Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Лабораторная работа №1
по дисциплине “Математический анализ и основы вычислений”

Семестр I

Выполнили: студенты
Бабич Александр Петрович
гр. J3111
ИСУ 412882
Кулинич Павел Васильевич
гр. J3111
ИСУ 466420

Отчет сдан: 13.01.2025

Санкт-Петербург
2025

Содержание

1 Введение

Оптимизация и решение уравнений являются важными задачами, которые находят широкое применение в различных областях науки и техники. В данной работе рассматриваются:

- Метод золотого сечения для поиска минимума унимодальной функции.
- Метод Брента для нахождения корня уравнения.

Основной целью работы является реализация алгоритмов, их тестирование и анализ полученных результатов.

2 Теоретическая часть

2.1 Определение унимодальной функции

Функция $f(x)$ называется унимодальной на отрезке $[a, b]$, если она непрерывна на $[a, b]$ и существуют числа α, β ($a \le \alpha \le \beta \le b$), такие что:

1. $f(x)$ строго убывает на $[a, \alpha]$.
2. $f(x)$ строго возрастает на $[\beta, b]$.
3. $f(x)$ достигает минимума на $[\alpha, \beta]$.

2.2 Метод золотого сечения

Метод золотого сечения относится к последовательным стратегиям оптимизации. Алгоритм использует две точки, делящие текущий интервал неопределённости в пропорции золотого сечения:

$$\varphi = \frac{3 - \sqrt{5}}{2}.$$

Алгоритм завершается, когда длина интервала становится меньше заданной точности ε.

2.3 Метод Брента

Метод Брента комбинирует подходы бисекции и интерполяции для нахождения корня уравнения $f(x) = 0$. Он гарантирует сходимость при условии непрерывности функции на заданном интервале.

3 Аналитический этап

3.1 Метод золотого сечения

1. **Функция средней сложности:** $f(x) = (x - 2)^2 + 1$. Эта функция унимодальна, с минимумом в точке $x = 2$.
2. **Промежуток для унимодальности:** Рассмотрим $[1, 3]$, где функция унимодальна.

3. **Точки локальных экстремумов:** Минимум в $x = 2$, тип экстремума — минимум.

4. **Наибольшее/наименьшее значения:** Наименьшее значение $f(2) = 1$, наибольшее $f(1) = f(3) = 2$.

5. **Выбранный алгоритм:** Метод золотого сечения, так как он оптимален для унимодальных функций.

6. **Сходимость:** Алгоритм сходится, так как длина интервала уменьшается пропорционально золотому сечению.

7. **Число итераций:** Для $\varepsilon = 0.01$ и $L_0 = 2$, $N \approx 30$.

8. **Исследование на различных промежутках:** На $[0, 4]$ метод успешно найдёт минимум, но на $[0, 1]$ результат будет некорректным.

3.2 Дополнительные задания для алгоритма золотого сечения

1. **Доказательство золотого сечения:** Рассмотрим интервал $[a, b]$. Алгоритм золотого сечения делит этот интервал на два подотрезка, пропорция между которыми равна $\varphi = \frac{3-\sqrt{5}}{2}$, где φ называется золотым сечением. На каждой итерации длина интервала уменьшается, и новые точки, которые вычисляются, делят этот интервал в золотое сечение, т.е. одна часть интервала будет меньше другой пропорционально φ. Таким образом, на каждом шаге сохраняется золотое сечение между длинами двух частей интервала, что позволяет достичь оптимального сокращения длины интервала.

2. **Погрешность метода золотого сечения:** Погрешность метода золотого сечения можно оценить через длину интервала L на каждой итерации. Если на k-й итерации длина интервала равна L_k, то погрешность можно выразить как:

$$\varepsilon_k = \frac{L_k}{2},$$

где L_k — длина интервала на k-й итерации. Погрешность стремится к нулю с каждым шагом, а скорость сходимости зависит от золотого сечения φ. В отличие от других методов поиска минимума, метод золотого сечения позволяет быстро уменьшать длину интервала и, следовательно, достигать высокой точности при относительно небольшом числе итераций.

3.3 Метод Брента

1. **Функции:** $f_1(x) = x^3 - x - 2$, $f_2(x) = \cos(x) - x$.

2. **Корни уравнений:** $f_1(x)$ имеет корень $x \approx 1.521$, $f_2(x)$ имеет корень $x \approx 0.739$.

