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 № 10

Выполнил:
Мамонтов Г. А.

Преподаватели:
Машина Е. А.
Малышева Т. А.

Санкт-Петербург, 2025 г

# Цель работы

Целью данной работы является приобретение навыков в анализе и программировании определенных численных методов для решения систем линейных алгебраических уравнений.

# Порядок выполнения работы

Решим нелинейное уравнение с помощью данного метода, заполнив таблицу. Затем реализуем численный метод на одном из доступных языков программирования.

# Рабочие формулы используемых методов

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

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

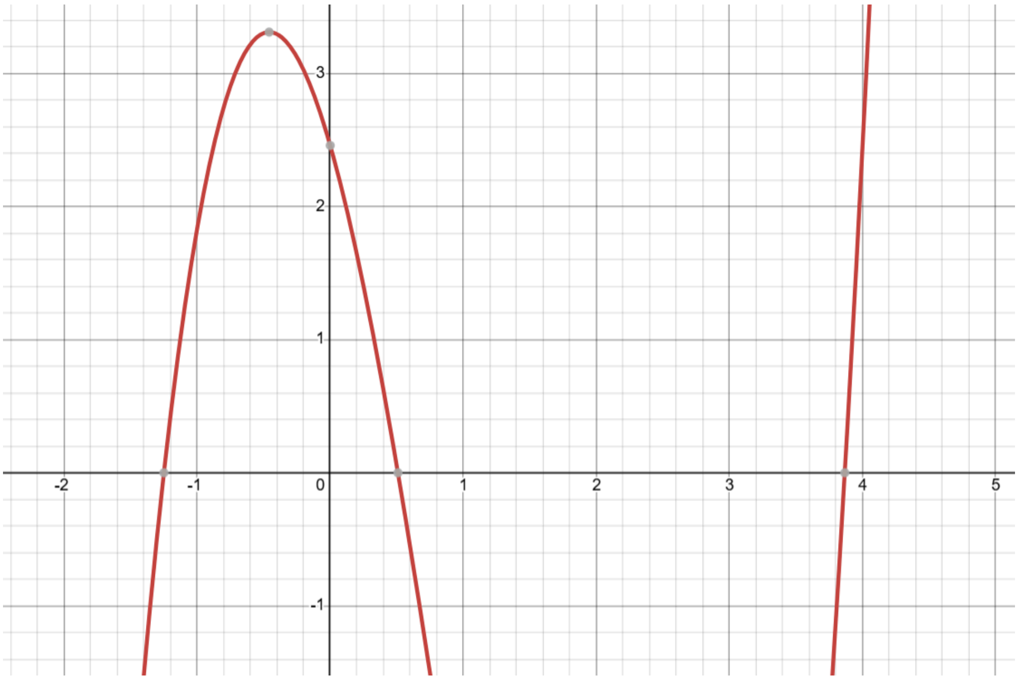
$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i) \qquad i = 1,2\ldots$$

Рабочая формула метода: $x_{i+1} = \varphi(x_i)$

# Графики функций на исследуемом интервале

Исследуемая функция: $x^3 - 3,125x^2 - 3,5x + 2,458$

График функции:




Из графика можно легко понять интервалы изоляции корней:
$[-2;\ -1],\ [0;\ 1],\ [3;\ 4]$

## Заполненные таблицы вычислительной части

Уточним корни с помощью указанных методов:

**Правый корень: Метод Ньютона**

Выбор начального приближения: проанализировав выпуклость графика и значения в концах отрезка приходим к выводу, что начальное приближение: $x = 4$ ; $\varepsilon = 0.01$

| № итерации | $x_k$ | $f(x_k)$ | $f'(x_k)$ | $x_{k+1}$ | $\lvert x_{k+1} - x_k\rvert$ |
|---|---|---|---|---|---|
| 1 | 4 | 2.458 | 19.5 | 3.874 | 0.126 |
| 2 | 3.874 | 0.140 | 17.311 | 3.866 | 0.008 |

