МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
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
Выполнил: студент 11 группы 2 курса магистратуры
Крутько А.С.
Преподаватель: доктор физ.-мат. наук, профессор Тимошенко Ю.К.

Воронеж 2024

Содержание

1 Цели и задачи работы

1.1 Цель 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 развитие алгоритмического мышления и приобретение опыта использования знаний и навыков по математике, численным методам и программированию для решения прикладных задач физико-технического характера.

1.2 Задачи работы:

Проблема: электрон находится в одномерной потенциальной яме с бесконечными стенками $U(x)$:

$$v(x) = \begin{cases} L_5(x), & x \in (-L, L), \\ \infty, & x \notin (-L, L), \end{cases}$$

Где $U(x) = v(x) * V_0$, $V_0 = 25$ эВ, $L = 3$ Å, $L_n(x)$ – полином Ляггера, $n = 5$.

1. Используя метод пристрелки, найти энергии, нормированные волновые функции и плотности вероятности для основного и 2-го возбужденного состояний. Привести как численные значения энергий, так и построить графики волновых функций и плотностей вероятности.

2. Вычислить для этих состояний квантовомеханические средние $\langle p(x) \rangle$ и $\langle p(x^2) \rangle$.

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

Одномерное стационарное уравнение Шрёдингера [2]:

$$\hat{H}\psi(x) = E\psi(x), \tag{1}$$

где $\hat{H}$ – оператор Гамильтона, E – собственные значения энергии, $\psi(x)$ – волновая функция.

С математической точки зрения оно представляет собой задачу определения собственных значений E и собственных функций ψ оператора Гамильтона $\hat{H}$. Для частицы с массой m, находящейся в потенциальном поле $U(x)$, оператор Гамильтона имеет вид

$$\hat{H} = \hat{T} + U(x), \tag{2}$$

где оператор кинетической энергии

$$\hat{T} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2}, \tag{3}$$

а $\hbar$ — постоянная Планка. Собственное значение оператора Гамильтона имеет смысл энергии соответствующей изолированной квантовой системы. Собственные функции называются волновыми функциями. Волновая функция однозначна и непрерывна во всём пространстве. Непрерывность волновой функции и её первой производной сохраняется и при обращении $U(x)$ в ∞ некоторой области пространства. В такую область частица вообще не может проникнуть, то есть в этой области, а также на её границе $\psi(x) = 0$.

Оценим нижнюю границу энергетического спектра. Пусть минимальное значение потенциальной функции равно $U_{\min}$. Очевидно, что $\langle T \rangle \geq 0$ и $\langle U \rangle \geq U_{\min}$. Потому из уравнения (1) следует, что:

$$E = \langle H \rangle = \int_{-\infty}^{+\infty} \psi^{\star}(x)\hat{H}\psi(x)\,dx = \langle T \rangle + \langle U \rangle > U_{\min}. \tag{4}$$

то есть, энергии всех состояний $> U_{min}$.

Особый практический интерес представляет случай, когда

$$\lim_{x\to\infty} U(x) = 0. \tag{5}$$

Потенциал такого типа называется также потенциальной ямой. Для данной $U(x)$ свойства решений уравнения Шрёдингера зависят от знака собственного значения E. Если $E < 0$. Частица с отрицательной энергией совершает финитное движение. Оператор Гамильтона имеет дискретный спектр, то есть собственные значения и соответствующие собственные функции можно снабдить номерами. При $E < 0$ уравнение (1) приобретает вид[2]:

$$\hat{H}\psi_k(x) = E_k\psi_k(x). \tag{6}$$

Квантовое состояние, обладающее наименьшей энергией, называется основным. Остальные состояния называют возбужденными состояниями. 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:

$$\int_{-\infty}^{+\infty} |\psi_k(x)|^2, dx = 1. \tag{7}$$

В дальнейшем будет рассматриваться только дискретный спектр. При этом необходимо пользоваться **осцилляционной теоремой**.

Осцилляционная теорема. Упорядочим собственные значения оператора Гамильтона в порядке возрастания, нумеруя энергию основного состояния индексом "0": E_0, E_1, E_2 …, E_k,… Тогда волновая функция $\psi_k(x)$ будет иметь k узлов (то есть, пересечений с осью абсцисс). Исключения: области, в которых потенциальная функция бесконечна.

