МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра вычислительной математики
и прикладных информационных технологий

ЛАБОРАТОРНАЯ РАБОТА №1

ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНОГО

УРАВНЕНИЯ ШРЁДИНГЕРА:

МЕТОД ПРИСТРЕЛКИ

Направление 01.04.02 Прикладная математика и информатика

Профиль Математическое моделирование и вычислительная математика

Зав. кафедрой _____ д. ф.-м.н., пр. А.И. Шашкин __.__.2023

Обучающийся _____ И.Б. Рахимов

Преподаватель _____ д.ф.-м.н., пр. Ю.К. Тимошенко

Воронеж 2023

Содержание

Цели и задачи

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

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

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

1) Используя метод пристрелки, найти:

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

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

2) Вычислить для этих состояний квантовомеханические средние $\langle T \rangle$ и $\langle U \rangle$.

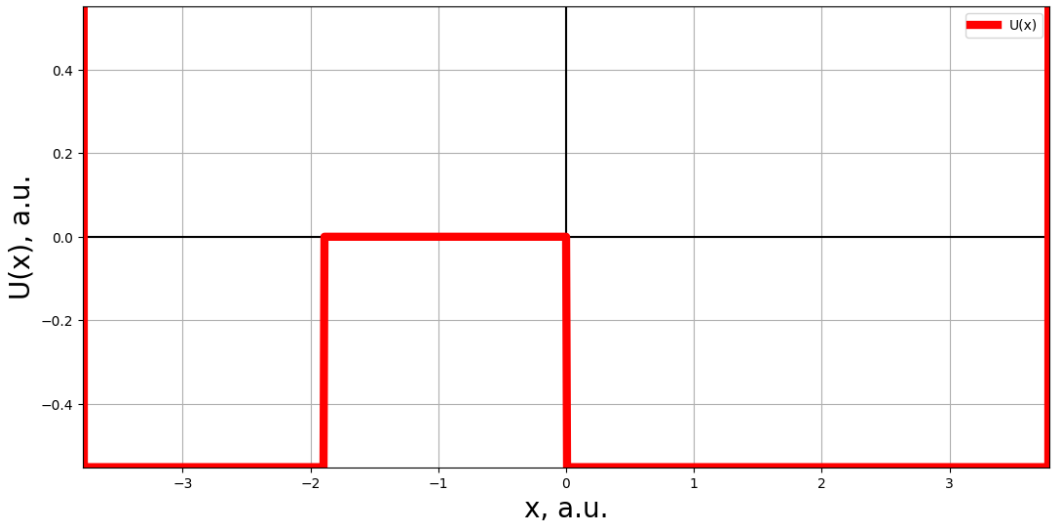

Рис. 1. Потенциальная функция

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

Одномерное стационарное уравнение Шрёдингера:

$$\hat{H}\psi(x) = E\psi(x), \tag{2}$$

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

$$\hat{H} = \hat{T} + U(x), \tag{3}$$

где $\hat{T}$ – оператор кинетической энергии:

$$\hat{T} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2}, \tag{4}$$

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

Подставляя (3) и (4) в (2) получим:

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + U(x)\right)\psi(x) = E\psi(x)$$

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + U(x)\right)\psi(x) - E\psi(x) = 0$$

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + U(x)\psi(x) - E\psi(x) = 0$$

$$\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} - U(x)\psi(x) + E\psi(x) = 0$$

$$\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + \big(E - U(x)\big)\psi(x) = 0 \quad (5)$$

Для упрощения (5) можно использовать атомные единицы Хартри [1], в этой системе приняты в качестве исходных единиц следующие значения: e – абсолютная величина заряда электрона; $\hbar$ – постоянная планка, m_e – масса электрона. То есть в атомных единицах Хартри $e = 1$, $\bar{h} = 1$ и $m_e = 1$. В такой системе (5) примет вид:

$$\frac{1}{2}\frac{d^2\psi(x)}{dx^2} + \big(E - U(x)\big)\psi(x) = 0$$

$$\frac{d^2\psi(x)}{dx^2} + 2\big(E - U(x)\big)\psi(x) = 0 \quad (6)$$

$$q(E,x) = 2\big(E - U(x)\big) \quad (7)$$

$$\frac{d^2\psi(x)}{dx^2} + q(E,x)\psi(x) = 0 \quad (8)$$

В следующем разделе будет найдете нижняя оценка энергетического спектра, которая сыграет важную роль в определении собственных значений E и собственных функций $\psi(x)$ оператора Гамильтона $\widehat{H}$.

