

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(СПбГУ)

Образовательная программа магистратуры
«Прикладные математика и физика»



Отчёт по учебной практике
Применение искусственных нейронных сетей к
решению спектральных задач квантовой механики

Выполнил студент
1 курса магистратуры
(группа 22.М21-фз)
Козлов Александр

Санкт-Петербург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 ТЕСТОВАЯ СПЕКТРАЛЬНАЯ ЗАДАЧА	6
1.1 ФОРМУЛИРОВКА	6
1.2 ИЗВЕСТНОЕ РЕШЕНИЕ	6
2 АРХИТЕКТУРА ПРОБНОЙ ФУНКЦИИ	7
2.1 АМПЛИТУДНАЯ ФУНКЦИЯ	7
2.2 НЕЙРОННАЯ СЕТЬ	7
3 ФУНКЦИОНАЛ ПОТЕРЬ	9
3.1 ВЫЧИСЛЕНИЕ МАТРИЧНЫХ ЭЛЕМЕНТОВ И СКАЛЯРНЫХ ПРОИЗВЕДЕНИЙ	9
4 ОБУЧЕНИЕ ПРОБНОЙ ФУНКЦИИ	10
4.1 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ . .	10
4.2 МЕТОД МНОГОМЕРНОЙ ОПТИМИЗАЦИИ	11
4.3 АЛГОРИТМ ОБУЧЕНИЯ ПРОБНОЙ ФУНКЦИИ	11
5 РЕЗУЛЬТАТЫ	12
5.1 3-МЕРНЫЙ СЛУЧАЙ	12
5.1.1 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ <code>custom</code>	13
5.1.2 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ <code>normal</code>	14
5.1.3 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ <code>uniform</code>	15
5.2 4-МЕРНЫЙ СЛУЧАЙ	15
5.2.1 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ <code>custom</code>	17
5.2.2 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ <code>normal</code>	19
5.2.3 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ <code>uniform</code>	19
5.3 5-МЕРНЫЙ СЛУЧАЙ	21
5.3.1 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ <code>custom</code>	22
5.3.2 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ <code>uniform</code>	23
ЗАКЛЮЧЕНИЕ	24

СПИСОК ЛИТЕРАТУРЫ	26
ПРИЛОЖЕНИЕ	27

ВВЕДЕНИЕ

Технологии машинного обучения все глубже проникают в жизнь человека, находя свое применение в широком спектре областей человеческой деятельности — от детектирования редких смертельных заболеваний до прогнозирования индексов климатических мод Земли. Неотъемлемой составляющей каждой технологии машинного обучения является используемая модель машинного обучения — некий «черный ящик», производящий трансформацию входных данных. В последние десятилетия наиболее широкое распространение получила такая модель машинного обучения, как искусственная нейронная сеть (НС). Главным преимуществом данной модели машинного обучения является большое количество свободных параметров, правильно подобрав которые, можно со сколь угодно высокой точностью (которая зависит от количества свободных параметров) аппроксимировать любую функциональную зависимость [1]. Стоит заметить, что из-за большого количества параметров, процесс их подбора долгое время оставался крайне тяжелой вычислительной задачей, успешно начать решать которую позволило развитие технологий многомерной оптимизации.

Ввиду того, что НС успешно решают многомерные задачи (обработка натурального языка, генерация изображений и т.д.), естественным представляется использовать НС при решении многомерных задач математической физики. В таком приложении зачастую используются полносвязные НС прямого распространения, которые получили название *physics-informed neural networks* (нейронные сети, знающие физику, НСЗФ) [2], хотя их отличие от остальных типов НС заключается лишь в форме минимизируемого функционала (в него добавлены имеющие физическую интерпретацию члены). Форма минимизируемого функционала (функционала потерь или целевого функционала) зависит от специфики решаемой физической задачи, так, если ставится задача получить приближенное решение некоторого уравнения и НС используется в качестве пробной функции, то в функционал потерь разумно включить норму невязки НС по уравнению, граничное условие, если таковое имеется, а также все прочие соотношения, которым должно удовлетворять решение.

В 90-ые годы прошлого века было предложено использовать НС для решения задач на собственные значения и собственные функции — в работе [3] предлагается решать стационарное уравнение Шредингера с помощью НС с одним скрытым слоем, включая в функционал потерь квадратичную норму невязки пробной функции по уравнению. Такой подход позволил находить основное состояние, а также несколько возбужденных состояний стационарных квантово-механических систем. Подход, применяемый в [3], подразумевает, что каждому состоянию отвечает отдельная НС, а состояния находятся последовательно, начиная с основного (при отыскании возбужденных состояний в функционал потерь включаются члены, отвечающие за ортогональность искомого состояния к уже найденным состояниям).

В работе [4] предложено использовать одну НС для отыскания сразу нескольких состояний, включая в функционал потерь члены, отвечающие за взаимную ортогональность рассматриваемых состояний, такой подход мог бы позволить значительно ускорить

решение спектральных задач высокой размерности. Кроме того, в данной работе уделяется внимание необходимости правильно задавать распределение точек в координатном пространстве, что крайне важно при решении задач в пространствах высокой размерности, и предлагается использовать в качестве плотности распределения точки среднюю по отыскиваемым состояниям плотность вероятности. Однако, работа [4] полна спорных моментов, главным из которых является используемый функционал потерь — в работе [5] приводится строгое доказательство того, что используемый в [4] функционал потерь имеет минимумы на функциях, не являющихся решениями задачи. Кроме того, в работе [5] произведена модернизация функционала потерь (в него добавлен член, отвечающий за невязку пробной функции по уравнению), чтобы тот достигал минимумы лишь на решении задачи, и с использованием модернизированного функционала потерь успешно решены спектральные квантово-механические задачи в размерностях 1 и 2. Целью настоящей работы является распространение успехов работы [5] на большие размерности.

1 ТЕСТОВАЯ СПЕКТРАЛЬНАЯ ЗАДАЧА

1.1 ФОРМУЛИРОВКА

В качестве тестового примера, на котором в дальнейшем будет отрабатываться рассматриваемая методика решения спектральных задач квантовой механики, будет выступать задача о квантово-механическом гармоническом осцилляторе. Такая задача легко формулируется в любых размерностях, аналитически решается и обладает хорошо разнесенным спектром, почему и была выбрана. Математически такая задача ставится следующим образом:

$$\hat{H}\psi(\mathbf{r}) = E\psi(\mathbf{r}), \quad \hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + \frac{k}{2}\sum_{d=1}^D|r_d|^2, \quad (1.1)$$

где \hat{H} — гамильтониан задачи, $\psi(\mathbf{r})$ — волновая функция задачи (волновая функция, которая удовлетворяет уравнению, называется собственной функцией), E — значение энергии (значение энергии, которое отвечает собственной функции, называется собственным значением), D — размерность координатного пространства. Требуется найти собственные функции и собственные значения. Далее будет рассматриваться безразмерное уравнение, полученное заменой $\hbar^2/m = 1$ и $k = 1$.

1.2 ИЗВЕСТНОЕ РЕШЕНИЕ

Задача о квантово-механическом осцилляторе широко известна и её аналитическое решение приведено во многих классических курсах квантовой механики, например, в [6]. Точный спектр такой задачи задается соотношением

$$E_{n_1, \dots, n_D} = \frac{D}{2} + \sum_{d=1}^D n_d, \quad (1.2)$$

где $n_d = 0, 1, 2, \dots$ для $d = \overline{1, D}$. Собственные функции задачи в одномерном случае имеют вид:

$$\psi_m(x) = \frac{1}{\sqrt{\sqrt{\pi} 2^m m!}} H_m(x) \exp(-x^2/2), \quad (1.3)$$

где m — номер состояния, а $H_m(x)$ — полином Эрмита [7]. В пространствах большей размерности собственные функции собираются из собственных функций одномерной задачи с точностью до нормировочного фактора следующим образом:

$$\psi_{n_1, \dots, n_D}(\mathbf{r}) = \prod_{d=1}^D \psi_{n_d}(r_d), \quad (1.4)$$

где $n_d = 0, 1, 2, \dots$ для $d = \overline{1, D}$, а $\psi_{n_d}(r_d)$ — собственная функция одномерной задачи.

2 АРХИТЕКТУРА ПРОБНОЙ ФУНКЦИИ

Пробная функция (ПФ) \mathcal{F} определяется как отображение из пространства $\mathbb{R}^{B \times D}$ в пространство $\mathbb{R}^{B \times M}$, где B — размер партии точек в координатном пространстве, D — размерность координатного пространства и M — число состояний, которые мы хотим найти. Таким образом, ПФ \mathcal{F} преобразует B точек в D -мерном пространстве в B значений M волновых функций в этих точках.

ПФ конструируется как произведение нейронной сети (НС) \mathcal{N} и амплитудной функции (АФ) \mathcal{A} . Использование НС обусловлено тем, что она является универсальным аппроксиматором, что позволит ПФ аппроксимировать любую функциональную зависимость, в том числе и решение спектральной задачи. АФ же используется из тех соображений, что ПФ должна удовлетворять граничным условиям и требуемым асимптотикам. Рассмотрим каждую из этих двух сущностей в отдельности.

2.1 АМПЛИТУДНАЯ ФУНКЦИЯ

АФ должна удовлетворять граничным условиям и известным асимптотикам решения. В дальнейшем будет рассматриваться тестовая задача (1.1), её основное состояние достаточно просто находится и волновая функция, отвечающая ему, широко известна — это гауссиан \mathcal{G} , который с точностью до нормировочного множителя имеет вид:

$$\mathcal{G}(\mathbf{r}) = \exp \left(-\frac{1}{2} \sum_{d=1}^D r_d^2 \right). \quad (2.1)$$

Гауссиан \mathcal{G} обеспечивает удовлетворение граничных условий на бесконечности и требуемых асимптотик (что легко видеть из аналитической формы решений (1.3)), поэтому он и был выбран в качестве АФ $\mathcal{F} = \mathcal{G}$.

