МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики, информатики и механики
Кафедра вычислительной математики и прикладных информационных технологий

ЛАБОРАТОРНАЯ РАБОТА №2
ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНОГО УРАВНЕНИЯ ШРЁДИНГЕРА: ТЕОРИЯ ВОЗМУЩЕНИЙ

Направление: 01.04.02 – Прикладная математика и информатика
Выполнил: студент 11 группы 2 курса магистратуры
Крутько А.С.
Преподаватель: доктор физ.-мат. наук, профессор Тимошенко Ю.К.

Воронеж 2024

Содержание

1 Цели и задачи работы

1.1 Цель работы.

Целями лабораторной работы являются практическое освоение информации, полученной при изучении курса «Компьютерное моделирование в математической физике» по теме «Численное решение стационарного уравнения Шрёдингера», а также развитие алгоритмического мышления и приобретение опыта использования знаний и навыков по математике, численным методам и программированию для решения прикладных задач физико-технического характера.

1.2 Задачи работы:

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

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

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

1. Используя метод 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 числовые 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.

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

Одномерное стационарное уравнение Шрёдингера [1]:

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

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

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

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

где оператор кинетической энергии

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

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

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

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

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

Особый практический интерес представляет случай, когда

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

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

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

Квантовое состояние, обладающее наименьшей энергией, называется основным. Остальные состояния называют возбужденными состояниями. В силу линейности стационарного уравнения Шрёдингера, волновые функции математически определены с точностью до постоянного множителя. Однако, из физических соображений, волновые функции должны быть нормированы следующим образом:

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

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

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

3 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; – 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 удаётся 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 ив возмущенной систем [3] имеют вид:

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

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

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, $\Psi_n^{(k)}$ – 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:

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

где

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

Штрих над знаком суммы означает пропуск слагаемого с $m = n$: $\sum'_m \equiv \sum_{m\neq n}$. Очевидно, что ряды в (11) сходятся, если выполняется равенство

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

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

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

При вычислении второй поправки к энергии и первой поправки к волновой функции основного состояния возмущенной системы бесконечные суммы в (11) 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 Приложение представлена программа на языке Python 3.12[2], реализованная в среде разработки PyCharm Community Edition 2024.3.1, численного решения одномерного стационарного уравнения Шрёдингера 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 124–127).

В строках 6–140 реализован алгоритм пристрелки, который подробно разобран в лабораторной работе №1, в данной программе этот алгоритм используется для реализации невозмущенной системы и вычисления её 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 (11). П программе за верхний предел сумм отвечает `k_max` в строке 233, он равен количеству вычисленных энергий невозмущенной системы, для длинного варианта задачи достаточно было 11.

В строках 154–161 реализована потенциальная функция возмущенной системы, для этого был создан пик высотой 1, который больше $\max(U(x))$ на отрезке $[-0.545; 0.545]$

В строках 164–165 реализован оператор возмущения.

В строках 168–172 и 175–178 реализованы функции возвращающие энергии и волновые функции невозмущенной системы.

В строках 181–184 и 187–190 реализованы функции которые вычисляют матричный элемент оператора возмущения по невозмущенным системам (12).

В строках 193–206 реализована функция вычисляющая поправку второго порядка (11).

В строках 209–215 и 218–225 реализованы функции вычисляющие поправку первого порядка для волновой функции возмущенной системы.

В строках 228–229 реализована функция вычисляющая первое приближение волновой функции (15)

В строках 232–325 реализована функция с одним параметром `root` определяющий 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 327–328 вызывается функция`result` для основного и второго возбужденного состояний возмущенной системы.

В строках 330–344 выводятся графики невозмущенной и возмущенной систем.

5 Результаты численных экспериментов

Ниже продемонстрированы результаты работы программного кода написанного на Python.

Потенциал из постановки задачи представлен на Рис. 1.

