

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 №3
ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНОГО
УРАВНЕНИЯ ШРЁДИНГЕРА:
ВАРИАЦИОННЫЙ ПРИНЦИП

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. Прямой вариационный метод (метод Ритца).....	5
2. Применение вариационного метода.....	9
3. Программная реализация.....	11
4. Результаты численных экспериментов и их анализ	12
Список использованных источников	13
Приложения	14
Приложение 1 Компьютерный код.....	14

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 вида $U(x) = V_0 \cdot v(x)$, $x \in (-L, +L)$ (рис. 1):

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

где $V_0 = 15$ эВ, $L = 2 \text{ \AA}$, $n = 1$.

С помощью вариационного метода найти:

- энергию основного состояния;
- волновую функцию основного состояния.

В качестве базисного набора – волновые функции частицы в одномерной прямоугольной яме с бесконечными стенками. Сравнить результаты с данными, полученными методом пристрелки.

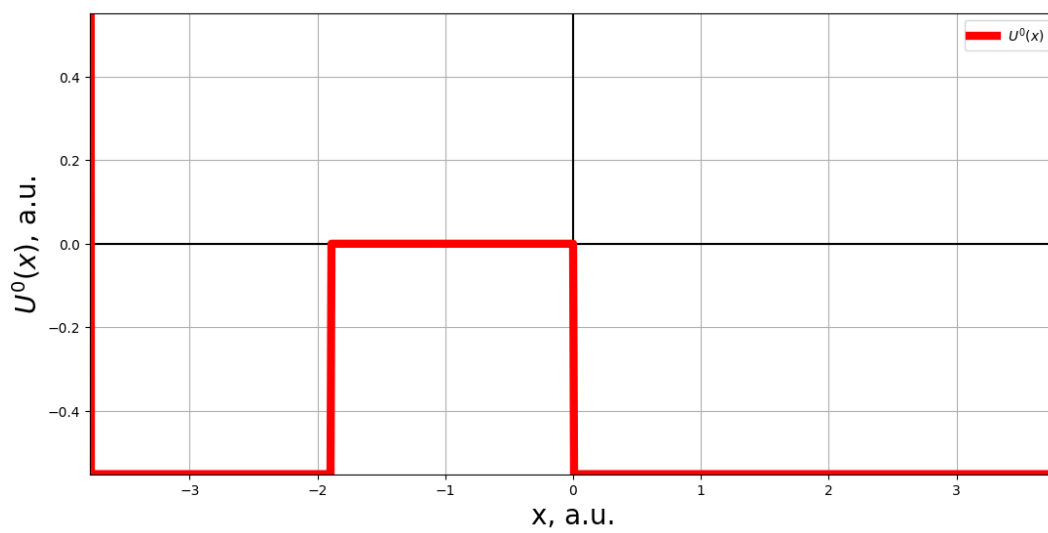


Рис. 1. График невозмущенной потенциальной функции

1. Одномерное стационарное уравнение Шрёдингера.

Прямой вариационный метод (метод Ритца)

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 с помощью прямого вариационного метода.

Рассмотрим стационарное уравнение Шрёдингера для дискретного спектра:

$$\hat{H}\psi_n(\vec{r}) = E_n\psi_n(\vec{r}). \quad (1)$$

Функции $\{\psi_n(\vec{r})\}$ ортонормированы:

$$\int \psi_m^*(\vec{r})\psi_n(\vec{r})d\vec{r} = \delta_{mn}. \quad (2)$$

Совокупность всех собственных функций для дискретного спектра $\{\psi_n(\vec{r})\}$ образует полную или замкнутую системы функций, т.е. любая другая функция $\tilde{\psi}(\vec{r})$, которая зависит от тех же переменных и удовлетворяет тем же граничным условиям, для которой существует интеграл $\int |\tilde{\psi}(\vec{r})|^2 dr$ может быть точно представлена в виде ряда:

$$\tilde{\psi}(\vec{r}) = \sum_n a_n \psi_n(\vec{r}). \quad (3)$$

Умножаем левую и правую части равенства (3) на $\psi_m^*(\vec{r})$ и проинтегрируем по всем пространству:

$$\int \tilde{\psi}(\vec{r})\psi_m^*(\vec{r})d\vec{r} = \sum_n a_n \int \psi_n(\vec{r})\psi_m^*(\vec{r})d\vec{r},$$

где $\int \psi_m^*(\vec{r})\psi_n(\vec{r})d\vec{r} = \delta_{mn}$.

