ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет программной инженерии и компьютерной техники

Дисциплина:
«Вычислительная математика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
Вариант 9.

Выполнил:
Студент гр. P32151
Понамарев Степан Андреевич

Проверил:
Машина Екатерина Алексеевна

Санкт-Петербург
2023г.

1. **Цель лабораторной работы**: изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, вы-полнить программную реализацию методов.

2. **Задание лабораторной работы**:

Задание:

1. Отделить корни заданного нелинейного уравнения графически (вид уравнения представлен в табл. 6)
2. Определить интервалы изоляции корней.
3. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью $\varepsilon=10^{-2}$.
4. Используемые методы для уточнения каждого из 3-х корней многочлена представлены в таблице 7.
5. Вычисления оформить в виде таблиц (1-5), в зависимости от заданного метода. Для всех значений в таблице удержать 3 знака после запятой.
   - 5.1 Для метода половинного деления заполнить таблицу 1.
   - 5.2 Для метода хорд заполнить таблицу 2.
   - 5.3 Для метода Ньютона заполнить таблицу 3.
   - 5.4 Для метода секущих заполнить таблицу 4.
   - 5.5 Для метода простой итерации заполнить таблицу 5.
6. Заполненные таблицы отобразить в отчете.

3. **Рабочий метод по варианту и рабочие формулы (вариант 9)**:
   1. Крайний левый корень: **Метод простой итерации**.
      - $x_{i+1} = \varphi(x_i)$
      - $x = \varphi(x)$ получается приведением $f(x)$ к эквивалентному виду
      - x_0 определяется начальным приближением
      - Критерий окончания итерационного процесса: $|x_n - x_{n-1}| < \varepsilon$
   2. Центральный корень: **Метод секущих**.
      - $x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i)$
      - Выбор x_0: $x_0 = a_0$ если $f(a_0) \cdot f''(a_0) > 0$, иначе $x_0 = b_0$
      - x_1 выбирается рядом с x_0
   3. Крайний правый корень: **Метод половинного деления**.
      - $x_0 = \frac{a_0 + b_0}{2}$
      - $x_{i+1} = \frac{a_i + b_i}{2}$
      - $a_{i+1} = x_i$ и $b_{i+1} = b_i$, если $f(x_i) \cdot f(a_0) > 0$, иначе $a_{i+1} = a_i$ и $b_{i+1} = x_i$
4. **Вычислительная реализация задачи**:
   1. Отделить корни заданного нелинейного уравнения графически