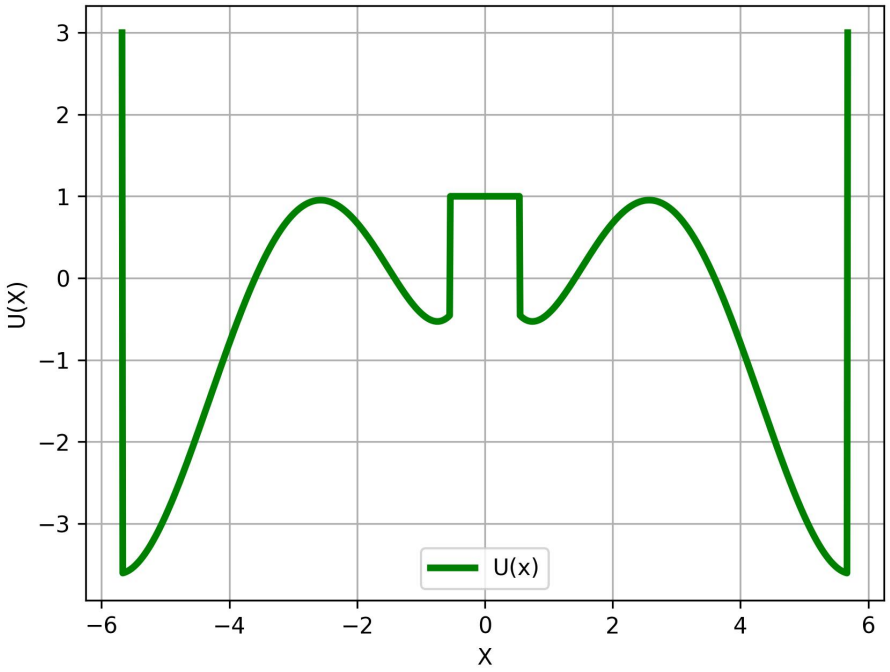


Рис. 1: Невозмущенная система

Возмущенная система представлена на Рис. 2.

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

Энергии, где e_0 – энергия основного состояния невозмущенной системы, а e – энергия возмущенной системы вычисленная с учетом поправок:

Энергия	**а.е.**
e_0	0.026451
e	0.498856

s – суммы из формулы второй поправки (12), которая поочередно вычислялась с каждым состоянием невозмущенной системы:

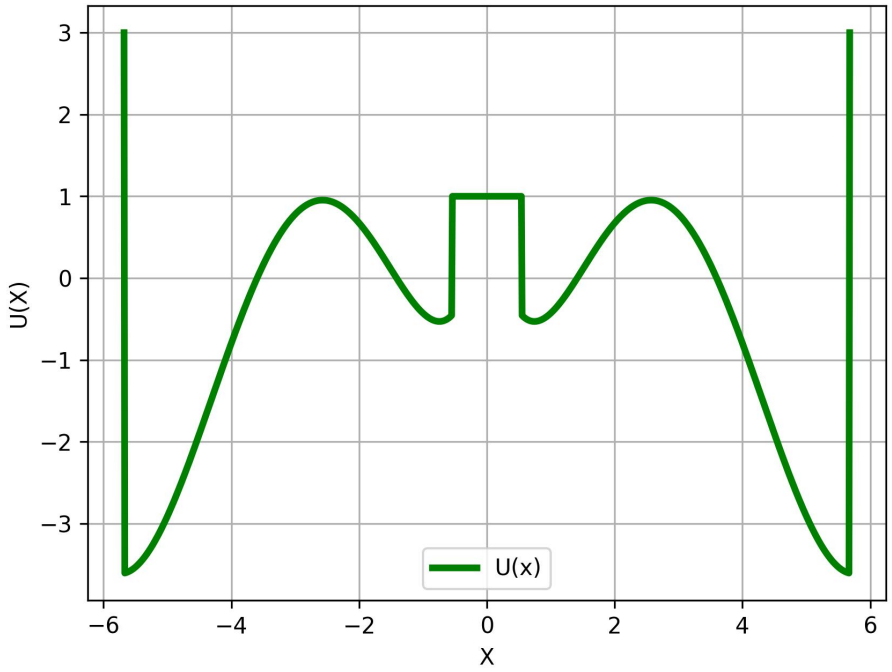
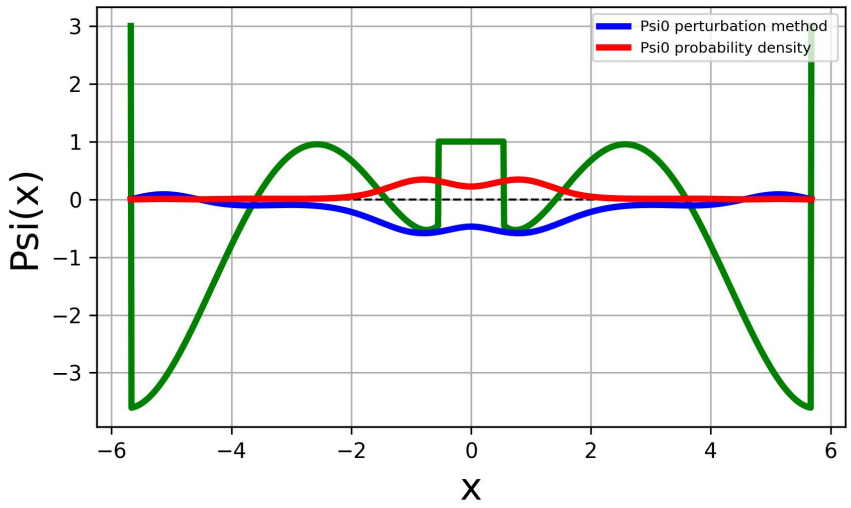


