МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №22

Студент:
Самарина Арина Анатольевна, Суржицкий Арсений Арсентьевич
Группа P3266

Преподаватель:
Машина Екатерина Александровна

Санкт-Петербург, 2024

1. Цель работы

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

2. Порядок выполнения работы

- **1 часть: Решение нелинейного уравнения**

Задание:

1. Отделить корни заданного нелинейного уравнения графически

$$f(x) = x^3 - 3{,}78x^2 + 1{,}25x + 3{,}49$$

2. Определить интервалы изоляции корней.
3. Уточнить корни нелинейного уравнения с точностью ε = 10^{-2}
4. Используемые методы для уточнения каждого из 3-х корней многочлена представлены в таблице 7.

Крайний правый корень: метод хорд
Крайний левый корень: метод Ньютона
Центральный корень: метод простой итерации

5. Вычисления оформить в виде таблиц (1-5), в зависимости от заданного метода. Для всех значений в таблице удержать 3 знака после запятой.

Для метода хорд заполнить таблицу 2.
Для метода Ньютона заполнить таблицу 3.
Для метода простой итерации заполнить таблицу 5. Проверить условие сходимости метода на выбранном интервале.

- **2 часть: Решение системы нелинейных уравнений**

Задание:

1. Отделить корни заданной системы нелинейных уравнений графически

$$\begin{cases} \sin(x - y) - xy = -1 \\ 0{,}3x^2 + y^2 = 2 \end{cases}$$

2. Используя метод Ньютона, решить систему нелинейных уравнений с точностью до 0,01.

- **3 часть: Программная реализация**

Для нелинейных уравнений
- Метод хорд
- Метод секущих
- Метод простой итерации

Для систем нелнейных уравнений
- Метод простой интерации

3. Рабочие формулы

Рабочая формула метода хорд:

$$x_{i+1} = x_i - \frac{x_0 - x_i}{f(a) - f(x_i)} f(x_i)$$

Рабочая формула метода Ньютона:

$$x_{i+1} = x_i - \frac{f(i)}{f'(x_i)}$$

Рабочая формула метод секущих

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i)$$

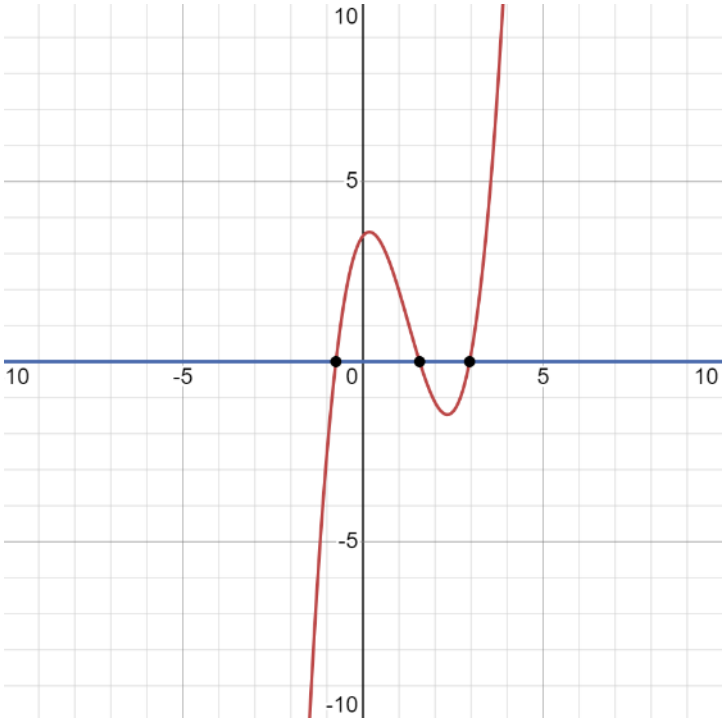
Рабочая формула метода простой интерации:

$x_{i+1} = \varphi(x_i)$, где $x = \varphi(x)$ – эквивалентный вид уравнения $f(x) = 0$

4. Вычислительная часть 1

1 - Отделить корни заданного нелинейного уравнения графически

Функция : $f(x) = x^3 - 3{,}78x^2 + 1{,}25x + 3{,}49$



2 Определить интервалы изоляции корней

Для определения интервалов изоляции корней данного уравнения можно воспользоваться табличным способом. Для этого нужно найти значения функции на различных интервалах и определить знак функции на каждом из них.

