Министерство науки и высшего образования РФ
ФГАОУ ВПО
Национальный исследовательский технологический университет «МИСИС»

Институт компьютерных наук (ИКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

Отчет по контрольной работе №1
по дисциплине «Методы оптимизации»
на тему «Алгоритмы поиска экстремумов одномерной функции»

Выполнил:
студент группы БИСТ-22-3

Котов С. С.

Проверил:
доц. каф. ИКТ

Мокрова Н. В.

Москва, 2025

Цель работы: ознакомиться с методами одномерного поиска (метод золотого сечения и метод касательных). Сравнить эффективность этих алгоритмов на тестовой функции.

Задание:

- Исследовать функцию $f(x) = (x + 1) * e^{\cos(x)}$ графически найти решение задачи поиска экстремума.
- Реализовать этап отделения корней (сканирование).
- Реализовать численные методы решения задачи поиска безусловного экстремума функции: метод Фибоначчи и метод касательных (Ньютона).
- Сделать выводы об эффективности методов оптимизации.

Теоретические сведения

Метод Фибоначчи.

Метод Фибоначчи – это метод одномерной оптимизации, предназначенный для поиска минимума (или максимума) унимодальной функции на заданном отрезке. Как и метод золотого сечения, он основан на итеративном сужении интервала неопределенности. В отличие от метода золотого сечения, метод Фибоначчи использует коэффициенты, основанные на числах Фибоначчи.

Числа Фибоначчи: Последовательность чисел Фибоначчи определяется рекуррентно:

- $F_0 = 1$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ для $n \geq 2$

Алгоритм, использованный в работе:

1. Инициализация:
   - Задаются начальные границы отрезка [a, b] и требуемая точность tol.
   - Вычисляется n - количество итераций, необходимых для достижения заданной точности. Это делается путем нахождения такого минимального n, что $F_n > (b - a) / tol$.
   - Вычисляются числа Фибоначчи от F_0 до F_n.
2. Вычисление внутренних точек:
   - $x1 = a + (F_{n-2} / F_n) * (b - a)$

  - $x2 = a + (F_{n-1} / F_n) * (b - a)$
  - Вычисляются значения функции f(x1) и f(x2).
3. Вычисление значений функции:
  - Для поиска *минимума*: f1 = f(x1), f2 = f(x2).
  - Для поиска *максимума*: чтобы использовать тот же алгоритм, что и для поиска минимума, функцию f(x) заменяют на -f(x). В коде это реализовано через f_adapt.
4. Итеративное сужение интервала:
  - Поиск минимума:
    - Если f(x1) > f(x2), то минимум находится на интервале [x1, b]. Новые значения: a = x1, x1 = x2, f(x1) = f(x2). Затем x2 пересчитывается: $x2 = a + (F_i / F_{i+1}) * (b - a)$, где i - текущий номер итерации (уменьшается от n-2 до 1). f(x2) тоже пересчитывается.
    - Если f(x1) <= f(x2), то минимум находится на интервале [a, x2]. Новые значения: b = x2, x2 = x1, f(x2) = f(x1). Затем x1 пересчитывается: $x1 = a + (F_{i-1} / F_{i+1}) * (b - a)$. f(x1) тоже пересчитывается.
  - Поиск максимума: Аналогично поиску минимума, но сравниваются f(x1) и f(x2) в обратном порядке (или используется функция -f(x)).
5. Условие остановки: Итерации продолжаются, пока i > 0 (или пока длина интервала (b - a) не станет меньше tol, что в данном алгоритме не проверяется явно на каждой итерации из-за предопределенного n).
6. **Результат:** В качестве приближенного значения точки экстремума принимается середина последнего интервала: (a + b) / 2.

Метод касательных (Ньютона).

Метод Ньютона (метод касательных) – это итерационный метод, который использует информацию о первой и второй производных функции для поиска её стационарных точек (точек, где первая производная равна нулю). Стационарные точки могут быть точками минимума, максимума или точками перегиба.

Алгоритм, использованный в коде:

1. Инициализация: задаётся начальное приближение x0, требуемая точность (tol) и максимальное количество итераций (max_iter).

