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

Факультет прикладной математики, информатики и механики
Кафедра математического и прикладного анализа

ЛАБОРАТОРНАЯ РАБОТА №2

ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНОГО УРАВНЕНИЯ ШРЁДИНГЕРА: ТЕОРИЯ ВОЗМУЩЕНИЙ

Направление: 01.04.02 - Прикладная математика и информатика
Профиль: Компьютерные технологии в задачах математической физики, оптимизации и управления

Выполнил:
студент 11 группы 2 курса магистратуры
Маркин Р.О.
Преподаватель:
доктор физ.-мат. наук, профессор
Тимошенко Ю.К.

Воронеж 2024

Содержание

1. Цели и задачи работы

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

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 возмущений, найти энергию, нормированную волновую функцию для основного и 2-го возбужденного состояний и плотности вероятности. Энергию вычислять с учетом поправок до второго порядка включительно, а волновую функцию с учетом поправок первого порядка. Возмущенную систему смоделировать, создав в потенциальной функции U(x) пик произвольной формы. 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 $\langle p(x^2) \rangle$.

3) Сравнить результаты с данными, полученными методом пристрелки для возмущенной системы.

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

Одномерное стационарное уравнение Шрёдингера[1]

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

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, 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 всем пространстве. Непрерывность волновой функции и её 1-й 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, 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

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

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

Особый практический интерес представляет случай, когда

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

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 [1]:

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

Квантовое состояние, обладающее наименьшей энергией, называется основным. Остальные состояния называют возбуждёнными 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

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

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. Теория возмущений. Алгоритм

К числу приближенных методов вычисления собственных значений и собственных функций оператора Гамильтона относится метод стационарных возмущений Релея–Шрёдингера[3], который мы далее будем просто называть «методом или теорией возмущений».

В рамках этого теоретического подхода предполагается, что оператор Гамильтона, чьи собственные значения и собственные функции требуется определить, может быть представлен в виде:

$$\hat{H} = \hat{H}^0 + \hat{V}, \tag{8}$$

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

Оператором возмущения может быть либо часть гамильтониана, которая не учитывалась в идеализированной задаче, либо потенциальная энергия, связанная с наличием внешнего воздействия.

Идеализированную систему, которую описывает гамильтониан $\hat{H}^0$, называют «невозмущенной» системой, а систему с гамильтонианом $\hat{H}$ — «возмущенной» системой. В рамках теории возмущений удается получить формулы, определяющие энергии и волновые функции стационарных состояний через известные значения энергий $E_n^{(0)}$ и волновых функций $\Psi_n^{(0)}$ невозмущенной системы.

Стационарные уравнения Шрёдингера для невозмущенной и возмущенной систем [3] имеют вид:

$$\hat{H}^0\Psi_n^{(0)}(x) = E_n^{(0)}\Psi_n^{(0)}(x); \tag{9}$$

$$\hat{H}\Psi_n(x) = E_n\Psi_n(x). \tag{10}$$

В теории возмущений решения уравнения (10) ищутся в виде рядов:

$$\begin{aligned} E_n &= E_n^{(0)} + E_n^{(1)} + E_n^{(2)} + \cdots = \sum_{k=0}^{\infty} E_n^{(k)}, \\ \Psi_n(x) &= \Psi_n^{(0)}(x) + \Psi_n^{(1)}(x) + \Psi_n^{(2)}(x) + \cdots = \sum_{k=0}^{\infty} \Psi_n^{(k)}(x), \end{aligned} \tag{11}$$

где $E_n^{(k)}$, $\Psi_n^{(k)}(x)$ — величины k-го порядка малости по возмущению $\hat{V}$, называемые k-ми поправками или поправками k-го порядка. Первые слагаемые рядов (11) определяются следующими формулами:

$$\begin{aligned} E_n^{(1)} &= V_{nn}, \\ E_n^{(2)} &= \sum_m{}' \frac{|V_{nm}|^2}{E_n^{(0)} - E_m^{(0)}}, \\ \Psi_n^{(1)}(x) &= \sum_m{}' \frac{V_{mn}}{E_n^{(0)} - E_m^{(0)}} \Psi_m^{(0)}(x), \end{aligned} \tag{12}$$

где

$$V_{mn} \equiv \langle m|V|n\rangle = \int_\Omega \Psi_m^{(0)*}(x) \hat{V} \Psi_n^{(0)}(x) dx. \tag{13}$$

Штрих над знаком суммы означает пропуск слагаемого с $m = n$: $\sum'_m \equiv \sum_{m \neq n}$. Очевидно, что ряды в (12) сходятся, если выполняется неравенство

$$|V_{nm}| \ll |E_n^{(0)} - E_m^{(0)}|. \tag{14}$$

Во многих случаях для решения задачи достаточно ограничиться вычислением энергии с учетом поправок до второго порядка включительно и волновой функции с учетом поправок первого порядка [3]:

$$\begin{aligned} E_n &= E_n^{(0)} + V_{nn} + \sum_m{}' \frac{|V_{nm}|^2}{E_n^{(0)} - E_m^{(0)}}, \\ \Psi_n(x) &= \Psi_n^{(0)}(x) + \sum_m{}' \frac{V_{mn}}{E_n^{(0)} - E_m^{(0)}} \Psi_m^{(0)}(x). \end{aligned} \tag{15}$$

При вычислении второй поправки к энергии и первой поправки к волновой функции основного состояния возмущенной системы бесконечные суммы в (12) заменяются конечными. Для оценки корректного численного

значения надо выполнить несколько расчетов соответствующей суммы для монотонно возрастающих величин. Если увеличение верхнего предела суммы, начиная с некоторого значения, не приводит к заметным изменениям суммы, то задача оценки значения верхнего значения суммы решена.