1.1. Нижняя оценка энергетического спектра

Оценим нижнюю границу энергетического спектра. Пусть минимальное значение функции $U(x)$ равно U_{min}. Тогда $\langle U\rangle > U_{min}$ и $\langle T\rangle \geq 0$ [1]. Поэтому из уравнения (2) следует:

$$E = \langle H\rangle = \int_{-\infty}^{+\infty} \psi^*(x)\widehat{H}\psi(x)dx = \langle T\rangle + \langle U\rangle > U_{min} \quad (9)$$

То есть энергии всех состояний больше U_{min}.

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

1.2. Свойства потенциальной энергии $\boldsymbol{U(x)}$

Особый практический интерес представляет случай, когда:

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

Для данной $U(x)$ свойства решений уравнения Шредингера зависят от знака собственного значения E. Рассмотрим два случая:

1) $E > 0$. Частица совершает инфинитное движение. Оператор Гамильтона имеет непрерывный спектр собственных значений. Квантовые состояния непрерывного спектра называют несвязными состояниями. Частица, находящаяся в несвязном состоянии, способна уйти на бесконечность.
2) $E < 0$. Частица с отрицательной энергией совершает финитное движений. Оператор Гамильтона имеет дискретный спектр, то есть собственные значения и соответствующие собственные функции можно снабдить номерами, называемыми квантовыми числами. При $E < 0$ уравнение (2) приобретает вид:

$$\widehat{H}\psi_k(x) = E_k\psi_k(x).$$

Квантовое состояние, обладающее наименьшей энергией, называется основным, остальные состояния дискретного спектра называются возбуждёнными состояниями. Частица, находящаяся в связанном состоянии, не способна уйти на бесконечность. То есть плотность вероятности $|\psi_k(x)|^2 \to 0$ при $x \to \pm\infty$, но на всех конечных расстояниях не равно нулю. В силу линейности стационарного уравнения Шрёдингера, волновые функции математически определены с точностью до постоянного множителя. Однако, из физических соображений, волновые функции должны быть нормированы следующим образом:

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

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

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

1.3. Осцилляционная теорема

Упорядоченные собственные значения оператора Гамильтона в порядке возрастания, нумеруя энергию основного состояния индексом «0»: $E_0, E_1, \dots, E_k$. Тогда волновая функция k-го состояния $\psi_k(x)$ будет иметь k узлов (пересечений с осью x). Исключение: области, в которых потенциальная функция бесконечна.

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

2. Метод пристрелки

Численное моделирование будет производится только на основном и низковозбуждённых состояниях электрона для потенциальной функции с бесконечными стенками (рис. 1). В данном отчёте для основного и первого возбуждённого состояния ($n = 0, n = 1$).

Так как за пределами $(-L, L)$ $U(x)$ обращается в ∞ граничные условия для волновой функции будут иметь вид:

$$\psi(-L) = 0 = \psi(L) \tag{10}$$

Так как для собственных значений известна оценка снизу (9). То удобно начинать с вычисления энергии и волновой функции основного состояния. Оценим грубо энергию основного состояния как:

$$U = U_{min} + \varepsilon,$$

где ε малая положительная величина.

Подставим значение этой энергии в уравнение (8). Это уравнение становится обыкновенным дифференциальным уравнением второго порядка с граничными условиями (10).

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

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

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

$$x_n = a + h * i, \qquad n = 0,1, \dots, N$$

Граничные условия (10) приобретают вид:

$$\psi_0 = \psi_N = 0$$

Задачу Коши для дифференциального уравнения (8) можно решить методом Нумерова для интегрирования “вперёд”:

$$\psi_{n+1} = [2(1 - 5cq_n)\psi_n - (1 + cq_{n-1})\psi_{n-1}](1 + cq_{n+1})^{-1} \qquad (11)$$

и “назад’:

$$\psi_{n-1} = [2(1 - 5cq_n)\psi_n - (1 + cq_{n+1})\psi_{n+1}](1 + cq_{n-1})^{-1} \qquad (12)$$

здесь $c = \frac{h^2}{12}, q_n = q(E, x_n)$. Для использования формулы (11) необходимо знать ψ_0 и ψ_1, а для формулы (12) – ψ_{n-1} и ψ_n. Значения ψ_0 и ψ_n известны, а ψ_1 и ψ_{n-1} нет. Однако если N достаточно велико, то можно считать, что $\psi_1 = d_1$, $\psi_{n-1} = d_2$, где d_1, d_2 – малые величины в силу непрерывности волновой функции. Так же в силу Осцилляционной теоремы для нечетных состояний d_1 и d_2 должны быть разных знаков, для чётных одинаковых.

