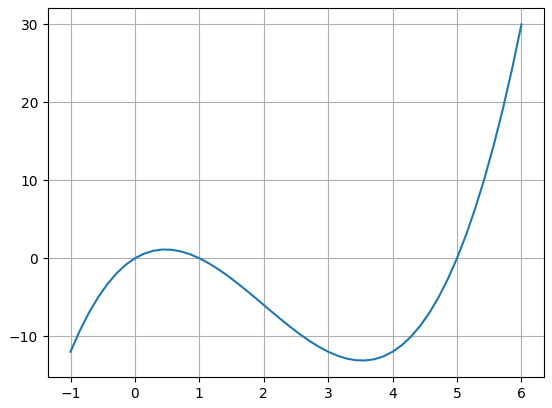
**Лабораторная работа по методам вычислений №1**

Выполнил:   
Верещагин Н.Е группа 7.1

**Тема:**   
*“Численные методы решения нелинейных уравнений”*

**Дано:**  
на интервале [-1 , 6]

**Цель работы:**  
Изучить численные методы решения нелинейных уравнений и ознакомиться с синтаксисом и бызовыми операторами языка Python  
  
**Ход работы:**  
1. Сроим график исследуемой функции f(x)  


2. Определяем интревалы, содержащие корни уравнения f(x) = 0

интервалы - [-0.9, 0.5]; [0.5, 3.5]; [3.5, 6].

корни - [0, 1, 5].

3. Написал код, реализующий численные методы поиска корня уравнения f(x) = 0

**Метод бисекции**

Этот метод используется для нахождения корня уравнения в заданном интервале с определённой точностью. Он прост в реализации и всегда даёт результат, если на концах отрезка значения функции имеют разные знаки. Однако он требует больше итераций по сравнению с другими методами.

**Метод хорд**

Реализован в соответствии с методическими рекомендациями. Позволяет находить корень быстрее, чем метод бисекции, так как использует приближения с помощью секущих (хорд). Подходит для случаев, когда вычисление производной затруднительно.

**Метод Ньютона**

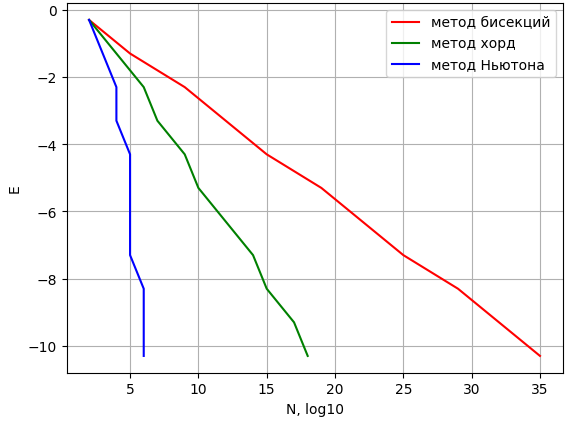
Этот метод требует дополнительного аргумента — производной функции, а также начального приближения. Работает быстрее остальных, так как использует касательные для поиска корня, но может не сойтись, если функция не удовлетворяет необходимым условиям.

**Построение графиков**

Для наглядного сравнения построены графики **относительной погрешности** каждого метода в зависимости от количества итераций. Использованы массивы для хранения результатов и matplotlib для визуализации. На каждой итерации точность уменьшается, что позволяет проследить динамику сходимости.

4**. Построил на одной координатной плоскости графики относительной погрешности каждого метода в зависимости от количества итераций**

Создал массивы для хранения количества итераций каждого метода, а также список значений эпсилона. В процессе работы цикла значение эпсилона постепенно уменьшается. Для построения графиков использовал matplotlib. Изначально реализовал расчёт по-другому, ориентируясь на абсолютную погрешность на каждой итерации, но позже выяснил, что это неверно, и переделал. В результате для второго промежутка получился следующий итог.



**5. Вывод**

Метод Ньютона показывает наилучшую эффективность, если его применение возможно. Благодаря использованию производной он работает быстрее остальных и обладает квадратичной сходимостью O(N2)O(N^2)O(N2). Уже на пятой итерации даёт точный результат.

Метод хорд является хорошей альтернативой в случаях, когда вычисление производной затруднено. Имеет линейную сходимость O(N)O(N)O(N), но при этом работает быстрее метода бисекции.

Метод бисекции удобен, когда другие способы недоступны, однако требует больше всего итераций. Также обладает линейной сходимостью O(N)O(N)O(N), но медленнее в сравнении с методом хорд.

**6. Используемые библиотеки и методы**

**import matplotlib.pyplot as plt** – для построения графиков.

* plt.plot() – строит график.
* plt.xlabel(), plt.ylabel() – подписывает оси.
* plt.legend() – добавляет легенду с названиями методов.
* plt.grid(True) – включает сетку на графике.

**import numpy as np** – для работы с массивами и числовыми вычислениями.

* np.linspace(-1, 6, 50) – создаёт массив значений для построения графика функции.
* np.sign(f(x)) – определяет знак функции.

**import math** – для работы с математическими функциями.

* math.log10(e) – используется для логарифмического представления точности.