4. Программная реализация алгоритма

В Приложении представлена программа на языке Python 3.0 [2], реализованная в среде разработки PyCharm Community Edition 2024.2.1, численного решения одномерного стационарного уравнения Шредингера для электрона в одномерной потенциальной яме. Программа реализует алгоритм теории возмущений, позволяющий находить собственные значения и соответствующие им волновые функции. Потенциальная функция (невозмущенная система) и параметры для нее соответствуют постановке задачи из первой главы. Энергия и длина ямы были переведены в атомные единицы Хартри (строки 105-108)

В строках 5-121 реализован алгоритм пристрелки, который подробно разобран в лабораторной работе №1, в данной программе этот алгоритм используется для реализации невозмущенной системы и вычисления ее решения(собственные значения и собственные функции оператора Гамильтона). Собственные значения и собственные функции невозмущенной системы будут использоваться для вычисления решения возмущенной системы. Путем компьютерного моделирования был вычислен верхний предел сумм (12). В программе за верхний предел сумм отвечает kmax в строке 199, он равне количеству вычисленных энергий невозмущенной системы, для данного варианта задачи было достаточно 14.

В строках 125-132 реализована потенциальная функция возмущенной системы, для этого был создан пик который больше max(U(x) на отрезке [2.5; 3.0]

В строках 134-135 реализован оператор возмущения.

В строках 137-141 и 143-146 реализованы функции возращающие энергии и волновые функции невозмущенной системы.

В строках 149-152 и 155-158 реализованы функции которые вычисляют матричный элемент оператора возмущения по невозмущенным системам (13).

В строках 162-175 реализована функция вычисляющая поправку второго порядка (12).

В строках 177-183 и 185-192 реализованы функции вычисляющие поправку первого порядка для волновой функции возмущенной системы.

В строках 195-196 реализована функция вычисляющая первое приближение волновой функции (15)

В строках 198-291 реализована функция с одним параметром root который определяет номер состояния возмущенной системы для которого требуется вычислить энергию и волновую функцию. В функции вычисляется энергия с учетом поправок до второго порядка включительно, и волновая функция с учетом поправок первого порядка. Также в функции реализован вывод графиков и запись данных в файл.

В строках 293-294 вызывается функция result для основного и второго возбужденного состояний возмущенной системы.

В строках 296-310 выводятся графики невозмущенной и возмущенной систем.

5. Результаты численных экспериментов и их анализ

Невозмущенная система из постановки задачи представлена на Рис. 1.

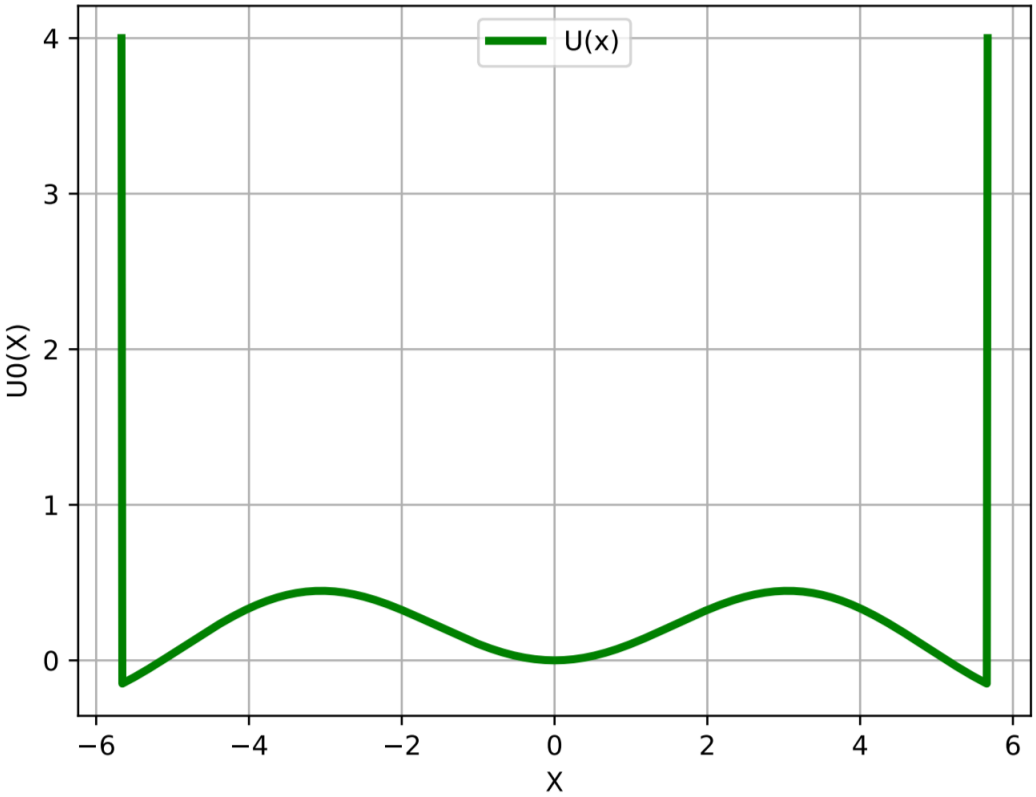

Рис. 1. Невозмущенная система

Возмущенная система представлена на Рис. 2.

