Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 4

по курсу «Численные методы»

Студент группы ИУ9-62Б Нащёкин Н.Д.

Преподаватель: Домрачева А.Б.

Москва, 2025

1 Задача

1. Нарисовать график функции f(x) и найти отрезки, где функция имеет простые корни и отличные от нуля первые две производные.
2. Найти с точностью 0.001 все корни уравнения $f(x) = 0$ методом деления отрезка пополам и методом Ньютона; определить число приближений в каждом случае.
3. Сравнить полученные результаты.

2 Основная теория

Будем предполагать, что на отрезке $[a, b]$ функция $f(x)$ имеет единственный простой корень. Если это не так, разобьём его на несколько отрезков, на каждом из которых выполняется указанное условие и будем искать корни отдельно.

Метод деления отрезка пополам

Метод деления отрезка пополам для нахождения нулей функции заключается в следующем: сначала делим отрезок $[a, b]$ пополам точкой $x = \frac{a+b}{2}$. Из двух получившихся отрезков нужно выбрать тот, на котором находится корень уравнения (где $f(x)$ меняет знак). Если $f(a)f(x) < 0$, то это отрезок $[a, x]$, если же $f(x)f(b) < 0$, то это $[x, b]$. Обозначим его за $[a_1, b_1]$. Повторяем процедуру с новым отрезком. Процедура продолжается до тех пор, пока не будет выполнено $b_k - a_k \leq 2 * \varepsilon$, где ε – требуемая точность вычисления корня. Результатом работы метода является очередная точка $x = \frac{a_k+b_k}{2}$.

Метод Ньютона (метод касательных)

Метод Ньютона является итерационным методом. Для получения $k+1$-ой итерации (точки x_{k+1}) из точки $(x_k, f(x_k))$ на графике функции проводим касательную. Точка пересечения с осью OX и есть следующее приближение. Математически этот процесс можно записать в виде

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \text{ где } k = 0, 1, 2, \ldots$$

Достаточное условие сходимости метода Ньютона – отличие от нуля первых двух производных функции $f(x)$ на отрезке $[a, b]$. Начальное приближение x_0 – тот конец отрезка, где знак функции $f(x)$ совпадает со знаком её второй производной $f''(x)$. Заданная погрешность ε будет достигнута, когда

$$f(x_k)f(x_k + sign(x_k - x_{k-1})\varepsilon) < 0, \text{ где } sign \text{ – функция знака.}$$

3 Практическая реализация

Листинг 1 — реализация программы

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

# x^3 - 5x - 1 = 0
roots_real = [-2.12842, -0.20164, 2.33006]


eps = 0.001


x = Symbol('x')
f = x ** 3 - 5 * x - 1
count_steps_half = 0
count_steps_newton = 0


def half_method(a: float, b: float):
    global count_steps_half
    count_steps_half += 1

    # Тут модификация на редкий случай, если попали в корень точно
    if b - a < 2 * eps or f.subs(x, (a + b) / 2) == 0:
        return (a + b) / 2

    if f.subs(x, a) * f.subs(x, (a + b) / 2) < 0:
        return half_method(a, (a + b) / 2)
    if f.subs(x, (a + b) / 2) * f.subs(x, b) < 0:
        return half_method((a + b) / 2, b)



def newton_method(a: float, b: float):
    global count_steps_newton
    x_k = 0
    x_k_1 = 0
    if f.subs(x, a) * f.diff().diff().subs(x, a) > 0:
        x_k_1 = a
    elif f.subs(x, b) * f.diff().diff().subs(x, b) > 0:
        x_k_1 = b

    x_k = x_k_1 - f.subs(x, x_k_1) / f.diff().subs(x, x_k_1)
    count_steps_newton = 1

    while f.subs(x, x_k) * f.subs(x, x_k + sign(x_k - x_k_1) * eps) >= 0:
        count_steps_newton += 1
```

```python
        x_k_1 = x_k
        x_k = x_k_1 - f.subs(x, x_k_1) / f.diff().subs(x, x_k_1)

    return x_k.evalf()


