- Возвращается строка, где:
 - $v[0]$ ассоциируется с состоянием $|0\rangle$.
 - $v[1]$ ассоциируется с состоянием $|1\rangle$.
- Каждая амплитуда форматируется с точностью до двух знаков после запятой ($:.2f$).

Пример:

```
vector_to_bracket_notation([0.71, 0.71])
```

Вывод:

```
0.71|0> + 0.71|1>
```

2.2 Обработка случая для произвольного числа состояний:

Если длина v больше 2, то это означает, что система может быть многокубитной (или многосостояний):

```
n = len(v)
num_qubits = (n - 1).bit_length()
```

- n — длина вектора v , равная количеству квантовых состояний.
- num_qubits вычисляется как количество кубитов, необходимое для представления n состояний.

Пример:

```
v = [1, 0, 0, 0]
n = len(v) # 4
num_qubits = (n - 1).bit_length() # 2 кубита
```

2.3 Создание строки в скобочной нотации:

python

 Копировать код

```
notation = "" + ".join(
    f"{k * k_v:.2f}|{format(i, f'0{num_qubits}b')})"
    for i, k_v in enumerate(v)
    if k * k_v != 0
)
```

- Цикл `for i, k_v in enumerate(v):`
 - Перебираются индексы i и значения k_v вектора v .
- Масштабирование амплитуды: $k \cdot k_v$.
 - Если $k \cdot k_v \neq 0$, то добавляем в строку.
 - Нулевые амплитуды исключаются.
- Формат `|{format(i, f'0{num_qubits}b')})`:
 - `format(i, f'0{num_qubits}b')` преобразует индекс i в двоичную строку с ведущими нулями.
 - Например, для $num_qubits = 3$ индексы 0, 1, 2, 3 преобразуются в 000, 001, 010, 011.

Амплитуды форматируются с точностью до двух знаков (`:.2f`).

3. Пример работы функции

3.1 Пример с двумя состояниями:

```
v = [0.6, 0.8]
print(vector_to_bracket_notation(v))
```

Вывод:

```
0.60|0> + 0.80|1>
```

3.2 Пример с четырьмя состояниями:

```
v = [1, 0, 0, 0]
print(vector_to_bracket_notation(v))
```

Шаги:

- Длина n=4, значит num_qubits=2.
- Скобочная нотация:
 - 1.00|00>, так как амплитуда первого состояния равна 1.
 - Остальные состояния имеют амплитуду 0 и не включаются.

Вывод:

```
1.00|00>
```

3.3 Пример с масштабированием:

```
v = [1, 0, 0, 0]
k = 2
print(vector_to_bracket_notation(v, k))
```

Вывод:

```
2.00|00>
```

3.4 Пример с нулями и несколькими состояниями:

```
v = [0.5, 0, 0.5, 0]
print(vector_to_bracket_notation(v))
```

Шаги:

- n=4, значит num_qubits=2.
- Нотация:
 - 0.50|00> для первого состояния.
 - 0.50|10> для третьего состояния (индекс 2).

Вывод:

```
0.50|00> + 0.50|10>
```

TASK 2

```
def measure_in_standard_basis(state)
```

Этот код реализует функцию измерения кубита в стандартном базисе ($|0\rangle$ и $|1\rangle$). В квантовых вычислениях измерение кубита приводит квантовое состояние в одно из классических состояний (0 или 1) с вероятностями, зависящими от амплитуд квантового состояния.


1. Что делает функция?

- Функция принимает на вход квантовое состояние кубита $state = [\alpha, \beta]$, где:
 - α и β — комплексные числа, амплитуды вероятностей для состояний $|0\rangle$ и $|1\rangle$, соответственно.
 - Возвращает:
 1. Результат измерения (0 или 1).
 2. Вероятности для каждого из состояний ($[p_0, p_1]$).
-

2. Как работает функция?