2.2 НЕЙРОННАЯ СЕТЬ

В качестве НС использовалась полносвязная НС прямого распространения с 3-мя внутренними слоями, число нейронов на которых обозначается через N_h и может изменяться пользователем. Входной слой преобразует входные данные X из пространства $\mathbb{R}^{B \times D}$ в пространство $\mathbb{R}^{B \times N_h}$ следующим образом (используется подход, применяемый в фреймворке PyTorch; см. <https://pytorch.org/>):

$$Y^{(1)} = f(XW_{(1)}^T + B_{(1)}), \quad (2.2)$$

где f — функция активации (во всей НС используется одна и та же функция активации), $W_{(1)} \in \mathbb{R}^{N_h \times D}$ — матрица весов входного слоя, $B_{(1)} \in \mathbb{R}^{N_h}$ — вектор сдвигов входного слоя. Первый и второй внутренние слои преобразуют данные аналогичным образом:

$$Y^{(i+1)} = f(Y^{(i)}W_{(i+1)}^T + B_{(i+1)}), \quad i = 1, 2 \quad (2.3)$$

где $W_{(i+1)} \in \mathbb{R}^{N_h \times N_h}$ — матрица весов i -ого внутреннего слоя, $B_{(i+1)} \in \mathbb{R}^{N_h}$ — вектор сдвигов i -ого внутреннего слоя. Последний внутренний слой преобразует данные к выходу следующим образом:

$$Y = Y^{(4)} = f(Y^{(3)}W_{(4)}^T + B_{(4)}), \quad (2.4)$$

где $W_{(4)} \in \mathbb{R}^{M \times N_h}$ — матрица весов последнего внутреннего слоя, $B_{(4)} \in \mathbb{R}^M$ — вектор сдвигов последнего внутреннего слоя.

Все матрицы весов W и вектора сдвигов B являются обучаемыми параметрами НС и подбираются в ходе ее обучения. Общее число свободных параметров у НС с такой архитектурой составляет

$$(D \times N_h + N_h) + 2 \cdot (N_h \times N_h + N_h) + (M \times N_h + M).$$

Таким образом, число обучаемых параметров линейно по размерности координатного пространства D , линейно по числу отыскиваемых состояний M и квадратично по числу нейронов в каждом из внутренних слоев N_h .

3 ФУНКЦИОНАЛ ПОТЕРЬ

В данном разделе рассматривается состав функционала потерь и объясняется смысл каждой из составляющих функционал потерь частей. Функционал потерь строится, как и в работе [5], из четырех частей:

1. Члена $R = \sum_{m=1}^M \left\| \hat{H}\psi_m - E_m\psi_m \right\|^2 / \|\psi_m\|^2$, отвечающего за невязку по уравнению;
2. Члена $A = \sum_{m=1}^M (\|\psi_m\|^2 - 1)^2$, отвечающего за нормировку пробных функций;
3. Члена $B = \sum_{m_1=2}^M \sum_{m_2=1}^{m_1} \langle \psi_{m_1} | \psi_{m_2} \rangle / \|\psi_{m_1}\|^2 / \|\psi_{m_2}\|^2$, отвечающего за ортогонализацию пробных функций;
4. Члена $E = \sum_{m=1}^M E_m$, отвечающего за минимизацию рассматриваемых собственных значений.

Эти части суммируются с соответствующими им весовыми коэффициентами и формируют итоговый функционал потерь:

$$\mathcal{L} = w_R R + w_A A + w_B B + w_E E. \quad (3.1)$$

3.1 ВЫЧИСЛЕНИЕ МАТРИЧНЫХ ЭЛЕМЕНТОВ И СКАЛЯРНЫХ ПРОИЗВЕДЕНИЙ

Для расчета функционала потерь (3.1) необходимо уметь вычислять матричные элементы и скалярное произведение. Если множество точек мощностью B в координатном пространстве, на котором оценивается значение функционала потерь, подчиняется некому распределению с функцией плотности вероятности W , то согласно методу выборки по важности матричный элемент некоторого оператора \hat{O} можно вычислить следующим образом:

$$\langle \psi_{m_1} | \hat{O} | \psi_{m_2} \rangle = \frac{1}{B} \sum_{b=1}^B \frac{\psi_{m_1}(\mathbf{r}_b) \cdot (\hat{O}\psi_{m_2})|_{\mathbf{r}_b}}{W(\mathbf{r}_b)}, \quad (3.2)$$

где $\{\mathbf{r}_b\}_{b=1}^B$ — точки в координатном пространстве, подчиняющиеся распределению с плотностью вероятности W . Аналогично вычисляются скалярные произведения и квадрат нормы функций в координатном пространстве.

4 ОБУЧЕНИЕ ПРОБНОЙ ФУНКЦИИ

В данном разделе будет приведена информация о процессе обучения пробной функции, будут рассмотрены такие темы, как семплирование точек в координатном пространстве и используемый метод многомерной оптимизации, кроме того, будет приведен краткий алгоритм обучения пробной функции.

4.1 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ

В настоящей работе, в отличие от работы [5], используются наборы точек в координатном пространстве, распределенные неравномерно. В качестве плотности вероятности распределения точек в координатном пространстве рассматривалось несколько функций. Во-первых, рассматривалось обычное равномерное распределение (**uniform**) точек в гиперкубе со стороной **std** и центром в начале координат. Во-вторых, рассматривалось гауссово нормальное распределение (**normal**)

$$W(\mathbf{r}) = \frac{1}{(\text{std} \sqrt{2\pi})^D} \times \exp \left[-\frac{1}{2} \sum_{d=1}^D \frac{r_d^2}{\text{std}^2} \right]. \quad (4.1)$$

В-третьих, в качестве весовой функции был взят усредненный по состояниям квадрат волновой функции (**custom**), то есть функция

$$W(\mathbf{r}) = \frac{1}{M} \sum_{m=1}^M |\psi_m(\mathbf{r})|^2. \quad (4.2)$$

Данная идея предлагается в работе [4].

Каждую итерацию обучения в начале итерации происходит трансформация набора точек в координатном пространстве таким образом, чтобы набор точек подчинялся распределению с плотностью вероятности, которая пропорциональна актуальной функции W . Для этого используется алгоритм Метрополиса [8], при этом набор точек в координатном пространстве на первой итерации обучения генерируется таким образом, что точки равномерно распределены в некоторой области. В Листинге 1 представлена реализация одного шага алгоритма Метрополиса на языке Python с использованием фреймворка PyTorch.

Листинг 1: Функция, осуществляющая один шаг алгоритма Метрополиса.

```
1 def MetropolisAlghorithmStep(x, f):
2     """One step of the Metropolis algorithm, it return the updated sample x"""
3     global parameters
4     x_prime = x + parameters["epsilon"] * (2 * torch.rand_like(x).to(device) - 1)
5     alpha = f(x_prime).to(device) / f(x).to(device)
6     doesXMove = torch.rand((x.shape[0])).to(device) <= alpha
7     x_prime = (
8         torch.mul(doesXMove.int(), x_prime.t()).t()
9         + torch.mul((1 - doesXMove.int()), x.t()).t()
10    )
11    return x_prime
```

4.2 МЕТОД МНОГОМЕРНОЙ ОПТИМИЗАЦИИ

В качестве метода многомерной оптимизации был избран метод AdamW [9], который выгодно отличается от метода Adam [10], использовавшегося в работе [5], тем, что подходит к проблеме «вшитой в алгоритм L2-регуляризации» (в обоих алгоритмах есть возможность включить «вшитую L2-регуляризацию», но производятся они по-разному, в этом и отличаются методы оптимизации) более правильно и позволяет лучше избегать переобучения нейросетевых моделей.

4.3 АЛГОРИТМ ОБУЧЕНИЯ ПРОБНОЙ ФУНКЦИИ

Пусть инициализирована пробная функция \mathcal{F} , чтобы обучить её в течение **STEPS** итерация требуется выполнить **STEPS** раз следующую последовательность действий:

1. Если итерация первая, то происходит генерация набора точек в координатном пространстве, при этом точки генерируются равномерно распределенные в кубе со стороной `std` (это гиперпараметр модели); если же итерация не первая, то к набору точек в координатном пространстве несколько раз применяет шаг алгоритма Метрополиса (с функцией плотности распределения W из (4.2));
2. Рассчитывается функционал потерь (3.1);
3. Вызывается оптимизатор (AdamW), изменяющий обучаемые параметры пробной функции из принципа минимизации функционала потерь.

5 РЕЗУЛЬТАТЫ

В данном разделе будут приведены результаты обучения пробных функций в различных размерностях. С помощью пробной функции решается тестовая задача (1.1) в соответствующей размерности. Исходный код пробной функции и процедуры ее обучения приведен в Приложении в Листинге 2.

5.1 3-МЕРНЫЙ СЛУЧАЙ

Для решения 3-мерной задачи была выбрана пробная функция с числом нейронов на каждом внутреннем слое $N_h = 100$. Отыскивалось $M = 5$ нижних по энергиям состояний. Весовые факторы в функционале потерь (3.1) были выбраны следующие:

$$w_R = 1, w_A = 1, w_B = 40, w_E = 1, \quad (5.1)$$

что соответствует выбору весовых множителей в функционале потерь в работе [5]. Полный список значений гиперпараметров пробной функции, которая использовалась при решении задачи в 3-мерном случае, приведен в Таблице 1. Обучение модели происходило на сервисе Kaggle (см. <https://www.kaggle.com/>) с графическим ускорителем Tesla P100-PCIE-16GB, обладающим 15.9GB памяти. Были рассмотрены варианты обучения проб-

Имя	Значение
Функция активации	sin
Амплитудная функция	$\exp\left(-\sum_{d=1}^d x^2/2\right)$
Число внутренних слоев	3
N_h	100
D	3
M	5
std	3
B	2^{13}
learningRate	10^{-3}
weightDecay	10^{-3}
w_R	1
w_A	1
w_B	40
w_E	1
ε	1

Таблица 1: Гиперпараметры пробной функции, которая использовалась при решении задачи о квантово-механическом гармоническом осцилляторе в 3-мерном случае.

ной функции с различным методом семплирования точек в координатном пространстве: `custom`, `normal` и `uniform`.

5.1.1 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ `custom`

Пробная функция обучалась в течение $\text{STEPS} = 15000$ итераций, что заняло 40 минут и 43 секунды. Зависимость оценочных значений собственных значений задачи от номера итерации обучения приведена на верхней панели Рис. 1. Видно, что за 15000 итераций обучения пробная функция приближается к решению задачи. Оценочные значения собственных значений на последней итерации обучения, а также их относительные ошибки приведены в Таблице 2. Видно, что нижние 4 состояния в результате обучения обеспечивают относительную ошибку по собственным значениям примерно в 0.5%, в то время как 4-ое состояние обеспечивает относительную ошибку выше — примерно в 3%.

На нижней панели Рис. 1 приведена зависимость значений членов функционала потерь от номера итерации обучения, что позволяет заключить, что на первых этапах обучения (до 4000-ой итерации) оптимизировались лишь члены, отвечающие за невязку по уравнению (R-term) и ортогонализацию (B-term); после этого уменьшение члена, отвечающего за ортогонализацию прекращается, он начинает возрастать, в то время как член, отвечающий за выполнение нормировки (A-term), начинает уменьшаться; на 10000-ой итерации оба эти члена стабилизируются и более сильно не меняются, а уменьшение члена, отвечающего за невязку по уравнению (R-term), продолжается до самого завершения обучения.