Вид уравнения: $y = -1.8\,x^3 - 2.94\,x^2 + 10.37\,x + 5.38$

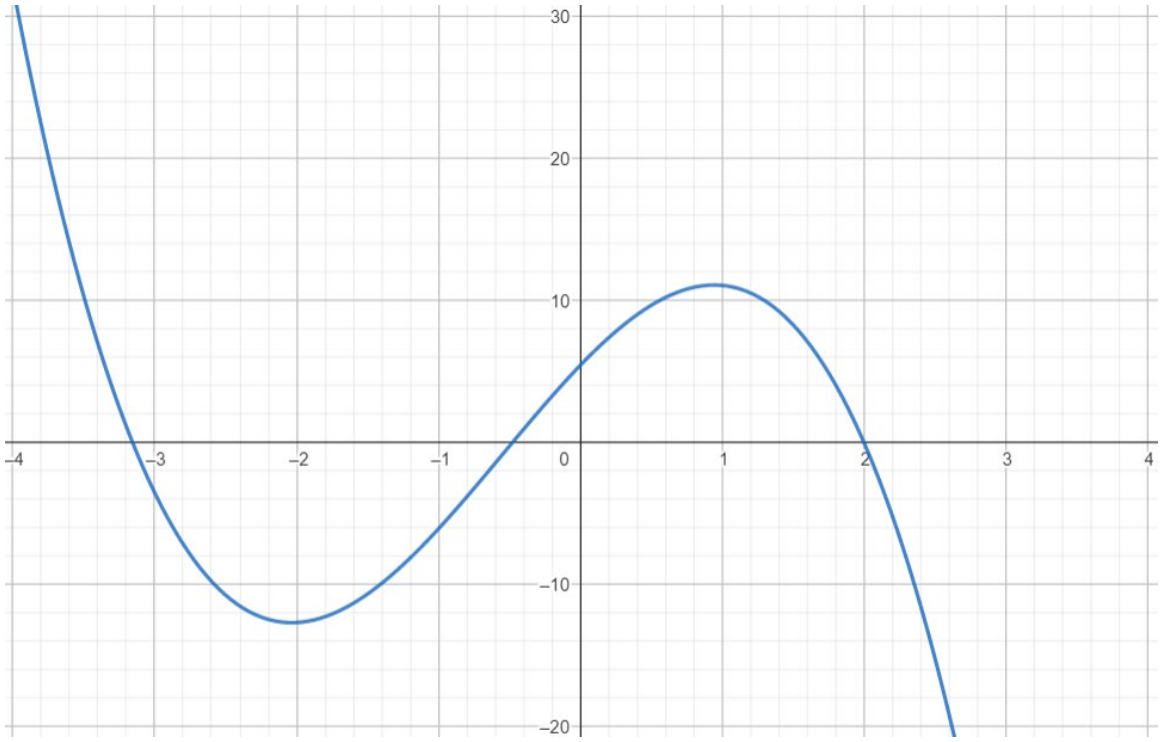

Рис. 1. График нелинейного уравнения

2. Интервалы изоляции корней:
   I. $x_1 \in [-3.4, -3]$
   II. $x_2 \in [-0.8, -0.2]$
   III. $x_3 \in [1.8, 2.2]$
3. Уточнение корней нелинейного уравнения с точностью $\varepsilon = 10^{-2}$.
   I. $x_1 \approx -3.157$
   II. $x_2 \approx -0.473$
   III. $x_3 \approx 1.998$
4. Таблицы вычислительной части лабораторной работы:
   I. Метод простой итерации для крайнего левого корня (x_1):
   - $f(x) = -1.8\,x^3 - 2.94\,x^2 + 10.37\,x + 5.38$
   - $\varphi(x) = \sqrt[3]{\frac{-2.94x^2+10.37x+5.38}{1.8}}$
   - $\varphi'(x) = \frac{1}{5.4}(-2.94x^2 + 10.37x + 5.38)^{-\frac{2}{3}}(5.88x + 10.37)$
   - $|\varphi'(-3.4)| = 0{,}297 < 1$, $|\varphi'(-3)| = 0{,}256 < 1$, условие сходимости выполняется.
   - $x_0 = -3$ – начальное приближение.
   - $\varphi(-3) = -2{,}076$

Таблица 1

№ итерации	x_k	x_{k+1}	$\varphi(x_{k+1})$	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$

1	-3,000	-2,076	-1,703	-12,714	0,924
2	-2,076	-1,703	-1,528	-11,918	0,373
3	-1,703	-1,528	-1,438	-10,909	0,175
4	-1,528	-1,438	-1,388	-10,257	0,090
5	-1,438	-1,388	-1,360	-9,867	0,049
6	-1,388	-1,360	-1,344	-9,637	0,028
7	-1,360	-1,344	-1,335	-9,501	0,016
8	-1,344	-1,335	-1,329	-9,420	0,009

II. Метод секущих для центрального корня (x_2):

- $a = -0.8$, $b = -0.2$
- $f(x) = -1.8\,x^3 - 2.94\,x^2 + 10.37\,x + 5.38$
- $f''(x) = -10{,}8x - 5{,}88$
- $f(a) \cdot f''(a) \approx -3{,}876 < 0$ – не подходит, значит берём $x_0 = b = -0.2$
- $x_1 = -0.25$ – точка рядом с x_0
- $f(-0.2) \approx 3.203$, $f(-0.25) \approx 2.632$

Таблица 2

№ итерации	x_{k-1}	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1	-0,200	-0,250	-0,480	-0,082	0,230
2	-0,250	-0,480	-0,474	0,001	0,007