x	f(x)
-2	-22,13
-1,5	-10,27
-1	-2,54
-0,5	1,8
0	3,49
0,5	3,3
1	1,96
1,5	0,24
2	-1,13
2,5	-1,39
3	0,22

Знаем, что если непрерывная функция f(x) на концах отрезка [a; b] принимает значения разных знаков, т.е. f(a)·f(b)<0, то на этом отрезке содержится хотя бы один корень уравнения

Тогда мы получаем 3 интервала изоляции корней уравнения: (-1;-0,5), (1.5; 2) и (2.5; 3)

3 - Уточнить корни нелинейного уравнения с точностью ε = 10^{-2}

$x_1 = -0{,}75$

$x_2 = 1.57$

$x_3 = 2.96$

4 - Используемые методы для уточнения каждого из 3-х корней

- Метод половинного деления

№ шага	a	b	x	f(a)	f(b)	f(x)	|a-b|
0	0	2	1	3,49	-1,13	1,96	
1	1	2	1,5	1,96	-1,13	0,24	
2	1,50	2	1,75	0,24	-1,13	-0,54	
3	1,50	1,75	1,63	0,24	-0,54	-0,18	
4	1,50	1,63	1,57	0,24	-0,54	0,01	
5	1,57	1,63	1,60	0,01	-0,54	-0,10	
6	1,57	1,60	1,59	0,01	-0,10	-0,06	
7	1,57	1,59	1,58	0,01	-0,10	-0,03	

$\boldsymbol{x^*} \approx \mathbf{1,57}$

- Метод Ньютона

$x_{i+1} = x_i - \frac{f(i)}{f'(x_i)}$

№ итерации	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}	$\lvert x_{n+1} - x_n \rvert$
0	0	3,49	1,25	-2,79	2,79
1	-2,79	-51,14	45,69	-1,67	4,46
2	-1,67	-13,80	22,24	-1,05	0,62
3	-1,05	-3,15	12,50	-0,80	0,25
4	-0,80	-0,44	9,22	-0,75	0,05
5	-0,75	0,00	8,61	-0,75	0,00

$n = 5$

$\boldsymbol{x^*} \approx \mathbf{-0.75}$

- Метод простой итерации

Преобразуем уравнение к виду

$$x = \varphi(x) = (3{,}78x^2 - 1{,}25x - 3{,}49)^{1/3}$$

$a_0 = 2{,}5, b_0 = 3$

$\varphi'(x) = 0{,}798045 \rightarrow |\varphi'(x)| < 1 \; \forall \; x \in [2{,}5; 3]$

Условие сходимости выполняется

№ итерации	x_i	x_{i+1}	$f(x_{i+1})$	$\lvert x_{n+1} - x_n \rvert$
0	2,50	2,57	-1,29	0,07
1	2,57	2,63	-1,18	0,06
2	2,63	2,69	-1,03	0,06
3	2,69	2,74	-0,89	0,05
4	2,74	2,78	-0,76	0,04
5	2,78	2,81	-0,66	0,03
6	2,81	2,84	-0,54	0,03
7	2,84	2,86	-0,46	0,02
8	2,86	2,88	-0,37	0,02
9	2,88	2,89	-0,33	0,01
10	2,89	2,90	-0,29	0,01
11	2,90	2,91	-0,24	0,01
12	2,91	2,92	-0,19	0,01
13	2,92	2,93	-0,14	0,01
14	2,93	2,94	-0,09	0,01
15	2,94	2,94		

