Федеральное государственное автономное

образовательное учреждение высшего образования

Национальный Исследовательский Университет ИТМО

Лабораторная работа №2

«Численное решение нелинейных уравнений и систем»

Дисциплина: Вычислительная математика

Вариант 13

Выполнил: Терехин Никита Денисович

Факультет: Программной инженерии и компьютерной техники

Группа: P3208

Преподаватель: Машина Екатерина Алексеевна

г. Санкт-Петербург, 2024 год

Оглавление

Цель работы.....3
Уравнение.....3
Вычислительная реализация задачи.....3
Решение нелинейного уравнения.....3
Графическое решение.....3
Аналитическое решение.....4
Решение системы нелинейных уравнений.....5
Графическое приближение.....6
Программная реализация задачи.....8
Для нелинейных уравнений.....8
Для системы уравнений.....13
Вспомогательные методы и классы.....15
Результаты выполнения программы.....15
Выводы.....17

Цель работы

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

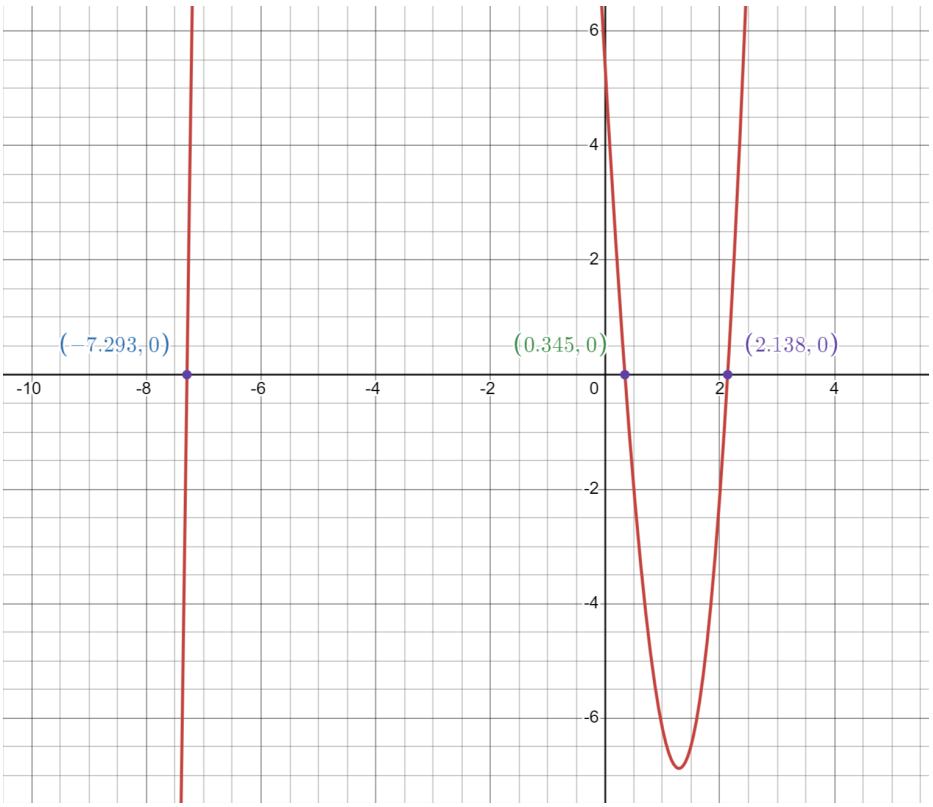
Уравнение

№ варианта	Функция
13	$x^3 + 4,81x^2 - 17,37x + 5,38$

Вычислительная реализация задачи

Решение нелинейного уравнения

Графическое решение



Аналитическое решение

Корень	Левый крайний x_1	Правый крайний x_2	Центральный x_3
Метод	Метод простой итерации	Метод хорд	Метод Ньютона
Интервал изоляции	$[-8; -7]$	$[2; 3]$	$[0; 1]$