III. Метод половинного деления для крайнего правого корня (x_3):

$a_0 = 1.8$

$b_0 = 2.2$

Таблица 3

№ шага	a	b	x	$f(a)$	$f(b)$	$f(x)$	$\lvert a - b \rvert$
1	1,800	2,200	2,000	4,023	-5,202	-0,040	0,400
2	1,800	2,000	1,900	4,023	-0,040	2,123	0,200
3	1,900	2,000	1,950	2,123	-0,040	1,075	0,100
4	1,950	2,000	1,975	1,075	-0,040	0,526	0,050
5	1,975	2,000	1,988	0,526	-0,040	0,245	0,025
6	1,988	2,000	1,994	0,245	-0,040	0,103	0,013
7	1,994	2,000	1,997	0,103	-0,040	0,032	0,006

5. Листинг программы (коды используемых методов):

main.py:

```
import numpy as np
```

```
from AnyEquationSolver import EquationSolver
from EquationManager import Equation
from Exceptions import InvalidAlgebraicEquationException
from InputManager import InputManager
from PreparedEquationManager import PreparedEquation
from PreparedSystemManager import PreparedSystem


def variants_function_solver():
    result = None
    variants = ["3x^3 + 1.7x^2 - 15.42x + 6.89",
                "x • sin(x) - 1",
                "e^sin(x) • ln(|x|) - 1.4",
                "arcsin(e^(-x)) - 0.3",
                "tg(x/2 • sin(x))"]
    values = [PreparedEquation(lambda x: 3 * x ** 3 + 1.7 * x **
2 - 15.42 * x + 6.89,
                               derivative_func=lambda x: -15.42
+ 3.4 * x + 9 * x ** 2),
              PreparedEquation(lambda x: x * np.sin(x) - 1,
                               derivative_func=lambda x:
np.sin(x) + x * np.cos(x)),
              PreparedEquation(lambda x: np.exp(np.sin(x)) *
np.log(abs(x)) - 1.4,
                               derivative_func=lambda x:
np.exp(np.sin(x)) * (
                                                   abs(x) * np.log(abs(x)) *
np.cos(x) + np.sign(x)) / abs(x)),
              PreparedEquation(lambda x: np.arcsin(np.exp(-x)) -
0.3,
                               derivative_func=lambda x: -
np.exp(-x) / np.sqrt(1 - np.exp(-2 * x))),
              PreparedEquation(lambda x: np.tan(x / 2 *
np.sin(x)),
                               derivative_func=lambda x:
(np.sin(x) + x * np.cos(x)) /
                                                                   (2 *
np.cos(x * np.sin(x) / 2) ** 2))]

    chosen_equation =
InputManager.multiple_choice_input(variants, values, "Выберите
функцию:")
    try:
        result = chosen_equation.solve()
    except ValueError:
        print("Возможно, функция не определена в некоторых
точках.\n"
              "Попробуйте ввести другие концы отрезка.")
    if result is None:
        print("Не удалось найти корень уравнения.")
    else:
        print("------------РЕШЕНИЕ------------")
        print("Найденный корень уравнения:", round(result, 5))
```

```
        print("Значение функции в точке: ",
round(chosen_equation.calculate(result), 9))
        print("Количество итераций:",
chosen_equation.iterations)
        if InputManager.yes_or_no_input("Показать график
функции?"):
            chosen_equation.draw_graphic(dot_x=result,
dot_y=chosen_equation.calculate(result))


def any_function_solver():
    try:
        line = InputManager.algebraic_expression_input("Введите
уравнение: ")
        eq = Equation(line)
        EquationSolver.solve(eq)
    except InvalidAlgebraicEquationException:
        print("Не удалось распарсить выражение")
        print('Попробуйте не использовать знак "=".')


def system_solver():
    name1 = "┌ 0.1x^2 + x + 0.2y^2 - 0.3" + "\n" + \
            "└ 0.2x^2 + y + 0.1xy - 0.7" + "\n" + \
            "Пример корня: (0.20 0.67)"
    ps1 = PreparedSystem([lambda x, y: 0.1 * x ** 2 + x + 0.2 *
y ** 2 - 0.3,
                          lambda x, y: 0.2 * x ** 2 + y + 0.1 *
x * y - 0.7],
                         derivative_funcs=[lambda x, y:
np.array([-0.2 * x, -0.4 * y]),
                                           lambda x, y:
np.array([-0.4 * x - 0.1 * y, -0.1 * x])])
    name2 = "┌ sin(x)" + "\n" + \
            "└ xy/20 - 0.5" + "\n" + \
            "Пример корня: (6.28 1.59)"
    ps2 = PreparedSystem([lambda x, y: np.sin(x),
                          lambda x, y: x * y / 20 - 0.5],
                         derivative_funcs=[lambda x, y:
np.array([np.cos(x), 0]),
                                           lambda x, y:
np.array([y / 20, x / 20])])
    chosen_system = InputManager.multiple_choice_input([name1,
name2], [ps1, ps2], "Выберите систему:")

    result = chosen_system.solve()
    if result is None:
        print("Не удалось найти корень системы.")
    else:
        print("------------РЕШЕНИЕ------------")
        print("Найденный корень системы:", result)
        print("Значение системы в этой точке:",
chosen_system.calculate(*result))
```

```
        print("Количество итераций:", chosen_system.iterations)
        print("Вектор погрешностей:",
chosen_system.get_error_rate())


if __name__ == '__main__':
    variants = ["Выбрать функцию и метод из предложенных",
                "Ввести произвольную функцию и решить её методом
половинного деления",
                "Выбрать систему уравнений из предложенных и
решить её методом простых итераций"]
    values = [variants_function_solver, any_function_solver,
system_solver]

    chosen_variant =
InputManager.multiple_choice_input(variants, values, "Чем хотите
себя побаловать?")

    chosen_variant()
```