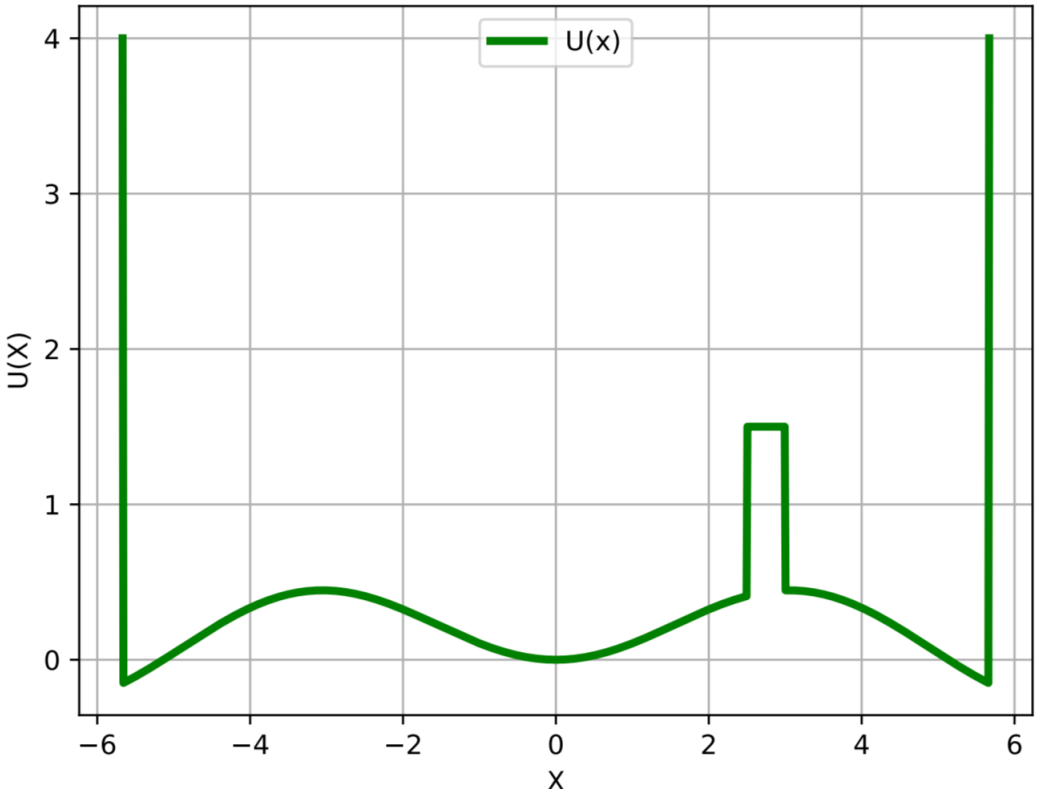

Рис. 2. Возмущенная система

При вычислении энергии с учетом поправок для основного состояния возмущенной системы были получены следующие значения на Рис. 3:

```
e0(1)= 0.2004808959960938
e= 0.21676094417656988    (1-approximation)
k= 1   s= -0.004500358344230809
k= 2   s= -0.006688825860448711
k= 3   s= -0.006690133200429368
k= 4   s= -0.007687705183441375
k= 5   s= -0.008791550330884907
k= 6   s= -0.009063823093435936
k= 7   s= -0.009085225116504886
k= 8   s= -0.009404571893642735
k= 9   s= -0.009729642770751477
k= 10   s= -0.00979269384940016
k= 11   s= -0.009814425800433324
k= 12   s= -0.009963430265648481
k= 13   s= -0.010092497764022854
k= 14   s= -0.01010933462794701
e= 0.20665160954862288    (2-approximation)
```

Рис. 3. Энергия с учетом поправок первого и второго порядка

Где e0(1) это энергия основного состояния невозмущенной системы. e это энергия возмущенной системы вычисленная с учетом поправок. s это сумма из формулы второй поправки (12) которая вычислялась поочередно с каждым состоянием невозмущенной системы.

На Рис. 4 представлена волновая функция и плотность вероятности возмущенной системы для основного состояния.