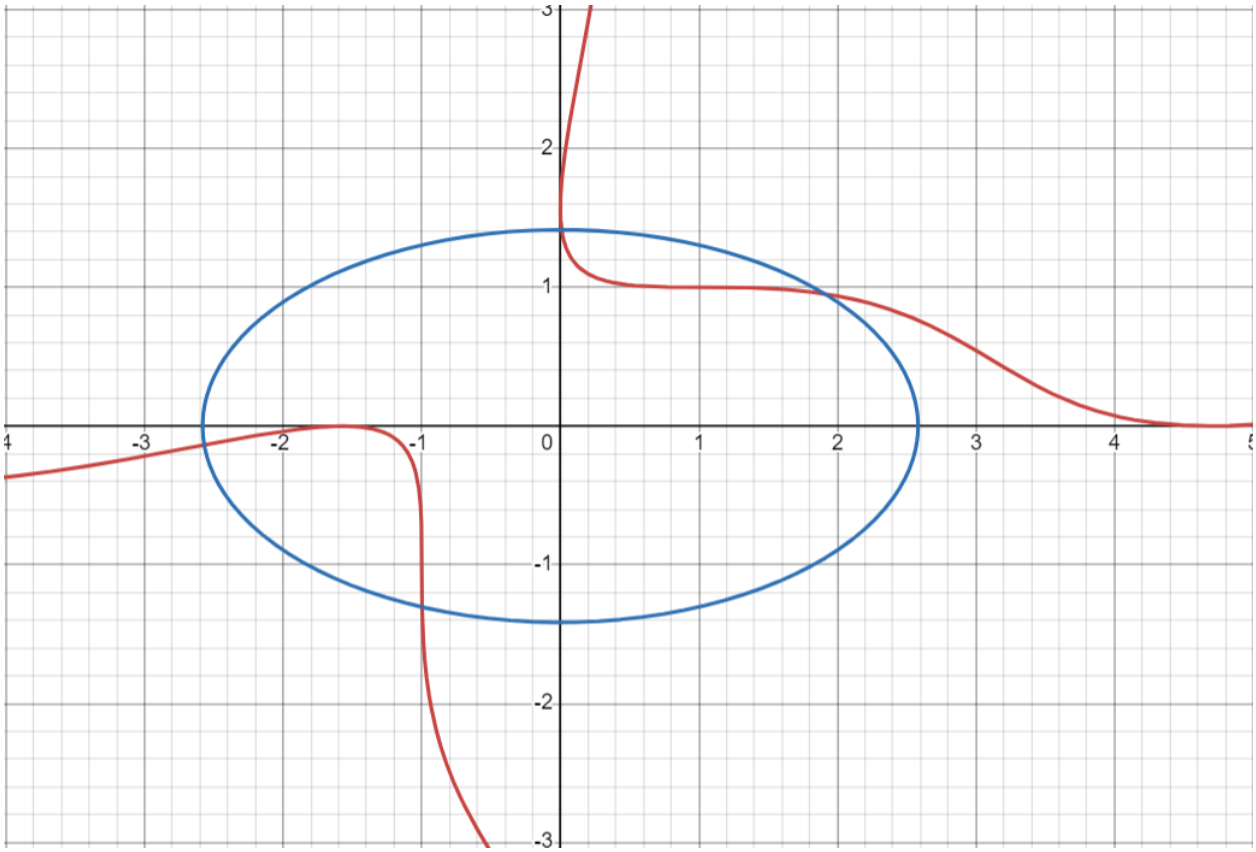
$n = 14$

$\boldsymbol{x^* \approx 2.94}$

5. Вычислительная часть 2

1 - Отделить корни заданной системы нелинейных уравнений графически

$$\begin{cases} sin(x-y) - xy = -1 \\ 0.3x^2 + y^2 = 2 \end{cases}$$



2 – Решить систему нелинейных уравнений с точностью ε = 10^{-2} по методу Ньютона

$$\begin{cases} sin(x-y) - xy = -1 \\ 0.3x^2 + y^2 = 2 \end{cases} \rightarrow \begin{cases} sin(x-y) - xy + 1 = 0 \\ 0.3x^2 + y^2 - 2 = 0 \end{cases}$$

Построим матрицу Якоби

$$\begin{vmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial x} \end{vmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = - \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

$$\rightarrow \begin{vmatrix} \cos(x-y) - y & -\cos(x-y) - x \\ 0.6x & 2y \end{vmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} -sin(x-y) + xy - 1 \\ -0.3x^2 - y^2 + 2 \end{pmatrix}$$

$$\begin{cases} (\cos(x-y) - y)\Delta x - (\cos(x-y) + x)\Delta y = -\sin(x-y) + xy - 1 \\ 0.6x\Delta x + 2y\Delta y = -0.3x^2 - y^2 + 2 \end{cases}$$