3. **Критерий остановки:** $|f(x)| < \varepsilon$ или длина интервала меньше ε.

4. **Число итераций:** Для точности $\varepsilon = 10^{-6}$ требуется 20 итераций для $f_1(x)$ и 20 итераций для $f_2(x)$.

5. **Среднеквадратичное отклонение:** Используется функция библиотеки numpy -std.

6. **Модификация:** При отсутствии корня на промежутке алгоритм его не возвращает

7. **Проверка существования корней:**

   - Для функции $f_1(x) = x^3 - x - 2$ на интервале $[1, 2]$, функция изменяет знак на концах интервала (отрицательное значение в точке $x = 1$ и положительное в точке $x = 2$), что подтверждает существование корня на этом интервале.
   - Для функции $f_2(x) = \cos(x) - x$ на интервале $[0, 1]$, функция также изменяет знак на концах интервала, что подтверждает существование корня в этом интервале.

4 Графики функций

4.1 График функции $f(x) = (x-2)^2 + 1$

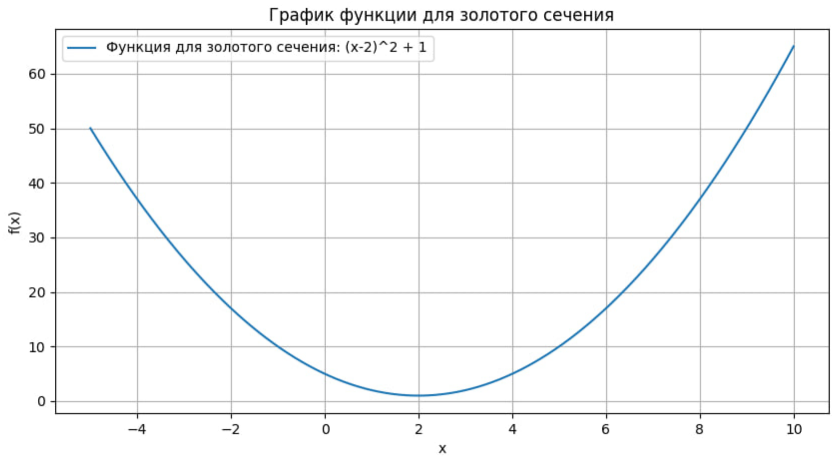

Рис. 1: График функции $f(x) = (x-2)^2 + 1$.

4.2 График функции $f_1(x) = x^3 - x - 2$

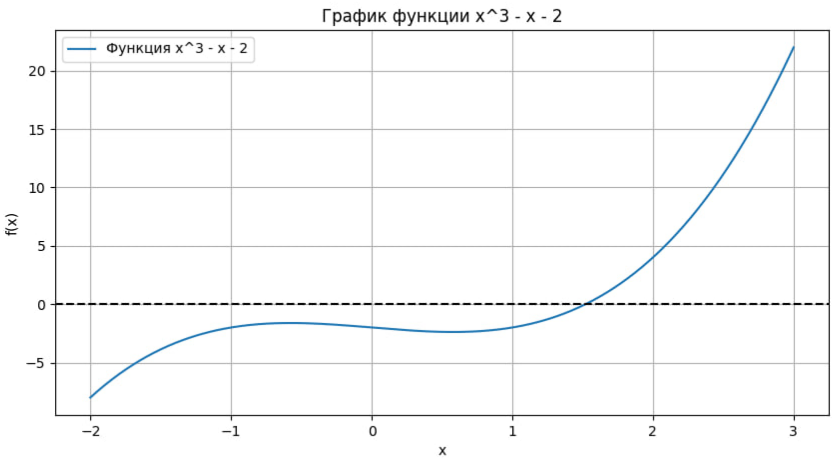

Рис. 2: График функции $f_1(x) = x^3 - x - 2$.