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

$$f(E) = \left.\frac{d\psi_>(x)}{dx}\right|_{x_m} - \left.\frac{d\psi_<(x)}{dx}\right|_{x_m} \qquad (13)$$

Где $\psi_>$, $\psi_<$ – волновые функции, полученные интегрированием “вперёд” и “назад” соответственно, x_m – узел сшивки. Так же перед вычислением необходимо масштабировать волновые функции $\psi_>$ и $\psi_<$ так, чтобы $\psi_>(x_m) = \psi_<(x_m)$, такое масштабирование называется математической нормировкой.

Для расчёта производных можно воспользоваться формулой численного дифференцирования, например:

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

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

$$E^{(0)} = U_{min} + \varepsilon$$

где ε – малая положительная величина и вычислим $f^{(0)}$ по формуле (13). Далее будем увеличивать энергии с шагом ΔE до тех пор, пока величины $f^{(i)}$ на двух соседних шагах i и $i-1$ не будут иметь различные знаки. Если шаг ΔE меньше, чем разность энергий соседних уровней, можно быть уверенным, что искомое собственное значение находится в отрезке $[E^{(i-1)}, E^{(i)}]$. Далее для уточнения собственного значения можно воспользоваться любым методом решения уравнений, например методом бисекций.

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

2.1. Квантовомеханических средние

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

$$\int_a^b f(x)dx \approx h\left(\frac{f(x_0) - f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i)\right),$$

где $h = \frac{b-a}{n}$, $x_i = a + h \cdot i$, $i = 0,1, \dots, n$.

Для этой формулы справедлива оценка погрешности R:

$$|R| \leq \frac{(b-a)^3}{12n^2} M,$$

где $M = \max\limits_{x\in[a,b]} |f(x_0)|$.

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

$$f''(x_0) \approx \frac{2f(x_0) - 5f(x_1) + 4f(x_2) - f(x_3)}{h^2},$$

$$f''(x_i) \approx \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{h^2}, \qquad i = 1,2,\dots,n-1,$$

$$f''(x_n) \approx \frac{2f(x_n) - 5f(x_{n-1}) + 4f(x_{n-2}) - f(x_{n-3})}{h^2}.$$

В следующей главе будет подробно описана программная реализация изложенного выше алгоритма численного решения одномерного стационарного уравнения Шрёдингера для электрона в потенциальном поле (1).

3. Программная реализация

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

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

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

- min_energy – минимальная энергия;
- forward_first_approximation – первая аппроксимация для интегрирования вперёд;
- backward_penultimate_approximation – предпоследняя аппроксимация для интегрирования назад;
- begin – начало отрезка интегрирования;
- end – конец отрезка интегрирования;
- num_intervals – количество интервалов, используемых для решения задачи;
- num_points – количество точек, используемых для решения задачи;
- connection – узел сшивки;
- step – расстояние между ближайшими точками сетки;
- x – сетка, на которую разбивается отрезок $[begin, end]$;
- energy_step – шаг, с которым ищется изменение знака энергии;
- max_energy_value – максимальное возможное значение энергии;

- max_energy_count – максимальное количество собственных значений, которые необходимо найти.

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

- min_energy – минимальное значение энергии
- energy_step – шаг, с которым ищется изменение знака энергии;
- max_energy_value – максимальное возможное значение энергии;
- max_energy_count – максимальное количество собственных значений, которые необходимо найти;
- x – сетка, на которую разбивается отрезок $[begin, end]$;
- step – расстояние между ближайшими точками сетки;
- forward_first_approximation – первая аппроксимация для интегрирования вперёд;
- backward_penultimate_approximation – предпоследняя аппроксимация для интегрирования назад;
- connection – узел сшивки;

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

- is_close_energy – вычисляет насколько близко находятся функции для интегрирования “вперёд” и “назад” для данного значения энергии 150-166;
- compute_q – фукнция для вычисления (7) 60-62;
- forward_integration, backward_integration – функции для интегрирования “вперёд” и “назад” соответственно, для заданного значения энергии 76-120;
- normalization – функция для достижения равенства в узле сшиви 126-137;

- is_close – функция для вычисления “близости” функция полученных с помощью интегрирования “вперёд” и “назад” 140-145;
- derivative – функция для аппроксимации первой производной 64-73.

