Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04

Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №2

Вариант 11

Выполнил: Савельева Диана Александровна

P32082

Преподаватель: Машина Екатерина Алексеевна

Санкт-Петербург, 2023 г.

***1. Цель:*** 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:***

*Для нелинейных уравнений.* Пользователь выбирает уравнение, корень/корни которого требуется вычислить (3-5 функций, в том числе и трансцендентные), из тех, которые предлагает программа. Ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя. Выполнить верификацию исходных данных. Необходимо анализировать наличие корня на введенном интервале. Если на интервале несколько корней или они отсутствуют – выдавать соответствующее сообщение. Программа должна реагировать на некорректные введенные данные. Для методов, требующих начальное приближение к корню (методы Ньютона, секущих, хорд с фиксированным концом), выбор начального приближения (а или b) вычислять в программе. Для метода простой итерации проверять достаточное условие сходимости метода на введенном интервале. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).

*Для систем нелинейных уравнений.* Пользователь выбирает предлагаемые программой системы двух нелинейных уравнений (2-3 системы). Организовать вывод графика функций. Начальные приближения ввести с клавиатуры. Для метода простой итерации проверить достаточное условие сходимости. Организовать вывод вектора неизвестных: 𝑥1, 𝑥2. Организовать вывод количества итераций, за которое было найдено решение. Организовать вывод вектора погрешностей: |𝑥𝑖 (𝑘) − 𝑥𝑖 (𝑘−1) |. Проверить правильность решения системы нелинейных уравнений.

***3. Рабочие формулы используемых методов:***

Методы для решения нелинейных уравнений:

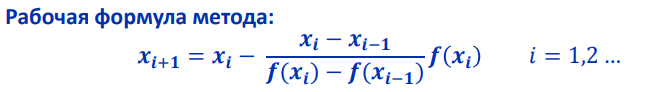
* *Метод половинного деления*

Идея метода: начальный интервал изоляции корня делим пополам, получаем начальное приближение к корню.

**

* *Метод секущих*

Идея: упростим метод Ньютона, заменив f'(x) разностным приближением.

**

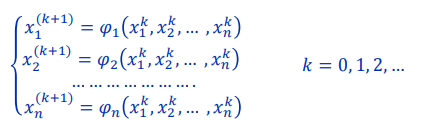
* *Метод простой итерации*

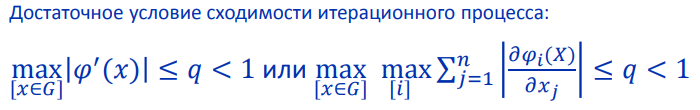
Идея: Уравнение 𝑓 𝑥 = 0 приведем к эквивалентному виду: 𝑥 = 𝜑(𝑥), выразив 𝑥 из исходного уравнения.

**

Методы для решения систем нелинейных уравнений:

* *Метод простой итерации*

**

**

***4. Графики функций:***

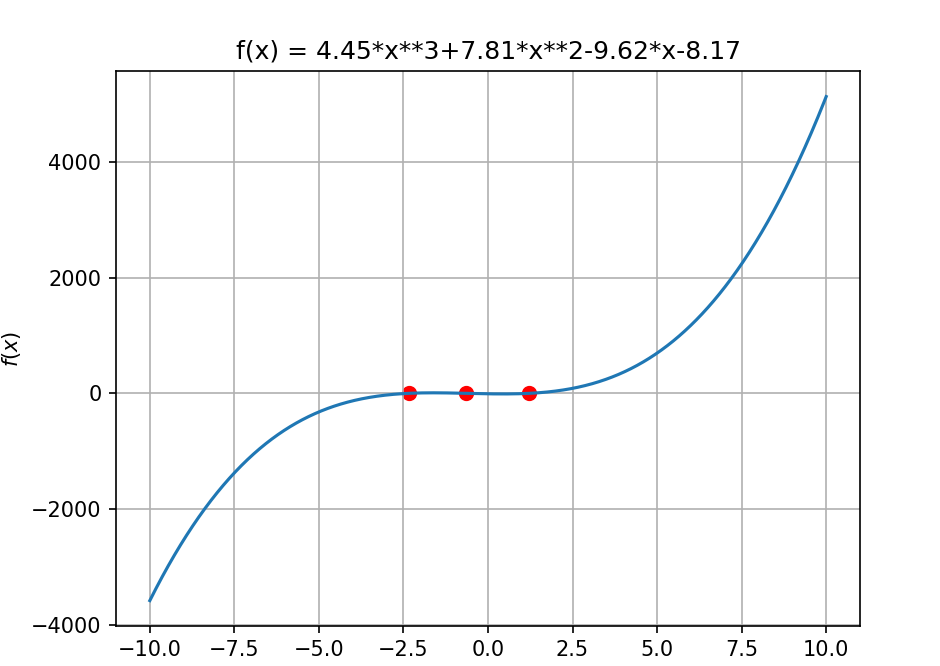


Рисунок 1 – График функции

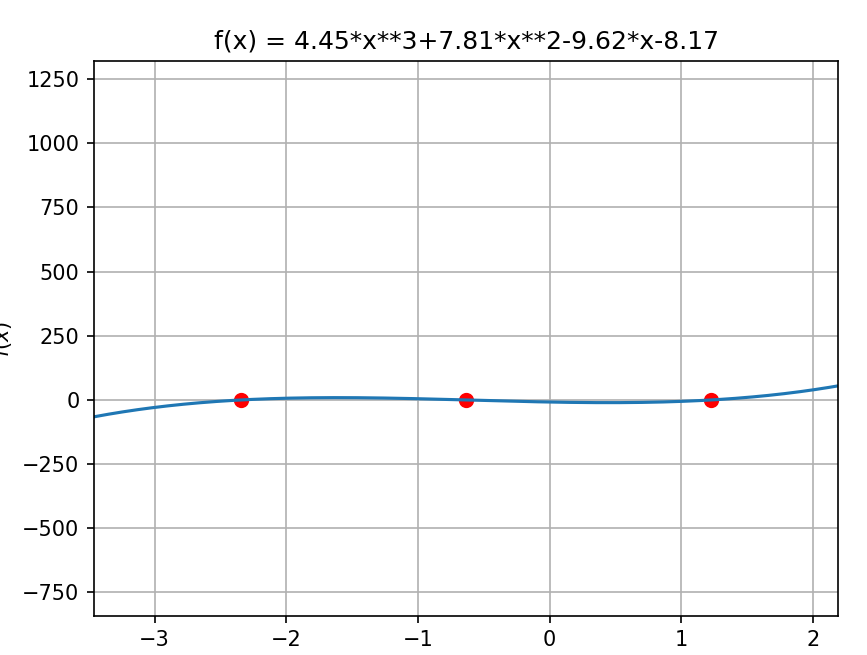


Рисунок 2 – График функции

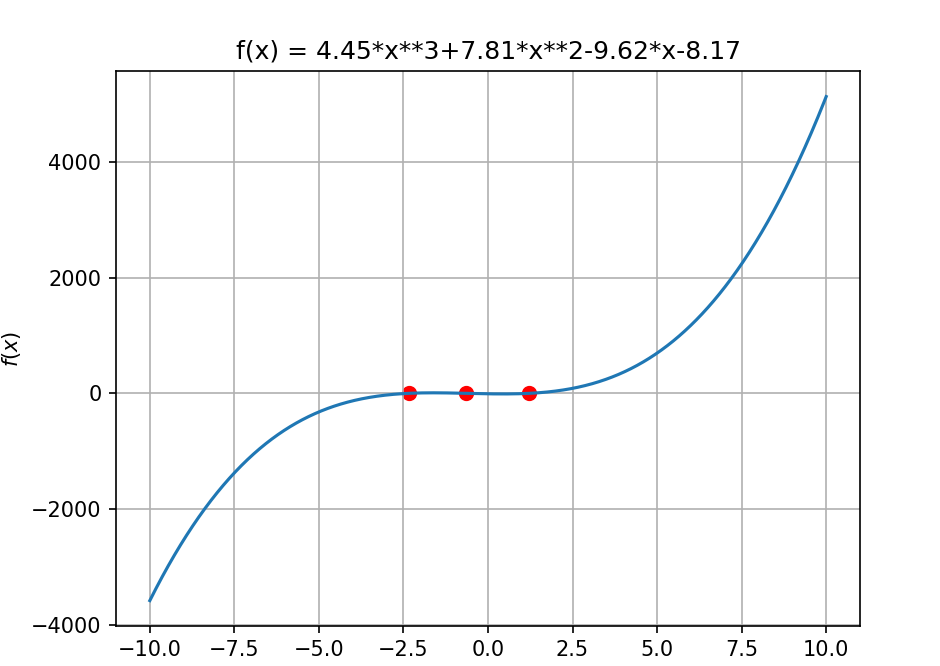
***5. Вычислительная часть:***

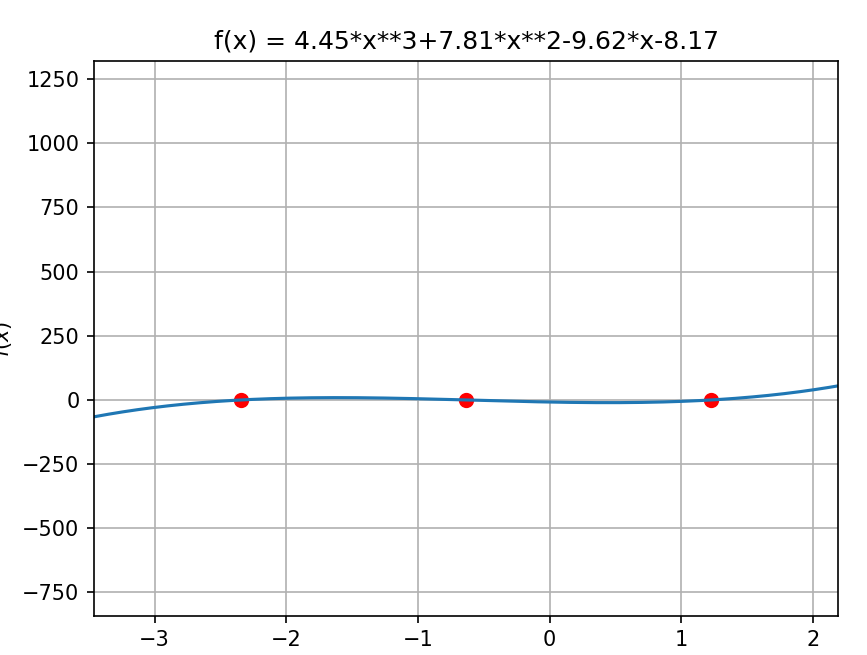