Выбираем $x_0 = -3, y_0 = -1$

$$\rightarrow \begin{cases} 0.584\Delta x + 3.416\Delta y = 2.909 \\ -1.8\Delta x - 2\Delta y = -1.7 \end{cases} \rightarrow \begin{cases} \Delta x = -0.002 \\ \Delta y = -0.852 \end{cases}$$

$$x_1 = x_0 + \Delta x = -3.002, \qquad y_0 = y_0 + \Delta y = -1.852$$

Аналогично, получается таблица

№ интерации	x_i	y_i	Δx_i	Δy_i	x_{i+1}	y_{i+1}
0	-3	-1	-0.0022	0.852	-3.0022	-0.1480
1	-3.0022	-0.1480	0.4008	0.0133	-2.6014	-0.1347
2	-2.6014	-0.1347	0.0312	-0.0014	-2.5702	-0.1361
3	**-2.5702**	**-0.1361**	0.0002 < ε	-0.00005 < ε	-	-

$n = 2$

$\boldsymbol{x^* \approx -2.5702}$

$\boldsymbol{y^* \approx -0.1361}$

6. Листинг программы

Main.py

```
from non_lineare_quation import *
from non_lineare_system import *
while True:
    try:
        print("Choose the variant:")
        input_variant = int(
            input(
                "1) Non-linear equation\n2) System of non-linear
equations\n"
            )
        )
        if input_variant in range(1, 3):
            break
        print("Invalid variant")
        continue
    except ValueError:
        print("Incorrect value entered")
        continue

switch_command = {
    1: non_lineare_quation,
    2: non_lineare_system,
    3: exit,
```

```
}
switch_command.get(input_variant, exit)()
```

