Министерство науки и высшего образования РФ
ФГБОУ ВО «Тверской государственный университет»
Математический факультет
Направление 02.04.01 Математика и компьютерные науки
Профиль «Математическое и компьютерное моделирование»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Вариационный квантовый алгоритм с оптимизацией методом отжига

Автор:
Алешин Д.А.
Подпись:

Научный руководитель:
д. ф.-м. н. Цирулёв А.Н.
Подпись:

Допущен к защите:
Руководитель ООП: Цветков В.П.

_____
(подпись, дата)

Тверь 2025

Оглавление

Введение

Глава 1

Общая схема квантовых вариационных алгоритмов

1.1 Базис Паули

1.1.1 Связь стандартного базиса и базиса Паули

Рассмотрим квантовую систему из n кубитов, где каждый кубит связан с двумерным гильбертовым пространством $\mathcal{H}$ и его эрмитово сопряжённым пространством $\mathcal{H}^\dagger$. Обозначим через $\mathcal{H}_n = \mathcal{H}^{\otimes n}$ и $\mathcal{H}_n^\dagger = (\mathcal{H}^\dagger)^{\otimes n}$ гильбертово пространство системы и его эрмитово сопряжение соответственно. Пространство линейных операторов, действующих на $\mathcal{H}$ и $\mathcal{H}^\dagger$ левым и правым умножением, задаётся как $L(\mathcal{H}_n) = \mathcal{H}_n \otimes \mathcal{H}_n^\dagger$. Тогда

$$\dim_{\mathbb{C}} \mathcal{H}_n = \dim_{\mathbb{C}} \mathcal{H}_n^\dagger = 2^n, \quad \dim_{\mathbb{C}} L(\mathcal{H}_n) = 2^{2n}.$$

Пространство $L(\mathcal{H}_n)$ наделено скалярным произведением Гильберта-Шмидта:

$$\langle \hat{A}, \hat{B} \rangle = \mathrm{tr}(\hat{A}^\dagger \hat{B}), \quad \hat{A}, \hat{B} \in L(\mathcal{H}_n), \tag{1.1}$$

которое естественно продолжает скалярное произведение в $\mathcal{H}_n$. Вещественное линейное пространство эрмитовых операторов далее обозначим как $H(\mathcal{H}_n)$.

Пусть $\{|0\rangle, |1\rangle\}$ образуют ортонормированный базис в однокубитном пространстве $\mathcal{H}$. Единичная матрица и матрицы Паули задаются как:

$$\sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ \sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},\ \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

а соответствующие операторы Паули представляются в виде:

$$\hat{\sigma}_0 = |0\rangle\langle 0| + |1\rangle\langle 1|,\ \hat{\sigma}_1 = |0\rangle\langle 1| + |1\rangle\langle 0|,$$
$$\hat{\sigma}_2 = -i|0\rangle\langle 1| + i|1\rangle\langle 0|,\ \hat{\sigma}_3 = |0\rangle\langle 0| - |1\rangle\langle 1|.$$

Эти операторы одновременно эрмитовы и унитарны, а также образуют базис в $\mathcal{H}$. Обратное преобразование выражается следующим образом:

$$|0\rangle\langle 0| = \frac{\hat{\sigma}_0 + \hat{\sigma}_3}{2},\ |0\rangle\langle 1| = \frac{\hat{\sigma}_1 + i\hat{\sigma}_2}{2},\ |1\rangle\langle 0| = \frac{\hat{\sigma}_1 - i\hat{\sigma}_2}{2},\ |1\rangle\langle 1| = \frac{\hat{\sigma}_0 - \hat{\sigma}_3}{2}.$$

Для $k, l, m \in \{1, 2, 3\}$ выполняются свойства: $\mathrm{tr}\hat{\sigma}_k = 0$, $\hat{\sigma}_k^2 = \hat{\sigma}_0$, а также

$$\hat{\sigma}_k\hat{\sigma}_l = -\hat{\sigma}_l\hat{\sigma}_k, \quad \hat{\sigma}_k\hat{\sigma}_l = i\,\mathrm{sign}(\pi)\hat{\sigma}_m, \quad (klm) = \pi(123), \tag{1.2}$$

где $\pi(123)$ — произвольная перестановка множества $\{1, 2, 3\}$.

Рассмотрим стандартный[1] бинарный базис в $\mathcal{H}_n$, образованный ортонормированными базисами $\{|0\rangle, |1\rangle\}$ в однокубитных пространствах. Позиция в тензорном произведении позволяет различать кубиты. Для фиксированного n элементы этого базиса и соответствующие им элементы двумерного базиса удобно записывать как:

$$|k\rangle = |k_1 \ldots k_n\rangle = |k_1\rangle \otimes \ldots \otimes |k_n\rangle, \quad \langle k| = \langle k_1 \ldots k_n| = \langle k_1| \otimes \ldots \otimes \langle k_n|,$$

где строки $k_1 \ldots k_n$ $(k_1, \ldots, k_n \in \{0, 1\})$ интерпретируются как двоичные числа с десятичным представлением k. Например, $|101\rangle = |5\rangle$ и $|00110\rangle = |6\rangle$.

[1] Мы избегаем термина «вычислительный», так как он может приводить к неоднозначности. И базис Паули, и стандартный базис являются вычислительными в одинаковом контексте.

В стандартном базисе:

$$|u\rangle = \sum_{k=0}^{2^n-1} u_k |k\rangle, \quad \hat{A} = \sum_{k,l=0}^{2^n-1} a_{kl} |k\rangle\langle l|,$$

где $|u\rangle \in \mathcal{H}_n$ и $\hat{A} \in L(\mathcal{H}_n)$.

Базис Паули $P(\mathcal{H}_n)$ в $L(\mathcal{H}_n)$ определяется как:

$$\{\hat{\sigma}_{k_1\ldots k_n}\}_{k_1,\ldots,k_n\in\{0,1,2,3\}}, \quad \hat{\sigma}_{k_1\ldots k_n} = \hat{\sigma}_{k_1} \otimes \ldots \otimes \hat{\sigma}_{k_n}, \tag{1.3}$$

где $\hat{\sigma}_{0\ldots0}$ — тождественный оператор. Базис $P(\mathcal{H}_n)$ содержит 4^n элементов. Для краткости будем использовать обозначение:

$$\hat{\sigma}_K = \hat{\sigma}_{k_1\ldots k_n},$$

где строка Паули $k_1 \ldots k_n$ ($k_1, \ldots, k_n \in \{0, 1, 2, 3\}$) соответствует числу K в десятичной системе ($0 \leq K \leq 4^n - 1$). Строка Паули K и элемент $\hat{\sigma}_K$ взаимно однозначно соответствуют друг другу.

Сравним $P(\mathcal{H}_n)$ со стандартным базисом. Для элементов базиса Паули выполняются:

$$\hat{\sigma}_{k_1\ldots k_n}\hat{\sigma}_{k_1\ldots k_n} = \hat{\sigma}_{0\ldots0}, \quad \operatorname{tr}\hat{\sigma}_{0\ldots0} = 2^n, \quad \operatorname{tr}\hat{\sigma}_{k_1\ldots k_n}\Big|_{k_1\ldots k_n\neq 0\ldots0} = 0. \tag{1.4}$$

Базис Паули является эрмитовым, унитарным и ортогональным относительно скалярного произведения (1.1). Отметим, что оператор $|k\rangle\langle l|$ из стандартного базиса не является унитарным или эрмитовым при $k \neq l$. Стандартный базис не включает тождественный оператор, который в этом базисе записывается как:

$$\sum_{k=0}^{2^n-1} |k\rangle\langle k|.$$

В базисе Паули любой оператор $\hat{U}$ из унитарной группы $U(\mathcal{H}_n)$ (где $\hat{U}^\dagger\hat{U} = \hat{\sigma}_{0\ldots0}$) раскладывается в виде:

$$\hat{U} = \sum_{i_1,\ldots,i_n\in\{0,1,2,3\}} U_{i_1\ldots i_n}\hat{\sigma}_{i_1\ldots i_n}, \quad \hat{U}^\dagger = \sum_{i_1,\ldots,i_n\in\{0,1,2,3\}} \overline{U}_{i_1\ldots i_n}\hat{\sigma}_{i_1\ldots i_n},$$

где коэффициенты удовлетворяют условиям:

$$\sum_{i_1,\ldots,i_n\in\{0,1,2,3\}} \overline{U}_{i_1\ldots i_n} U_{i_1\ldots i_n} = 1, \qquad \sum_{\substack{i_1,\ldots,i_n,j_1,\ldots,j_n\in\{0,1,2,3\} \\ (i_1,\ldots,i_n)\neq(j_1,\ldots,j_n)}} \overline{U}_{i_1\ldots i_n} U_{j_1\ldots j_n} = 0.$$

Последнее условие эквивалентно $2^{2n-1}(2^n-1)$ независимым соотношениям.