Рис. 2: Возмущенная система

Энергия невозмущенной системы	**Энергия, а.е.**
1	$-3.546785896711328e-14$
2	-0.012860815109642064
3	-0.01286081510981086
4	-0.01286081510986983
5	-0.025191454299853946
6	-0.025191454299854834
7	-0.03018338913184664
8	-0.030183389131850178
9	-0.03177543923923762
10	-0.0317754392392409
11	-0.032087871459281776

На Рис. 3 представлена волновая функция и плотность вероятности возмущенной системы для основного состояния.

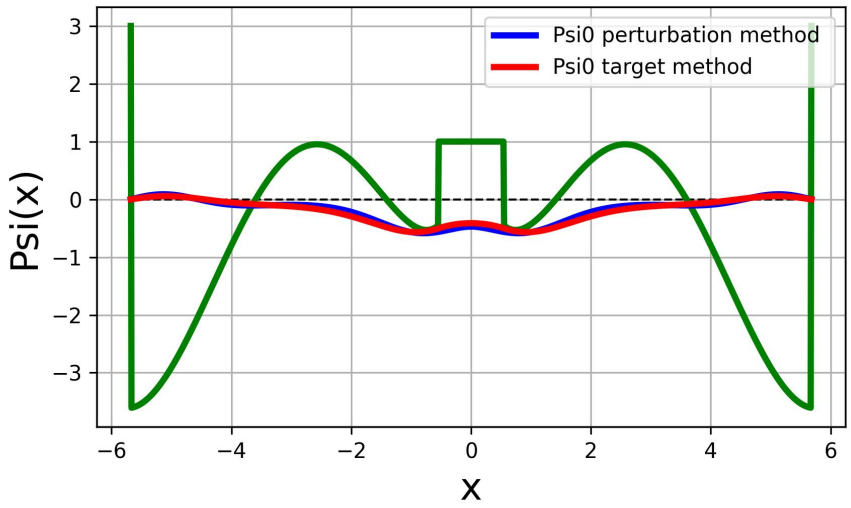


Psi0 perturbation method (E = 0.3399)
Psi0 probability density)

Рис. 3: Волновая функция основного состояния возмущенной системы

Данная функция соответствует основному состоянию осцилляционной теоремы. Т.к, как уже было установленно в первой лабораторной работе, основным состоянием в случае данной функции является состояние в котором волновая функция имеет два пересечения с осью абсцисс.

