Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

**Отчет по лабораторной работе №2**

**по дисциплине «Вычислительная математика»**

**Вариант 11**

**Выполнили**:

Студенты группы P3265

Кручинина Дарья Сергеевна

Москвитина Полина

**Преподаватель:**

Машина Екатерина Алексеевна

**Цели работы**

Решение уравнений методам секущих, методом половинного деления и методом простых итераций.

## **Вычислительная реализация задачи**

### Часть 1. Решение нелинейного уравнения

#### 

Для первого корня: Метод половинного деления

Для второго корня: Метод секущих

Для третьего корня: Методом простой итерации

#### ε = 0. 01

Уточнение корня уравнения методом половинного деления

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага | a | b | x | f(a) | f(b) | f(x) | |a-b| |
| 0 | 3,700 | 4,000 | 3,850 | -2,620 | 2,458 | -0,271 | 0,300 |
| 1 | 3,850 | 4,000 | 3,925 | -0,271 | 2,458 | 1,045 | 0,150 |
| 2 | 3,850 | 3,925 | 3,888 | -0,271 | 1,045 | 0,375 | 0,075 |
| 3 | 3,850 | 3,888 | 3,869 | -0,271 | 0,384 | 0,054 | 0,038 |
| 4 | 3,850 | 3,869 | 3,860 | -0,271 | 0,054 | -0,109 | 0,019 |
| 5 | 3,860 | 3,869 | 3,865 | -0,101 | 0,054 | -0,024 | 0,009 |

Уточнение корня уравнения методом секущих

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № итерации | *xk-1* | *f*(*xk-1* ) | *xk* | *f*(*xk*) |  | *f*(*xk+1*) | │*xk* − *xk*+1│ |
| *xk*+1 |
|  |
| 1 | -1,600 | -4,038 | -1,500 | -2,698 | -1,299 | -0,457 | 0,201 |
| 2 | -1,500 | -2,698 | -1,300 | -0,469 | -1,258 | -0,073 | 0,042 |
| 3 | -1,300 | -0,470 | -1,258 | -0,075 | -1,250 | -0,003 | 0,008 |

Уточнение корня уравнения методом простой итерации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации | *xk* | *f*(*xk* ) | *xk*+1 |  | │*xk* − *xk*+1│ |
| 1 | 0,400 | 0,622 | 0,512 | 0,508 | 0,112 |
| 2 | 0,512 | -0,019 | 0,508 | 0,508 | 0,004 |

