МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники



Лабораторная работа №1

**по дисциплине: «Численные методы решения прикладных задач»**

**«Решение нелинейных алгебраических уравнений»**

Выполнил: Проверил:

Студент гр. «*АПИМ-224*», «*АВТФ*» Профессор Разуваев Владислав ВалерьевичЗеркаль С. М.

« » 20 г. « » 20 г.

(подпись) (подпись)

Новосибирск 2025

**Содержание**

[Цель работы 3](#_Toc208340598)

[Задание 3](#_Toc208340599)

[Ход работы 3](#_Toc208340600)

[Вывод 10](#_Toc208340601)

# Цель работы

Изучить методы решения нелинейных алгебраических уравнений и реализовать их.

# Задание

* Исследовать функцию f(x) на наличие корней графически. Найти и выписать интервалы, на которых существуют единственный корень уравнения.
* Для каждого интервала изоляции [a,b] найти корни уравнения.
* Для каждого интервала изоляции [a,b] с заданной точностью e=0.01; 0.001; 0.0001 найти корни уравнения с использованием метода деления отрезка пополам, метода Ньютона, методапростых итераций.

Вариант задания: 23

Уравнение вида: , далее ЗУ (заданное уравнение)

# Ход работы

Построим график функции

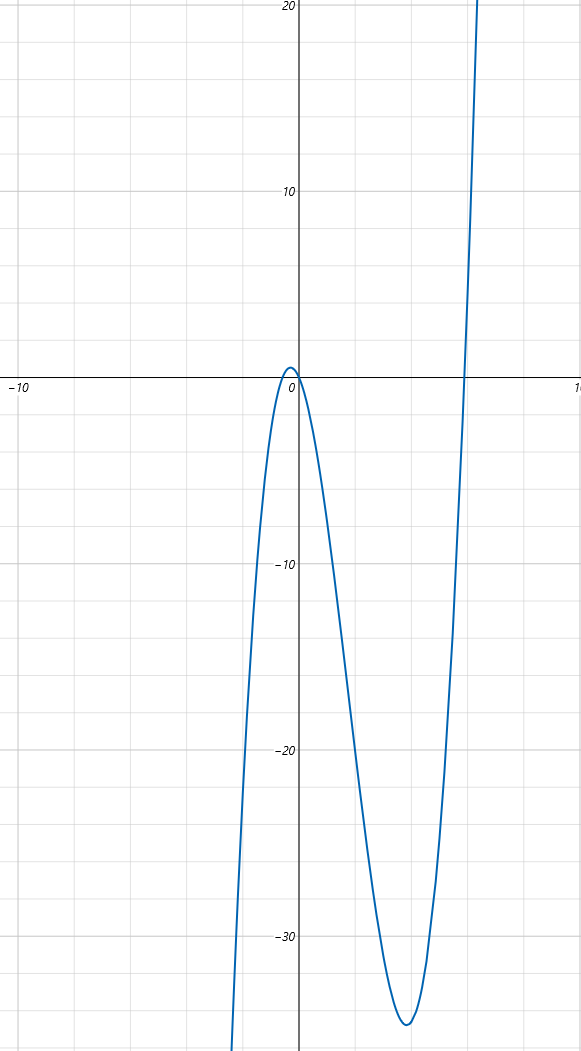


Рисунок 1 График заданной функции

Как видно из графика ЗУ имеет 3 корня.