На Рис. (4) представлено сравнение волновых функций основного состояния возмущенной системы вычисленных методом пристрелки и методом возмущений.



Psi0 perturbation method (E = 0.3399)
Psi0 target method (E = 0.3023)

Рис. 4: Сравнение волновых функций полученных разными методами

Ниже представлены значения энергий основного состояния и квантовомеханических

средних $\langle p(x) \rangle$ и $\langle p(x^2) \rangle$, вычисленные методом пристрелки и методом возмущений:

Метод решения	**Энергия, а.е.**	$\langle p(x) \rangle$	$\langle p(x^2) \rangle$
Метод возмущений	0.339903	$0.000000e+00$	$4.420559e-01$
Метод пристрелки	0.302288	$0.000000e+00$	$3.434077e-01$

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

Энергии, где e_0 – энергия второго возбужденного состояния невозмущенной системы, а e – энергия возмущенной системы вычисленная с учетом поправок:

Энергия	**а.е.**
e_0	0.8155639648437506
e	0.8450739082702535

s – суммы из формулы второй поправки (12), которая поочередно вычислялась с каждым состоянием невозмущенной системы:

Энергия невозмущенной системы	**Энергия, а.е.**
0	0.012860815109606596
1	0.012860815109628453
3	0.012860815109407921
4	0.012860815109389715
5	0.011316012393792629
6	0.011316012393790565
7	0.01075306784158517
8	0.010753067841582727
9	0.010574304144826133
10	0.010574304144824201
11	0.010534361777482728

На Рис. 5 представлена волновая функция и плотность вероятности возмущенной системы для второго возбужденного состояния.

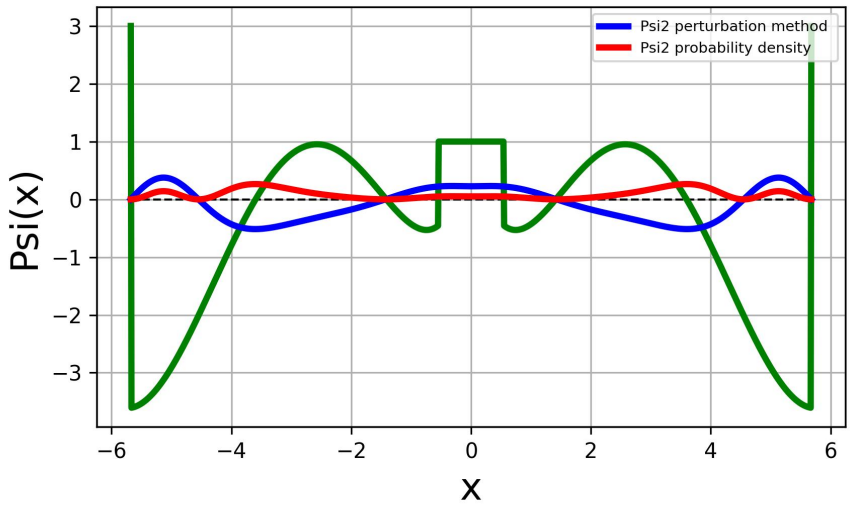


Рис. 5: Волновая функция второго возбужденного состояния возмущенной системы

Данная функция соответствует основному состоянию осцилляционной теоремы. Т.к, как уже было установлено в первой лабораторной работе, основным состоянием в случае функции основного состояния является состояние в котором волновая функция имеет два пересечения с осью абсцисс, следовательно, для второго возбужденного имеется 4 пересечения с осью абсцисс.

На Рис. 6 представлено сравнение волновых функций второго возбужденного состояния возмущенной системы вычисленных методом пристрелки и методом возмущений.