Эрмитовы операторы в базисе Паули разлагаются с вещественными коэффициентами.

1.1.2 Коммутационное и антикоммутационное соотношение

Коммутатор определяет взаимодействие операторов при их перестановке. Для операторов A и B он задаётся как:

$$[A, B] = AB - BA.$$

Если $[A, B] = 0$, операторы коммутируют; в противном случае — нет. В базисе Паули коммутаторы выражаются через символ Леви-Чивиты ε_{ijk}:

$$[\sigma_i, \sigma_j] = 2i\varepsilon_{ijk}\sigma_k,$$

где ε_{ijk} равен 1 при чётной перестановке индексов (i, j, k), -1 при нечётной и 0 в остальных случаях. Например:

$$[\sigma_1, \sigma_2] = 2i\sigma_3, \quad [\sigma_2, \sigma_3] = 2i\sigma_1.$$

Антикоммутатор характеризует симметричное произведение операторов:

$$\{A, B\} = AB + BA.$$

Если $\{A, B\} = 0$, операторы антикоммутируют. Для операторов Паули:

$$\{\sigma_i, \sigma_j\} = 2\delta_{ij},$$

где δ_{ij} — символ Кронекера (1 при $i = j$, 0 иначе). Примеры:

$$\{\sigma_1, \sigma_2\} = 0, \quad \{\sigma_2, \sigma_3\} = 0.$$

Коммутаторы и антикоммутаторы применяются в квантовой механике для анализа свойств систем (спин электрона, кубиты), в квантовой теории поля (взаимодействия частиц) и квантовых вычислениях (алгоритмы, коррекция ошибок). Коммутаторы помогают определить совместную измеримость наблюдаемых, а антикоммутаторы — описать фермионные системы.

1.1.3 Коммутирующие элементы в алгебре Ли группы $SU(2^n)$

Рассмотрим базис Паули и множество строк Паули длины n:

$$\mathrm{Str}_n = \{K = k_1 \dots k_n\}_{k_1,\dots,k_n \in \{0,1,2,3\}}.$$

1. Множество $\mathbb{F}_4 = \{0, 1, 2, 3\}$ образует квадгруппу Клейна с умножением:

$$0 \cdot k = k, \quad k \cdot k = 0, \quad k \cdot l = m,$$

где $k, l, m \in \{1, 2, 3\}$ и (klm) — произвольная перестановка $(1\,2\,3)$.

2. Функция $s : \mathbb{F}_4 \times \mathbb{F}_4 \to \{1, i, -i\}$ задаётся значениями:

$$\begin{aligned} &s(0,0) = s(0,k) = s(k,0) = s(k,k) = 1, \quad k = 1,2,3, \\ &s(1,2) = s(2,3) = s(3,1) = i, \qquad\qquad s(2,1) = s(3,2) = s(1,3) = -i. \end{aligned}$$

3. Функция $S : \mathrm{Str}_n \times \mathrm{Str}_n \to \{1, -1, i, -i\}$ определяется как:

$$S_{KL} = s(k_1, l_1) \cdot s(k_2, l_2) \cdot \ldots \cdot s(k_n, l_n),$$

где $K = k_1 k_2 \dots k_n$ и $L = l_1 l_2 \dots l_n$.

Симметрия функции S зависит от числа пар (k_r, l_r) (на позициях r в строках K и L), где $k_r, l_r \in \{1, 2, 3\}$ и $k_r \neq l_r$, а также от их взаимного порядка. Пусть ω^+_{KL} и ω^-_{KL} — количество пар вида $(1,2), (2,3), (3,1)$ и $(2,1), (3,2), (1,3)$ соответственно, и $\omega_{KL} = \omega^+_{KL} + \omega^-_{KL}$. Тогда

$$S_{(KL)} = \frac{S_{KL}}{2}\left(1 + (-1)^{\omega_{KL}}\right), \quad S_{[KL]} = \frac{S_{KL}}{2}\left(1 - (-1)^{\omega_{KL}}\right), \tag{1.5}$$

где

$$S_{KL} = i^{\omega_{KL}}(-1)^{\omega^-_{KL}}.$$

Здесь $S_{(KL)}$ и $S_{[KL]}$ — симметричная и антисимметричная части S_{KL}. Значения S_{KL}, $S_{(KL)}$ и $S_{[KL]}$ приведены в таблице 2.

$\omega_{KL}\mathrm{mod}4$	0	2	0	2	1	3	1	3
$\omega^{-}_{KL}\mathrm{mod}4$	0	1	1	0	0	1	1	0
S_{KL}		1	-1	-1	i	i	$-i$	$-i$
$S_{(KL)}$	1	1	-1	-1	0	0	0	0
$S_{[KL]}$	0	0	0	0	i	i	$-i$	$-i$

Таблица 1: Множитель до $\hat{\sigma}_M$ в (1.6) для $\hat{\sigma}_K\hat{\sigma}_L, \{\hat{\sigma}_K, \hat{\sigma}_L\}$, и $[i\hat{\sigma}_K, i\hat{\sigma}_L]$.

Композицию элементов базиса Паули, их антикоммутаторов и коммутаторов можно компактно выразить в виде, удобном для программной реализации:

$$\hat{\sigma}_K\hat{\sigma}_L = S_{KL}\hat{\sigma}_M, \quad \{\hat{\sigma}_K, \hat{\sigma}_L\} = S_{(KL)}\hat{\sigma}_M, \quad [i\hat{\sigma}_K, i\hat{\sigma}_L] = -S_{[KL]}\hat{\sigma}_M, \quad (1.6)$$

где

$$\hat{\sigma}_M = \hat{\sigma}_{m_1\ldots m_n}, \quad m_r = k_r \cdot l_r \quad (r = 1, \ldots, n). \quad (1.7)$$

Две строки Паули длины n могут коммутировать, даже имея ненулевые элементы в одних и тех же позициях. Например, операторы $\hat{\sigma}_{11}$, $\hat{\sigma}_{22}$ и $\hat{\sigma}_{33}$ коммутируют друг с другом. Унитарная матрица перехода из стандартного базиса $\{|i_1 \ldots i_n\rangle\langle j_1 \ldots j_n|\}$ в базис Паули содержит только элементы 0, ± 1 и $\pm i$. Например:

$$|00\ldots0\rangle\langle 00\ldots0| \to \frac{1}{2^n} \sum_{i_1,\ldots,i_n \in \{0,3\}} \hat{\sigma}_{i_1\ldots i_n}.$$

Общее выражение для стандартных ортогональных проекторов имеет вид:

$$|i_1 \ldots i_n\rangle\langle i_1 \ldots i_n| = \frac{1}{2^n} \sum_{k_1,\ldots,k_n \in \{0,3\}} \mathcal{X}^{i_1}_{k_1} \cdots \mathcal{X}^{i_n}_{k_n} \hat{\sigma}_{k_1\ldots k_n},$$

где

$$\mathcal{X}^0_0 = \mathcal{X}^0_3 = \mathcal{X}^1_0 = 1, \quad \mathcal{X}^1_3 = -1.$$

Из выражения (1.6) следует, что: 1. Множество $\{i\hat{\sigma}_K\}_{K=0}^{4^n-1}$ образует ортонормированный базис в $\mathfrak{su}(2^n)$. 2. Множество

$$\widetilde{P}(\mathcal{H}_n) = \{\epsilon\hat{\sigma}_K \mid K \in \mathrm{Str}_n,\ \epsilon \in \{\pm 1, \pm i\}\},$$

содержащее 4^{n+1} элементов, образует группу — т.н. n-кубитную группу Паули.

Нормализатор группы Паули в унитарной группе:

$$\mathcal{C}(\mathcal{H}_n) = \left\{\hat{U} \in U(\mathcal{H}_n) \mid \hat{U}\hat{\sigma}_K\hat{U}^\dagger \in \widetilde{P}(\mathcal{H}_n),\ \ \forall\hat{\sigma}_K \in \widetilde{P}(\mathcal{H}_n)\right\},$$

называется группой Клиффорда. Исходя из (1.2), (1.4) и (1.7) получаем следующее утверждение

Утверждение 1. *Взаимные унитарные преобразования базисных операторов Паули подчиняются соотношениям* $\hat{\sigma}_{i_1\ldots i_n}\hat{\sigma}_{k_1\ldots k_n}\hat{\sigma}_{i_1\ldots i_n} = \pm\hat{\sigma}_{i_1\ldots i_n}$, *где знак плюс стоит тогда и только тогда, когда количество троек* $(i_m k_m i_m)_{m\in\{1,\ldots,n\}}$, *удовлетворяют условиям* $i_m \neq k_m$, $i_m \neq 0$, *и* $k_m \neq 0$ *четности.*

l	0	1	2	3	4	5	6	7
$l_2l_1l_0$	000	001	010	011	100	101	110	111
$k_2k_1k_0$	011	011	011	011	011	011	011	011
$\bar{l} \wedge k$	011	010	001	000	011	010	001	000
$l \wedge k$	000	001	010	011	000	001	010	011
$l \wedge \overline{k}$	000	000	000	000	100	100	100	100
$\hat{\sigma}_I$	$\hat{\sigma}_{011}$	$\hat{\sigma}_{012}$	$\hat{\sigma}_{021}$	$\hat{\sigma}_{022}$	$\hat{\sigma}_{311}$	$\hat{\sigma}_{312}$	$\hat{\sigma}_{121}$	$\hat{\sigma}_{322}$

Таблица 2: Элементы базиса Паули, возникающие для $k = 011$.