data_input.py

```
import os
from validate_input import count_roots_on_interval,
quation_solution, validate_roots
import numpy as np
import matplotlib.pyplot as plt


def try_to_convert_to_int(number):
    try:
        number_float = float(number)
        if number_float.is_integer():
            return int(number_float)
        else:
            return number_float
    except ValueError:
        return float(number)


def hand_input(quation, method):
    while True:
        a = choose_left_interval()
        b = choose_right_interval(a)
        count_roots = count_roots_on_interval(quation, a, b,
0.001)
        if not (validate_roots(count_roots)):
            continue
        break
    inaccuracy = choose_inaccuracy()
    return a, b, inaccuracy

def file_input(quation, method):
    current_working_directory = os.path.dirname(__file__)
    file_name = input("Enter the relative path to your file\n")
    print()
    file_path = os.path.join(current_working_directory,
file_name)
    with open(file_path, "r", encoding="utf-8") as file:
    # Читаем строки из файла
        lines = file.readlines()


    if len(lines) == 3:
        data =[]
        for line in lines:
            try:
                var = line.replace(",", ".")
                var = float(var)
                var = try_to_convert_to_int(var)
```

```
                data.append(var)
            except ValueError:
                print("Incorrect data in the specified file")
                exit()

            except UnboundLocalError:
                print("Incorrect data in the specified file")
                exit()
            continue

        a = data[0]
        b = data[1]
        inaccuracy = data[2]
        print("Readed a = ", a, "b = ", b, "inaccuracy = ",
inaccuracy)
        if(float(b) - float(a) - 0.00001 < 0):
            print("The right boundary must be greater than the
left boundary!")
            exit()
        count_roots = count_roots_on_interval(quation, a, b,
0.001)
        if not (validate_roots(count_roots)):
            exit()
    else:
        print("Uncorrect count of lines in the specified file")
        exit()
    return a, b, inaccuracy

def choose_inaccuracy():
    while True:
        try:
            inaccuracy= (
            input(
                "Enter the inaccuracy: "
            ).replace(",", ".")
            )
            inaccuracy = float(inaccuracy)
            return try_to_convert_to_int(inaccuracy)
        except ValueError:
            print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue

def input_selection(quation,method):
    while True:
        try:
            print("Choose the quation:")
            input_selection = int(
                input(
                    "1) Hand input \n2) File input\n"
```

```
                )
            )
            if input_selection in range(1,3):
                break
            print("Invalid command")
            continue
        except ValueError:
            print("Incorrect value entered")
            continue
    switch_command = {
        1: hand_input,
        2: file_input,
    }
    a, b, inaccuracy = switch_command.get(input_selection,
exit)(quation, method)
    return a, b, inaccuracy

def choose_quation():
    while True:
        try:
            print("Choose the quation:")
            input_quation = int(
                input(
                    "1) x^2 - 3x + 2 \n2) x^3 + 2x^2 - 5\n3)
sin(x) - cos(x)\n"
                )
            )
            if input_quation in range(1,4):
                break
            print("Invalid command")
            continue
        except ValueError:
            print("Incorrect value entered")
            continue
    return input_quation

def choose_system():
    while True:
        try:
            print("Choose the quation:")
            input_quation = int(
                input(
                    "1) 0.1x^2 +x + 0.2y^2 - 0.3\n   0.2x^2 + y
+ 0.1xy -0.7\n2) 3y^2+0.5x^2-x\n   sin(x)^2-y\n"
                )
            )
            if input_quation in range(1,3):
                break
            print("Invalid command")
            continue
        except ValueError:
            print("Incorrect value entered")
            continue
```

```
    return input_quation

def choose_method():
    while True:
        try:
            print("Choose the method:")
            input_variant = int(
                input(
                    "1) Chord method\n2) Newton's method\n3)
Simple iteration method\n"
                )
            )
            if input_variant in range(1, 4):
                break
            print("Invalid command")
            continue
        except ValueError:
            print("Incorrect value entered")
            continue
    return input_variant

def choose_system_method():
    while True:
        try:
            print("Choose the method:")
            input_variant = int(
                input(
                    "1) Simple iteration method\n"
                )
            )
            if input_variant in range(1, 2):
                break
            print("Invalid command")
            continue
        except ValueError:
            print("Incorrect value entered")
            continue
    return input_variant

def choose_left_interval():
    while True:
        try:
            interval= (
            input(
                "Enter the left boundary of the interval: "
            ).replace(",", ".")
            )
            interval = int(interval)
            return try_to_convert_to_int(interval)

        except ValueError:
            try:
                interval =try_to_convert_to_int(interval)
```

```
                return interval
            except ValueError:
                print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue


def choose_right_interval(a):
    while True:
        try:
            interval= (
            input(
                "Enter the right boundary of the interval: "
            ).replace(",", ".")
            )
            if(float(interval) - float(a) - 0.00001 < 0):
                print("The right boundary must be greater than
the left boundary!")
                continue
            interval = int(interval)
            return try_to_convert_to_int(interval)

        except ValueError:
            try:
                interval =try_to_convert_to_int(interval)
                return interval
            except ValueError:
                print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue

def choose_x():
    while True:
        try:
            interval= (
            input(
                "Enter the x value: "
            ).replace(",", ".")
            )
            interval = int(interval)
            return try_to_convert_to_int(interval)

        except ValueError:
            try:
                interval =try_to_convert_to_int(interval)
                return interval
            except ValueError:
```

```
                print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue


def choose_y():
    while True:
        try:
            interval= (
            input(
                "Enter the y value: "
            ).replace(",", ".")
            )
            interval = int(interval)
            return try_to_convert_to_int(interval)

        except ValueError:
            try:
                interval =try_to_convert_to_int(interval)
                return interval
            except ValueError:
                print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue

def choose_output_format():
    while True:
        try:
            print("Choose the output format:")
            input_variant = int(
                input(
                    "1) In console\n2) In file\n"
                )
            )
            if input_variant in range(1, 3):
                break
            print("Invalid command")
            continue
        except ValueError:
            print("Incorrect value entered")
            continue
    return input_variant

def output_data(solution, function, iterations, quation):
    input_variant = choose_output_format()
    switch_command = {
        1: output_in_console,
```

```
        2: output_in_file,
    }
    switch_command.get(input_variant, exit)(solution, function,
iterations, quation)


def output_in_console(solution, function, iterations, quation):
    print("Quation: ", quation, "Solution: ", solution, "f(",
solution, ") = ", function, "iterations = ", iterations)

def output_in_file(solution, function, iterations, quation):
    try:
        filename = input("Enter the file name to save the data:
")

        file_path = os.path.join('./lb2/solutions', filename)

        if not os.path.exists('./lb2/solutions'):
            os.makedirs('./lb2/solutions')

        with open(file_path, 'w') as file:
            file.write("Quation: " + str(quation) + '\n')
            file.write("Solution: " + str(solution) + '\n')
            file.write("f("+ str(solution)+ ") = " +
str(function) + '\n')
            file.write("iterations = " + str(iterations) + '\n')

        print(f"The values of variables have been successfully
written to the file: {file_path}")

    except FileNotFoundError:
        print("The specified path does not exist.")
    except PermissionError:
        print("You do not have permission to create a file in
this directory.")
def draw_grapth(quation,function, a, b):
    x_values = np.linspace(-a-a*0.3, b+b*0.3, 100)
    y_values = quation_solution(quation, x_values)

    # Построим график функции
    plt.plot(x_values, y_values, label=function, color='b')

    # Добавим заголовок и метки осей
    plt.title('График функции ' + function)
    plt.xlabel('x')
    plt.ylabel('f(x)')

    # Добавим сетку
    plt.grid(True)

    # Отобразим линии вспомогательных осей
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)
```

```
    # Добавим легенду
    plt.legend()

    # Отобразим график
    plt.show()
```