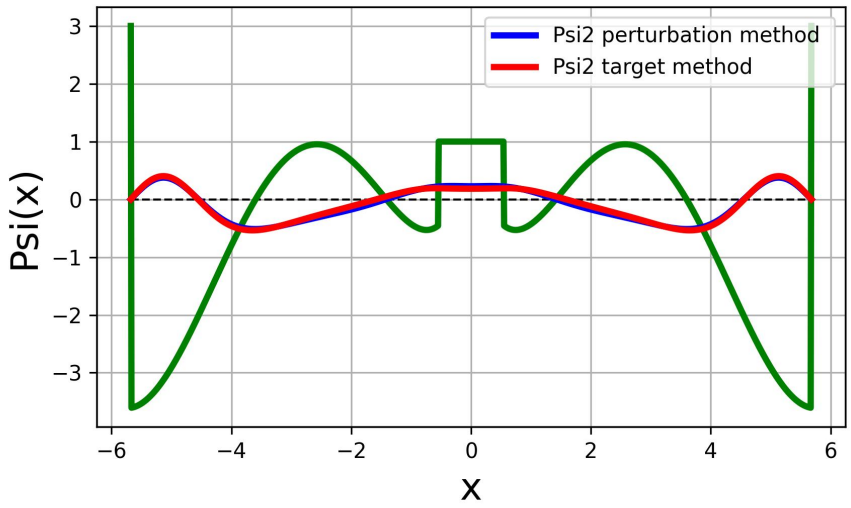


Рис. 6: Сравнение волновых функций полученных разными методами

Ниже представлены значения энергий второго возбужденного состояния и квантово-механических средних $\langle p(x) \rangle$ и $\langle p(x^2) \rangle$, вычисленные методом пристрелки и методом возмущений:

Метод решения	**Энергия, а.е.**	$\langle p(x) \rangle$	$\langle p(x^2) \rangle$
Метод возмущений	0.855608	$0.000000e+00$	$2.374319e+00$
Метод пристрелки	0.848595	$0.000000e+00$	$2.649263e+00$

6 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. Полученные волновые функции соответствуют осцилляционной теореме. Кроме того, для каждого состояния были вычисленные квантовомеханические средние $\langle p(x)\rangle, \langle p(x^2)\rangle$. Полученные волновые функции соответствуют осцилляционной теореме. Сравнивая результаты полученные методом возмущений с методом пристрелки можно сказать что метод возмущений менее точен и требует больших затрат чем метод пристрелки: как со стороны разработчика, так и со стороны ЭВМ на котором производятся вычисления.