2.1 Вычисление вероятностей:

python

 Копировать код

```
probabilities = [round(abs(state[0])**2, 2), round(abs(state[1])**2, 2)]
```

Каждая из амплитуд α и β связана с вероятностью наблюдения состояния через квадрат их модуля:

$$p_0 = |\alpha|^2, \quad p_1 = |\beta|^2$$


- `abs(state[0])` : Берёт модуль комплексного числа α для состояния $|0\rangle$.
- `**2` : Возводит модуль в квадрат, что соответствует вероятности.
- `round(..., 2)` : Округляет результат до двух знаков после запятой для удобства.

Пример: Если `state = [0.6, 0.8]`, то:

$$p_0 = |0.6|^2 = 0.36, \quad p_1 = |0.8|^2 = 0.64$$

Значит:

python


 Копировать код

```
probabilities = [0.36, 0.64]
```



2.2 Симуляция измерения:

python

 Копировать код

```
result = np.random.choice([0, 1], p=probabilities)
```


- `np.random.choice([0, 1], p=probabilities)`:
 - Это метод из библиотеки **NumPy**, который случайно выбирает одно значение из списка $[0, 1]$, учитывая переданные вероятности (p_0 и p_1).
 - Вероятность того, что будет выбрано 0, равна p_0 , а вероятность выбора 1 — p_1 .

Пример: Если `probabilities = [0.36, 0.64]`, то:

- 0 выбирается с вероятностью 36%.
- 1 выбирается с вероятностью 64%.

2.3 Возврат результата:

python

 Копировать код


```
return result, probabilities
```

- Функция возвращает:
 1. `result`: результат измерения (0 или 1).
 2. `probabilities`: список вероятностей для состояний $|0\rangle$ и $|1\rangle$.
-

3. Пример работы функции

Пример 1: Равные вероятности ($|\alpha|^2 = |\beta|^2 = 0.5$):

python

 Копировать код

```
import numpy as np

state = [1/np.sqrt(2), 1/np.sqrt(2)] # Состояние: [0.707, 0.707]
result, probabilities = measure_in_standard_basis(state)


print("Результат измерения:", result)
print("Вероятности:", probabilities)
```

Объяснение:

- $|\alpha|^2 = |0.707|^2 \approx 0.5$.
- $|\beta|^2 = |0.707|^2 \approx 0.5$.
- $probabilities = [0.5, 0.5]$: 0 и 1 равновероятны.
- Результат будет случайным (либо 0, либо 1) с вероятностью 50%.

Пример 2: Неравные вероятности:

python

 Копировать код

```
state = [0.6, 0.8] # Состояние: [0.6, 0.8]
result, probabilities = measure_in_standard_basis(state)


print("Результат измерения:", result)
print("Вероятности:", probabilities)
```

Объяснение:

- $|\alpha|^2 = |0.6|^2 = 0.36$.
- $|\beta|^2 = |0.8|^2 = 0.64$.
- $probabilities = [0.36, 0.64]$: вероятность 1 выше (64%).

Вывод:

less

 Копировать код

```
Результат измерения: 1
Вероятности: [0.36, 0.64]
```



4. Особенности кода

1. Квантовая механика:

- В квантовых системах вероятности определяются квадратом модуля амплитуды.
- Измерение разрушает квантовое состояние, оставляя систему в классическом состоянии (0 или 1).

2. NumPy для случайного выбора:

- `np.random.choice` моделирует вероятностное измерение. Это важно для симуляции реального поведения квантовых систем.

3. Округление:


- Вероятности округляются до двух знаков. Это полезно для улучшения читаемости, хотя в реальных приложениях лучше избегать округления (чтобы сохранить точность).

```
def measure_in_rotated_basis(state)
```

Этот код реализует измерение кубита в повернутом базисе, используя преобразование Адамара (H). Повернутый базис определяется матрицей Адамара и позволяет производить измерение относительно альтернативного набора состояний, таких как $|+\rangle$ и $|-\rangle$ вместо стандартных $|0\rangle$ и $|1\rangle$.