InputManager.py:

```
import numpy as np

from Exceptions import InvalidAlgebraicEquationException


class InputManager:
    @staticmethod
    def string_input(message=""):
        buf = ""
        while buf == "":
            buf = input(message).strip()
        return buf

    @staticmethod
    def _check_number(buf):
        try:
            float(buf.replace(',', '.'))
            return True
        except ValueError:
            return False

    @staticmethod
    def _convert_to_number(num):
        try:
            return float(num.replace(',', '.'))
        except ValueError:
            return None

    @staticmethod
    def float_input(message=""):
        number = None
```

```
        while number is None:
            number =
InputManager._convert_to_number(InputManager.string_input(messag
e))
        return number

    @staticmethod
    def int_input(message=""):
        return int(InputManager.float_input(message))

    @staticmethod
    def yes_or_no_input(message=""):
        answer = "0"
        while answer[0].lower() not in ["y", "n", "д", "н"]:
            answer = InputManager.string_input(message + "
[y/n]: ")
        return answer[0].lower() in ["y", "д"]

    @staticmethod
    def matrix_input():
        lines_number = InputManager.int_input("Введите
количество строк матрицы: ")
        print("Введите матрицу вида\n"
              "a_11, a_12, ..., a_1n, b_1\n"
              "...\n"
              "a_n1, a_n2, ..., a_nn, b_n\n")
        matrix = [[0] * (lines_number + 1) for _ in
range(lines_number)]
        for i in range(lines_number):
            line = InputManager.string_input().split()
            while len(line) != lines_number + 1 or not
all([InputManager._check_number(t) for t in line]):
                print("Некорректный ввод строки")
                line = InputManager.string_input().split()
            for j in range(lines_number + 1):
                matrix[i][j] =
InputManager._convert_to_number(line[j])
        print()
        return np.array(matrix)

    """
    x => x = 0
    ln(x) => ln(x) = 0
    log_2(x)
    sin(x)
    x^2 - x
    cos(x)^2 + sin(x)^2 - 1
    cos(x)^2+sin(x)^2-1
    cos(x_1)^2)+sin(x_2)^2 = 0

    нижний индекс поддерживается только у:
        логарифма
        икса (сначала идёт индекс, потом степень - x_1^2
```

```
    """

    @staticmethod
    def algebraic_expression_input(message=""):
        line = InputManager.string_input(message)
        line = line.replace(" ", "")
        line = line.replace(",", ".")
        line = line.replace("^", "**")
        if line.__contains__('='):
            raise InvalidAlgebraicEquationException
        return line

        pass

    @staticmethod
    def vector_dict_input(var_names, message=""):
        if message != "":
            print(message)
        result = dict()
        for name in var_names:
            result[name] = InputManager.float_input(f"\tВведите
{name}: ")
        return result

    @staticmethod
    def enum_input(variants_list, message):
        buf = ""
        while buf not in variants_list:
            buf = InputManager.string_input(message)
        return buf

    @staticmethod
    def multiple_choice_input(variant_list, values_list,
message=""):
        n = len(variant_list)
        if n == 0 or len(variant_list) != len(values_list):
            raise ValueError

        if message != "":
            print(message)

        for i in range(n):
            s = f"{i + 1}."
            lines = variant_list[i].split('\n')
            print('\t' + s, lines[0])
            for line in lines[1:]:
                print("\t" + ' ' * len(s), line)

        i = int(InputManager.enum_input([str(i) for i in
range(1, n + 1)], f"Введите число от 1 до {n}: ")) - 1
        return values_list[i]
```