non_lineare_quation.py

```
from data_imput import *
from validate_input import check_convergencecondition,
converted_quation, quation_df2_solution, quation_df_solution,
quation_solution

def str_quation(quation):
    if quation == 1:
        return "x^2 - 3x + 2"
    elif quation == 2:
        return "x^3 + 2x^2 - 5"
    elif quation == 3:
        return "cos(x) + x^2"

def validate_initial_approximation(quation,a,b):
    fa = quation_solution(quation,a)
    fb =  quation_solution(quation,b)
    dfa2 =quation_df2_solution(quation,a)
    dfb2 = quation_df2_solution(quation,b)

    if fa * dfa2 > 0:
        return a
    elif fb * dfb2 > 0:
        return b
    else:
        raise ValueError("No suitable initial approximation
found on the interval [a, b]")

def Chord_method(quation, method):
    a, b, inaccuracy = input_selection(quation,method)
    a1 =a
    b1 = b
    iterations = 0
    max_iter = 10000
    x0 = 99999
    while abs(quation_solution(quation, x0)) > inaccuracy and
iterations < max_iter:
        x0 = a1 - ((b1-a1)/(quation_solution(quation,b1)-
quation_solution(quation,a1)))*quation_solution(quation,a1)
        if quation_solution(quation, x0) *
quation_solution(quation, a1) < 0:
            b1 = x0
        else:
            a1 = x0
        iterations += 1
```

```
    solution = try_to_convert_to_int(x0)

    output_data(solution, quation_solution(quation, solution),
iterations, str_quation(quation))
    draw_grapth(quation, str_quation(quation), a1, b1)

def Newton_method(quation, method):
    a, b, inaccuracy = input_selection(quation,method)

    approximation = validate_initial_approximation(quation, a,
b)

    iterations = 0
    max_iter = 1000
    x = approximation
    while abs(quation_solution(quation,x)) > inaccuracy and
iterations < max_iter:
        x = x - quation_solution(quation,x) /
quation_df_solution(quation,x)
        iterations += 1

    solution = try_to_convert_to_int(x)
    output_data(solution, quation_solution(quation, solution),
iterations, str_quation(quation))
    draw_grapth(quation, str_quation(quation), a, b)

def Simple_iteration_method(quation, method):
    a, b, inaccuary = input_selection(quation,method)
    # lambda_L = -1/(max(abs(quation_df_solution(quation,a)),
abs(quation_df_solution(quation,b))))
    # print("L = ",lambda_L)
    q = check_convergencecondition(quation, a, b)
    approximation = validate_initial_approximation(quation, a,
b)
    x = approximation
    iterations = 0
    max_iter = 1000
    if q>1:
        print("The convergence condition is NOT met")
        exit()
    elif( 0< q <= 0.5):
        while abs(converted_quation(quation,x)-x) > inaccuary
and iterations < max_iter:
            x = converted_quation(quation,x)
            print(x)
            iterations += 1
    elif(0.5 <q <1):
        while abs(converted_quation(quation,x)-x) > ((1-
q)/q)*inaccuary and iterations < max_iter:
            x = converted_quation(quation,x)
            iterations += 1
```

```
    solution = try_to_convert_to_int(x)
    output_data(solution, quation_solution(quation, solution),
iterations, str_quation(quation))
    draw_grapth(quation, str_quation(quation), a, b)


def non_lineare_quation():
    quation = choose_quation()
    method = choose_method()


    switch_command = {
        1: Chord_method,
        2: Newton_method,
        3: Simple_iteration_method,
        4: exit,
    }
switch_command.get(method, exit)(quation,method)
```