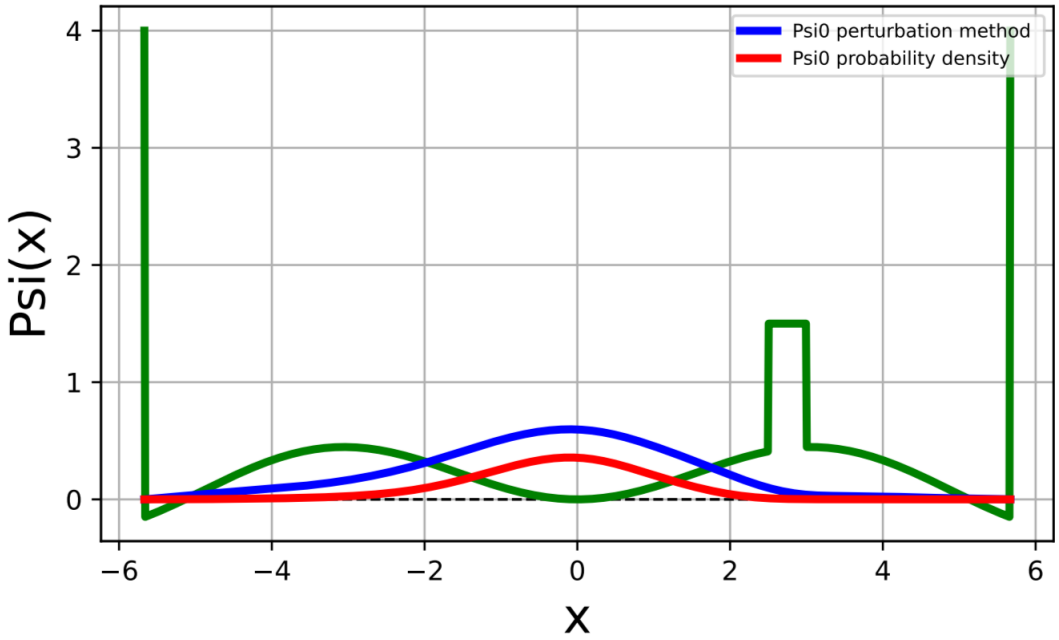

Рис. 4. Волновая функция и плотность вероятности основного состояния

На графиках можно видеть, что функции согласно осциляционной теореме соответствуют основному состоянию так как нету пересечений с осью абцисс, кроме бесконечных потенциальных стенок которые не учитываются.

На Рис. 5 представлено сравнение волновых функций основного состояния возмущенной системы вычисленных методом пристрелки и методом возмущений.

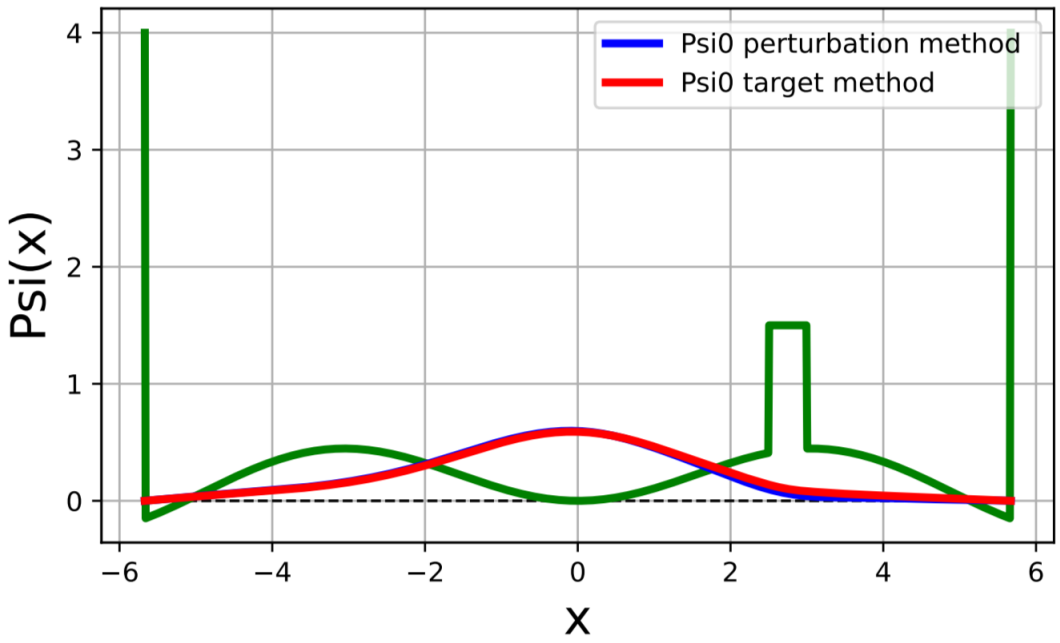

Рис. 5. Волновая функция основного состояния возмущенной системы

На Рис. 6 представлены значения энергий основного состояния и квантовомеханических средних $\langle p(x) \rangle$ и $\langle p(x^2) \rangle$ вычисленные методом пристрелки и методом возмущений

```
E_perturbation = 0.206652, <p_x> = 0.000000e+00, <p_x^2> = 2.001141e-01
E_target = 0.210341, <p_x> = 0.000000e+00, <p_x^2> = 1.812072e-01
```

Рис. 6. Энергия и квантовомеханические средние $\langle p(x) \rangle$ и $\langle p(x^2) \rangle$ для основного состояния

Здесь E perturbation это энергия вычисленная методом возмущений, E target это энергия вычисленная методом пристрелки.