2. Итеративное уточнение:
   - Вычисляется первая производная f'(x) и вторая производная f''(x) в текущей точке x. В коде для этого используются функции df(f, x) и ddf(f, x), которые вычисляют производные численно.
   - Следующее приближение вычисляется по формуле: x_new = x_current - f'(x_current) / f''(x_current)
3. Условие остановки: Итерации продолжаются до тех пор, пока абсолютная разница между двумя последовательными приближениями (|x_new - x_current|) не станет меньше заданной точности (tol), *или* не будет достигнуто максимальное число итераций (max_iter), *или* пока вторая производная не станет слишком мала по модулю, или пока производная не вернет бесконечность.
4. **Результат:** в качестве приближенного значения точки экстремума принимается последнее полученное приближение x_current.

Метод находит стационарные точки, а они могут быть и минимумами, и максимумами, и точками перегиба.

Реализация методов (без машинный вариант)

Функция: $f(x) = (x + 1) * e^{\cos(x)}$

- Метод Фибоначчи
  1. Определение количества итераций (n) для заданной точности (tol):

  Пусть tol = 0.1 (требуемая точность для x, как в примере). Нам нужно найти такое n, что F_n > (b - a) / tol.
  В нашем случае: (b - a) / tol = (-1 - (-2)) / 0.1 = 1 / 0.1 = 10

  2. Вычисляем числа Фибоначчи:
     - $F_0 = 1$
     - $F_1 = 1$
     - $F_2 = 2$
     - $F_3 = 3$
     - $F_4 = 5$
     - $F_5 = 8$
     - $F_6 = 13$

  Поскольку $F_6 = 13 > 10$, то n = 6. Это означает, что для достижения точности 0.1 по x нам потребовалось бы 6 итераций.

3. Ручной расчет:
   - Инициализация:
     - a = -2
     - b = -1
     - n = 3 (поскольку мы ограничиваемся двумя итерациями + 1 начальное вычисление; F_3=2)
     - F_0 = 1, F_1 = 1, F_2 = 2, F_3 = 3
   - Вычисление x_1 и x_2:
     - $x_1 = a + (F_{n-2} / F_n) * (b - a) = -2 + (F_1 / F_3) * (-1 - (-2)) = -2 + (1 / 3) * 1 = -2 + 1/3 = -1.666... \approx -1.667$
     - $x_2 = a + (F_{n-1} / F_n) * (b - a) = -2 + (F_2 / F_3) * (-1 - (-2)) = -2 + (2 / 3) * 1 = -2 + 2/3 = -1.333... \approx -1.333$
     - $f(x_1) = f(-1.667) \approx (-1.667 + 1) * e^\wedge(\cos(-1.667)) \approx -0.667 * e^\wedge(-0.096) \approx -0.667 * 0.908 \approx -0.606$
     - $f(x_2) = f(-1.333) \approx (-1.333 + 1) * e^\wedge(\cos(-1.333)) \approx -0.333 * e^\wedge(0.236) \approx -0.333 * 1.266 \approx -0.422$
   - Итерация 1:
     - Сравниваем: $f(x_1) \approx -0.606 < f(x_2) \approx -0.422$. Значит, минимум на интервале $[a, x_2]$.
     - Обновляем:
       1. $b = x_2 \approx -1.333$
       2. $x_2 = x_1 \approx -1.667$
       3. $f(x_2) = f(x_1)$
       4. $x_1 = a + (F_{i-1} / F_{i+1}) * (b - a)$, где i = n - 2 = 1. Значит, $x_1 = -2 + (F_0 / F_2) * (-1.333 - (-2)) = -2 + (1/2)*0.667= -2 + 0.3335 = -1.6665 \approx -1.667$
       5. $f(x_1) = (-1.667+1) * e^\wedge(\cos(-1.667) \approx -0.606$
   - Итерация 2:
     - Сравниваем f(x1) и f(x2).
       1. x1 = -1.667, f(x1) ≈ -0.606
       2. x2 = -1.667. f(x2) ≈ -0.606
   - Так как значения x1 и x2, а также f(x1) и f(x2) совпали в пределах точности вычислений, дальнейшие итерации не дадут сужения интервала.
   - В качестве результата берем среднее (a+b)/2

- Метод касательных (Ньютона)
  Начальное приближение: x0 = -1.5.