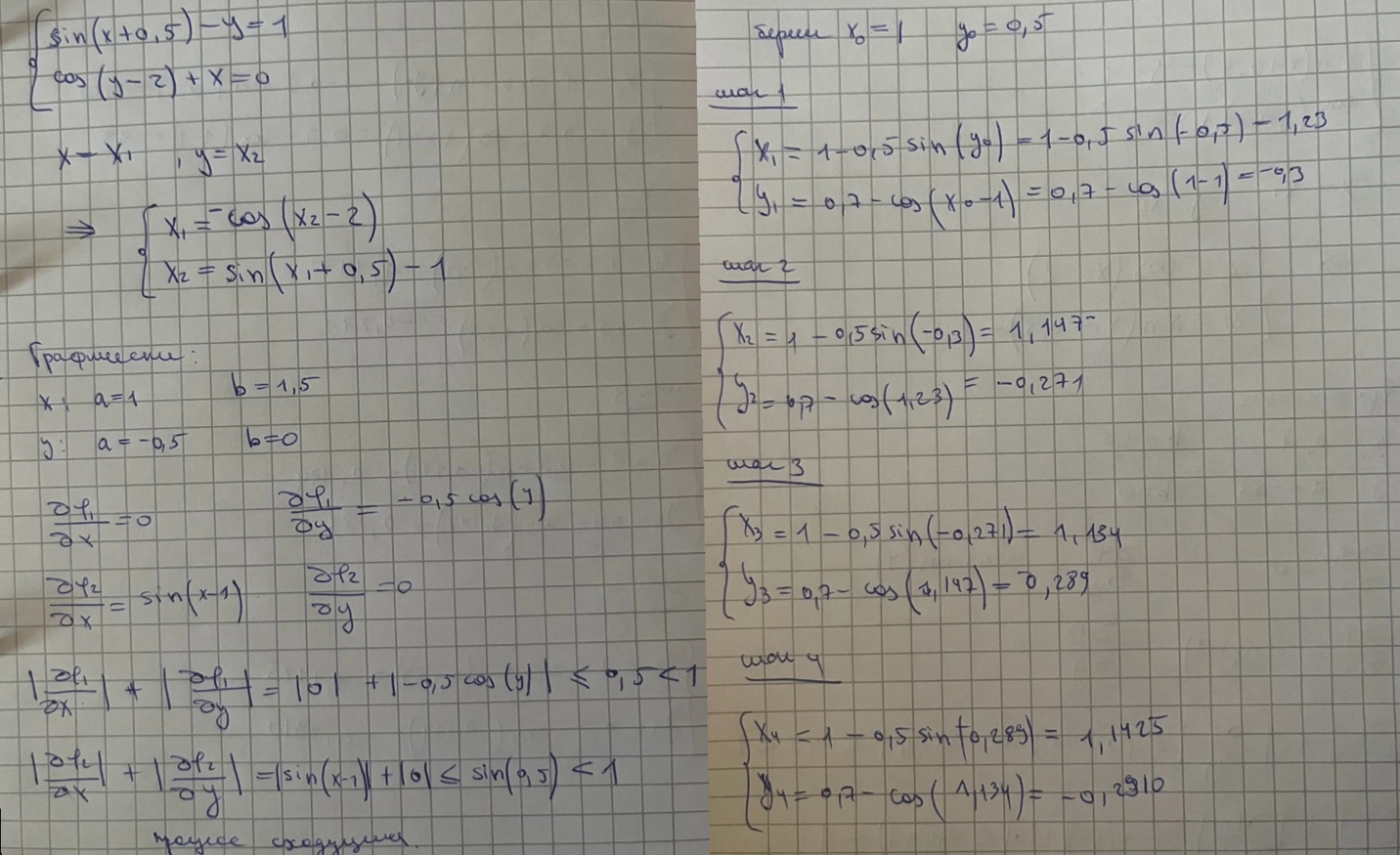
#### 

### Часть 2. Решение системы нелинейных уравнений

### 

#### Метод

Метод простой итерации



## **Программная реализация задачи**

**Решение нелинейных уравнений:**

Метод Хорд

Метод Ньютона

Метод простой итерации

**Решение систем нелинейных уравнений:**

Метод Ньютона

**Листинг программы**

Код написан на языке Python

import numpy as np

import matplotlib.pyplot as plt

from Functions import F1, F2, F3

from NewtonMethod import NewtonMethod

from ChordMethod import ChordsMethod

from SimpleIterationsMethod import SimpleIterationsMethod

from SystemNewtonMethod import SystemNewtonMethod

def main():

print("Введите номер функции, которую хотите решить:")

print("1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458")

print("2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34")

print("3) F3(x) = e^x - 3")

print("4) F4(x, y):")

print(" x^2 + y^2 - 4 = 0")

print(" -3 \* x^2 + y = 0")

function\_choice = int(input())

if function\_choice not in range(1, 5):

raise ValueError("Неверный выбор функции!")

print("Выберите способ решения:")

print("1) Метод Ньютона")

print("2) Метод хорд")

print("3) Метод простой итерации")

print("4) Метод Ньютона для систем")

choice\_method = int(input())

if choice\_method not in range(1, 5):

raise ValueError("Неверный выбор метода!")

print("Как бы вы хотели получить значения границы интервала, "

"начальное приближение к корню (в случае, где оно не высчитывается) и погрешность вычисления?")

print("1 - вручную")

print("2 - из файла")

data\_input = int(input())

if data\_input not in [1, 2]:

raise ValueError("Неверный выбор способа получения данных!")

segment = [0, 0]

EPS = 0

if data\_input == 1:

print("Введите границу отрезка через пробел:")

segment[0], segment[1] = map(float, input().split())

if segment[0] >= segment[1]:

raise ValueError("Начальное значение интервала должно быть меньше конечного значения!")

if (function\_choice == 1 and F1(segment[0]) \* F1(segment[1]) > 0) or \

(function\_choice == 2 and F2(segment[0]) \* F2(segment[1]) > 0) or \

(function\_choice == 3 and F3(segment[0]) \* F3(segment[1]) > 0):

raise ValueError("На заданном интервале нет корня!")

print("Введите погрешность:")

EPS = float(input())

if EPS < 0:

raise ValueError("Некорректно введена погрешность!")