Таким образом заменяя m на n , получаем:

$$a_n = \int \tilde{\psi}(\vec{r})\psi_n^*(\vec{r})d\vec{r}$$

В частности, можно разложить дельта-функцию Дирака в ряд:

$$\delta(\vec{r}' - \vec{r}) = \sum_n a_n(\vec{r}') \psi_n(\vec{r}), \quad (4)$$

где

$$a_n(\vec{r}') = \int \delta(\vec{r}' - \vec{r}) \psi_n^*(\vec{r}) d\vec{r} = \psi_n^*(\vec{r}').$$

То есть функцию (4) можно записать в виде:

$$\sum_n \psi_n^*(\vec{r}') \psi_n(\vec{r}) = \delta(\vec{r}' - \vec{r}) \quad (5)$$

Формулу (5) часто называют условие полноты базиса $\{\psi_n(\vec{r})\}$. Поэтому, когда говорят, что базис является ортонормированным и полным, то имеют ввиду:

$$\left\{ \begin{array}{l} \int \psi_m^*(\vec{r}) \psi_n(\vec{r}) d\vec{r} = \delta_{mn} \\ \sum_n \psi_n^*(\vec{r}') \psi_n(\vec{r}) = \delta(\vec{r}' - \vec{r}) \end{array} \right.$$

Так же равенство (2) в форме:

$$\int \psi_n^*(\vec{r}) \psi_n(\vec{r}) d\vec{r} = \int |\psi_n(\vec{r})|^2 d\vec{r} = 1$$

называется условие нормировки, а волновые функции, удовлетворяющие этому условию, называются нормированными функциями. Далее, для нормирования функций $\{\psi_n(\vec{r})\}$ величина $|\psi_n(\vec{r})|^2 d\vec{r}$ определяет вероятность $dW(\vec{r})$ значений координат системы в интервале $(\vec{r}, \vec{r} + d\vec{r})$ в состоянии с волновой функцией $\psi_n(\vec{r})$. В этом случае величину

$$\rho(\vec{r}) = \frac{dW(\vec{r})}{d\vec{r}} = |\psi_n(\vec{r})|^2$$

называют плотностью вероятности.

Вернёмся к уравнению Шрёдингера, умножим левую и правую части на $\psi_n^*(\vec{r})$ и проинтегрируем по всем пространству:

$$\int \psi_n^*(\vec{r}) \hat{H} \psi_n(\vec{r}) d\vec{r} = E_n \int \psi_n^*(\vec{r}) \psi_n(\vec{r}) d\vec{r}$$

Интеграл в левой части является квантовомеханическим средним значением \hat{H} в состоянии $\psi_n(\vec{r})$:

$$\langle \hat{H} \rangle = \int \psi_n^*(\vec{r}) \hat{H} \psi_n(\vec{r}) d\vec{r}$$

Найдём квантовомеханическое среднее гамильтониана в состоянии с волновой функцией $\psi_n(\vec{r})$, используя формулу (3):

$$\begin{aligned} \int \tilde{\psi}^*(\vec{r}) \hat{H} \tilde{\psi}(\vec{r}) d\vec{r} &= \sum_{n,m} a_m^* a_n \int \psi_m^*(\vec{r}) \hat{H} \psi_n(\vec{r}) d\vec{r} = \\ &= \sum_{n,m} a_m^* a_n \int \psi_m^*(\vec{r}) E_n \psi_n(\vec{r}) d\vec{r} = \\ &= \sum_{n,m} a_m^* a_n E_n \underbrace{\int \psi_m^*(\vec{r}) \psi_n(\vec{r}) d\vec{r}}_{\delta_{mn}} = \sum_{n,m} |a_n|^2 E_n \end{aligned} \quad (6)$$