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

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

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

4. Результаты численных экспериментов и их анализ

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

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

1) для основного состояния:

$$n = 0$$

$$E_0 = 0.3404957040724568 \text{ а. е.}$$

$$\langle U \rangle_> = -0.4946963847555298$$

$$\langle U \rangle_< = -0.4946946871802064$$

$$\langle T \rangle_> = 0.15420018816513176$$

$$\langle T \rangle_< = 0.15419849059093088$$

$$R_> = E_0 - (\langle T \rangle_> + \langle U \rangle_>) = 4.925179412640368e - 07$$

$$R_< = E_0 - (\langle T \rangle_< + \langle U \rangle_<) = \ 4.925168187730478e - 07$$

2) для первого возбужденного состояния:

$$n = 1$$

$$E_1 = -0.02222421969745659 \text{ а. е.}$$

$$\langle U \rangle_> = -0.3587456231348717$$

$$\langle U \rangle_< = -0.35873986656146833$$

$$\langle T \rangle_> = 0.33651966855729465$$

$$\langle T \rangle_< = 0.33651391200385367$$

$$R_> = E_1 - (\langle T \rangle_> + \langle U \rangle_>) = 1.734880120445037e - 06$$

$$R_< = E_1 - (\langle T \rangle_< + \langle U \rangle_<) = 1.7348601580799428e - 06$$

где E_i – энергия $i-$го состояния, $\langle T \rangle$ и $\langle U \rangle$ – квантовомеханические средние кинетической и потенциальной энергии соответственно, R – погрешность вычисления квантовомеханическсих средних. Нижний индекс означает какая

волновая функция использовалась для вычисления квантовомеханическсих средних и их погрешностей $>$ – волновая функция была получена интегрирование “вперёд”, $<$ – “назад”.

Видно, что квантовомеханическсие средние вычислены с шестым порядком точности, дополнительно можно проверить результаты вычислений зная что $\langle T \rangle > 0$ и $\langle U \rangle > U_{min}$. В данной задаче $U_{min} = -0.5512476571974569\ a.e.$