1.2 Вариационная квантовая оптимизация

Пусть $\mathcal{H}_n$ — гильбертово пространство квантовой системы, состоящей из n кубитов, $\mathcal{S} \subset \mathcal{H}_n$ — пространство векторов состояния (т.е. векторов, нормированных на единицу), $L(\mathcal{H}_n)$ — алгебра операторов на $\mathcal{H}_n$ и $\hat{H} \in L(\mathcal{H}_n)$ — эрмитов оператор. Определим функцию

$$E(|u\rangle) = \langle u|\hat{H}|u\rangle, \quad |u\rangle \in \mathcal{S}. \tag{1.8}$$

В простейшей постановке ***квантовой задачи оптимизации*** *требуется найти вектор состояния, на котором целевая функция (функция стоимости)* E *принимает минимальное значение*, т.е., в формальной записи, решить задачу

$$E(|u\rangle) \xrightarrow{|u\rangle\in\mathcal{S}} \min\,. \tag{1.9}$$

Ниже, для определенности и краткости, будем называть $\hat{H}$ гамильтонианом системы, а целевую функцию E — энергией.

Сложность алгоритмов прямого вычисления собственных значений гамильтониана $\hat{H}$ растет экспоненциально с ростом числа кубитов, поэтому для больших систем используются вариационные методы решения задачи оптимизации (1.9).

Вариационными квантовыми алгоритмами обычно называют такие гибридные квантово-классические алгоритмы, нацеленные на решение квантовых задач оптимизации посредством квантовых вычислений или их классической имитации, которые проводят вариационную настройку параметров квантовой схемы. Параметрически управляемое квантовое устройство, обычно представленное квантовой цепью, реализует анзац, т.е. унитарное преобразование стандартного начального состояния $|0\rangle^{\otimes n}$ или, как вариант, предудущего полученного состояния. На каждом шаге регулирующие параметры подбираются так, чтобы минимизировать энергию (целевую функцию). Обычно это выполняется путём измерения энергии состояний, предоставляемых вариационной схемой, и обновления параметров для минимизации целевой функции.

В точной математической формулировке сказанное означает, что в функции (1.8) вектор состояния $|u\rangle$ зависит от набора m параметров $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$, которые принимают значения в некоторой связной и односвязной области $\Omega \in \mathbb{R}^m$. Вариационная формулировка квантовой задачи оптимизации (1.9) имеет вид

$$E\big(\,|u(\boldsymbol{\theta})\rangle\,\big) \xrightarrow{\boldsymbol{\theta}\in\Omega} \min, \qquad E\big(\,|u(\boldsymbol{\theta})\rangle\,\big) = \langle u(\boldsymbol{\theta})|\,\hat{H}\,|u(\boldsymbol{\theta})\rangle\,. \tag{1.10}$$

Итак, цель вариационного квантового алгоритма — найти такой набор параметров, на котором энергия достигает минимума. Число параметров m в наборе $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$ зависит от конкретной задачи, в частности,

от числа кубитов в квантовом устройстве. Для n кубитов размерность пространства состояний, $N = 2^n$, растет экспоненциально с ростом числа кубитов. Поэтому вариационный квантовый алгоритм должен быть организован и выполнен так, чтобы выполнялось условие $m \ll N$, поскольку в противном случае высокий класс сложности алгоритма сделает его неэффективным с практической точки зрения.

Но наиболее важным вопросом является выбор зависимости вектора состояния $|u(\boldsymbol{\theta})\rangle$ от параметров. В вариационных квантовых алгоритмах используется *анзац* (унитарное преобразование) вида

$$|u(\boldsymbol{\theta})\rangle = \hat{U}(\boldsymbol{\theta})\,|u_0\rangle\,. \tag{1.11}$$

В общем случае форма анзаца определяет, какими будут параметры $\boldsymbol{\theta}$ и как их можно настроить для минимизации энергии (целевой функции). Структура анзаца, как правило, будет зависеть от поставленной задачи, так как во многих случаях можно использовать информацию о проблеме, чтобы подобрать анзац: это "анзац, подсказнный задачей". Однако можно построить анзацы достаточно общего вида, которые пригодны для использования в некоторых классах задач даже тогда, когда интуиция и известная информация о задаче не позволят его уточнить. Стандартно анзац выбирается в виде композиции m последовательно примененных унитарных преобразований

$$\hat{U}(\boldsymbol{\theta}) = \hat{U}_m(\theta_m)\cdots\hat{U}_1(\theta_1). \tag{1.12}$$

В композиции (1.12) выбор операторов определяется типом задачи и технической возможностью их реализации на конкретном квантовом устройстве. Например, можно выбрать

$$\hat{U}_K(\theta_K) = \hat{W}_K \exp\bigl(i\theta_K\hat{\sigma}_K\bigr) = \hat{W}_K\bigl(\cos\theta_K\hat{\sigma}_{0\ldots0} + i\sin\theta_K\hat{\sigma}_K\bigr), \tag{1.13}$$

где $1 \leqslant K \leqslant m$, $\hat{\sigma}_K = \hat{\sigma}_{k_1}\otimes\ldots\otimes\hat{\sigma}_{k_n}$, $k_1,\ldots,k_n \in \{0,1,2,3\}$, $K = k_1\ldots k_n$ (т.е. K — десятичное представление строки $k_1\ldots k_n$, рассматриваемой как число по основанию 4), n — число кубитов, а $\hat{W}_K$ — независящий от параметров унитарный оператор. Как правило, в строке $k_1\ldots k_n$ только отдельные числа отличны от нуля, так что в тензорном произведении $\hat{\sigma}_{k_1}\otimes\ldots\otimes\hat{\sigma}_{k_n}$ часть операторов являются тождественными.

В другом распространенном варианте операторы в композиции (1.12) имеют вид

$$\hat{U}_K(\theta_K) = \hat{W}_K\big(\mathrm{e}^{i\theta_{k_1}\hat{\sigma}_{k_1}} \otimes \ldots \otimes \mathrm{e}^{i\theta_{k_n}\hat{\sigma}_{k_n}}\big), \tag{1.14}$$

где по-прежнему $1 \leqslant K \leqslant m$ и $K = k_1 \ldots k_n$. Если в (1.13) все операторы $\hat{W}_K$ могут быть тождественными, то в (1.14), по крайней мере некоторые операторы $\hat{W}_K$ должны быть запутывающими и, следовательно, как минимум двухкубитными.

Таким образом, анзацы (1.12), (1.13) и (1.14) конкретизируют вариационную квантовую задачу оптимизации (1.10) и (1.11) в отношении параметрической зависимости вектора состояния,

$$|u(\boldsymbol{\theta})\rangle = \hat{U}(\boldsymbol{\theta})\,|0\ldots0\rangle\,,$$

где начальное состояние имеет вид $|0\ldots0\rangle = |0\rangle^{\otimes n}$. Если предположить далее, что мы в состоянии уверенно приготовить начальное состояние, реализовать анзац на физическом устройстве и вычислить значение энергии $E\big(\,|u(\boldsymbol{\theta})\rangle\,\big) = \langle u(\boldsymbol{\theta})|\hat{H}\,|u(\boldsymbol{\theta})\rangle$ посредством измерений (с привлечением классического компьютера), то следующий — основной — вопрос можно сформулировать так: как искать параметры, которые обеспечивают глобальный минимум энергии. Этот этап выполняется с помощью классического компьютера, так что вариационный квантовый алгоритм — гибридный квантово-классический алгоритм: параметризованная квантовая схема и измерительный прибор представляют квантовую часть, а алгоритм настройки параметров — классическую.