PreparedEquationManager.py:

```
import numpy as np
from matplotlib import pyplot as plt

from InputManager import InputManager


class PreparedEquation:

    def __init__(self, func, derivative_func=None):
        self.func = func
        self.derivative_func = derivative_func
        self.iterations = 0

    def calculate(self, arg):
        try:
            return self.func(arg)
        except Exception:
            raise ValueError

    def solve(self):
        variants = ["Метод хорд", "Метод Ньютона", "Метод
простых итераций"]
        values = [self.Chord_Method, self.Newtons_Method,
self.Simple_Iteration_Method]
        chosen_method =
InputManager.multiple_choice_input(variants, values, "Выберите
метод решения уравнения: ")
        return chosen_method()

    def _enter_root_isolation(self):
        print("Введите интервал изоляции корня:")
        left = InputManager.float_input("\tЛевый конец отрезка:
")
        right = InputManager.float_input("\tПравый конец
отрезка: ")

        while not self.calculate(left) * self.calculate(right) <
0:
            print("Значения функции на концах отрезка должны
быть разного знака!")
            if InputManager.yes_or_no_input(f"Показать график
функции для корректировки?"):
                self.draw_graphic(left, right)
            print("Введите другие значения концов:")
            left = InputManager.float_input("\tЛевый конец
отрезка: ")
            right = InputManager.float_input("\tПравый конец
отрезка: ")
        if self.calculate(left) > 0:
            # функция принимает значение > 0 на правом конце, и
```

```
< 0 на левом
            # от left до right функция возрастает
            left, right = right, left
        self.left, self.right = left, right
        return left, right

    def draw_graphic(self, left=None, right=None, dot_x=None,
dot_y=None):
        if left is None or right is None:
            if self.left is None or self.right is None:
                print("Невозможно нарисовать график. Не
определены концы отрезка")
            else:
                left = self.left
                right = self.right
        grid = abs(left - right) / 30
        x_axis = np.linspace(min(left, right) - grid, max(left,
right) + grid, 32)
        plt.plot(x_axis, self.calculate(x_axis))
        plt.grid(True, which='both')
        if (min(left, right) - grid <= 0 and 0 <= max(left,
right) + grid):
            plt.axvline(x=0, color='k')
        plt.axhline(y=0, color='k')
        if dot_x is not None:
            if dot_y is None:
                dot_y = 0
            plt.plot(dot_x, dot_y, "ro")
        plt.show()

    def Chord_Method(self):
        left, right = self._enter_root_isolation()

        # основная часть решения
        epsilon = 10 ** (-8)
        x = left - self.calculate(left) * (right - left) /
(self.calculate(right) - self.calculate(left))
        while abs(self.calculate(x)) > epsilon:
            x = left - self.calculate(left) * (right - left) /
(self.calculate(right) - self.calculate(left))
            if self.calculate(x) > 0:
                right = x
            else:
                left = x
            self.iterations += 1

        return x

    def Simple_Iteration_Method(self):
        left, right = self._enter_root_isolation()
        halflife = -0.01 / max(abs(self.derivative_func(left)),
abs(self.derivative_func(right)))
        phi = lambda x: x + halflife * self.calculate(x)
```

```
        print(f"Используемая лямбда={halflife}")

        # основная часть решения
        epsilon = 10 ** (-8)
        x = InputManager.float_input("Введите начальное
приближение: ")
        while not (min(left, right) < x and x < max(left,
right)):
            x = InputManager.float_input("Введите начальное
приближение внутри интервала изоляции корня: ")
        try:
            self.calculate(x)
        except Exception:
            print("Невозможно посчитать значение функции в
заданной точке.")
            return None


        while abs(self.calculate(x)) > epsilon and
self.iterations <= 1000000:
            x = phi(x)
            self.iterations += 1
        if self.iterations == 1000000:
            return None
        return x

    def Newtons_Method(self):
        x = InputManager.float_input("Введите начальное
приближение: ")

        epsilon = 10 ** (-8)
        while abs(self.calculate(x)) > epsilon:
            x = x - self.calculate(x) / self.derivative_func(x)
            self.iterations += 1

        return x
```

