МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра вычислительной математики  
и прикладных информационных технологий

ЛАБОРАТОРНАЯ РАБОТА №1

ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНОГО

УРАВНЕНИЯ ШРЁДИНГЕРА:

МЕТОД ПРИСТРЕЛКИ

Направление 01.04.02 Прикладная математика и информатика

Профиль Математическое моделирование и вычислительная математика

|  |  |  |  |
| --- | --- | --- | --- |
| Зав. кафедрой | \_\_\_\_\_\_\_\_\_\_ | д. ф.-м.н., пр. | А.И. Шашкин \_\_.\_\_.2023 |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_ |  | И.Б. Рахимов |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_ | д.ф.-м.н., пр. | Ю.К. Тимошенко |

Воронеж 2023

Содержание

Цели и задачи 3

1. Одномерное стационарное уравнение Шрёдингера. Математический формализм. Общие свойства решений 5

1.1. Нижняя оценка энергетического спектра 6

1.2. Свойства потенциальной энергии 7

1.3. Осцилляционная теорема 8

2. Метод пристрелки 9

2.1. Квантовомеханических средние 11

3. Программная реализация 13

4. Результаты численных экспериментов и их анализ 16

Список использованных источников 19

Приложения 20

Приложение 1 Компьютерный код 20

# Цели и задачи

### Цели работы

Целями лабораторной работы являются практическое освоение информации, полученной при изучении курса «Компьютерное моделирование в математической физике» по теме «Численное решение стационарного уравнения Шрёдингера», а также развития алгоритмического мышления и приобретения опыта использования знаний и навыков по математике, численным методам и программированию для решения прикладных задач физико–технического характера.

### Задачи работы

**Формулировка проблемы:** электрон находится в потенциальном поле (рис. 1):

где , , .

1. Используя метод пристрелки, найти:

* энергии;
* нормированные волновые функции;
* плотности вероятности для основного и первого возбужденного состояний.

Привести как численные значения энергий, так и построить графики волновых функций и плотностей вероятности.

2) Вычислить для этих состояний квантовомеханические средние и .

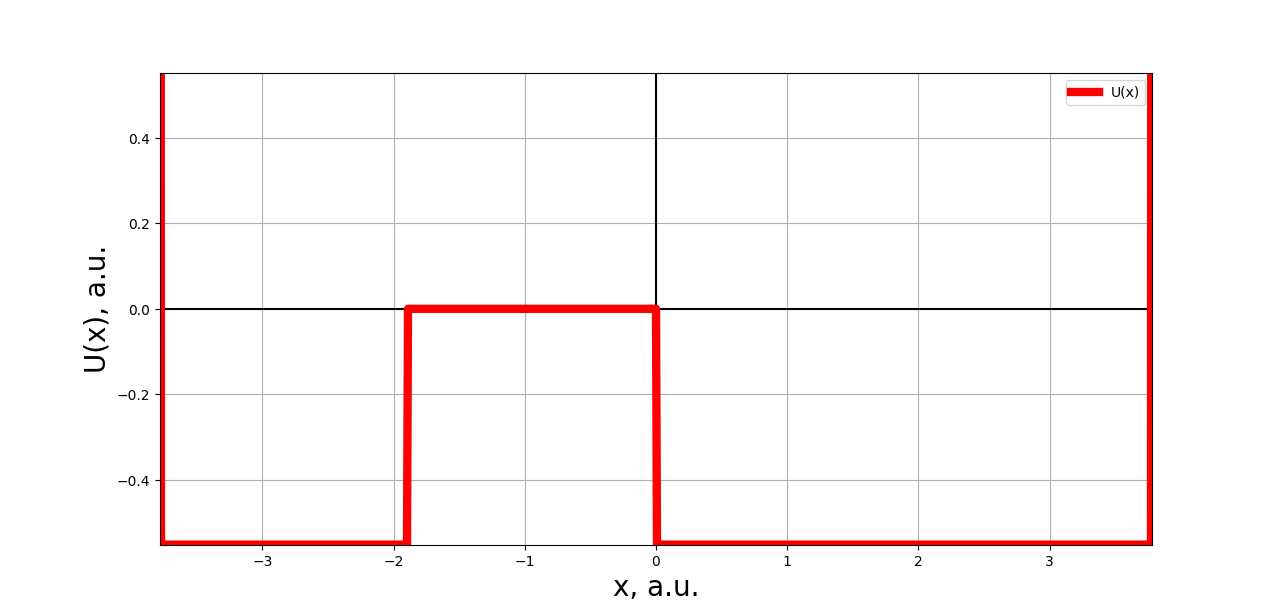


Рис. 1. Потенциальная функция

# Одномерное стационарное уравнение Шрёдингера. Математический формализм. Общие свойства решений

Одномерное стационарное уравнение Шрёдингера:

с математической точки зрения представляет собой задачу определения собственных значений и собственных функций оператора Гамильтона . Для частицы с массой , находящийся в потенциальном поле , оператор Гамильтона имеет вид:

где – оператор кинетической энергии:

а - постоянная Планка. Собственной значение оператора Гамильтона имеет смысл энергии соответствующей изолированной квантовой системы. Собственные функции называются волновыми функциями. Они полностью определяют квантовые состояния системы. Волновая функция однозначна и непрерывна во всём пространстве. Непрерывность волновой функции и её первой производной сохраняется и при обращении в в некоторой области пространства. В такую область частица вообще не может проникнуть, то есть в этой области , а также на её границе, что следует из непрерывности волновой функции.

