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 №2

ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНОГО

УРАВНЕНИЯ ШРЁДИНГЕРА:

ТЕОРИЯ ВОЗМУЩЕНИЙ

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. Теория возмущений 5
1.1. Частица в одномерной потенциальной яме 8
2. Программная реализация 9
3. Результаты численных экспериментов и их анализ 10
Список использованных источников 12
Приложения 13
Приложение 1 Компьютерный код 13

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 (невозмущенная система) $U^0(x) = V_0 \cdot v^0(x), x \in (-L, +L)$ (рис. 1):

$$v^0(x) = \begin{cases} -1, & x \in \left(-L, -\frac{L}{2}\right], x \in [0, L) \\ 0, & x \in \left(-\frac{L}{2}, 0\right) \\ \infty, & x \leq L, x \geq L \end{cases}, \quad (1)$$

где $V_0 = 15$ эВ, $L = 2\ \dot{A}$, $n = 1$.

С помощью теории возмущений найти:

- энергию основного состояния, с учётом поправок до второго порядка включительно;
- волновую функцию основного состояния, с учётом поправок первого порядка.

Возмущённую систему смоделировать самостоятельно, создав в потенциальной функции $U^0(x)$ из лабораторной работы №1 пик произвольной формы. Сравнить результаты с данными, полученными методом пристрелки.

Для потенциальной функции $U(x) = V_0 \cdot v(x)$ возмущённой системы выбран пик следующего вида (Рис. 2):

$$v(x) = \begin{cases} v^0(x) + 0.5, & x \in \left(-\frac{L}{2} + \frac{L}{5}, -\frac{L}{5}\right) \\ v^0(x), & x \in \left(-L, -\frac{L}{2} + \frac{L}{5}\right], x \in \left[-\frac{L}{5}, L\right) \end{cases}, \quad (2)$$