Пусть $n = 0$ соответствует основному состоянию, тогда энергии возбуждённого состояния $E_i > E_0 \forall i \in [1, \infty)$. Заменяем в правой части (6) все E_n на E_0 . Тогда получим неравенство:

$$\int \tilde{\psi}^*(\vec{r}) \hat{H} \tilde{\psi}(\vec{r}) d\vec{r} \geq E_0 \sum_{n,m} |a_n|^2 = E_0. \quad (7)$$

Здесь используется информированность функции $\tilde{\psi}(\vec{r})$:

$$1 = \int \tilde{\psi}^*(\vec{r}) \tilde{\psi}(\vec{r}) d\vec{r} = \sum_{n,m} a_m^* a_n E_n \underbrace{\int \psi_m^*(\vec{r}) \psi_n(\vec{r}) d\vec{r}}_{\delta_{mn}} = \sum_{n,m} |a_n|^2$$

Равенство в (7) соответствует $\tilde{\psi}(\vec{r}) = \psi_0(\vec{r})$. Функцию $\tilde{\psi}(\vec{r})$ называют «пробной функцией», содержащей некоторое количество параметров c_1, c_2, \dots подлежащих определению.

В рамках вариационного подхода волновую функцию основного состояния с энергией E_0 приближённо ищут путём минимизации по параметрам c_1, c_2, \dots интеграла:

$$E_0 \simeq \int \tilde{\psi}^*(\vec{r}; c_1, c_2, \dots) \hat{H} \tilde{\psi}(\vec{r}; c_1, c_2, \dots) d\vec{r}$$

при условии нормировки:

$$\int |\tilde{\psi}(\vec{r}; c_1, c_2, \dots)|^2 d\vec{r} = 1$$

Обозначим

$$J(c_1, c_2, \dots) = \int \tilde{\psi}^*(\vec{r}; c_1, c_2, \dots) \hat{H} \tilde{\psi}(\vec{r}; c_1, c_2, \dots) d\vec{r} \quad (8)$$

Таким образом, для приближения вычисления волновой функции основного состояния необходимо найти минимум функции многих переменных (8):

$$\frac{\partial J}{\partial c_1} = \frac{\partial J}{\partial c_2} = \dots = \frac{\partial J}{\partial c_N} = 0$$

Такой метод отыскания волновой функции и энергии основного состояния называется прямым вариационным методом или методом Ритца.

2. Применение вариационного метода

Базисная функция должна быть близка к тому решению, которое мы хотим получить, поэтому для решения в качестве базисного набора можно выбрать волновые функции частицы в одномерной прямоугольной яме с бесконечными стенками, для которой известно аналитическое решение для собственных значений и собственных функций:

$$E_k = k^2 \cdot \frac{\pi^2 \hbar^2}{8ma^2}$$

$$\psi_k(x) = \begin{cases} \frac{1}{\sqrt{a}} \cos\left(\frac{\pi k x}{2a}\right), & k = 1, 3, 5, \dots \\ \frac{1}{\sqrt{a}} \sin\left(\frac{\pi k x}{2a}\right), & k = 2, 4, 6, \dots \end{cases}$$

Пусть имеется оператор \hat{H}^0 собственные функции $\{\varphi_k(x)\}$ которого образуют полный ортонормированный базис:

$$\hat{H}^0 \varphi_k(x) = E_k^0 \varphi_k(x)$$

где k соответствует квантовому состоянию.

Если имеется функция $\psi(x)$, зависящая от тех же параметров, что и базисная, удовлетворяющая тем же граничным условиям, что и базисная, то можно разложить эту функцию в виде линейной комбинации базисных:

$$\psi(x) = \sum_{k=0}^{\infty} c_k \varphi_k(x) \quad (9)$$

Подставим (9) в (1):

$$\hat{H} \sum_{k=0}^{\infty} c_k \varphi_k(x) = E \sum_{k=0}^{\infty} c_k \varphi_k(x)$$

Левую и правую части умножим на $\varphi_m^*(x)$ и проинтегрируем по всем пространству:

$$\sum_{k=0}^{\infty} c_k \int \varphi_m^*(x) \hat{H} \varphi_k(x) dx = E \sum_{k=0}^{\infty} c_k \underbrace{\int \varphi_m^*(x) \varphi_k(x) dx}_{\delta_{mk}} \quad (10)$$