Подставляя и в получим:

Для упрощения можно использовать атомные единицы Хартри [1], в этой системе приняты в качестве исходных единиц следующие значения: – абсолютная величина заряда электрона; – постоянная планка, – масса электрона. То есть в атомных единицах Хартри , и . В такой системе примет вид:

В следующем разделе будет найдете нижняя оценка энергетического спектра, которая сыграет важную роль в определении собственных значений и собственных функций оператора Гамильтона .

## Нижняя оценка энергетического спектра

Оценим нижнюю границу энергетического спектра. Пусть минимальное значение функции равно . Тогда и [1]. Поэтому из уравнения ) следует:

То есть энергии всех состояний больше .

В следующем разделе в дополнении к нижней оценке энергетического спектра будут изложены свойства потенциального поля , которые будут необходимы для численного решения уравнения Шрёдингера.

## Свойства потенциальной энергии

Особый практический интерес представляет случай, когда:

Для данной свойства решений уравнения Шредингера зависят от знака собственного значения . Рассмотрим два случая:

1. *.* Частица совершает инфинитное движение. Оператор Гамильтона имеет непрерывный спектр собственных значений. Квантовые состояния непрерывного спектра называют несвязными состояниями. Частица, находящаяся в несвязном состоянии, способна уйти на бесконечность.
2. *.* Частица с отрицательной энергией совершает финитное движений. Оператор Гамильтона имеет дискретный спектр, то есть собственные значения и соответствующие собственные функции можно снабдить номерами, называемыми квантовыми числами. При уравнение приобретает вид:

Квантовое состояние, обладающее наименьшей энергией, называется основным, остальные состояния дискретного спектра называются возбуждёнными состояниями. Частица, находящаяся в связанном состоянии, не способна уйти на бесконечность. То есть плотность вероятности при , но на всех конечных расстояниях не равно нулю. В силу линейности стационарного уравнения Шрёдингера, волновые функции математически определены с точностью до постоянного множителя. Однако, из физических соображений, волновые функции должны быть нормированы следующим образом:

Можно показать, что собственные значения дискретного спектра в одномерном случае невырождены, то есть все собственные значения имеют уникальные значения.

В дальнейшем мы будет заниматься численным моделированием квантовых состояний только дискретного спектра. При этом необходимо пользоваться осцилляционной теоремой, которая будет представлена в следующем разделе.

## Осцилляционная теорема

Упорядоченные собственные значения оператора Гамильтона в порядке возрастания, нумеруя энергию основного состояния индексом «0»: . Тогда волновая функция -го состояния будет иметь узлов (пересечений с осью ). Исключение: области, в которых потенциальная функция бесконечна.

В следующей главе будет описан алгоритм, который позволит с помощью полученной теоретической информации получить численное решение уравнения Шрёдингера.

# Метод пристрелки

Численное моделирование будет производится только на основном и низковозбуждённых состояниях электрона для потенциальной функции с бесконечными стенками (рис. 1). В данном отчёте для основного и первого возбуждённого состояния ().

Так как за пределами обращается в граничные условия для волновой функции будут иметь вид:

Так как для собственных значений известна оценка снизу . То удобно начинать с вычисления энергии и волновой функции основного состояния. Оценим грубо энергию основного состояния как:

где малая положительная величина.

Подставим значение этой энергии в уравнение . Это уравнение становится обыкновенным дифференциальным уравнением второго порядка с граничными условиями .

Предположим, что оценка точно совпадает с собственным значением, тогда численно решенная задача Коши для полученного дифференциального уравнения интегрированием “вперёд” (начальный узел сетки на левой границе) и интегрированием “назад” (начальный узел на правой границе), должны получится практически одинаковые результаты.

На практике оценка собственного значения в нулевом приближение отличается от точного результата, поэтому волновые функции, полученные интегрированием “вперёд” и “назад”, так же отличаются. Используя таким образом вычисленные волновые функции, можно сформулировать критерий позволяющий уточнить собственное значение с заданной точностью. Рассмотрим алгоритм, реализующий эту идею.