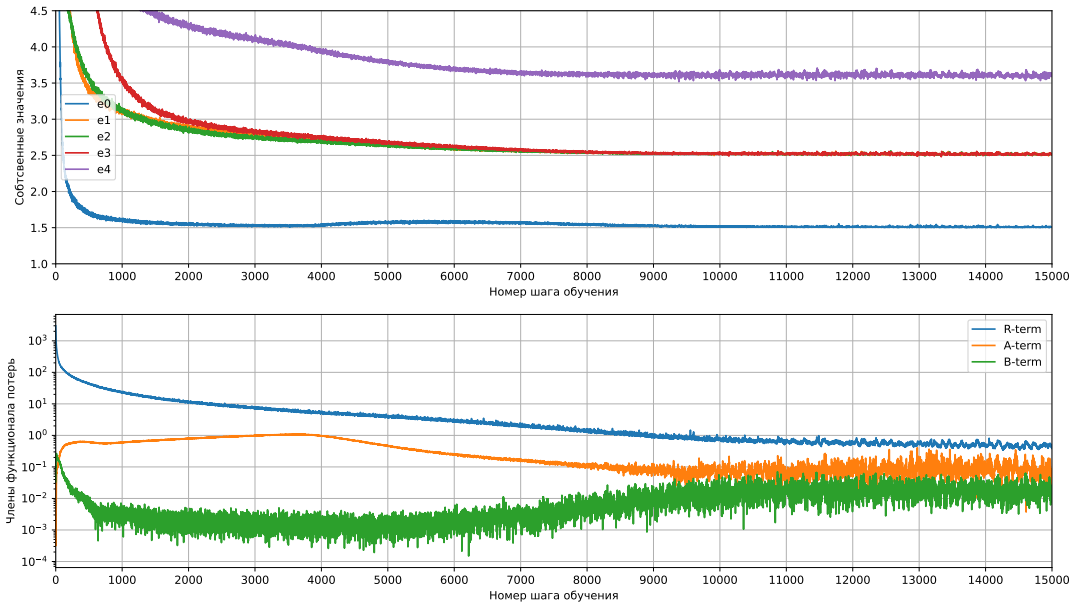


Рис. 1: (Верхняя панель): Зависимость оценочных значений собственных значений задачи (1.1) в 3-мерном случае от номера итерации обучения при семплировании `custom`; (нижняя панель): зависимость значений членов функционала потерь от номера итерации обучения при семплировании `custom`.

m	E_m^{approx}	$\Delta E_m / E_m$
0	1.5078	0.0052
1	2.5137	0.0055
2	2.5160	0.0064
3	2.5113	0.0045
4	3.6113	0.0318

Таблица 2: Оценочные значения собственных значений на последней итерации обучения и их относительные ошибки, полученные в результате обучения пробной функции с гиперпараметрами, указанными в Таблице 1, в течение $STEPS = 15000$ итераций при семплировании **custom**.

5.1.2 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ **normal**

Пробная функция обучалась в течение $STEPS = 15000$ итераций, что заняло 36 минут и 13 секунд. Зависимость оценочных значений собственных значений задачи от номера итерации обучения приведена на верхней панели Рис. 2. Видно, что за 15000 итераций обучения пробная функция приближается к решению задачи. Оценочные значения собственных значений на последней итерации обучения, а также их относительные ошибки приведены в Таблице 3. Видно, что основное состояние обеспечило ошибку в 5%, следующие 3 состояния — ошибку в 1-2%, в то время как 4-ое состояние обеспечивает относительную ошибку примерно в 4%.

На нижней панели Рис. 2 приведена зависимость значений членов функционала потерь от номера итерации обучения, что позволяет заключить, что в ходе обучения члены функционала потерь вели себя примерно так же, как и было при семплировании **custom**. Однако, при семплировании **normal** дисперсия оценочных значений энергий значительно выше, чем дисперсия оценочных значений энергий при использовании семплирования **custom**, то есть семплирование **custom** дает более плавную сходимость к решению, чем семплирование с неизменной функцией распределения **normal**.

m	E_m^{approx}	$\Delta E_m / E_m$
0	1.5751	0.0501
1	2.5203	0.0081
2	2.5398	0.0159
3	2.5609	0.0244
4	3.6561	0.0446

Таблица 3: Оценочные значения собственных значений на последней итерации обучения и их относительные ошибки, полученные в результате обучения пробной функции с гиперпараметрами, указанными в Таблице 1, в течение $STEPS = 15000$ итераций при семплировании **normal**.

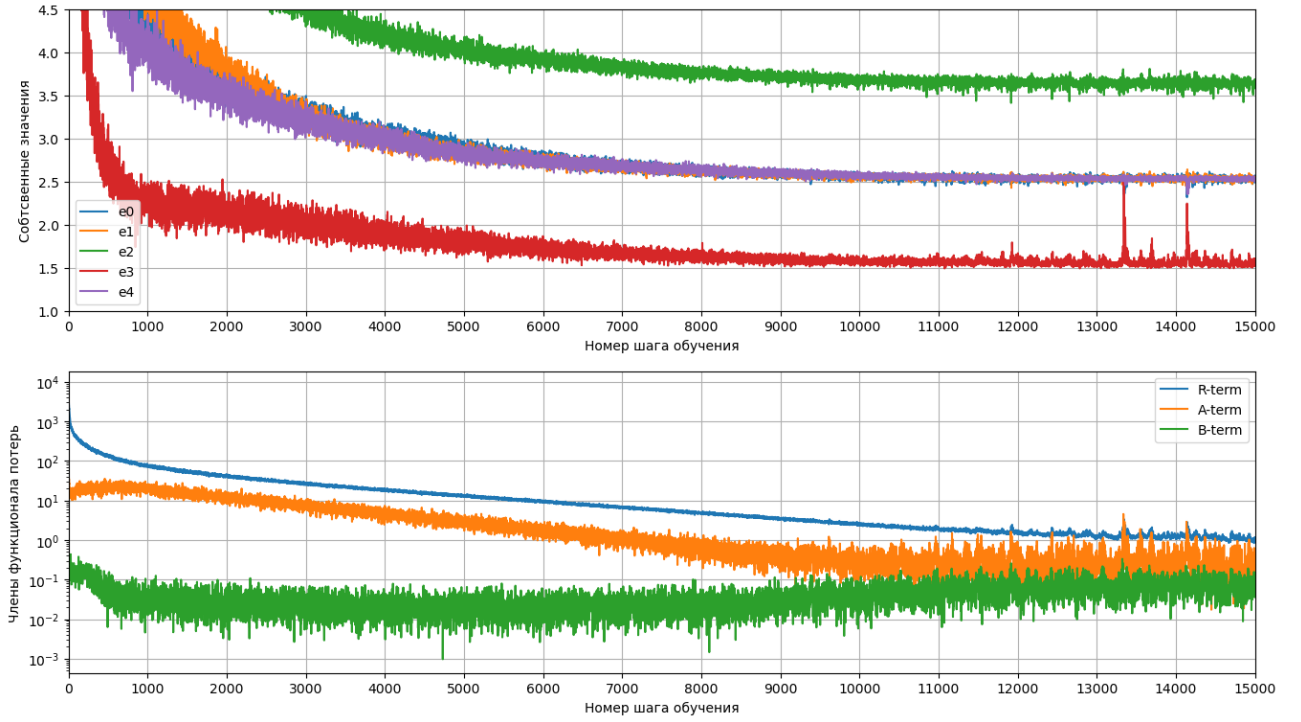


Рис. 2: (Верхняя панель): Зависимость оценочных значений собственных значений задачи (1.1) в 3-мерном случае от номера итерации обучения при семплировании `normal`; (нижняя панель): зависимость значений членов функционала потерь от номера итерации обучения при семплировании `normal`.

5.1.3 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ `uniform`

Пробная функция обучалась в течение $\text{STEPS} = 15000$ итераций, что заняло 35 минут и 18 секунд. Зависимость оценочных значений собственных значений задачи от номера итерации обучения приведена на верхней панели Рис. 3. Видно, что за 15000 итераций обучения пробная функция приближается к решению задачи. Оценочные значения собственных значений на последней итерации обучения, а также их относительные ошибки приведены в Таблице 4. Видно, что основное состояние обеспечило ошибку в 0.2%, следующие 3 состояния — ошибку в 1-2%, в то время как 4-ое состояние обеспечивает относительную ошибку примерно в 1.5%.

На нижней панели Рис. 3 приведена зависимость значений членов функционала потерь от номера итерации обучения, что позволяет заключить, что член, отвечающий за нормировку (A-term) почти не изменяется на протяжении обучения, а остальные члены вели себя так же, как и в случае семплирования `custom`. Дисперсия оценочных значений энергий для возбужденных состояний значительно увеличивается, начиная с итераций обучения где-то между 11000–12000.

5.2 4-МЕРНЫЙ СЛУЧАЙ

Для решения 4-мерной задачи была выбрана пробная функция с числом нейронов на каждом внутреннем слое $N_h = 100$. Отыскивалось $M = 5$ нижних по энергиям

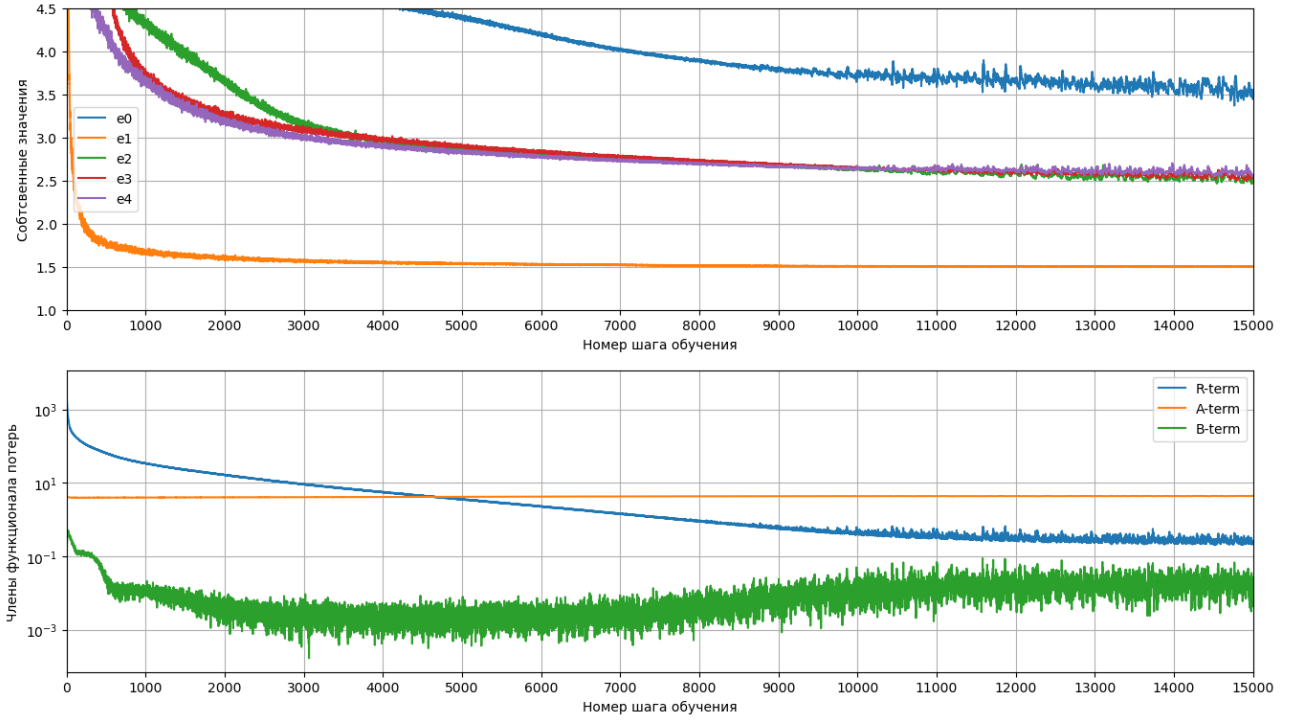


Рис. 3: (Верхняя панель): Зависимость оценочных значений собственных значений задачи (1.1) в 3-мерном случае от номера итерации обучения при семплировании `uniform`; (нижняя панель): зависимость значений членов функционала потерь от номера итерации обучения при семплировании `uniform`.

