Задание 3

Нодири Хисравхон

Рассмотрим функцию $f(x) = x^4 + x^2 + x + 1$ на интервале $[-1, 0]$ с точностью $\epsilon = 0.003$.

1 Метод половинного деления

1.1 Шаги

1. Начальные границы $a = -1$, $b = 0$, $f(a) = 1$, $f(b) = 3$.
2. Шаг 1: $c = \frac{-1+0}{2} = -0.5$, $f(c) = 0.3125$. Так как $f(a) \cdot f(c) < 0$, новый интервал $[-1, -0.5]$.
3. Шаг 2: $c = \frac{-1-0.5}{2} = -0.75$, $f(c) = 0.78515625$. Новый интервал $[-1, -0.75]$.
4. Шаг 3: $c = \frac{-1-0.75}{2} = -0.875$, $f(c) = 0.3330078125$. Новый интервал $[-1, -0.875]$.
5. Шаг 4: $c = \frac{-1-0.875}{2} = -0.9375$, $f(c) = 0.152587890625$. Новый интервал $[-1, -0.9375]$.
6. Шаг 5: $c = \frac{-1-0.9375}{2} = -0.96875$, $f(c) = 0.034423828125$. Новый интервал $[-1, -0.96875]$.

После пяти шагов получен интервал, содержащий корень функции с учетом заданной точности $\epsilon = 0.003$.

1.2 Код на Python для метода половинного деления

```
def bisection_method(f, a, b, eps):
    while (b - a) / 2 > eps:
        c = (a + b) / 2
        if f(c) == 0:
            break
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2

f = lambda x: x**4 + x**2 + x + 1
a, b, eps = -1, 0, 0.003

root = bisection_method(f, a, b, eps)
print(f"Корень, найденный методом половинного деления: {root}")
```

2 Метод золотого сечения

Рассмотрим функцию $f(x) = x^4 + x^2 + x + 1$ на интервале $[-1, 0]$ с точностью $\epsilon = 0.003$.

2.1 Вычисления

Используем константу золотого сечения $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

1. Начальные границы $a = -1$, $b = 0$.
2. **Шаг 1:**
   - $c = b - \frac{b-a}{\varphi} \approx -0.618$
   - $d = a + \frac{b-a}{\varphi} \approx -0.382$
   - Выбор интервала: $[-1, -0.382]$
3. **Шаг 2:**
   - Пересчитываем c и d для $[-1, -0.382]$.
   - $c \approx -0.764$, $d \approx -0.618$
   - Новый интервал: $[-0.764, -0.382]$

4. **Шаг 3:**
   - Для $[-0.764, -0.382]$, $c \approx -0.618$, $d \approx -0.528$
   - Интервал: $[-0.618, -0.382]$
5. **Шаг 4:**
   - Для $[-0.618, -0.382]$, $c \approx -0.528$, $d \approx -0.472$
   - Интервал: $[-0.528, -0.382]$
6. **Шаг 5:**
   - Для $[-0.528, -0.382]$, $c \approx -0.472$, $d \approx -0.438$
   - Конечный интервал: $[-0.472, -0.382]$

После пяти шагов интервал, содержащий корень функции, сужен до $[-0.472, -0.382]$ с учетом заданной точности $\epsilon = 0.003$.

2.2 Код

```
def golden_section_search(f, a, b, eps):
    phi = (1 + 5**0.5) / 2

    while abs(b - a) > eps:
        c = b - (b - a) / phi
        d = a + (b - a) / phi
        if f(c) < f(d):
            b = d
        else:
            a = c
    return (a + b) / 2

f = lambda x: x**4 + x**2 + x + 1

a, b, eps = -1, 0, 0.003

minimum = golden_section_search(f, a, b, eps)
print(f"Корень, найденный методом золотого сечения {minimum}")
```

3 Метод хорд

Рассмотрим функцию $f(x) = x^4 + x^2 + x + 1$ на интервале $[-1, 0]$ с точностью $\epsilon = 0.003$.