3 Метод пристрелки. Алгоритм

Прежде чем перейти к алгоритму пристрелки нужно преобразовать уравнение (1). В данном случае граничные условия для волновой функции [2]:

$$\psi(a) = \psi(b) = 0 \tag{8}$$

где в точках a и b по оси абсцисс построены бесконечные потенциальные стенки. Для решения уравнения Шрёдингера удобно использовать атомные единицы Хартри ($e = 1, \hbar = 1$ и $m_e = 1$). В этих единицах уравнение (6), предполагая, что $m = m_e$ приобретает вид [2]:

$$\left[-\frac{1}{2}\frac{d^2}{dx^2} + U(x)\right]\psi x = E\psi x. \tag{9}$$

Преобразуем (9) к форме:

$$\frac{d^2\psi x}{dx^2} + q(E,x)\psi(x) = 0, \tag{10}$$

где

$$q(E,x) = 2\left[E - U(x)\right]. \tag{11}$$

Решение стационарного уравнения Шрёдингера сводится к нахождению собственных значений и собственных функций оператора Гамильтона, так как для собственных значений известна оценка снизу (4), то удобно начинать с вычисления энергии и волновой функции основного состояния. Оценим грубо энергию основного состояния $E_0^{(0)} = U_{min} + \delta$,где δ - малая величина ($\delta > 0$). Подставим значение этой энергии в уравнение (9). Это уравнение теперь становится обыкновенным дифференциальным уравнением 2-го порядка с граничными условиями (8). Рассмотрим алгоритм, использующий эту идею.

Зададим на интервале $[a, b]$ сетку из N узлов с постоянным шагом $h = \frac{(b-a)}{N-1}$:

$$x_n = a + (n-1)h, n = 1, 2, 3, \ldots, N. \tag{12}$$

Граничные условия (8) приобретают вид:

$$\psi_1 = \psi_N = 0. \tag{13}$$

Задача Коши для дифференциального уравнения (9) часто решается методом Нумерова. В рамках этого метода значения функции в узле сетки находят интегрируя «вперёд»:

$$\psi_{n+1} = \left[2(1 - 5cq_n)\psi_n - (1 + cq_{n-1})\psi_{n-1}\right]\left(1 + cq_{n+1}^{-1}\right), \tag{14}$$

либо интегрируя «назад»

$$\psi_{n+1} = \left[2(1 - 5cq_n)\psi_n - (1 + cq_{n+1})\psi_{n+1}\right]\left(1 + cq_{n-1}^{-1}\right), \tag{15}$$

Здесь $c = h^2/12, q_n = q(E, x_n)$. При использовании формулы (14) необходимо знать ψ_1 и ψ_2, а формулы (15) — ψ_{N-1} и ψ_N. Значения ψ_1 и ψ_N нам известны (13), а ψ_2 и ψ_{N-1} — нет. Однако, если N достаточно, то для простоты можно считать, что $\psi_2 = d2, \psi_{N-1} = d2$, где $d1, d2$ — малые числа.

Для оценки близости E к собственному значению будем вычислять разность производных волновых функций, полученных интегрированием «вперёд»и «назад», в некотором внутреннем узле сетки x_m:

$$f(E) = \left.\frac{d\psi_{>}x}{dx}\right|_{x_m} - \left.\frac{d\psi_{<}x}{dx}\right|_{x_m}. \tag{16}$$

где

$$\left.\frac{d\psi}{dx}\right|_{x_m} = \frac{\psi(x_m - 2h) - \psi(x_m + 2h) + 8\left[\psi(x_m + h) - \psi(x_m - h)\right]}{12h}. \tag{17}$$

Здесь $\psi_{>}, \psi_{<}$ - волновые функции полученные интегрированием «вперёд» и «назад» соответственно, x_m — узел сшивки производных. Естественно, перед вычислением (16) необходимо масштабировать функции $\psi_{>}$ и $\psi_{<}$ так, чтобы $\psi_{>}(x_m) = \psi_{<}(x_m)$.

Будем увеличивать энергию с шагом $\triangle E$ до тех пор, пока величины $f^{(i)}$ на двух соседних шагах i и $i-1$ не будут иметь различные знаки. Далее для уточнения собственного значения с наперед заданной точностью ϵ используется метод бисекции.