PreparedSystemManager.py:

```
import numpy as np
from matplotlib import cm
from matplotlib import pyplot as plt

from InputManager import InputManager


class PreparedSystem:
    def __init__(self, funcs, derivative_funcs=None):
        """
        funcs: вектор функций
        derivative_funcs: матрица частных производных
        """
        self.solutions = None
```

```
        self.funcs = funcs
        self.derivative_funcs = derivative_funcs
        self.before_x_vector = []
        self.iterations = 0
        self.x_left, self.x_right = None, None
        self.y_left, self.y_right = None, None

    def calculate(self, x, y):
        return np.array([self.funcs[0](x, y), self.funcs[1](x,
y)])

    def calculate_derivatives(self, *args):
        return np.array([f(*args) for f in
self.derivative_funcs])

    def get_error_rate(self):
        return abs(np.array(self.before_x_vector) -
self.solutions)

    def _get_sufficient_convergence_condition(self, x, y):
        return max(sum(abs(i)) for i in
self.calculate_derivatives(x, y))

    def check_sufficient_convergence_condition(self, x, y):
        return self._get_sufficient_convergence_condition(x, y)
< 1

    def solve(self):
        return self.Simple_Iteration_Method()

    def _enter_root_isolation(self):
        print("Введите интервал изоляции корня:")
        x_left = InputManager.float_input("\tЛевый конец отрезка
по x: ")
        x_right = InputManager.float_input("\tПравый конец
отрезка по x: ")
        y_left = InputManager.float_input("\tЛевый конец отрезка
по y: ")
        y_right = InputManager.float_input("\tПравый конец
отрезка по y: ")
        while not (self.calculate(x_left, y_left)[0] *
self.calculate(x_right, y_right)[0] < 0 and
                   self.calculate(x_left, y_left)[1] *
self.calculate(x_right, y_right)[1] < 0):
            if not (x_left < x_right and y_left < y_right):
                print("Значения левых концов должны быть меньше
правых.")
                continue
            print("Значения функции на углах квадрата должны
быть разного знака для обеих функций!")
            if InputManager.yes_or_no_input(f"Показать график
системы для корректировки?"):
                self.draw_graphic(x_left, x_right, y_left,
```

```
y_right)
            print("Введите другие значения концов:")
            x_left = InputManager.float_input("\tЛевый конец
отрезка по x: ")
            x_right = InputManager.float_input("\tПравый конец
отрезка по x: ")
            y_left = InputManager.float_input("\tЛевый конец
отрезка по y: ")
            y_right = InputManager.float_input("\tПравый конец
отрезка по y: ")
        self.x_left, self.x_right = min(x_left, x_right),
max(x_left, x_right)
        self.y_left, self.y_right = min(y_left, y_right),
max(y_left, y_right)

        return

    def draw_graphic(self, x_left=None, x_right=None,
y_left=None, y_right=None):
        if None in [x_left, x_right, y_left, y_right]:
            if None in [self.x_left, self.x_right, self.y_left,
self.y_right]:
                print("Невозможно нарисовать график. Не
определены концы отрезка")
                return
            else:
                x_left = self.x_left
                x_right = self.x_right
                y_left = self.y_left
                y_right = self.y_right
        x = np.arange(x_left, x_right, abs(x_left - x_right) /
30)
        y = np.arange(y_left, x_right, abs(y_left - y_right) /
30)
        x, y = np.meshgrid(x, y)
        Z = self.calculate(x, y)
        fig = plt.figure()
        ax = fig.add_subplot(projection='3d')
        # ax.axes.set_zlim3d(-5, 5)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.plot_surface(x, y, Z[0], cmap=cm.coolwarm)
        ax.plot_surface(x, y, Z[1], cmap="plasma")
        plt.show()

    def Simple_Iteration_Method(self):
        self._enter_root_isolation()
        if InputManager.yes_or_no_input("Показать график системы
на введённом квадрате?"):
            self.draw_graphic()

        x = InputManager.float_input("Введите начальное
```

```
приближение по x: ")
        y = InputManager.float_input("Введите начальное
приближение по y: ")
        while not (self.x_left < x < self.x_right and
self.y_left < y < self.y_right):
            print("Начальное приближение должно быть внутри
интервала изоляции корня!")
            x = InputManager.float_input("Введите начальное
приближение по x: ")
            y = InputManager.float_input("Введите начальное
приближение по y: ")

        if not self.check_sufficient_convergence_condition(x,
y):
            print("Не выполняется достаточное условие сходимости
итерационного процесса.")
            print("Попробуйте ввести другое начальное
приближение.")
            if not InputManager.yes_or_no_input("Продолжить
выполнение?"):
                return

        # lambda_x = -
max(self._get_sufficient_convergence_condition(self.x_left,
self.y_left),
        #
self._get_sufficient_convergence_condition(self.x_right,
self.y_right))
        lambda_x = -1
        phi_x = lambda x, y: x + self.calculate(x, y)[0] /
lambda_x
        # lambda_y = -
max(self._get_sufficient_convergence_condition(self.x_left,
self.y_left),
        #
self._get_sufficient_convergence_condition(self.x_right,
self.y_right))
        lambda_y = -1
        phi_y = lambda x, y: y + self.calculate(x, y)[1] /
lambda_y
        epsilon = 10 ** (-5)
        self.before_x_vector = [x + 100, y + 100]
        while max(abs(x - self.before_x_vector[0]),
                  abs(y - self.before_x_vector[1])) > epsilon
and self.iterations <= 100000:
            self.before_x_vector[0], self.before_x_vector[1] =
x, y
            x, y = phi_x(x, y), phi_y(x, y)
            self.iterations += 1
        self.solutions = np.array([x, y])
        return self.solutions
```