Итог уточнения: x = 3.866

**Левый корень: Метод половинного деления**

Начальный интервал: $[-\ 2 : \ -\ 1]$

| № шага | a | b | x | f(a) | f(b) | f(x) | $\lvert a - b \rvert$ |
|---|---|---|---|---|---|---|---|
| 1 | -2 | -1 | -1.5 | -11.042 | 1.833 | -2.698 | 1 |
| 2 | -1.5 | -1 | -1.25 | -2.698 | 1.833 | -0.003 | 0.5 |

$\lvert -\ 0.003 \rvert < \ 0.01$ ⇒ выполнен критерий завершения

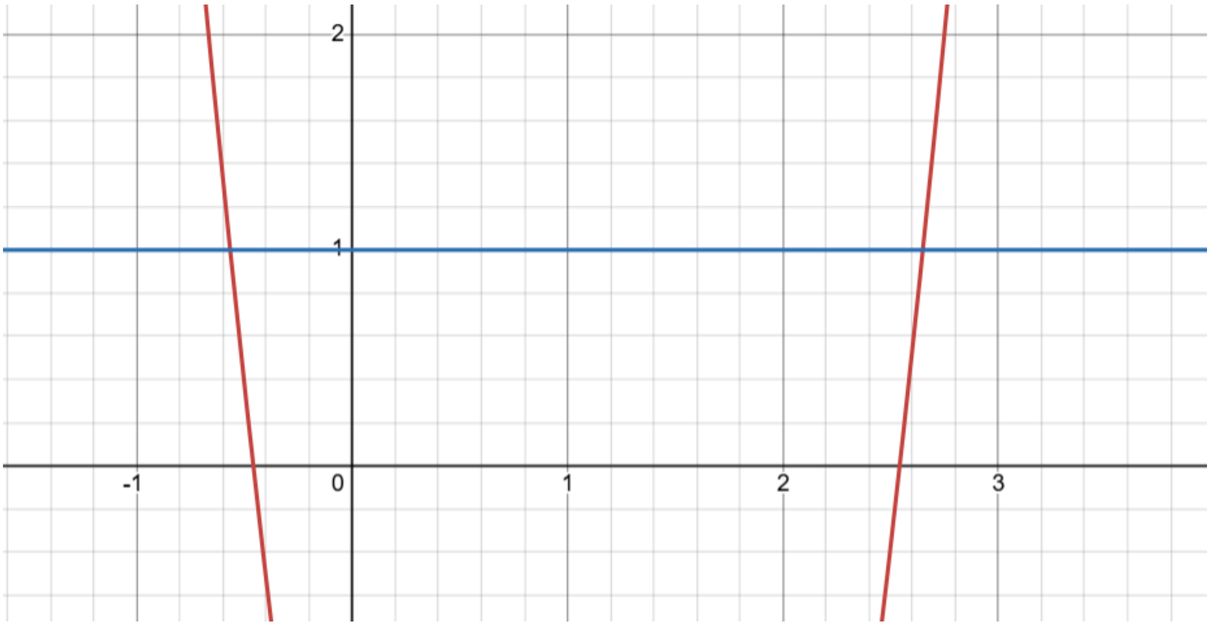
Итог уточнения: x = -1.25

**Средний корень: Метод простой итерации**

λ = 0.1

$x_k = x_{k-1} + \lambda * f(x_{k-1})$

Проверка:



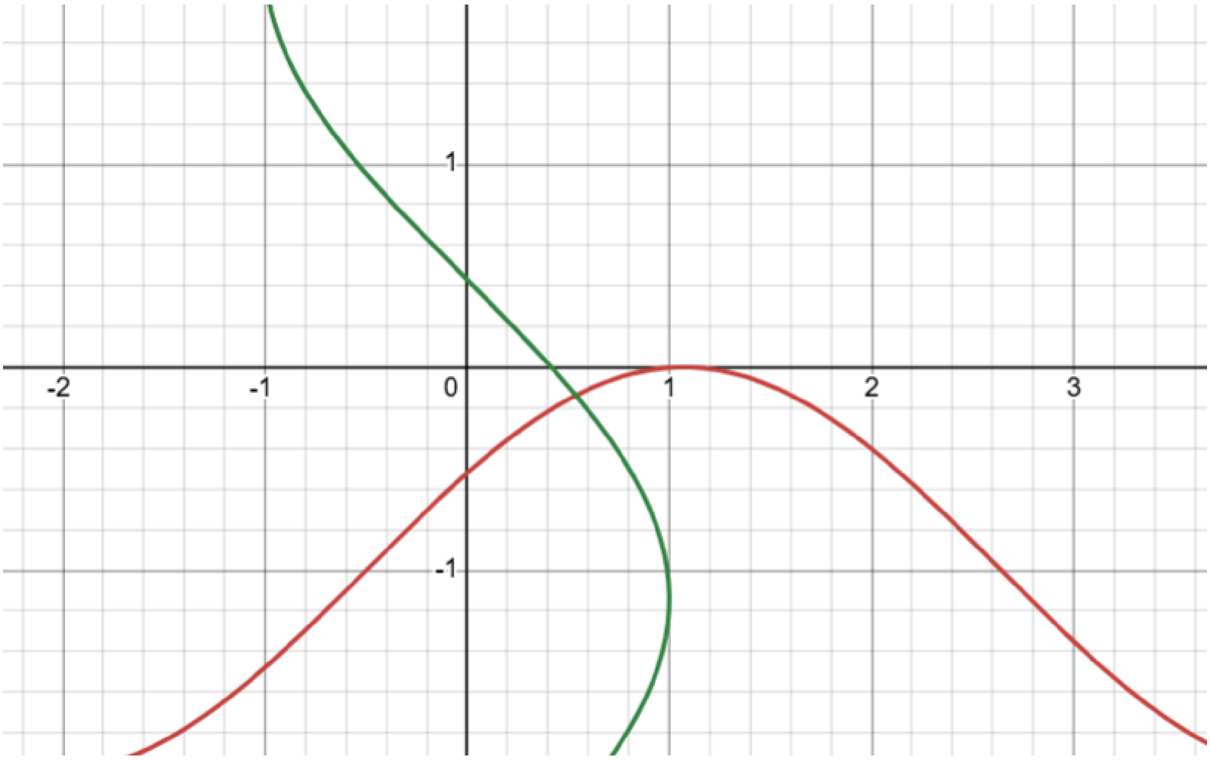

Как видно из графика, производная на всем отрезке [0 ; 1] не превышает 1, что и является условием сходимости

| № итерации | $x_k$ | $x_{k+1}$ | $f(x_{k+1})$ | $\lvert x_{k+1} - x_k \rvert$ |
|---|---|---|---|---|
| 1 | 0.5 | 0.505 | 0.021 | 0.005 |

# Подробное решение системы нелинейных уравнений

**Отделим корни заданной системы нелинейных уравнений графически:**

Система:
$sin(x + 5) - y = 1$
$cos(y - 2) + x = 0$




Как видно из графиков функций, составляющих систему, корень будет на промежутке $[0, 1]$

**Для начала проверим сходимость метода**

$$J = \left[\frac{\frac{\delta\varphi_1}{\delta x}\ \frac{\delta\varphi_1}{\delta y}}{\frac{\delta\varphi_2}{\delta x}\ \frac{\delta\varphi_2}{\delta y}}\right] = \left[\frac{0 \quad sin(y-2)}{cos(x+0.5)\ \ 0}\right]$$

Проверим, что спектральный радиус < 1

$$\rho = \sqrt{|sin(y - 2) \cdot cos(x + 0.5)|}$$

Судя по графику, x находится где-то между 0.4 и 0.6, y - между -0.1 и -0.3

$x \in [0.4,\ 0.6] \Rightarrow x + 0.5 \in [0.9,\ 1.1] \Rightarrow |cos(x + 0.5)| \leq 0.62$

$y \in [-0.1,\ -0.3] \Rightarrow y - 2 \in [-2.1,\ -2.3] \Rightarrow |sin(y - 2)| \leq 0.87$