4 Программная реализация алгоритма

В Приложение программная реализация задачи выполнена на языке Python 3.12 [3], реализованная в среде разработки PyCharm Community Edition 2024.3.1, численного решения одномерного стационарного уравнения Шредингера для электрона в одномерной потенциальной яме. Программа реализует алгоритм пристрелки, позволяющей находить собственные значения и соответствующие им волновые функции. Потенциальная функция и параметры для нее соответствуют постановке задачи из первой главы (1).

Код программы написан таким образом, что решение поставленной задачи выведено в отдельный файл **solve.py**. В данном файле приводится класс решающий основную часть задачи, а также вспомогательные функции. Рассмотрим код данного файла.

Вспомогательная функция `draw_potential_graph()` отвечает за отрисовку отдельно от основного кода графика потенциальной функции.

Ниже рассматривается код класса `Solver` – класса, отвечающего за решение поставленной задачи:

Задание параметров из постановки задачи происходит в конструкторе класса `Solver`, такие как: количество точек системы n, параметры потенциальной функции L, V_0, задаётся приближенное значение энергии E_{min} и шаг $step$, также задаётся узел сшивки r. Кроме того в нем же энергия и длина ямы были переведены в атомные единицы Хартри (строки 46–49).

В методе `u_func` (строки 64–75) реализована потенциальная функция $U(x)$ из постановки задачи.

В методе `find_exact_energies` класса (строки 147–158) реализован метод пристрелки соответсвующий алгоритму из главы 3.

Реализованы методы `energy_scan` и `bisection_method` в строках 136–145 и 160–172 соответственно. Метод `find_exact_energies` вычисляет все значения энергий на отрезке $[E_{min}, E_{max}]$ с шагом `step`, метод `bisection_method` реализует метод бисекции.

В строках 108–134 реализован метод в котором с помощью метода Нумерова вычисляются волновые функции интегрированием вперед и назад, а также вычисляется разность этих функций в узле сшивки. В строках 77–78 реализована часть уравнения Шрёдингера и в строках 81–83 реализована формула для вычисления производной в узле сшивки.

В строках 86–88 реализована метод для нормировки волновой функции.

В строках 92–97 реализовано вычисление квантовомеханических средних по формуле $\langle P_x \rangle = \int_a^b \Psi_n(x) \hat{P} \Psi_n(x), dx = -i\hbar \int_a^b \Psi_n(x) \frac{d\Psi_n(x)}{dx}, dx$.

В строках 174–212 реализован метод для вывода графиков и записи данных в файл.

В строках 215–225 реализован основной метод класса вызывается метод пристрелки и происходит вычисление всех состояний на заданном диапазоне.

5 Результаты численных экспериментов

Ниже продемонстрированы результаты работы программного кода написанного на Python.

5.1 Иллюстрация работы программы

Потенциал из постановки задачи представлен на Рис. 1

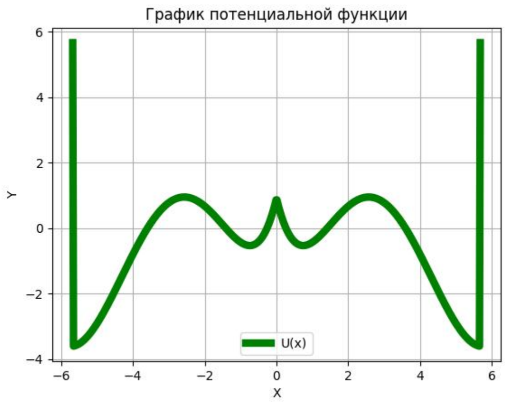
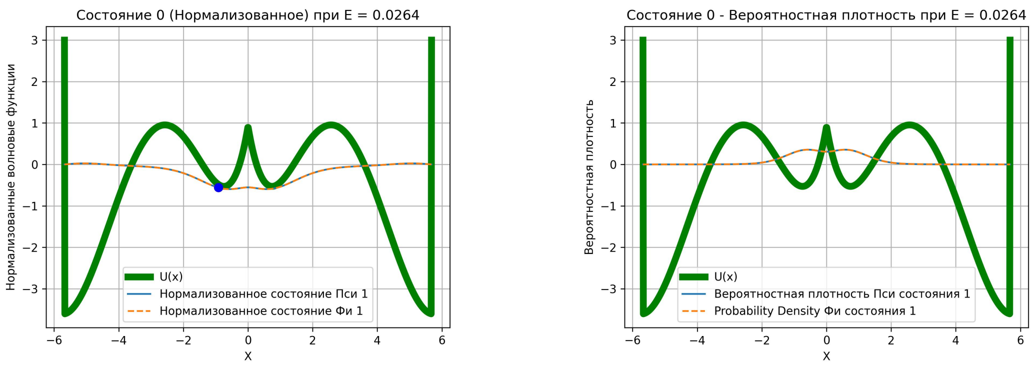