При вычислении энергии с учетом поправок для второго возбужденного состояния возмущенной системы были получены следующие значения на Рис. 7:

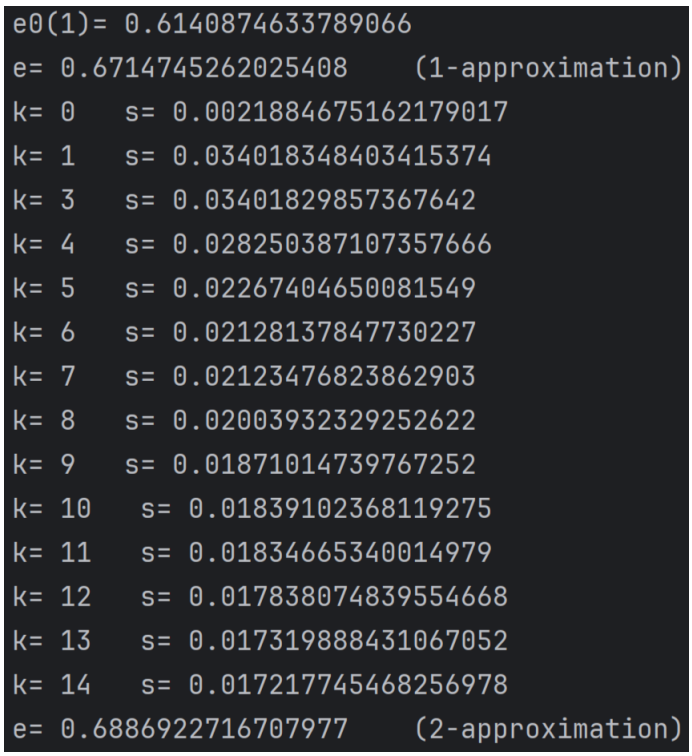

Рис. 7. Энергия с учетом поправок первого и второго порядка

Где e0(1) это энергия второго возбужденного состояния невозмущенной системы. e это энергия возмущенной системы вычисленная с учетом поправок. s это сумма из формулы второй поправки (12) которая вычислялась поочередно с каждым состоянием невозмущенной системы.

На Рис. 8 представлена волновая функция и плотность вероятности возмущенной системы для второго возбужденного состояния.

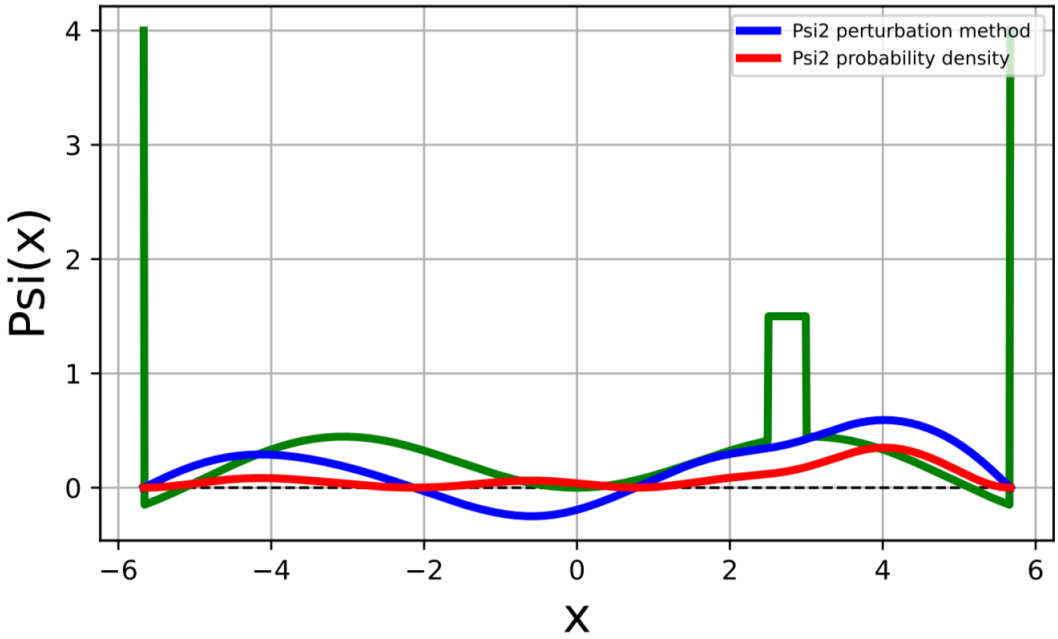

Рис. 8. Волновая функция и плотность вероятности второго возбужденного состояния

На графиках можно видеть, что функции согласно осциляционной теореме соответствуют второму возбужденному состоянию так как есть два пересечения с осью абцисс, кроме бесконечных потенциальных стенок которые не учитываются.

На Рис. 9 представлено сравнение волновых функций второго возбужденного состояния возмущенной системы вычисленных методом пристрелки и методом возмущений.