$\sqrt{0.62 \cdot 0.87} \approx 0.73 < 1$

Значит метод итераций сойдется в этой области

**Используя метод простой итерации решим систему нелинейных уравнений с точностью до 0.01**

Преобразуем систему:
$x = -cos(y - 2)$
$y = sin(x + 0.5) - 1$

Выполним итерации при начальном приближении $x = 0.5$ ; $y = -0.2$

| № | $x_n$ | $y_n$ | $x_{n+1}$ **= -cos(**$y_n$ **- 2)** | $y_{n+1}$ **= sin(**$x_n$ **+ 0.5) - 1** | **Δ*x*** | **Δ*y*** |
|---|---|---|---|---|---|---|
| 0 | 0.5000 | -0.2000 | -cos(-2.2000) ≈ 0.5885 | sin(1.0) - 1 ≈ -0.1585 | — | — |
| 1 | 0.5885 | -0.1585 | -cos(-2.1585) ≈ 0.5527 | sin(1.0885) - 1 ≈ -0.1160 | 0.0358 | 0.0425 |
| 2 | 0.5527 | -0.1160 | -cos(-2.1160) ≈ 0.5166 | sin(1.0527) - 1 ≈ -0.1309 | 0.0361 | 0.0149 |
| 3 | 0.5166 | -0.1309 | -cos(-2.1309) ≈ 0.5302 | sin(1.0166) - 1 ≈ -0.1464 | 0.0136 | 0.0155 |
| 4 | 0.5302 | -0.1464 | -cos(-2.1464) ≈ 0.5397 | sin(1.0302) - 1 ≈ -0.1429 | 0.0095 | 0.0035 |

После четвертой итерации:
$\Delta x = 0.0095 < 0.01;\ \Delta y = 0.0035 < 0.01$
Условие точности выполнено, значит завершаем итерации