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

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

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

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

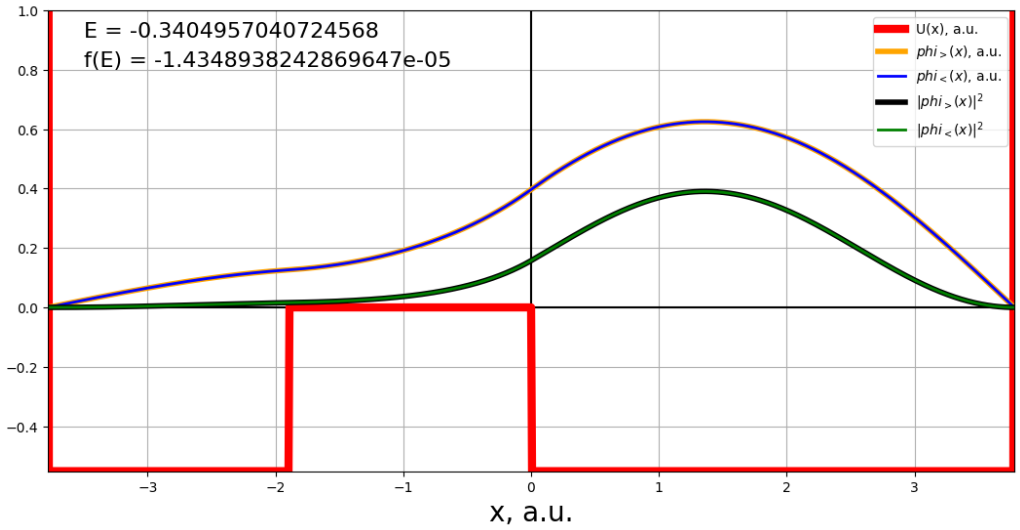

Рис. 2. Основное состояние

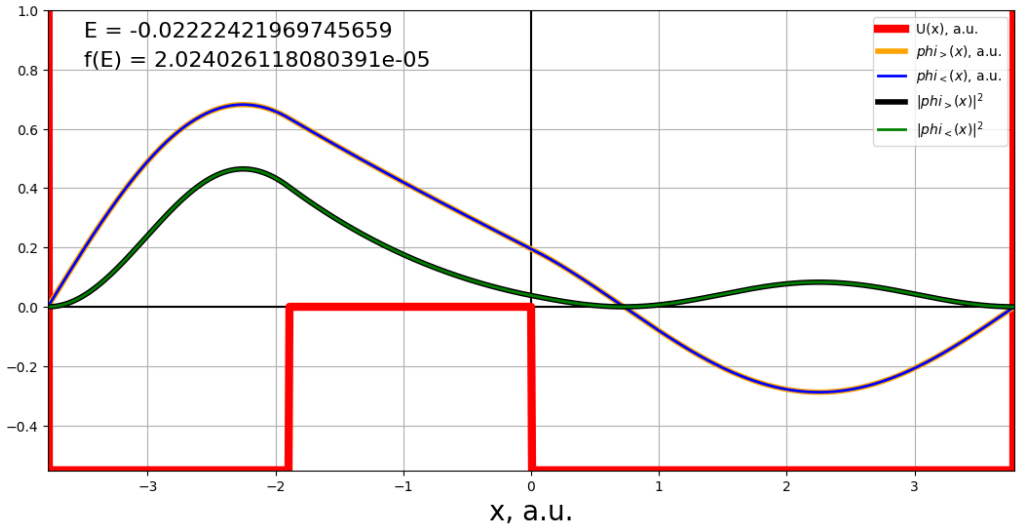

Рис. 3. Первое возбуждённое состояние

Список использованных источников

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

Приложения

Приложение 1 Компьютерный код

```
1
2   import matplotlib.pyplot as plt
3   from math import sqrt
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
14                                  # шагов 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 # фукнция 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
148 # вперёд и назад 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,
155                        # первое приближение для интегрирования вперёд
156              backward_penultimate_approximation:float,
157                          # n - 1 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 # возвращет интервалы в которых
169 # находятся 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)
```

```
        close = is_close_energy(energy,
                                x,
                                step,
                                sign * forward_first_approximation,
                                backward_penultimate_approximation,
                                connection)

        if close * prev_close < 0:
            prev_close = close
            intervals.append(energy)
            intervals.append(energy)
            level += 1

            if(len(intervals) >= 2 * max_energy_count):
                break
        else:
            intervals[-1] = energy

    intervals.pop()
    return intervals

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

    sum = 0.0
    for i in range(1, size - 1):
        sum += f[i]
    return step * ((f[0] + f[size - 1]) / 2.0 + sum)

def quantum_normalize(psi:list[float], step:float) -> list[float]:
    density = list(map(lambda x: x * x, psi))
    c = integrate(density, step)
    root = 1.0 / sqrt(c)
    normalized = list(map(lambda x: root * x, psi))
    return normalized

def second_derivation(f:list[float], step:float):
    num_points = len(f)
    derivation = [float] * num_points

    coef = 1 / (step * step)

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

    for i in range(1, num_points - 1):
```

```
247             derivation[i] = coef * (f[i - 1] - 2 * f[i] + f[i + 1])
248
249         return derivation
250
251     w = 10.0
252
253     v0 = 15 # Электронвольт
254     half_width = 2.0 # Ангстрем
255
256     print(f"v0 = {v0} Электронвольт")
257     print(f"l = {half_width} Ангстрем")
258
259     # Перевод величин в атомную систему единиц
260     v0 = v0 / 27.211 # атомных единиц
261     half_width = half_width / 0.5292 # бор
262
263     print(f"v0 = {v0} атомных единиц")
264     print(f"l = {half_width} бор")
265
266     def main():
267         min_energy = -v0
268         print("Минимальное     значение     потенциальной     функции     =
269     {}".format(min_energy))
270         print()
271
272         forward_first_approximation = 1.e-9
273         backward_penultimate_approximation = 1.e-9
274
275         begin = -half_width
276         end = +half_width
277
278         num_intervals = 1000
279         num_points = num_intervals + 1
280
281         connection = 300
282
283         step = (end - begin) / num_intervals
284         x = [begin + step * i for i in range(num_points)]
285
286         energy_step = 0.01
287         max_energy_value = 1000
288         max_energy_count = 2
289
290         intervals = eigen_value_intervals(min_energy,
291                                           energy_step,
292                                           max_energy_value,
293                                           max_energy_count,