3.1 Вычисления

1. Начальные точки $x_0 = -1$ и $x_1 = 0$.
2. **Шаг 1:**
$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)} = 0 - (1)\frac{0 - (-1)}{1 - 3} = -0.5$$
3. **Шаг 2:**
$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)} = -0.5 - (0.3125)\frac{-0.5 - 0}{0.3125 - 1} \approx -0.8333$$
4. **Шаг 3:**
$$x_4 = x_3 - f(x_3)\frac{x_3 - x_2}{f(x_3) - f(x_2)} \approx -0.8333 - (0.6357421875)\frac{-0.8333 - (-0.5)}{0.6357421875 - 0.3125} \approx -0.9444$$
5. **Шаг 4:**
$$x_5 = x_4 - f(x_4)\frac{x_4 - x_3}{f(x_4) - f(x_3)} \approx -0.9444 - (0.108184814453125)\frac{-0.9444 - (-0.8333)}{0.108184814453125 - 0.6357421875} \approx -0.9722$$
6. **Шаг 5:**
$$x_6 = x_5 - f(x_5)\frac{x_5 - x_4}{f(x_5) - f(x_4)} \approx -0.9722 - (0.02587890625)\frac{-0.9722 - (-0.9444)}{0.02587890625 - 0.108184814453125} \approx -0.9861$$

После пяти шагов находим приближенное значение корня функции, удовлетворяющее заданной точности $\epsilon = 0.003$.

3.2 Код

```
def secant_method(f, x0, x1, eps):
    while abs(x1 - x0) > eps:
        x_temp = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        x0, x1 = x1, x_temp
    return x1

f = lambda x: x**4 + x**2 + x + 1

x0, x1, eps = -1, 0, 0.003

root = secant_method(f, x0, x1, eps)
print(f"Корень, найденный методом хорда: {root}")
```

4 Метод Ньютона

Производная функции: $f'(x) = 4x^3 + 2x + 1$.

Шаг 1:

$$x_0 = -0.5$$

$$f(x_0) = (-0.5)^4 + (-0.5)^2 + (-0.5) = 0.0625 + 0.25 - 0.5 = -0.1875$$

$$f'(x_0) = 4(-0.5)^3 + 2(-0.5) + 1 = -0.5 - 1 + 1 = -0.5$$

$$x_1 = -0.5 - \frac{-0.1875}{-0.5} = -0.875$$

Шаг 2:

$$x_1 = -0.875$$

$$f(x_1) = (-0.875)^4 + (-0.875)^2 + (-0.875) = 0.476806640625$$

$$f'(x_1) = 4(-0.875)^3 + 2(-0.875) + 1 = -3.4296875$$

$$x_2 = -0.875 - \frac{0.476806640625}{-3.4296875} \approx -0.7359766514806378$$

Шаг 3:

$$x_2 = -0.7359766514806378$$

$$f(x_2) = (-0.7359766514806378)^4 + (-0.7359766514806378)^2 + (-0.7359766514806378) = 0.09908230310996258$$

$$f'(x_2) = 4(-0.7359766514806378)^3 + 2(-0.7359766514806378) + 1 = -2.0665545581814864$$

$$x_3 = -0.7359766514806378 - \frac{0.09908230310996258}{-2.0665545581814864} \approx -0.6880310007269735$$

Шаг 4:

$$x_3 = -0.6880310007269735$$

$$f(x_3) = (-0.6880310007269735)^4 + (-0.6880310007269735)^2 + (-0.6880310007269735) = 0.009450585170213355$$

$$f'(x_3) = 4(-0.6880310007269735)^3 + 2(-0.6880310007269735) + 1 = -1.6788807854857568$$

$$x_4 = -0.6880310007269735 - \frac{0.009450585170213355}{-1.6788807854857568} \approx -0.6824019023109839$$

Шаг 5:

$$x_4 = -0.6824019023109839$$

$$f(x_4) = (-0.6824019023109839)^4 + (-0.6824019023109839)^2 + (-0.6824019023109839) = 0.0001211973678438838$$

$$f'(x_4) = 4(-0.6824019023109839)^3 + 2(-0.6824019023109839) + 1 = -1.635906611731993$$

$$x_5 = -0.6824019023109839 - \frac{0.0001211973678438838}{-1.635906611731993} \approx -0.6823278165612383$$

Код:

```
def f(x):
    return x**4 + x**2 + x

def df(x):
    return 4*x**3 + 2*x + 1

def newton_method(x0, iterations):
    x = x0
    for i in range(iterations):
        fx = f(x)
        dfx = df(x)
        x = x - fx / dfx
        print(f"Step {i+1}:")
        print(f"f(x{i+1}) = {fx}")
        print(f"f'(x{i+1}) = {dfx}")
        print(f"x{i+1} = {x}\n")
    return x

x0 = -0.5
iterations = 5
final_x = newton_method(x0, iterations)

print(f"The approximate root after {iterations} iterations is {final_x}")
```