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

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №18

Выполнил:
Студент группы P3213
Хафизов Булат Ленарович
Преподаватель:
Малышева Татьяна
Алексеевна

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург, 2022

Цель работы

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

Ход работы

№ шага	a	b	x	f(a)	f(b)	f(x)	$\lvert a - b \rvert$
0	3	4	3.5	4.01	-0.42	3.003	1
1	3.5	4	3.75	3.003	-0.42	1.64	0.5
2	3.75	4	3.875	1.64	-0.42	0.703	0.25
3	3.875	4	3.938	0.703	-0.42	0.161	0.125
4	3.938	4	3.969	0.161	-0.42	-0.123	0.062
5	3.938	3.969	3.954	0.161	-0.123	0.016	0.031
6	3.954	3.969	3.962	0.016	-0.123	-0.058	0.015
7	3.954	3.962	3.958	0.016	-0.058	-0.021	0.008

Таблица 1 - Уточнение крайнего правого корня методом половинного деления (хорд)

№ шага	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}	$\lvert x_{n+1} - x_n \rvert$
0	0	1.34	-7,12	0,188	0,188
1	0.188	0.195	-5,094	0,226	0,038
2	0.226	0.009	-4,71	0,228	0,002

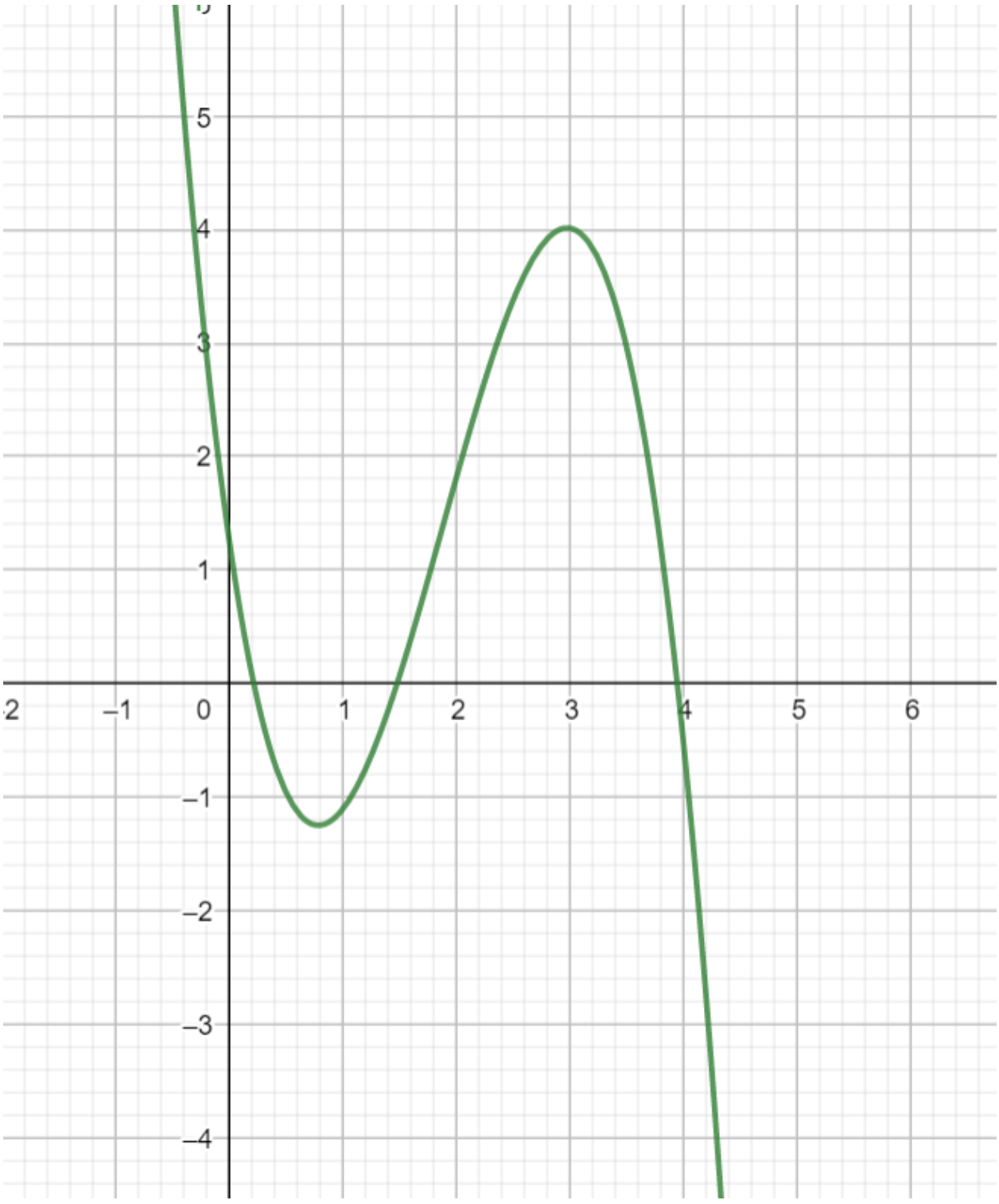
Таблица 2 - Уточнение крайнего левого корня методом Ньютона