****

**Задание:**

*1. Отделить корни заданного нелинейного уравнения графически.*

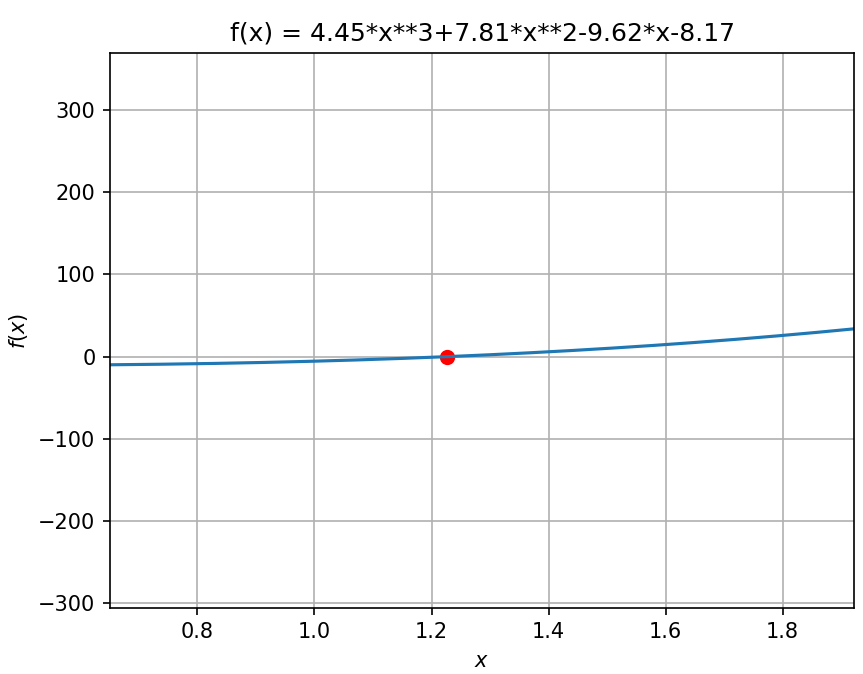
*2. Определить интервалы изоляции корней.*



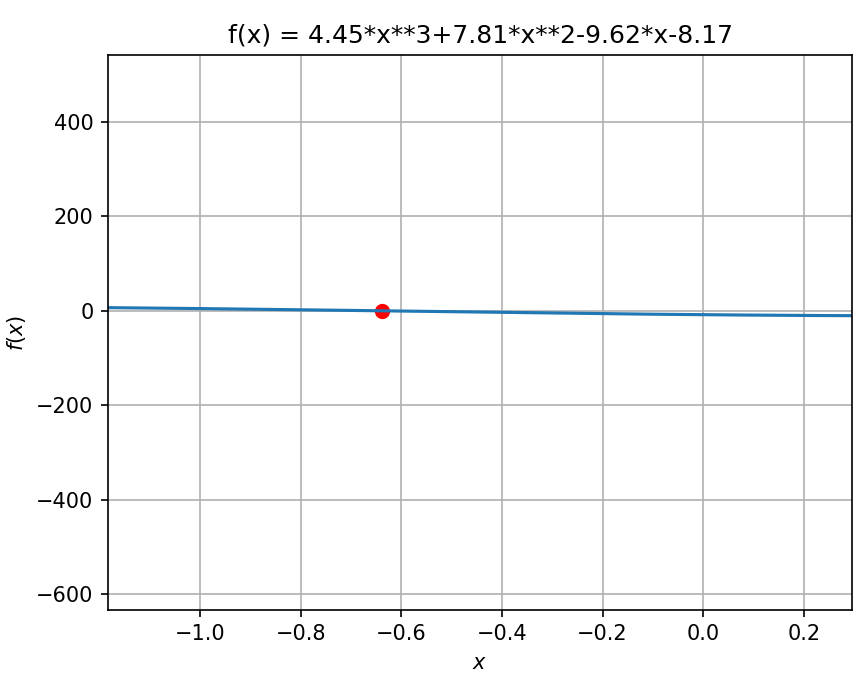


Корни (с сохранением 4 знаков после запятой) на промежутке [-10;10]:

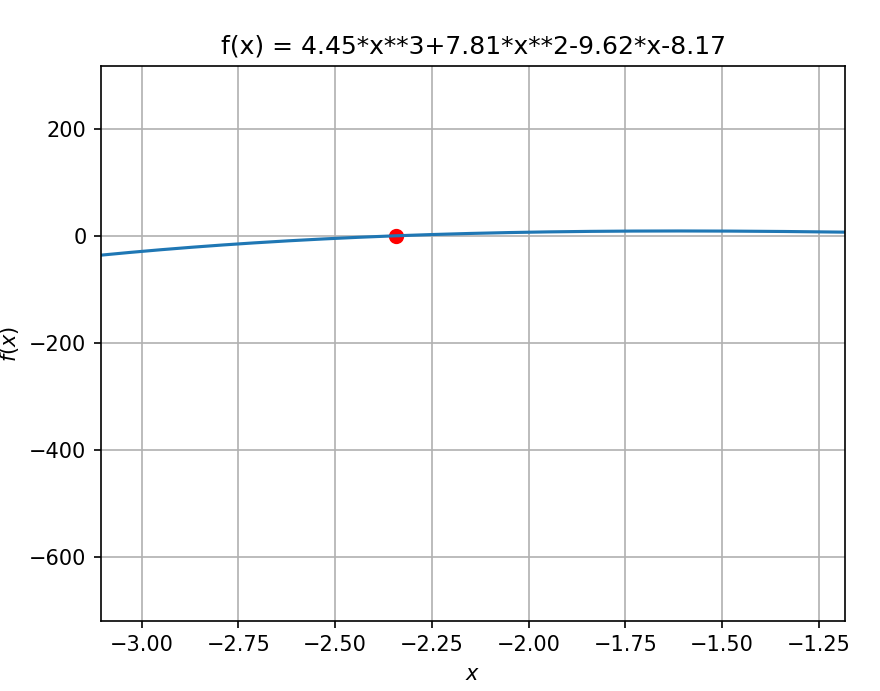
1. Корень x1 = 1.2268, a1 = 1.2, b1 = 1.4.



2. Корень x2 = -0.6386. Интервал изоляции корня: a2 =-1, b2 =0.



3. Корень x3 = -2.3432. Интервал изоляции корня: a3 = -2.5, b3 = -2



*3. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью ε=10-2.*

*4. Используемые методы для уточнения каждого из 3-х корней многочлена представлены в таблице 7.*

* Крайний правый корень – 1. Метод половинного деления.

**Уточнение корня уравнения методом половинного деления**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *№ шага* | *a* | *b* | *x* | *f(a)* | *f(b)* | *f(x)* | *|a-b|* |
| *1* | 1.200 | 1.400 | 1.300 | -0.778 | 5.880 | 2.299 | 0.2 |
| *2* | 1.200 | 1.300 | 1.250 | -0.778 | 2.299 | 0.699 | 0.1 |
| *3* | 1.200 | 1.250 | 1.225 | -0.778 | 0.699 | -0.054 | 0.05 |
| *4* | 1.225 | 1.250 | 1.237 | -0.054 | 0.699 | 0.318 | 0.025 |
| *5* | 1.225 | 1.237 | 1.231 | -0.054 | 0.303 | 0.123 | 0.012 |
| *6* | 1.225 | 1.231 | **1.228** | -0.054 | 0.123 | 0.034 | 0.006 |

Число шагов n =

Нахождение начального приближения к корню по формуле:



* Крайний левый корень – 2. Метод хорд.

**Уточнение корня уравнения методом хорд**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *№ шага* | *a* | *b* | *x* | *f(a)* | *f(b)* | *f(x)* | *|xk-xk-1|* |
| *1* | -2.500 | -2.000 | -2.290 | -4.848 | 6.71 | 1.363 | 0 |
| *2* | -2.500 | -2.290 | -2.336 | -4.838 | 1.376 | 0.181 | 0.045 |
| *3* | -2.500 | -2.336 | -2.342 | -4.838 | 0.195 | 0.024 | 0.005 |

Число шагов n =

* Центральный корень – 5. Метод простой итерации.

Начальное уравнение: 4.45x3+7.81x2-9.62x-8.17 = 0

Эквивалентное уравнение: x = -(8.17 – 7.81x2 - 4.45x3)/9.62

Производная эквивалентного уравнения:

fi(x)` = 2\*7.81/9.62x+4.45/9.62\*3x2

fi(-0.8)` = 2\*7.81/9.62x+4.45/9.62\*3x2=|-0.235|<1

fi(-0.6)` = 2\*7.81/9.62x+4.45/9.62\*3x2=|0|<1 Условие сходимости выполняется.

a0 = -0.8, b0 = -0.6

**Уточнение корня уравнения методом простой итерации**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *№ итерации* | *xk* | *xk+1* | *f(xk+1)* | *| xk+1-xk |* |
| *1* | -0.800 | -0.560 | -1.022 | 0.233 |
| *2* | -0.566 | -0.474 | -2.322 | 0.092 |
| *3* | -0.457 | -2.553 | -2.553 | 0.016 |
| *4* | -0.452 | -2.629 | -2.629 | 0.005 |

***6. Листинг программы:***

import numpy as np  
from numpy import inf, nan  
import pandas as pd  
from Exceptions.IncorrectValueException import IncorrectValueException  
from NonlinearEquations.NonlinearEquationsSolver import NonlinearEquationsSolver  
from NonlinearEquations.Equations import \*  
import numexpr as ne  
  