m	E_m^{approx}	$\Delta E_m / E_m$
0	1.5028	0.0018
1	2.5213	0.0085
2	2.5539	0.0215
3	2.5662	0.0265
4	3.4464	0.0152

Таблица 4: Оценочные значения собственных значений на последней итерации обучения и их относительные ошибки, полученные в результате обучения пробной функции с гиперпараметрами, указанными в Таблице 1, в течение $STEPS = 15000$ итераций при семплировании `uniform`.

состояний. Весовые факторы в функционале потерь (3.1) были выбраны следующие:

$$w_R = 1, w_A = 1, w_B = 40, w_E = 1, \quad (5.2)$$

что соответствует выбору весовых множителей в функционале потерь в работе [5]. Полный список значений гиперпараметров пробной функции, которая использовалась при решении задачи в 4-мерном случае, приведен в Таблице 5. Обучение модели происходило на сервисе Kaggle (см. <https://www.kaggle.com/>) с графическим ускорителем Tesla P100-PCIE-16GB, обладающим 15.9GB памяти. Были рассмотрены варианты обучения проб-

Имя	Значение
Функция активации	\sin
Амплитудная функция	$\exp\left(-\sum_{d=1}^d x^2/2\right)$
Число внутренних слоев	3
N_h	100
D	4
M	5
std	3
B	2^{14}
learningRate	10^{-3}
weightDecay	10^{-3}
w_R	1
w_A	1
w_B	40
w_E	1
ε	1

Таблица 5: Гиперпараметры пробной функции, которая использовалась при решении задачи о квантово-механическом гармоническом осцилляторе в 4-мерном случае.

ной функции с различным методом семплирования точек в координатном пространстве: `custom`, `normal` и `uniform`.

5.2.1 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ `custom`

Пробная функция обучалась в течение `STEPS = 15000` итераций, что заняло 1 час 23 минуты и 11 секунд. Зависимость оценочных значений собственных значений задачи от номера итерации обучения приведена на верхней панели Рис. 4. Видно, что за 15000 итераций обучения пробная функция приближается к решению задачи. Оценочные значения собственных значений на последней итерации обучения, а также их относительные ошибки приведены в Таблице 6. Видно, что энергия основного состояния определилась с относительной погрешностью в 0.4%, а энергии первого возбужденного состояния определились с точностью примерно в 1%.

На нижней панели Рис. 4 приведена зависимость значений членов функционала потерь от номера итерации обучения, что позволяет заключить, что на первых этапах

обучения (примерно до 2000-ой итерации) оптимизировались лишь члены, отвечающие за невязку по уравнению (R-term) и ортогонализацию (B-term); после этого начинает уменьшаться и член, отвечающий за нормировку пробной функции (A-term); на примерно 5700-ой итерации член, отвечающий на нормировку, прекращает уменьшаться, возрастая до примерно 7500-ой итерации, примерно в это же время член, отвечающий за ортогонализацию, прекращает уменьшаться и начинает возрастать, что он делает до самого конца обучения; член, отвечающий за нормировку, примерно с 7500-ой итерации начинает уменьшаться, что делает до самого окончания обучения; член, отвечающий за невязку по уравнению, все время обучения уменьшается.

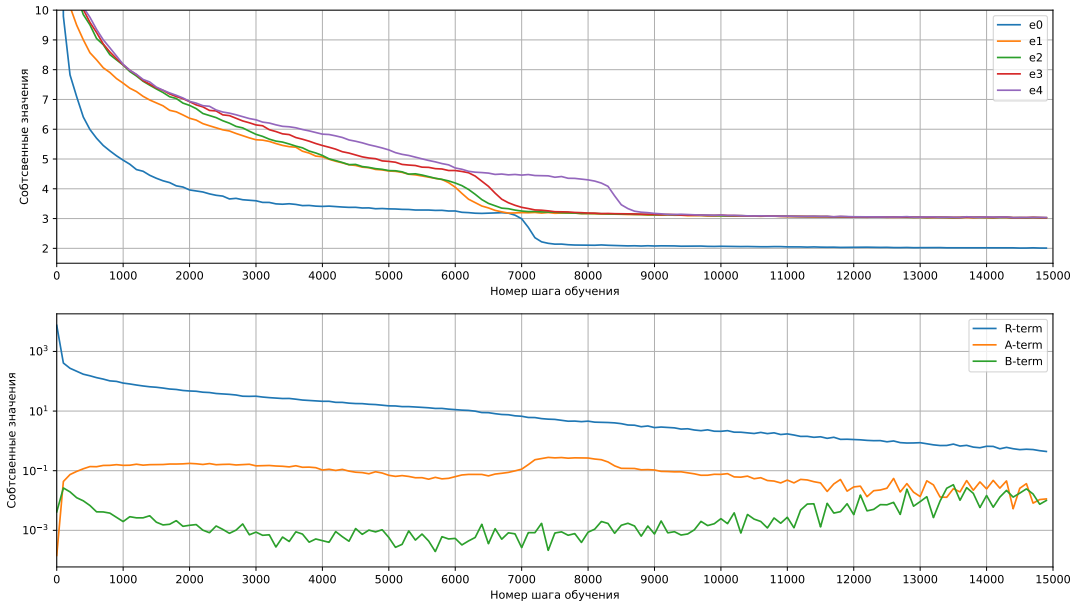


Рис. 4: (Верхняя панель): Зависимость оценочных значений собственных значений задачи (1.1) в 4-мерном случае от номера итерации обучения; (нижняя панель): зависимость значений членов функционала потерь от номера итерации обучения.

m	E_m^{approx}	$\Delta E_m / E_m$
0	2.0081	0.0040
1	3.0259	0.0086
2	3.0266	0.0089
3	3.0267	0.0089
4	3.0350	0.0117

Таблица 6: Оценочные значения собственных значений на последней итерации обучения и их относительные ошибки, полученные в результате обучения пробной функции с гиперпараметрами, указанными в Таблице 5, в течение $STEPS = 15000$ итераций.

5.2.2 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ `normal`

Пробная функция обучалась в течение `STEPS = 15000` итераций, что заняло 1 час 18 минут и 26 секунд. Зависимость оценочных значений собственных значений задачи от номера итерации обучения приведена на верхней панели Рис. 5. Видно, что за 15000 итераций обучения пробная функция не приближается к решению задачи. Оценочные значения собственных значений на последней итерации обучения, а также их относительные ошибки приведены в Таблице 7. Видно, что все энергии далеки от верных значений.

На нижней панели Рис. 5 приведена зависимость значений членов функционала потерь от номера итерации обучения, что позволяет заключить, что в ходе обучения члены функционала потерь уменьшались, однако, член, отвечающий за невязку по уравнению (R-term) за 15000 итераций обучения так и остался больше 1.

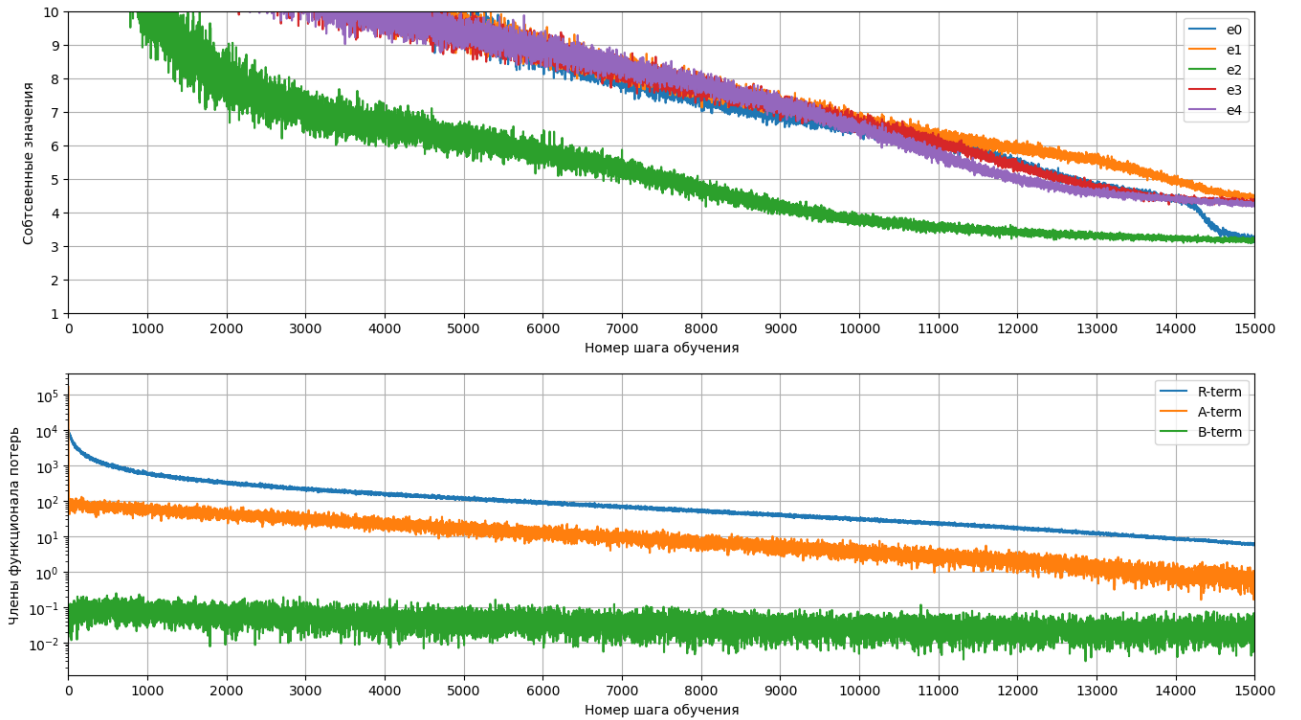


Рис. 5: (Верхняя панель): Зависимость оценочных значений собственных значений задачи (1.1) в 3-мерном случае от номера итерации обучения при семплировании `normal`; (нижняя панель): зависимость значений членов функционала потерь от номера итерации обучения при семплировании `normal`.