Перепишем (10) в следующем виде:

$$\sum_{k=0}^{\infty} (H_{mk} - E \delta_{mk}) C_k = 0 \quad (11)$$

где

$$H_{mk} = \int \varphi_m^*(x) \hat{H} \varphi_k(x) dx \quad (12)$$

$$C_k = \sum_{m=0}^{\infty} c_m \delta_{km}$$

Для того чтобы применить на практике формулу (11) необходимо привести бесконечную систему к виду:

$$(H - E \cdot I)C = 0$$

где

$$H = \begin{pmatrix} H_{00} & \cdots & H_{0n} \\ \vdots & \ddots & \vdots \\ H_{n0} & \cdots & H_{nn} \end{pmatrix} \quad (13)$$

$$C = \begin{pmatrix} C_0^{(0)} & \cdots & C_0^{(n)} \\ \vdots & \ddots & \vdots \\ C_n^{(0)} & \cdots & C_n^{(n)} \end{pmatrix}$$

$$E = \begin{pmatrix} E_0 & 0 & \cdots & 0 \\ 0 & E_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & E_n \end{pmatrix}$$

$$I = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

Для численного нахождения второй производной используются формулы из первой лабораторной работы.

3. 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 1-338 находится реализация метода пристрелки, которая подробно была описана в отчёте по первой лабораторной работе. Дополнительно для решения одномерного стационарного уравнения Шрёдингера с помощью вариационного принципа были добавлены следующие функции:

- волновая функция для частицы в прямоугольной потенциальной яме 340-354;
- вычисление подынтегральной функции в формуле (12) 356-366;
- вычисление интеграла (12) 368-376;
- вычисление матрицы (13) 378-380.

В строках 382-384 реализовано решение одномерного стационарного уравнения Шрёдингера с помощью формулы (11). Далее в строках 385-388 находится минимальное собственное значение, которое соответствует энергии основного состояния. В строках 390-396 получаются значения волновой функции основного состояния с помощью формулы (9). В заключении в строках 402-415 строятся графики полученные с помощью вариационного принципа и метода пристрелки, для сравнения в следующей главе.

4. Результаты численных экспериментов и их анализ

В данной работе необходимо было найти энергию основного состояния и волновую функцию основного состояния.

На рисунке 2 представлены результаты численного моделирования для основного состояния системы методом Ритца – голубой график и методом пристрелки – оранжевый график, по которым видно, что функции, найденные двумя способами достаточно близки.

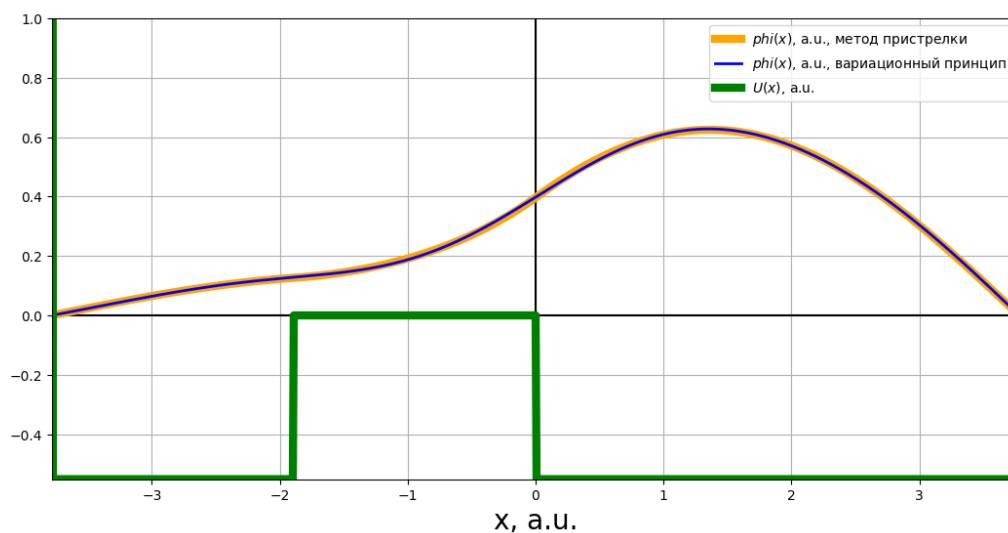


Рис. 2. Основное состояние

Энергия основного состояния, рассчитанная по методу Ритца равна:

$$E = -0.3401529582693288 \text{ а. е.}$$

Это значение является достаточно близким к значению энергии, найденной с помощью метода пристрелки:

$$E = -0.3404957040724568 \text{ а. е.}$$

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

```
1  import matplotlib.pyplot as plt
2  from math import sqrt, cos, sin, pi
3  from numpy import linalg as LA
4
5  #####
6  ### bisection_method
7  #####
8
9  # метод деления отрезка пополам
10 def bisection_method(f, # функция корень которой необходимо найти
11                      begin:float, end:float, # начало и конец отрезка
12                                           # в котором находится корень
13                      steps:int, # максимальное количество шагов
14                               # алгоритма
15                      eps:float):# точность приближения корня
16     l = begin
17     r = end
18
19     step = 0
20
21     while step < steps:
22         step += 1
23         center = 0.5 * (r + l)
24
25         f_l = f(l)
26         f_c = f(center)
27         f_r = f(r)
28
29         if abs(f_c) < eps:
30             break
31
32         if f_l * f_c < 0:
33             r = center
34         else: # f_c * f_r < 0
35             l = center
36
37     return 0.5 * (r + l)
38
39 #####
40 ### quantum_mechanics
41 #####
42
43 def v(x:float) -> float:
44     if (x > -half_width and x < (-half_width / 2.0)) or
45         (x > 0 and x < half_width):
46         return -1
```

```

47     elif x >= -half_width / 2.0 and x <= 0:
48         return 0
49     else:
50         return w
51
52     # потенциальная функция
53     def U(x:float) -> float:
54         return v0 * v(x)
55
56     def q(e:float,
57           x:float) -> float:
58         return 2.0 * (e - U(x))
59
60     def compute_q(x:list[float],
61                  energy:float) -> list[float]:
62         return [q(energy, xi) for xi in x]
63
64     def derivative(function:list[float],
65                   index:int,
66                   eps:float) -> float:
67         der1 = function[index - 2] - function[index + 2]
68         der2 = function[index + 1] - function[index - 1]
69         return (
70             (der1 + 8.0 * der2)
71             / #-----
72             (12.0 * eps)
73         )
74
75     # интегрирование вперёд
76     def forward_integration(num_intervals:int,
77                           penultimate_approximation:float,
78                           q:list[float],
79                           step:float) -> list[float]:
80         num_points = num_intervals + 1
81         forward = [float] * (num_points)
82
83         forward[0] = 0.0
84         forward[1] = penultimate_approximation
85
86         c = step ** 2 / 12.0
87
88         for i in range(1, num_intervals):
89             p1 = 2.0 * (1.0 - 5.0 * c * q[i]) * forward[i]
90             p2 = (1.0 + c * q[i - 1]) * forward[i - 1]
91             p3 = (1.0 + c * q[i + 1])
92             forward[i + 1] = (
93                 (p1 - p2)
94                 / #-----
95                 p3
96             )

```

```

97
98     return forward
99
100 # интегрирование назад
101 def backward_integration(num_intervals:int,
102                          first_approximation:float,
103                          q:list[float],
104                          step:float) -> list[float]:
105     num_points = num_intervals + 1
106     backward = [float] * num_points
107
108     backward[num_intervals] = 0
109     backward[num_intervals - 1] = first_approximation
110
111     c = step ** 2 / 12.0
112
113     for i in range(num_intervals - 1, 0, -1):
114         f1 = 2.0 * (1.0 - 5.0 * c * q[i]) * backward[i]
115         f2 = (1.0 + c * q[i + 1]) * backward[i + 1]
116         backward[i - 1] = (
117             (f1 - f2)
118             / #-----
119             (1.0 + c * q[i - 1])
120             )
121
122     return backward
123
124 # функция для нормировки forward
125 # и достижения равенства на узле сшивки
126 def normalization(forward:list[float],
127                   backward:list[float],
128                   connection:int):
129     # нормировка
130     norm = abs(max(forward, key = abs))
131     forward = list(map(lambda x: x / norm, forward))
132
133     # равенство на узле сшивки - connection
134     coef = forward[connection] / backward[connection]
135     backward = list(map(lambda x: coef * x, backward))
136
137     return forward, backward
138
139 # функция возвращающая разницу производных на узле сшивки
140 def is_close(forward:list[float],
141              backward:list[float],
142              connection:int,
143              eps:float) -> float:
144     return (derivative(forward, connection, eps)
145             - derivative(backward, connection, eps))
146