AMOUNT\_OF\_COLUMNS\_EASY\_ITERATIONS = 5  
  
  
class MethodEasyIteration(NonlinearEquationsSolver):  
 def methodEasyIteration(self, number\_of\_equation: int):  
 a = self.getLeftBorder()  
 b = self.getRightBorder()  
 eps = self.getEpsilon()  
 lamb = calculateLambda(a, b, number\_of\_equation)  
 EQUATIONS[number\_of\_equation]['FI'] = EQUATIONS[number\_of\_equation]['FI'].replace('la', f'{lamb}')  
 EQUATIONS[number\_of\_equation]['DERIVATIVE\_FI'] = EQUATIONS[number\_of\_equation]['DERIVATIVE\_FI'].replace('la',  
 f'{lamb}')  
 print('\t\t\tМетод простой итерации')  
 print(f'1. Левая граница a={a}')  
 print(f'2. Правая граница b={b}')  
 print(f'3. Точность epsilon={eps}')  
 maxIterationNumber = self.calculateMaxIteration(eps)  
 iterations = [[0.0 for x in range(AMOUNT\_OF\_COLUMNS\_EASY\_ITERATIONS)]  
 for x in range(maxIterationNumber)]  
 count\_of\_iterations = 0  
 if self.checkConditionConvergence(a, b, number\_of\_equation):  
 print('Условие сходимости выполняется.')  
 else:  
 print('Условие сходимости не выполняется.')  
 for i in range(maxIterationNumber):  
 iterations[i][0] = count\_of\_iterations  
 iterations[i][1] = a  
 iterations[i][2] = calculateFiValue(iterations[i][1], number\_of\_equation)  
 iterations[i][3] = calculateFunctionValue(iterations[i][2], number\_of\_equation)  
 if np.isinf(iterations[i][2]) or np.isnan(iterations[i][3]) or np.isinf(iterations[i][3]) or np.isnan(iterations[i][3]):  
 raise IncorrectValueException('На данном промежутке либо несколько корней, либо нет корней.')  
 iterations[i][4] = abs(iterations[i][2] - iterations[i][1])  
 count\_of\_iterations += 1  
 if iterations[i][4] <= eps:  
 break  
 a = iterations[i][2]  
 else:  
 raise IncorrectValueException('На данном промежутке либо несколько корней, либо нет корней.')  
 self.printTableForMethodEasy(iterations, count\_of\_iterations)  
 return iterations[count\_of\_iterations - 1][2], iterations[count\_of\_iterations - 1][3], count\_of\_iterations  
  
 def checkConditionConvergence(self, a0, b0, number\_of\_equation):  
 a\_derivative = abs(calculateFunctionValue(a0, number\_of\_equation))  
 b\_derivative = abs(calculateFunctionValue(b0, number\_of\_equation))  
 if a\_derivative < 1 and b\_derivative < 1:  
 return True  
 else:  
 return False  
  
 def printTableForMethodEasy(self, table, count\_of\_iterations):  
 print('№ итерации| x(k) | x(k+1) | f(x(k+1)) | |x(k+1) - xk| |')  
 for i in range(count\_of\_iterations):  
 print(f' {table[i][0]} | {table[i][1]} | {table[i][2]} | '  
 f'{table[i][3]} | {table[i][4]} |')

from Exceptions.IncorrectValueException import IncorrectValueException  
from NonlinearEquations.NonlinearEquationsSolver import NonlinearEquationsSolver  
from NonlinearEquations.Equations import \*  
import numexpr as ne  
  
AMOUNT\_OF\_COLUMNS\_HALF\_DIVISION = 8  
  
  
class MethodHalfDivision(NonlinearEquationsSolver):  
 # Метод половинного деления  
 def methodHalfDivision(self, number\_of\_equation):  
 a = self.getLeftBorder()  
 b = self.getRightBorder()  
 eps = self.getEpsilon()  
 print('\t\t\tМетод половинного деления')  
 print(f'1. Левая граница a={a}')  
 print(f'2. Правая граница b={b}')  
 print(f'3. Точность epsilon={eps}')  
 maxIterationNumber = self.calculateMaxIteration(eps)  
 iterations = [[0.0 for x in range(AMOUNT\_OF\_COLUMNS\_HALF\_DIVISION)]  
 for x in range(maxIterationNumber)]  
 count\_of\_iterations = 0  
 for i in range(maxIterationNumber):  
 x = (a + b) / 2  
 iterations[i][0] = i + 1  
 iterations[i][1], iterations[i][2] = a, b  
 iterations[i][3] = x  
 iterations[i][4] = calculateFunctionValue(a, number\_of\_equation)  
 iterations[i][5] = calculateFunctionValue(b, number\_of\_equation)  
 iterations[i][6] = calculateFunctionValue(x, number\_of\_equation)  
 iterations[i][7] = abs(a - b)  
 if iterations[i][4] \* iterations[i][6] > 0 and iterations[i][5] \* iterations[i][6] > 0:  
 raise IncorrectValueException('Невозможно найти корень на данном интервале. Его нет или их несколько.')  
 if iterations[i][4] \* iterations[i][6] > 0:  
 a = x  
 else:  
 b = x  
 count\_of\_iterations += 1  
 if iterations[i][7] <= eps:  
 break  
 self.printTableForMethodHalfDivision(iterations, count\_of\_iterations)  
 return iterations[count\_of\_iterations - 1][3], iterations[count\_of\_iterations - 1][6], count\_of\_iterations  
  
 def printTableForMethodHalfDivision(self, table, count\_of\_iterations):  
 print('№ итерации| a | b | x | f(a) | f(b) | f(x) | |a-b| |')  
 for i in range(count\_of\_iterations):  
 print(f' {table[i][0]} | {table[i][1]} | {table[i][2]} | '  
 f'{table[i][3]} | {table[i][4]} | {table[i][5]} | {table[i][6]} | {table[i][7]} |')

