

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	3
1.1	Цель работы.	3
1.2	Задачи работы:	3
2	Одномерное стационарное уравнение Шрёдингера. Математический формализм. Общие свойства решений	4
3	Метод пристрелки. Алгоритм	5
4	Программная реализация алгоритма	7
5	Результаты численных экспериментов	8
5.1	Иллюстрация работы программы	8
5.2	Значения искомых параметров	9
6	Заключение	10

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 $U(x)$:

$$v(x) = \begin{cases} J_2(x), & x \in (-L, L), \\ \infty, & x \notin (-L, L), \end{cases}$$

Где $U(x) = v(x) * V_0$, $V_0 = 25$ эВ, $L = 3$ Å, $J_n(x)$ – функция Бесселя, $n = 2$.

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]:

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

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

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

где оператор кинетической энергии

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

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:

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

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

Особый практический интерес представляет случай, когда

$$\lim_{x \rightarrow \infty} U(x) = 0. \quad (5)$$

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]:

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

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:

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

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": $E_0, E_1, E_2 \dots, E_k, \dots$. 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]:

$$\psi(a) = \psi(b) = 0 \quad (8)$$

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]:

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

Преобразуем (9) к форме:

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

где

$$q(E, x) = 2 [E - U(x)]. \quad (11)$$

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 $h = \frac{(b-a)}{N-1}$:

$$x_n = a + (n-1)h, n = 1, 2, 3, \dots, N. \quad (12)$$

Граничные условия (8) приобретают вид:

$$\psi_1 = \psi_N = 0. \quad (13)$$

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

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

либо интегрируя «назад»

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

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 :

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

где

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

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 ΔE 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 42–45).

В методе `u_func` (строки 58–75) реализована потенциальная функция $U(x)$ из постановки задачи.

В метод `find_exact_energies` класса (строки 142–153) реализован метод пристрелки соответствующий алгоритму из главы 3.

Реализованы методы `energy_scan` и `bisection_method` в строках 131–140 и 155–167 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 110–129 реализован метод в котором с помощью метода Нумеров вычисляются волновые функции интегрированием вперед и назад, а также вычисляется разность этих функций в узле сшивки. В строках 72–73 реализована часть уравнения Шрёдингера и в строках 81–83 реализована формула для вычисления производной в узле сшивки.

В строках 81–83 реализована метод для нормировки волновой функции.

В строках 86–92 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 169–207 реализован метод для вывода графиков и записи данных в файл.

В строках 210–219 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

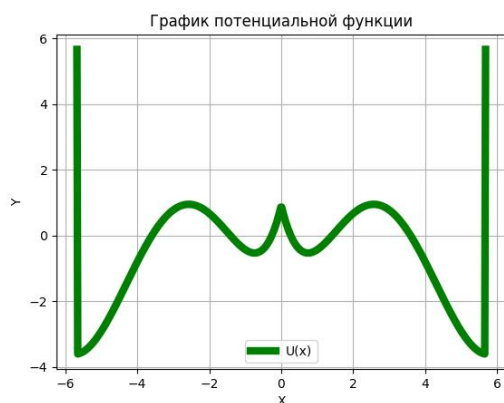
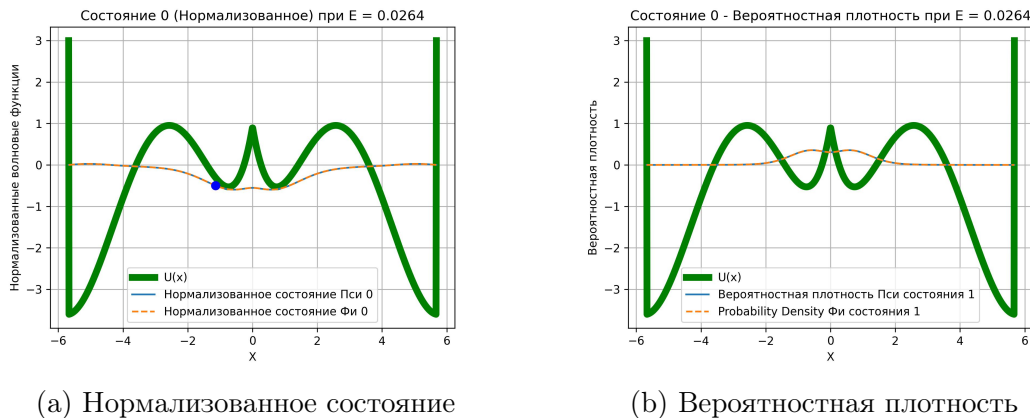


Рис. 1: Вероятностная плотность

Для основного состояния была получена энергия $E = 0.498856$ и следующая волновая функция (Рис. 2a) и плотность вероятности (Рис. 2b):



(a) Нормализованное состояние

(b) Вероятностная плотность

Рис. 2: Графики для состояния 0

На рисунках можно видеть, что функции, согласно осцилляционной теореме, соответствуют основному состоянию так как нет пересечений с осью абсцисс, кроме бесконечных потенциальных стенок которые не учитываются.

