Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа № 5.А

Вариант - 5

Выполнили:
Азатжонова М. А.
Маркозубова А. К.

Санкт-Петербург,
2024

СОДЕРЖАНИЕ

Стр.

Задание

В индивидуальных заданиях приводятся функции f(x) = 0

В задании 5:

А) Решить нелинейное уравнение вида f(x) = 0. Использовать нужно 2 способа приближенного решения на выбор (бисекция, метод Ньютона, метод простых итераций). Корень либо корни уравнения нужно локализовать самостоятельно

Ход решения

Рассмотрим функцию f(x) = $\sin(|x| + 2.3^x)$ на интервале [-1, 2]. Для локализации корней построим график функции:

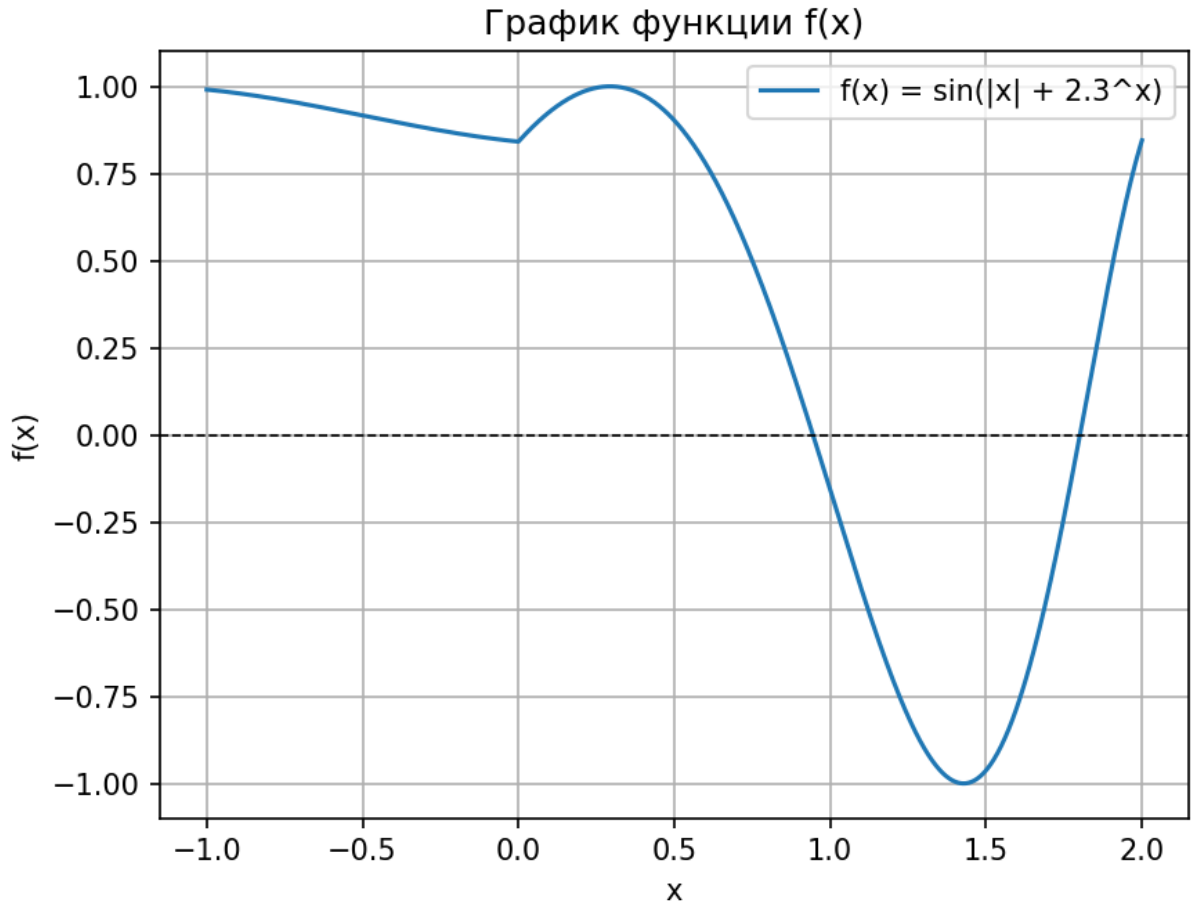

Рисунок 1 — Функция sin $(|x| + 2.3^x)$

На графике видно, что функция пересекает ось X дважды и на интервале [-1, 2] существуют корни. Локализуем корни для дальнейшего применения методов: первый корень находится на интервале [0.5, 1]; второй корень находится на интервале [1.5, 2].

Метод бисекции

Метод бисекции применяется для нахождения корня нелинейного уравнения $f(x) = 0$ на интервале [a,b], если значения функции на концах интервала имеют противоположные знаки, то есть $f(a) \cdot f(b) < 0$. Этот метод основан

на делении интервала пополам и выборе подынтервала, в котором функция меняет знак.