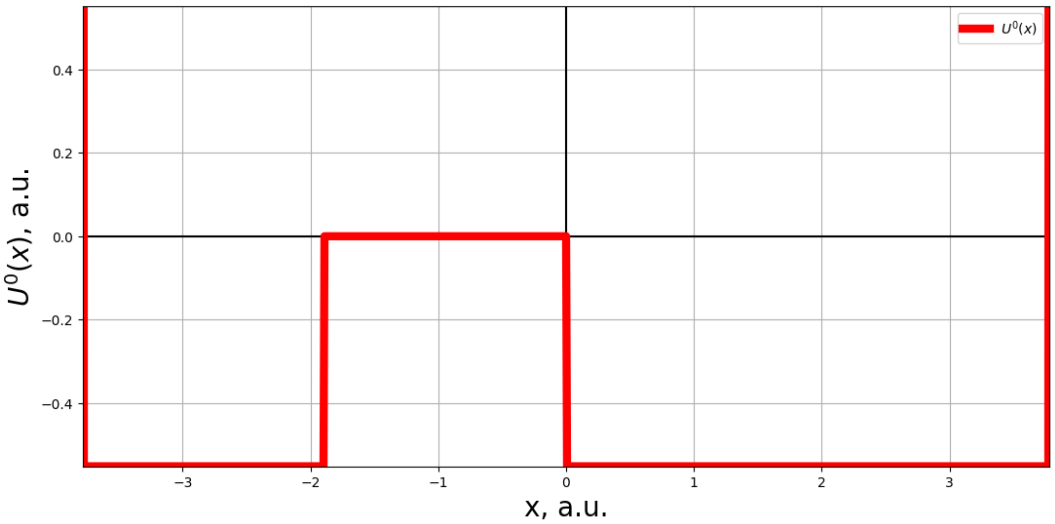

Рис. 1. График невозмущенной потенциальной функции

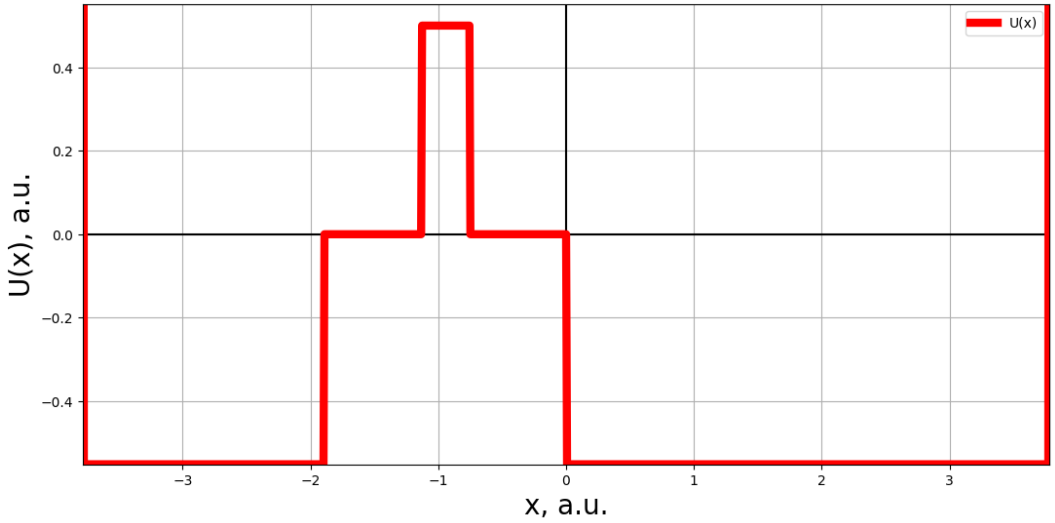

Рис. 2. График возмущенной потенциальной функции

1. Одномерное стационарное уравнение Шрёдингера. Теория возмущений

В данном отчете не будет подробно описана математическая часть одномерного стационарного уравнение Шрёдингера, так как это уже было сделано в отчете по первой лабораторной работе, вместо этого здесь будет подробно разобрано его решение в рамках теории возмущений.

К числу наиболее распространённых у физиков приближенных методов вычисления собственных значений и собственных функций оператора Гамильтона относится метод стационарных возмущений Релея-Шрёдингера, так же называемый методом или теорий возмущений. В рамках этого подхода предполагается что оператор Гамильтона, чьи собственные значения и собственные функции требуется определить, может быть представлен в виде:

$$\hat{H} = \hat{H}^0 + \hat{V}, \tag{3}$$

где $\hat{H}^0$ – гамильтониан идеализированной задачи, решение которой можно найти либо аналитически, либо относительно простым численным путём, $\hat{V}$ – называется оператором возмущения, или просто возмущением.

Оператором возмущения может быть либо часть гамильтониана, которая не учитывалась в идеализированной задаче, либо потенциальная энергия, связанная с наличием внешнего воздействия. Существенно, что оператор возмущения должен содержать «малый параметр», который будет обеспечивать сходимость рядов теории возмущений, о которых далее пойдёт речь.

Идеализированную систему, которую описывает гамильтониан $\hat{H}^0$, называют невозмущенной системой, а систему с гамильтонианом $\hat{H}$ – возмущенной системой. В рамках теории возмущений удаётся получить формулы, которые определяют энергии и волновые функции стационарных состояний через известные значения энергии $E_n^{(0)}$ и волновых функций $\psi_n^{(0)}$ невозмущенной системы.

Стационарные уравнения Шрёдингера для невозмущенной и возмущенной систем имеют вид:

$$\widehat{H}^0\psi_n^{(0)}(\xi) = E_n^{(0)}\psi_n^{(0)}(\xi), \tag{4}$$

$$\widehat{H}\psi_n^{(0)}(\xi) = E_n\psi_n^{(0)}(\xi). \tag{5}$$

Буквой ξ обозначена совокупность всех независимых координат. Задание ξ определяет положение точки в абстрактном пространстве, которое называют конфигурационным пространством.

Для системы, состоящей из одной частицы, конфигурационной пространство совпадает с обычным декартовым пространством $\xi = (x, y, z)$, $d\xi = dx\ dy\ dz$ [2].

Для частицы в одномерной пространстве ξ – это одна из декартовы координат, например x.