Уточним первый корень

Рабочая формула метода простой итерации

$$x_{i+1} = \varphi(x_i) \quad \varphi(x) = x + \lambda f(x)$$

$$f(x) = x^3 + 4,81x^2 - 17,37x + 5,38$$

$$f'(x) = 3x^2 + 9,62\,x - 17,37$$

$$\varphi(x) = x + \lambda(x^3 + 4,81x^2 - 17,37x + 5,38)$$

$$m = \max_{[-8,-7]} (f'(x)) = 97,67 \qquad \text{при } x = -8$$

$$\lambda = -\frac{1}{m} = -0,0102$$

$$\varphi(x) = x - 0,0102(x^3 + 4,81x^2 - 17,37x + 5,38) =$$

$$= -0,0102x^3 - 0,0491x^2 + 1,1772\,x - 0,0549$$

Условие сходимости $|\varphi'(-8)| << 1 \quad |\varphi'(-7)| < 1$ выполнено

№ итерации	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_k - x_{k+1}\rvert$
1	-8,0000	-7,3925	48,1869	0,6075
2	-7,3925	-7,3199	4,4443	0,0726
3	-7,3199	-7,3022	0,8820	0,0177
4	-7,3022	-7,2976	0,0643	0,0046
5	-7,2976	-7,2964	-0,1447	0,0012
6	-7,2964	-7,2961	-0,1996	0,0003

Уточненное значение $x_1 = -7,2961$

Второй корень

Рабочая формула метода хорд

$$x_i = \frac{a_i f(b_i) - b_i f(a_i)}{f(b_i) - f(a_i)}$$

№ шага	a	b	x	$f(a)$	$f(b)$	$f(x)$	$\lvert x_k - x_{k+1} \rvert$
1	2,0000	3,0000	2,0826	-2,1200	23,5600	-0,9007	0,0826
2	3,0000	2,0826	2,1163	23,5600	-0,9007	-0,3585	0,0338
3	3,0000	2,1163	2,1296	23,5600	-0,3585	-0,1390	0,0132
4	3,0000	2,1296	2,1347	23,5600	-0,1390	-0,0533	0,0051

Уточненное значение $x_2 = 2,1347$

Третий корень

Рабочая формула метода Ньютона

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

№ шага	x_k	$f(x_k)$	$f'(x)$	x_{k+1}	$\lvert x_k - x_{k+1} \rvert$
1	0,0000	5,3800	-17,3700	0,3097	0,3097
2	0,3097	0,4911	-14,10261	0,3446	0,0348
3	0,3446	0,0070	-13,69921	0,3451	0,0005

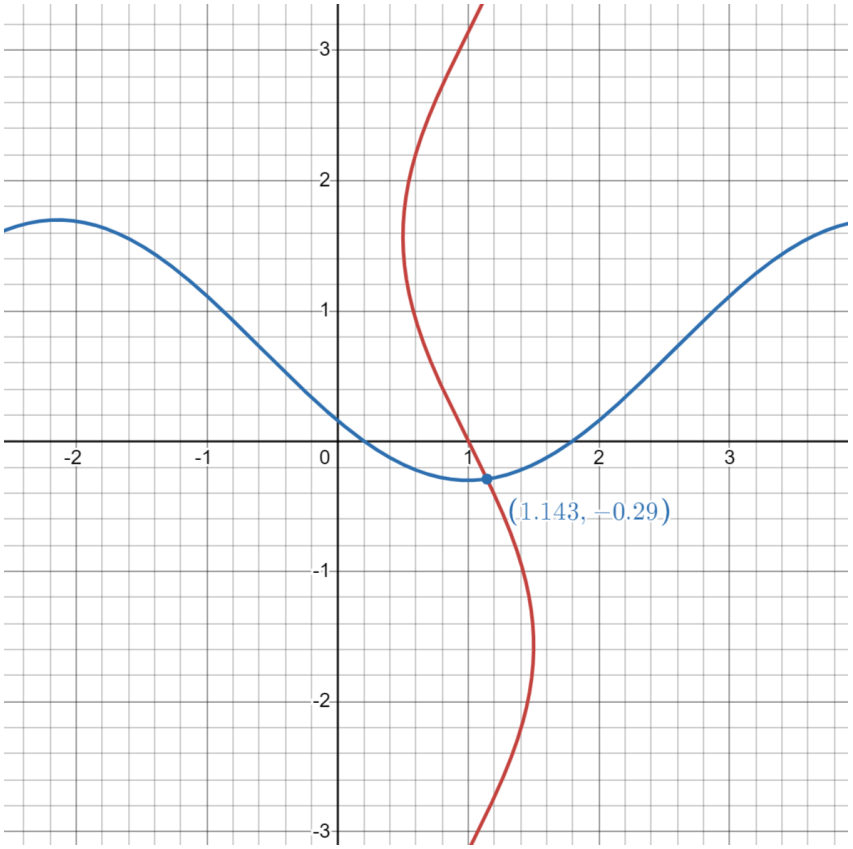
Уточненное значение $x_3 = 0,3451$

Решение системы нелинейных уравнений

№ варианта	Система нелинейных уравнений	Метод

13	$\begin{cases} \sin y + 2x = 0 \\ y + \cos(x-1) = 0,7 \end{cases}$	Метод простых итераций