5.2.3 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ `uniform`

Пробная функция обучалась в течение `STEPS = 15000` итераций, что заняло 1 час 15 минут и 28 секунд. Зависимость оценочных значений собственных значений задачи от номера итерации обучения приведена на верхней панели Рис. 6. Видно, что за 15000 итераций обучения пробная функция приближается к решению задачи. Оценочные значения собственных значений на последней итерации обучения, а также их относительные

m	E_m^{approx}	$\Delta E_m / E_m$
0	3.1124	0.5562
1	3.1714	0.0571
2	4.2004	0.4001
3	4.2544	0.4181
4	4.4275	0.4758

Таблица 7: Оценочные значения собственных значений на последней итерации обучения и их относительные ошибки, полученные в результате обучения пробной функции с гиперпараметрами, указанными в Таблице 5, в течение $STEPS = 15000$ итераций при семплировании **normal**.

ошибки приведены в Таблице 8. Видно, что основное состояние обеспечило ошибку в 0.3%, следующие 2 состояния — ошибку в 2%, в то время как 3-е и 4-е состояния обеспечивают относительную ошибку примерно в 4-5%.

На нижней панели Рис. 6 приведена зависимость значений членов функционала потерь от номера итерации обучения, что позволяет заключить, что член, отвечающий за нормировку (A-term) почти не изменяется на протяжении обучения, а остальные члены вели себя так же, как и в случае семплирования **custom**. Дисперсия оценочных значений энергий для возбужденных состояний значительно увеличивается, начиная с итераций обучения где-то между 10000–11000.

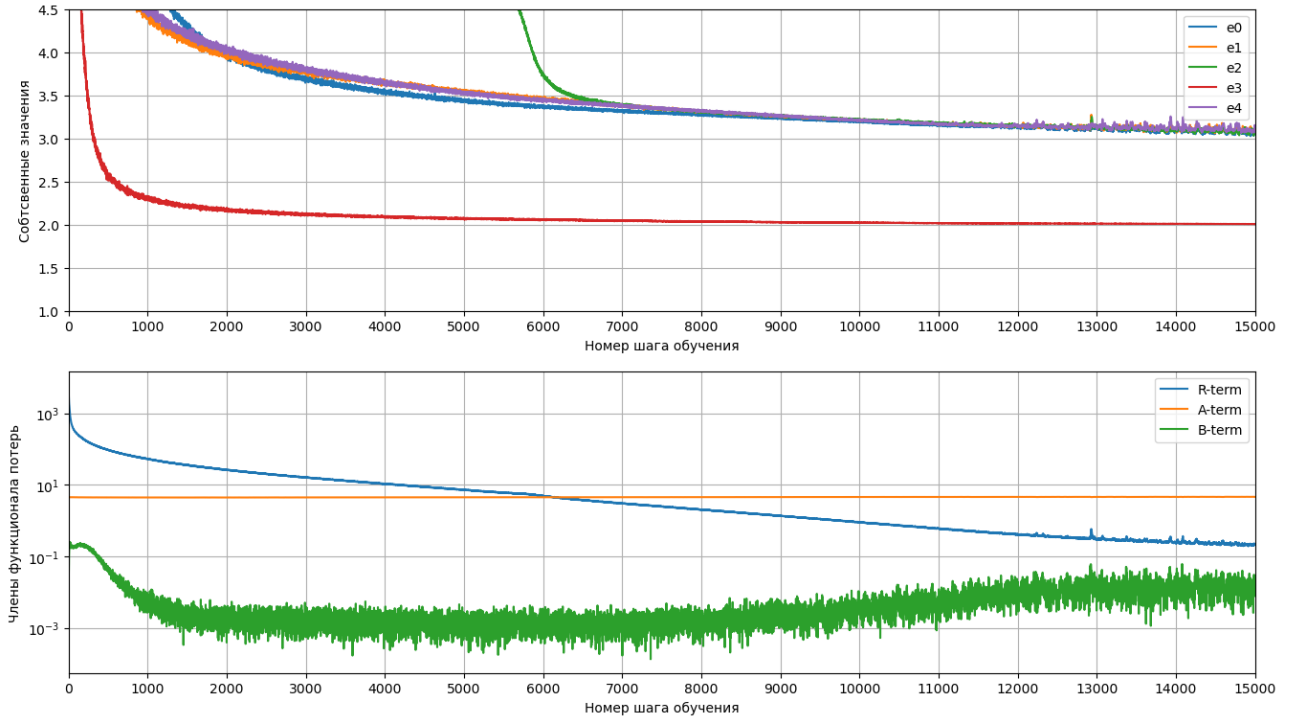


Рис. 6: (Верхняя панель): Зависимость оценочных значений собственных значений задачи (1.1) в 4-мерном случае от номера итерации обучения при семплировании **uniform**; (нижняя панель): зависимость значений членов функционала потерь от номера итерации обучения при семплировании **uniform**.

m	E_m^{approx}	$\Delta E_m/E_m$
0	2.00556	0.0027
1	3.0714	0.0238
2	3.0781	0.0260
3	3.1214	0.0404
4	3.1419	0.0473

Таблица 8: Оценочные значения собственных значений на последней итерации обучения и их относительные ошибки, полученные в результате обучения пробной функции с гиперпараметрами, указанными в Таблице 5, в течение $STEPS = 15000$ итераций при семплировании **uniform**.

5.3 5-МЕРНЫЙ СЛУЧАЙ

Для решения 5-мерной задачи была выбрана пробная функция с числом нейронов на каждом внутреннем слое $N_h = 100$. Отыскивалось $M = 5$ нижних по энергиям состояний. Весовые факторы в функционале потерь (3.1) были выбраны следующие:

$$w_R = 1, w_A = 1, w_B = 40, w_E = 1, \quad (5.3)$$

что соответствует выбору весовых множителей в функционале потерь в работе [5]. Полный список значений гиперпараметров пробной функции, которая использовалась при решении задачи в 5-мерном случае, приведен в Таблице 9. Обучение модели происходило на сервисе Kaggle (см. <https://www.kaggle.com/>) с графическим ускорителем Tesla P100-PCIE-16GB, обладающим 15.9GB памяти. Были рассмотрены варианты обучения проб-

Имя	Значение
Функция активации	\sin
Амплитудная функция	$\exp\left(-\sum_{d=1}^d x^2/2\right)$
Число внутренних слоев	3
N_h	100
D	5
M	5
std	3
B	2^{15}
learningRate	10^{-3}
weightDecay	10^{-3}
w_R	1
w_A	1
w_B	40
w_E	1
ε	1

Таблица 9: Гиперпараметры пробной функции, которая использовалась при решении задачи о квантово-механическом гармоническом осцилляторе в 5-мерном случае.

ной функции с различным методом семплирования точек в координатном пространстве:

`custom` и `uniform`. Метод семплирования `normal` не рассматривался, так как в размерности 4 его использование не привело к удовлетворительным результатам.

5.3.1 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ `custom`

Данная пробная функция обучалась в течение $\text{STEPS} = 15000$ итераций, что заняло 3 часа 11 минут и 43 секунды. Зависимость оценочных значений собственных значений задачи от номера итерации обучения приведена на верхней панели Рис. 7. Видно, что за 15000 итераций обучения пробная функция приближается к решению задачи. Оценочные значения собственных значений на последней итерации обучения, а также их относительные ошибки приведены в Таблице 10. Видно, что энергия основного состояния определена с относительной погрешностью в 0.1%, а энергии первого возбужденного состояния определились с точностью от 0.1% до 0.2%.

На нижней панели Рис. 7 приведена зависимость значений членов функционала потерь от номера итерации обучения. Виден общий для предыдущих примеров паттерн — член R-term убывает все время обучения, член A-term сперва возрастает, потом начинает убывать, член B-term сперва стремительно убывает, а затем возрастает. При этом заметна некоторая ступенчатость изменения члена функционала потерь, ответственного за нормализацию пробной функции (A-term), на том этапе обучения, когда оценочные значения резко приходят к точным значениям (итерации обучения от 7000 до 10000).

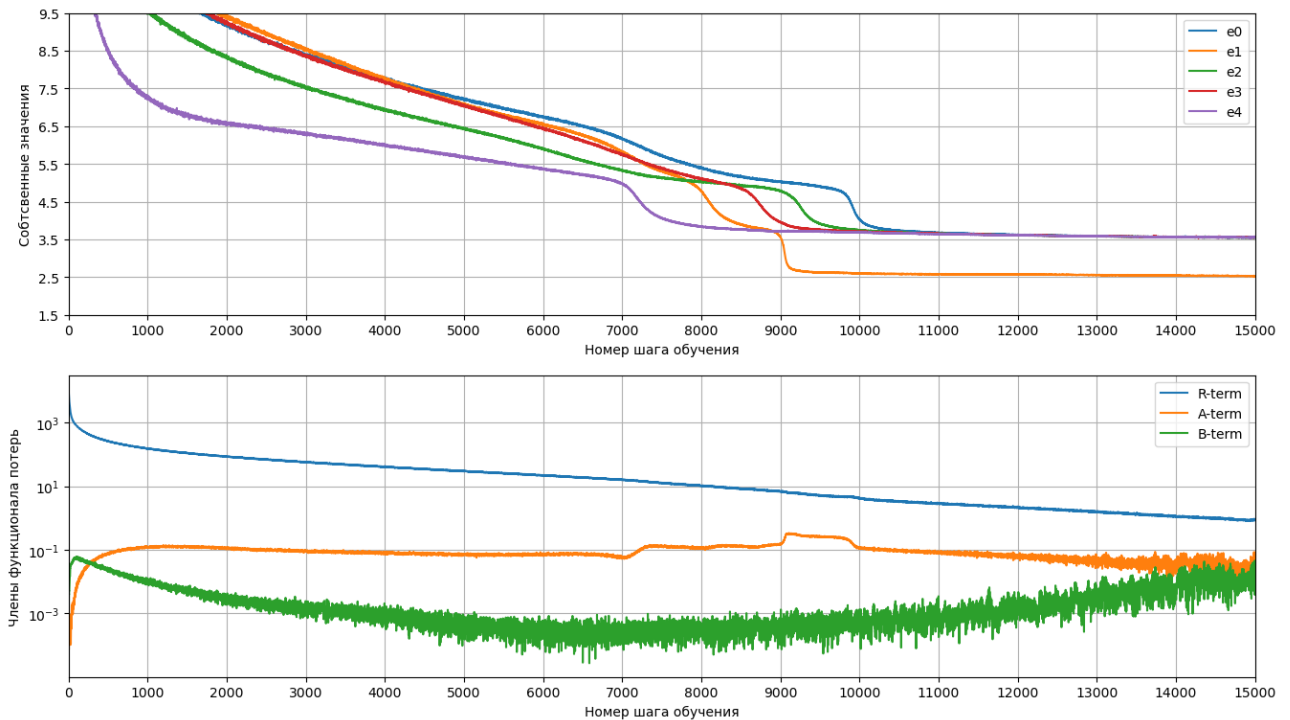


Рис. 7: (Верхняя панель): Зависимость оценочных значений собственных значений задачи (1.1) в 5-мерном случае от номера итерации обучения при семплировании `custom`; (нижняя панель): зависимость значений членов функционала потерь от номера итерации обучения при семплировании `custom`.

m	E_m^{approx}	$\Delta E_m / E_m$
0	2.5235	0.0094
1	3.5429	0.0122
2	3.5495	0.0141
3	3.5527	0.0150
4	3.5646	0.0184

Таблица 10: Оценочные значения собственных значений на последней итерации обучения и их относительные ошибки, полученные в результате обучения пробной функции с гиперпараметрами, указанными в Таблице 9, в течение $STEPS = 15000$ итераций.