Волновые функции возмущенной и невозмущенной систем ортонормированы:

$$\int_{\Omega} \psi_n^*(\xi)\psi_m(\xi)d\xi = \delta_{nm},$$

$$\int_{\Omega} \psi_n^{(0)*}(\xi)\psi_m^{(0)}(\xi)d\xi = \delta_{nm},$$

где символ «$*$» означает комплексное сопряжение, а δ_{nm} – символ Кронекера, который определяется следующим образом:

$$\delta_{nm} = \begin{cases} 1, & n = m \\ 0, & n \neq m \end{cases}.$$

В теории возмущений решение уравнения (5) ищутся в виде рядов:

$$\begin{cases} E_n = E_n^{(0)} + E_n^{(1)} + E_n^{(2)} + \cdots = \sum_{k=0}^{\infty} E_n^{(k)} \\ \psi_n(\xi) = \psi_n^{(0)}(\xi) + \psi_n^{(1)}(\xi) + \psi_n^{(2)}(\xi) + \cdots = \sum_{k=0}^{\infty} \psi_n^{(k)}(\xi) \end{cases}, \tag{6}$$

где $E_n^{(k)}$, $\psi_n^{(k)}(\xi)$ – величины k-го порядка малости по возмущению $\hat{V}$, называемые k-ми поправками или поправками k-го порядка. Первые слагаемые рядов (6) определяются следующими формулами:

$$\begin{cases} E_n^{(1)} = V_{nn} \\ E_n^{(2)} = \sum_m{}' \dfrac{|V_{nm}|^2}{E_n^{(0)} - E_m^{(0)}} \\ \psi_n^{(1)}(\xi) = \sum_m{}' \dfrac{V_{nm}}{E_n^{(0)} - E_m^{(0)}} \psi_m^{(0)}(\xi) \end{cases} \tag{7}$$

где

$$V_{mn} = \langle n|V|m\rangle = \int_\Omega \psi_n^{(0)*}(\xi)\hat{V}\psi_m^{(0)}(\xi)d\xi \tag{8}$$

Штрих над знаком суммы означает пропуск слагаемого, имеющего равные индексы, то есть $m = n$.

Интеграл (8) называется матричным элементом оператора $\hat{V}$ по невозмущенным волновым функциям. Первая поправка к собственному значению $E_n^{(1)}$ равняется квантовомеханическому среднему значению возмущения в состоянии $\psi_n^{(0)}$, а поправка второго порядка к энергии основного состояния не может быть положительной.

Ряды (7) сходятся если выполняется неравенство:

$$|V_{mn}| << |E_n^{(0)} - E_m^{(0)}|$$

Во многих случаях для решения задачи достаточно ограничиться вычислением энергии с учетом поправок до второго порядка включительно и волновой функции с учетом поправок первого порядка.

$$\begin{cases} E_n = E_n^{(0)} + V_{nn} + \sum_m{}' \dfrac{|V_{nm}|^2}{E_n^{(0)} - E_m^{(0)}} \\ \psi_n(\xi) = \psi_n^{(0)}(\xi) + \sum_m{}' \dfrac{V_{nm}}{E_n^{(0)} - E_m^{(0)}} \psi_m^{(0)}(\xi) \end{cases} \tag{9}$$

1.1. Частица в одномерной потенциальной яме

Частицу в одномерной потенциальной яме с бесконечными стенками будем рассматривать как невозмущенную систему (рис. 1). Она соответствует потенциальной функции $U^0(x)$. График возмущённой потенциальной функции представлен соответствует $U(x)$ (рис. 2).

Возмущение представляет собой разность двух функций:

$$\hat{V}(x) = V(x) = U(x) - U^0(x)$$

Для вычисления энергии и волновой функции возмущенной системы используются собственные значения и собственные функции, найденные численно в первой лабораторной работе методом пристрелки.

В следующей главе будет подробно объяснена программная реализация на языке программирования Python решения одномерного стационарного уравнения Шрёдингера с помощью теории возмущений.

2. Программная реализация

В Приложении 1 представлена программа численного решения одномерного стационарного уравнения Шрёдингера. 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 1-440 находится реализация метода пристрелки, которая подробно была описана в отчёте по первой лабораторной работе. Дополнительно для решения одномерного стационарного уравнения Шрёдингера с помощью теории возмущений были добавлены следующие функции:

- возмущенная потенциальная функция в строках 58-64;
- оператор возмущение 66-68.