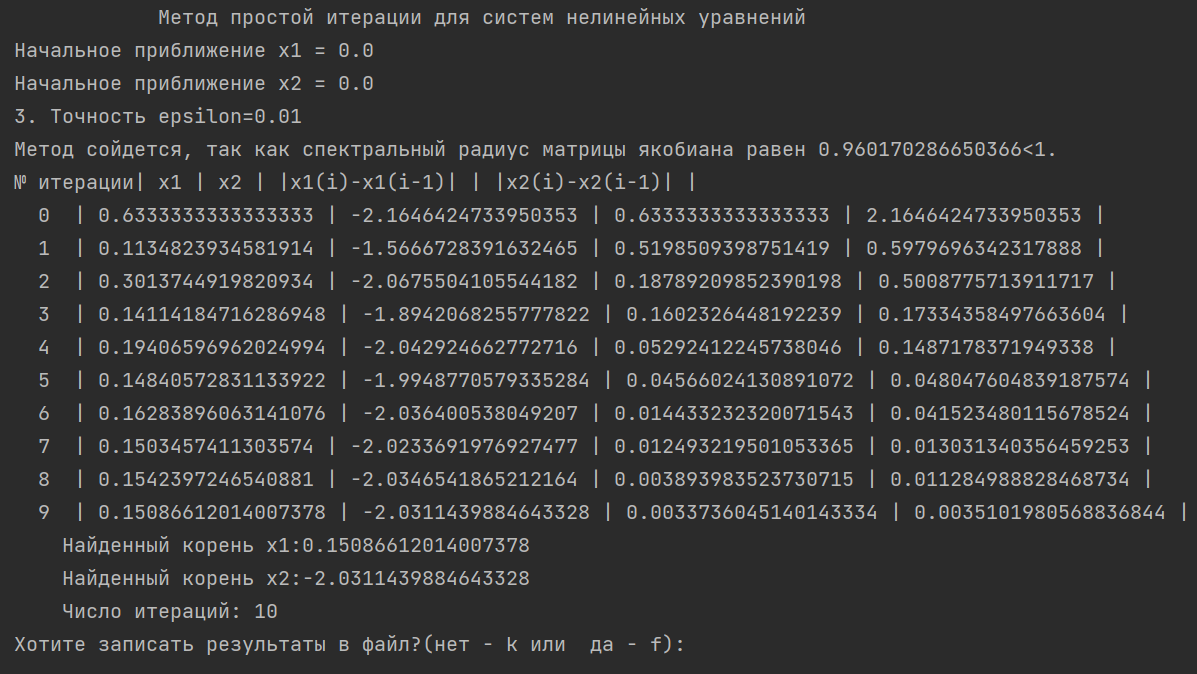
from NonlinearEquations.NonlinearEquationsSolver import NonlinearEquationsSolver  
from NonlinearEquations.Equations import EQUATIONS  
from NonlinearEquations.Equations import \*  
  
AMOUNT\_OF\_COLUMNS\_SECANT = 6  
  
  
class MethodSecant(NonlinearEquationsSolver):  
 # Метод секущих  
 def methodSecant(self, number\_of\_equatation: int):  
 a = self.getLeftBorder()  
 b = self.getRightBorder()  
 eps = self.getEpsilon()  
 print('\t\t\tМетод секущих')  
 if checkConvergenceCondition(a, b, number\_of\_equatation):  
 print('Достаточное условие сходимости выполянется. Вычисляем начальные приближения x0 и x1.')  
 if self.calculateStart(a, number\_of\_equatation) > 0:  
 b = a - a \* 0.25  
 elif self.calculateStart(b, number\_of\_equatation) > 0:  
 a = b  
 b = a + a \* 0.25  
 else:  
 print(  
 'Достаточное условие сходимости не выполнилось. На интервале или нет корней, или существует несколько.')  
 return  
 print(f'1. Значение x0 ={a}')  
 print(f'2. Значение x1 ={b}')  
 print(f'3. Точность epsilon={eps}')  
 maxIterationNumber = self.calculateMaxIteration(eps)  
 iterations = [[0.0 for x in range(AMOUNT\_OF\_COLUMNS\_SECANT)]  
 for x in range(maxIterationNumber)]  
 count\_of\_iterations = 0  
 for i in range(maxIterationNumber):  
 iterations[i][0] = count\_of\_iterations  
 iterations[i][1] = a  
 iterations[i][2] = b  
 iterations[i][3] = b - (b - a) / (calculateFunctionValue(b, number\_of\_equatation) -  
 calculateFunctionValue(a, number\_of\_equatation)) \* calculateFunctionValue(  
 b, number\_of\_equatation)  
 iterations[i][4] = calculateFunctionValue(iterations[i][3], number\_of\_equatation)  
 iterations[i][5] = abs(iterations[i][3] - iterations[i][2])  
 count\_of\_iterations += 1  
 if iterations[i][5] <= eps:  
 break  
 a, b = iterations[i][2], iterations[i][3]  
 self.printTableForMethodSecant(iterations, count\_of\_iterations)  
 return iterations[count\_of\_iterations - 1][3], iterations[count\_of\_iterations - 1][4], count\_of\_iterations  
  