1.3 Общая схема алгоритма и анзац

1.4 Пример, иллюстрирующий особенности алгоритма

Для иллюстрации алгоритма рассмотрим гамильтониан

$$\hat{H} = 2\,\hat{\sigma}_{03} + \hat{\sigma}_{30} - 4\,\hat{\sigma}_{11}, \tag{1.15}$$

который в стандартном базисе $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ имеет матрицу

$$H = \begin{pmatrix} 3 & 0 & 0 & -4 \\ 0 & -1 & -4 & 0 \\ 0 & -4 & 1 & 0 \\ -4 & 0 & 0 & -3 \end{pmatrix}.$$

Используя систему Maple находим собственные значения и собственные состояния в порядке возрастания собственных значений, начиная с основного состояния $|u_0\rangle$ с собственным значением E_0:

$$E_0 = -5, \qquad |u_0\rangle = \frac{1}{\sqrt{5}}|00\rangle + \frac{2}{\sqrt{5}}|11\rangle, \tag{1.16}$$

$$E_1 = -\sqrt{17}, \quad |u_1\rangle = \frac{\sqrt{17}+1}{\sqrt{34+2\sqrt{17}}}|01\rangle + \frac{\sqrt{8}}{\sqrt{17+\sqrt{17}}}|10\rangle, \tag{1.17}$$

$$E_2 = \sqrt{17}, \quad |u_1\rangle = -\frac{\sqrt{17}-1}{\sqrt{34-2\sqrt{17}}}|01\rangle + \frac{\sqrt{8}}{\sqrt{17-\sqrt{17}}}|10\rangle, \tag{1.18}$$

$$E_3 = 5, \qquad |u_3\rangle = -\frac{2}{\sqrt{5}}|00\rangle + \frac{1}{\sqrt{5}}|11\rangle. \tag{1.19}$$

Рассмотрим далее пошаговое выполнение вариационного квантового алгоритма, который позволяет найти состояние, близкое к основному.

Первый шаг — выбор анзаца, т.е. унитарного преобразования $\hat{U}(\boldsymbol{\theta})$. В гамильтониан (1.15) не входят операторы вида $\hat{\sigma}_{k2}$ и $\hat{\sigma}_{k2}$ с $k \neq 2$, поэтому имеет смысл сразу выбирать анзац так, чтобы при действии на $k \neq 2$ он давал вектор состояния с вещественными коэффициентами. Других наводящих соображений относительно формы анзаца не видно, поэтому следует рассмотреть разные варианты. В общем случае вектор параметров $\boldsymbol{\theta}$ четырехмерен. В простейшем варианте анзац с четырехмерным вектором параметров $\boldsymbol{\theta} = (\xi, \lambda, \mu, \nu)$ можно выбирать как композицию экспонент

$$\hat{U}(\boldsymbol{\theta}) = \mathrm{e}^{i\xi\hat{\sigma}_{02}}\mathrm{e}^{i\lambda\hat{\sigma}_{03}}\mathrm{e}^{i\mu\hat{\sigma}_{30}}\mathrm{e}^{i\nu\hat{\sigma}_{11}} \tag{1.20}$$

операторов Паули, присутствующих в гамильтониане (1.15). Вычислим

вначале

$$\mathrm{e}^{i\mu\hat{\sigma}_{30}}\mathrm{e}^{i\nu\hat{\sigma}_{11}}\,|00\rangle = \big(\cos\mu\,\hat{\sigma}_{00} + i\sin\mu\,\hat{\sigma}_{30}\big)\big(\cos\nu\,|00\rangle + i\sin\nu\,|11\rangle\big)$$

$$= \cos\mu\cos\nu\,|00\rangle - \sin\mu\sin\nu\,|11\rangle + i\sin\mu\cos\nu\,|00\rangle + i\cos\mu\sin\nu\,|11\rangle$$

$$= \mathrm{e}^{i\mu}\cos\nu\,|00\rangle + i\mathrm{e}^{i\mu}\sin\nu\,|11\rangle = \mathrm{e}^{i\mu}\cos\nu\,|00\rangle + \mathrm{e}^{i(\mu+\pi/2)}\sin\nu\,|11\rangle\,.$$

Действуя на результат оператором $\mathrm{e}^{i\lambda\hat{\sigma}_{03}}$, получим следующий промежуточный вектор состояния:

$$\mathrm{e}^{i\lambda\hat{\sigma}_{03}}\mathrm{e}^{i\mu\hat{\sigma}_{30}}\mathrm{e}^{i\nu\hat{\sigma}_{11}}\,|00\rangle = \mathrm{e}^{i\mu}\cos\nu\big(\cos\lambda\,\hat{\sigma}_{00} + i\sin\lambda\hat{\sigma}_{03}\big)\,|00\rangle$$
$$+ \mathrm{e}^{i(\mu+\pi/2)}\sin\nu\big(\cos\lambda\,\hat{\sigma}_{00} + i\sin\lambda\hat{\sigma}_{03}\big)\,|11\rangle$$

$$= \mathrm{e}^{i\mu}\cos\nu\big(\cos\lambda + i\sin\lambda\big)\,|00\rangle + \mathrm{e}^{i(\mu+\pi/2)}\sin\nu\big(\cos\lambda + i\sin\lambda\big)\,|11\rangle$$
$$= \mathrm{e}^{i(\mu+\lambda)}\cos\nu\,|00\rangle + \mathrm{e}^{i(\mu+\pi/2-\lambda)}\sin\nu\,|11\rangle\,. \qquad (1.21)$$

Очевидно, что этот анзац не является универсальным.

Варьируя параметры λ, μ, ν, можно получить основное состояние (1.16) с точностью до несущественного множителя $\mathrm{e}^{i(\mu+\pi/4)}$, например, при

$$\lambda = \pi/4, \quad \mu \in \mathbb{R}, \quad \cos\nu = 1/\sqrt{5}, \quad \sin\nu = 2/\sqrt{5}. \qquad (1.22)$$

Здесь μ — любое, поэтому к нужному результату приводит более простой анзац (при $\mu = 0$) $\hat{U}(\boldsymbol{\theta}) = \mathrm{e}^{i\lambda\hat{\sigma}_{03}}\mathrm{e}^{i\nu\hat{\sigma}_{11}}$, однако заранее это нам не известно. Более того основное состояние (1.16) можно достигнуть (что заранее также неизвестно и неочевидно) даже однопараметрическим анзацем

$$\mathrm{e}^{i\nu\hat{\sigma}_{12}}\,|00\rangle = \big(\cos\nu\,\hat{\sigma}_{00} + i\sin\nu\,\hat{\sigma}_{12}\big)\,|00\rangle = \cos\nu\,|00\rangle + \sin\nu\,|11\rangle\,,$$

с теми же значениями $\cos\nu$ и $\sin\nu$, что и в (1.22).

Действуя на (1.21) оператором $\mathrm{e}^{i\xi\hat{\sigma}_{02}}$, получим вектор состояния

$$|\Phi\rangle = \hat{U}(\boldsymbol{\theta})\,|00\rangle$$

$$= \big(\cos\xi\,\hat{\sigma}_{00} + i\sin\xi\,\hat{\sigma}_{02}\big)\big(\mathrm{e}^{i(\mu+\lambda)}\cos\nu\,|00\rangle + \mathrm{e}^{i(\mu+\pi/2-\lambda)}\sin\nu\,|11\rangle\big)$$

$$= \mathrm{e}^{i(\mu+\lambda)}\sin\xi\cos\nu\,|01\rangle - \mathrm{e}^{i(\mu+\pi/2-\lambda)}\sin\xi\sin\nu\,|10\rangle$$
$$+ \mathrm{e}^{i(\mu+\lambda)}\cos\xi\cos\nu\,|00\rangle + \mathrm{e}^{i(\mu+\pi/2-\lambda)}\cos\xi\sin\nu\,|11\rangle\,, \qquad (1.23)$$

который зависит от четырех параметров. Из формы данного вектора видно, что анзац (1.20) универсален (с учетом замечания о вещественности коэффициентов, сделанного выше). Основное состояние достигается при произвольном $\mu \in \mathbb{R}$ и

$$\xi = 0,\ \lambda = \{\pi/4, 7\pi/4\},\ \cos\nu = 1/\sqrt{5},\ \sin\nu = \{2/\sqrt{5}, -2/\sqrt{5}\} \quad (1.24)$$

или

$$\xi = \pi,\ \lambda = \{\pi/4, 7\pi/4\},\ \cos\nu = -1/\sqrt{5},\ \sin\nu = \{-2/\sqrt{5}, 2/\sqrt{5}\}. \quad (1.25)$$

Для сокращения записи имеет смысл освободиться в (1.23) от фазового множителя и записать вектор состояния в виде

$$|\Phi\rangle = \sin\xi\big(\cos\nu\,|01\rangle - \mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu\,|10\rangle\big) + \cos\xi\big(\cos\nu\,|00\rangle + \mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu\,|11\rangle\big) \quad (1.26)$$