# Листинг программы

Untitled2.ipynb

```python
import matplotlib.pyplot as plt
import numpy as np
import math
from typing import Callable, Tuple, Optional

class NonlinearEquationSolver:
    def __init__(self):
        self.equations = {
            1: ("x^3 - x + 1", lambda x: x**3 - x + 1),
            2: ("sin(x) + 0.5", lambda x: math.sin(x) + 0.5),
            3: ("e^x - 2", lambda x: math.exp(x) - 2),
            4: ("x^2 - 4", lambda x: x**2 - 4),
            5: ("cos(x) - x", lambda x: math.cos(x) - x)
        }
        self.methods = {
            1: ("Метод хорд", self.chord_method),
            2: ("Метод секущих", self.secant_method),
            3: ("Метод простой итерации", self.simple_iteration_method)
        }

    def display_menu(self, items: dict, title: str) -> int:
        """Отображает меню и возвращает выбор пользователя"""
        print(f"\n{title}:")
        for key, (desc, _) in items.items():
            print(f"{key}. {desc}")
        while True:
            try:
                choice = int(input("Выберите пункт: "))
                if choice in items:
                    return choice
                print("Неверный выбор. Попробуйте снова.")
            except ValueError:
                print("Введите число.")

    def get_input_source(self) -> Tuple[float, float, float]:
        """Получает источник ввода данных"""
        print("\nВыберите источник ввода данных:")
        print("1. Клавиатура")
        print("2. Файл")
        choice = input("Ваш выбор: ")

        if choice == "1":
            a = float(input("Введите левую границу интервала a: "))
```

```
            b = float(input("Введите правую границу интервала b: "))
            eps = float(input("Введите точность eps: "))
            return a, b, eps
        elif choice == "2":
            filename = input("Введите имя файла: ")
            with open(filename, 'r') as f:
                a, b, eps = map(float, f.readline().split())
            return a, b, eps

    def get_output_destination(self) -> str:
        """Получает место вывода результатов"""
        print("\nВыберите куда выводить результаты:")
        print("1. Экран")
        print("2. Файл")
        choice = input("Ваш выбор: ")
        return "screen" if choice == "1" else "file"

    def verify_interval(self, f: Callable[[float], float], a: float, b:
float) -> bool:
        """Проверяет наличие корня на интервале"""
        if a >= b:
            print("Ошибка: a должно быть меньше b")
            return False

        fa = f(a)
        fb = f(b)

        if fa * fb > 0:
            print(f"Ошибка: На интервале [{a}, {b}] функция не меняет
знак (f(a)={fa:.3f}, f(b)={fb:.3f})")
            return False

        return True

    def find_initial_guess(self, f: Callable[[float], float], a: float,
b: float) -> float:
        """Выбирает начальное приближение для методов"""
        fa = f(a)
        fb = f(b)
        return a if abs(fa) < abs(fb) else b

    def chord_method(self, f: Callable[[float], float], a: float, b:
float, eps: float) -> Tuple[float, float, int]:
```

```
        """Метод хорд"""
        x_prev = a
        x_curr = b
        iterations = 0

        while True:
            iterations += 1
            x_next = x_curr - f(x_curr) * (x_curr - x_prev) /
(f(x_curr) - f(x_prev))

            if abs(x_next - x_curr) < eps:
                break

            x_prev, x_curr = x_curr, x_next

        return x_next, f(x_next), iterations

    def secant_method(self, f: Callable[[float], float], a: float, b:
float, eps: float) -> Tuple[float, float, int]:
        """Метод секущих"""
        x0 = a
        x1 = b
        iterations = 0

        while True:
            iterations += 1
            if abs(f(x1) - f(x0)) < 1e-12:
                print("Предупреждение: знаменатель близок к нулю в
методе секущих")
                break

            x_next = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))

            if abs(x_next - x1) < eps:
                break

            x0, x1 = x1, x_next

        return x_next, f(x_next), iterations

    def simple_iteration_method(self, f: Callable[[float], float], a:
float, b: float, eps: float) -> Tuple[Optional[float], Optional[float],
int]:
```

```
    """Метод простой итерации с строгой проверкой условия
сходимости"""
    try:
        # Шаг для вычисления производных
        h = max((b - a) / 100, 1e-5)
        x_points = np.linspace(a, b, 100)

        # Вычисляем максимальное значение производной f
        df_values = []
        for x in x_points:
            try:
                df = (f(x + h) - f(x)) / h
                df_values.append(abs(df))
            except:
                continue

        if not df_values:
            print("Ошибка: не удалось вычислить производную функции
на интервале")
            return None, None, 0

        df_max = max(df_values)

        if df_max < 1e-12:
            print("Ошибка: производная слишком близка к нулю")
            return None, None, 0

        # Выбираем параметр преобразования
        lambda_ = 1 / df_max
        phi = lambda x: x - lambda_ * f(x)

        # Проверяем условие сходимости |phi'(x)| < 1
        dphi_values = []
        for x in x_points:
            try:
                dphi = (phi(x + h) - phi(x)) / h
                dphi_values.append(abs(dphi))
            except:
                continue

        if not dphi_values:
            print("Ошибка: не удалось вычислить производную phi(x)")
            return None, None, 0
```

```
            dphi_max = max(dphi_values)

            # Жёсткая проверка условия сходимости
            if dphi_max >= 1:
                print(f"Ошибка: условие сходимости не выполняется
(max|phi'|={dphi_max:.3f} ≥ 1)")
                print("Метод простой итерации не может быть применён")
                return None, None, 0

            # Если условие сходимости выполняется, выполняем итерации
            x_prev = (a + b) / 2
            iterations = 0
            max_iter = 1000

            while iterations < max_iter:
                iterations += 1
                try:
                    x_next = phi(x_prev)
                except:
                    print("Ошибка при вычислении phi(x)")
                    return None, None, 0

                if abs(x_next - x_prev) < eps:
                    return x_next, f(x_next), iterations

                x_prev = x_next

            print("Предупреждение: достигнуто максимальное число
итераций")
            return x_prev, f(x_prev), iterations

        except Exception as e:
            print(f"Ошибка в методе простой итерации: {str(e)}")
            return None, None, 0

    def plot_function(self, f: Callable[[float], float], a: float, b:
float, root: Optional[float] = None):
        """Строит график функции"""
        x = np.linspace(a, b, 400)
        y = np.vectorize(f)(x)

        plt.figure(figsize=(10, 6))
```

```python
        plt.plot(x, y, label="f(x)")
        plt.axhline(0, color='black', linewidth=0.5)
        plt.axvline(0, color='black', linewidth=0.5)

        if root is not None:
            plt.scatter([root], [f(root)], color='red', label=f"Корень ({root:.5f})")

        plt.title("График функции")
        plt.xlabel("x")
        plt.ylabel("f(x)")
        plt.grid(True)
        plt.legend()
        plt.show()

    def run(self):
        """Основной цикл программы"""
        print("Программа для решения нелинейных уравнений")

        # Выбор уравнения
        eq_choice = self.display_menu(self.equations, "Доступные уравнения")
        eq_name, f = self.equations[eq_choice]
        print(f"\nВыбрано уравнение: {eq_name}")

        # Ввод данных
        a, b, eps = self.get_input_source()
        print(f"\nИсходные данные: a={a}, b={b}, eps={eps}")

        # Проверка интервала
        if not self.verify_interval(f, a, b):
            return

        # Выбор метода
        method_choice = self.display_menu(self.methods, "Доступные методы")
        method_name, method_func = self.methods[method_choice]
        print(f"\nВыбран метод: {method_name}")

        # Вычисление корня
        if method_choice == 3:  # Метод простой итерации
            root, f_root, iterations = method_func(f, a, b, eps)
```

```
            # Если метод не выполнился из-за невыполнения условия
сходимости
            if root is None:
                print("\nРасчёт не выполнен из-за невыполнения условий
сходимости.")
                return
        else:
            x0 = self.find_initial_guess(f, a, b)
            root, f_root, iterations = method_func(f, a, b, eps)

        # Вывод результатов
        output_dest = self.get_output_destination()
        result = (f"\nРезультаты для уравнения {eq_name} на интервале
[{a}, {b}]:\n"
                f"Метод: {method_name}\n"
                f"Найденный корень: {root:.8f}\n"
                f"Значение функции в корне: {f_root:.8f}\n"
                f"Количество итераций: {iterations}\n"
                f"Точность: {eps}")

        if output_dest == "screen":
            print(result)
        else:
            with open("result.txt", "w") as f_out:
                f_out.write(result)
            print("Результаты сохранены в файл 'result.txt'")

        # Построение графика
        self.plot_function(f, a, b, root)

if __name__ == "__main__":
    solver = NonlinearEquationSolver()
    solver.run()
```