Рис. 1: Вероятностная плотность

Для основного состояния была получена энергия $E = 0.026429$ и следующая волновая функция (Рис. 2a) и плотность вероятности (Рис. 2b):



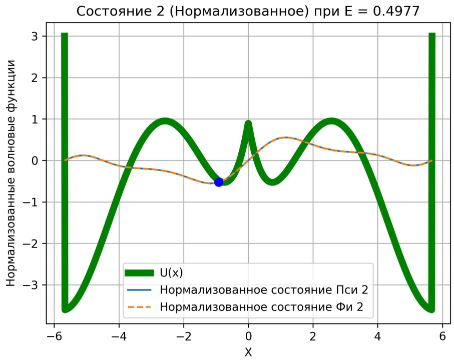
(a) Нормализованное состояние

(b) Вероятностная плотность

Рис. 2: Графики для основного состояния

При анализе заданной потенциала было обнаружено: основным состоянием для него является состояние при котором волновая функция имеет два пересечения с осью абсцисс.

Для второго возбужденного состояния была получена энергия $E = 0.815564$ и следующая волновая функция (Рис. 3a) и плотность вероятности (Рис. 3b):



(a) Нормализованное состояние

(b) Вероятностная плотность

Рис. 3: Графики для состояния 2

На рисунке (3a) можно заметить что волновая функция имеет 4 пересечения с осью абсцисс. Соответственно, согласно осцилляционной теореме, функция соответствует второму возбужденному состоянию так как есть четыре пересечения с осью абсцисс, кроме бесконечных потенциальных стенок которые не учитываются.

5.2 Значения искомых параметров

Ниже результаты численных экспериментов, полученных в результате работы программы выведены в таблицу:

Квантовомеханические средние $\langle p(x)\rangle$ и $\langle p(x^2)\rangle$ для основного, первого и второго возбужденного состояний:

Состояние	**Энергия, а.е.**	$\langle p(x)\rangle$	$\langle p(x^2)\rangle$
Основное	0.026429	$0.000000e+00$	$3.070530e-01$
2-е возбужденное	0.815564	$0.000000e+00$	$2.435363e+00$

6 Заключение

Таким образом, было получено численное решение для задачи о частице в одномерной квантовой яме с бесконечными стенками при помощи метода пристрелки. Были получены значения энергий и волновые функции основного и второго возбужденного состояний. Полученные волновые функции соответствуют осцилляционной теореме, в данном варианте можно четко заметить что, по сравнению с основным состоянием, количество пересечений с осью абсцисс увеличилось на два. Кроме того, для каждого состояния были вычисленные квантовомеханические средние $\langle p(x) \rangle$, $\langle p(x^2) \rangle$.

