МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра вычислительной математики  
и прикладных информационных технологий

ЛАБОРАТОРНАЯ РАБОТА №3

ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНОГО

УРАВНЕНИЯ ШРЁДИНГЕРА:

ВАРИАЦИОННЫЙ ПРИНЦИП

Направление 01.04.02 Прикладная математика и информатика

Профиль Математическое моделирование и вычислительная математика

|  |  |  |  |
| --- | --- | --- | --- |
| Зав. кафедрой | \_\_\_\_\_\_\_\_\_\_ | д. ф.-м.н., пр. | А.И. Шашкин \_\_.\_\_.2023 |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_ |  | И.Б. Рахимов |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_ | д.ф.-м.н., пр. | Ю.К. Тимошенко |

Воронеж 2023

Содержание

Цели и задачи 3

1. Одномерное стационарное уравнение Шрёдингера. Прямой вариационный метод (метод Ритца) 5

2. Применение вариационного метода 9

3. Программная реализация 11

4. Результаты численных экспериментов и их анализ 12

Список использованных источников 13

Приложения 14

Приложение 1 Компьютерный код 14

# Цели и задачи

### Цели работы

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

### Задачи работы

**Формулировка проблемы:** электрон находится в потенциальном поле вида (рис. 1):

где , , .

С помощью вариационного метода найти:

* энергию основного состояния;
* волновую функцию основного состояния.

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

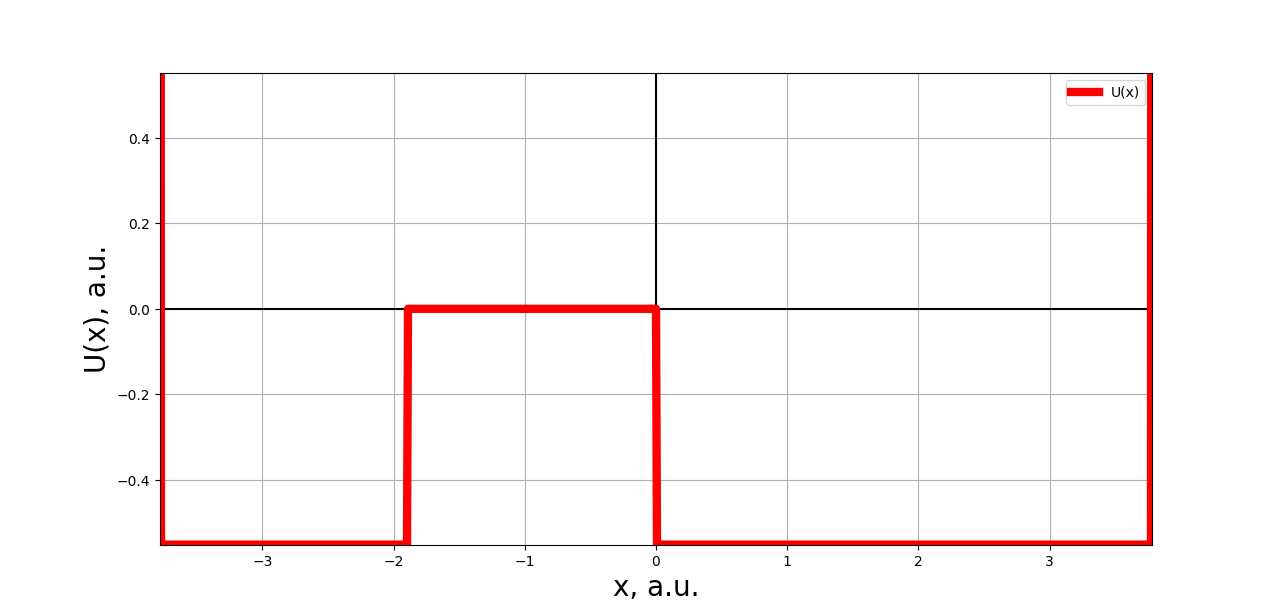


Рис. 1. График потенциальной функции

# Одномерное стационарное уравнение Шрёдингера. Прямой вариационный метод (метод Ритца)

В данном отчете не будет подробно описана математическая часть одномерного стационарного уравнение Шрёдингера, так как это уже было сделано в отчете по первой лабораторной работе, вместо этого здесь будет подробно разобрано его решение с помощью прямого вариационного метода.