1. Матрица Адамара (H):

python

 Копировать код

```
H = np.array([[1 / np.sqrt(2), 1 / np.sqrt(2)],  
              [1 / np.sqrt(2), -1 / np.sqrt(2)]])
```

- Это матрица Адамара, которая выполняет линейное преобразование кубита, преобразуя стандартный базис $|0\rangle, |1\rangle$ в повернутый базис $|+\rangle, |-\rangle$:

$$H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad H|1\rangle = |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

- В стандартном базисе, матрица H записывается как:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

2. Функция `measure_in_rotated_basis`:

Функция измеряет кубит `state` в повернутом базисе с использованием преобразования Адамара.

Входные данные:

- `state`: квантовое состояние $([\alpha, \beta])$, где:
 - α и β — амплитуды состояний $|0\rangle$ и $|1\rangle$, соответственно.


Выходные данные:

- Результат измерения (0 или 1).
- Вероятности измерения в повернутом базисе $([p_0, p_1])$.

3. Пошаговый разбор функции

3.1 Преобразование состояния в повёрнутый базис:

python

 Копировать код

```
rotated_state = np.dot(H, state)
```

- `np.dot(H, state)` вычисляет матричное произведение:

$$\text{rotated_state} = H \cdot \text{state}.$$

- Это переводит вектор состояния $\text{state} = [\alpha, \beta]$ из стандартного базиса $(|0\rangle, |1\rangle)$ в повёрнутый базис $(|+\rangle, |-\rangle)$.


Пример: Если $\text{state} = [1/\sqrt{2}, 1/\sqrt{2}]$, то:

$$H \cdot \text{state} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Это соответствует состоянию $|+\rangle$.

3.2 Вычисление вероятностей:

python

 Копировать код

```
probabilities = [round(abs(rotated_state[0])**2, 2), round(abs(rotated_state[1])**2, 2)]
```

- Каждая вероятность (p_0 и p_1) связана с квадратом модуля соответствующего элемента `rotated_state`:

$$p_0 = |r_0|^2, \quad p_1 = |r_1|^2,$$

где r_0 и r_1 — амплитуды состояний $|+\rangle$ и $|-\rangle$.


- `abs(rotated_state[0])**2`: Вычисляет квадрат модуля амплитуды состояния $|+\rangle$.
- `abs(rotated_state[1])**2`: Вычисляет квадрат модуля амплитуды состояния $|-\rangle$.
- `round(..., 2)`: Округляет вероятности до двух знаков.

Пример: Если `rotated_state = [1, 0]`, то:

$$p_0 = |1|^2 = 1.0, \quad p_1 = |0|^2 = 0.0.$$

3.3 Симуляция измерения:

python

 Копировать код

```
result = np.random.choice([0, 1], p=probabilities)
```


- `np.random.choice([0, 1], p=probabilities)`:
 - Случайно выбирает 0 или 1, основываясь на вероятностях p_0 и p_1 .

Пример: Если `probabilities = [0.36, 0.64]`, то:

- 0 выбирается с вероятностью 36%.
- 1 выбирается с вероятностью 64%.

3.4 Возврат результата:

python

 Копировать код


```
return result, probabilities
```

- Возвращает:
 1. `result`: результат измерения (0 или 1).
 2. `probabilities`: вероятности для состояний $|+\rangle$ и $|-\rangle$.

4. Пример работы функции

Пример 1: Кубит в состоянии $|0\rangle$:

python

 Копировать код

```
import numpy as np

state = [1, 0] # |0>
result, probabilities = measure_in_rotated_basis(state)

print("Результат измерения:", result)
print("Вероятности:", probabilities)
```

Объяснение:

- Состояние $\text{state} = |0\rangle$ преобразуется в повернутый базис:

$$H \cdot [1, 0] = \frac{1}{\sqrt{2}}[1, 1].$$

- Амплитуды в повернутом базисе: $r_0 = 1/\sqrt{2}, r_1 = 1/\sqrt{2}$.


- Вероятности:

$$p_0 = |1/\sqrt{2}|^2 = 0.5, \quad p_1 = |1/\sqrt{2}|^2 = 0.5.$$

- Измерение случайно возвращает 0 или 1 с равной вероятностью 50%.