Приложение

```
"""
                  .  .
                                                                         :

                                                      Python 3.0,
                                    PyCharm Community Edition 2024.2.1,
                                         Windows 10.
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import eval_laguerre

def draw_potential_graph():
    n = 500
    c_energy = 27.212
    c_length = 0.5292
    v0 = 25.0 / c_energy
    l = 3.0 / c_length
    a, b = -l, l
    x = np.linspace(a - 0.01, b + 0.01, n)

    def u_func():
        u_val = np.zeros(n)
        for i in range(n):
            if np.abs(x[i]) <= l:
                u_val[i] = v0 * eval_laguerre(5, np.abs(x[i]))
            else:
                u_val[i] = l

        return u_val

    y = u_func()

    plt.plot(x, y, 'g-', linewidth=6.0, label="U(x)")
    plt.title(f"                                                    ")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid(True)
    plt.legend()

    plt.savefig('Potential_func_graph.jpg')
    plt.show()


class Solver:
    # Params
    def __init__(self):
        self.U_min = 0.0
        self.c_energy = 27.212
        self.c_length = 0.5292
        self.V0 = 25.0 / self.c_energy
        self.L = 3.0 / self.c_length
        self.A, self.B = -self.L - 0.01, self.L + 0.01
        self.n = 1000
        self.h = (self.B - self.A) / (self.n - 1)
        self.c, self.W = self.h ** 2 / 12.0, 3.0
        self.Psi, self.Fi, self.X = np.zeros(self.n), np.zeros(self.n), np.
    linspace(self.A, self.B, self.n)
```

```
        self.r = (self.n - 1) // 2 - 80
        self.limit_value = 7.0

        self.d1, self.d2 = 1.e-09, 1.e-09
        self.tol = 1.e-06

        self.E_min, self.E_max, self.step = self.U_min, 3, 0.001


    def u_func(self, x):
        #                    ,                    x
        if np.isscalar(x):
            # x -
            return self.V0 * eval_laguerre(5, abs(x)) if abs(x) <= self.L else self.W
        u_val = np.zeros(self.n)
        for i in range(self.n):
            if np.abs(x[i]) <= self.L:
                u_val[i] = self.V0 * eval_laguerre(5, np.abs(x[i]))
            else:
                u_val[i] = self.W
        return u_val

    def q(self, e, x):
        return 2.0 * (e - self.u_func(x))


    @staticmethod
    def derivative_func(y, h, m):
        return (y[m - 2] - y[m + 2] + 8.0 * (y[m + 1] - y[m - 1])) / (12.0 * h)


    def normalize_wave_function(self, y):
        norm = np.sqrt(np.trapz(y ** 2, self.X))
        return y / norm


    @staticmethod
    def mean_momentum(psi, x):
        h_bar = 1.0
        d_psi_dx = np.gradient(psi, x)
        integrand = psi.conj() * d_psi_dx
        mean_px = -1j * h_bar * np.trapz(integrand, x)
        return mean_px.real


    @staticmethod
    def mean_square_momentum(psi, x):
        h_bar = 1.0
        d2_psi_dx2 = np.gradient(np.gradient(psi, x), x)
        integrand = psi.conj() * d2_psi_dx2
        mean_px2 = -h_bar**2 * np.trapz(integrand, x)
        return mean_px2.real

    def f_fun(self, e, n):
        f = np.array([self.c * self.q(e, self.X[i]) for i in np.arange(n)])
        self.Psi[0] = 0.0
        self.Fi[n - 1] = 0.0
        self.Psi[1] = self.d1
```

```
        self.Fi[n - 2] = self.d2

        for i in np.arange(1, n - 1, 1):
            p1 = 2.0 * (1.0 - 5.0 * f[i]) * self.Psi[i]
            p2 = (1.0 + f[i - 1]) * self.Psi[i - 1]
            self.Psi[i + 1] = (p1 - p2) / (1.0 + f[i + 1])

        for i in np.arange(n - 2, 0, -1):
            f1 = 2.0 * (1.0 - 5.0 * f[i]) * self.Fi[i]
            f2 = (1.0 + f[i + 1]) * self.Fi[i + 1]
            self.Fi[i - 1] = (f1 - f2) / (1.0 + f[i - 1])

        p1 = np.abs(self.Psi).max()
        p2 = np.abs(self.Psi).min()
        big = p1 if p1 > p2 else p2

        self.Psi[:] = self.Psi[:] / big

        coefficient = self.Psi[self.r] / self.Fi[self.r]
        self.Fi[:] = coefficient * self.Fi[:]

        return Solver.derivative_func(self.Psi, self.h, self.r) - Solver.derivative_func(self.Fi, self.h, self.r)

    def energy_scan(self, e_min, e_max, step):
        energies = []
        values = []
        e = e_min