№ шага	x_{k-1}	$f(x_{k-1})$	x_k	$f(x_k)$	x_{k+1}	$f(x_{k+1})$	$\lvert x_k - x_{k+1} \rvert$
0	2	1.78	1.8	1.063	1.503	0.052	0.297
1	1.8	1.063	1.503	0.052	1.488	0.005	0.015
2	1.503	0.052	1.488	0.005	1.486	-0.001	0.006

Таблица 3 - Уточнение центрального корня методом секущих

№ итерации	x_k	$f(x_k)$	x_{k+1}	$\varphi(x_k)$	$\lvert x_k - x_{k+1} \rvert$
1	1	-1.11	1.91	1.91	0.91
2	1.91	1.457	1.505	1.505	0.405
3	1.505	0.057	1.487	1.487	0.018
4	1.487	0.000	1.486	1.486	0.000

Таблица 4 - Уточнение корня уравнения методом простой итерации



Листинг программы

Основной класс:

```
import numpy as np
import matplotlib.pyplot as plt
from lab2.NonLinearSystems import CalculatorSystems


def halfDivision_method(a, b, f, e):
    if f(a) * f(b) > 0:
        print("Уравнение содержит 0 или несколько корней!")
        exit(0)
```

```
    n = 0
    table = [['№', 'a', 'b', 'x', 'F(a)', 'F(b)', 'F(x)', '|a-b|']]
    while True:
        x = (a + b) / 2
        table.append([n, a, b, x, f(a), f(b), f(x), abs(a - b)])
        if f(a) * f(x) > 0:
            a = x
        else:
            b = x
        n += 1
        if abs(a - b) <= e or abs(f(x)) < e:
            break
    x = (a + b) / 2
    return x, f(x), n, table


def d(n, x, f, h=0.00000001):
    """ Найти значение производной функции """
    if n <= 0:
        return None
    elif n == 1:
        return (f(x + h) - f(x)) / h

    return (d(n - 1, x + h, f) - d(n - 1, x, f)) / h


def simpleIteration_method(a, b, f, e, maxitr=100):
    log = [['№', 'x_i', 'x_(i+1)', 'g(x_i)', 'f(x_i)', '|x_(i+1) - x_i|']]
    x0 = 0

    def g(g_x):
        return g_x + (-1 / d(1, g_x, f)) * f(g_x)

    if abs(d(1, a, g)) < 1 and abs(d(1, b, g)) < 1:
        if f(a) * d(2, a, f) > 0:
            x0 = a
        elif f(b) * d(2, b, f) > 0:
            x0 = b
    else:
        print("Выберите другой интервал! Не выполняется условие сходимости!")
        exit(0)
    x = g(x0)
    log.append([0, x0, x, g(x0), f(x0), abs(x - x0)])
    itr = 1
    while abs(x - x0) > e and itr < maxitr:
        if d(1, x, g) >= 1:
            return None
        x0, x = x, g(x)
        log.append([itr, x0, x, g(x0), f(x0), abs(x - x0)])
        itr += 1
    return x, f(x), itr, log



def plot(x, y):
    """ Отрисовать график по заданным x и y """
    plt.gcf().canvas.manager.set_window_title("График функции")
    ax = plt.gca()
    ax.spines['left'].set_position('zero')
    ax.spines['bottom'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.plot(1, 0, marker=">", ms=5, color='k',
            transform=ax.get_yaxis_transform(), clip_on=False)
```

```
    ax.plot(0, 1, marker="^", ms=5, color='k',
            transform=ax.get_xaxis_transform(), clip_on=False)
    plt.plot(x, y)
    plt.show()


def getfunc(num):
    if num == '1':
        return np.linspace(-1, 3, 200), \
               lambda x: x ** 3 - 2.92 * (x ** 2) + 1.435 * x + 0.791
    elif num == '2':
        return np.linspace(-3, 2, 200), \
               lambda x: x ** 3 - x + 4
    elif num == '3':
        return np.linspace(-20, 20, 200), \
               lambda x: np.sin(x) + 0.1
    elif num == '4':
        return np.meshgrid(np.linspace(-5, 5, 200), np.linspace(-5, 5, 200)),
\
               lambda x, y: x ** 2 + y ** 2 - 4
    elif num == '5':
        return np.meshgrid(np.linspace(-5, 5, 200), np.linspace(-5, 5, 200)),
\
               lambda x, y: y - 3 * (x ** 2)
    else:
        return None


def file_read(filename, type_func):
    file = open(filename, 'r')
    data = {}
    try:
        a, b, e = map(float, file.readline().strip().split())
        if a > b:
            a, b = b, a
        elif a == b:
            print("Некорректные данные в файле! 'a' не может равняться 'b'!")
            exit(0)
        data['a'] = a
        data['b'] = b
        if e < 0:
            print("Некорректные данные в файле! Погрешность не может быть
отрицательной!")
            exit(0)
        data['e'] = e
    except ValueError:
        print("Границы интервала и погрешность должны быть числами,
введенными через пробел!")
        exit(0)
    return data



def nonlinearSingle():
    print("Выберите функцию:")
    print("1: x³ - 2.92x² + 4.435x + 0.791")
    print("2: x³ - x + 4")
    print("3: sin(x) + 0.1")
    type_func = input()
    function_data = getfunc(type_func)
    while function_data is None:
        print("Выберите функцию из списка.")
        function_data = getfunc(input("Функция: "))
    x, function = function_data
```

```
    print("Выберите метод решения.")
    print(" 1 - Метод половинного деления")
    print(" 2 - Метод простой итерации")
    method = input()
    while (method != '1') and (method != '2') and (method != '3'):
        print("Выберите метод решения из списка.")
        method = input("Метод решения: ")
    print("Теперь нужно ввести данные для функции (границы
интервала/начальное приближение к корню и погрешность"
          " вычисления).\n"
          "Выберите способ ввода: из исходного файла (-) или ввести с
клавиатуры (+)")
    type_read = input()
    while type_read != '-' and type_read != '+':
        print("Выберите способ ввода из списка.")
        method = input("Способ ввода: ")
    if type_read == "-":
        data = file_read(input("Введите название файла: "), method)
    else:
        data = keyboard_read(method)
    data['function'] = function
    answer = None
    if method == '1':
        answer = halfDivision_method(data['a'], data['b'], data['function'],
data['e'])
    elif method == '2':
        answer = simpleIteration_method(data['x0'], data['function'],
data['e'])
        if answer is None:
            print("Не выполняется условие сходимости!")
            exit(0)
    print("Записать ответ в файл (-) или вывести в консоль (+)?")
    type_write = input()
    if type_write == '+':
        print(f"Корень уравнения: {answer[0]}")
        print(f"Значение функции в корне: {answer[1]}")
        print(f"Число итераций: {answer[2]}")
        logChoice = input("Вывести таблицу трассировки? (+/-)\n")
        if logChoice == '+':
            for j in range(len(answer[3][0])):
                print('%12s' % answer[3][0][j], end='')
            print()
            for i in range(1, len(answer[3])):
                for j in range(len(answer[3][i])):
                    print('%12.3f' % answer[3][i][j], end='')
                print()
    else:
        output = open('output.txt', 'w', encoding='utf-8')
        output.write(f"Корень уравнения: {answer[0]}\n")
        output.write(f"Значение функции в корне: {answer[1]}\n")
        output.write(f"Число итераций: {answer[2]}\n")
        for j in range(len(answer[3][0])):
            output.write('%12s' % answer[3][0][j])
        output.write("\n")
        for i in range(1, len(answer[3])):
            for j in range(len(answer[3][i])):
                output.write('%12.3f' % answer[3][i][j])
            output.write('\n')
        output.close()
    plot(x, function(x))


def nonlinearSystem():
    print("В данной задаче будет рассмотрена система уравнений:")
```

```
    print("x² + y² = 4")
    print("y = 3x²")
    try:
        x0 = float(input("Введите начальное приближение x0: "))
        y0 = float(input("Введите начальное приближение y0: "))
        e = float(input("Введите погрешность: "))
        arr1, functionF = getfunc('4')
        arr2, functionG = getfunc('5')
        calculator = CalculatorSystems(x0, y0, e, functionF, functionG)
        calculator.calculate()
        del calculator
    except ValueError:
        print("Начальное приближение и погрешность должны быть числами!")
        exit(0)


def keyboard_read(type_func):
    data = {}
    a = float(input("Введите левую границу интервала: "))
    b = float(input("Введите правую границу интервала: "))
    e = float(input("Введите погрешность: "))
    try:
        if a > b:
            a, b = b, a
        elif a == b:
            print("Некорректные данные! 'a' не может равняться 'b'!")
            exit(0)
        data['a'] = a
        data['b'] = b
        if e < 0:
            print("Некорректные данные! Погрешность не может быть
отрицательной!")
            exit(0)
        data['e'] = e
    except ValueError:
        print("Границы интервала и погрешность должны быть числами,
введенными через пробел!")
        exit(0)
    return data



def main():
    print("Лабораторная работа №2")
    print("Вариант №18")
    print("Численное решение нелинейных уравнений/систем уравнений")
    print("Что вы хотите решить?:")
    print("1: Уравнение")
    print("2: Систему уравнений")
    type_input = input()
    if type_input == '1':
        nonlinearSingle()
    else:
        nonlinearSystem()


main()
```