Второй шаг — вычисление энергии состояния, т.е. среднего значения $\langle\Phi|\hat{H}|\Phi\rangle$. Заметим, что первый и второй шаги должны выполняться на квантовых устройствах, а при классической симуляции алгоритма необходимо проводить явные вычисления. Из (1.15) и (1.26) находим

$$\begin{aligned}
\hat{H}|\Phi\rangle &= \sin\xi\big(2\,\hat{\sigma}_{03} + \hat{\sigma}_{30} - 4\hat{\sigma}_{11}\big)\big(\cos\nu\,|01\rangle - \mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu\,|10\rangle\big) \\
&\quad + \cos\xi\big(2\,\hat{\sigma}_{03} + \hat{\sigma}_{30} - 4\hat{\sigma}_{11}\big)\big(\cos\nu\,|00\rangle + \mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu\,|11\rangle\big) \\
&= \sin\xi\Big\{\big(4\mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu - \cos\nu\big)\,|01\rangle \\
&\qquad - \big(\mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu + 4\cos\nu\big)\,|10\rangle\Big\} \\
&\quad + \cos\xi\Big\{\big(3\cos\nu - 4\mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu\big)\,|00\rangle \\
&\qquad - \big(4\cos\nu + 3\mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu\big)\,|11\rangle\Big\}.
\end{aligned}$$

Поскольку

$$\langle\Phi| = \sin\xi\big(\cos\nu\,\langle 01| - \mathrm{e}^{-i(\pi/2-2\lambda)}\sin\nu\,\langle 10|\big) + \cos\xi\big(\cos\nu\,\langle 00| + \mathrm{e}^{-i(\pi/2-2\lambda)}\sin\nu\,\langle 11|\big),$$

то

$$
\begin{aligned}
E_\Phi &= \langle\Phi|\,\hat{H}\,|\Phi\rangle \\
&= \sin^2\xi\Big\{\cos\nu\big(4\mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu-\cos\nu\big) \\
&\qquad\qquad +\sin\nu\big(\sin\nu+4\mathrm{e}^{-i(\pi/2-2\lambda)}\cos\nu\big)\Big\} \\
&\quad+\cos^2\xi\Big\{\cos\nu\big(3\cos\nu-4\mathrm{e}^{i(\pi/2-2\lambda)}\sin\nu\big) \\
&\qquad\qquad -\sin\nu\big(4\mathrm{e}^{-i(\pi/2-2\lambda)}\cos\nu+3\sin\nu\big)\Big\} \\
&= \sin^2\xi\big(4\sin2\lambda\sin2\nu-\cos2\nu\big)+\cos^2\xi\big(3\cos2\nu-4\sin2\lambda\sin2\nu\big). \quad (1.27)
\end{aligned}
$$

Разумеется, если взять значения ξ, λ, $\cos\nu$, $\sin\nu$ как в (1.24) или в (1.25), то мы получим энергию основного состояния (1.16), т.е. $E_\Phi=-5$.

Третий шаг — изменение значений параметров λ,μ,ν (с целью минимизации E_Φ) и возвращение к первому шагу; предполагается, что в начале выполнения алгоритма начальные значения параметров заданы. Из (1.27) видно, что на значение E_Φ параметр μ не влияет, а параметры λ,ν должны варьироваться в области $[0,\pi]\times[0,\pi]$. Однако изначально это неизвестно, поэтому все четыре параметра должны варьироваться в области $[0,2\pi]\times[0,2\pi]\times[0,2\pi]\times[0,2\pi]$. Имеет смысл установить независимость энергии от параметра μ (и ее зависимость от остальных параметров) в начале работы алгоритма.

Мы уже знаем, что глобальный минимум энергии достигается для четырех наборов параметров (1.24) и (1.25). Соответствующие собственные векторы, вычисленные по выражению (1.26) отличаются от (1.16) только фазовыми множителями. Теперь необходимо выяснить, имеются ли у функции (трех переменных) (1.27) другие локальные минимумы.

Используя систему Maple, вычислим производные

$$
\partial_\xi E_\Phi,\ \ \partial_\lambda E_\Phi,\ \ \partial_\nu E_\Phi,
$$

$$
A=\partial^2_\xi E_\Phi,\ \ B=\partial^2_\lambda E_\Phi,\ \ C=\partial^2_\nu E_\Phi,
$$
$$
K=\partial_{\xi\lambda} E_\Phi,\ \ L=\partial_{\xi\nu} E_\Phi,\ \ M=\partial_{\lambda\nu} E_\Phi.
$$

Находим

$$\begin{aligned}
\partial_\xi E_\Phi &= 4\sin 2\xi\big(2\sin 2\lambda\sin 2\nu - \cos 2\nu\big),\\
\partial_\lambda E_\Phi &= -8\cos 2\xi\cos 2\lambda\sin 2\nu,\\
\partial_\nu E_\Phi &= \sin^2\xi\big(8\sin 2\lambda\cos 2\nu + 2\sin 2\nu\big) - \cos^2\xi\big(6\sin 2\nu + 8\sin 2\lambda\cos 2\nu\big),\\
A &= 8\cos 2\xi\big(2\sin 2\lambda\sin 2\nu - \cos 2\nu\big),\\
B &= 16\cos 2\xi\sin 2\lambda\sin 2\nu,\\
C &= 4\cos^2\xi\big(4\sin 2\lambda\sin 2\nu - 3\cos 2\nu\big) - 4\sin^2\xi\big(4\sin 2\lambda\sin 2\nu - \cos 2\nu\big),\\
K &= 16\sin 2\xi\cos 2\lambda\sin 2\nu,\\
L &= 4\sin 2\xi\big(4\sin 2\lambda\cos 2\nu + 2\sin 2\nu\big),\\
M &= -16\cos 2\xi\cos 2\lambda\cos 2\nu.
\end{aligned}$$

Необходимые и достаточные условия минимума имеют вид

$$\partial_\xi E_\Phi = 0, \quad \partial_\lambda E_\Phi = 0, \quad \partial_\nu E_\Phi = 0,$$

$$A > 0, \qquad \det\begin{pmatrix} A & K \\ K & B \end{pmatrix} > 0, \qquad \det\begin{pmatrix} A & K & L \\ K & B & M \\ L & M & C \end{pmatrix} > 0.$$

Снова проводя вычисления с помощью системы Maple, обнаруживаем четыре точки локального минимума с энергией $E_\Phi = -\sqrt{17}$:

$$\begin{aligned}
&\xi = \frac{\pi}{2}, \qquad \lambda = \frac{\pi}{4}, \qquad \nu = \pi - \frac{1}{2}\arctan(4),\\
&\xi = \frac{\pi}{2}, \qquad \lambda = \frac{\pi}{4}, \qquad \nu = 2\pi - \frac{1}{2}\arctan(4),\\
&\xi = \frac{\pi}{2}, \qquad \lambda = \frac{3\pi}{4}, \qquad \nu = \frac{1}{2}\arctan(4),\\
&\xi = \frac{\pi}{2}, \qquad \lambda = \frac{3\pi}{4}, \qquad \nu = \pi + \frac{1}{2}\arctan(4).
\end{aligned}$$

Таким образом, в процессе оптимизации целевой функции должны использоваться методы, которые позволяют избежать попадания в точку локального минимума, например, метод отжига.

1.5 Оптимизация

Глава 2

Вариационный квантовый алгоритм на основе метода отжига

2.1 Метод отжига

Метод отжига, как фундаментальная концепция в решении задач глобальной оптимизации, находит широкое применение в квантовых вычислениях, особенно в контексте вариационных квантовых алгоритмов. Основная идея метода заключается в постепенном снижении "температуры"системы, чтобы достичь состояния минимальной энергии. В этом разделе подробно рассмотрим как классический, так и квантовый подходы к отжигу, их теоретические основы и практическое применение.

Классический метод отжига основывается на аналогии с физическим процессом термического отжига, при котором материал медленно охлаждается, чтобы избежать образования дефектов и достичь состояния минимальной энергии. Математическое основание метода связано с распределением Больцмана, которое описывает вероятность состояния системы при заданной температуре T:

$$p(x) = \frac{1}{Z(T)} \exp\left(-\frac{E(x)}{k_B T}\right), \tag{2.1}$$

где $E(x)$ — энергия состояния x, k_B — константа Больцмана, а $Z(T)$ — статистическая сумма. Процесс отжига моделирует систему, которая может переходить между состояниями x и y с вероятностью, зависящей от разности энергий $\Delta E = E(y) - E(x)$:

$$p(x \to y) = \min\left(1, \exp\left(-\frac{\Delta E}{k_B T}\right)\right). \tag{2.2}$$

С течением времени, температура T постепенно уменьшается, что приводит к уменьшению вероятности перехода в состояния с более высокой энергией, в то время как система стремится к состоянию глобального минимума энергии.

Квантовый алгоритм отжига использует преимущества квантовой механики, такие как суперпозиция и туннелирование, для более эффективного поиска глобального минимума. В отличие от классического подхода, квантовый отжиг позволяет системе преодолевать энергетические барьеры, используя когерентное туннелирование, что значительно увеличивает вероятность нахождения глобального минимума.