Зададим на интервале сетку из узлов с постоянным шагом :

Граничные условия приобретают вид:

Задачу Коши для дифференциального уравнения можно решить методом Нумерова для интегрирования “вперёд”:

и “назад’:

здесь . Для использования формулы необходимо знать и , а для формулы – и . Значения и известны, а и нет. Однако если достаточно велико, то можно считать, что , , где , – малые величины в силу непрерывности волновой функции. Так же в силу Осцилляционной теоремы для нечетных состояний и должны быть разных знаков, для чётных одинаковых.

Для оценки близости к собственному значению вычисляется разность производных волновых функций, полученных интегрированием “вперёд” и “назад” в некотором узле сетки :

Где *,*  – волновые функции, полученные интегрированием “вперёд” и “назад” соответственно, – узел сшивки. Так же перед вычислением необходимо масштабировать волновые функции итак, чтобы , такое масштабирование называется математической нормировкой.

Для расчёта производных можно воспользоваться формулой численного дифференцирования, например:

Вычисление позволяет организовать процедуру поиска собственного значения. Выбирается нулевое приближение к энергии основного приближения энергии следующим образом:

где – малая положительная величина и вычислим по формуле (. Далее будем увеличивать энергии с шагом до тех пор, пока величины на двух соседних шагах и не будут иметь различные знаки. Если шаг меньше, чем разность энергий соседних уровней, можно быть уверенным, что искомое собственное значение находится в отрезке . Далее для уточнения собственного значения можно воспользоваться любым методом решения уравнений, например методом бисекций.

В следующем разделе будут перечислены численные методы для приближения интеграла и второй производной, использующийся для вычисления квантовомеханических средних.

## Квантовомеханических средние

Дополнительно в данной работе необходимо вычислить квантовомеханические средние и для этого понадобиться выбрать метод численного интегрирования, в данной работе использовался метод трапеций для промежутка интегрирования на разделённого на одинаковые отрезки:

где , , .

Для этой формулы справедлива оценка погрешности :

где .

Для квантовомеханического среднего кинетической энергии дополнительно необходимо вычислить вторую производную, для этого использовались следующие формулы второго порядка точности:

В следующей главе будет подробно описана программная реализация изложенного выше алгоритма численного решения одномерного стационарного уравнения Шрёдингера для электрона в потенциальном поле .

# Программная реализация

В Приложении 1 представлена программа численного решения одномерного стационарного уравнения Шрёдингера для электрона в потенциальном поле . Использовались атомные единицы Хартри. Программа реализована на языке Python 3.10.7 в графической среде разработки «PyCharm Community Edition 2023.2.5», использовался интерпретатор CPython, операционная система Windows 11 Профессиональная.

В строках 253-257 и 43-54 задаётся начальное условие задачи, полуширина отрезка и потенциальная функция соответственно, в строках 259-264 все величины, используемые в задаче, переводятся в атомные единицы Хартри. Далее в 470й строке вызывается основная функция разработанной программы – main, её определение находится в строках 266-468.

Начало функции 267-288 состоит из определения переменных задачи, таких как:

* min\_energy – минимальная энергия;
* forward\_first\_approximation – первая аппроксимация для интегрирования вперёд;
* backward\_penultimate\_approximation – предпоследняя аппроксимация для интегрирования назад;
* begin – начало отрезка интегрирования;
* end – конец отрезка интегрирования;
* num\_intervals – количество интервалов, используемых для решения задачи;
* num\_points – количество точек, используемых для решения задачи;
* connection – узел сшивки;
* step – расстояние между ближайшими точками сетки;
* x – сетка, на которую разбивается отрезок ;
* energy\_step – шаг, с которым ищется изменение знака энергии;
* max\_energy\_value – максимальное возможное значение энергии;
* max\_energy\_count – максимальное количество собственных значений, которые необходимо найти.

Далее объявленные данные используются в вызове функции eigen\_value\_intervals 290-299, которая объявлена в строках 170-216, она принимает на вход, следующий данные:

* min\_energy – минимальное значение энергии
* energy\_step – шаг, с которым ищется изменение знака энергии;
* max\_energy\_value – максимальное возможное значение энергии;
* max\_energy\_count – максимальное количество собственных значений, которые необходимо найти;
* x – сетка, на которую разбивается отрезок ;
* step – расстояние между ближайшими точками сетки;
* forward\_first\_approximation – первая аппроксимация для интегрирования вперёд;
* backward\_penultimate\_approximation – предпоследняя аппроксимация для интегрирования назад;
* connection – узел сшивки;

Используя эти данные, функция находит и возвращает отрезки, в которых, должны находится собственные значения оператора Гамильтона. Для этого используются следующие вспомогательные функции:

* is\_close\_energy – вычисляет насколько близко находятся функции для интегрирования “вперёд” и “назад” для данного значения энергии 150-166;
* compute\_q – фукнция для вычисления 60-62;
* forward\_integration, backward\_integration – функции для интегрирования “вперёд” и “назад” соответственно, для заданного значения энергии 76-120;
* normalization – функция для достижения равенства в узле сшиви 126-137;
* is\_close – функция для вычисления “близости” функция полученных с помощью интегрирования “вперёд” и “назад” 140-145;
* derivative – функция для аппроксимации первой производной 64-73.