Графическое приближение



Рабочая формула метода простой итерации для системы

$$\begin{cases} x_1^{(k+1)} = \varphi_1\left(x_1^k,\, x_2^k, \ldots,\, x_n^k\right) \\ x_2^{(k+1)} = \varphi_2\left(x_1^k,\, x_2^k, \ldots,\, x_n^k\right) \\ \ldots \\ x_n^{(k+1)} = \varphi_n\left(x_1^k,\, x_2^k, \ldots,\, x_n^k\right) \end{cases}$$

Приведем систему уравнений к эквивалентному виду

$$\begin{cases} x = 1 - \frac{\sin y}{2} \\ y = 0,7 - \cos(x-1) \end{cases}$$

Интервал изоляции $y \in [-1; 0]$ $x \in [1; 2]$

Проверим условие сходимости:

$\frac{\partial \varphi_1}{\partial x} = 0$ $\frac{\partial \varphi_2}{\partial x} = sin(x - 1)$ $\frac{\partial \varphi_1}{\partial y} = -\frac{\cos y}{2}$ $\frac{\partial \varphi_2}{\partial y} = 0$

$\left|\frac{\partial \varphi_1}{\partial x}\right| + \left|\frac{\partial \varphi_1}{\partial y}\right| < 1$ $\left|\frac{\partial \varphi_2}{\partial x}\right| + \left|\frac{\partial \varphi_2}{\partial y}\right| < 1$

Выберем начальное приближение: $x^{(0)} = 1$ $y^{(0)} = -1$

1 шаг

$x^{(1)} = 1 - \frac{sin(-1)}{2} = 1,4207$ $|x^{(0)} - x^{(1)}| = 0,4207 > \varepsilon$

$y^{(1)} = 0,7 - cos(1 - 1) = -0,3000$ $|y^{(0)} - y^{(1)}| = 0,7000 > \varepsilon$

2 шаг

$x^{(2)} = 1 - \frac{sin(-0,3)}{2} = 1,1478$ $|x^{(2)} - x^{(1)}| = 0,2729 > \varepsilon$

$y^{(2)} = 0,7 - cos(1,4207 - 1) = -0,2128$ $|y^{(2)} - y^{(1)}| = 0,0872 > \varepsilon$

3 шаг

$x^{(3)} = 1 - \frac{sin(-0,2128)}{2} = 1,1056$ $|x^{(2)} - x^{(3)}| = 0,0422 > \varepsilon$

$y^{(3)} = 0,7 - cos(1,1478 - 1) = -0,2891$ $|y^{(2)} - y^{(3)}| = 0,0763 > \varepsilon$

4 шаг

$x^{(4)} = 1 - \frac{sin(-0,2891)}{2} = 1,1425$ $|x^{(4)} - x^{(3)}| = 0,0369 > \varepsilon$

$y^{(4)} = 0,7 - cos(1,1056 - 1) = -0,2944$ $|y^{(4)} - y^{(3)}| = -0,0053 > \varepsilon$

5 шаг

$x^{(5)} = 1 - \frac{sin(-0,2944)}{2} = 1,1451$ $|x^{(4)} - x^{(5)}| = 0,0025 < \varepsilon$

$y^{(5)} = 0,7 - cos(1,1425 - 1) = -0,2899$ $|y^{(4)} - y^{(5)}| = 0,0045 < \varepsilon$