В строках 440-546 реализовано решение одномерного стационарного уравнения Шрёдингера в рамках теории возмущений. В строках 460-468 определяется функция для получения матричного элемента оператора возмущения (8). В данной работе вычисление интегралов, необходимых для получения матричных элементов (8), осуществляется с использованием метода трапеций для численного нахождения интегралов, он так же был подробно разобрал в отчёте по первой лабораторной работе.

Используя формулы (9) рассчитываются:

- первая поправка энергии в строках 472-474;
- вторая поправка энергии в 476-485;
- первая поправка собственной функции 487-491.

Результаты расчётов визуализируются с помощью Python библиотеки matplotlib в строках 504-540.

Результаты численных вычислений будут подробно проанализированы в следующей главе.

3. Результаты численных экспериментов и их анализ

В данной работе необходимо было найти энергию основного состояния, с учётом поправок до второго порядка включительно и волновую функцию основного состояния, с учётом поправок первого порядка.

На рисунке 3 представлены результаты численного моделирования для возмущенной системы с волновой функции в первом приближении и рассчитанные методом пристрелки, оранжевая линия соответствует методу пристрелки для поиска собственной функции основного состояния, а голубая – теории возмущений. Смотря на графики видно, что функции, найденные двумя способами достаточно близки.

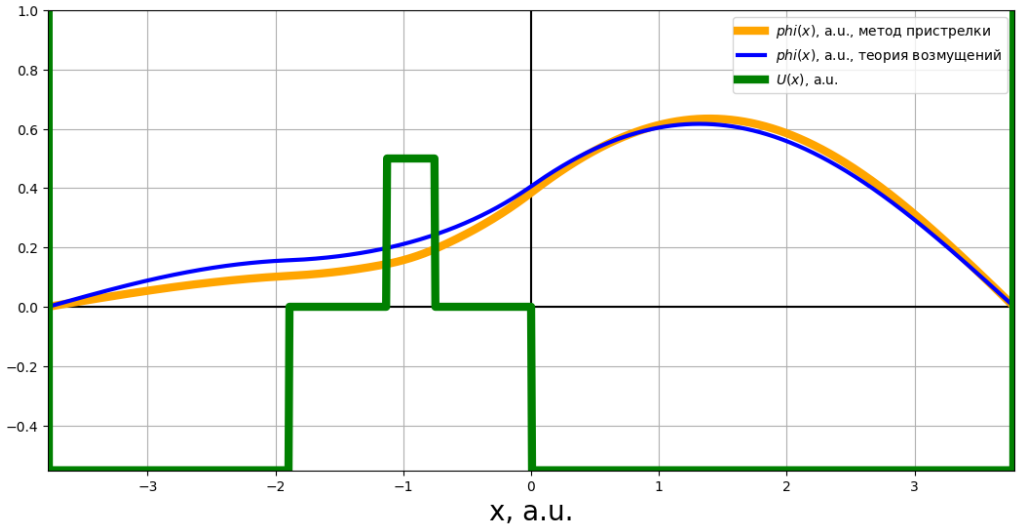

Рис. 3. Основное состояние

Вычисленная энергия основного состояния методом пристрелки:

$$E = -0.3498609384474568 \text{ а. е.}$$

и методом теории возмущений:

$$E = -0.3487921786216082 \text{ а. е.}$$

Видно, что значения совпадают до второго знака. Для вычисления энергии основного состояния с использованием теории возмущений были вычислены следующие поправки с использованием формул (9):

$$E_0^{(0)} = -0.3404957040724568$$

$$E_0^{(1)} = -0.007565257716425303$$

$$E_0^{(2)} = -0.0007312168327261536$$

Из первой главы известно, что поправка второго порядка к энергии основного состояния не может быть положительной, видно, что результаты вычислительного эксперимента соотносятся с теоретической информацией.