Приложение

```
"""
                    .   .
                                                                              :
                                          .
                                                           Python 3.12,
                                          PyCharm Community Edition 2024.3.1,
                                               Windows 10.
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import eval_laguerre


def u0_func(x):
    if abs(x) <= L:
        return V0 * eval_laguerre(5, abs(x))
    else:
        return 3.0


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


def find_derivative(y_arg, h_arg, m):
    return (y_arg[m - 2] - y_arg[m + 2] + 8.0 * (y_arg[m + 1] - y_arg[m - 1])) / (12.0 * h_arg)


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


def mean_momentum(psi_arg, x_arg):
    h_bar = 1.0
    dPsi_dx = np.gradient(psi_arg, x_arg)
    integrand = psi_arg.conj() * dPsi_dx
    mean_Px = -1j * h_bar * np.trapz(integrand, x_arg)
    return mean_Px.real


def mean_square_momentum(psi_arg, x_arg):
    h_bar = 1.0
    d2Psi_dx2 = np.gradient(np.gradient(psi_arg, x_arg), x_arg)
    integrand = psi_arg.conj() * d2Psi_dx2
    mean_Px2 = -h_bar**2 * np.trapz(integrand, x_arg)
    return mean_Px2.real


def f_fun(e, n_arg, potential_func):
    F = np.array([c * q_func(e, X[i], potential_func) for i in np.arange(n_arg)])
    Psi.fill(0.0)
    Psi[0] = 0.0
    Fi[n_arg - 1] = 0.0
    Psi[1] = d1
    Fi[n_arg - 2] = d2
```

```
54
55      for i in np.arange(1, n_arg - 1, 1):
56          p1 = 2.0 * (1.0 - 5.0 * F[i]) * Psi[i]
57          p2 = (1.0 + F[i - 1]) * Psi[i - 1]
58          Psi[i + 1] = (p1 - p2) / (1.0 + F[i + 1])
59
60      for i in np.arange(n_arg - 2, 0, -1):
61          f1 = 2.0 * (1.0 - 5.0 * F[i]) * Fi[i]
62          f2 = (1.0 + F[i + 1]) * Fi[i + 1]
63          Fi[i - 1] = (f1 - f2) / (1.0 + F[i - 1])
64
65      p1 = np.abs(Psi).max()
66      p2 = np.abs(Psi).min()
67      big = p1 if p1 > p2 else p2
68
69      Psi[:] = Psi[:] / big
70      coefficient = Psi[r] / Fi[r]
71      Fi[:] = coefficient * Fi[:]
72
73      return find_derivative(Psi, h, r) - find_derivative(Fi, h, r)
74
75
76  def energy_scan(e_min_arg, e_max_arg, step_arg, potential_func):
77      energies = []
78      values = []
79      E = e_min_arg
80      while E <= e_max_arg:
81          f_value = f_fun(E, n, potential_func)
82          energies.append(E)
83          values.append(f_value)
84          E += step_arg
85      return energies, values
86
87
88  def find_exact_energies(e_min, e_max, step_arg, tol_arg, potential_func):
89      energies, values = energy_scan(e_min, e_max, step_arg, potential_func)
90      inner_exact_energies = []
91      for i in range(1, len(values)):
92          Log1 = values[i] * values[i - 1] < 0.0
93          Log2 = np.abs(values[i] - values[i - 1]) < limit
94          if Log1 and Log2:
95              E1, E2 = energies[i - 1], energies[i]
96              exact_energy = bisection_method(E1, E2, tol_arg, potential_func)
97              f_fun(exact_energy, n, potential_func)
98              inner_exact_energies.append(exact_energy)
99      return inner_exact_energies
100
101
102 def bisection_method(e1, e2, tol_arg, potential_func):
103     while abs(e2 - e1) > tol_arg:
104         Emid = (e1 + e2) / 2.0
105         f1, f2, f_mid = f_fun(e1, n, potential_func), f_fun(e2, n,
    potential_func), f_fun(Emid, n, potential_func)
106         if f1 * f_mid < 0.0:
107             e2 = Emid
108         else:
109             e1 = Emid
110         if f2 * f_mid < 0.0:
111             e1 = Emid
112         else:
```

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


def count_zeros(psi, x):
    y_axis_crossings_count = 0
    for i in range(1, len(psi) - 1):
        if psi[i - 1] * psi[i] < 0:
            y_axis_crossings_count += 1
    return y_axis_crossings_count

c_energy = 27.212
c_length = 0.5292
V0 = 25.0 / c_energy
L = 3.0 / c_length
A, B = -L - 0.01, L + 0.01
n = 1000
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 - 80
limit = 7.0

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

E_min, E_max, step = 0.0, 9.0, 0.001
exact_energies = find_exact_energies(E_min + 0.001, E_max, step, tol,
    u0_func)

if len(exact_energies) == 0:
    print("Error.")
else:
    print("Energies:")
    for i, E in enumerate(exact_energies):
        f_fun(E, n, u0_func)
        zeros_count = count_zeros(Psi, X)
        print(f"Energy level {i}: {E:.6f}, cross: {zeros_count}")

results_file = open(f"./python_results/result.txt", "a")


def u_func(x):
    if abs(x) < L:
        if -0.545 <= x <= 0.545:
            return 1
        else:
            return V0 * eval_laguerre(5, abs(x))
    else:
        return 3.0


def v_func(x):
    return u_func(x) - u0_func(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_func)
    Psi_copy = normalize_wave_function(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_func(x) * psi_k2