Далее на полученных интервалах осуществляется поиск собственных значений 301-325, с помощью функции bisection\_method, которая осуществляет поиск корней методом бисекции и объявлена в строках 5-37.

Далее для полученных значений энергии в строках 327-413 строятся графики волновой функции после интегрирования “вперёд”, “назад”, а также плотность вероятности. Для построения графиков используется Python библиотека matplotlib, полученные графики можно посмотреть на рисунках 2 и 3.

Так же в данной работе необходимо было вычислить квантовомеханические средние и , для всех средних необходимо численно вычислить интеграл, для этого использовался метод трапеций, который подробно описан в разделе 2.2, его реализация находится в 218-224. Дополнительно для квантовомеханического среднего необходимо вычислить приближение второй производной, функция, реализующая это находится в строках 233-249.

# Результаты численных экспериментов и их анализ

В данной работе необходимо было найти энергии, нормированные волновые функции и плотности вероятности для основного и первого возбужденного состояний.

С помощью численных методов описанных в главе 2 и реализация которых подробно прокомментирована в главе 3, получены следующие результаты:

1. для основного состояния:
2. для первого возбужденного состояния:

где – энергия состояния, и – квантовомеханические средние кинетической и потенциальной энергии соответственно, – погрешность вычисления квантовомеханическсих средних. Нижний индекс означает какая волновая функция использовалась для вычисления квантовомеханическсих средних и их погрешностей – волновая функция была получена интегрирование “вперёд”, – “назад”.

Видно, что квантовомеханическсие средние вычислены с шестым порядком точности, дополнительно можно проверить результаты вычислений зная что и . В данной задаче

Вычисленные волновые функции и плотности вероятности изображены на рисунках 1 и 2, для основного и первого возбужденного состояния соответственно. На данных графиках изображены следующие данные:

* красный график – потенциальная энергия;
* оранжевый график – волновая функция, полученная интегрированием “вперёд”;
* голубой график – волновая функция, полученная интегрированием “назад”;
* чёрный график – плотность вероятности вычисленная с использованием волновая функция, полученная интегрированием “вперёд”;
* зелёный график – плотность вероятности вычисленная с использованием волновая функция, полученная интегрированием “назад”.

Так же дополнительно выведено значение энергии, погрешность его вычисления, а так же подробная легенда с единицами измерения, если они есть.

Опираясь на осцилляционную теорему, можно сказать, что графики волновых функций соответствуют найденным собственным значениям, т.к. при основном состоянии волновая функция не пересекает ось х. При первом возбуждённом состоянии, волновая функция один раз проходит через ось х.