elif data\_input == 2:

with open("test.txt", "r") as file:

segment[0], segment[1], EPS = map(float, file.readline().split())

if choice\_method == 1:

result = NewtonMethod.SolveEquation(segment, EPS, function\_choice)

print("Решение: x =", result)

elif choice\_method == 2:

result = ChordsMethod.SolveEquation(segment, EPS, function\_choice)

print("Решение: x =", result)

elif choice\_method == 3:

result = SimpleIterationsMethod.SolveEquation(segment, EPS, function\_choice)

print("Решение: x =", result)

elif choice\_method == 4:

result = SystemNewtonMethod.SolveEquation(segment[0], segment[1], EPS)

print("Приближенное решение: x =", result[0], ", y =", result[1])

if \_\_name\_\_ == "\_\_main\_\_":

main()

from Functions import F4, F5, DF4\_DX, DF4\_DY, DF5\_DX, DF5\_DY

import numpy as np

class SystemNewtonMethod:

@staticmethod

def SolveEquation(x0, y0, eps):

x = x0

y = y0

iterations = 0

errors = []

dx = (-F4(x, y) \* DF5\_DY() + F5(x, y) \* DF4\_DY(y)) / (DF4\_DX(x) \* DF5\_DY() - DF5\_DX(x) \* DF4\_DY(y))

dy = (-F5(x, y) \* DF4\_DX(x) + F4(x, y) \* DF5\_DX(x)) / (DF4\_DX(x) \* DF5\_DY() - DF5\_DX(x) \* DF4\_DY(y))

x += dx

y += dy

iterations += 1

while abs(dx) > eps and abs(dy) > eps:

dx = (-F4(x, y) \* DF5\_DY() + F5(x, y) \* DF4\_DY(y)) / (DF4\_DX(x) \* DF5\_DY() - DF5\_DX(x) \* DF4\_DY(y))

dy = (-F5(x, y) \* DF4\_DX(x) + F4(x, y) \* DF5\_DX(x)) / (DF4\_DX(x) \* DF5\_DY() - DF5\_DX(x) \* DF4\_DY(y))

x += dx

y += dy

iterations += 1

errors.append(np.sqrt(dx \* dx + dy \* dy))

print("Количество итераций:", iterations)

print("Вектор погрешностей:", errors)

result = [x, y]

return result

class SimpleIterationsMethod:

@staticmethod

def solve\_equation(segment, eps, function\_choice):

def f1(x):

# Implementation of function F1

pass

def f2(x):

# Implementation of function F2

pass

def f3(x):

# Implementation of function F3

pass

def df1(x):

# Implementation of derivative of function F1

pass

def df2(x):

# Implementation of derivative of function F2

pass

def df3(x):

# Implementation of derivative of function F3

pass

functions = {1: (f1, df1), 2: (f2, df2), 3: (f3, df3)}

lambda\_val = -1 / max(functions[function\_choice][1](segment[0]), functions[function\_choice][1](segment[1]))

if 1 + lambda\_val \* functions[function\_choice][1](segment[0]) > 1 or \

1 + lambda\_val \* functions[function\_choice][1](segment[1]) > 1:

raise RuntimeError("Условие сходимости не выполняется.")

x0 = segment[0]

counter = 0

while True:

x = x0 + lambda\_val \* functions[function\_choice][0](x0)

counter += 1

if abs(x - x0) < eps:

print("Корень уравнения:", x)

print("Значение функции в х:", functions[function\_choice][0](x))

print("Количество итераций:", counter)

return x

x0 = x

raise ValueError("Невозможно посчитать начальное приближение!")

class NewtonMethod:

@staticmethod

def find\_root(segment, function\_choice):

def f1(x):

# Implementation of function F1

pass

def f2(x):

# Implementation of function F2

pass

def f3(x):

# Implementation of function F3

pass

def ddf1(x):

# Implementation of second derivative of function F1

pass

def ddf2(x):

# Implementation of second derivative of function F2

pass

def ddf3(x):

# Implementation of second derivative of function F3

pass

functions = {1: (f1, ddf1), 2: (f2, ddf2), 3: (f3, ddf3)}

x0 = 0

if functions[function\_choice][0](segment[0]) \* functions[function\_choice][1](segment[0]) > 0:

x0 = segment[0]

elif functions[function\_choice][0](segment[1]) \* functions[function\_choice][1](segment[1]) > 0:

x0 = segment[1]

elif functions[function\_choice][0](segment[0]) \* functions[function\_choice][1](segment[0]) < 0 and \

functions[function\_choice][0](segment[1]) \* functions[function\_choice][1](segment[1]) < 0:

x0 = abs(segment[0] + segment[1]) / 2.0

else:

raise ValueError("Невозможно выбрать начальное приближение!")