Для второго возбужденного состояния была получена энергия $E = 0.828367$ и следующая волновая функция (Рис. 3a) и плотность вероятности (Рис. 3b):



(a) Нормализованное состояние



(b) Вероятностная плотность

Рис. 3: Графики для состояния 1

На рисунках можно видеть что функции, согласно осцилляционной теореме, соответствуют 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	0.026451	0.000000e + 00	3.072237e - 01
1-е возбужденное	0.498856	0.000000e + 00	1.006510e + 00
2-е возбужденное	0.828367	0.000000e + 00	2.410640e + 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. Кроме того, для каждого состояния были вычисленные квантовомеханические средние $\langle p(x) \rangle, \langle p(x^2) \rangle$.

Приложение

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.special import eval_laguerre
4
5 def draw_potential_graph():
6     n = 500
7     c_energy = 27.212
8     c_length = 0.5292
9     v0 = 25.0 / c_energy
10    l = 3.0 / c_length
11    a, b = -l, l
12    x = np.linspace(a - 0.01, b + 0.01, n)
13
14    def u_func():
15        u_val = np.zeros(n)
16        for i in range(n):
17            if np.abs(x[i]) <= l:
18                u_val[i] = v0 * eval_laguerre(5, np.abs(x[i]))
19            else:
20                u_val[i] = 1
21
22        return u_val
23
24    y = u_func()
25
26    plt.plot(x, y, 'g-', linewidth=6.0, label="U(x)")
27    plt.title(f"Potential function graph")
28    plt.xlabel("X")
29    plt.ylabel("Y")
30    plt.grid(True)
31    plt.legend()
32
33    plt.savefig('Potential_func_graph.jpg')
34    plt.show()
35
36
37 class Solver:
38     # Params
39     def __init__(self):
40         self.U_min = -0.149124
41         self.c_energy = 27.212
42         self.c_length = 0.5292
43         self.V0 = 25.0 / self.c_energy
44         self.L = 3.0 / self.c_length
45         self.A, self.B = -self.L, self.L
46         self.n = 650
47         self.h = (self.B - self.A) / (self.n - 1)
48         self.c, self.W = self.h ** 2 / 12.0, 3.0
49         self.Psi, self.Fi, self.X = np.zeros(self.n), np.zeros(self.n), np.
50         linspace(self.A, self.B, self.n)
51         self.r = (self.n - 1) // 2 - 80
52         self.limit_value = 4.0
53
54         self.d1, self.d2 = 1.e-09, 1.e-09
55         self.tol = 1e-6
56
57         self.E_min, self.E_max, self.step = self.U_min + 0.01, 2.0, 0.01
```

```

58
59     def u_func(self, x):
60         # Check if x is a scalar
61         if np.isscalar(x):
62             # x - scalar
63             return self.V0 * eval_laguerre(5, abs(x)) if abs(x) <= self.L
64     else self.W
65         u_val = np.zeros(self.n)
66         for i in range(self.n):
67             if np.abs(x[i]) <= self.L:
68                 u_val[i] = self.V0 * eval_laguerre(5, np.abs(x[i]))
69             else:
70                 u_val[i] = self.L
71         return u_val
72
73     def q(self, e, x):
74         return 2.0 * (e - self.u_func(x))
75
76     @staticmethod
77     def derivative_func(y, h, m):
78         return (y[m - 2] - y[m + 2] + 8.0 * (y[m + 1] - y[m - 1])) / (12.0 *
79             h)
80
81     def normalize_wave_function(self, y):
82         norm = np.sqrt(np.trapz(y ** 2, self.X))
83         return y / norm
84
85     @staticmethod
86     def mean_momentum(psi, x):
87         h_bar = 1.0
88         d_psi_dx = np.gradient(psi, x)
89         integrand = psi.conj() * d_psi_dx
90         mean_px = -1j * h_bar * np.trapz(integrand, x)
91         return mean_px.real
92
93     @staticmethod
94     def mean_square_momentum(psi, x):
95         h_bar = 1.0
96         d2_psi_dx2 = np.gradient(np.gradient(psi, x), x)
97         integrand = psi.conj() * d2_psi_dx2
98         mean_px2 = -h_bar**2 * np.trapz(integrand, x)
99         return mean_px2.real
100
101     def f_fun(self, e, n):
102         f = np.array([self.c * self.q(e, self.X[i]) for i in np.arange(n)])
103         self.Psi[0] = 0.0
104         self.Fi[n - 1] = 0.0
105         self.Psi[1] = self.d1
106         self.Fi[n - 2] = self.d2
107
108         for i in np.arange(1, n - 1, 1):
109             p1 = 2.0 * (1.0 - 5.0 * f[i]) * self.Psi[i]
110             p2 = (1.0 + f[i - 1]) * self.Psi[i - 1]
111             self.Psi[i + 1] = (p1 - p2) / (1.0 + f[i + 1])
112
113         for i in np.arange(n - 2, 0, -1):

```