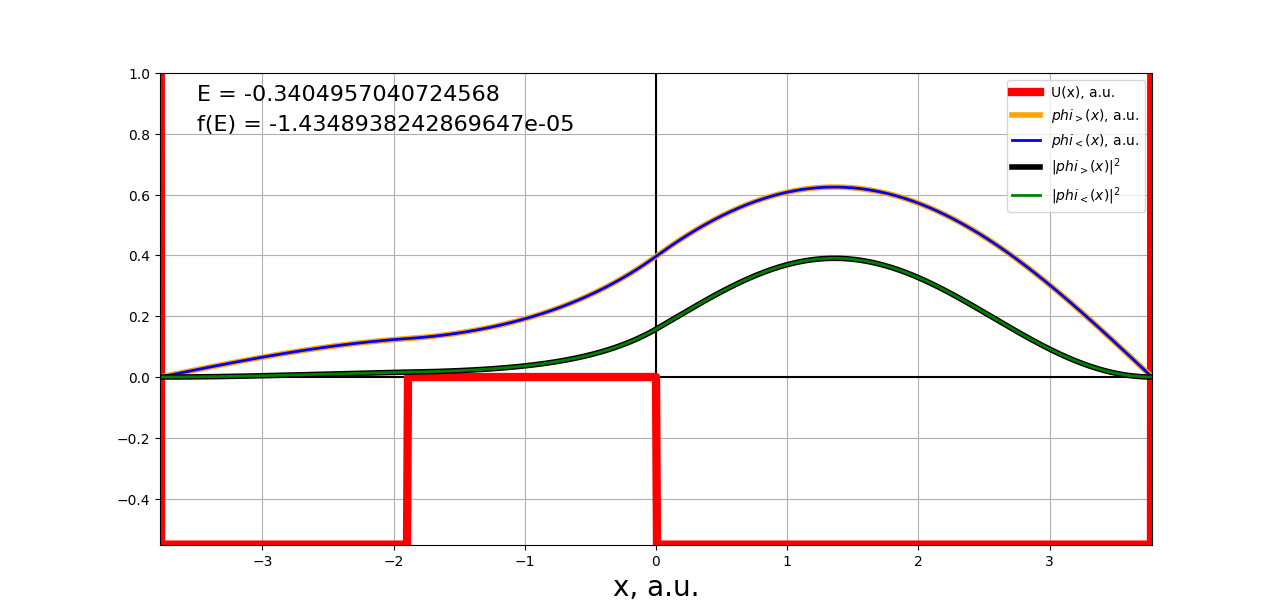


Рис. 2. Основное состояние

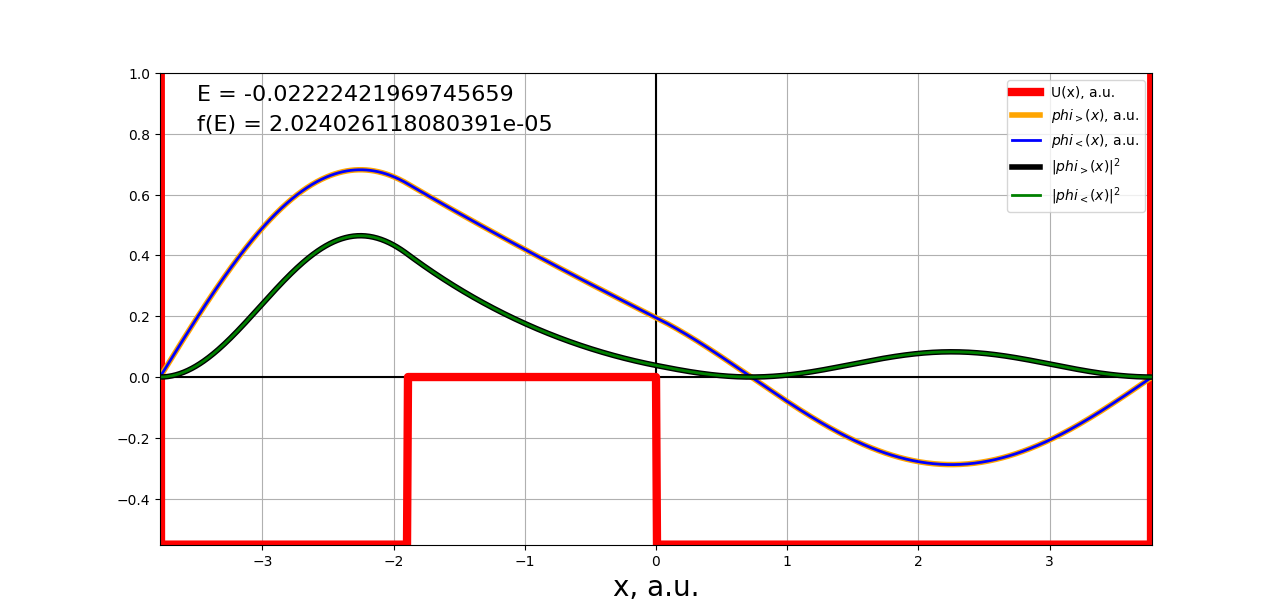


Рис. 3. Первое возбуждённое состояние

# Список использованных источников

1. Тимошенко Ю. К. Численное решение стационарного уравнения Шрёдингера: метод пристрелки. Учебное пособие. Воронеж: Научная книга, 2019. 35 с.
2. Давыдов А. С. Квантовая механика. СПб.: БХВ-Петербург, 2011. 704 с.
3. Бизли Д. Python. Подробный справочник. СПб.: Символ-Плюс, 2010. 864 с.
4. Марчук А. Х. Введение в Python для студентов-астрономов. Методическое пособие. СПб.: СПбГУ, 2016. 49 с.
5. Доля П. Г. Введение в научный Python. Харьков: ХНУ, 2016. 265 с.

# Приложения

## Приложение 1 Компьютерный код

import matplotlib.pyplot as plt

from math import sqrt

######################################################################

### bisection\_method

######################################################################

# метод деления отрезка пополам

def bisection\_method(f, # функция корень которой необходимо найти

begin:float, end:float, # начало и конец отрезка

# в котором находится корень

steps:int, # максимальное количество

# шагов алгоритма

eps:float): # точность приближения корня

l = begin

r = end

step = 0

while step < steps:

step += 1

center = 0.5 \* (r + l)

f\_l = f(l)

f\_c = f(center)

f\_r = f(r)

if abs(f\_c) < eps:

break

if f\_l \* f\_c < 0:

r = center

else: # f\_c \* f\_r < 0

l = center

return 0.5 \* (r + l)