294                                           x,
295                                           step,
296                                           forward_first_approximation,
```

```
                                    backward_penultimate_approximation,
                                       connection)
    print("intervals: ", intervals)

    bisection_method_steps = 100
    bisection_method_eps = 0.0001

    energies = []
    level = 0
    f = lambda e : is_close_energy(e,
                                    x,
                                    step,
                    forward_sign * forward_first_approximation,
                                    backward_penultimate_approximation,
                                    connection)
    for i in range(max_energy_count):
        l = intervals[2 * i]
        r = intervals[2 * i + 1]
        forward_sign = (1 if level % 2 == 0 else -1)
        energy = bisection_method(
            f,
            l, r,
            bisection_method_steps,
            bisection_method_eps)
        energies.append(energy)

        level += 1
    print("energies: ", energies, " a.u.")
    print()

    # значения для построения графиков
    u = [U(xi) for xi in x]

    q_n0 = compute_q(x, energies[0])
    forward_n0 = forward_integration(num_intervals,
                                forward_first_approximation, q_n0, step)
    backward_n0 = backward_integration(num_intervals,
                        backward_penultimate_approximation, q_n0, step)
    forward_n0, backward_n0 = normalization(forward_n0,
                                             backward_n0, connection)
    forward_n0 = quantum_normalize(forward_n0, step)
    backward_n0 = quantum_normalize(backward_n0, step)
    forward_probability_n0 = list(map(lambda x: x ** 2, forward_n0))
    backward_probability_n0 = list(map(lambda x: x ** 2, backward_n0))

    q_n1 = compute_q(x, energies[1])
    forward_n1 = forward_integration(num_intervals,
                                forward_first_approximation, q_n1, step)
    backward_n1 = backward_integration(num_intervals,
                        backward_penultimate_approximation, q_n1, step)
```

```
347       forward_n1, backward_n1 = normalization(forward_n1,
348                                               backward_n1, connection)
349       forward_n1 = quantum_normalize(forward_n1, step)
350       backward_n1 = quantum_normalize(backward_n1, step)
351       forward_probability_n1 = list(map(lambda x: x ** 2, forward_n1))
352       backward_probability_n1 = list(map(lambda x: x ** 2, backward_n1))
353
354       # Графики
355
356       # U(x)
357       plt.axis([begin, end, min_energy, v0])
358       plt.grid(True)
359       plt.axhline(0, color='black')
360       plt.axvline(0, color='black')
361       plt.xlabel("x, B", fontsize = 20, color = "k")
362       plt.ylabel("U(x), a.u.", fontsize = 20, color = "k")
363       plt.plot(x, u, linewidth = 6, color = "red", label = "U(x)")
364       plt.legend()
365       plt.savefig("U(x).png")
366       plt.show()
367
368       # n0
369       plt.axis([begin, end, min_energy, v0 if v0 > 1 else 1])
370       plt.grid(True)
371       plt.axhline(0, color='black')
372       plt.axvline(0, color='black')
373       plt.xlabel("x, B", fontsize = 20, color = "k")
374       plt.plot(x, u, linewidth = 6, color = "red" ,
375                label = "U(x), a.u.")
376       plt.plot(x, forward_n0, linewidth = 4, color = "orange",
377                label = "$phi_>(x)$, a.u.")
378       plt.plot(x, backward_n0, linewidth = 2, color = "blue",
379                label = "$phi_<(x)$, a.u.")
380       plt.plot(x, forward_probability_n0, linewidth = 4,
381                color = "black", label = "$|phi_>(x)|^2$")
382       plt.plot(x, backward_probability_n0, linewidth = 2,
383                color = "green" , label = "$|phi_<(x)|^2$")
384       plt.legend()
385       plt.text(-3.5, 0.91, "E = " + str(energies[0]),