Квантовый отжиг моделируется с помощью временного гамильтониана, который постепенно изменяется от начального состояния к целевому:

$$H(t) = (1 - s(t))H_B + s(t)H_P, \tag{2.3}$$

где H_B — начальный гамильтониан, часто представляющий собой простую задачу, такую как сумма операторов Паули X, а H_P — проблема-специфический гамильтониан. Функция $s(t)$, изменяющаяся от 0 до 1, управляет эволюцией системы от начального состояния к состоянию минимальной энергии.

Эволюция квантовой системы описывается уравнением Шрёдингера:

$$i\hbar\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = H(t)\left|\psi(t)\right\rangle, \tag{2.4}$$

где $\hbar$ — приведённая постоянная Планка. Это уравнение описывает, как квантовая система изменяется со временем под воздействием изменяющегося гамильтониана.

Важным аспектом квантового отжига является адъективная эволюция системы, которая позволяет системе оставаться в состоянии минимальной энергии в течение всего процесса. Это достигается за счёт медленного изменения параметра $s(t)$ в соответствии с адъективным теоремой:

$$\frac{ds}{dt} \ll \frac{\Delta^2}{\hbar \left\| \frac{dH}{ds} \right\|}, \tag{2.5}$$

где Δ — энергетический разрыв между основным и первым возбужденными состояниями гамильтониана.

2.2 Алгоритм

2.3 Сравнительные результаты тестирования

Заключение

Литература

[1] V. V. Nikonov, A. N. Tsirulev. *Pauli basis formalism in quantum computations*. Volume 8, No 3, pp. 1 – 14, 2020.
(*doi:10.26456/mmg/2020-831*)

[2] J. Preskill. *Quantum Computing in the NISQ era and beyond*. Quantum, vol. 2, p. 79, 2018.
(*quantum-journal:q-2018-08-06-79*)

[3] M. Cerezo, et al. *Variational Quantum Algorithms*. Nature Reviews Physics, vol. 3, pp. 625-644, 2021.
(*nature:42254-021-00348-9*)

[4] A. Peruzzo, et al. *A variational eigenvalue solver on a photonic quantum processor*. Nature Communications, vol. 5, p. 4213, 2014.
(*nature:ncomms5213*)

[5] E. Farhi, J. Goldstone, and S. Gutmann. *A Quantum Approximate Optimization Algorithm*. arXiv preprint arXiv:1411.4028, 2014.
(*arXiv:1411.4028*)

[6] J. R. McClean, et al. *The theory of variational hybrid quantum-classical algorithms*. New Journal of Physics, vol. 18, p. 023023, 2016.
(*iopscience:1367-2630-18-2-023023*)

[7] A. Kandala, et al. *Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets*. Nature, vol. 549, pp. 242-246, 2017.
(*nature:nature23879*)

[8] A. W. Harrow, A. Hassidim, and S. Lloyd. *Quantum algorithm for linear systems of equations*. Physical Review Letters, vol. 103, no. 15, p. 150502,

2009.
(*aps:PhysRevLett.103.150502*)

[9] J. Biamonte, et al. *Quantum machine learning*. Nature, vol. 549, pp. 195-202, 2017.
(*nature:nature23474*)

[10] A. A. Lopatin. *Квантовая механика и её приложения*. Санкт-Петербургский Государственный Университет.
(*math.spbu:user/gran/sb1/lopatin*)

[11] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, M. Head-Gordon. *Simulated Quantum Computation of Molecular Energies*. Science, vol. 309, no. 5741, pp. 1704-1707, 2005.
(*science:1113479*)

[12] M. Schuld, I. Sinayskiy, F. Petruccione. *An introduction to quantum machine learning*. Contemporary Physics, vol. 56, no. 2, pp. 172-185, 2015.
(*tandfonline:00107514.2014.964942*)

[13] A. Daskin, S. Kais. *Decomposition of unitary matrices for finding quantum circuits: Application to molecular Hamiltonians*. The Journal of Chemical Physics, vol. 141, no. 23, p. 234115, 2014.
(*aip:1.4904315*)

[14] J. Romero, R. Babbush, J. R. McClean, C. Hempel, P. J. Love, A. Aspuru-Guzik. *Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz*. Quantum Science and Technology, vol. 4, no. 1, p. 014008, 2018.
(*iopscience:2058-9565/aad3e4*)

[15] V. Havlicek, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, J. M. Gambetta. *Supervised learning with quantum-enhanced feature spaces*. Nature, vol. 567, pp. 209-212, 2019.
(*nature:s41586-019-0980-2*)

[16] N. Moll, P. Barkoutsos, L. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, et al. *Quantum*

optimization using variational algorithms on near-term quantum devices. Quantum Science and Technology, vol. 3, no. 3, p. 030503, 2018. (*iopscience:2058-9565/aab822*)

Приложение Python

```python
#constants/file_paths.py
import sys
from pathlib import Path

def get_base_path() -> Path:
    """Возвращает путь к директории с EXE (или к проекту в dev-режиме)."""
    if getattr(sys, "frozen", False):
        # Для скомпилированного EXE берём директорию исполняемого файла
        return Path(sys.argv[0]).parent
    else:
        # Для разработки возвращаем корень проекта
        return Path(__file__).parent.parent

# Путь к hamiltonian_operators.txt (внешний файл)
HAMILTONIAN_FILE_PATH = get_base_path() / "params" / "hamiltonian_operators.txt"

# Файл вывода создаётся в текущей рабочей директории
OUTPUT_FILE_PATH = get_base_path() / "output.log"


#constants/pauli.py
PAULI_MAP = {
    (1, 2): (1j, 3),
    (2, 1): (-1j, 3),
    (3, 1): (1j, 2),
    (1, 3): (-1j, 2),
    (2, 3): (1j, 1),
    (3, 2): (-1j, 1),
}


#params/hamiltonian_operators.txt
# Действительная часть | Мнимая часть | Строка Паули

2.0 0.0 03
1.0 0.0 30
-4.0 0.0 11
```

```python


#utils/console_and_print.py
from rich.console import Console
from typing import Any
from constants.file_paths import OUTPUT_FILE_PATH


def console_and_print(console: Console, message: Any) -> None:
    """
    Выводит результат выполнения программы в файл и консоль

    :param console: Объект Console для вывода в консоль
    :param message: Сообщение для вывода
    """
    console.print(message)
    with open(OUTPUT_FILE_PATH, "a", encoding="utf-8") as file:
        file.write(console.export_text() + "\n")


#utils/create_table.py
from rich.table import Table
from rich.panel import Panel
from rich import box
from typing import List, Dict, Any


def create_table(
    columns: List[Dict[str, str]],
    data: List[List[Any]],
    title: str,
    border_style: str = "yellow",
) -> Panel:
    table = Table(box=box.ROUNDED, border_style="yellow")
    for col in columns:
        table.add_column(
            col["name"],
            justify=col.get("justify", "default"),
            style=col.get("style", ""),
        )
    for row in data:
        table.add_row(*row)
    return Panel(table, title=title, border_style=border_style)


#utils/format_ansatz.py
from .format_complex_number import format_complex_number


def format_ansatz(
    pauli_operators: list[list[int]], result: dict[tuple, complex]
```

```
) -> tuple[str, str]:
    """
    Форматирует анзац в символьное и численное представление.

    :param pauli_operators: Список операторов Паули (списки индексов).
    :param result: Словарь с результатами вычислений {оператор: коэффициент}.
    :return: Символьное представление анзаца и его численное значение.
    """
    # Формируем U(θ)
    ansatz_symbolic = "U(θ) = " + " * ".join(
        [
            f"e^(iθ_{i+1}*{format_complex_number(op[0])}*σ_{''.join(map(str, op[1]))})"
            for i, op in enumerate(pauli_operators)
        ]
    )

    # Формируем U
    ansatz_numeric = "U = " + " + ".join(
        [
            f"{format_complex_number(c)}*{format_complex_number(op[0])}*σ_{''.join(op[1:])}"
            if isinstance(op, (list, tuple))
            and len(op) > 1
            and isinstance(op[0], complex)
            else f"{format_complex_number(c)}*σ_{''.join(map(str, op))}"
            for op, c in result.items()
        ]
    )

    return ansatz_symbolic, ansatz_numeric


#utils/format_complex_number.py
from sympy import re as sp_re, im as sp_im
from .format_number import format_number


def format_complex_number(c) -> str:
    """
    Универсальный форматировщик комплексных чисел для SymPy и стандартных типов.
    """
    # Извлечение компонентов через SymPy
    real = float(sp_re(c))
    imag = float(sp_im(c))

    real_str = format_number(real) if abs(real) > 1e-12 else ""
    imag_str = ""