Вывод:


less

 Копировать код

```
Результат измерения: 1
Вероятности: [0.5, 0.5]
```

Пример 2: Кубит в состоянии $|+\rangle$:

python

 Копировать код

```
state = [1/np.sqrt(2), 1/np.sqrt(2)] # |+⟩
result, probabilities = measure_in_rotated_basis(state)


print("Результат измерения:", result)
print("Вероятности:", probabilities)
```

Объяснение:

- $\text{state} = |+\rangle$, и применение H переводит его обратно в $|0\rangle$.
- Вероятности:
$$p_0 = 1.0, \quad p_1 = 0.0.$$
- Измерение всегда возвращает 0.

Вывод:

less

 Копировать код

```
Результат измерения: 0
Вероятности: [1.0, 0.0]
```



5. Ключевые моменты

1. **Матрица Адамара** переводит состояния между стандартным ($|0\rangle, |1\rangle$) и повернутым ($|+\rangle, |-\rangle$) базисами.
2. **Измерение в повернутом базисе** моделируется с учётом вероятностей для состояний $|+\rangle$ и $|-\rangle$.
3. **NumPy**: Используется для линейной алгебры (`np.dot`) и генерации случайных выборов (`np.random.choice`).

TASK 3

```
def measure_quantum_register(state)
```

Этот код моделирует процесс измерения **квантового регистра**. Квантовый регистр — это система из n -кубитов, которая может находиться в суперпозиции 2^n -состояний. Результат измерения — одно из этих состояний, выбираемое случайным образом на основе вероятностей, определённых квадратами амплитуд.


Подробный разбор

1. Входные данные

- `state` : Список комплексных чисел, представляющий квантовое состояние регистра. Каждое число — это амплитуда соответствующего состояния (например, $|000\rangle, |001\rangle, \dots, |111\rangle$).

2. Вычисление вероятностей

python

 Копировать код

```
probabilities = [round(abs(amplitude)**2, 2) for amplitude in state]
```

- Каждая амплитуда в квантовом состоянии — это **комплексное число** (α_i), где квадрат его модуля ($|\alpha_i|^2$) соответствует вероятности того, что при измерении регистр окажется в состоянии i .
- `abs(amplitude)**2` : Вычисляет квадрат модуля амплитуды ($|\alpha_i|^2$).
- `round(..., 2)` : Округляет вероятность до двух десятичных знаков, чтобы избежать численных ошибок.

Пример: Пусть квантовый регистр имеет состояние:


$$\text{state} = [0.6 + 0j, 0.8 + 0j].$$

Тогда вероятности будут:

$$\text{probabilities} = [|0.6|^2, |0.8|^2] = [0.36, 0.64].$$

3. Нормализация вероятностей

python

 Копировать код

```
probabilities = [p / sum(probabilities) for p in probabilities]
```


- Если из-за округлений сумма вероятностей ($\sum_i |\alpha_i|^2$) не равна 1, то её нормализуют, деля каждую вероятность на общую сумму.
- Это гарантирует, что вероятности остаются корректными и суммируются в 1.

Пример: Если $\text{probabilities} = [0.36, 0.63]$ (сумма 0.99), то нормализация приведёт их к:

$$\text{probabilities} = \left[\frac{0.36}{0.99}, \frac{0.63}{0.99} \right] \approx [0.36, 0.64].$$

4. Измерение

python

 Копировать код

```
result = np.random.choice(range(len(state)), p=probabilities)
```


- `np.random.choice(range(len(state)), p=probabilities)`:
 - Генерирует случайное число i из диапазона $[0, 1, \dots, N - 1]$, где $N = \text{len}(\text{state})$.
 - Выбор делается на основе вероятностей probabilities , где вероятность выбора i -го состояния равна p_i .

Пример: Если $\text{probabilities} = [0.36, 0.64]$, то:

- 0 выбирается с вероятностью 36%,
- 1 выбирается с вероятностью 64%.

5. Возврат результата

python

 Копировать код


```
return result, probabilities
```

- `result` : Индекс состояния, выбранного в результате измерения.
- `probabilities` : Список вероятностей всех состояний.