4.3 График функции $f_2(x) = \cos(x) - x$

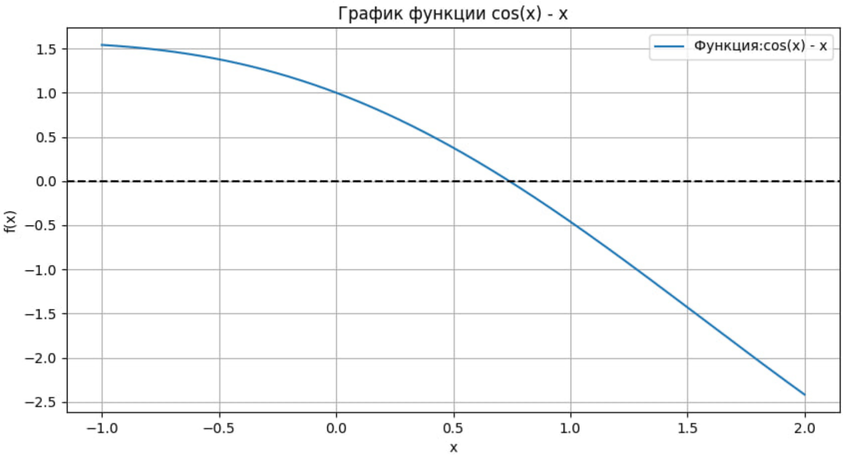

Рис. 3: График функции $f_2(x) = \cos(x) - x$.

5 Практическая реализация

5.1 Код на Python

Реализация методов была выполнена на языке Python. Основные функции:

- Алгоритм золотого сечения для поиска минимума унимодальной функции.
- Метод Брента для нахождения корней функций.

Код представлен в приложении А.

5.1.1 Метод золотого сечения

Минимум функции $f(x) = (x-2)^2 + 1$ на интервале $[-100, 99]$:

- Минимальное значение: $f_{min} = 1.00$.
- Точка минимума: $x_{min} = 2.00$.
- Число итераций: 21.
- Число вызовов функции: 23.

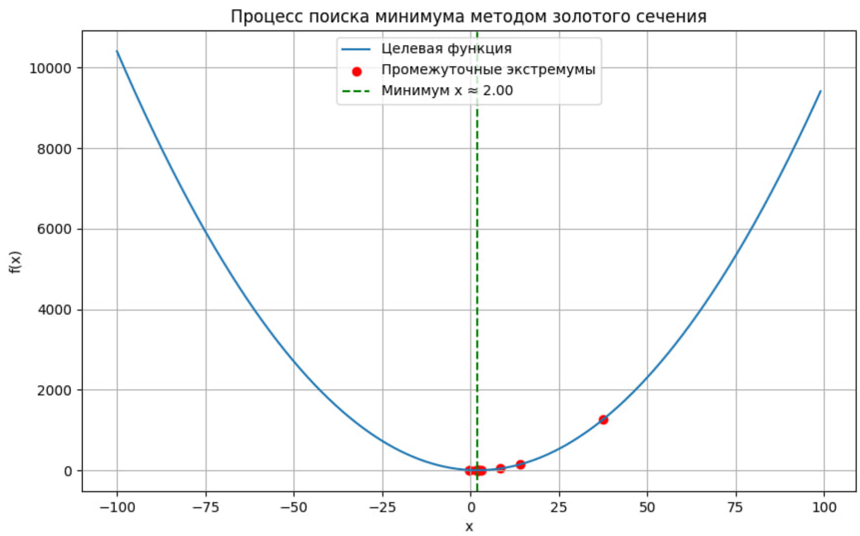

Рис. 4: График функции с промежуточными экстремумами.

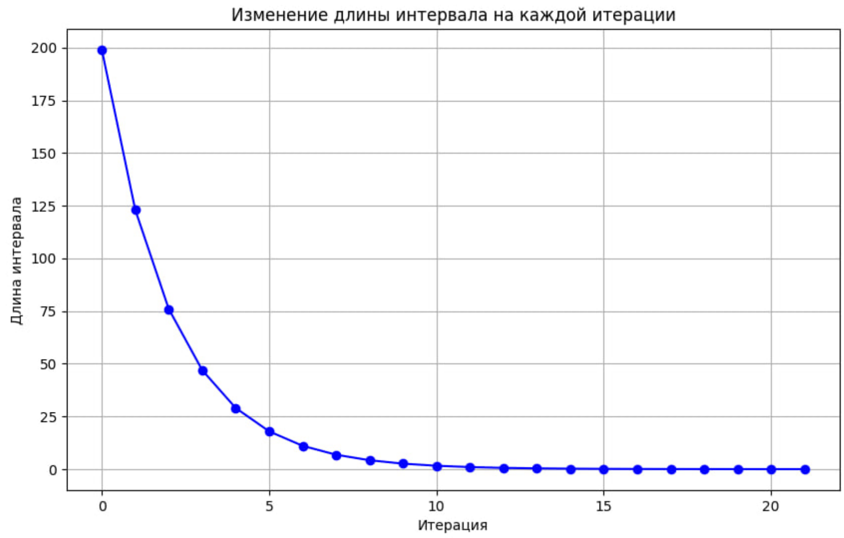

Рис. 5: График изменения длины промежутка неопределенности с возрастанием числа итераций.

```
--- Результаты работы алгоритма золотого сечения ---
Минимальное значение функции: 1.00000
Точка минимума: 2.00149
Количество итераций: 21
Количество вызовов функции: 23
История сужения интервала: [199, 122.98876376122908, 76.01123623877092, 46.97752752245816, 29.03370871631276, 17.9438188061454, 11.0
```

Рис. 6: Результат выполнения.