 def calculateStart(self, x, number\_of\_equation):  
 return calculateFunctionValue(x, number\_of\_equation) \  
 \* calculateDerivativeSecondValue(x, number\_of\_equation)  
  
 def printTableForMethodSecant(self, table, count\_of\_iterations):  
 print('№ итерации| x(i-1) | xi | x(i+1) | f(x(i+1)) | |x(i+1) - xi| |')  
 for i in range(count\_of\_iterations):  
 print(f' {table[i][0]} | {table[i][1]} | {table[i][2]} | '  
 f'{table[i][3]} | {table[i][4]} | {table[i][5]} |')

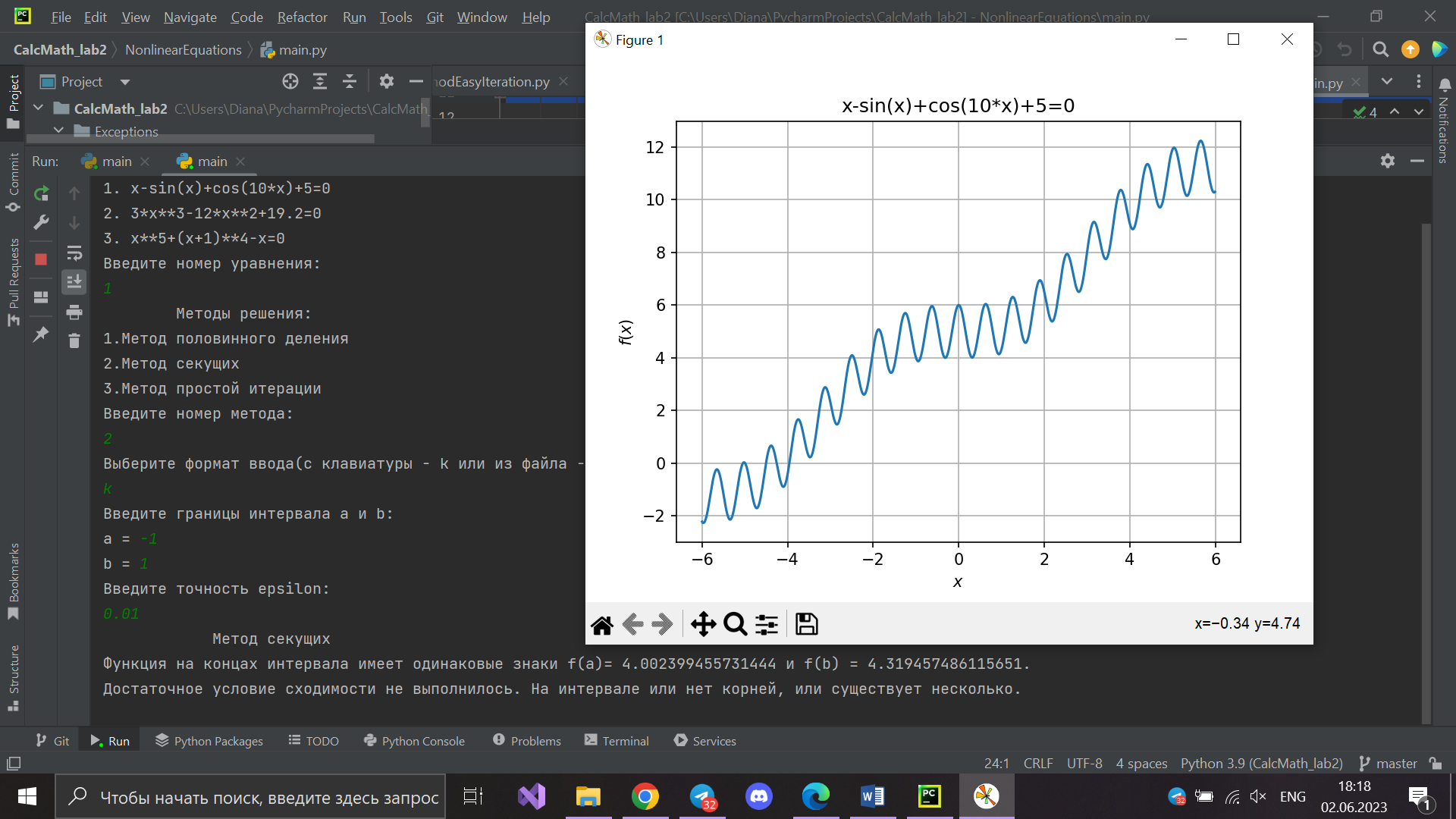
from NonlinearEquationsValidator import NonlinearEquationsValidator  
  