# По теореме Штурма (была на алгебре)
def find_segments():
    # Многочлен и система Штурма
    poly = Poly(f, x)
    seq = sturm(poly.as_expr(), x)

    # Оценка Коши (граница, где лежат все корни)
    coeffs = poly.all_coeffs()
    a_n = coeffs[0]
    others = coeffs[1:]
    bound = 1 + max(abs(c) / abs(a_n) for c in others)

    # Подсчёт числа смен знака в системе
    def var_count(val):
        vals = [p.subs(x, val).evalf() for p in seq]
        # нули пропускаются (по теореме)
        signs = [1 if v > 0 else -1 for v in vals if v != 0]
        return sum(1 for i in range(len(signs) - 1) if signs[i] != signs[i + 1])

    # Первые две производные
    f1 = f.diff(x)
    f2 = f.diff(x, 2)

    # Рекурсивная изоляция корней на отрезке
    def isolate(a, b):
        va, vb = var_count(a), var_count(b)
        roots_inside = va - vb
        if roots_inside == 0:
            return []

        m = (a + b) / 2
        # если больше одного
        if roots_inside > 1:
            return isolate(a, m) + isolate(m, b)

        # проверка, что f' и f'' не обнуляются на [a, b]
        d1a, d1b = f1.subs(x, a).evalf(), f1.subs(x, b).evalf()
        d2a, d2b = f2.subs(x, a).evalf(), f2.subs(x, b).evalf()
        cond1 = (d1a != 0 and d1b != 0 and d1a * d1b > 0)
        cond2 = (d2a != 0 and d2b != 0 and d2a * d2b > 0)
        if cond1 and cond2:
            return [(float(a), float(b))]
        else:
            # если хотя бы одна из производных обнуляется или меняет знак - дробим дальше
```

```
            return isolate(a, m) + isolate(m, b)

    # Запуск на отрезке [-bound, +bound]
    return isolate(-bound, bound)


def plot_fun():
    f_num = lambdify(x, f, modules=["numpy"])

    x_vals = np.linspace(-3, 3, 1000)
    y_vals = f_num(x_vals)

    plt.plot(x_vals, y_vals, label='f(x) = x^3 - 5x - 1')
    plt.axhline(0, color='gray', linestyle='--')  # ось X
    plt.axvline(0, color='gray', linestyle='--')  # ось Y
    plt.title("График функции f(x)")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.grid(True)
    plt.legend()
    plt.show()


def main():
    plot_fun()

    segments = find_segments()
    global count_steps_half
    print(f'Отрезки, найденные по теореме Штурма: {segments}')

    roots_half = []
    roots_newton = []
    step_counts_half = []
    step_counts_newton = []
    for segment in segments:
        roots_half.append(half_method(segment[0], segment[1]))
        step_counts_half.append(count_steps_half)
        count_steps_half = 0

        roots_newton.append(newton_method(segment[0], segment[1]))
        step_counts_newton.append(count_steps_newton)

    for i in range(len(roots_half)):
        print(f'Корень номер {i + 1}. Реальное значение: {roots_real[i]}. '
            f'Метод деления пополам: {roots_half[i]:.5f}, число шагов: {step_counts_half[i]}. '
            f'Метод Ньютона: {roots_newton[i]:.5f}, число шагов: {step_counts_newton[i]}')
        print(f'Абсолютная погрешность. Для метода деления пополам: '
            f'{abs(roots_real[i] - roots_half[i]):.5f}, '
            f'для метода Ньютона: {abs(roots_real[i] - roots_newton[i]):.5f}')
        print()
```