5.1.2 Метод Брента

Результаты нахождения корней:

- Для функции $f_1(x) = x^3 - x - 2$: $x \approx 1.521$, число итераций: 20, СКО: 1.2042889297e-07
- Для функции $f_2(x) = \cos(x) - x$: $x \approx 0.739$, число итераций: 20, СКО: 3.2116784898e-08

```
--- Результаты работы метода Брента ---
Корень функции x^3 - x - 2: 1.5213799477, Итерации: 20, СКО: 1.2042889297e-07
Корень функции cos(x) - x: 0.7390851974, Итерации: 20, СКО: 3.2116784898e-08
```

Рис. 7: Результат выполнения.

A Приложение А: Код программы

```
import math as m

import matplotlib.pyplot as plt
import numpy as np


# --- Алгоритм золотого сечения ---
def golden_ratio_algorithm(a: float, b: float, eps: float, func):
    """
    Реализует алгоритм золотого сечения для поиска минимума функции.

    Args:
        a (float): Левая граница интервала поиска.
        b (float): Правая граница интервала поиска.
        eps (float): Точность поиска минимума.
        func (function): Целевая функция, минимум которой нужно найти.

    Returns:
        tuple: Кортеж, содержащий:
            - f_min (float): Минимальное значение функции.
            - x_min (float): Точка минимума.
            - iterations (int): Количество итераций.
            - function_calls (int): Количество вызовов целевой функции.
            - delta_history (list): История изменения длины интервала.
            - x_history (list): История предполагаемых экстремумов.
    """
    k = 0
    function_calls = 0
    delta = b - a
    delta_history = [delta]
    x_history = []

    ratio_coeff = (3 - m.sqrt(5)) / 2
    y = a + ratio_coeff * (b - a)
    z = a + b - y

    f_y = func(y)
    f_z = func(z)
    function_calls += 2

    while abs(b - a) > eps:
        if f_y <= f_z:
            b = z
            z = y
            f_z = f_y
            y = a + b - y
            f_y = func(y)
        else:
```

```
            a = y
            y = z
            f_y = f_z
            z = a + b - z
            f_z = func(z)

        function_calls += 1
        delta = abs(b - a)
        delta_history.append(delta)
        x_star = (a + b) / 2
        x_history.append(x_star)
        k += 1

    x_star = (a + b) / 2
    f_x_star = func(x_star)

    return f_x_star, x_star, k, function_calls, delta_history, x_history


# --- Целевая функция для золотого сечения ---
def function(argument: float) -> float:
    return (argument - 2) ** 2 + 1


# --- Метод Брента для нахождения корня уравнения ---
def brent_method_root(a1: float, b1: float, eps1: float, func):
    """
    Реализует метод Брента для поиска корня уравнения.

    Args:
        a1 (float): Левая граница интервала поиска.
        b1 (float): Правая граница интервала поиска.
        eps1 (float): Точность поиска корня.
        func (function): Функция, корень которой нужно найти.

    Returns:
        tuple: Кортеж, содержащий:
            - root (float): Найденный корень.
            - iterations (int): Количество итераций.
    """
    if func(a1) * func(b1) > 0:
        print("Корень не существует на данном интервале!")
        return None, 0, 0, []

    iteration_count = 0
    while abs(b1 - a1) > eps1:
        iteration_count += 1
        x = (a1 + b1) / 2
        f_x = func(x)
```

```
        if func(a1) * f_x < 0:
            b1 = x
        else:
            a1 = x

        if abs(f_x) < eps1:
            return x, iteration_count

    return (a1 + b1) / 2, iteration_count


# --- Функции для нахождения корней ---
def function1(x):
    """
    Функция с гарантированным корнем (около 1.521).
    """
    return x ** 3 - x - 2


def function2(x):
    """
    Функция с гарантированным корнем (около 0.739).
    """
    return np.cos(x) - x


# --- Визуализация результатов ---
def plot_golden_ratio_results(a, b, x_history, delta_history, x_min, function):
    """
    Визуализирует результаты работы алгоритма золотого сечения.
    """