non_lineare_system.py

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from data_imput import
choose_inaccuracy,choose_x,choose_y,choose_system,choose_system_
method
def non_lineare_system():
    system_num = choose_system()
    method = choose_system_method()


    switch_command = {
        1: simple_iteration_method,
        2: exit,
    }
    switch_command.get(method, exit)(system_num,method)

def simple_iteration_method(system,method) :
    x0 = choose_x()
    y0 = choose_y()
    a = x0
    b = y0
    inaccuracy = choose_inaccuracy()
    max_iter = 1000
    iteration =0
```

```
    if not check_convergence(system, method, x0, y0):
        print("The iteration matrix does not\nsatisfy the
convergence condition.")
        exit()

    for i in range(max_iter):
        x = f1(x0, y0,system)
        y = f2(x0, y0,system)
        if (abs(x-x0)< inaccuracy)and(abs(y-y0)< inaccuracy):
            print(f"\n\nx = {x}\ny = {y}")
            print(f"Iterations = {iteration}")
            print(f"inncaury x = {x-f1(x,y,system)}\ninncaury y
= {y-f2(x,y,system)}")
            break
        x0, y0 = x, y
        iteration += 1


    plt.figure(figsize=(12, 6))

    x_values = np.linspace(-5, 5, 1000)
    y1_pos = np.array([f1_y_positive(x, system) for x in
x_values])
    y1_neg = np.array([f1_y_negative(x, system) for x in
x_values])
    y2_pos = np.array([f2_y_positive(x, system) for x in
x_values])
    y2_neg = np.array([f2_y_negative(x, system) for x in
x_values])
    plt.subplot(1, 2, 1)
    plt.plot(x_values, y1_pos, label=f'{str_quation_1(system)}
(+)', color='r')
    plt.plot(x_values, y1_neg, label=f'{str_quation_1(system)}
(-)',  color='r')
    plt.plot(x_values, y2_pos, label=f'{str_quation_2(system)}
(+)', color='b')
    plt.plot(x_values, y2_neg, label=f'{str_quation_2(system)}
(-)', color='b')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.axhline(0, color='black', linewidth=1.5)
    plt.axvline(0, color='black', linewidth=1.5)
    plt.title(f'System {system}')
    plt.grid(True)
    plt.legend()

    plt.tight_layout()
    plt.show()

def f1(x,y,system):
    if(system == 1):
        return 0.3 - 0.1*x*x- 0.2*y*y
    elif(system == 2):
```

```
        return 3*y*y + 0.5*x*x
    else:
        print("System out of choice")
        exit()
def f2(x,y,system):
    if(system == 1):
        return 0.7 -0.2*x*x - 0.1*x*y
    elif(system == 2):
        return np.sin(x)**2
    else:
        print("System out of choice")
        exit()
def check_convergence(system, method, x0, y0):
    jacobian_matrix = np.array([[f1_dx(system, x0, y0),
f1_dy(system, x0, y0)],
                                [f2_dx(system, x0, y0),
f2_dy(system, x0, y0)]])
    eigenvalues = np.linalg.eigvals(jacobian_matrix)
    if np.all(np.abs(eigenvalues) < 1):
        return True
    else:
        return False

def f1_dx(system, x, y):
    if system == 1:
        return -0.2 * x
    elif system == 2:
        return x
    else:
        print("System out of choice")
        exit()

def f1_dy(system, x, y):
    if system == 1:
        return -0.4 * y
    elif system == 2:
        return 6*y
    else:
        print("System out of choice")
        exit()

def f2_dx(system, x, y):
    if system == 1:
        return -0.4 * x - 0.1 * y
    elif system == 2:
        return np.sin(2*x)
    else:
        print("System out of choice")
        exit()