Пример работы функции

Пример 1: Двухкубитный регистр

python

 Копировать код

```
import numpy as np

state = [0.6 + 0j, 0.8 + 0j] # Квантовый регистр в состоянии суперпозиции
result, probabilities = measure_quantum_register(state)


print("Результат измерения:", result)
print("Вероятности:", probabilities)
```

Разбор:

1. Амплитуды: $[0.6, 0.8]$.
2. Вероятности: $[0.36, 0.64]$.
3. Измерение:
 - Результат 0: вероятность 36%.
 - Результат 1: вероятность 64%.

Пример вывода:

less


 Копировать код

```
Результат измерения: 1
Вероятности: [0.36, 0.64]
```



Пример 2: Трёхкубитный регистр

python

 Копировать код

```
state = [0.5 + 0j, 0.5 + 0j, 0.5 + 0j, 0.5 + 0j, 0 + 0j, 0 + 0j, 0 + 0j, 0 + 0j]
result, probabilities = measure_quantum_register(state)


print("Результат измерения:", result)
print("Вероятности:", probabilities)
```

Разбор:

1. Состояние содержит 8 амплитуд ($2^3 = 8$).
2. Ненулевые амплитуды: $[0.5, 0.5, 0.5, 0.5]$.
3. Вероятности: $[0.25, 0.25, 0.25, 0.25, 0, 0, 0, 0]$.
4. Измерение:
 - Результат 0, 1, 2, или 3: вероятность 25% для каждого.

Пример вывода:

less

 Копировать код

```
Результат измерения: 2
Вероятности: [0.25, 0.25, 0.25, 0.25, 0.0, 0.0, 0.0, 0.0]
```

Ключевые моменты

1. **Суперпозиция:** Квантовый регистр может находиться в состоянии суперпозиции, и измерение "разрушает" её, выбирая одно из возможных состояний.
2. **Вероятности:** Вероятность каждого результата определяется квадратом модуля амплитуды состояния.
3. **Нормализация:** Гарантирует корректность вероятностей.
4. **Случайность:** Результат измерения выбирается случайным образом с учётом рассчитанных вероятностей.

TASK 4

```
def measure_single_qubit(state, qubit_index)
```

Эта функция моделирует измерение одного кубита в квантовом регистре, состоящем из нескольких кубитов. В результате измерения состояние регистра изменяется (так называемое "схлопывание"), и результатом является 0 или 1, в зависимости от состояния измеряемого кубита.

Подробный разбор

1. Входные параметры

- `state` : Вектор комплексных чисел, представляющий квантовое состояние n -кубитного регистра (размер 2^n).
 - Каждый индекс в векторе соответствует одному из 2^n возможных состояний (например, $|000\rangle, |001\rangle, \dots, |111\rangle$).
 - Каждое число — это амплитуда состояния.
 - `qubit_index` : Индекс кубита, который нужно измерить (от 0 до $n - 1$).
-

Шаги выполнения:

1. Вычисление вероятностей для 0 и 1 на указанном кубите:

Для того чтобы вычислить вероятность получения 0 или 1 на выбранном кубите, мы будем проходить по всем состояниям квантового регистра и учитывать, какой бит соответствует кубиту, который мы измеряем.

Вероятность для 0 (`probability_0`):

```
python Копировать код  
  
probability_0 = sum(abs(amplitude)**2 for i, amplitude in enumerate(state) if not (i >> qubit_index & 1))
```

- **Шаг 1:** Для каждого состояния (индекса i в векторе `state`) проверяется, равен ли бит на позиции `qubit_index` нулю. Для этого используется побитовая операция:
 - $(i \gg \text{qubit_index})$ — сдвигает битовое представление числа i вправо на qubit_index позиций.
 - $\& 1$ — оставляет только младший бит, который указывает, равен ли выбранный кубит 0 или 1.
- **Шаг 2:** Если выбранный бит равен 0, то амплитуда $|a_i|^2$ для этого состояния добавляется к вероятности 0.


Вероятность для 1 (`probability_1`):

```
python Копировать код  
  
probability_1 = sum(abs(amplitude)**2 for i, amplitude in enumerate(state) if (i >> qubit_index) & 1)
```

- **Шаг 1:** Аналогично предыдущему случаю, но проверяется, равен ли бит на позиции `qubit_index` единице.
- **Шаг 2:** Если выбранный бит равен 1, то амплитуда для этого состояния добавляется к вероятности 1.