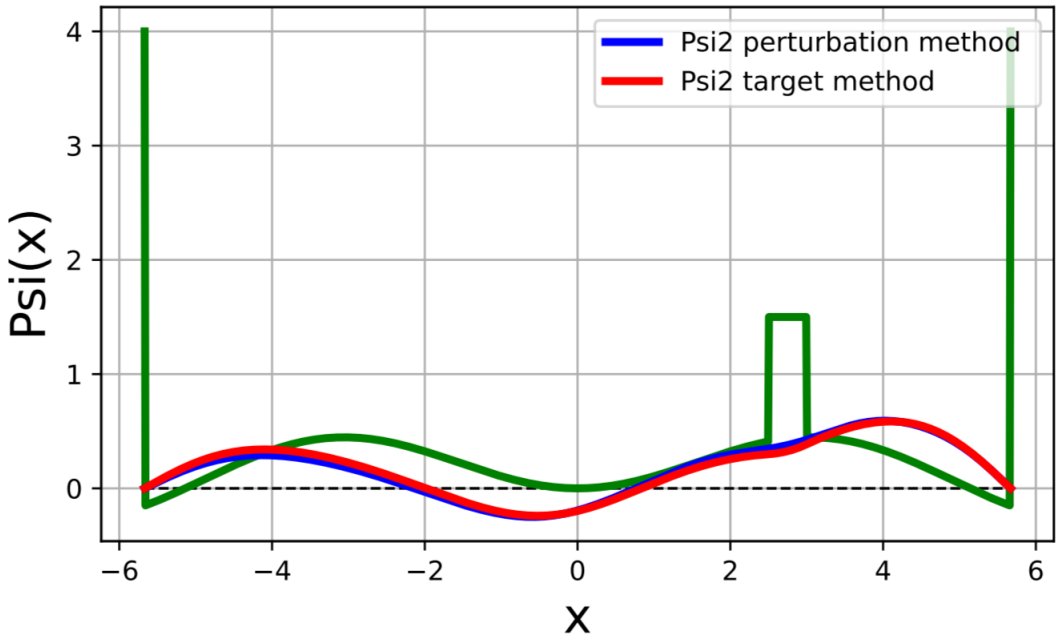

Рис. 9. Волновая функция второго возбужденного состояния возмущенной системы

На Рис. 10 представлены значения энергий второго возбужденного состояния и квантовомеханических средних $\langle p(x) \rangle$ и $\langle p(x^2) \rangle$ вычисленные методом пристрелки и методом возмущений

```
E_perturbation = 0.688692, <p_x> = 0.000000e+00, <p_x^2> = 6.323043e-01
E_target = 0.676564, <p_x> = 0.000000e+00, <p_x^2> = 6.751142e-01
```

Рис. 10. Энергия и квантовомеханические средние $\langle p(x) \rangle$ и $\langle p(x^2) \rangle$ для второго возбужденного состояния

Здесь E perturbation это энергия вычисленная методом возмущений, E target это энергия вычисленная методом пристрелки.

Заключение.

Таким образом, было получено численное решение для задачи о частице в одномерной квантовой яме с бесконечными стенками при помощи метода возмущений. Были получены значения энергий и волновые функции основного и второго возбужденного состояний, а также для каждого состояния были вычисленны квантовомеханические средние $\langle p(x) \rangle$, $\langle p(x^2) \rangle$. Полученные волновые функции соответствуют осциляционной теореме. Сравнивая результаты полученные методом возмущений с методом пристрелки можно сказать, что метод возмущений менее точен.

Приложение 1. Компьютерный код

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import jn

def U0(x):
    if abs(x) < L:
        return V0 * jn(2, x)
    else:
        return 4.0


def q(e, x, potential_func):
    return 2.0 * (e - potential_func(x))

def deriv(Y, h, m):
    return (Y[m - 2] - Y[m + 2] + 8.0 * (Y[m + 1] - Y[m - 1])) / (12.0 *
        h)

def normalize_wavefunction(Y):
    norm = np.sqrt(np.trapz(Y**2, X))
    return Y / norm

def mean_momentum(Psi, X):
    hbar = 1.0
    dPsi_dx = np.gradient(Psi, X)
    integrand = Psi.conj() * dPsi_dx
    mean_Px = -1j * hbar * np.trapz(integrand, X)
    return mean_Px.real

def mean_square_momentum(Psi, X):
    hbar = 1.0
    d2Psi_dx2 = np.gradient(np.gradient(Psi, X), X)
    integrand = Psi.conj() * d2Psi_dx2
    mean_Px2 = -hbar**2 * np.trapz(integrand, X)
    return mean_Px2.real