  
# Класс для хранения данных для вычислений  
class NonlinearEquationsSolver:  
  
 def \_\_init\_\_(self, epsilon, left\_border=None, right\_border=None, x0=None):  
 self.\_\_right\_border = right\_border if right\_border is None else NonlinearEquationsValidator.validateNumber(  
 right\_border)  
 self.\_\_left\_border = left\_border if left\_border is None else NonlinearEquationsValidator.validateNumber(  
 left\_border)  
 if self.\_\_right\_border is not None and self.\_\_left\_border is not None:  
 NonlinearEquationsValidator.validateBorders(left\_border, right\_border)  
 self.\_\_x0 = x0 if x0 is None else NonlinearEquationsValidator.validateNumber(x0)  
 self.\_\_epsilon = NonlinearEquationsValidator.validateEpsilon(epsilon)  
  
 def getEpsilon(self):  
 return self.\_\_epsilon  
  
 def getX0(self):  
 return self.\_\_x0  
  
 def getLeftBorder(self):  
 return self.\_\_left\_border  
  
 def getRightBorder(self):  
 return self.\_\_right\_border  
  
 def calculateMaxIteration(self, eps):  
 return int(100 / eps) if eps < 1 else int(100 \* eps)

from Exceptions.IncorrectValueException import IncorrectValueException  
from SystemsOfNonlinearEquations.NonlinearSystemEquationsSolver import NonlinearSystemEquationsSolver  
import numpy as np  
from SystemsOfNonlinearEquations.SystemsOfEquations import \*  
  
AMOUNT\_OF\_COLUMNS\_EASY\_ITERATIONS = 5  
  
  
class MethodEasyIteration(NonlinearSystemEquationsSolver):  
  
 def methodEasyIteration(self, number\_of\_system):  
 starts = self.getStart()  
 eps = self.getEpsilon()  
 print('\t\t\tМетод простой итерации для систем нелинейных уравнений')  
 for i in range(len(starts)):  
 print(f'Начальное приближение x{i + 1} = {starts[i]}')  
 print(f'3. Точность epsilon={eps}')  
 maxIterationNumber = self.calculateMaxIteration(eps)  
 iterations = [[0.0 for x in range(AMOUNT\_OF\_COLUMNS\_EASY\_ITERATIONS)]  
 for x in range(maxIterationNumber)]  
 count\_of\_iterations = 0  
 if not self.checkConditionConvergence(starts, number\_of\_system):  
 return  
 for i in range(maxIterationNumber):  
 iterations[i][0] = count\_of\_iterations  
 iterations[i][1] = calculateEquivalent(number\_of\_system, 1, starts[0], starts[1])  
 iterations[i][2] = calculateEquivalent(number\_of\_system, 0, starts[0], starts[1])  
 iterations[i][3] = abs(iterations[i][1] - starts[0])  
 iterations[i][4] = abs(iterations[i][2] - starts[1])  
 count\_of\_iterations += 1  
 if max(iterations[i][3], iterations[i][4]) <= eps:  
 break  
 starts = iterations[i][1], iterations[i][2]  
 else:  
 raise IncorrectValueException('На данном промежутке либо несколько корней, либо нет корней.')  
 self.printTableForMethodEasy(iterations, count\_of\_iterations)  
 return iterations[count\_of\_iterations - 1][1], iterations[count\_of\_iterations - 1][2], count\_of\_iterations  
  
 def checkConditionConvergence(self, starts, number\_of\_system):  
 yak\_1 = calculateYacobian(number\_of\_system, 0, 0, starts[0], starts[1])  
 yak\_2 = calculateYacobian(number\_of\_system, 0, 1, starts[0], starts[1])  
 yak\_3 = calculateYacobian(number\_of\_system, 1, 0, starts[0], starts[1])  
 yak\_4 = calculateYacobian(number\_of\_system, 1, 1, starts[0], starts[1])  
 J = np.array([[yak\_1, yak\_2],  
 [yak\_3, yak\_4]])  
  
 # Вычисляем собственные значения матрицы Якоби  
 eigvals = np.linalg.eigvals(J)  
  
 # Вычисляем спектральный радиус матрицы  
 rho = np.max(np.abs(eigvals))  
 # Проверяем условие сходимости.  
 if rho < 1:  
 print(f"Метод сойдется, так как спектральный радиус матрицы якобиана равен {rho}<1.")  
 return True  
 else:  
 print(f"Метод не сойдется, так как спектральный радиус матрицы якобиана равен {rho}>1.")  
 return False  
 def printTableForMethodEasy(iterations, table, count\_of\_iterations):  
 print('№ итерации| x1 | x2 | |x1(i)-x1(i-1)| | |x2(i)-x2(i-1)| |')  
 for i in range(count\_of\_iterations):  
 print(f' {table[i][0]} | {table[i][1]} | {table[i][2]} | '  
 f'{table[i][3]} | {table[i][4]} |')

from NonlinearEquationsValidator import NonlinearEquationsValidator  
  
# Класс для хранения данных для вычислений  
class NonlinearSystemEquationsSolver:  
 def \_\_init\_\_(self, epsilon, start: []):  
 self.\_\_start = start  
 self.\_\_epsilon = epsilon  
 def getEpsilon(self):  
 return self.\_\_epsilon  
 def getStart(self):  
 return self.\_\_start  
  
 def calculateMaxIteration(self, eps):  
 return int(100 / eps) if eps < 1 else int(100 \* eps)

***7. Результаты выполнения программы:***

******



***8. Выводы:***

Я изучила численные методы решения нелинейных уравнений и их систем, нашла корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнила программную реализацию методов.