Методы вычислений

Лабораторная работа 1

Численные методы решения нелинейных уравнений

Выполнила Николаева Ксения, 9 группа

Постановка задачи

Дано нелинейное уравнение:

$$f(x) = xe^x + x^2 - 1 = 0, \quad x < 0$$

Необходимо:

- Графически отделить корень уравнения $f(x) = 0$.
- Сузить отрезок отделенного корня методом дихотомии с точностью $varepsilon = 10^{-2}$.
- Найти решение уравнения $f(x) = 0$ с точностью $varepsilon = 10^{-7}$ методами:
  - Методом Ньютона с постоянной производной,
  - Методом Ньютона,
  - Методом секущих.
- Провести сравнительный анализ полученных результатов.

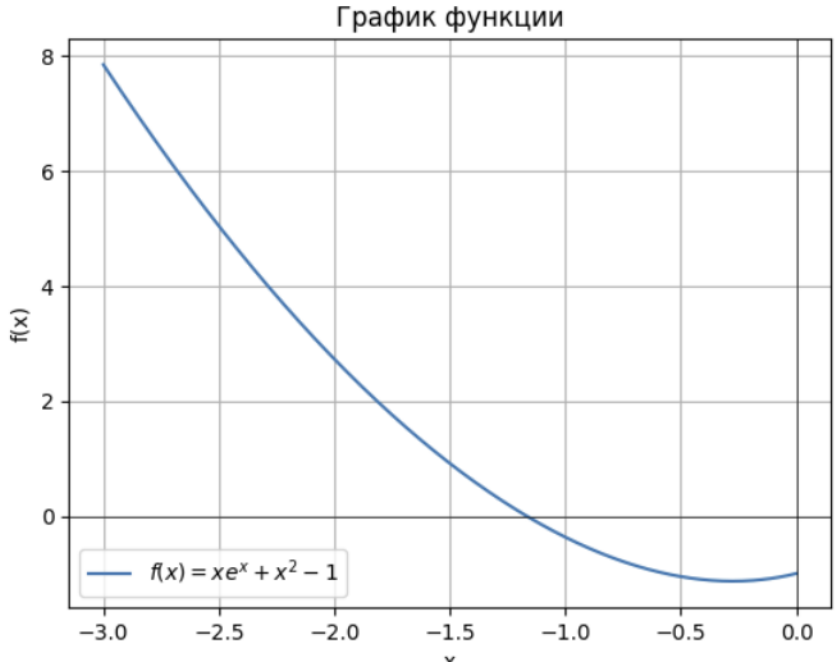

Рис. 1: График функции, отделенный отрезок [-2, -1]

Результаты работы

Метод дихотомии

Метод дихотомии используется для нахождения корня функции на отрезке $[a, b]$. Он основан на повторном делении отрезка пополам. Если функция $f(x)$ имеет разные знаки на концах отрезка, то на одном из подотрезков будет корень. Алгоритм продолжает делить отрезок до тех пор, пока длина отрезка не станет меньше заданного порога ϵ.

```
def dichotomy_method(a, b, epsilon=1e-2):
    k = 0
    while (b - a) / 2 > epsilon:
        c = (a + b) / 2
        if f(c) == 0:
            return c, results
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        k += 1
    return (a + b) / 2, results
```

Корень по методу дихотомии: $x \approx -1.1640625$

Метод Ньютона с постоянной производной

Метод Ньютона с постоянной производной используется для нахождения корня функции, при этом предполагается, что производная функции $f'(x)$ постоянна. Алгоритм итеративно улучшает приближение корня, используя формулу $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.

```
    def newton_method_constant_derivative(x0, epsilon=1e-7):
    k = 0
```

k	a_k	b_k	$f(a_k)$	$f(b_k)$	$b_k - a_k$
0	-2.000000	-1.00000	2.729329	-0.367879	1.000000
1	-1.500000	-1.00000	0.915305	-0.367879	0.500000
2	-1.250000	-1.00000	0.204369	-0.367879	0.250000
3	-1.250000	-1.12500	0.204369	-0.099609	0.125000
4	-1.187500	-1.12500	0.047989	-0.099609	0.062500
5	-1.187500	-1.15625	0.047989	-0.026916	0.031250
6	-1.171875	-1.15625	0.010261	-0.026916	0.015625

Таблица 1: Результаты метода дихотомии

```
x = x0
while abs(f(x)) > epsilon:
    x = x - f(x) / f_prime(x0)
    k += 1
return x
```