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

```
42              Fi[n - 2] = d2
43
44
45              for i in np.arange(1, n - 1, 1):
46                  p1 = 2.0 * (1.0 - 5.0 * F[i]) * Psi[i]
47                  p2 = (1.0 + F[i - 1]) * Psi[i - 1]
48                  Psi[i + 1] = (p1 - p2) / (1.0 + F[i + 1])
49
50
51              for i in np.arange(n - 2, 0, -1):
52                  f1 = 2.0 * (1.0 - 5.0 * F[i]) * Fi[i]
53                  f2 = (1.0 + F[i + 1]) * Fi[i + 1]
54                  Fi[i - 1] = (f1 - f2) / (1.0 + F[i - 1])
55
56              p1 = np.abs(Psi).max()
57              p2 = np.abs(Psi).min()
58              big = p1 if p1 > p2 else p2
59
60              Psi[:] = Psi[:] / big
61              coef = Psi[r] / Fi[r]
62              Fi[:] = coef * Fi[:]
63
64              return deriv(Psi, h, r) - deriv(Fi, h, r)
65
66          def energy_scan(E_min, E_max, step, potential_func):
67              energies = []
68              values = []
69              E = E_min
70              while E <= E_max:
71                  f_value = f_fun(E, n, potential_func)
72                  energies.append(E)
73                  values.append(f_value)
74                  E += step
75              return energies, values
76
77          def find_exact_energies(E_min, E_max, step, tol, potential_func):
78              energies, values = energy_scan(E_min, E_max, step, potential_func)
79              exact_energies = []
80              for i in range(1, len(values)):
81                  Log1 = values[i] * values[i - 1] < 0.0
82                  Log2 = np.abs(values[i] - values[i - 1]) < porog
83                  if Log1 and Log2:
84                      E1, E2 = energies[i - 1], energies[i]
85                      exact_energy = bisection_method(E1, E2, tol, potential_func)
86                      f_fun(exact_energy, n, potential_func)
```

```
                    exact_energies.append(exact_energy)
            return exact_energies

        def bisection_method(E1, E2, tol, potential_func):
            while abs(E2 - E1) > tol:
                Emid = (E1 + E2) / 2.0
                f1, f2, fmid = f_fun(E1, n, potential_func), f_fun(E2, n,
                    potential_func), f_fun(Emid, n, potential_func)
                if f1 * fmid < 0.0:
                    E2 = Emid
                else:
                    E1 = Emid
                if f2 * fmid < 0.0:
                    E1 = Emid
                else:
                    E2 = Emid
            return (E1 + E2) / 2.0


        cenergy = 27.212
        clength = 0.5292
        V0 = 25.0 / cenergy
        L = 3.0 / clength
        A, B = -L, L
        n = 1001
        h = (B - A) / (n - 1)
        c, W = h ** 2 / 12.0, 3.0
        Psi, Fi, X = np.zeros(n), np.zeros(n), np.linspace(A, B, n)
        r = (n-1)//2 - 15
        porog = 4.0

        d1, d2 = 1.e-09, 1.e-09
        tol = 1e-6

        E_min, E_max, step = -0.1, 9.0, 0.01
        exact_energies = find_exact_energies(E_min + 0.001, E_max, step, tol, U0
            )

        file1 = open(f"result.txt", "a")

        def U(x):
            if abs(x) < L:
                if 2.5 <= x <= 3.0:
                    return 1.5
                else:
```

```
                return V0 * jn(2, x)
            else:
                return 4.0

        def V(x):
            return U(x) - U0(x)

        def e0(k):
            if k < len(exact_energies):
                return exact_energies[k]
            else:
                raise ValueError(f"There is no level with the {k} number")

        def psi_interp(k, x):
            f_fun(exact_energies[k], n, U0)
            Psi_copy = normalize_wavefunction(Psi.copy())
            return np.interp(x, X, Psi_copy)


        def funct(x, k1, k2):
            psi_k1 = psi_interp(k1, x)
            psi_k2 = psi_interp(k2, x)
            return psi_k1 * V(x) * psi_k2


        def matel(k1, k2):
            integrand_values = np.array([funct(x, k1, k2) for x in X])
            res = np.trapz(integrand_values, X)
            return res if abs(res) > 1e-14 else 0.0



        def e_corr_2(kmax, root):
            s = 0.0
            for k in range(0, root):
                s += matel(root, k)**2 / (e0(root) - e0(k))
                energy_diff = abs(e0(root) - e0(k))
                if abs(matel(root, k)) >= energy_diff:
                    print(f"Error: ")
                print("k=", k, " s=", s)
                print("k=", k, " s=", s, file=file1)
            for k in range(root + 1, kmax + 1):
                s += matel(root, k)**2 / (e0(root) - e0(k))
                print("k=", k, " s=", s)
                print("k=", k, " s=", s, file=file1)
```

```
175             return s
176
177         def c_psi_corr_1(kmax, root):
178             c = np.zeros(kmax)
179             for k in range(0, root):
180                 c[k] = matel(root, k) / (e0(root) - e0(k))
181             for k in range(root + 1, kmax + 1):
182                 c[k - 1] = matel(root, k) / (e0(root) - e0(k))
183             return c
184
185         def psi_corr_1(x, c, n, root):
186             kmax = len(c)
187             s = 0.0
188             for k in range(0, root):
189                 s += c[k] * psi_interp(k, x)
190             for k in range(root + 1, kmax + 1):
191                 s += c[k - 1] * psi_interp(k, x)
192             return s
193
194
195         def psi(x, c, n, root):