def f2_dy(system, x, y):
    if system == 1:
        return 0
```

```
    elif system == 2:
        return 0
    else:
        print("System out of choice")
        exit()
def f1_y_positive(x, system):
    if system == 1:
        argument = (0.3 - x - 0.1 * x * x) / 0.2
        return np.sqrt(argument) if argument >= 0 else np.nan
    elif system == 2:
        argument = (x - 0.5 * x * x) / 3
        return np.sqrt(argument) if argument >= 0 else np.nan
    else:
        print("System out of choice")
        exit()

def f1_y_negative(x, system):
    if system == 1:
        argument = (0.3 - x - 0.1 * x * x) / 0.2
        return -np.sqrt(argument) if argument >= 0 else np.nan
    elif system == 2:
        argument = (x - 0.5 * x * x) / 3
        return -np.sqrt(argument) if argument >= 0 else np.nan
    else:
        print("System out of choice")
        exit()

def f2_y_positive(x, system):
    if system == 1:
        argument = (0.7 - 0.2 * x * x) / (1 + 0.1 * x)
        return argument
    elif system == 2:
        return np.abs(np.sin(x))
    else:
        print("System out of choice")
        exit()

def f2_y_negative(x, system):
    if system == 1:
        argument = (0.7 - 0.2 * x * x) / (1 + 0.1 * x)
        return argument
    elif system == 2:
        return np.abs(np.sin(x))
    else:
        print("System out of choice")
        exit()
def str_quation_1(system):
    if system == 1:
        return "0.1x^2 +x + 0.2y^2 - 0.3"
    elif system == 2:
        return "3y^2+0.5x^2-x"
def str_quation_2(system):
    if system == 1:
```

```
        return "0.2x^2 + y + 0.1xy -0.7"
    elif system == 2:
        return "sin(x)^2-y"
```

validate_input.py

```
from math import cos, sin

import numpy as np

def quation_solution(quation,x):
    if quation == 1:
        return x**2 - 3*x + 2
    elif quation == 2:
        return x**3 + 2*(x**2) - 5
    elif quation == 3:
        return np.cos(x) - x**2
    else:
        print("Unknown quation")
        exit()

def quation_df_solution(quation, x):
    if quation == 1:
        return 2*x - 3
    elif quation == 2:
        return 3*(x**2) + 4*x
    elif quation == 3:
        return -2*x - sin(x)
    else:
        print("Unknown quation")
        exit()

def quation_df2_solution(quation, x):
    if quation == 1:
        return 2
    elif quation == 2:
        return 6*x +4
    elif quation == 3:
        return -2 - cos(x)
    else:
        print("Unknown quation")
        exit()

def converted_quation(quation,x):
    if quation == 1:
        return x + (x**2 - 3*x + 2)*0.1
    elif quation == 2:
        return x + (x**3 + 2*x**2 - 5)*0.1
    elif quation == 3:
        return x + (cos(x)- (x**2))*(0.1)
    else:
        print("Unknown quation")
        exit()
```

```
def convertred_df_quation(quation, x):
    if quation == 1:
        return x/5 + 0.7
    elif quation == 2:
        return (3*x**2)/(10)+(2*x)/5+1
    elif quation == 3:
        return -(sin(x))/(10)-x/5+1
    else:
        print("Unknown quation")
        exit()

def check_convergencecondition(quation, a, b):
    print("q(a) = ", convertred_df_quation(quation, a), "q(b) =
", convertred_df_quation(quation, b))
    return max(abs(convertred_df_quation(quation, a)),
abs(convertred_df_quation(quation, b)))

def count_roots_on_interval(quation, a, b, inaccuracy):
    b1 = b
    num_roots = 0
    x = a
    while x < b1:
        if quation_solution(quation, x) *
quation_solution(quation, x + inaccuracy) < 0:
            num_roots += 1
        x += inaccuracy
    return num_roots
def validate_roots(num_roots):
    num_roots = int(num_roots)
    if num_roots <1:
        print("No roots")
        return False
    elif num_roots > 1:
        print("More than one root (", num_roots, ")")
        return False
    return True
```

7. Выводы

В ходе лабораторной работы были изучены и реализованы численные методы решения нелинейных уравнений и их систем. Программа успешно находила корни, подтвердив эффективность методов Ньютона, простых итераций и бисекции. Тестирование показало точность и стабильность результатов. Работа подчеркнула важность выбора метода и начальных приближений, создавая основу для дальнейшего использования в численном анализе и математическом моделировании.