Итоговое приближение $x = 1,1451$ $y = -0,2899$

Программная реализация задачи

Для нелинейных уравнений

```
# класс функции

class Describable:
    option_name: str = 'option'

    def __init__(self, description: str) -> None:
        self.description = description


class Function(Describable):
    option_name = 'function'

    def __init__(self, func: Callable[[float], float],
first_derivation: Callable[[float], float],
                 second_derivation: Callable[[float], float],
description: str) -> None:
        super().__init__(description)
        self.func = func
        self.first_derivation = first_derivation
        self.second_derivation = second_derivation

FUNCTIONS: Final[list[Function]] = [
    Function(lambda x: x * x * x + 4.81 * x * x - 17.37 * x + 5.38,
             lambda x: 3 * x * x + 2 * 4.81 * x - 17.37,
             lambda x: 6 * x + 2 * 4.81,
             'x^3 + 4,81x^2 - 17,37x + 5,38'),
    Function(lambda x: 2 * x * x * x - 1.89 * x * x - 5 * x + 2.34,
             lambda x: 6 * x * x - 2 * 1.89 * x - 5,
             lambda x: 12 * x - 2 * 1.89,
             '2x^3 - 1,89x^2 - 5x + 2,34'),
    Function(lambda x: math.exp(x / 3) - 2 * math.cos(x + 4),
             lambda x: math.exp(x / 3) / 3 + 2 * math.sin(x + 4),
             lambda x: math.exp(x / 3) / 9 + 2 * math.cos(x + 4),
             'e^(x / 3) - 2cos(x + 4)'),
    Function(lambda x: x ** 2 - math.exp(x),
             lambda x: 2 * x - math.exp(x),
             lambda x: 2 - math.exp(x),
             'x^2 - e^x')
]
```

```
# классы методов

class Method(Describable):
```

```
    option_name = 'method'

    def __init__(self, description: str) -> None:
        super().__init__(description)
        self.step: int = 0
        self.prev: float = math.inf
        self.eps: float = 0.01
        self.b: float = 10
        self.a: float = -10
        self.func: Function | None = None

    @abstractmethod
    def execute(self) -> tuple[float, int, float]:
        pass

    def set_arguments(self, func: Function, a: float, b: float,
eps: float) -> None:
        self.func = func
        self.a = a
        self.b = b
        self.eps = eps


class DichotomyMethod(Method):
    def __init__(self) -> None:
        super().__init__('Dichotomy method')

    def execute(self) -> tuple[float, int, float]:
        self.step = 0
        self.prev = 0
        if self.func is not None:
            ans: float = self.do_iteration(self.func.func)
            return ans, self.step, self.func.func(ans)
        else:
            raise TypeError('Function is not defined')

    def do_iteration(self, func: Callable[[float], float]) ->
float:
        self.step += 1
        x: float = (self.a + self.b) / 2
        if abs(x - self.prev) < self.eps:
            return x
        self.prev = x
        if func(x) * func(self.a) < 0:
            self.b = x
            return self.do_iteration(func)
        else:
            self.a = x
```

```
            return self.do_iteration(func)


class NewtonMethod(Method):
    def __init__(self) -> None:
        super().__init__("Newton's method")

    def execute(self) -> tuple[float, int, float]:
        self.step = 0
        if self.func is not None:
            self.set_first_approx(self.func)
            ans: float = self.do_iteration(self.func)
            return ans, self.step, self.func.func(ans)
        else:
            raise TypeError('Function is not defined')

    def do_iteration(self, func: Function):
        self.step += 1
        x: float = self.prev - func.func(self.prev) /
func.first_derivation(self.prev)
        if abs(x - self.prev) < self.eps:
            return x
        self.prev = x
        return self.do_iteration(func)

    def set_first_approx(self, func: Function) -> None:
        while True:
            if func.func(self.a) * func.second_derivation(self.a) >
0:
                self.prev = self.a
                break
            elif func.func(self.b) * func.second_derivation(self.b)
> 0:
                self.prev = self.b
                break
            else:
                x: float = (self.a + self.b) / 2
                self.step += 1
                if func.func(self.a) * func.func(x) < 0:
                    self.b = x
                else:
                    self.a = x


class SecantsMethod(NewtonMethod):
    def __init__(self) -> None:
        super().__init__()
        self.description = 'Secants Method'
```

```
        self.pprev: float = math.inf

    def execute(self) -> tuple[float, int, float]:
        self.step = 0
        if self.func is not None:
            self.set_first_approx(self.func)
            ans: float = self.do_iteration(self.func)
            return ans, self.step, self.func.func(ans)
        else:
            raise TypeError('Function is not defined')

    def do_iteration(self, func: Function):
        self.step += 1
        x: float = (self.prev - (self.prev - self.pprev) /
(func.func(self.prev) - func.func(self.pprev))
                    * func.func(self.prev))
        if abs(x - self.prev) < self.eps:
            return x
        self.prev = x
        return self.do_iteration(func)

    def set_first_approx(self, func: Function) -> None:
        super().set_first_approx(func)
        self.pprev = self.prev + self.eps


class SimpleIterationMethod(Method):
    def __init__(self) -> None:
        super().__init__('Simple Iteration Method')
        self.scale = 100
        self.param: float = 0

    def execute(self) -> tuple[float, int, float]:
        if self.func is not None:
            phi: Callable[[float], float] = lambda x: x +
self.param * self.func.func(x)
            if self.check_convergence(self.func):
                ans: float = self.do_iteration(phi)
                return ans, self.step, self.func.func(ans)
            else:
                print('Convergence condition not met. Try to use
smaller interval')
                return 0, 0, 0
        else:
            raise TypeError('Function is not defined')

    def do_iteration(self, phi: Callable[[float], float]) -> float:
        self.step += 1
```

```
        x: float = phi(self.prev)
        if abs(x - self.prev) < self.eps:
            return x
        self.prev = x
        return self.do_iteration(phi)

    def set_arguments(self, func: Function, a: float, b: float,
eps: float) -> None:
        super().set_arguments(func, a, b, eps)
        max_val: float = max([func.first_derivation(x / self.scale)
for x in range(int(self.a * self.scale),

int(self.b * self.scale))])
        self.param = 1 / max_val
        if func.first_derivation(self.a) > 0 and
func.first_derivation(self.b) > 0:
            self.param *= -1

    def check_convergence(self, func: Function) -> bool:
        phi_derivation: Callable[[float], float] = lambda x: 1 +
self.param * func.first_derivation(x)
        self.prev = self.a if phi_derivation(self.a) <
phi_derivation(self.b) else self.b
        return max(phi_derivation(self.a), phi_derivation(self.b))
< 1
```

```
# для рисования графиков

def draw_and_show(function: Callable[[float], float], point:
tuple[float, float] | None = None) -> list[float]:
    x: list[float] = [i / SCALE - GRID for i in range(2 * GRID *
SCALE)]
    y: list[float] = [function(num) for num in x]
    bounds: list[float] = [x[i] for i in range(1, len(y)) if (y[i -
1] * y[i] < 0)]

    ax: Axes = plt.axes()
    ax.spines['left'].set_position('zero')
    ax.spines['bottom'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    if bounds:
        l_limit: float = min(-4.0, bounds[0], bounds[-1]) - 1
        r_limit: float = max(bounds[0], bounds[-1], 4.0) + 1
    else:
        l_limit = -GRID
        r_limit = GRID
```

```
    ax.grid(which='major', alpha=0.5)
    ax.grid(which='minor', alpha=0.2)
    ax.set_xticks([i * 0.5 - GRID for i in range(GRID * 4)],
minor=True)
    ax.set_yticks([i * 0.5 - GRID for i in range(GRID * 4)],
minor=True)
    ax.set_xticks([i * 2 - GRID for i in range(GRID)])
    ax.set_yticks([i * 2 - GRID for i in range(GRID)])
    ax.set_xlim(l_limit, r_limit)
    ax.set_ylim(-GRID, GRID)

    plt.plot(x, y, linewidth=2)
    if point is not None:
        plt.plot(point[0], point[1], 'bo')
        plt.annotate(f'[{round(point[0], 2)}, {round(point[1],
2)}]',
                     xy=(point[0], point[1]), textcoords='offset
points',
                     xytext=(10, 10), ha='right', va='bottom',
fontsize=10,
                     weight='bold', color='darkblue')
    plt.show()
    return bounds
```