На каждом шаге вычисляется середина текущего интервала: $c = \frac{a+b}{2}$. Затем проверяется значение функции в точке c. Если $f(c) = 0$, то найден точный корень, и процесс завершается. В противном случае выбирается подынтервал, где функция меняет знак: если $f(a) \cdot f(c) < 0$, то новым интервалом становится [a,c]; если $f(b) \cdot f(c) < 0$, то новым интервалом становится [c,b].

Процесс повторяется до тех пор, пока длина интервала $\frac{b-a}{2}$ не станет меньше заданной точности ϵ.

Применив данный метод для нахождения корней нашего уравнения, получаем:

```
Корень 1 (метод бисекции): 0.9448518753051758
Корень 2 (метод бисекции): 1.801039695739746
```

Рисунок 2 — Корни методом бисекции

Метод Ньютона

Основная идея метода Ньютона заключается в том, чтобы использовать линейное приближение функции в окрестности текущего приближения корня. На каждом шаге строится касательная к графику функции в текущей точке, и следующее приближение определяется как точка пересечения этой касательной с осью X. Формула для вычисления нового приближения имеет вид: $x_{n+1} = x_n - \frac{f(x_n)}{f`(x_n)}$.

Алгоритм метода Ньютона включает следующие шаги:

1. Задать начальное приближение x_0

2. Для каждого шага вычислить значение функции $f(x_n)$ и её производной $f`(x_n)$

3. Найти следующее приближение по формуле

4. Проверить условие остановки: если $|f(x_{n+1})| < \epsilon$ или $|x_{n+1} - x_n| < \epsilon$, то процесс завершается.

Применив данный метод для нахождения корней нашего уравнения, получаем:

```
Корень 1 (метод Ньютона): 0.9448511383686304
Корень 2 (метод Ньютона): 1.8010391934285375
```

Рисунок 3 — Корни методом Ньютона

Выводы

В ходе лабораторной работы мы исследовали решение нелинейных уравнений методами бисекций или Ньютона. Оба метода сошлись к довольно близким корням.

Метод бисекции менее чувствителен к выбору начальных условий и обеспечивает гарантированную сходимость на интервале, где функция меняет знак. Ошибка метода бисекции оценивается как $\epsilon = \frac{a-b}{2}$.

Метод Ньютона обладает квадратичной сходимостью, что означает, что вблизи корня скорость его сходимости значительно выше по сравнению с методом бисекции. Однако метод требует выполнения нескольких условий: производная $f`(x_n)$ должна существовать и не равняться нулю в процессе вычислений; а начальное приближение должно быть достаточно близким к истинному корню, чтобы метод сходился. Ошибка метода Ньютона оценивается как $\epsilon = |f(x_n)|$ или как $\epsilon = |x_{n+1} - x_n|$.

Приложение

Ниже представлен полный код программы на языке программирования Python, который был написан для выполнения данной лабораторной работы:

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return np.sin(np.abs(x) + 2.3**x)

x = np.linspace(-1, 2, 1000)
y = f(x)

plt.plot(x, y, label="f(x) = sin(|x| + 2.3^x)")
plt.axhline(0, color='black', linewidth=0.8, linestyle='--')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('График функции f(x)')
plt.legend()
plt.grid()
plt.show()

def bisection_method(func, a, b, tol=1e-6):
    if func(a) * func(b) >= 0:
        raise ValueError("На концах интервала значения функции должны иметь разный знак.")

    while (b - a) / 2 > tol:
        c = (a + b) / 2
        if func(c) == 0:
            return c  # Точный корень
        elif func(a) * func(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2

root1 = bisection_method(f, 0.5, 1)
root2 = bisection_method(f, 1.5, 2)
print("Корень 1 (метод бисекции):", root1)
print("Корень 2 (метод бисекции):", root2)
```

```python
def df(x):
    sign = 1 if x >= 0 else -1
    return np.cos(np.abs(x) + 2.3**x) * (sign + 2.3**x * np.log(2.3))

def newton_method(func, d_func, x0, tol=1e-6, max_iter=100):
    x = x0
    for _ in range(max_iter):
        fx = func(x)
        dfx = d_func(x)
        if dfx == 0:
            raise ValueError("Производная равна нулю. Метод не применим.")
        x_new = x - fx / dfx
        if abs(fx) < tol or abs(x_new - x) < tol:
            return x_new
        x = x_new
    raise ValueError(f"Метод Ньютона не сошёлся за {max_iter} итераций.")


root1_newton = newton_method(f, df, 1)
root2_newton = newton_method(f, df, 2)
print("Корень 1 (метод Ньютона):", root1_newton)
print("Корень 2 (метод Ньютона):", root2_newton)
```