196             return psi_interp(root, x) + psi_corr_1(x, c, n, root)
197
198         def result(root):
199             kmax = int(len(exact_energies) - 1)
200             print("=======================================")
201             print("=======================================",file=file1)
202             print(f"State {root}")
203             print(f"State {root}", file=file1)
204             print("kmax = ", kmax)
205             print("kmax = ", kmax, file=file1)
206             e1 = e0(root) + matel(root, root)
207             print("e0(1)=", e0(root))
208             print("e0(1)=", e0(root), file=file1 )
209             print("e=", e1, " (1-approximation)")
210             print("e=", e1, " (1-approximation)", file=file1)
211             e2 = e0(root) + matel(root, root) + e_corr_2(kmax, root)
212             print("e=", e2, " (2-approximation)")
213             print("e=", e2, " (2-approximation)", file=file1)
214
215             c1 = c_psi_corr_1(kmax, root)
216             for i in range(len(c1)):
217                 print("c[{:1d}] = {:15.8e}".format(i, c1[i]))
218                 print("c[{:1d}] = {:15.8e}".format(i, c1[i]), file=file1)
219             psi_arr = np.array([psi(x, c1, n, root) for x in X])
```

```
E_min2, E_max2, step2 = 0.0, 3.0, 0.01
exact_energies2 = find_exact_energies(E_min2 + 0.001, E_max2, step2,
    tol, U)
f_fun(exact_energies2[root], n, U)
Psi_copy2 = normalize_wavefunction(Psi.copy())
psi_arr2 = normalize_wavefunction(psi_arr.copy())
psi_density = psi_arr2 ** 2

mean_Px = mean_momentum(psi_arr2, X)
mean_Px2 = mean_square_momentum(psi_arr2, X)
print(f"E_perturbation = {e2:.6f}, <p_x> = {mean_Px:.6e}, <p_x^2> =
    {mean_Px2:.6e}")
print(f"E_perturbation = {e2:.6f}, <p_x> = {mean_Px:.6e}, <p_x^2> =
    {mean_Px2:.6e}", file=file1)
mean_Px = mean_momentum(Psi_copy2, X)
mean_Px2 = mean_square_momentum(Psi_copy2, X)
print(f"E_target = {exact_energies2[root]:.6f}, <p_x> = {mean_Px:.6e
    }, <p_x^2> = {mean_Px2:.6e}")
print(f"E_target = {exact_energies2[root]:.6f}, <p_x> = {mean_Px:.6e
    }, <p_x^2> = {mean_Px2:.6e}", file=file1)
print("=======================================")
print("=======================================", file=file1)


Zero = np.zeros(n, dtype=float)
Upot = np.array([U(X[i]) for i in np.arange(n)])

plt.xlabel("x", fontsize=18, color="k")
plt.ylabel("Psi(x)", fontsize=18, color="k")
plt.plot(X, Zero, 'k--', linewidth=1.0)
plt.plot(X, Upot, 'g-', linewidth=3.0, label="U(x)")
line1, = plt.plot(X, psi_arr2, 'b-', linewidth=3.0, label="")
line2, = plt.plot(X, psi_density, 'r-', linewidth=3.0, label="")

plt.subplots_adjust(bottom=0.3)

plt.figtext(0.5, 0.02,
            f"Psi{root} perturbation method (E = {e2:.4f})\n"
            f"Psi{root} probability density)",
            ha="center", fontsize=10)

plt.legend([line1, line2],
            [f"Psi{root} perturbation method ", f"Psi{root}
                probability density"],
            fontsize=7, loc='upper right')
```

```
259
260             plt.grid(True)
261             plt.twinx()
262             plt.yticks([])
263             plt.savefig(f"State{root} probability density.pdf", dpi=300)
264             plt.show()
265             plt.close('all')
266
267
268             plt.xlabel("x", fontsize=18, color="k")
269             plt.ylabel("Psi(x)", fontsize=18, color="k")
270             plt.plot(X, Zero, 'k--', linewidth=1.0)
271             plt.plot(X, Upot, 'g-', linewidth=3.0, label="U(x)")
272             line1, = plt.plot(X, psi_arr2, 'b-', linewidth=3.0, label="")
273             line2, = plt.plot(X, Psi_copy2, 'r-', linewidth=3.0, label="")
274
275             plt.subplots_adjust(bottom=0.3)
276
277             plt.figtext(0.5, 0.02,
278                         f"Psi{root} perturbation method (E = {e2:.4f})\n"
279                         f"Psi{root} target method (E = {exact_energies2[root]:.4f
                                })",
280                         ha="center", fontsize=10)
281
282             plt.legend([line1, line2],
283                         [f"Psi{root} perturbation method ", f"Psi{root} target
                                method"],
284                         fontsize=10, loc='upper right')
285
286             plt.grid(True)
287             plt.twinx()
288             plt.yticks([])
289             plt.savefig(f"State{root}.pdf", dpi=300)
290             plt.show()
291             plt.close('all')
292
293         result(0)
294         result(2)
295
296         plt.plot(X, [U0(x) for x in X], 'g-', linewidth=3.0, label="U(x)")
297         plt.xlabel("X")
298         plt.ylabel("U0(X)")
299         plt.grid(True)
300         plt.legend()
301         plt.savefig("U0(X).pdf", dpi=300)
```

```
302             plt.show()
303
304             plt.plot(X, [U(x) for x in X], 'g-', linewidth=3.0, label="U(x)")
305             plt.xlabel("X")
306             plt.ylabel("U(X)")
307             plt.grid(True)
308             plt.legend()
309             plt.savefig("U(X).pdf", dpi=300)
310             plt.show()
```

Список литературы

1. Тимошенко Ю.К. Численное решение стационарного уравнения Шредингера: метод пристрелки. Учебное пособие. Воронеж: Научная книга, 2019. 35 с.

2. Доля П.Г. Введение в научный Python. Харьков: ХНУ, 2016. 265 с.

3. Тимошенко Ю. К. Численное решение стационарного уравнения Шрёдингера: теория возмущений. Учебное пособие. Воронеж: Научная книга, 2019. 14 с