Для системы уравнений

```
# класс системы

class FunctionSystem(Describable):
    option_name = 'system'

    def __init__(self, first: Callable[[float, float], float],
second: Callable[[float, float], float],
                 first_x_derivation: Callable[[float, float],
float],
                 first_y_derivation: Callable[[float, float],
float],
                 second_x_derivation: Callable[[float, float],
float],
                 second_y_derivation: Callable[[float, float],
float],
                 first_y_from_x: Callable[[float], float |
list[float]],
                 second_y_from_x: Callable[[float], float |
list[float]], description: str):
        super().__init__(description)
        self.first = first
```

```
        self.second = second
        self.first_x_derivation = first_x_derivation
        self.first_y_derivation = first_y_derivation
        self.second_x_derivation = second_x_derivation
        self.second_y_derivation = second_y_derivation
        self.first_y_from_x = first_y_from_x
        self.second_y_from_x = second_y_from_x

SYSTEMS: Final[list[FunctionSystem]] = [
    FunctionSystem(lambda x, y: math.sin(y) + 2 * x,
                   lambda x, y: math.cos(x - 1) + y - 0.7,
                   lambda x, y: 2,
                   lambda x, y: math.cos(y),
                   lambda x, y: -math.sin(x - 1),
                   lambda x, y: 1,
                   lambda x: [math.asin(-2 * x) + 2 * math.pi * i
if 2 * abs(x) <= 1 else math.nan
                              for i in range(-3, 3)] + [math.asin(2
* x) + 2 * math.pi * i - math.pi if 2 * abs(x) <= 1
                                                       else
math.nan for i in range(-3, 3)],
                   lambda x: 0.7 - math.cos(x - 1),
                   '| sin(y) + 2x = 0\n    | cos(x - 1) + y = 0,7'),
    FunctionSystem(lambda x, y: math.tan(x * y + 0.3) - x ** 2,
                   lambda x, y: 0.5 * x ** 2 + 2 * y ** 2 - 1,
                   lambda x, y: y / (math.cos(x * y + 0.3))**2 - 2
* x,
                   lambda x, y: x / (math.cos(x * y + 0.3))**2,
                   lambda x, y: x,
                   lambda x, y: 4 * y,
                   lambda x: [(math.atan(x**2) + 2 * math.pi * i -
0.3) / x if x != 0
                              else math.inf for i in range(-5, 5)],
                   lambda x: [math.sqrt((1 - 0.5 * x**2) / 2) if
abs(x)**2 * 0.5 <= 1 else math.nan,
                              -math.atan(math.sqrt((1 - 0.5 *
x**2)) / 2) if abs(x)**2 * 0.5 <= 1 else math.nan],
                   '| tg(xy + 0,3) = x^2\n    | 0.5x^2 + 2y^2 = 1')
]
```

```
def get_cramer_answer(mat: list[list[float]]) -> tuple[float,
float]:
    det_a: float = mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]
    det_x: float = mat[0][2] * mat[1][1] - mat[1][2] * mat[0][1]
    det_y: float = mat[0][0] * mat[1][2] - mat[1][0] * mat[0][2]
    return det_x / det_a, det_y / det_a

# реализация метода Ньютона
```

```
def calculate_multiple_equations() -> None:
    sys: FunctionSystem = request_from_list(SYSTEMS)
    draw_and_show_system(sys.first_y_from_x, sys.second_y_from_x)
    reader: AbstractReader = request_from_list(READERS)
    x, y, precision = reader.read_tuple('Input approximation point
using x and y coordinates: ')

    while True:
        mat = [[sys.first_x_derivation(x, y),
sys.first_y_derivation(x, y), -sys.first(x, y)],
               [sys.second_x_derivation(x, y),
sys.second_y_derivation(x, y), -sys.second(x, y)]]
        delta_x, delta_y = get_cramer_answer(mat)
        x += delta_x
        y += delta_y
        if abs(delta_y) < precision and abs(delta_x) < precision:
            break

    print("Newton's method calculations: ", round(x, 3), round(y,
3))
    draw_and_show_system(sys.first_y_from_x, sys.second_y_from_x,
(x, y))
```