# Результаты выполнения программы при различных исходных данных

Программа для решения нелинейных уравнений

Доступные уравнения:
1. x^3 - x + 1
2. sin(x) + 0.5
3. e^x - 2
4. x^2 - 4
5. cos(x) - x
Выберите пункт: 1

Выбрано уравнение: x^3 - x + 1

Выберите источник ввода данных:
1. Клавиатура
2. Файл
Ваш выбор: 1
Введите левую границу интервала a: -2
Введите правую границу интервала b: -1
Введите точность eps: 0.001

Исходные данные: a=-2.0, b=-1.0, eps=0.001

Доступные методы:
1. Метод хорд
2. Метод секущих
3. Метод простой итерации
Выберите пункт: 2

Выбран метод: Метод секущих

Выберите куда выводить результаты:
1. Экран
2. Файл
Ваш выбор: 1

Результаты для уравнения x^3 - x + 1 на интервале [-2.0, -1.0]:
Метод: Метод секущих

Найденный корень: -1.32472525
Значение функции в корне: -0.00003110
Количество итераций: 5
Точность: 0.001

## Выводы

Данная лабораторная работа помогла укрепить и применить теоретические знания про численные методы нахождения корней нелинейных уравнений и систем. Также я научился программировать некоторые методы, параллельно узнав для себя несколько дополнительных возможностей языка программирования.