1. Инициализация:

- x0 = -1.5
- Находим первую и вторую производные:
  - f(x) = (x + 1) * e^(cos(x))
  - f'(x) = e^(cos(x)) + (x + 1) * e^(cos(x)) * (-sin(x)) = e^(cos(x)) * (1 - (x + 1) * sin(x))
  - f''(x) = -sin(x)*e^(cos(x))*(1-(x+1)sin(x)) + e^(cos(x))*(-sin(x)-(x+1)cos(x)) = e^(cos(x)) * (-sin(x) - (x + 1)cos(x) - sin(x) + (x+1)sin(x)^2) = e^(cos(x)) * (-2sin(x) - (x+1)cos(x) + (x+1)sin²(x))

2. Итерация 1:

- Вычисляем f'(-1.5):

  f'(-1.5) = e^(cos(-1.5)) * (1 - (-1.5 + 1) * sin(-1.5)) ≈ e^(0.0707) * (1 - (-0.5) * (-0.997)) ≈ 1.073 * (1 - 0.4985) ≈ 1.073 * 0.5015 ≈ 0.538

- Вычисляем f''(-1.5):

  f''(-1.5) ≈ e^(0.0707) * (-2*(-0.997) - (-0.5)*0.0707 + (-0.5)*(-0.997)^2) ≈ 1.073 * (1.994 + 0.0353 - 0.497) ≈ 1.073 * 1.532 ≈ 1.644

- Вычисляем x_1:

  $x_1 = x_0$ - f'(x_0) / f''(x_0) ≈ -1.5 - 0.538 / 1.644 ≈ -1.5 - 0.327 ≈ -1.827

3. Итерация 2:

- Вычисляем f'(-1.827):

  f'(-1.827) = e^(cos(-1.827)) * (1 - (-1.827 + 1) * sin(-1.827)) ≈ e^(-0.254) * (1 - (-0.827) * (-0.974)) ≈ 0.776 * (1-0.805) ≈ 0.776 * 0.195 ≈ 0.151

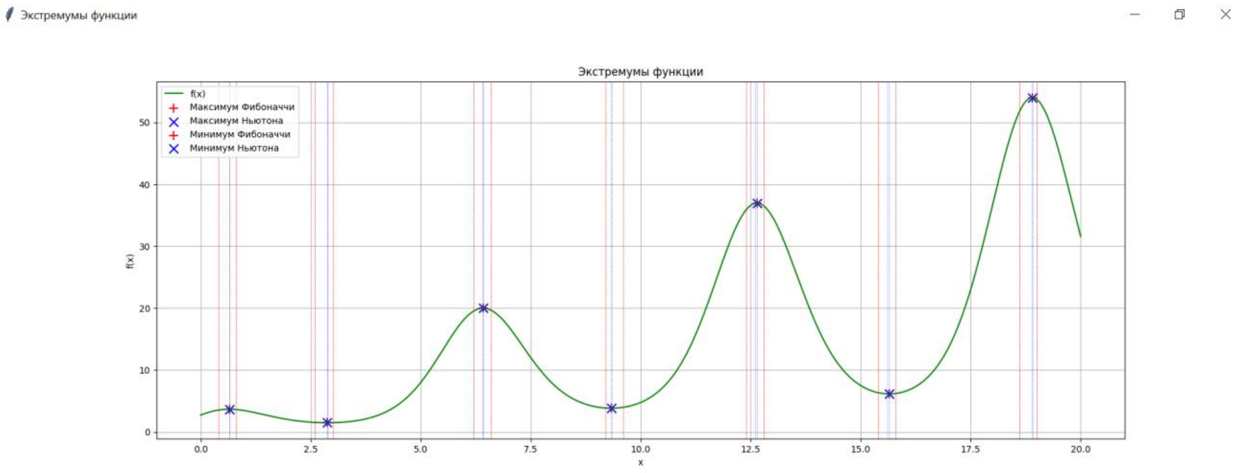
- Вычисляем f''(-1.827):

  f''(-1.827) ≈ e^(-0.254) * (-2*(-0.974) - (-1.827+1)*(-0.254) + (-1.827+1)*(-0.974)^2) ≈ 0.776 *(1.948 - 0.210 -0.785) ≈ 0.776*0.953 ≈ 0.739

- Вычисляем x_2:

  $x_2 = x_1$ - f'(x_1) / f''(x_1) ≈ -1.827 - 0.151 / 0.739 ≈ -1.827 - 0.204 ≈ -2.031