EquationManager.py:

```
import re

import numpy as np
from math import sin, cos, asin, acos, tan, atan, log2, log10
from numpy import log as ln, log10 as lg
from numpy import log10 as log_10
from numpy import log2 as log_2
from numpy.lib.scimath import logn
from numpy.lib.scimath import logn as log_n

from Exceptions import InvalidVectorException
from InputManager import InputManager


class Equation:
    equation = ""
    raw_equation = ""
    var_names = []
    var_values = dict([("pi", np.pi), ("e", np.e)])
    _valid_vars_template = re.compile(r"[a-zA-Z_]+[0-9]*")
    _valid_functions_template =
re.compile(r"a?sin|lg|a?cos|a?tan|log_?(?:2|10)?|logn|pi|e")

    # _valid_vars_template = re.compile(r"[a-zA-Z](?:_[0-9]*)?")
# другой вариант выделений переменных

    def __init__(self, line):
        self.equation = line
        self.raw_equation = line
        self.var_names = self.find_vars()
        # print(self.var_names)

    def find_vars(self):
        vars = []
        for i in
self._valid_vars_template.findall(self.equation):
            if self._valid_functions_template.fullmatch(i) is
None and i not in vars:
                vars.append(i)
        return vars

    def set_valid_vars_template(self, template):
        self._valid_vars_template = template
        self.var_names =
self._valid_vars_template.findall(self.raw_equation)

    def get_var_list(self):
        return self.var_names.copy()

    def _get_variable_value_with_match(self, match,
x_vector_dict):
```

```
        name = match.group(0)
        value = x_vector_dict.get(name, None)
        if value is None:
            return name
        return f'({value})'

    def calculate(self, x_vector_dict={}):
        expression = self.equation

        expression = re.sub(self._valid_vars_template, lambda m:
self._get_variable_value_with_match(m, x_vector_dict),
                            expression)
        expression = re.sub(r"\d+(?:\.\d+)?", lambda x:
f'({x.group(0)})', expression)
        expression = expression.replace(")(", ")*(")
        # print(expression)
        return eval(expression)
```