######################################################################

### quantum\_mechanics

######################################################################

def v(x:float) -> float:

if (x > -half\_width and x < -half\_width / 2.0) or

(x > 0 and x < half\_width):

return -1

elif x >= -half\_width / 2.0 and x <= 0:

return 0

else:

return w

# потенциальная функция

def U(x:float) -> float:

return v0 \* v(x)

def q(e:float,

x:float) -> float:

return 2.0 \* (e - U(x))

def compute\_q(x:list[float],

energy:float) -> list[float]:

return [q(energy, xi) for xi in x]

def derivative(function:list[float],

index:int,

eps:float) -> float:

der1 = function[index - 2] - function[index + 2]

der2 = function[index + 1] - function[index - 1]

return (

(der1 + 8.0 \* der2)

/ #-------------------

(12.0 \* eps)

)

# интегрирование вперёд

def forward\_integration(num\_intervals:int,

penultimate\_approximation:float,

q:list[float],

step:float) -> list[float]:

num\_points = num\_intervals + 1

forward = [float] \* (num\_points)

forward[0] = 0.0

forward[1] = penultimate\_approximation

c = step \*\* 2 / 12.0

for i in range(1, num\_intervals):

p1 = 2.0 \* (1.0 - 5.0 \* c \* q[i]) \* forward[i]

p2 = (1.0 + c \* q[i - 1]) \* forward[i - 1]

p3 = (1.0 + c \* q[i + 1])

forward[i + 1] = (

(p1 - p2)

/ #-----------

p3

)

return forward

# интегрирование назад

def backward\_integration(num\_intervals:int,

first\_approximation:float,

q:list[float],

step:float) -> list[float]:

num\_points = num\_intervals + 1

backward = [float] \* num\_points

backward[num\_intervals] = 0

backward[num\_intervals - 1] = first\_approximation

c = step \*\* 2 / 12.0

for i in range(num\_intervals - 1, 0, -1):

f1 = 2.0 \* (1.0 - 5.0 \* c \* q[i]) \* backward[i]

f2 = (1.0 + c \* q[i + 1]) \* backward[i + 1]

backward[i - 1] = (

(f1 - f2)

/ #--------------------

(1.0 + c \* q[i - 1])

)

return backward

# фукнция для нормировки forward

# и достижения равенства на узле сшивки

def normalization(forward:list[float],

backward:list[float],

connection:int):

# нормировка

norm = abs(max(forward, key = abs))

forward = list(map(lambda x: x / norm, forward))

# равенство на узле сшивки - connection

coef = forward[connection] / backward[connection]

backward = list(map(lambda x: coef \* x, backward))

return forward, backward

# функция возвращающая разницу производных на узле сшивки

def is\_close(forward:list[float],

backward:list[float],

connection:int,

eps:float) -> float:

return (derivative(forward, connection, eps)

- derivative(backward, connection, eps))

# функция вычисляет значения волновой функции

# вперёд и назад по известным данным

# и возвращающая разницу производных на узле сшивки

def is\_close\_energy(

energy:float, # значение энергии

x:list[float], # сетка

step:float, # шаг сетки

forward\_first\_approximation:float,

# первое приближение для интегрирования вперёд

backward\_penultimate\_approximation:float,

# n - 1 приближение для интегрирования назад

connection:int) -> float: # узел сшивки

num\_intervals = len(x) - 1

q = compute\_q(x, energy)

forward = forward\_integration(num\_intervals,

forward\_first\_approximation, q, step)

backward = backward\_integration(num\_intervals,

backward\_penultimate\_approximation, q, step)

forward, backward = normalization(forward, backward, connection)

return is\_close(forward, backward, connection, step)

# возвращет интервалы в которых

# находятся собственные значения оператора Гамильтона

def eigen\_value\_intervals(min\_energy:float,

energy\_step:float,

max\_energy\_value:float,

max\_energy\_count:int,

x:list[float],

step:float,

forward\_first\_approximation:float,

# 1 приближение для интегрирования вперёд

backward\_penultimate\_approximation:float,

# n - 1 приближения для интегрирования назад

connection:int) -> list[float]:

level = 0

sign = (1 if level % 2 == 0 else -1)

energy = min\_energy

prev\_close = is\_close\_energy(energy,

x,

step,

sign \* forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

intervals = []

intervals.append(energy)

while(energy < max\_energy\_value):

energy = energy + energy\_step

sign = (1 if level % 2 == 0 else -1)

close = is\_close\_energy(energy,

x,

step,

sign \* forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

if close \* prev\_close < 0:

prev\_close = close

intervals.append(energy)

intervals.append(energy)

level += 1

if(len(intervals) >= 2 \* max\_energy\_count):

break

else:

intervals[-1] = energy

intervals.pop()

return intervals

def integrate(f:list[float], step:float) -> float:

size = len(f)