        while e <= e_max:
            f_value = self.f_fun(e, self.n)
            energies.append(e)
            values.append(f_value)
            e += step
        return energies, values

    def find_exact_energies(self, e_min, e_max, step, tol):
        energies, values = self.energy_scan(e_min, e_max, step)
        exact_energies = []
        for i in range(1, len(values)):
            log1 = values[i] * values[i - 1] < 0.0
            log2 = np.abs(values[i] - values[i - 1]) < self.limit_value
            if log1 and log2:
                e1, e2 = energies[i - 1], energies[i]
                exact_energy = self.bisection_method(e1, e2, tol)
                self.f_fun(exact_energy, self.n)
                exact_energies.append(exact_energy)
        return exact_energies

    def bisection_method(self, e1, e2, tol):
        while abs(e2 - e1) > tol:
            e_mid = (e1 + e2) / 2.0
            f1, f2, f_mid = self.f_fun(e1, self.n), self.f_fun(e2, self.n), self.f_fun(e_mid, self.n)
            if f1 * f_mid < 0.0:
                e2 = e_mid
            else:
                e1 = e_mid
            if f2 * f_mid < 0.0:
                e1 = e_mid
            else:
```

```
                e2 = e_mid
        return (e1 + e2) / 2.0

    def plot_wave_functions(self, energies):
        for i, E in enumerate(energies):
            self.f_fun(E, self.n)
            psi_norm = self.normalize_wave_function(self.Psi.copy())
            fi_norm = self.normalize_wave_function(self.Fi.copy())
            mean_px = Solver.mean_momentum(fi_norm, self.X)
            mean_px2 = Solver.mean_square_momentum(fi_norm, self.X)
            file = open("result.txt", "w")
            file.close()
            file1 = open("result.txt", "a")
            print(f"                   {i}: E = {E:.6f}, <p_x> = {mean_px:.6e}, <p_x^2> = {mean_px2:.6e}")
            print(f"                   {i}: E = {E:.6f}, <p_x> = {mean_px:.6e}, <p_x^2> = {mean_px2:.6e}", file = file1)

            plt.scatter(self.X[self.r], psi_norm[self.r], color='red', s=50, zorder=5)  #                  Psi
            plt.scatter(self.X[self.r], fi_norm[self.r], color='blue', s=50, zorder=5)  #                  Fi
            plt.plot(self.X, [self.u_func(x) for x in self.X], 'g-', linewidth=6.0, label="U(x)")
            plt.plot(self.X, psi_norm, label=f"          {i+1}")
            plt.plot(self.X, fi_norm, '--', label=f"          {i+1}")
            plt.title(f"                    {i + 1 if i != 0 else i} (          )         E = {E:.4f}")
            plt.xlabel("X")
            plt.ylabel("          ")
            plt.grid(True)
            plt.legend()
            plt.savefig(f"Condition_{i + 1 if i != 0 else i}_(normalized).jpg", dpi=300)
            plt.show()


            prob_density_psi = psi_norm**2
            prob_density_fi = fi_norm**2
            plt.plot(self.X, [self.u_func(x) for x in self.X], 'g-', linewidth=6.0, label="U(x)")
            plt.plot(self.X, prob_density_psi, label=f"          {i+1}")
            plt.plot(self.X, prob_density_fi, '--', label=f"Probability Density                {i+1}")
            plt.title(f"                    {i + 1 if i != 0 else i} -          E = {E:.4f}")
            plt.xlabel("X")
            plt.ylabel("                              ")
            plt.grid(True)
            plt.legend()
            plt.savefig(f"Condition_{i + 1 if i != 0 else i}_(Probability_density).jpg", dpi=300)
            plt.show()
```

```
    def solve(self):
        exact_energies = self.find_exact_energies(self.E_min, self.E_max, self.step, self.tol)

        if len(exact_energies) == 0:
            print("                :                .")
        else:
            print("                  :")
            for i, E in enumerate(exact_energies):
                print(f"                    {i}:                  = {E:.6f}")

            self.plot_wave_functions(exact_energies)
```

Листинг 1: Код файла solver.py

Список литературы

[1] Ландау Л.Д., Лифшиц Е.М. *Квантовая механика.* М.: Физматлит, 2004.

[2] Тимошенко Ю.К. *Численное решение стационарного уравнения Шрёдингера.* Воронеж, 2019.

[3] Бизли Д. *Python. Подробный справочник.* СПб.: Символ-Плюс, 2010.