Рассмотрим стационарное уравнение Шрёдингера для дискретного спектра:

Функции ортонормированы:

Совокупность всех собственных функций для дискретного спектра образует полную или замкнутую системы функций, т.е. любая другая функция , которая зависит от тех же переменных и удовлетворяет тем же граничным условиям, для которой существует интеграл может быть точно представлена в виде ряда:

Умножаем левую и правую части равенства на и проинтегрируем по всем пространству:

где .

Таким образом заменяя на , получаем:

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

где

То есть функцию можно записать в виде:

Формулу часто называют условие полноты базиса . Поэтому, когда говорят, что базис является ортонормированным и полным, то имеют ввиду:

Так же равенство в форме:

называется условие нормировки, а волновые функции, удовлетворяющие этому условию, называются нормированными функциями. Далее, для нормирования функций величина определяет вероятность значений координат системы в интервале в состоянии с волновой функцией . В этом случае величину

называют плотностью вероятности.

Вернёмся к уравнению Шрёдингера, умножим левую и правую части на и проинтегрируем по всем пространству:

Интеграл в левой части является квантовомеханическим средним значением в состоянии :

Найдём квантовомеханическое среднее гамильтониана в состоянии с волновой функцией , используя формулу :

Пусть соответствует основному состоянию, тогда энергии возбуждённого состояния . Заменим в правой части все на Тогда получим неравенство:

Здесь используется информированность функции :

Равенство в соответствует . Функцию называют «пробной функцией», содержащей некоторое количество параметров подлежащих определению.

В рамках вариационного подхода волновую функцию основного состояния с энергией приближённо ищут путём минимизации по параметрам интеграла:

при условии нормировки:

Обозначим

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

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

# Применение вариационного метода

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

Пусть имеется оператор собственные функции которого образуют полный ортонормированный базис:

где соответствует квантовому состоянию.

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

Подставим в :

Левую и правую части умножим на и проинтегрируем по всем пространству:

Перепишем в следующем виде:

где

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

где

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

# Программная реализация

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

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

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

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

# Результаты численных экспериментов и их анализ

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

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

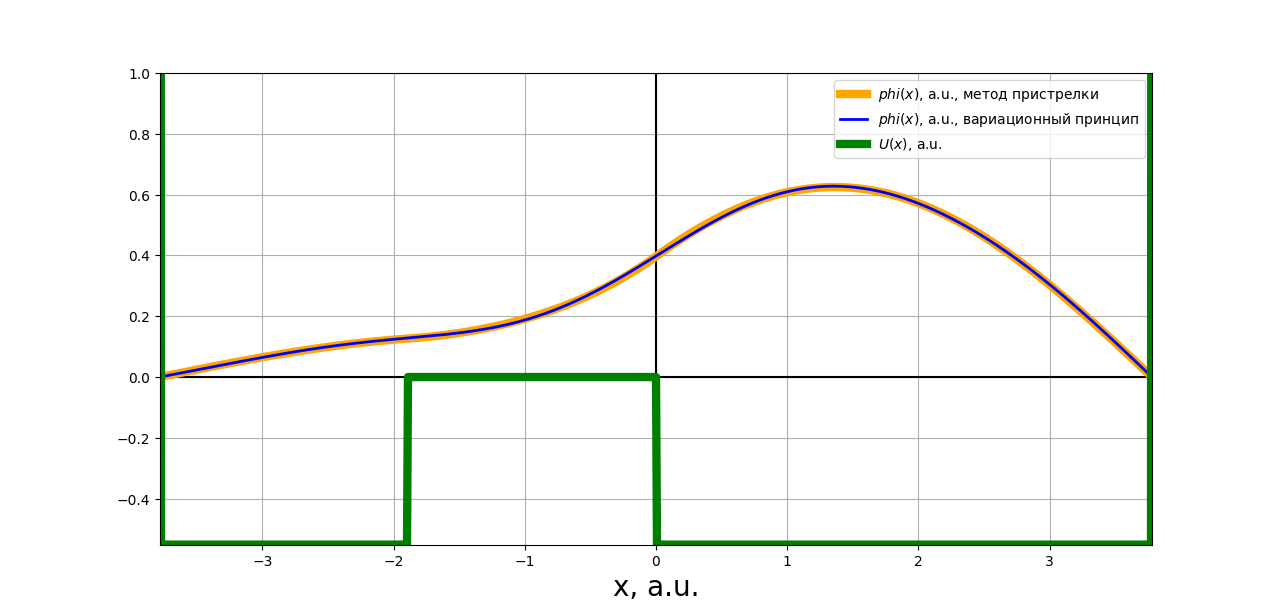


Рис. 2. Основное состояние

Энергия основного состояния, рассчитанная по методу Ритца равна:

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

# Список использованных источников

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

# Приложения

## Приложение 1 Компьютерный код

import matplotlib.pyplot as plt

from math import sqrt, cos, sin, pi

from numpy import linalg as LA

######################################################################

### bisection\_method

######################################################################

# метод деления отрезка пополам

def bisection\_method(f, # функция корень которой необходимо найти

begin:float, end:float, # начало и конец отрезка

# в котором находится корень

steps:int, # максимальное количество шагов

# алгоритма

eps:float):# точность приближения корня

l = begin

r = end

step = 0

while step < steps:

step += 1

center = 0.5 \* (r + l)

f\_l = f(l)

f\_c = f(center)

f\_r = f(r)

if abs(f\_c) < eps:

break

if f\_l \* f\_c < 0:

r = center

else: # f\_c \* f\_r < 0

l = center

return 0.5 \* (r + l)

######################################################################

### quantum\_mechanics

######################################################################

def v(x:float) -> float:

if (x > -half\_width and x < (-half\_width / 2.0)) or

(x > 0 and x < half\_width):

return -1

elif x >= -half\_width / 2.0 and x <= 0:

return 0

else:

return w

# потенциальная функция

def U(x:float) -> float:

return v0 \* v(x)

def q(e:float,

x:float) -> float:

return 2.0 \* (e - U(x))

def compute\_q(x:list[float],

energy:float) -> list[float]:

return [q(energy, xi) for xi in x]

def derivative(function:list[float],

index:int,

eps:float) -> float:

der1 = function[index - 2] - function[index + 2]

der2 = function[index + 1] - function[index - 1]

return (

(der1 + 8.0 \* der2)

/ #-------------------

(12.0 \* eps)

)

# интегрирование вперёд

def forward\_integration(num\_intervals:int,

penultimate\_approximation:float,

q:list[float],

step:float) -> list[float]:

num\_points = num\_intervals + 1

forward = [float] \* (num\_points)

forward[0] = 0.0

forward[1] = penultimate\_approximation

c = step \*\* 2 / 12.0

for i in range(1, num\_intervals):

p1 = 2.0 \* (1.0 - 5.0 \* c \* q[i]) \* forward[i]

p2 = (1.0 + c \* q[i - 1]) \* forward[i - 1]

p3 = (1.0 + c \* q[i + 1])

forward[i + 1] = (

(p1 - p2)

/ #-----------

p3

)

return forward

# интегрирование назад

def backward\_integration(num\_intervals:int,

first\_approximation:float,

q:list[float],

step:float) -> list[float]:

num\_points = num\_intervals + 1

backward = [float] \* num\_points

backward[num\_intervals] = 0

backward[num\_intervals - 1] = first\_approximation

c = step \*\* 2 / 12.0

for i in range(num\_intervals - 1, 0, -1):

f1 = 2.0 \* (1.0 - 5.0 \* c \* q[i]) \* backward[i]

f2 = (1.0 + c \* q[i + 1]) \* backward[i + 1]

backward[i - 1] = (

(f1 - f2)

/ #--------------------

(1.0 + c \* q[i - 1])

)

return backward

# фукнция для нормировки forward

# и достижения равенства на узле сшивки

def normalization(forward:list[float],

backward:list[float],

connection:int):

# нормировка

norm = abs(max(forward, key = abs))

forward = list(map(lambda x: x / norm, forward))

# равенство на узле сшивки - connection

coef = forward[connection] / backward[connection]

backward = list(map(lambda x: coef \* x, backward))

return forward, backward

# функция возвращающая разницу производных на узле сшивки

def is\_close(forward:list[float],

backward:list[float],

connection:int,

eps:float) -> float:

return (derivative(forward, connection, eps)

- derivative(backward, connection, eps))

# функция вычисляет значения волновой функции вперёд и назад

# по известным данным

# и возвращающая разницу производных на узле сшивки

def is\_close\_energy(

energy:float, # значение энергии

x:list[float], # сетка

step:float, # шаг сетки

forward\_first\_approximation:float, # первое

# приближение для интегрирования вперёд

backward\_penultimate\_approximation:float, # n – 1

# приближение для интегрирования назад

connection:int) -> float: # узел сшивки

num\_intervals = len(x) - 1

q = compute\_q(x, energy)

forward = forward\_integration(num\_intervals,

forward\_first\_approximation, q, step)

backward = backward\_integration(num\_intervals,

backward\_penultimate\_approximation, q, step)

forward, backward = normalization(forward, backward, connection)

return is\_close(forward, backward, connection, step)