5.3.2 СЕМПЛИРОВАНИЕ ТОЧЕК В КООРДИНАТНОМ ПРОСТРАНСТВЕ С РАСПРЕДЕЛЕНИЕМ `uniform`

Пробная функция обучалась в течение $STEPS = 15000$ итераций, что заняло 2 часа 57 минут и 14 секунд. Зависимость оценочных значений собственных значений задачи от номера итерации обучения приведена на верхней панели Рис. 8. Видно, что за 15000 итераций обучения пробная функция приближается к решению задачи. Оценочные значения собственных значений на последней итерации обучения, а также их относительные ошибки приведены в Таблице 11. Видно, что основное состояние обеспечило ошибку в 0.3%, следующие 2 состояния — ошибку в 2%, в то время как 3-е и 4-е состояния обеспечивают относительную ошибку примерно в 3%.

На нижней панели Рис. 6 приведена зависимость значений членов функционала потерь от номера итерации обучения, что позволяет заключить, что член, отвечающий за нормировку (A-term) почти не изменяется на протяжении обучения, а остальные члены вели себя так же, как и в случае семплирования `custom` (за исключением того, что B-term с 1000-ой итерации по примерно 2000-ую итерацию возрастает).

m	E_m^{approx}	$\Delta E_m / E_m$
0	2.5082	0.0032
1	3.5716	0.0204
2	3.5766	0.0218
3	3.6021	0.0291
4	3.6108	0.0316

Таблица 11: Оценочные значения собственных значений на последней итерации обучения и их относительные ошибки, полученные в результате обучения пробной функции с гиперпараметрами, указанными в Таблице 9, в течение $STEPS = 15000$ итераций при семплировании `uniform`.

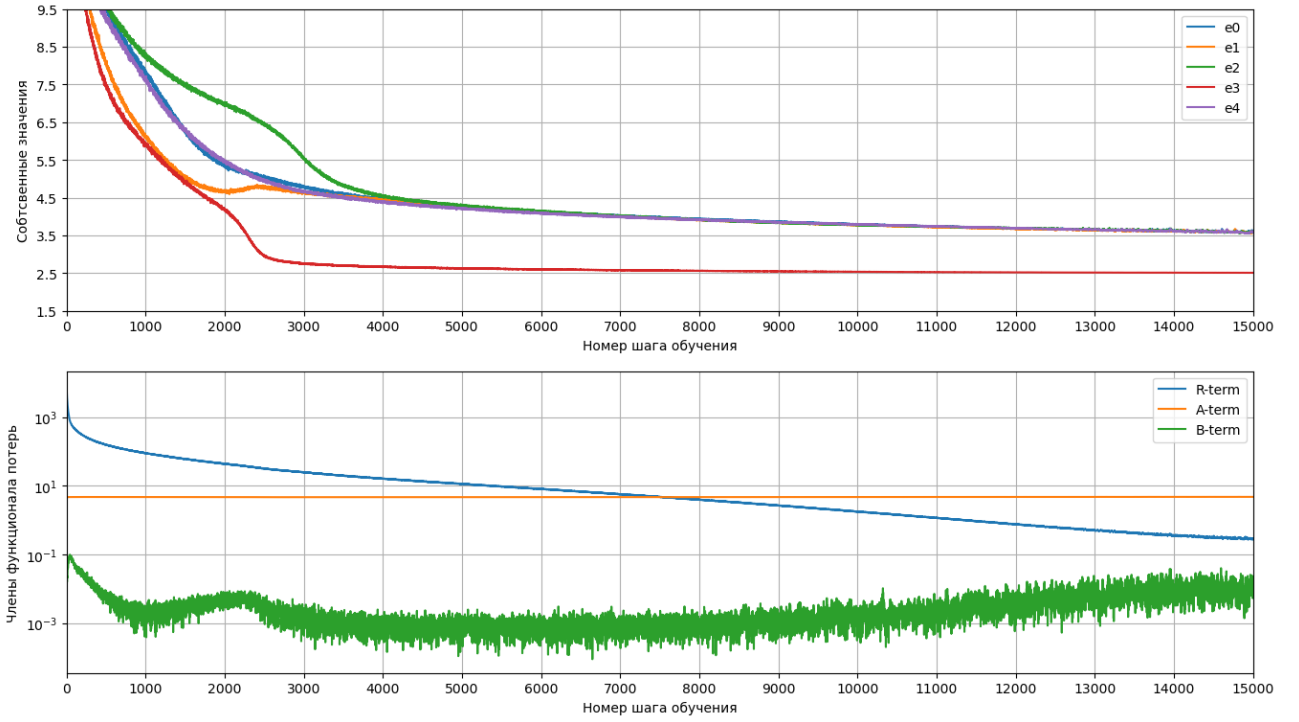


Рис. 8: (Верхняя панель): Зависимость оценочных значений собственных значений задачи (1.1) в 5-мерном случае от номера итерации обучения при семплировании **uniform**; (нижняя панель): зависимость значений членов функционала потерь от номера итерации обучения при семплировании **uniform**.

ЗАКЛЮЧЕНИЕ

В данной работе были развиты идеи работы [5]. В частности была немного изменена структура пробной функции и добавлена возможность выбора режима генерации точек в координатном пространстве. Было рассмотрено три характерных режима: режим **uniform** — генерация точек в координатном пространстве с равномерным распределением внутри некоторого гиперкуба, — режим **normal** — генерация точек в координатном пространстве согласно нормальному распределению — и режим **custom** — генерация точек в координатном пространстве согласно усредненной по оцениваемым состояниям плотности вероятности пробных функций. Были проведены тесты в размерностях 3, 4 и 5 (пробная функция обучалась каждый раз в течение 15000 итераций обучения). Было обнаружено, что при размерности 3 задачу (в данной работе рассматривалась лишь задача о квантовом гармоническом осцилляторе) удаётся решить при использовании каждого из рассматриваемых методов генерации точек в координатном пространстве; в случае 4-мерной задачи использование метод **normal** не позволило решить задачу, в то время как использование остальных методов генерации точек в координатном пространстве позволило решить задачу за заданное число итераций обучения; в 5-мерном случае рассматривались только режимы **uniform** и **custom**, их использование позволило успешно решить задачу.

Таким образом, только режимы генерации **uniform** и **custom** успешно позволили решить задачи во всех рассматриваемых размерностях. При этом, важно отметить, что отличительной чертой использования режима генерации **uniform** является то, что проб-

ные функции не нормируются требуемым образом — за время обучения с данным режимом генерации нигде не удалось пронаблюдать уменьшения члена функционала потерь, отвечающего за нормализацию пробных функций на 1. В то время как использование режима генерации `custom` позволяет успешно минимизировать член функционала потерь, который отвечает за нормализацию пробных функций на 1. Отсюда следует заключить, что использование метода генерации точек в координатном пространстве с распределением, плотность вероятности которого пропорциональна усредненной по рассматриваемым состояниям плотности вероятности, является наиболее оптимальным. Такой метод семплирования позволяет обучаться модель преимущественно в той области координатного пространства, где выше плотность вероятности и волновые функции сильнее всего отличны от нуля (то есть в тех областях, которые наиболее важны для успешной аппроксимации решения). Такой метод семплирования особенно важен в пространствах высокой размерности, так как с ростом размерности координатного пространства объём области, где волновые функции значительно отличны от нуля, экспоненциально падает, что значительно осложняет использование методов, завязанных на равномерном распределении точек в координатном пространстве.

В будущем предполагается исследование больших размерностей, однако эти изыскания требуют и больших вычислительных мощностей (требуется большой объём глобальной памяти на графическом ускорителе). В настоящей работе рассматривались примеры с лишь 5 состояниями, в дальнейшем было бы интересно решать задачи с большим числом состояний. Кроме того, было бы интересно отработать методику на различных спектральных задачах имеющих сложноустроенный энергетический спектр.

СПИСОК ЛИТЕРАТУРЫ

- [1] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [2] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017.
- [3] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural network methods in quantum mechanics. *Computer Physics Communications*, 104(1):1–14, 1997.
- [4] Hong Li, Qilong Zhai, and Jeff Z. Y. Chen. Neural-network-based multistate solver for a static schrödinger equation. *Phys. Rev. A*, 103:032405, Mar 2021.
- [5] Цыренов Эрдэм Цыденжапович. Выпускная квалификационная работа. Численное решение стационарного уравнения Шредингера с использованием искусственных нейронных сетей. *СПбГУ*, 2022.
- [6] Лифшиц Е.М. Ландау Л.Д. *Том 3. Квантовая механика. Нерелятивистская теория.* Теоретическая физика в десяти томах. М.: Наука. Гл. ред. физ-мат. лит., 4-е изд., испр., 1986.
- [7] Eric W. Weisstein. Hermite polynomial. From MathWorld—A Wolfram Web Resource. <https://mathworld.wolfram.com/HermitePolynomial.html>.
- [8] Bernd A. Berg. Introduction to markov chain monte carlo simulations and their statistical analysis, 2004.
- [9] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.
- [10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

ПРИЛОЖЕНИЕ

Листинг 2: Исходный код класса пробной функции и процедуры обучения пробной функции, написанный на языке Python с использованием фреймворка PyTorch.