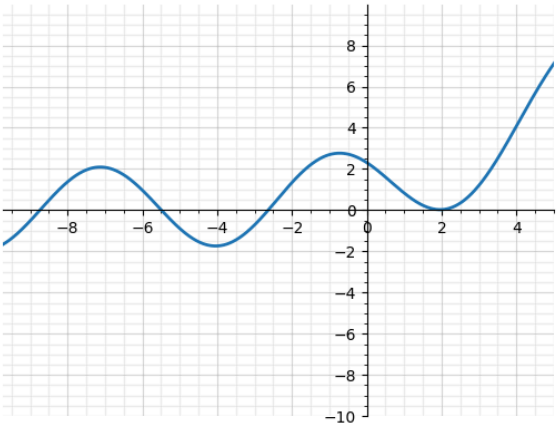
Вспомогательные методы и классы

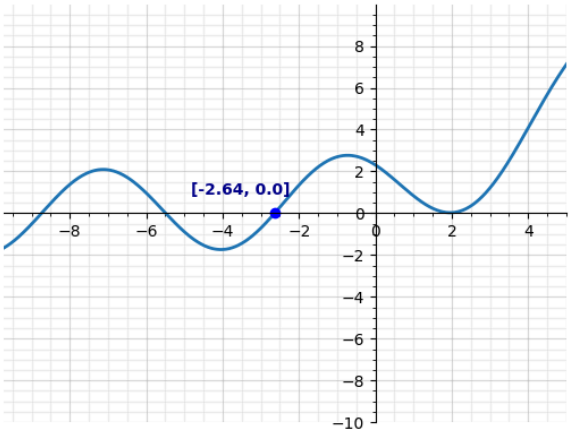
https://github.com/ITerNik/Computational-Math-2024/tree/main/P3208/Terekhin_367558/lab2

Результаты выполнения программы

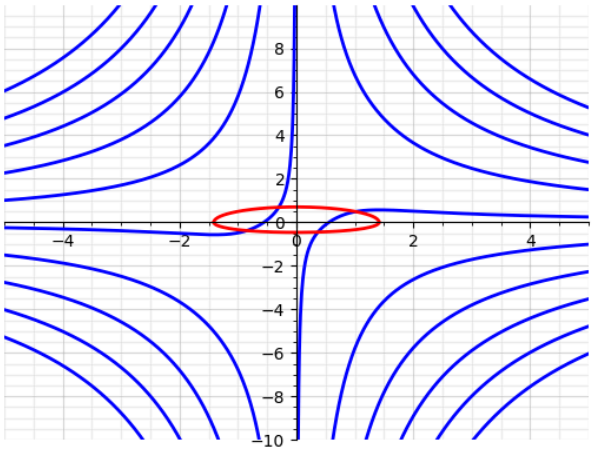
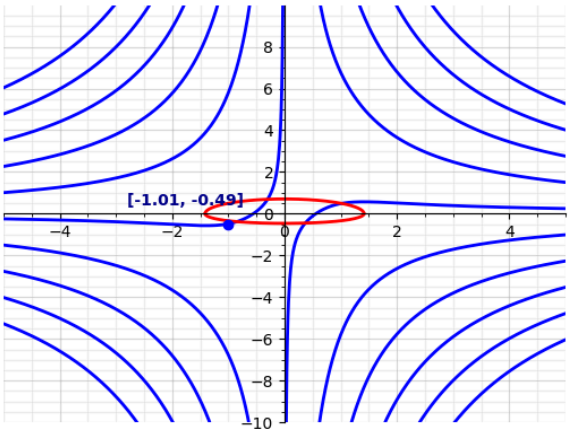
```
1. x^3 + 4,81x^2 - 17,37x + 5, 38
2. x^3 - 1,89x^2 - 5x + 2,34
3. e^(x / 3) - 2cos(x + 4)
4. x^2 - e^x
Choose function:
-1
No such function. Try again
letters
No such function. Try again
3
1.From console
2. From file
Choose option:
1
Input first approximation interval using two numbers:
```

```
-4 -2
Input precision:
0.001
1. Dichotomy method
2.Newton's method
3.Secants Method
4.Simple Iteration Method
Choose method:
3
Final answer: -2.6382,
Steps: 5,
Function value: 0.0
```






```
1. | sin(y) + 2x = 0
   | cos(x - 1) + y = 0,
2. | tg(xy + 0,3) = x^2
   | 0.5x^2 + 2y^2 = 1
Choose system:
2
1. From console
2. From file
Choose option:
2
Enter file name with extension:
text.txt
No such file. Try again:
test.txt
Newton's method calculations:  -1.014 -0.493
```

Выводы

В ходе выполнения работы удалось изучить численные методы решения нелинейных уравнений и их систем, найти корни нелинейных уравнений и систем нелинейных уравнений, выполнить программную реализацию методов

Получилось понять как автоматизировать численные методы из математики, переносить их в код с учетом ошибок ввода пользователей и погрешности

Все методы для решения единичного уравнения показывают в равной степени эффективную работу. Некоторые методы сильно зависят от выбора изоляции корней. Итерационный метод, хоть и прост в реализации, может расходиться на большом интервале изоляции

В системах уравнений также важно начальное приближение. Скорость сходимости метода простых итераций в этом плане зависима