Класс для обработки системы:

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


class CalculatorSystems:
```

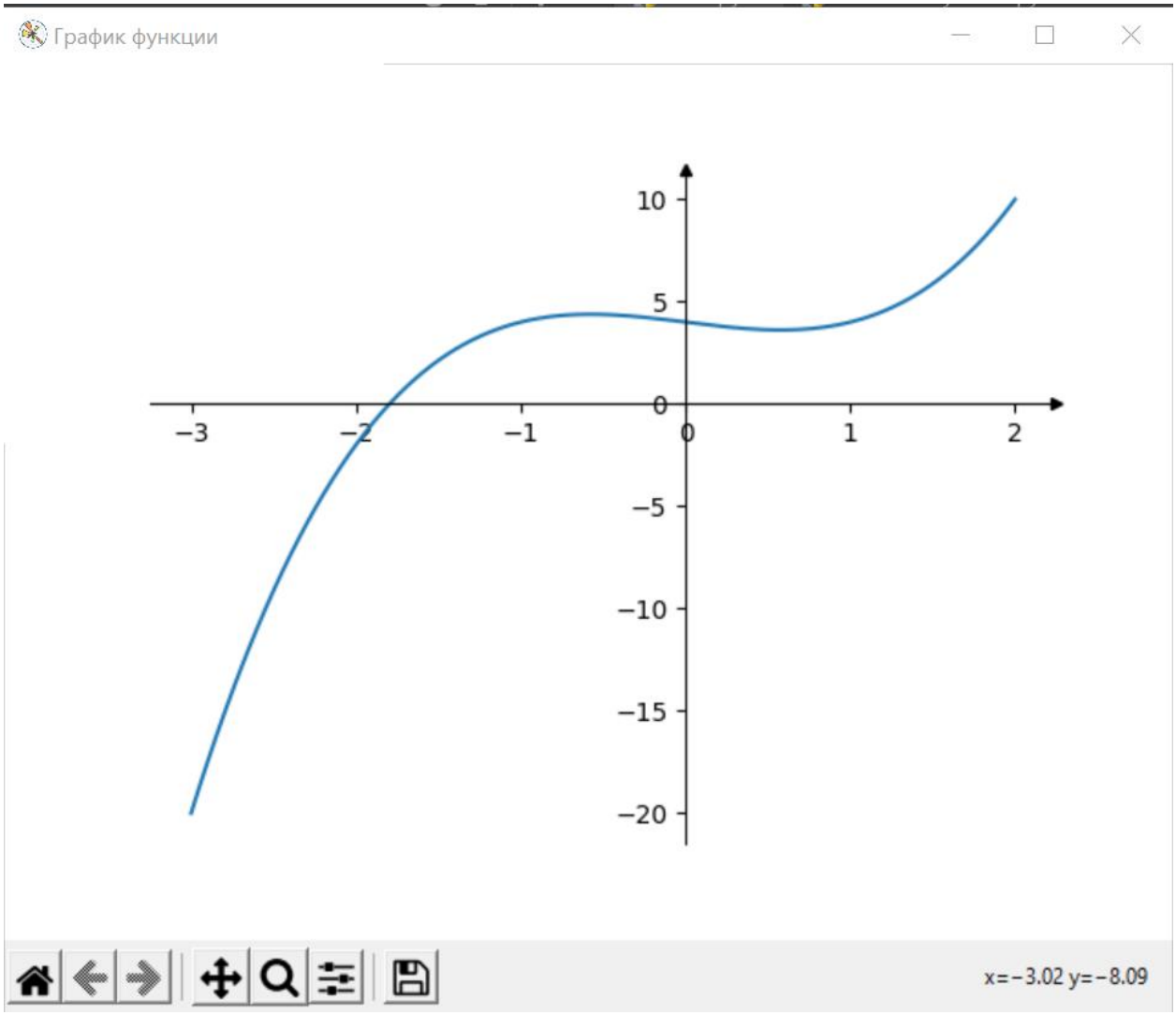
```
    e = 0
    x = 0
    y = 0
    x0 = 0
    y0 = 0
    itr = 0
    f = None
    g = None

    def __init__(self, x0, y0, e, f, g):
        self.x0 = x0
        self.y0 = y0
        self.e = e
        self.f = f
        self.g = g

    def calculate(self):
        while 1 and self.itr < 250000:
            self.x = self.x0 - determinant(
                self.get_a(1, self.x0, self.y0)) / determinant(
                self.jacobian(self.x0, self.y0))
            self.y = self.y0 - determinant(
                self.get_a(2, self.x0, self.y0)) / determinant(
                self.jacobian(self.x0, self.y0))
            self.itr += 1
            if abs(self.x - self.x0) <= self.e and abs(
                    self.y - self.y0) <= self.e:
                self.e = abs(self.x - self.x0 + self.y - self.y0) / 2
                break
            self.x0 = self.x
            self.y0 = self.y
        printResult(self)
        plot(self.f, self.g)

    def x_derivative(self, type_eq, x, y, h=0.00001):
        if type_eq == 1:
            return (self.f(x + h, y) - self.f(x, y)) / h
        elif type_eq == 2:
            return (self.g(x + h, y) - self.g(x, y)) / h

    def y_derivative(self, type_eq, x, y, h=0.00001):
        if type_eq == 1:
            return (self.f(x, y + h) - self.f(x, y)) / h
        elif type_eq == 2:
            return (self.g(x, y + h) - self.g(x, y)) / h