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # # Imports and device detection
5  # Date time
6  import datetime as dt
7
8  # System
9  import sys
10
11 # Time
12 import time
13
14 # NumPy
15 import numpy as np
16
17 # Pandas
18 import pandas as pd
19
20 # PyTorch framework
21 import torch
22 from torch import nn
23
24 # Device
25 if torch.cuda.is_available():
26     device = torch.device("cuda")
27     print(f"Using {device}: {torch.cuda.get_device_name()}")
28 else:
29     device = torch.device("cpu")
30     print("No GPU found, using cpu")
31
32
33 # # Parameters
34 # Dictionary of model parameters
35 parameters = {}
36
37
38 def fillParameters(
39     AF: str, # Activation function
40     AMPLITUDE: str, # Amplitude function
41     Nh: int, # Number of nodes in each hidden layer
42     D: int, # Dimension of the coordinate space
43     M: int, # Number of states we want to find
44     std: float, # Standard deviation of sample distribution
45     B: int, # Batch size
46     learningRate: float, # Starting value of the learning rate (we use ADAM
47     # optimizer)
48     weightDecay: float, # Weight decay (we use ADAM optimizer)
49     wR: float, # Residual term weight
50     wA: float, # Normalisation term weight
51     wB: float, # Orthogonalisation term weight
52     wE: float, # Energy term weight
53     epsilon: float, # Metropolis algorithm parameter
54 ):
55     """Fills dictionary of model parameters"""
```

```

56     global parameters
57     parameters["AF"] = AF
58     parameters["AMPLITUDE"] = AMPLITUDE
59     parameters["Nh"] = Nh
60     parameters["D"] = D
61     parameters["M"] = M
62     parameters["std"] = std
63     parameters["B"] = B
64     parameters["learningRate"] = learningRate
65     parameters["weightDecay"] = weightDecay
66     parameters["wR"] = wR
67     parameters["wA"] = wA
68     parameters["wB"] = wB
69     parameters["wE"] = wE
70     s = "staticWeights"
71     for key in parameters.keys():
72         if key != "configuration":
73             s = s + f"{key}{{parameters[key]}}"
74     parameters["configuration"] = s # Model configuration
75
76
77 # # Activation functions
78
79
80 class Sin(nn.Module):
81     """Custom sin activation function class"""
82
83     def forward(self, x: torch.Tensor) -> torch.Tensor:
84         return torch.sin(x)
85
86     def diff(self, x: torch.Tensor) -> torch.Tensor:
87         return torch.cos(x)
88
89     def diff2(self, x: torch.Tensor) -> torch.Tensor:
90         return -torch.sin(x)
91
92
93 class Tanh(nn.Module):
94     """Custom tanh activation function class"""
95
96     def forward(self, x: torch.Tensor) -> torch.Tensor:
97         return torch.tanh(x)
98
99     def diff(self, x: torch.Tensor) -> torch.Tensor:
100         return 1 / torch.cosh(x) ** 2
101
102     def diff2(self, x: torch.Tensor) -> torch.Tensor:
103         return -2 * torch.sinh(x) / torch.cosh(x) ** 3
104
105
106 # # Amplitude functions
107
108
109 class Gaussian(nn.Module):
110     """Custom Gaussian amplitude function class"""
111
112     def forward(self, x: torch.Tensor) -> torch.Tensor:
113         global parameters
114         global device
115         assert x.shape[0] > 0, "Wrong shape 0 of the tensor x"
116         assert x.shape[1] == parameters["D"], "Wrong shape 1 of the tensor x"
117         result = torch.ones((x.shape[0], parameters["M"])).to(device)

```

```

118     result = torch.mul(result.t(), torch.exp(-torch.sum(x**2, axis=1) / 2)).t()
119     assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
120     assert result.shape[1] == parameters["M"], "Wrong_shape_1_of_the_result"
121     return result
122
123 def gradient(self, x: torch.Tensor) -> torch.Tensor:
124     global parameters
125     global device
126     assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
127     assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
128     result = torch.ones((x.shape[0], parameters["D"], parameters["M"])).to(device)
129     for d in range(parameters["D"]):
130         # first term
131         a = torch.ones((x.shape[0], parameters["M"])).to(device)
132         a = torch.mul(
133             a.t(),
134             torch.exp(-torch.sum(x**2, axis=1) / 2) * (-2 * x[:, d] / 2),
135         ).t()
136         # keeping result
137         result[:, d, :] = a
138     assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
139     assert result.shape[1] == parameters["D"], "Wrong_shape_1_of_the_result"
140     assert result.shape[2] == parameters["M"], "Wrong_shape_2_of_the_result"
141     return result
142
143 def laplacian(self, x: torch.Tensor) -> torch.Tensor:
144     global parameters
145     global device
146     assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
147     assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
148     result = torch.ones((x.shape[0], parameters["M"])).to(device)
149     result = torch.mul(
150         result.t(),
151         torch.exp(-torch.sum(x**2, axis=1) / 2)
152         * (
153             torch.sum((-2 * x / 2) ** 2, axis=1)
154             - 2 * torch.sum(torch.ones_like(x) / 2, axis=1)
155         ),
156     ).t()
157     assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
158     assert result.shape[1] == parameters["M"], "Wrong_shape_1_of_the_result"
159     return result
160
161
162 class Exponent4(nn.Module):
163     """Custom Exponent amplitude function class"""
164
165     def forward(self, x: torch.Tensor) -> torch.Tensor:
166         global parameters
167         global device
168         assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
169         assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
170         result = torch.ones((x.shape[0], parameters["M"])).to(device)
171         result = torch.mul(
172             result.t(),
173             torch.exp(-torch.sum(x**4, axis=1) / parameters["std"] ** 4),
174         ).t()
175         assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
176         assert result.shape[1] == parameters["M"], "Wrong_shape_1_of_the_result"
177         return result
178
179     def gradient(self, x: torch.Tensor) -> torch.Tensor:

```

```

180     global parameters
181     global device
182     assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
183     assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
184     result = torch.ones((x.shape[0], parameters["D"], parameters["M"])).to(device)
185     for d in range(parameters["D"]):
186         # first term
187         a = torch.ones((x.shape[0], parameters["M"])).to(device)
188         a = torch.mul(
189             a.t(),
190             torch.exp(-torch.sum(x**4, axis=1) / parameters["std"] ** 4)
191             * (-4 * x[:, d] ** 3 / parameters["std"] ** 4),
192         ).t()
193         # keeping result
194         result[:, d, :] = a
195     assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
196     assert result.shape[1] == parameters["D"], "Wrong_shape_1_of_the_result"
197     assert result.shape[2] == parameters["M"], "Wrong_shape_2_of_the_result"
198     return result
199
200 def laplacian(self, x: torch.Tensor) -> torch.Tensor:
201     global parameters
202     global device
203     assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
204     assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
205     result = torch.ones((x.shape[0], parameters["M"])).to(device)
206     result = torch.mul(
207         result.t(),
208         torch.exp(-torch.sum(x**4, axis=1) / parameters["std"] ** 4)
209         * (
210             torch.sum((-4 * x**3 / parameters["std"] ** 4) ** 2, axis=1)
211             - 12 * torch.sum(x**2 / parameters["std"] ** 4, axis=1)
212         ),
213     ).t()
214     assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
215     assert result.shape[1] == parameters["M"], "Wrong_shape_1_of_the_result"
216     return result
217
218
219 # # Neural network (NN)
220
221
222 class NN(nn.Module):
223     """Class for a neural network"""
224
225     def __init__(self):
226         super().__init__()
227         global parameters
228         if parameters["AF"] == "sin":
229             self.AF = Sin()
230         elif parameters["AF"] == "tanh":
231             self.AF = Tanh()
232         else:
233             print("Wrong_AF")
234             sys.exit(1)
235
236     # NUMBER OF HIDDEN LAYERS IS ASSUMED TO BE 3
237     self.stack = nn.Sequential(
238         nn.Linear(parameters["D"], parameters["Nh"]),
239         self.AF,
240         nn.Linear(parameters["Nh"], parameters["Nh"]),
241         self.AF,
242         nn.Linear(parameters["Nh"], parameters["Nh"]),

```

```

242         self.AF,
243         nn.Linear(parameters["Nh"], parameters["M"]),
244         self.AF,
245     )
246     for i in range(len(self.stack)):
247         if "weight" in dir(self.stack[i]):
248             torch.nn.init.normal_(self.stack[i].weight, 0, np.sqrt(0.1))
249     global device
250     # chooseDevice()
251     self.stack.to(device)
252
253     def forward(self, x: torch.Tensor) -> torch.Tensor:
254         global parameters
255         global device
256         assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
257         assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
258         result = self.stack(x).to(device)
259         assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
260         assert result.shape[1] == parameters["M"], "Wrong_shape_1_of_the_result"
261         return result
262
263     def gradient(self, x: torch.Tensor) -> torch.Tensor:
264         global parameters
265         global device
266         assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
267         assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
268         result = torch.ones((x.shape[0], parameters["D"], parameters["M"])).to(device)
269         for d in range(parameters["D"]):
270             ydiff = self.AF.diff(self.stack[0](x)) * self.stack[0].weight[:, d]
271             y = self.stack[0 * 2 + 1](self.stack[0 * 2](x))
272             # NUMBER OF HIDDEN LAYERS IS ASSUMED TO BE 3
273             for i in range(1, 3 + 1):
274                 ydiff = self.AF.diff(self.stack[i * 2](y)) * torch.matmul(
275                     ydiff, self.stack[i * 2].weight.t()
276                 )
277                 y = self.stack[i * 2 + 1](self.stack[i * 2](y))
278             # keeping result
279             result[:, d, :] = ydiff
280         assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
281         assert result.shape[1] == parameters["D"], "Wrong_shape_1_of_the_result"
282         assert result.shape[2] == parameters["M"], "Wrong_shape_2_of_the_result"
283         return result
284
285     def laplacian(self, x: torch.Tensor) -> torch.Tensor:
286         global parameters
287         global device
288         assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
289         assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
290         preresult = torch.ones((x.shape[0], parameters["D"], parameters["M"])).to(
291             device
292         )
293         for d in range(parameters["D"]):
294             ydiff = self.AF.diff(self.stack[0](x)) * self.stack[0].weight[:, d]
295             ydiff2 = self.AF.diff2(self.stack[0](x)) * self.stack[0].weight[:, d] ** 2
296             y = self.stack[0 * 2 + 1](self.stack[0 * 2](x))
297             # NUMBER OF HIDDEN LAYERS IS ASSUMED TO BE 3
298             for i in range(1, 3 + 1):
299                 # second derivative
300                 ydiff2 = self.AF.diff2(self.stack[i * 2](y)) * torch.matmul(
301                     ydiff, self.stack[i * 2].weight.t()
302                 ) ** 2 + self.AF.diff(self.stack[i * 2](y)) * torch.matmul(
303                     ydiff2, self.stack[i * 2].weight.t()

```

