Шаг 1 Разложение гамильтониана на линейную комбинацию операторов Паули

Для того чтобы разложить вашу матрицу гамильтониана на линейную комбинацию операторов Паули для системы из двух кубитов, мы будем использовать 4x4 матрицу, которая может быть представлена через операторы Паули для двух кубитов.

Гамильтониан H можно представить как линейную комбинацию операторов Паули:

$$H = a_0 I \otimes I + a_1 \sigma_x \otimes I + a_2 I \otimes \sigma_x + a_3 \sigma_y \otimes \sigma_y + a_4 \sigma_z \otimes \sigma_z + \dots$$

где:

- I — единичная матрица 2×2
- σx, σy, σz— операторы Паули для одного кубита.

Шаг 2: Тензорные произведения для двух кубитов

Для двух кубитов операторы Паули для двух кубитов (тензорное произведение) выглядят следующим образом:

$$I \otimes I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\sigma_x \otimes I = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$I \otimes \sigma_x = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$\sigma_y \otimes \sigma_y = \begin{pmatrix} 0 & 0 & 0 & i \\ 0 & 0 & -i & 0 \\ 0 & i & 0 & 0 \\ -i & 0 & 0 & 0 \end{pmatrix}$$

$$\sigma_z \otimes \sigma_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Шаг 3: Разложение матрицы гамильтониана

Наша матрица гамильтониана может быть разложена как линейная комбинация тензорных произведений операторов Паули:

$$H = a_0 I \otimes I + a_1 \sigma_x \otimes I + a_2 I \otimes \sigma_x + a_3 \sigma_y \otimes \sigma_y + a_4 \sigma_z \otimes \sigma_z$$

Шаг 4: Решение системы для коэффициентов

Решив систему уравнений для коэффициентов, мы получаем следующие значения (решения производятся при помощи кода и представляют переменную H_op:

[-1.05237325+0.j, 0.39793742+0.j, 0.1809312 +0.j, -0.39793743+0.j, -0.01128011+0.j])

Асимптотическая сложность для задачи:

1. **Подготовка гамильтониана**: **Сложность**: O(N^3)
2. **Алгоритм VQE**: Таким образом, общая сложность будет примерно O(p·k·d), где:
   p — количество параметров схемы,
   k — количество итераций оптимизатора,
   d — количество кубитов (для квантовой операции).
3. Сравнение энергии **Сложность**: O(1)

Сам код реализации данной части математики:

```python
from qiskit_aer import AerSimulator
import numpy as np
from qiskit.quantum_info import SparsePauliOp
from qiskit.circuit.library import EfficientSU2
from qiskit_algorithms import VQE
from qiskit_algorithms.optimizers import COBYLA
from qiskit_aer.primitives import Estimator

# Инициализация симулятора
backend = AerSimulator()

# Гамильтониан элемента C в виде матрицы
Hamiltonian = [
    [-1.06365335, 0, 0, 0.1809312],
    [0, -1.83696799, 0.1809312, 0],
    [0, 0.1809312, -0.24521829, 0],
    [0.1809312, 0, 0, -1.06365335]
]

# Преобразуем гамильтониан в операторы Паули с использованием SparsePauliOp
# Этот шаг требует верного представления гамильтониана через операторы Паули
H_op = SparsePauliOp.from_operator(Hamiltonian)
```

Рис. 1 – Подключение необходимых библиотек и библиотека производит тензорные произведения для двух кубитов, раскладывает матрицы гамильтониана и решение системы для коэффициентов

```python
# Определение вариационного анзаца
ansatz = EfficientSU2(num_qubits=2,
                      entanglement='linear',
                      reps=5,
                      skip_final_rotation_layer=True)

# Генерация оптимизированного анзацass
ansatz_opt = ansatz

# Определение оптимизатора (COBYLA)
optimizer = COBYLA(maxiter=1500)

# Инициализация Estimator для вычислений
estimator = Estimator(
    run_options={"shots": 2048, "seed": 28},
)

# Инициализация точек для параметров анзаца
initial_point_values = 2 * np.pi * np.random.rand(ansatz_opt.num_parameters)

# Определение и запуск алгоритма VQE
vqe = VQE(
    estimator=estimator,
    ansatz=ansatz_opt,
    optimizer=optimizer,
    initial_point=initial_point_values
)
```

Рис. 2 - создаём, по документации и другим источникам, среду для просчёта VQE и задаём параметры системы

```python

# Вычисление минимальной энергии с использованием гамильтониана
result = vqe.compute_minimum_eigenvalue(H_op)

# Нормализация минимальной энергии
normalized_energy = result.eigenvalue.real  # Просто без смещения для проверки

print("Минимальная энергия: ", normalized_energy)

# Сравнение с таблицей значений для элементов
element_energies = {
    "H2": -1.9,
    "Be": -14.7,
    "He": -2.9,
    "Li": -7.3
}

# Поиск элемента, который наиболее близок к нормализованной энергии
closest_element = min(element_energies, key=lambda x: abs(element_energies[x] - normalized_energy))
print("Ближайший элемент: ", closest_element)

```

Рис. 3 – запускаем просчёт минимальной энергии системы и сравниваем с таблицей элементов. Во всех запусках ответом является H2 с точностью около 2%.