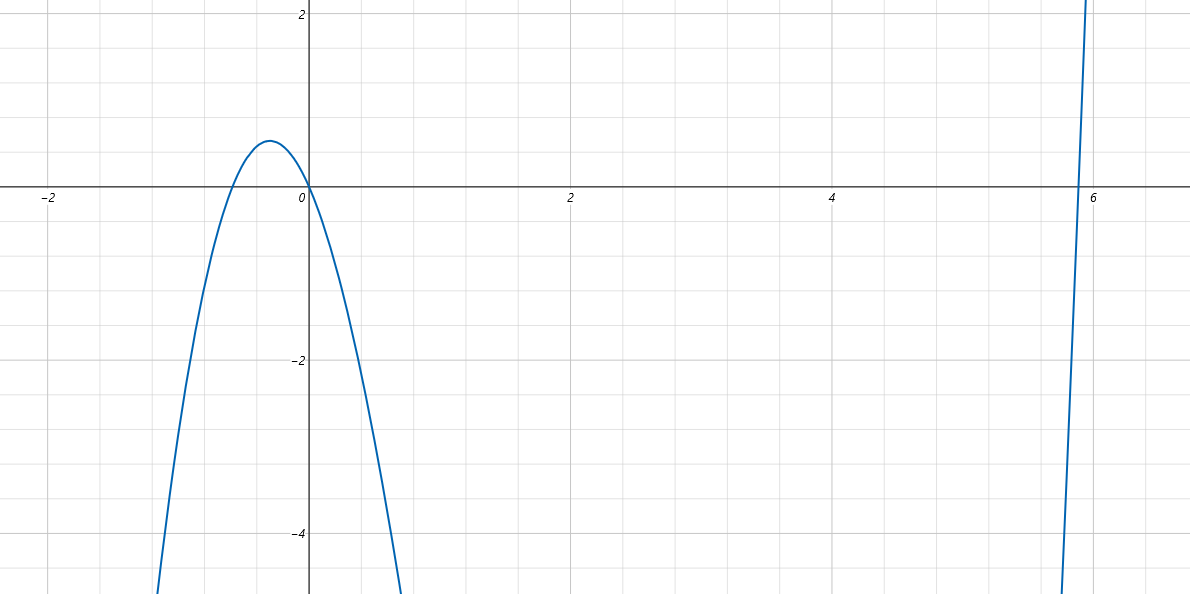


Рисунок 1 График заданной функции (увеличенный масштаб)

Интервалы, на которых существует ровно 1 корень:

[-2, -0.1];

[-0.1, 1];

[5, 6]

ЗУ решается аналитически:

Найденные аналитечески корни принадлежат выделенным интервалам

**Метод деления отрезка пополам**

Метод половинного деления (или бисекции) основан на **теореме Больцано–Коши**: если функция f(x) непрерывна на отрезке [a,b] и значения функции на **концах** отрезка имеют разные знаки (f(a) \* f(b) < 0), то на этом интервале существует хотя бы один корень уравнения f(x) = 0.

Суть метода заключается в последовательном делении отрезка пополам и выборе той половины, на концах которой функция имеет разные знаки.  
Таким образом, на каждой итерации интервал уменьшается вдвое, а его середина приближается к корню.

*Листинг 1 – Реализация метода половинного деления на языке Python*

def bisection\_Method(func, a : float, b : float, eps : float) -> tuple:

    """

    Метод половинного деления для нахождения корня уравнения f(x) = 0.

    Args:

        func (function): Исследуемая функция

        a (float): граница интервала [a, b], на котором f(a)\*f(b) < 0

        b (float): граница интервала [a, b], на котором f(a)\*f(b) < 0

        eps (float): Требуемая точность

    """

    if func(a) \* f(b) > 0: raise ValueError("Функция на концах интервала должна иметь разные знаки")

    if a > b: raise ValueError("Неправильно определены границы интервала")

    iterations = 0

    while (b - a) / 2 > eps:

        iterations += 1

        mid = (a + b) / 2

        if f(mid) == 0: return (mid, iterations) # нашли точный корень

        elif f(a) \* f(mid) < 0: b = mid

        else: a = mid

    return ((a + b) / 2, iterations)

if \_\_name\_\_ == "\_\_main\_\_":

    f = lambda x: x\*\*3 - 5.3\*x\*\*2 - 3.45\*x

    eps = [0.01, 0.001, 0.0001]

    for e in eps:

        print(f"EPS: {e:.4f}")

        root, iters = bisection\_Method(f, a=-2, b=-0.1, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        root, iters = bisection\_Method(f, a=-0.1, b=1, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        root, iters = bisection\_Method(f, a=5, b=6, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        print(" ")

**Метод Ньютона**

Метод Ньютона — это итерационный способ нахождения корня уравнения. Идея метода: приближать функцию её касательной в текущей точке и находить пересечение касательной с осью x.

Начальное приближен будет браться исходя из графика и интервалов

*Листинг 2 – Реализация метода Ньютона на языке Python*

def newton\_Method(func, df, x0 : float, eps : float, max\_iter=1000) -> tuple:

    """

    Метод Ньютона для нахождения корня f(x) = 0.