386                fontsize = 16, color = 'black')
387       plt.text(-3.5, 0.81, "f(E) = " + str(f(energies[0])),
388                fontsize = 16, color = 'black')
389       plt.savefig("n0.png")
390       plt.show()
391
392       # n1
393       plt.axis([begin, end, min_energy, v0 if v0 > 1 else 1])
394       plt.grid(True)
395       plt.axhline(0, color='black')
396       plt.axvline(0, color='black')
```

```
397       plt.xlabel("x, B", fontsize = 20, color = "k")
398       plt.plot(x, u, linewidth = 6, color = "red", label = "U(x), a.u.")
399       plt.plot(x, forward_n1, linewidth = 4, color = "orange",
400                label = "$phi_>(x)$, a.u.")
401       plt.plot(x, backward_n1, linewidth = 2, color = "blue",
402                label = "$phi_<(x)$, a.u.")
403       plt.plot(x, forward_probability_n1, linewidth = 4, color = "black"
404                label = "$|phi_>(x)|^2$")
405       plt.plot(x, backward_probability_n1, linewidth = 2,
406                color = "green" , label = "$|phi_<(x)|^2$")
407       plt.legend()
408       plt.text(-3.5, 0.91, "E = " + str(energies[1]), fontsize = 16,
409                color = 'black')
410       plt.text(-3.5, 0.81, "f(E) = " + str(f(energies[1])),
411                fontsize = 16, color = 'black')
412       plt.savefig("n1.png")
413       plt.show()
414
415       # квантово механические средние
416       forward_U_n0 = [forward_n0[i] * u[i] * forward_n0[i]
417                       for i in range(num_points)]
418       backward_U_n0 = [backward_n0[i] * u[i] * backward_n0[i]
419                        for i in range(num_points)]
420       U_n0_forward = integrate(forward_U_n0, step)
421       U_n0_backward = integrate(backward_U_n0, step)
422       print("<U n0> forward =", U_n0_forward)
423       print("<U n0> backward =", U_n0_backward)
424
425       forward_T_n0 = second_derivation(forward_n0, step)
426       backward_T_n0 = second_derivation(backward_n0, step)
427       forward_T_n0 = [-0.5 * forward_n0[i] * forward_T_n0[i]
428                       for i in range(num_points)]
429       backward_T_n0 = [-0.5 * backward_n0[i] * backward_T_n0[i]
430                        for i in range(num_points)]
431       T_n0_forward = integrate(forward_T_n0, step)
432       T_n0_backward = integrate(backward_T_n0, step)
433       print("<T n0> forward =", T_n0_forward)
434       print("<T n0> backward =", T_n0_backward)
435
436       print()
437
438       forward_U_n1 = [forward_n1[i] * u[i] * forward_n1[i]
439                       for i in range(num_points)]
440       backward_U_n1 = [backward_n1[i] * u[i] * backward_n1[i]
441                        for i in range(num_points)]
442       U_n1_forward = integrate(forward_U_n1, step)
443       U_n1_backward = integrate(backward_U_n1, step)
444       print("<U n1> forward =", U_n1_forward)
445       print("<U n1> backward =", U_n1_backward)
446
```

```
447       forward_T_n1 = second_derivation(forward_n1, step)
448       backward_T_n1 = second_derivation(backward_n1, step)
449       forward_T_n1 = [-0.5 * forward_n1[i] * forward_T_n1[i]
450                       for i in range(num_points)]
451       backward_T_n1 = [-0.5 * backward_n1[i] * backward_T_n1[i]
452                        for i in range(num_points)]
453       T_n1_forward = integrate(forward_T_n1, step)
454       T_n1_backward = integrate(backward_T_n1, step)
455       print("<T n1> forward =", T_n1_forward)
456       print("<T n1> backward =", T_n1_backward)
457
458       print()
459
460       print("E[0] - (<T> + <U>) forward =",
461             energies[0] - (T_n0_forward + U_n0_forward))
462       print("E[0] - (<T> + <U>) backward =",
463             energies[0] - (T_n0_backward + U_n0_backward))
464
465       print("E[1] - (<T> + <U>) forward =",
466             energies[1] - (T_n1_forward + U_n1_forward))
467       print("E[1] - (<T> + <U>) backward =",
468             energies[1] - (T_n1_backward + U_n1_backward))
469
470   main()
```