```



```

147 # функция вычисляет значения волновой функции вперёд и назад
148 # по известным данным
149 # и возвращающая разницу производных на узле сшивки
150 def is_close_energy(
151     energy:float, # значение энергии
152     x:list[float], # сетка
153     step:float, # шаг сетки
154     forward_first_approximation:float, # первое
155         # приближение для интегрирования вперёд
156     backward_penultimate_approximation:float, # n - 1
157         # приближение для интегрирования назад
158     connection:int) -> float: # узел сшивки
159     num_intervals = len(x) - 1
160     q = compute_q(x, energy)
161     forward = forward_integration(num_intervals,
162         forward_first_approximation, q, step)
163     backward = backward_integration(num_intervals,
164         backward_penultimate_approximation, q, step)
165     forward, backward = normalization(forward, backward, connection)
166     return is_close(forward, backward, connection, step)
167
168 # возвращает интервалы в которых находятся
169 # собственные значения оператора Гамильтона
170 def eigen_value_intervals(min_energy:float,
171     energy_step:float,
172     max_energy_value:float,
173     max_energy_count:int,
174     x:list[float],
175     step:float,
176     forward_first_approximation:float,
177     # 1 приближение для интегрирования вперёд
178     backward_penultimate_approximation:float,
179     # n - 1 приближения для интегрирования назад
180     connection:int) -> list[float]:
181     level = 0
182     sign = (1 if level % 2 == 0 else -1)
183     energy = min_energy
184     prev_close = is_close_energy(energy,
185         x,
186         step,
187         sign * forward_first_approximation,
188         backward_penultimate_approximation,
189         connection)
190
191     intervals = []
192     intervals.append(energy)
193
194     while(energy < max_energy_value):
195         energy = energy + energy_step
196         sign = (1 if level % 2 == 0 else -1)

```

```

197         close = is_close_energy(energy,
198                                   x,
199                                   step,
200                                   sign * forward_first_approximation,
201                                   backward_penultimate_approximation,
202                                   connection)
203
204         if close * prev_close < 0:
205             prev_close = close
206             intervals.append(energy)
207             intervals.append(energy)
208             level += 1
209
210             if(len(intervals) >= 2 * max_energy_count):
211                 break
212         else:
213             intervals[-1] = energy
214
215     intervals.pop()
216     return intervals
217
218 def integrate(f:list[float], step:float) -> float:
219     size = len(f)
220
221     sum = 0.0
222     for i in range(1, size - 1):
223         sum += f[i]
224     return step * ((f[0] + f[size - 1]) / 2.0 + sum)
225
226 def quantum_normalize(psi:list[float], step:float) -> list[float]:
227     density = list(map(lambda x: x * x, psi))
228     c = integrate(density, step)
229     root = 1.0 / sqrt(c)
230     normalized = list(map(lambda x: root * x, psi))
231     return normalized
232
233 def second_derivation(f:list[float], step:float):
234     num_points = len(f)
235     derivation = [float] * num_points
236
237     coef = 1 / (step * step)
238
239     derivation[0] = coef * (2 * f[0] - 5 * f[1] + 4 * f[2] - f[3])
240     derivation[num_points - 1] = (coef *
241                                   (2 * f[num_points - 1]
242                                    - 5 * f[num_points - 2]
243                                    + 4 * f[num_points - 3]
244                                    - f[num_points - 4]))
245
246     for i in range(1, num_points - 1):