    Args:

        func (function): функция

        df (function): производная функции

        x0 (float): начальное приближение

        eps (float): требуемая точность

        max\_iter (int, optional): ограничение на число итераций

    Returns:

        tuple: корень и количество итерация

    """

    iterations = 0

    x = x0

    while iterations < max\_iter:

        iterations += 1

        fx = f(x)

        dfx = df(x)

        if dfx == 0: raise ValueError("Производная равна нулю, метод Ньютона не применим")

        x\_new  = x - fx / dfx

        if abs(x\_new - x) < eps: return (x\_new, iterations)

        x = x\_new

    raise RuntimeError("Метод Ньютона не сошёлся за указанное число итераций")

if \_\_name\_\_ == "\_\_main\_\_":

    # 2 method

    f = lambda x: x\*\*3 - 5.3\*x\*\*2 - 3.45\*x

    df = lambda x: 3\*x\*\*2 - 10.6\*x - 3.45

    eps = [0.01, 0.001, 0.0001]

    for e in eps:

        print(f"EPS: {e:.4f}")

        root, iters = newton\_Method(func=f, df=df, x0=-1.0, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        root, iters = newton\_Method(func=f, df=df, x0=1.0, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        root, iters = newton\_Method(func=f, df=df, x0=6.0, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        print(" ")

**Метод простых итераций**

Метод простых итераций сводит уравнение f(x) = 0 к эквивалентному виду:

Тогда решение ищется последовательностью:

При правильном выборе функции метод сходится к корню.  
Критерий сходимости: в окрестности корня.

*Листинг 3 – Реализация метода простых итераций на языке Python*

def simple\_Iteration\_Method(phi, x0 : float, eps : float, max\_iter=1000) -> tuple:

    """

    Метод простых итераций для решения уравнения x = phi(x).

    Args:

        phi (function): функция итерации

        x0 (float): начальное приближение

        eps (float): требуемая точность

        max\_iter (int, optional): ограничение на число итераций. Defaults to 1000.

    Returns:

        tuple: корень и количество итерация

    """

    iterations = 0

    x = x0

    while iterations < max\_iter:

        iterations += 1

        x\_new = phi(x)

        if abs(x\_new - x) < eps: return (x\_new, iterations)

        x = x\_new

    raise RuntimeError("Метод простых итераций не сошёлся за указанное число итераций")

if \_\_name\_\_ == "\_\_main\_\_":

    # 3 method

    phi\_0 = lambda x: x - 0.001 \* (x\*\*3 - 5.3\*x\*\*2 - 3.45\*x)

    phi\_1 = lambda x: ((x\*\*2 - 3.45\*x) / 5.3)

    phi\_2 = lambda x: (5.3\*x\*\*2 + 3.45\*x)\*\*(1/3)

    eps = [0.01, 0.001, 0.0001]

    for e in eps:

        print(f"EPS: {e:.4f}")

        root, iters = simple\_Iteration\_Method(phi=phi\_0, x0=-0.8, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        root, iters = simple\_Iteration\_Method(phi=phi\_1, x0=1.0, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        root, iters = simple\_Iteration\_Method(phi=phi\_2, x0=6.0, eps=e)

        print(f" Value: {root:.6f}, Iters: {iters}")

        print(" ")

Метод простых итераций часто вызывает трудности именно на этапе выбора функции φ(x). Уравнение можно переписать по-разному и проверить сходимость их по производным. Например:

Согласно критерию сходимости формы φ(x), равные и сходятся в окресности 2го и третьего интервалов.

Для реализации функции φ(x) на 1м интревале выбран вид:

Методом подбора был выбран k равный 0.001.

|  |  |  |  |
| --- | --- | --- | --- |
| Методо решения | Точность | Значение корня | Количество итераций |
| Деления отрезка пополам | 0.01 | -0.582422  -0.005469  5.882812 | 7  6  6 |
| Деления отрезка пополам | 0.001 | -0.585205  0.000439  5.885742 | 10  10  9 |
| Деления отрезка пополам | 0.0001 | -0.586075  -0.000031  5.886169 | 14  13  13 |
| Ньютона | 0.01 | -0.586126  0.000045  5.886129 | 4  4  2 |
| Ньютона | 0.001 | -0.586126  0.000000  5.886124 | 4  5  3 |
| Ньютона | 0.0001 | -0.586124  0.000000  5.886124 | 5  5  3 |
| Простых итераций | 0.01 | -0.798856  -0.002815  5.897654 | 1  13  5 |
| Простых итераций | 0.001 | -0.777549  0.000329  5.887299 | 21  18  10 |
| Простых итераций | 0.0001 | -0.611154  -0.000038  5.886244 | 487  23  15 |

# Вывод

Метод половинного деления гарантирует сходимость, но обычно работает медленнее, чем метод Ньютона или метод простых итераций. Метод Ньютона сходится очень быстро (обычно квадратично), если начальное приближение выбрано удачно и функция достаточно «хорошая» (гладкая и вблизи корня). Метод простых итераций работает медленнее, чем Ньютона, но проще по вычислениям, если удалось определить спомогательные функции.