    # 1. График функции с точками экстремума
    x_values = np.linspace(a, b, 1000)
    y_values = [function(x) for x in x_values]

    plt.figure(figsize=(10, 6))
    plt.plot(x_values, y_values, label="Целевая функция")
    plt.scatter(
        x_history, [function(x) for x in x_history], color="red", label="Промежуточ
    )
    plt.axvline(x=x_min, color="green", linestyle="--", label=f"Минимум x  {x_min:.
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.title("Процесс поиска минимума методом золотого сечения")
    plt.legend()
    plt.grid(True)
    plt.show()

    # 2. График изменения длины интервала
    plt.figure(figsize=(10, 6))
```

```
    plt.plot(range(len(delta_history)), delta_history, marker="o", color="blue")
    plt.xlabel("Итерация")
    plt.ylabel("Длина интервала")
    plt.title("Изменение длины интервала на каждой итерации")
    plt.grid(True)
    plt.show()


def plot_all_functions():
    """
    Визуализирует все функции, используемые в коде.
    """
    # 1. График целевой функции для золотого сечения
    x_values = np.linspace(-5, 10, 400)
    y_values = [function(x) for x in x_values]
    plt.figure(figsize=(10, 5))
    plt.plot(x_values, y_values, label="Функция для золотого сечения: (x-2)^2 + 1")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.title("График функции для золотого сечения")
    plt.legend()
    plt.grid(True)
    plt.show()

    # 2. График function1
    x_values = np.linspace(-2, 3, 400)
    y_values = [function1(x) for x in x_values]
    plt.figure(figsize=(10, 5))
    plt.plot(x_values, y_values, label="Функция x^3 - x - 2")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.title("График функции x^3 - x - 2")
    plt.axhline(y=0, color="k", linestyle="--")  # Добавляем ось X
    plt.legend()
    plt.grid(True)
    plt.show()

    # 3. График function2
    x_values = np.linspace(-1, 2, 400)
    y_values = [function2(x) for x in x_values]
    plt.figure(figsize=(10, 5))
    plt.plot(x_values, y_values, label="Функция:cos(x) - x")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.title("График функции cos(x) - x")
    plt.axhline(y=0, color="k", linestyle="--")  # Добавляем ось X
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
if __name__ == "__main__":
    # --- Параметры задачи для золотого сечения ---
    a, b, eps = -100, 99, 0.01

    # --- Выполнение алгоритма золотого сечения ---
    result = golden_ratio_algorithm(a, b, eps, function)
    f_min, x_min, iterations, func_calls, delta_history, x_history = result
    print("--- Результаты работы алгоритма золотого сечения ---")
    print(f"Минимальное значение функции: {f_min:.5f}")
    print(f"Точка минимума: {x_min:.5f}")
    print(f"Количество итераций: {iterations}")
    print(f"Количество вызовов функции: {func_calls}")
    print(f"История сужения интервала: {delta_history}")

    # --- Визуализация результатов золотого сечения ---
    plot_golden_ratio_results(a, b, x_history, delta_history, x_min, function)

    # --- Параметры для метода Брента ---
    a1, b1, eps1 = 1, 2, 1e-6
    a2, b2, eps2 = 0, 1, 1e-6

    # --- Поиск корней методом Брента ---
    root1, iter1 = brent_method_root(a1, b1, eps1, function1)
    root2, iter2 = brent_method_root(a2, b2, eps2, function2)

    # --- Истинные корни для расчёта СКО ---
    true_root1 = 1.5213797068045676
    true_root2 = 0.7390851332151607

    # --- Расчёт СКО через NumPy с использованием .std() ---
    mse1 = np.array([root1, true_root1]).std()
    mse2 = np.array([root2, true_root2]).std()

    # --- Вывод результатов для метода Брента ---
    print("\n--- Результаты работы метода Брента ---")
    print(f"Корень функции x^3 - x - 2: {root1:.10f}, Итерации: {iter1}, СКО: {mse1
    print(f"Корень функции cos(x) - x: {root2:.10f}, Итерации: {iter2}, СКО: {mse2:

    # --- Визуализация всех функций ---
    plot_all_functions()
```