# возвращет интервалы в которых находятся

# собственные значения оператора Гамильтона

def eigen\_value\_intervals(min\_energy:float,

energy\_step:float,

max\_energy\_value:float,

max\_energy\_count:int,

x:list[float],

step:float,

forward\_first\_approximation:float,

# 1 приближение для интегрирования вперёд

backward\_penultimate\_approximation:float,

# n - 1 приближения для интегрирования назад

connection:int) -> list[float]:

level = 0

sign = (1 if level % 2 == 0 else -1)

energy = min\_energy

prev\_close = is\_close\_energy(energy,

x,

step,

sign \* forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

intervals = []

intervals.append(energy)

while(energy < max\_energy\_value):

energy = energy + energy\_step

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):

derivation[i] = coef \* (f[i - 1] - 2 \* f[i] + f[i + 1])

return derivation

w = 10.0

v0 = 15 # Электронвольт

half\_width = 2.0 # Ангстрем

print(f"v0 = {v0} Электронвольт")

print(f"l = {half\_width} Ангстрем")

# Перевод величин в атомную систему единиц

v0 = v0 / 27.211 # атомных единиц

half\_width = half\_width / 0.5292 # атомных единиц

print(f"v0 = {v0} атомных единиц")

print(f"l = {half\_width} атомных единиц")

def main():

min\_energy = -v0

print("Минимальное значение потенциальной функции = {}".format(min\_energy))

print()

forward\_first\_approximation = 1.e-9

backward\_penultimate\_approximation = 1.e-9

begin = -half\_width

end = +half\_width

num\_intervals = 1000

num\_points = num\_intervals + 1

connection = 300

step = (end - begin) / num\_intervals

x = [begin + step \* i for i in range(num\_points)]

energy\_step = 0.01

max\_energy\_value = 1000

max\_energy\_count = 2

intervals = eigen\_value\_intervals(min\_energy,

energy\_step,

max\_energy\_value,

max\_energy\_count,

x,

step,

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)

def phi(k:int, x:list[float]):

num\_points = len(x)

out = [float] \* num\_points

denominator = 1.0 / sqrt(half\_width)

for i in range(num\_points):

arg = (pi \* (k + 1) \* x[i]) / (2.0 \* half\_width)

if k % 2:

out[i] = denominator \* sin(arg)

else:

out[i] = denominator \* cos(arg)

return out

def Hphi(k:int, x:list[float]):

num\_points = len(x)

out = [float] \* num\_points

phi\_k = phi(k, x)

derivation = second\_derivation(phi\_k, step)

for i in range(num\_points):

out[i] = -0.5 \* derivation[i] + u[i] \* phi\_k[i]

return out

def H(m:int, k:int, x:list[float]):

phi\_m = phi(m, x)

Hphi\_k = Hphi(k, x)

num\_points = len(x)

mul = [float] \* num\_points

for i in range(num\_points):

mul[i] = phi\_m[i] \* Hphi\_k[i]

return integrate(mul, step)

def get\_Matrix(size:int, x:list[float]):

return [[H(m, k, x) for k in range(size)]

for m in range(size)]

size = 6

matrix = get\_Matrix(size, x)

eigenvalues, eigenvectors = LA.eigh(matrix)

low = 0

for i in range(1, len(eigenvalues)):

if eigenvalues[i] < eigenvalues[low]:

low = i

num\_points = len(x)

wave\_func = [0.0] \* num\_points

for k in range(size):

phi\_k = phi(k, x)

for i in range(num\_points):

wave\_func[i] += eigenvectors[k][low] \* phi\_k[i]

wave\_func = quantum\_normalize(wave\_func, step)

En = eigenvalues[low]

print(f"прямой вариационный метод - E = {En}")

print(f"метод пристрелки - E = {energies[0]}")

plt.axis([begin, end, min\_energy, v0 if v0 > 1 else 1])

plt.grid(True)

plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.xlabel("x, a.u.", fontsize = 20, color = "k")

plt.plot(x, forward\_n0, linewidth = 4, color = "orange",

label = "$phi(x)$, a.u., метод пристрелки")

plt.plot(x, wave\_func, linewidth = 2, color = "blue" ,

label = "$phi(x)$, a.u., вариационный принцип",

linestyle = 'dashed')

plt.plot(x, u, linewidth = 6, color = "green" ,

label = "$U(x)$, a.u.")

plt.legend()

plt.show()

main()