```

```
print("Минимальное значение потенциальной функции =
```

```

297                                     backward_penultimate_approximation,
298                                     connection)
299     print("intervals: ", intervals)
300
301     bisection_method_steps = 100
302     bisection_method_eps = 0.0001
303
304     energies = []
305     level = 0
306     f = lambda e : is_close_energy(e,
307                                     x,
308                                     step,
309                                     forward_sign * forward_first_approximation,
310                                     backward_penultimate_approximation,
311                                     connection)
312     for i in range(max_energy_count):
313         l = intervals[2 * i]
314         r = intervals[2 * i + 1]
315         forward_sign = (1 if level % 2 == 0 else -1)
316         energy = bisection_method(
317             f,
318             l, r,
319             bisection_method_steps,
320             bisection_method_eps)
321         energies.append(energy)
322
323         level += 1
324     print("energies: ", energies, " a.u.")
325     print()
326
327     # значения для построения графиков
328     u = [U(xi) for xi in x]
329
330     q_n0 = compute_q(x, energies[0])
331     forward_n0 = forward_integration(num_intervals,
332                                     forward_first_approximation, q_n0, step)
333     backward_n0 = backward_integration(num_intervals,
334                                       backward_penultimate_approximation, q_n0, step)
335     forward_n0, backward_n0 = normalization(forward_n0, backward_n0,
336                                             connection)
337     forward_n0 = quantum_normalize(forward_n0, step)
338     backward_n0 = quantum_normalize(backward_n0, step)
339
340     def phi(k:int, x:list[float]):
341         num_points = len(x)
342         out = [float] * num_points
343
344         denominator = 1.0 / sqrt(half_width)
345
346         for i in range(num_points):

```

```

347         arg = (pi * (k + 1) * x[i]) / (2.0 * half_width)
348
349         if k % 2:
350             out[i] = denominator * sin(arg)
351         else:
352             out[i] = denominator * cos(arg)
353
354     return out
355
356 def Hphi(k:int, x:list[float]):
357     num_points = len(x)
358     out = [float] * num_points
359
360     phi_k = phi(k, x)
361     derivation = second_derivation(phi_k, step)
362
363     for i in range(num_points):
364         out[i] = -0.5 * derivation[i] + u[i] * phi_k[i]
365
366     return out
367
368 def H(m:int, k:int, x:list[float]):
369     phi_m = phi(m, x)
370     Hphi_k = Hphi(k, x)
371
372     num_points = len(x)
373     mul = [float] * num_points
374     for i in range(num_points):
375         mul[i] = phi_m[i] * Hphi_k[i]
376     return integrate(mul, step)
377
378 def get_Matrix(size:int, x:list[float]):
379     return [[H(m, k, x) for k in range(size)]
380             for m in range(size)]
381
382 size = 6
383 matrix = get_Matrix(size, x)
384 eigenvalues, eigenvectors = LA.eigh(matrix)
385 low = 0
386 for i in range(1, len(eigenvalues)):
387     if eigenvalues[i] < eigenvalues[low]:
388         low = i
389
390 num_points = len(x)
391 wave_func = [0.0] * num_points
392 for k in range(size):
393     phi_k = phi(k, x)
394     for i in range(num_points):
395         wave_func[i] += eigenvectors[k][low] * phi_k[i]
396 wave_func = quantum_normalize(wave_func, step)

```

```

397
398     En = eigenvalues[low]
399     print(f"прямой вариационный метод - E = {En}")
400     print(f"метод пристрелки - E = {energies[0]}")
401
402     plt.axis([begin, end, min_energy, v0 if v0 > 1 else 1])
403     plt.grid(True)
404     plt.axhline(0, color='black')
405     plt.axvline(0, color='black')
406     plt.xlabel("x, a.u.", fontsize = 20, color = "k")
407     plt.plot(x, forward_n0, linewidth = 4, color = "orange",
408             label = "$\phi(x)$, a.u., метод пристрелки")
409     plt.plot(x, wave_func, linewidth = 2, color = "blue" ,
410             label = "$\phi(x)$, a.u., вариационный принцип",
411             linestyle = 'dashed')
412     plt.plot(x, u, linewidth = 6, color = "green" ,
413             label = "$U(x)$, a.u.")
414     plt.legend()
415     plt.show()
416
417     main()

```