AnyEquationSolver.py:

```
from InputManager import InputManager


class EquationSolver:
    @staticmethod
    def solve(equation):
        result = EquationSolver.Half_Division_Method(equation)
        if result is None:
            print("Не удалось найти корень уравнения.")
        else:
            print("Корень уравнения:", result)

    @staticmethod
    def Half_Division_Method(equation):

        if len(equation.get_var_list()) != 1:
            print("Для решения методом половинного деления
необходимо, "
                  "чтобы в уравнении участвовало только одно
неизвестное.")
            print("Неизвестные в данном уравнении:",
equation.get_var_list())
            return

        k = equation.get_var_list()[0]
        a = InputManager.float_input(f"\tВведите левый край
начального приближения: {k} = ")
        b = InputManager.float_input(f"\tВведите правый край
начального приближения: {k} = ")

        while not (equation.calculate({k: a}) *
equation.calculate({k: b}) < 0):
            print("Необходимо, чтобы значения функции на концах
```

```
отрезков были разных знаков!")
            if not InputManager.yes_or_no_input("Ввести концы
заново?"):
                return
            a = InputManager.float_input(f"\tВведите левый край
начального приближения: {k} = ")
            b = InputManager.float_input(f"\tВведите правый край
начального приближения: {k} = ")

        epsilon = 0.001
        print("Хотите ввести точность? По умолчанию epsilon
равно 10^-3.")
        if InputManager.yes_or_no_input("Ввести epsilon?"):
            epsilon = InputManager.float_input("epsilon=")

        if not equation.calculate({k: a}) < 0:
            a, b = b, a
        while not abs(a - b) < epsilon:
            mid = (a + b) / 2
            if equation.calculate({k: mid}) < 0:
                a = mid
            else:
                b = mid

        return (a + b) / 2
```

Exceptions.py:

```
class InvalidVectorException(ValueError):
    def __init__(self, message):
        # Call the base class constructor with the parameters it
needs
        super().__init__(message)


class InvalidAlgebraicEquationException(ValueError):
    def __init__(self, message):
        # Call the base class constructor with the parameters it
needs
        super().__init__(message)
```

6. Результаты выполнения программы при различных исходных данных.

```
Чем хотите себя побаловать?
    1. Выбрать функцию и метод из предложенных
    2. Ввести произвольную функцию и решить её методом половинного деления
    3. Выбрать систему уравнений из предложенных и решить её методом простых итераций
Введите число от 1 до 3: 1
Выберите функцию:
    1. 3x^3 + 1.7x^2 - 15.42x + 6.89
    2. x * sin(x) - 1
    3. e^sin(x) * ln(|x|) - 1.4
    4. arcsin(e^(-x)) - 0.3
    5. tg(x/2 * sin(x))
Введите число от 1 до 5: 3
Выберите метод решения уравнения:
    1. Метод хорд
    2. Метод Ньютона
    3. Метод простых итераций
Введите число от 1 до 3: 1
Введите интервал изоляции корня:
    Левый конец отрезка: 0.1
    Правый конец отрезка: 3
Значения функции на концах отрезка должны быть разного знака!
Показать график функции для корректировки? [y/n]: n
Введите другие значения концов:
    Левый конец отрезка: 0.1
    Правый конец отрезка: 2
------------РЕШЕНИЕ------------
Найденный корень уравнения: 1.67873
Значение функции в точке:  9e-09
Количество итераций: 21
Показать график функции? [y/n]: l
Показать график функции? [y/n]: y
```

Рис. 3. Пример работы программы

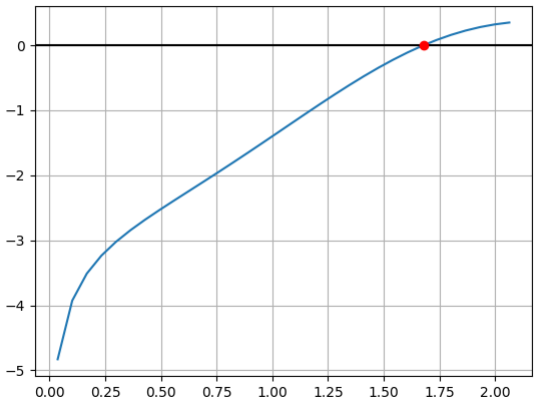

Рис. 2. График функции из примера на рис. 1

7. Выводы

В ходе выполнения лабораторной работы я изучил и реализовал несколько методов вычисления корней. Также отработал навыки создания user-friendly консольных приложений.