2. Нормализация вероятностей:

python


 Копировать код

```
probability_0 = round(probability_0, 2)
probability_1 = round(probability_1, 2)
```

- После вычисления вероятностей для 0 и 1, они округляются до двух знаков после запятой для точности и удобства.
 - Эти вероятности будут использоваться для вычисления вероятности измерения каждого возможного результата.
-
-

3. Измерение кубита:

python

 Копировать код

```
result = np.random.choice([0, 1], p=probabilities)
```


- С помощью функции `np.random.choice` случайным образом выбирается результат измерения кубита:
 - Если вероятность 0 выше, вероятностью того, что результат будет 0, больше.
 - Если вероятность 1 выше, результатом будет 1.
 - Функция использует распределение вероятностей, переданное в параметре `p=probabilities`, где `probabilities = [probability_0, probability_1]`.
-

4. Схлопывание регистра (обновление состояния):

После того как результат измерения выбран, мы обновляем квантовое состояние, исключая все те состояния, которые не соответствуют результату измерения.

Инициализация нового состояния:

python


 Копировать код

```
new_state = np.zeros_like(state)
```

- Создаём новый вектор состояния, который будет содержать новое состояние после измерения.

Обновление нового состояния:

python

 Копировать код

```
for i in range(len(state)):
    if ((i >> qubit_index) & 1) == result:
        new_state[i] = state[i] / np.sqrt(probabilities[result]) if probabilities[result]
```

- Для каждого состояния проверяется, соответствует ли результат измерения (0 или 1) состоянию кубита на позиции `qubit_index`. Это делается с помощью операции `(i >> qubit_index) & 1`.
- Если результат измерения совпадает с состоянием кубита, амплитуда этого состояния сохраняется в новое состояние.
 - Амплитуда нормализуется, делясь на корень из вероятности того, что кубит был измерен в этом состоянии. Это нужно для того, чтобы новое состояние было корректно нормализовано.

Округление амплитуд:

python


 Копировать код

```
new_state = np.round(new_state, 4)
```

- После обновления состояния, амплитуды в новом состоянии округляются до 4 знаков после запятой, чтобы избежать численных ошибок.
-

5. Возврат результата:

python

 Копировать код


```
return result, new_state, probabilities
```

- Функция возвращает:
 1. `result` — результат измерения кубита (0 или 1).
 2. `new_state` — новое состояние регистра после схлопывания.
 3. `probabilities` — вероятности получения 0 и 1 на измеренном кубите.
-

Пример работы:

Предположим, что у нас есть двухкубитный регистр, который находится в состоянии $|00\rangle$:

python

 Копировать код

```
import numpy as np


state = [0.6 + 0j, 0.8 + 0j, 0 + 0j, 0 + 0j] # |00>: 0.6, |01>: 0.8
result, new_state, probabilities = measure_single_qubit(state, qubit_index=0)

print("Результат измерения:", result)
print("Новое состояние:", new_state)
print("Вероятности:", probabilities)
```

- Регистры $|00\rangle$ и $|01\rangle$ имеют амплитуды 0.6 и 0.8, соответственно. Мы измеряем первый кубит.
- Вероятность получения 0 на первом кубите будет $0.6^2 + 0.8^2 = 1$, а вероятность получения 1 на первом кубите — 0.
- Система остаётся в исходном состоянии, если измерение даёт 0, и схлопнется в состояние $|00\rangle$.

Пример вывода:

less

 Копировать код

```
Результат измерения: 0
Новое состояние: [0.6+0.j 0.8+0.j 0. +0.j 0. +0.j]
Вероятности: [1.0, 0.0]
```

Резюме:

1. **Измерение кубита** осуществляется путём вычисления вероятностей для получения 0 или 1 на выбранном кубите, основываясь на амплитудах всех состояний в квантовом регистре.
2. После измерения **состояние регистра схлопывается** — исключаются состояния, несовместимые с результатом измерения, и происходит нормализация амплитуд.
3. Функция возвращает результат измерения, новое состояние и вероятности для каждого результата.


```
def state_to_bracket_notation(state)
```

Этот код выполняет преобразование квантового состояния в **бра-кет нотацию**. Бра-кет нотация — это способ представления квантовых состояний, использующий символы $|\cdot\rangle$ для отображения состояний, в которых система может быть, и их амплитуды.