```

116         f1 = 2.0 * (1.0 - 5.0 * f[i]) * self.Fi[i]
117         f2 = (1.0 + f[i + 1]) * self.Fi[i + 1]
118         self.Fi[i - 1] = (f1 - f2) / (1.0 + f[i - 1])
119
120     p1 = np.abs(self.Psi).max()
121     p2 = np.abs(self.Psi).min()
122     big = p1 if p1 > p2 else p2
123
124     self.Psi[:] = self.Psi[:] / big
125
126     coefficient = self.Psi[self.r] / self.Fi[self.r]
127     self.Fi[:] = coefficient * self.Fi[:]
128
129     return Solver.derivative_func(self.Psi, self.h, self.r) - Solver.
derivative_func(self.Fi, self.h, self.r)
130
131     def energy_scan(self, e_min, e_max, step):
132         energies = []
133         values = []
134         e = e_min
135         while e <= e_max:
136             f_value = self.f_fun(e, self.n)
137             energies.append(e)
138             values.append(f_value)
139             e += step
140         return energies, values
141
142     def find_exact_energies(self, e_min, e_max, step, tol):
143         energies, values = self.energy_scan(e_min, e_max, step)
144         exact_energies = []
145         for i in range(1, len(values)):
146             log1 = values[i] * values[i - 1] < 0.0
147             log2 = np.abs(values[i] - values[i - 1]) < self.limit_value
148             if log1 and log2:
149                 e1, e2 = energies[i - 1], energies[i]
150                 exact_energy = self.bisection_method(e1, e2, tol)
151                 self.f_fun(exact_energy, self.n)
152                 exact_energies.append(exact_energy)
153         return exact_energies
154
155     def bisection_method(self, e1, e2, tol):
156         while abs(e2 - e1) > tol:
157             e_mid = (e1 + e2) / 2.0
158             f1, f2, f_mid = self.f_fun(e1, self.n), self.f_fun(e2, self.n),
self.f_fun(e_mid, self.n)
159             if f1 * f_mid < 0.0:
160                 e2 = e_mid
161             else:
162                 e1 = e_mid
163             if f2 * f_mid < 0.0:
164                 e1 = e_mid
165             else:
166                 e2 = e_mid
167         return (e1 + e2) / 2.0
168
169     def plot_wave_functions(self, energies):
170         for i, E in enumerate(energies):
171             self.f_fun(E, self.n)
172             psi_norm = self.normalize_wave_function(self.Psi.copy())
173             fi_norm = self.normalize_wave_function(self.Fi.copy())

```

```

174         mean_px = Solver.mean_momentum(fi_norm, self.X)
175         mean_px2 = Solver.mean_square_momentum(fi_norm, self.X)
176         file = open("result.txt", "w")
177         file.close()
178         file1 = open("result.txt", "a")
179         print(f"Condition {i}: E = {E:.6f}, <p_x> = {mean_px:.6e}, <p_x
~2> = {mean_px2:.6e}")
180         print(f"Condition {i}: E = {E:.6f}, <p_x> = {mean_px:.6e}, <p_x
~2> = {mean_px2:.6e}", file = file1)
181
182         plt.scatter(self.X[self.r], psi_norm[self.r], color='red', s=50,
zorder=5) # Point at Psi
183         plt.scatter(self.X[self.r], fi_norm[self.r], color='blue', s=50,
zorder=5) # Point at Fi
184         plt.plot(self.X, [self.u_func(x) for x in self.X], 'g-',
linewidth=6.0, label="U(x)")
185         plt.plot(self.X, psi_norm, label=f"Normalized condition Psi {i}"
)
186         plt.plot(self.X, fi_norm, '--', label=f"Normalized condition Phi
{i}")
187         plt.title(f"Condition {i} (Normalized) for E = {E:.4f}")
188         plt.xlabel("X")
189         plt.ylabel("Normalized wave functions")
190         plt.grid(True)
191         plt.legend()
192         plt.savefig(f"Condition_{i}_(normalized).jpg", dpi=300)
193         plt.show()
194
195
196         prob_density_psi = psi_norm**2
197         prob_density_fi = fi_norm**2
198         plt.plot(self.X, [self.u_func(x) for x in self.X], 'g-',
linewidth=6.0, label="U(x)")
199         plt.plot(self.X, prob_density_psi, label=f"Probability density
Psi condition {i+1}")
200         plt.plot(self.X, prob_density_fi, '--', label=f"Probability
Density Phi condition {i+1}")
201         plt.title(f"Condition {i} - Probability density where E = {E:.4f
}")
202         plt.xlabel("X")
203         plt.ylabel("Probability density")
204         plt.grid(True)
205         plt.legend()
206         plt.savefig(f"Condition_{i}_(Probability_density).jpg", dpi=300)
207         plt.show()
208
209
210     def solve(self):
211         e_min, e_max, step = self.U_min + 0.01, 3.0, 0.01
212         exact_energies = self.find_exact_energies(e_min, e_max, step, self.
tol)
213
214         if len(exact_energies) == 0:
215             print("Error: energies were not found.")
216         else:
217             print("Energies:")
218             for i, E in enumerate(exact_energies):
219                 print(f"Condition {i}: Energy = {E:.6f}")
220

```

```
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.