```

```
    if abs(imag) > 1e-12:
        abs_imag = abs(imag)
        imag_value = format_number(abs_imag)

        # Специальные случаи для ±1
        if imag_value == "1":
            imag_str = "i" if imag > 0 else "-i"
        else:
            imag_sign = "" if imag > 0 else "-"
            imag_str = f"{imag_sign}{imag_value}i"

    # Сборка результата
    parts = []
    if real_str:
        parts.append(real_str)
    if imag_str:
        parts.append(imag_str)

    if not parts:
        return "0"

    # Корректное объединение компонентов
    result = parts[0]
    for part in parts[1:]:
        if part.startswith("-"):
            result += f"-{part[1:]}"
        else:
            result += f"+{part}"

    # Фикс артефактов форматирования
    return result.replace("+ -", "- ").replace("1i",
        "i").replace(".0i", "i")


#utils/format_number.py
def format_number(num: float | int) -> str:
    """
    Универсальный форматировщик чисел с точным определением целочисленны
        х значений
    и подавлением артефактов вычислений с плавающей точкой.
    """
    # Проверка на целое число с учетом погрешности вычислений
    if abs(num - round(num)) < 1e-15:
        return str(int(round(num)))

    # Форматирование с ручным управлением десятичными знаками
    s = f"{num:.14f}".rstrip("0").rstrip(".")

    # Исправление артефактов вида ".5" → "0.5"
    if s.startswith("."):
        s = "0" + s
```

```
185     elif s.startswith("-."):
186         s = s.replace("-.", "-0.")
187 
188     # Удаление лишних десятичных знаков после 4-го
189     if "." in s:
190         int_part, dec_part = s.split(".")
191         dec_part = dec_part[:4].ljust(4, "0").rstrip("0")
192         s = f"{int_part}.{dec_part}" if dec_part else int_part
193 
194     return s
195 
196 
197 #utils/get_operator_for_console.py
198 from .format_complex_number import format_complex_number
199 from typing import Union
200 
201 
202 def get_operator_for_console(c: Union[complex, float, int], i: str) ->
        str:
203     """
204     Функция создана для 'красивого' вывода оператора Паули в консоль.
205     Избегает ситуаций, когда у нас выводится конструкция вида '1*σ'
206 
207     :param c: Коэффициент оператора Паули.
208     :param i: Строка оператора Паули.
209     :return: Отформатированную строку.
210     """
211     if c == 1:
212         return f"σ_{i}"
213     else:
214         return f"{format_complex_number(c)}*σ_{i}"
215 
216 
217 #utils/initialize_environment.py
218 from rich.console import Console  # Для красивого вывода в консоль
219 from constants.file_paths import OUTPUT_FILE_PATH
220 
221 
222 def initialize_environment() -> Console:
223     """Инициализирует окружение и возвращает консольный объект."""
224     if OUTPUT_FILE_PATH.exists():
225         OUTPUT_FILE_PATH.unlink()
226     return Console(force_terminal=True, color_system="truecolor",
            record=True)
227 
228 
229 #utils/print_composition_table.py
230 from rich.console import Console  # Для красивого вывода в консоль
231 from typing import Tuple, List, Callable
232 from .console_and_print import console_and_print
233 from .create_table import create_table
```

```python
from .format_complex_number import format_complex_number


def print_composition_table(
    console: Console,
    pauli_compose: Callable[[List[int], List[int]], Tuple[complex, List[int]]],
    pauli_strings: List[List[int]],
) -> None:
    """Выводит таблицу композиций операторов Паули."""
    results = []
    for s1 in pauli_strings:
        for s2 in pauli_strings:
            coeff, product = pauli_compose(s1, s2)
            results.append((s1, s2, format_complex_number(coeff), product))

    table_data = [
        [str(s1), str(s2), str(h).lower(), str(p)] for s1, s2, h, p in results
    ]
    console_and_print(
        console,
        create_table(
            columns=[
                {"name": "Оператор 1", "style": "cyan", "justify": "center"},
                {"name": "Оператор 2", "style": "magenta", "justify": "center"},
                {"name": "Коэффициент", "style": "green", "justify": "center"},
                {"name": "Результат", "style": "red", "justify": "center"},
            ],
            data=table_data,
            title="Композиции операторов Паули",
            border_style="green",
        ),
    )


#utils/print_hamiltonian.py
from rich.console import Console
from rich.panel import Panel
from typing import Tuple, List
from .get_operator_for_console import get_operator_for_console
from .console_and_print import console_and_print


def print_hamiltonian(
    console: Console, pauli_operators: List[Tuple[complex, List[int]]]
```

```
) -> None:
    """Выводит представление гамильтониана в консоль."""
    hamiltonian_str = "H = " + " + ".join(
        [get_operator_for_console(c, ''.join(map(str, i))) for c, i in
           pauli_operators]
    )
    console_and_print(
        console,
        Panel(
            hamiltonian_str,
            title="[bold]Введенный гамильтониан[/bold]",
            border_style="green",
        ),
    )


#utils/print_pauli_table.py
from rich.console import Console  # Для красивого вывода в консоль
from typing import Tuple, List
from .format_complex_number import format_complex_number
from .console_and_print import console_and_print
from .create_table import create_table


def print_pauli_table(
    console: Console, pauli_operators: List[Tuple[complex, List[int]]]
) -> None:
    """Выводит таблицу операторов Паули."""
    table_data = [[format_complex_number(c), str(i)] for c, i in
       pauli_operators]
    console_and_print(
        console,
        create_table(
            columns=[
                {"name": "Коэффициент", "style": "cyan"},
                {"name": "Индекс", "style": "magenta", "justify":
                   "center"},
            ],
            data=table_data,
            title="Операторы Паули",
            border_style="purple",
        ),
    )


#utils/read_file_lines.py
from pathlib import Path


def read_file_lines(file_path, ignore_comments):
    """
```

```
326     Читает строки из файла, игнорируя комментарии (строки, начинающиеся
            с '#').
327
328     :param file_path: Путь к файлу.
329     :param ignore_comments: Игнорировать строки, начинающиеся с '#'.
330     :return: Список строк.
331     :raises FileNotFoundError: Если файл не найден.
332     """
333     file_path = Path(file_path) if not isinstance(file_path, Path) else
            file_path
334     if not file_path.exists():
335         raise FileNotFoundError(f"Файл {file_path} не найден.")
336     with open(file_path, "r") as file:
337         return [
338             line.strip()
339             for line in file
340             if not (ignore_comments and line.strip().startswith("#"))
341         ]
342
343
344 #utils/read_hamiltonian_data.py
345 import numpy as np  # Для работы с комплексными числами и математическим
        и операциями
346 from .read_file_lines import read_file_lines
347
348
349 def read_hamiltonian_data(file_path):
350     """
351     Читает данные из файла hamiltonian_operators.txt и возвращает два сп
            иска:
352     - Список операторов Паули в виде коэффициента оператора и строки Пау
            ли.
353     - Список строк операторов Паули.
354
355     :param file_path: Путь к файлу.
356     :return: (pauli_operators, pauli_strings)
357     :raises FileNotFoundError: Если файл не найден.
358     """
359     lines = read_file_lines(file_path, ignore_comments=False)
360     pauli_operators = []
361     pauli_strings = []
362     for line in lines:
363         parts = line.strip().split()
364         if len(parts) == 3:
365             real_part, imag_part, index_str = (
366                 float(parts[0]),
367                 float(parts[1]),
368                 str(parts[2]),
369             )
370             coefficient = np.complex128(real_part + imag_part * 1j)
371             index_list = [int(c) for c in index_str]
```

```
            if coefficient != 0:
                pauli_operators.append((coefficient, index_list))
            pauli_strings.append(index_list)
    return pauli_operators, pauli_strings


#program.py
import numpy as np
import sys
import io
from rich.progress import Progress, BarColumn, TextColumn, SpinnerColumn
from rich.panel import Panel
from typing import Tuple, List, Dict

# Импорт самописных util функций
from utils.console_and_print import console_and_print
from utils.print_pauli_table import print_pauli_table
from utils.read_hamiltonian_data import read_hamiltonian_data
from utils.print_hamiltonian import print_hamiltonian
from utils.print_composition_table import print_composition_table
from utils.format_ansatz import format_ansatz
from utils.initialize_environment import initialize_environment

# Импорт констант
from constants.file_paths import HAMILTONIAN_FILE_PATH
from constants.pauli import PAULI_MAP

# Установка кодировки для корректного вывода
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding="utf-8")
sys.stderr = io.TextIOWrapper(sys.stderr.buffer, encoding="utf-8")


def generate_random_theta(m: int) -> np.ndarray:
    """Генерирует массив из m случайных углов в диапазоне [0, 2π)."""
    return np.random.uniform(0, 2*np.pi, size=m).astype(np.float64)


def multiply_pauli(i: int, j: int) -> Tuple[complex, int]:
    """
    Вычисляет произведение базисных операторов Паули.
    Возвращает: (коэффициент, индекс результата)
    """
    if i == j:
        return (1, 0)
    if i == 0:
        return (1, j)
    if j == 0:
        return (1, i)
    return PAULI_MAP.get((i, j), (1, 0))