Подробное объяснение работы кода


Входной параметр:

- `state` — это список (или массив) амплитуд вероятности для всех возможных состояний квантового регистра. Если регистр содержит n кубитов, то длина вектора будет 2^n , и каждый элемент этого вектора представляет собой амплитуду для соответствующего состояния.

Пример:

- Пусть у нас есть двухкубитный регистр, где состояние записано как:

```
python
```

 Копировать код

```
state = [0.6 + 0.2j, 0.8 + 0.4j, 0, 0]
```

Здесь амплитуды для состояний $|00\rangle$, $|01\rangle$, $|10\rangle$, и $|11\rangle$ соответственно.

Шаги выполнения функции:

1. Создание списка для хранения членов:

```
python
```

[Копировать код](#)

```
terms = []
```

Создается пустой список `terms`, в который будут добавляться строки с представлением каждого состояния в бра-кет нотации.

2. Цикл по состояниям:

```
python
```

[Копировать код](#)


```
for i, amplitude in enumerate(state):
```

Этот цикл перебирает все элементы векторного состояния `state`. Для каждого элемента:

- `i` — это индекс текущего состояния (например, для $|00\rangle$ индекс $i = 0$, для $|01\rangle$ — индекс $i = 1$).
- `amplitude` — это амплитуда для текущего состояния (например, для $|00\rangle$ это может быть $0.6 + 0.2j$).

3. Проверка, является ли амплитуда ненулевой:

python

 Копировать код

```
if amplitude != 0:
```

Амплитуда для состояния добавляется в список `terms`, только если она не равна нулю. Это предотвращает добавление состояний с нулевой амплитудой в конечную строку.

4. Формирование строки в бра-кет нотации:

python

 Копировать код

```
terms.append(f"{amplitude:.4f}|{i:02b}")
```

Для каждого состояния создается строка в бра-кет нотации:


- `{amplitude:.4f}` — амплитуда состояния округляется до четырех знаков после запятой для вывода в удобочитаемом формате.
- `|{i:02b}>` — индекс i состояния преобразуется в двоичную строку с двумя знаками (например, для $i = 0$ это будет 00, для $i = 1$ — 01).

Пример:

- Для амплитуды $0.6 + 0.2j$ и индекса 0 создается строка `"0.6000|00">`.
- Для амплитуды $0.8 + 0.4j$ и индекса 1 создается строка `"0.8000|01">`.

5. Возврат результата:

python


 Копировать код

```
return " + ".join(terms)
```

После завершения цикла, список `terms` будет содержать все члены в бра-кет нотации, которые затем объединяются в одну строку через `" + "`. Это позволяет представить квантовое состояние как сумму состояний, каждое из которых сопровождается соответствующей амплитудой.

Пример: Если у нас есть состояние $|00\rangle$ с амплитудой $0.6 + 0.2j$ и $|01\rangle$ с амплитудой $0.8 + 0.4j$, то результатом будет строка:


arduino

 Копировать код

```
"0.6000|00> + 0.8000|01>"
```

Пример работы функции:


python

 Копировать код

```
state = [0.6 + 0.2j, 0.8 + 0.4j, 0, 0]
result = state_to_bracket_notation(state)
print(result)
```

Результат:

arduino

 Копировать код

```
"0.6000|00> + 0.8000|01>"
```

Заключение:

Функция `state_to_bracket_notation` преобразует квантовое состояние из векторного представления в бра-кет нотацию. Она:

- Перебирает все возможные состояния квантового регистра.
- Формирует строку для каждого состояния в виде амплитуда $|i\rangle$, где i — это индекс состояния в двоичной форме.
- Исключает состояния с нулевыми амплитудами и объединяет все строки в итоговую запись с помощью оператора " + ".