def matrix_el(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 += matrix_el(root, k) ** 2 / (e0(root) - e0(k))
        energy_diff = abs(e0(root) - e0(k))
        if abs(matrix_el(root, k)) >= energy_diff:
            print(f"                                                : ")
        print("k = ", k, " s = ", s)
        print("k = ", k, " s = ", s, file=results_file)
    for k in range(root + 1, kmax + 1):
        s += matrix_el(root, k) ** 2 / (e0(root) - e0(k))
        print("k = ", k, " s = ", s)
        print("k = ", k, " s = ", s, file=results_file)
    return s


def c_psi_corr_1(kmax, root):
    c = np.zeros(kmax)
    for k in range(0, root):
        c[k] = matrix_el(root, k) / (e0(root) - e0(k))
    for k in range(root + 1, kmax + 1):
        c[k - 1] = matrix_el(root, k) / (e0(root) - e0(k))
    return c


def psi_corr_1(x, c, n, root):
    kmax = len(c)
    s = 0.0
    for k in range(0, root):
        s += c[k] * psi_interp(k, x)
    for k in range(root + 1, kmax + 1):
        s += c[k - 1] * psi_interp(k, x)
    return s


def psi(x, c, n, root):
    return psi_interp(root, x) + psi_corr_1(x, c, n, root)


```

```
def result(root):
    kmax = int(len(exact_energies) - 1)
    print("-----------------------------------------------------")
    print("-----------------------------------------------------", file=
    results_file)
    print(f"State {root}")
    print(f"State {root}", file=results_file)
    print("kmax = ", kmax)
    print("kmax = ", kmax, file=results_file)
    e1 = e0(root) + matrix_el(root, root)
    print("e0(1) = ", e0(root))
    print("e0(1) = ", e0(root), file=results_file)
    print("e = ", e1, "  (1-approximation)")
    print("e = ", e1, "  (1-approximation)", file=results_file)
    e2 = e0(root) + matrix_el(root, root) + e_corr_2(kmax, root)
    print("e = ", e2, "  (2-approximation)")
    print("e = ", e2, "  (2-approximation)", file=results_file)

    c1 = c_psi_corr_1(kmax, root)
    for i in range(len(c1)):
        print("c[{:1d}] = {:15.8e}".format(i, c1[i]))
        print("c[{:1d}] = {:15.8e}".format(i, c1[i]), file=results_file)
    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_func)
    f_fun(exact_energies2[root], n, u_func)
    Psi_copy2 = normalize_wave_function(Psi.copy())
    psi_arr2 = normalize_wave_function(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=results_file)
    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=results_file)
    print("-----------------------------------------------------")
    print("-----------------------------------------------------", file=
    results_file)


    Zero = np.zeros(n, dtype=float)
    Upot = np.array([u_func(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')

    plt.grid(True)
    plt.twinx()
    plt.yticks([])
    plt.savefig(f"./python_results/State{root} probability density.jpg", dpi=300)
    plt.show()
    plt.close('all')


    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_copy2, '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} target method (E = {exact_energies2[root]:.4f})",
                ha="center", fontsize=10)

    plt.legend([line1, line2],
               [f"Psi{root} perturbation method ", f"Psi{root} target method"],
               fontsize=10, loc='upper right')

    plt.grid(True)
    plt.twinx()
    plt.yticks([])
    plt.savefig(f"./python_results/State{root}.jpg", dpi=300)
    plt.show()
    plt.close('all')

result(0)
result(2)

plt.plot(X, [u0_func(x) for x in X], 'g-', linewidth=3.0, label="U(x)")
plt.xlabel("X")
plt.ylabel("U0(X)")
plt.grid(True)
plt.legend()
plt.savefig("./python_results/U0(X).jpg", dpi=300)
plt.show()

plt.plot(X, [u_func(x) for x in X], 'g-', linewidth=3.0, label="U(x)")
plt.xlabel("X")
plt.ylabel("U(X)")
```

```
341 plt.grid(True)
342 plt.legend()
343 plt.savefig("./python_results/U(X).jpg", dpi=300)
344 plt.show()
```

Листинг 1: Код файла solver.py

Список литературы

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

[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. 14с.