Список использованных источников

1. Тимошенко Ю. К. Численное решение стационарного уравнения Шрёдингера: метод пристрелки. Учебное пособие. Воронеж: Научная книга, 2019. 35 с.
2. Тимошенко Ю. К. Численное решение стационарного уравнения Шрёдингера: теория возмущений. Учебное пособие. Воронеж: Научная книга, 2019. 35 с.
3. Давыдов А. С. Квантовая механика. СПб.: БХВ-Петербург, 2011. 704 с.
4. Бизли Д. Python. Подробный справочник. СПб.: Символ-Плюс, 2010. 864 с.
5. Марчук А. Х. Введение в Python для студентов-астрономов. Методическое пособие. СПб.: СПбГУ, 2016. 49 с.
6. 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

#############################################################################
### shooting begin
#############################################################################

#############################################################################
### 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
```

```
47        (x > 0 and x < half_width):
48          return -1
49      elif x >= -half_width / 2.0 and x <= 0:
50          return 0
51      else:
52          return w
53
54  # Потенциальная функция невозмущенной системы
55  def U(x:float) -> float:
56      return v0 * v(x)
57
58  # Потенциальная функция возмущенной системы
59  def U1(x:float) -> float:
60      if (x > (-half_width / 2 + half_width / 5) and
61          x < (-half_width / 5)):
62          return U(x) + 0.5
63      else:
64          return U(x)
65
66  # возмущение
67  def V(x:float):
68      return U(x) - U1(x)
69
70  def U_pr(x:float):
71      return U(x) + V(x)
72
73  SOLVE_U = U
74
75  def q(e:float,
76        x:float) -> float:
77      return 2.0 * (e - SOLVE_U(x))
78
79  def compute_q(x:list[float],
80                energy:float) -> list[float]:
81      return [q(energy, xi) for xi in x]
82
83  def derivative(function:list[float],
84                 index:int,
85                 eps:float) -> float:
86      der1 = function[index - 2] - function[index + 2]
87      der2 = function[index + 1] - function[index - 1]
88      return (
89                 (der1 + 8.0 * der2)
90             / #-------------------
91                    (12.0 * eps)
92             )
93
94  # интегрирование вперёд
95  def forward_integration(num_intervals:int,
96                          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 из-
вестным 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 опера-
тора 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))
```

```
247       c = integrate(density, step)
248       root = 1.0 / sqrt(c)
249       normalized = list(map(lambda x: root * x, psi))
250       return normalized
251
252   def second_derivation(f:list[float], step:float):
253       num_points = len(f)
254       derivation = [float] * num_points
255
256       coef = 1 / (step * step)
257
258       derivation[0] = coef * (2 * f[0] - 5 * f[1] + 4 * f[2] - f[3])
259       derivation[num_points - 1] = (coef *
260                                     (2 * f[num_points - 1]
261                                    - 5 * f[num_points - 2]
262                                    + 4 * f[num_points - 3]
263                                        - f[num_points - 4]))
264
265       for i in range(1, num_points - 1):
266           derivation[i] = coef * (f[i - 1] - 2 * f[i] + f[i + 1])
267
268       return derivation
269
270   w = 10.0
271
272   v0 = 15 # Электронвольт
273   half_width = 2.0 # Ангстрем
274
275   print(f"v0 = {v0} Электронвольт")
276   print(f"l = {half_width} Ангстрем")
277
278   # Перевод величин в атомную систему единиц
279   v0 = v0 / 27.211 # атомных единиц
280   half_width = half_width / 0.5292 # бор
281
282   print(f"v0 = {v0} атомных единиц")
283   print(f"l = {half_width} бор")
284
285   def main():
286       global SOLVE_U
287
288       min_energy = -v0
289       print("Минимальное     значение     потенциальной     функции     =
290   {}".format(min_energy))
291       print()
292
293       forward_first_approximation = 1.e-9
294       backward_penultimate_approximation = 1.e-9
295
296       begin = -half_width
```