sum = 0.0

for i in range(1, size - 1):

sum += f[i]

return step \* ((f[0] + f[size - 1]) / 2.0 + sum)

def quantum\_normalize(psi:list[float], step:float) -> list[float]:

density = list(map(lambda x: x \* x, psi))

c = integrate(density, step)

root = 1.0 / sqrt(c)

normalized = list(map(lambda x: root \* x, psi))

return normalized

def second\_derivation(f:list[float], step:float):

num\_points = len(f)

derivation = [float] \* num\_points

coef = 1 / (step \* step)

derivation[0] = coef \* (2 \* f[0] - 5 \* f[1] + 4 \* f[2] - f[3])

derivation[num\_points - 1] = (coef \*

(2 \* f[num\_points - 1]

- 5 \* f[num\_points - 2]

+ 4 \* f[num\_points - 3]

- f[num\_points - 4]))

for i in range(1, num\_points - 1):

derivation[i] = coef \* (f[i - 1] - 2 \* f[i] + f[i + 1])

return derivation

w = 10.0

v0 = 15 # Электронвольт

half\_width = 2.0 # Ангстрем

print(f"v0 = {v0} Электронвольт")

print(f"l = {half\_width} Ангстрем")

# Перевод величин в атомную систему единиц

v0 = v0 / 27.211 # атомных единиц

half\_width = half\_width / 0.5292 # бор

print(f"v0 = {v0} атомных единиц")

print(f"l = {half\_width} бор")

def main():

min\_energy = -v0

print("Минимальное значение потенциальной функции = {}".format(min\_energy))

print()

forward\_first\_approximation = 1.e-9

backward\_penultimate\_approximation = 1.e-9

begin = -half\_width

end = +half\_width

num\_intervals = 1000

num\_points = num\_intervals + 1

connection = 300

step = (end - begin) / num\_intervals

x = [begin + step \* i for i in range(num\_points)]

energy\_step = 0.01

max\_energy\_value = 1000

max\_energy\_count = 2

intervals = eigen\_value\_intervals(min\_energy,

energy\_step,

max\_energy\_value,

max\_energy\_count,

x,

step,

forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

print("intervals: ", intervals)

bisection\_method\_steps = 100

bisection\_method\_eps = 0.0001

energies = []

level = 0

f = lambda e : is\_close\_energy(e,

x,

step,

forward\_sign \* forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

for i in range(max\_energy\_count):

l = intervals[2 \* i]

r = intervals[2 \* i + 1]

forward\_sign = (1 if level % 2 == 0 else -1)

energy = bisection\_method(

f,

l, r,

bisection\_method\_steps,

bisection\_method\_eps)

energies.append(energy)

level += 1

print("energies: ", energies, " a.u.")

print()

# значения для построения графиков

u = [U(xi) for xi in x]

q\_n0 = compute\_q(x, energies[0])

forward\_n0 = forward\_integration(num\_intervals,

forward\_first\_approximation, q\_n0, step)

backward\_n0 = backward\_integration(num\_intervals,

backward\_penultimate\_approximation, q\_n0, step)

forward\_n0, backward\_n0 = normalization(forward\_n0,

backward\_n0, connection)

forward\_n0 = quantum\_normalize(forward\_n0, step)

backward\_n0 = quantum\_normalize(backward\_n0, step)

forward\_probability\_n0 = list(map(lambda x: x \*\* 2, forward\_n0))

backward\_probability\_n0 = list(map(lambda x: x \*\* 2, backward\_n0))

q\_n1 = compute\_q(x, energies[1])

forward\_n1 = forward\_integration(num\_intervals,

forward\_first\_approximation, q\_n1, step)

backward\_n1 = backward\_integration(num\_intervals,

backward\_penultimate\_approximation, q\_n1, step)

forward\_n1, backward\_n1 = normalization(forward\_n1,

backward\_n1, connection)

forward\_n1 = quantum\_normalize(forward\_n1, step)

backward\_n1 = quantum\_normalize(backward\_n1, step)

forward\_probability\_n1 = list(map(lambda x: x \*\* 2, forward\_n1))

backward\_probability\_n1 = list(map(lambda x: x \*\* 2, backward\_n1))

# Графики

# U(x)

plt.axis([begin, end, min\_energy, v0])

plt.grid(True)

plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.xlabel("x, B", fontsize = 20, color = "k")

plt.ylabel("U(x), a.u.", fontsize = 20, color = "k")

plt.plot(x, u, linewidth = 6, color = "red", label = "U(x)")

plt.legend()

plt.savefig("U(x).png")

plt.show()

# n0

plt.axis([begin, end, min\_energy, v0 if v0 > 1 else 1])

plt.grid(True)

plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.xlabel("x, B", fontsize = 20, color = "k")

plt.plot(x, u, linewidth = 6, color = "red" ,

label = "U(x), a.u.")