Метод Ньютона

Метод Ньютона используется для нахождения корня функции, используя ее производную. Начальное приближение x_0 подставляется в формулу итерации, и на каждом шаге улучшение приближения вычисляется с учетом текущей производной функции.

```
def newton_method(x0, epsilon=1e-7):
k = 0
x = x0
while abs(f(x)) > epsilon:
    x = x - f(x) / f_prime(x)
    k += 1
return x
```

Метод секущих

Метод секущих является итерационным методом нахождения корней функции. Он не требует вычисления производной, вместо этого используется секущая линия, проходящая через две последние точки. На каждом шаге новые значения для x_0 и x_1 обновляются в зависимости от значения функции в этих точках.

```
def secant_method(x0, x1, epsilon=1e-7):
k = 0
while abs(f(x1)) > epsilon:
    x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
    x0, x1 = x1, x2
    k += 1
return x1
```

k	Метод Ньютона с постоянной производной	Метод Ньютона	Метод секущих
0	-1.167591	-1.167591	-1.118778
1	-1.167586	-1.167586	-1.171989
2	—	—	-1.167481
3	—	—	-1.167585
4	—	—	-1.167586
$\lvert x_k - x_{k-1}\rvert$	1.407421×10^{-5}	1.407421×10^{-5}	1.138174×10^{-1}
$\lvert x_k - x_{k-1}\rvert$	4.713562×10^{-8}	3.925593×10^{-11}	1.053443×10^{-2}
—	—	—	2.489199×10^{-4}
—	—	—	5.173854×10^{-7}
—	—	—	2.552580×10^{-11}

Таблица 2: Сводные данные по результатам работы методов

Выводы

• Метод дихотомии потребовал больше всего итераций, но гарантированно сужал интервал.
• Метод Ньютона и его модификация с постоянной производной показали быстрый сход, но требуют знания производной.
• Метод секущих сошелся за 5 итераций, не требуя производной, что делает его эффективным.
• Все методы показали сход к корню $x \approx -1.1675855268$, что подтверждает их корректность и сходимость.

Листинг программы

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

# Определяем функцию
def f(x):
    return x * np.exp(x) + x**2 - 1

# Построение графика
x_vals = np.linspace(-3, 0, 400)
y_vals = f(x_vals)

plt.plot(x_vals, y_vals, label=r'$f(x) = x e^x + x^2 - 1$')
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.title('График функции')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.show()


# Метод дихотомии
def dichotomy_method(a, b, epsilon=1e-2):
    k = 0
    results = []
    while (b - a) / 2 > epsilon:
        c = (a + b) / 2
        results.append([k, a, b, f(a), f(b), b - a])
        if f(c) == 0:
            return c, results
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        k += 1
    results.append([k, a, b, f(a), f(b), b - a])
    return (a + b) / 2, results

# Отрезок отделенного корня
a, b = -2, -1  # Начальный интервал
root_dichotomy, dichotomy_results = dichotomy_method(a, b, epsilon=1e-2)
print(f"Корень по методу дихотомии: {root_dichotomy}")

# Производная функции
def f_prime(x):
```

```
    return np.exp(x) * (x + 1) + 2 * x

# Метод Ньютона с постоянной производной
def newton_method_constant_derivative(x0, epsilon=1e-7):
    k = 0
    x = x0
    results = []
    while abs(f(x)) > epsilon:
        x = x - f(x) / f_prime(x0)
        results.append([k, x, abs(f(x))])
        k += 1
    return x, results

# Корень по методу Ньютона с постоянной производной
root_newton_constant, newton_constant_results = newton_method_constant_derivative(root_d
print(f"Корень по методу Ньютона (с постоянной производной): {root_newton_constant}")

# Метод Ньютона
def newton_method(x0, epsilon=1e-7):
    k = 0
    x = x0
    results = []
    while abs(f(x)) > epsilon:
        x = x - f(x) / f_prime(x)
        results.append([k, x, abs(f(x))])
        k += 1
    return x, results

# Корень по методу Ньютона
root_newton, newton_results = newton_method(root_dichotomy, epsilon=1e-7)
print(f"Корень по методу Ньютона: {root_newton}")

# Метод секущих
def secant_method(x0, x1, epsilon=1e-7):
    k = 0
    results = []
    while abs(f(x1)) > epsilon:
        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        results.append([k, x2, abs(f(x2))])
        x0, x1 = x1, x2
        k += 1
    return x1, results
```