```
297       end = +half_width
298
299       num_intervals = 1000
300       num_points = num_intervals + 1
301
302       connection = 300
303
304       step = (end - begin) / num_intervals
305       x = [begin + step * i for i in range(num_points)]
306
307       energy_step = 0.01
308       max_energy_value = 1000
309       max_energy_count = 2
310
311       bisection_method_steps = 100
312       bisection_method_eps = 0.0001
313
314   ###########################################################################
315   ### shooting perturbation
316   ###########################################################################
317
318       SOLVE_U = U_pr
319
320       intervals_pr = eigen_value_intervals(min_energy,
321                                            energy_step,
322                                            max_energy_value,
323                                            max_energy_count,
324                                            x,
325                                            step,
326                                            forward_first_approximation,
327                                            back-
328   ward_penultimate_approximation,
329                                            connection)
330       print("intervals_pr: ", intervals_pr)
331
332       energies_pr = []
333       level_pr = 0
334       f_pr = lambda e : is_close_energy(e,
335         x,
336         step,
337         forward_sign * forward_first_approximation,
338         backward_penultimate_approximation,
339         connection)
340       for i in range(max_energy_count):
341           l = intervals_pr[2 * i]
342           r = intervals_pr[2 * i + 1]
343           forward_sign = (1 if level_pr % 2 == 0 else -1)
344           energy = bisection_method(
345               f_pr,
346               l, r,
```

```
347             bisection_method_steps,
348             bisection_method_eps)
349         energies_pr.append(energy)
350 
351         level_pr += 1
352     print("energies_pr: ", energies_pr, " a.u.")
353     print()
354 
355     q_n0_pr = compute_q(x, energies_pr[0])
356     forward_n0_pr = forward_integration(num_intervals,
357                     forward_first_approximation, q_n0_pr, step)
358     backward_n0_pr = backward_integration(num_intervals,
359                      backward_penultimate_approximation,
360                      q_n0_pr, step)
361     forward_n0_pr, backward_n0_pr = normalization(forward_n0_pr,
362                                       backward_n0_pr, connection)
363     forward_n0_pr = quantum_normalize(forward_n0_pr, step)
364     backward_n0_pr = quantum_normalize(backward_n0_pr, step)
365 
366     q_n1_pr = compute_q(x, energies_pr[1])
367     forward_n1_pr = forward_integration(num_intervals,
368                     forward_first_approximation, q_n1_pr, step)
369     backward_n1_pr = backward_integration(num_intervals,
370                      backward_penultimate_approximation,
371                      q_n1_pr, step)
372     forward_n1_pr, backward_n1_pr = normalization(forward_n1_pr,
373                                       backward_n1_pr, connection)
374     forward_n1_pr = quantum_normalize(forward_n1_pr, step)
375     backward_n1_pr = quantum_normalize(backward_n1_pr, step)
376 
377 ##########################################################################
378 ### shooting idealized
379 ##########################################################################
380 
381     SOLVE_U = U
382 
383     intervals = eigen_value_intervals(min_energy,
384       energy_step,
385       max_energy_value,
386       max_energy_count,
387       x,
388       step,
389       forward_first_approximation,
390       backward_penultimate_approximation,
391       connection)
392     print("intervals: ", intervals)
393 
394     energies = []
395     level = 0
396     f = lambda e : is_close_energy(e,
```

```
397         x,
398         step,
399         forward_sign * forward_first_approximation,
400         backward_penultimate_approximation,
401         connection)
402       for i in range(max_energy_count):
403           l = intervals[2 * i]
404           r = intervals[2 * i + 1]
405           forward_sign = (1 if level % 2 == 0 else -1)
406           energy = bisection_method(
407               f,
408               l, r,
409               bisection_method_steps,
410               bisection_method_eps)
411           energies.append(energy)
412
413           level += 1
414       print("energies: ", energies, " a.u.")
415       print()
416
417       q_n0 = compute_q(x, energies[0])
418       forward_n0 = forward_integration(num_intervals,
419                    forward_first_approximation, q_n0, step)
420       backward_n0 = backward_integration(num_intervals,
421                     backward_penultimate_approximation, q_n0, step)
422       forward_n0, backward_n0 = normalization(forward_n0,
423                                 backward_n0, connection)
424       forward_n0 = quantum_normalize(forward_n0, step)
425       backward_n0 = quantum_normalize(backward_n0, step)
426
427       q_n1 = compute_q(x, energies[1])
428       forward_n1 = forward_integration(num_intervals,
429                    forward_first_approximation, q_n1, step)
430       backward_n1 = backward_integration(num_intervals,
431                     backward_penultimate_approximation, q_n1, step)
432       forward_n1, backward_n1 = normalization(forward_n1,
433                                 backward_n1, connection)
434       forward_n1 = quantum_normalize(forward_n1, step)
435       backward_n1 = quantum_normalize(backward_n1, step)
436
437   ###########################################################################
438   ### shooting end
439   ###########################################################################
440
441   ###########################################################################
442   ### perturbation begin
443   ###########################################################################
444
445       # энергии
446       # energies
```