```python


if __name__ == '__main__':
    main()
```

Мой вариант – 22. По условию требуется искать корни для уравнения $x^3 - 5x - 1 = 0$. При запуске программы график функции строится с помощью библиотеки *matplotlib*. График функции приведён на рисунке 1. Для автоматического поиска всех отрезков, на которых уравнение имеет один корень, а две первых производных не обнуляются, был реализован метод их нахождения с помощью теоремы Штурма и системы Штурма, который изучался на курсе алгебры. Этот метод был немного доработан для выполнения достаточного условия сходимости метода Ньютона: если хотя бы одна из производных обнуляется, отрезок дробится дальше. Полученная функция $find\_segments()$ возвращает список всех отрезков, на которых функция имеет простой корень. В отдельных функциях были реализованы метод деления отрезка пополам и метод Ньютона. Результат работы программы приведён на рисунке 2.

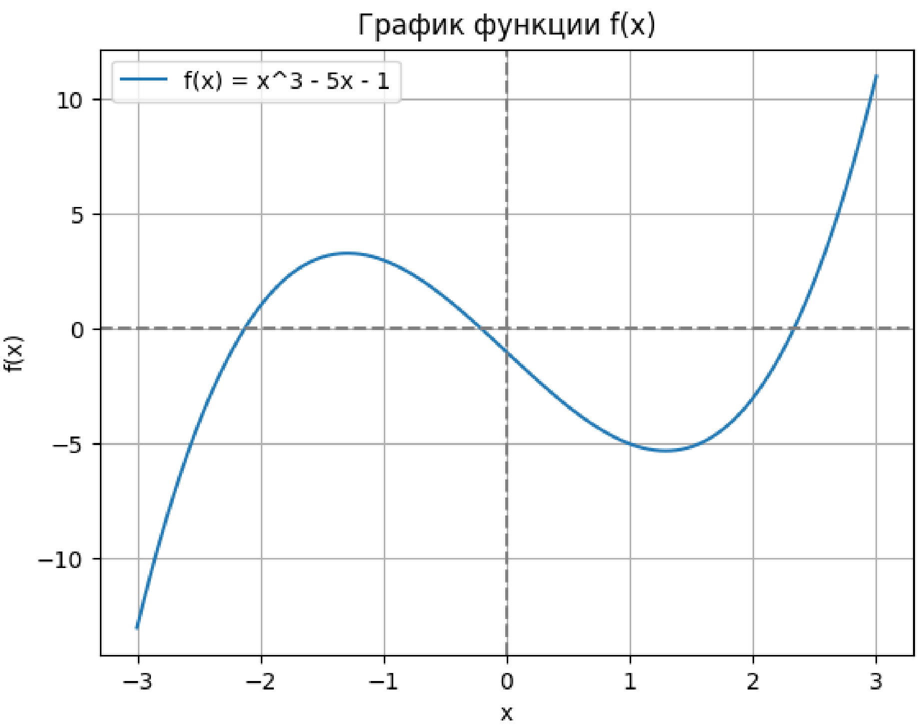

Рис. 1 — График функции

```
eps = 0.001
Отрезки, найденные по теореме Штурма: [(-3.0, -1.5), (-0.375, -0.1875), (1.5, 3.0)]
Корень номер 1. Реальное значение: -2.12842. Метод деления пополам: -2.12915, число шагов: 11. Метод Ньютона: -2.12842, число шагов: 4
Абсолютная погрешность. Для метода деления пополам: 0.00073, для метода Ньютона: 0.00000

Корень номер 2. Реальное значение: -0.20164. Метод деления пополам: -0.20142, число шагов: 8. Метод Ньютона: -0.20162, число шагов: 1
Абсолютная погрешность. Для метода деления пополам: 0.00022, для метода Ньютона: 0.00002

Корень номер 3. Реальное значение: 2.33006. Метод деления пополам: 2.32983, число шагов: 11. Метод Ньютона: 2.33020, число шагов: 3
Абсолютная погрешность. Для метода деления пополам: 0.00023, для метода Ньютона: 0.00014


Process finished with exit code 0
```

Рис. 2 — Результат выполнения программы

4 Вывод

В данной лабораторной работе были реализованы метод деления отрезка пополам и метод Ньютона для приближённого нахождения корней нелинейного уравнения. По результатам работы программы можно сказать, что метод Ньютона требует заметно меньшее количество шагов для схождения, а также во всех трёх случаях корни, найденные методом Ньютона, оказались ближе к точным значениям этих корней. При выполнении работы был реализован поиск отрезков, на которых уравнение имеет один корень, с помощью теоремы Штурма, что делает программу универсальной: можно задать любую функцию $\varphi(x)$ и искать приближённые значения корней для уравнения $\varphi(x) = 0$.