plt.plot(x, forward\_n0, linewidth = 4, color = "orange",

label = "$phi\_>(x)$, a.u.")

plt.plot(x, backward\_n0, linewidth = 2, color = "blue",

label = "$phi\_<(x)$, a.u.")

plt.plot(x, forward\_probability\_n0, linewidth = 4,

color = "black", label = "$|phi\_>(x)|^2$")

plt.plot(x, backward\_probability\_n0, linewidth = 2,

color = "green" , label = "$|phi\_<(x)|^2$")

plt.legend()

plt.text(-3.5, 0.91, "E = " + str(energies[0]),

fontsize = 16, color = 'black')

plt.text(-3.5, 0.81, "f(E) = " + str(f(energies[0])),

fontsize = 16, color = 'black')

plt.savefig("n0.png")

plt.show()

# n1

plt.axis([begin, end, min\_energy, v0 if v0 > 1 else 1])

plt.grid(True)

plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.xlabel("x, B", fontsize = 20, color = "k")

plt.plot(x, u, linewidth = 6, color = "red", label = "U(x), a.u.")

plt.plot(x, forward\_n1, linewidth = 4, color = "orange",

label = "$phi\_>(x)$, a.u.")

plt.plot(x, backward\_n1, linewidth = 2, color = "blue",

label = "$phi\_<(x)$, a.u.")

plt.plot(x, forward\_probability\_n1, linewidth = 4, color = "black"

label = "$|phi\_>(x)|^2$")

plt.plot(x, backward\_probability\_n1, linewidth = 2,

color = "green" , label = "$|phi\_<(x)|^2$")

plt.legend()

plt.text(-3.5, 0.91, "E = " + str(energies[1]), fontsize = 16,

color = 'black')

plt.text(-3.5, 0.81, "f(E) = " + str(f(energies[1])),

fontsize = 16, color = 'black')

plt.savefig("n1.png")

plt.show()

# квантово механические средние

forward\_U\_n0 = [forward\_n0[i] \* u[i] \* forward\_n0[i]

for i in range(num\_points)]

backward\_U\_n0 = [backward\_n0[i] \* u[i] \* backward\_n0[i]

for i in range(num\_points)]

U\_n0\_forward = integrate(forward\_U\_n0, step)

U\_n0\_backward = integrate(backward\_U\_n0, step)

print("<U n0> forward =", U\_n0\_forward)

print("<U n0> backward =", U\_n0\_backward)

forward\_T\_n0 = second\_derivation(forward\_n0, step)

backward\_T\_n0 = second\_derivation(backward\_n0, step)

forward\_T\_n0 = [-0.5 \* forward\_n0[i] \* forward\_T\_n0[i]

for i in range(num\_points)]

backward\_T\_n0 = [-0.5 \* backward\_n0[i] \* backward\_T\_n0[i]

for i in range(num\_points)]

T\_n0\_forward = integrate(forward\_T\_n0, step)

T\_n0\_backward = integrate(backward\_T\_n0, step)

print("<T n0> forward =", T\_n0\_forward)

print("<T n0> backward =", T\_n0\_backward)

print()

forward\_U\_n1 = [forward\_n1[i] \* u[i] \* forward\_n1[i]

for i in range(num\_points)]

backward\_U\_n1 = [backward\_n1[i] \* u[i] \* backward\_n1[i]

for i in range(num\_points)]

U\_n1\_forward = integrate(forward\_U\_n1, step)

U\_n1\_backward = integrate(backward\_U\_n1, step)

print("<U n1> forward =", U\_n1\_forward)

print("<U n1> backward =", U\_n1\_backward)

forward\_T\_n1 = second\_derivation(forward\_n1, step)

backward\_T\_n1 = second\_derivation(backward\_n1, step)

forward\_T\_n1 = [-0.5 \* forward\_n1[i] \* forward\_T\_n1[i]

for i in range(num\_points)]

backward\_T\_n1 = [-0.5 \* backward\_n1[i] \* backward\_T\_n1[i]

for i in range(num\_points)]

T\_n1\_forward = integrate(forward\_T\_n1, step)

T\_n1\_backward = integrate(backward\_T\_n1, step)

print("<T n1> forward =", T\_n1\_forward)

print("<T n1> backward =", T\_n1\_backward)

print()

print("E[0] - (<T> + <U>) forward =",

energies[0] - (T\_n0\_forward + U\_n0\_forward))

print("E[0] - (<T> + <U>) backward =",

energies[0] - (T\_n0\_backward + U\_n0\_backward))

print("E[1] - (<T> + <U>) forward =",

energies[1] - (T\_n1\_forward + U\_n1\_forward))

print("E[1] - (<T> + <U>) backward =",

energies[1] - (T\_n1\_backward + U\_n1\_backward))

main()