```
447
448       # волновые функциий
449       wave = [forward_n0, forward_n1]
450
451       # Потенциальная функция невозмущенной системы
452       U0 = U
453
454       # Потенциальная функция возмущенной системы
455       # U1
456
457       # возмущение
458       # V
459
460       def matrix_V(n:int, m:int):
461           v_n_m = [wave[n][i] * v[i] * wave[m][i]
462                     for i in range(num_points)]
463           return integrate(v_n_m, step)
464
465       x = [begin + step * i for i in range(num_points)]
466       v = [V(xi) for xi in x]
467       u0 = [U0(xi) for xi in x]
468       u1 = [U1(xi) for xi in x]
469
470       energy_index = 0
471
472       # первая поправка энергии
473       first_order_correction_e = matrix_V(energy_index, energy_index)
474       print("первая поправка энергии=", first_order_correction_e)
475
476       # вторая поправка энергии
477       secont_order_correction_e = 0
478       for i in range(len(energies)):
479           if(i != energy_index):
480               secont_order_correction_e += (
481                   abs(matrix_V(energy_index, i)) ** 2
482               / #-------------------------------------
483                  (energies[energy_index] - energies[i])
484               )
485       print("первая поправка энергии=", secont_order_correction_e)
486
487       # первая поправка собственной функции
488       const =  matrix_V(0, 1) / (energies[0] - energies[1])
489       mul = [const * w for w in wave[1]]
490       first_order_correction_w = [wave[0][i] + mul[i]
491                                   for i in range(len(wave[1]))]
492
493       # вывод
494       energy = energies[0]
495       print("E_0 = ", energy)
496       energy += first_order_correction_e
```

```
497     print("E_0 = ", energy, " 1-oe приближение, ", "поправка =",
498 first_order_correction_e)
499     energy += secont_order_correction_e
500     print("E_0 = ", energy, " 2-oe приближение, ", "поправка =", se-
501 cont_order_correction_e)
502 
503     # Графики
504     plt.axis([begin, end, min_energy, v0])
505     plt.grid(True)
506     plt.axhline(0, color='black')
507     plt.axvline(0, color='black')
508     plt.xlabel("x, B", fontsize = 20, color = "k")
509     plt.ylabel("$U^0(x)$, a.u.", fontsize = 20, color = "k")
510     plt.plot(x, u0, linewidth = 6, color = "red", label = "$U^0(x)$")
511     plt.legend()
512     plt.show()
513 
514     plt.axis([begin, end, min_energy, v0])
515     plt.grid(True)
516     plt.axhline(0, color='black')
517     plt.axvline(0, color='black')
518     plt.xlabel("x, B", fontsize = 20, color = "k")
519     plt.ylabel("U(x), a.u.", fontsize = 20, color = "k")
520     plt.plot(x, u1, linewidth = 6, color = "red", label = "U(x)")
521     plt.legend()
522     plt.show()
523 
524     plt.axis([begin, end, min_energy, v0 if v0 > 1 else 1])
525     plt.grid(True)
526     plt.axhline(0, color='black')
527     plt.axvline(0, color='black')
528     plt.xlabel("x, B", fontsize = 20, color = "k")
529     plt.plot(x, forward_n0_pr, linewidth = 6,
530              color = "orange",
531              label = "$phi(x)$, a.u., метод пристрелки")
532     plt.plot(x, first_order_correction_w, linewidth = 3,
533              color = "blue",
534              label = "$phi(x)$, a.u., теория возмущений")
535     plt.plot(x, u1, linewidth = 6,
536              color = "green",
537              label = "$U(x)$, a.u.")
538     plt.legend()
539     plt.show()
540 
541 ########################################################################
542 ### perturbation end
543 ########################################################################
544 
545 main()
```