```

```python
def pauli_compose(s1: List[int], s2: List[int]) -> Tuple[complex, List[int]]:
    """
    Вычисляет композицию двух операторов Паули.
    Возвращает: (коэффициент, результирующий оператор)
    """
    coefficient = 1.0
    result = []
    for a, b in zip(s1, s2):
        coeff, idx = multiply_pauli(a, b)
        coefficient *= coeff
        result.append(idx)
    return coefficient, result


def calculate_ansatz(
    theta: np.ndarray, pauli_operators: List[Tuple[complex, List[int]]]
) -> Tuple[Dict[Tuple[int, ...], complex], str, str]:
    """
    Вычисляет анзац в виде произведения экспонент операторов Паули.
    Возвращает: (словарь операторов, символьное представление, численное представление)
    """
    operator_length = len(pauli_operators[0][1])
    result = {tuple([0] * operator_length): 1.0}

    for t, (_, op) in zip(theta, pauli_operators):
        cos_t = np.cos(t)
        sin_t = np.sin(t)
        new_result: Dict[Tuple[int, ...], complex] = {}

        for existing_op, existing_coeff in result.items():
            # Слагаемое с cos(θ)*I
            identity_coeff = existing_coeff * cos_t
            new_result[existing_op] = new_result.get(existing_op, 0) + identity_coeff

            # Слагаемое с i*sin(θ)*σ
            pauli_coeff = existing_coeff * 1j * sin_t
            compose_coeff, compose_op = pauli_compose(list(existing_op), op)
            final_coeff = pauli_coeff * compose_coeff
            final_op = tuple(compose_op)
            new_result[final_op] = new_result.get(final_op, 0) + final_coeff

        result = new_result

    symbolic_str, numeric_str = format_ansatz(pauli_operators, result)
    return result, symbolic_str, numeric_str

```

```python

def compute_uhu(
    u_dict: Dict[Tuple[int, ...], complex], h_terms:
        List[Tuple[complex, List[int]]]
) -> Dict[Tuple[int, ...], complex]:
    """
    Вычисляет оператор U† H U.
    Возвращает: словарь {оператор: коэффициент}
    """
    uhu_dict: Dict[Tuple[int, ...], complex] = {}

    for coeff_h, op_h in h_terms:
        for j_op, j_coeff in u_dict.items():
            conjugated_j_coeff = np.conj(j_coeff)
            c1, op_uh = pauli_compose(list(j_op), op_h)

            for k_op, k_coeff in u_dict.items():
                c2, op_uhu = pauli_compose(op_uh, list(k_op))
                total_coeff = conjugated_j_coeff * k_coeff * coeff_h *
                    c1 * c2

                # Стабилизация малых значений
                if abs(total_coeff) < 1e-12:
                    continue

                op_tuple = tuple(op_uhu)
                uhu_dict[op_tuple] = uhu_dict.get(op_tuple, 0) +
                    total_coeff
    return uhu_dict


def calculate_expectation(uhu_dict: Dict[Tuple[int, ...], complex]) ->
    float:
    """
    Вычисляет <0|U†HU|0> для состояния |0...0>.
    Возвращает: ожидаемое значение
    """
    expectation = 0.0
    for op, coeff in uhu_dict.items():
        if all(p in {0, 3} for p in op):
            expectation += coeff.real
    return expectation


def generate_neighbor_theta(
    current_theta: np.ndarray, step_size: float = 0.1
) -> np.ndarray:
    """Генерирует соседнее решение, добавляя случайное изменение к текущ
        ему theta."""
    perturbation = np.random.normal(scale=step_size,
        size=current_theta.shape)
```

```
    return np.clip(current_theta + perturbation, 0.0, 1.0)


def simulated_annealing(
    initial_theta: np.ndarray,
    pauli_operators: list,
    initial_temp: float = 1000.0,
    cooling_rate: float = 0.99,
    min_temp: float = 1e-5,
    num_iterations_per_temp: int = 500,
    step_size: float = 0.5,
) -> tuple:
    """Реализует алгоритм имитации отжига с термализацией."""
    current_theta = initial_theta.copy()
    best_theta = current_theta.copy()
    best_energy = float("inf")
    rng = np.random.default_rng()

    with Progress(
        SpinnerColumn(),
        TextColumn("[progress.description]{task.description}"),
        BarColumn(bar_width=None),
        TextColumn("[progress.percentage]{task.percentage:>3.0f}%"),
    ) as progress:
        task = progress.add_task("[cyan]Отжиг...", total=100)

        temp = initial_temp
        iteration = 0

        while temp > min_temp:
            for _ in range(num_iterations_per_temp):
                # Генерация соседнего решения с адаптивным шагом
                perturbation = rng.normal(0,
                    step_size*(temp/initial_temp),
                    size=current_theta.shape)
                neighbor_theta = (current_theta + perturbation) %
                    (2*np.pi)

                # Вычисление энергии нового состояния
                ansatz_dict, _, _ = calculate_ansatz(neighbor_theta,
                    pauli_operators)
                uhu_dict = compute_uhu(ansatz_dict, pauli_operators)
                current_energy = calculate_expectation(uhu_dict)

                # Критерий принятия решения
                energy_diff = current_energy - best_energy
                if energy_diff < 0 or rng.random() <
                    np.exp(-energy_diff / temp):
                     current_theta = neighbor_theta.copy()

                     if current_energy < best_energy:
```

```
560                         best_theta = current_theta.copy()
561                         best_energy = current_energy
562 
563                 iteration += 1
564                 if iteration % 100 == 0:
565                     progress.update(task, advance=1)
566 
567             temp *= cooling_rate
568 
569     return best_theta, best_energy
570 
571 
572 def main():
573     """Основная логика программы."""
574     console = initialize_environment()
575 
576     # Явная проверка существования файла
577     if not HAMILTONIAN_FILE_PATH.exists():
578         msg = (
579             f"Файл [bold]{HAMILTONIAN_FILE_PATH}[/] не найден!\n"
580             "Убедитесь, что рядом с EXE есть папка [bold]params[/] с фай
                лом [bold]hamiltonian_operators.txt[/]."
581         )
582         console_and_print(console, Panel(msg, border_style="red"))
583         return
584 
585     try:
586         pauli_operators, pauli_strings =
               read_hamiltonian_data(HAMILTONIAN_FILE_PATH)
587 
588         print_hamiltonian(console, pauli_operators)
589         print_pauli_table(console, pauli_operators)
590         print_composition_table(console, pauli_compose, pauli_strings)
591 
592     except FileNotFoundError:
593         console_and_print(
594             console,
595             Panel(
596                 f"[red]Файл {HAMILTONIAN_FILE_PATH} не найден[/red]",
                       border_style="red"
597             ),
598         )
599         return
600 
601     if len(pauli_operators) < 2:
602         console_and_print(
603             console,
604             Panel("[red]Требуется минимум 2 оператора Паули[/red]",
                   border_style="red"),
605         )
606         return
```

```
607
608     best_energy = float("inf")
609     best_result = None
610
611     # Собираем все результаты для анализа
612     all_results = []
613
614     for m in range(2, len(pauli_operators) + 1):
615         initial_theta = generate_random_theta(m)
616
617         optimized_theta, energy = simulated_annealing(
618             initial_theta=initial_theta,
619             pauli_operators=pauli_operators,
620             initial_temp=100.0,
621             cooling_rate=0.95,
622             min_temp=1e-3,
623             num_iterations_per_temp=100,
624             step_size=0.1,
625         )
626
627         all_results.append(
628             {
629                 "m": m,
630                 "theta": optimized_theta,
631                 "energy": energy,
632             }
633         )
634
635         # Обновляем лучший результат
636         if energy < best_energy:
637             best_energy = energy
638             best_result = all_results[-1]
639
640     _, ansatz_symbolic, ansatz_numeric = calculate_ansatz(
641         best_result["theta"], pauli_operators[: best_result["m"]]
642     )
643
644     console_and_print(
645         console,
646         Panel(
647             ansatz_symbolic,
648             title="[bold]Символьное представление анзаца[/]",
649             border_style="green",
650         ),
651     )
652
653     console_and_print(
654         console,
655         Panel(
656             ansatz_numeric,
657             title="[bold]Численное представление анзаца[/]",
```

```
658                 border_style="purple",
659             ),
660         )
661
662         console_and_print(
663             console,
664             Panel(
665                 f"{best_result['energy']:.6f}",
666                 title="[bold]Энергия (<0|U†HU|0> для состояния |0...0>)[/]",
667                 border_style="green",
668             ),
669         )
670
671         input('text')
672
673
674 if __name__ == "__main__":
675     main()
```