return x0

@staticmethod

def solve\_equation(segment, eps, function\_choice):

def f1(x):

# Implementation of function F1

pass

def f2(x):

# Implementation of function F2

pass

def f3(x):

# Implementation of function F3

pass

def df1(x):

# Implementation of derivative of function F1

pass

def df2(x):

# Implementation of derivative of function F2

pass

def df3(x):

# Implementation of derivative of function F3

pass

functions = {1: (f1, df1), 2: (f2, df2), 3: (f3, df3)}

x = NewtonMethod.find\_root(segment, function\_choice)

counter = 0

while True:

x\_new = x - functions[function\_choice][0](x) / functions[function\_choice][1](x)

counter += 1

if abs(x\_new - x) <= eps:

print("Полученное значение x:", x\_new)

print("Количество итераций:", counter)

print("Значение функции в х:", functions[function\_choice][0](x\_new))

return x\_new

x = x\_new

return x # This line will never be executed, added for completeness

import math

def F1(x):

return x \*\* 3 - 3.125 \* x \*\* 2 - 3.5 \* x + 2.458

def DF1(x):

return 3 \* x \*\* 2 - 6.25 \* x - 3.5

def DDF1(x):

return 6 \* x - 6.25

def F2(x):

return 2 \* x \*\* 3 - 1.89 \* x \*\* 2 - 5 \* x + 2.34

def DF2(x):

return 6 \* x \*\* 2 - 3.78 \* x - 5

def DDF2(x):

return 12 \* x - 3.78

def F3(x):

return math.exp(x) - 3

def DF3(x):

return math.exp(x)

def DDF3(x):

return math.exp(x)

def F4(x, y):

return x \*\* 2 + y \*\* 2 - 4

def F5(x, y):

return -3 \* x \*\* 2 + y

def DF4\_DX(x):

return 2 \* x

def DF4\_DY(y):

return 2 \* y

def DF5\_DX(x):

return -6 \* x

def DF5\_DY():

return 1

class ChordsMethod:

@staticmethod

def find\_root(segment, function\_choice):

def f1(x):

# Implementation of function F1

pass

def f2(x):

# Implementation of function F2

pass

def f3(x):

# Implementation of function F3

pass

if function\_choice == 1:

fa = f1(segment[0])

fb = f1(segment[1])

elif function\_choice == 2:

fa = f2(segment[0])

fb = f2(segment[1])

elif function\_choice == 3:

fa = f3(segment[0])

fb = f3(segment[1])

else:

raise ValueError("Невозможно посчитать начальное приближение!")

if fa \* fb > 0:

raise ValueError("Начальные точки на отрезке не обеспечивают смены знака функции!")

return segment[0] - ((segment[1] - segment[0]) \* fa / (fb - fa))

@staticmethod

def solve\_equation(segment, eps, function\_choice):

def f1(x):

# Implementation of function F1

pass

def f2(x):

# Implementation of function F2

pass

def f3(x):

# Implementation of function F3

pass

x = ChordsMethod.find\_root(segment, function\_choice)

counter = 0

while True:

if function\_choice == 1:

if abs(f1(x)) > eps:

if f1(segment[0]) \* f1(x) < 0:

segment[1] = x

elif f1(segment[1]) \* f1(x) < 0:

segment[0] = x

x = ChordsMethod.find\_root(segment, function\_choice)

counter += 1

else:

break

elif function\_choice == 2:

if abs(f2(x)) > eps:

if f2(segment[0]) \* f2(x) < 0:

segment[1] = x

elif f2(segment[1]) \* f2(x) < 0:

segment[0] = x

x = ChordsMethod.find\_root(segment, function\_choice)

counter += 1

else:

break

elif function\_choice == 3:

if abs(f3(x)) > eps:

if f3(segment[0]) \* f3(x) < 0:

segment[1] = x

elif f3(segment[1]) \* f3(x) < 0:

segment[0] = x

x = ChordsMethod.find\_root(segment, function\_choice)

counter += 1

else:

break

print("Полученное значение x:", x)

print("Количество итераций:", counter)

if function\_choice == 1:

print("Значение функции в х:", f1(x))

elif function\_choice == 2:

print("Значение функции в х:", f2(x))

elif function\_choice == 3:

print("Значение функции в х:", f3(x))

return x