    def jacobian(self, x, y):
        return [[self.x_derivative(1, x, y),
                 self.y_derivative(1, x, y)],
                [self.x_derivative(2, x, y),
                 self.y_derivative(2, x, y)]]

    def get_a(self, mode, x, y):
        if mode == 1:
            return [[self.f(x, y),
                     self.y derivative(1, x, y)],
                    [self.g(x, y),
                     self.y_derivative(2, x, y)]]
        elif mode == 2:
            return [[self.x_derivative(1, x, y),
                     self.f(x, y)],
                    [self.x_derivative(2, x, y),
                     self.g(x, y)]]
```

```
def determinant(matrix):
    return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]


def plot(f, g):
    plt.gcf().canvas.manager.set_window_title("График функции")
    ax = plt.gca()
    ax.spines['left'].set_position('zero')
    ax.spines['bottom'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.plot(1, 0, marker=">", ms=5, color='k',
            transform=ax.get_yaxis_transform(), clip_on=False)
    ax.plot(0, 1, marker="^", ms=5, color='k',
            transform=ax.get_xaxis_transform(), clip_on=False)
    xRange = np.arange(-10, 10, 0.025)
    yRange = np.arange(-10, 10, 0.025)
    X, Y = np.meshgrid(xRange, yRange)
    F = f(X, Y)
    G = g(X, Y)
    plt.contour(X, Y, F, [0])
    plt.contour(X, Y, G, [0])
    plt.show()


def printResult(calculator):
    print("Записать ответ в файл (-) или вывести в консоль (+)?")
    type_write = input()
    if type_write == '+':
        print("Корни: x=" + str(calculator.x) + ", y=" + str(calculator.y) +
"\n" +
              "Количество итераций: " + str(calculator.itr) + "\n" +
              "Погрешность: " + str(calculator.e) + "\n")
    else:
        output = open('output.txt', 'w', encoding='utf-8')
        output.write("Корни: x=" + str(calculator.x) + ", y=" +
str(calculator.y) + "\n" +
                     "Количество итераций: " + str(calculator.itr) + "\n" +
                     "Погрешность: " + str(calculator.e) + "\n")
        output.close()
```

Вывод программы

```
Лабораторная работа №2
Вариант №18
Численное решение нелинейных уравнений/систем уравнений
Что вы хотите решить?:
1: Уравнение
2: Систему уравнений
1
Выберите функцию:
1: x³ - 2.92x² + 4.435x + 0.791
2: x³ - x + 4
3: sin(x) + 0.1
2
Выберите метод решения.
 1 - Метод половинного деления
 2 - Метод простой итерации
1
Теперь нужно ввести данные для функции (границы интервала/начальное
приближение к корню и погрешность вычисления).
Выберите способ ввода: из исходного файла (-) или ввести с клавиатуры (+)
+
```

```
Введите левую границу интервала: -2
Введите правую границу интервала: -1
Введите погрешность: 0.01
Записать ответ в файл (-) или вывести в консоль (+)?
+
Корень уравнения: -1.7890625
Значение функции в корне: 0.06273031234741211
Число итераций: 6
Вывести таблицу трассировки? (+/-)
+
           №           a           b           x        F(a)        F(b)
F(x)        |a-b|
       0.000      -2.000      -1.000      -1.500      -2.000       4.000
2.125       1.000
       1.000      -2.000      -1.500      -1.750      -2.000       2.125
0.391       0.500
       2.000      -2.000      -1.750      -1.875      -2.000       0.391
-0.717       0.250
       3.000      -1.875      -1.750      -1.812      -0.717       0.391
-0.142       0.125
       4.000      -1.812      -1.750      -1.781      -0.142       0.391
0.130       0.062
       5.000      -1.812      -1.781      -1.797      -0.142       0.130
-0.005       0.031
```



Вывод

В результате выполнения данной лабораторной работой я познакомился с численными методами решения нелинейных уравнений и реализовал метод хорд, метод секущих и метод простой итерации на языке программирования Python, закрепив знания.