```

304         )
305         # first derivative
306         ydiff = self.AF.diff(self.stack[i * 2](y)) * torch.matmul(
307             ydiff, self.stack[i * 2].weight.t()
308         )
309         # output of the current layer
310         y = self.stack[i * 2 + 1](self.stack[i * 2](y))
311         # Keeping preresult
312         preresult[:, d, :] = ydiff2
313         result = torch.sum(preresult, axis=1)
314         assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
315         assert result.shape[1] == parameters["M"], "Wrong_shape_1_of_the_result"
316         return result
317
318
319 # # Trial function
320
321
322 class TrialFunction(nn.Module):
323     """Trial function class"""
324
325     def __init__(self):
326         super().__init__()
327         self.nn = NN()
328         global parameters
329         if parameters["AMPLITUDE"] == "gaussian":
330             self.amplitude = Gaussian()
331         elif parameters["AMPLITUDE"] == "exponent4":
332             self.amplitude = Exponent4()
333
334     def forward(self, x: torch.Tensor) -> torch.Tensor:
335         global parameters
336         global device
337         assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
338         assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
339         result = self.nn(x).to(device) * self.amplitude(x).to(device)
340         assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
341         assert result.shape[1] == parameters["M"], "Wrong_shape_1_of_the_result"
342         return result
343
344     def gradient(self, x: torch.Tensor) -> torch.Tensor:
345         global parameters
346         global device
347         assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
348         assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"
349         result = torch.ones((x.shape[0], parameters["D"], parameters["M"])).to(device)
350         a = self.amplitude(x)
351         ag = self.amplitude.gradient(x)
352         nn = self.nn(x)
353         nng = self.nn.gradient(x)
354         for d in range(parameters["D"]):
355             result[:, d, :] = nng[:, d, :] * a + nn * ag[:, d, :]
356         assert result.shape[0] == x.shape[0], "Wrong_shape_0_of_the_result"
357         assert result.shape[1] == parameters["D"], "Wrong_shape_1_of_the_result"
358         assert result.shape[2] == parameters["M"], "Wrong_shape_2_of_the_result"
359         return result
360
361     def laplacian(self, x: torch.Tensor) -> torch.Tensor:
362         global parameters
363         global device
364         assert x.shape[0] > 0, "Wrong_shape_0_of_the_tensor_x"
365         assert x.shape[1] == parameters["D"], "Wrong_shape_1_of_the_tensor_x"

```



```

366         result = (
367             self.nn.laplacian(x) * self.amplitude(x)
368             + self.nn(x) * self.amplitude.laplacian(x)
369             + 2 * torch.sum(self.nn.gradient(x) * self.amplitude.gradient(x), axis=1)
370         )
371         assert result.shape[0] == x.shape[0], "Wrong shape 0 of the result"
372         assert result.shape[1] == parameters["M"], "Wrong shape 1 of the result"
373         return result
374
375     def weight(self, x: torch.Tensor) -> torch.Tensor:
376         """Weight function"""
377         global parameters
378         global device
379         assert x.shape[0] > 0, "Wrong shape 0 of the tensor x"
380         assert x.shape[1] == parameters["D"], "Wrong shape 1 of the tensor x"
381         result = torch.mean(self.forward(x) ** 2, axis=1)
382         # result = (
383         #     1
384         #     / (parameters["std"] * float(np.sqrt(2 * np.pi))) ** parameters["D"]
385         #     * torch.exp(-0.5 * torch.sum(x * x, axis=1) / parameters["std"] ** 2)
386         # )
387         assert result.shape[0] == x.shape[0], "Wrong shape 0 of the result"
388         return result
389
390     def l2Norm(self, x: torch.Tensor) -> torch.Tensor:
391         """Computes L2-norm of the trial function"""
392         global parameters
393         global device
394         assert x.shape[0] > 0, "Wrong shape 0 of the tensor x"
395         assert x.shape[1] == parameters["D"], "Wrong shape 1 of the tensor x"
396         result = torch.zeros(parameters["M"]).to(device)
397         f = self.forward(x)
398         w = self.weight(x)
399         for m in range(parameters["M"]):
400             result[m] = torch.mean(abs(f[:, m]) ** 2 / w)
401         assert result.shape[0] == parameters["M"], "Wrong shape 0 of the result"
402         return result
403
404     def spectrum(self, x: torch.Tensor) -> torch.Tensor:
405         """Finds current spectrum"""
406         global parameters
407         global device
408         assert x.shape[0] > 0, "Wrong shape 0 of the tensor x"
409         assert x.shape[1] == parameters["D"], "Wrong shape 1 of the tensor x"
410         result = torch.zeros(parameters["M"]).to(device)
411         f = self.forward(x)
412         lap = self.laplacian(x)
413         w = self.weight(x)
414         v = 0.5 * torch.sum(x * x, axis=1)
415         for m in range(parameters["M"]):
416             result[m] = torch.mean(f[:, m] * (-0.5 * lap[:, m] + v * f[:, m]) / w)
417         result = result / self.l2Norm(x)
418         assert result.shape[0] == parameters["M"], "Wrong shape 0 of the result"
419         return result
420
421     def save(self):
422         """Saves model weights and biases"""
423         torch.save(
424             self.state_dict(),
425             "../models/" + parameters["configuration"] + ".pt",
426         )
427

```

```

428     def load(self):
429         """Loads model weights and biases"""
430         self.load_state_dict("../models/" + parameters["configuration"] + ".pt")
431
432
433     ## Metropolis algorithm
434
435
436     def MetropolisAlghorithmStep(x, f):
437         """One step of the Metropolis algorithm, it return the updated sample x"""
438         global parameters
439         x_prime = x + parameters["epsilon"] * (2 * torch.rand_like(x).to(device) - 1)
440         alpha = f(x_prime).to(device) / f(x).to(device)
441         doesXMove = torch.rand((x.shape[0])).to(device) <= alpha
442         x_prime = (
443             torch.mul(doesXMove.int(), x_prime.t()).t()
444             + torch.mul((1 - doesXMove.int()), x.t()).t()
445         )
446         return x_prime
447
448
449     ## Training
450
451
452     def training(
453         trialFunction: TrialFunction, # Trial function we will train
454         steps: int, # Number of steps
455     ):
456         """Trains trial function"""
457         global parameters
458         global device
459         print("Model configuration:")
460         print(parameters["configuration"])
461         print("")
462         print("Training has been started...")
463         print("")
464         print(
465             "Step, Time [s], Residual-term, Normalization-term,"
466             + " Orthogonalization-term, Sorted energies"
467         )
468         time0 = time.time()
469         time_history = np.full((steps), np.nan)
470         loss_history = np.full((steps), np.nan)
471         r_history = np.full((steps), np.nan)
472         a_history = np.full((steps), np.nan)
473         b_history = np.full((steps), np.nan)
474         spectrum_history = np.full((parameters["M"], steps), np.nan)
475         optimizer = torch.optim.AdamW(
476             params=list(trialFunction.parameters()),
477             lr=parameters["learningRate"],
478             weight_decay=parameters["weightDecay"],
479         )
480         lossFunction = nn.MSELoss()
481         for step in range(steps):
482             # Sample
483             if step == 0:
484                 x = parameters["std"] * (
485                     torch.rand((parameters["B"], parameters["D"]))) - 0.5
486                 ).to(device)
487             else:
488                 for i in range(10):
489                     x = MetropolisAlghorithmStep(x, trialFunction.weight)

```

```

490     # Spectrum (Energies)
491     e = trialFunction.spectrum(x)
492     # Forward (trial function)
493     f = trialFunction(x)
494     # Laplacian
495     lap = trialFunction.laplacian(x)
496     # Weight function
497     w = trialFunction.weight(x)
498     # Squared norm
499     sqrNorm = trialFunction.l2Norm(x)
500     # R-term
501     r = sum(
502         [
503             torch.mean(
504                 (
505                     -0.5 * lap[:, m]
506                     + 0.5 * f[:, m] * torch.sum(x**2, axis=1)
507                     - f[:, m] * e[m]
508                 )
509                 ** 2
510                 / w
511             )
512             / sqrNorm[m]
513             for m in range(parameters["M"])
514         ]
515     )
516     # A-term
517     a = torch.sum((sqrNorm - torch.tensor(1).to(device)) ** 2)
518     # B-term
519     b = torch.zeros(parameters["M"], parameters["M"]).to(device)
520     for m1 in range(1, parameters["M"]):
521         for m2 in range(0, m1):
522             maxs = (torch.max(f[:, m1]) * torch.max(f[:, m2])).detach() + 0.001
523             torch.Tensor.detach(maxs)
524             b[m1, m2] = (
525                 torch.square(torch.mean(f[:, m1] * f[:, m2] / w))
526                 / sqrNorm[m1]
527                 / sqrNorm[m2]
528             )
529     # The total loss
530     loss = lossFunction(
531         parameters["wR"] * r
532         + parameters["wA"] * a
533         + parameters["wB"] * torch.sum(b)
534         + parameters["wE"] * torch.sum(e),
535         torch.tensor(0.0).to(device),
536     )
537     # Backpropagation
538     optimizer.zero_grad()
539     loss.backward()
540     optimizer.step()
541     # Save a state of the model
542     time_ = time.time() - time0
543     time_history[step] = time_
544     if np.isnan(loss.item()):
545         print("LOSS IS NAN")
546         return
547     loss_history[step] = loss.item()
548     r_history[step] = r.cpu().detach().numpy()
549     a_history[step] = a.cpu().detach().numpy()
550     b_history[step] = torch.sum(b).cpu().detach().numpy()
551     spectrum_history[:, step] = e.cpu().detach().numpy()

```

```

552         # Release some memory
553         if torch.cuda.is_available():
554             torch.cuda.empty_cache()
555         # Print the state of the model
556         if step % 100 == 0:
557             print(
558                 f"{step}, {time_:.2f}, {r_history[step]:.4e}, "
559                 + f"{a_history[step]:.4e}, {b_history[step]:.4e}, "
560                 + f"{np.sort(spectrum_history[:, step])}"
561             )
562         # Save history
563         d = {
564             "Time[s]": time_history,
565             "R-term": r_history,
566             "A-term": a_history,
567             "B-term": b_history,
568         }
569         for i in range(parameters["M"]):
570             d[f"e{i}"] = spectrum_history[i, :]
571         history_df = pd.DataFrame(data=d)
572         now = dt.datetime.now()
573         dt_string = now.strftime("%d.%m.%Y_%H:%M:%S")
574         history_df.to_csv("../data/" + dt_string + parameters["configuration"] + ".csv")
575         return history_df

```