Результаты работы программы



Тип	Метод	x	f(x)	Итерации	Диапазон
Максимум	Фибоначчи	0.650753	3.657791	17	0.400000 - 0.800000
Максимум	Ньютона	0.650752	3.657791	3	0.649788 - 0.650752
Минимум	Фибоначчи	2.880985	1.476767	17	2.600000 - 3.000000
Минимум	Ньютона	2.880986	1.476767	3	2.881800 - 2.880986
Максимум	Фибоначчи	6.418417	19.982079	17	6.200000 - 6.600000
Максимум	Ньютона	6.418397	19.982079	2	6.400000 - 6.418397
Минимум	Фибоначчи	9.327816	3.817279	17	9.200000 - 9.600000
Минимум	Ньютона	9.327800	3.817279	3	9.328300 - 9.327800
Максимум	Фибоначчи	12.639751	36.977043	17	12.400000 - 12.800000
Максимум	Ньютона	12.639752	36.977043	2	12.600000 - 12.639752

Таблица 1. Результаты поиска экстремумов методами Фибоначчи и Ньютона (касательных)

Тип	Метод	x	f(x)	Итерации	Диапазон
Максимум	Фибоначчи	0.650753	3.657791	17	0.400000 - 0.800000
Максимум	Ньютона	0.650752	3.657791	3	0.649788 - 0.650752
Минимум	Фибоначчи	2.880985	1.476767	17	2.600000 - 3.000000
Минимум	Ньютона	2.880986	1.476767	3	2.881800 - 2.880986
Максимум	Фибоначчи	6.418417	19.982079	17	6.200000 - 6.600000
Максимум	Ньютона	6.418397	19.982079	2	6.400000 - 6.418397
Минимум	Фибоначчи	9.327816	3.817279	17	9.200000 - 9.600000
Минимум	Ньютона	9.327800	3.817279	3	9.328300 - 9.327800
Максимум	Фибоначчи	12.639751	36.977043	17	12.400000 - 12.800000
Максимум	Ньютона	12.639752	36.977043	2	12.600000 - 12.639752

Тип	Метод	x	f(x)	Итерации	Диапазон
Максимум	Фибоначчи	6.418417	19.982079	17	6.200000 - 6.600000
Максимум	Ньютона	6.418397	19.982079	2	6.400000 - 6.418397
Минимум	Фибоначчи	9.327816	3.817279	17	9.200000 - 9.600000
Минимум	Ньютона	9.327800	3.817279	3	9.328300 - 9.327800
Максимум	Фибоначчи	12.639751	36.977043	17	12.400000 - 12.800000
Максимум	Ньютона	12.639752	36.977043	2	12.600000 - 12.639752
Минимум	Фибоначчи	15.647883	6.135474	17	15.400000 - 15.800000
Минимум	Ньютона	15.647859	6.135474	2	15.600000 - 15.647859
Максимум	Фибоначчи	18.899833	54.025044	17	18.600000 - 19.000000
Максимум	Ньютона	18.899829	54.025044	3	18.900921 - 18.899829

Сравнение эффективности

- **Точность:** Оба метода могут находить экстремумы с высокой точностью. Однако, точность метода Фибоначчи заранее определяется количеством итераций (или заданной tol), в то время как точность метода Ньютона зависит от поведения функции и её производных в окрестности экстремума и может быть как очень высокой, так и низкой (если метод расходится).
- **Скорость сходимости (количество итераций):** Метод Ньютона обычно сходится значительно быстрее метода Фибоначчи. Метод Фибоначчи имеет линейную сходимость, а метод Ньютона имеет квадратичную сходимость вблизи корня (при выполнении определенных условий). Это означает, что количество верных цифр в приближении метода Ньютона примерно удваивается с каждой итерацией (вблизи решения), в то время как в методе Фибоначчи количество верных цифр увеличивается на постоянную величину с

каждой итерацией. Количество итераций в методе Фибоначчи определяется заранее (или по достижении tol).

- **Универсальность:**
  - Метод Фибоначчи требует, чтобы на начальном интервале функция была унимодальной. Предварительное сканирование, используемое в коде, разбивает область определения на интервалы унимодальности.
  - Метод Ньютона не требует унимодальности, но он чувствителен к начальному приближению и может:
    - Разойтись (уйти в бесконечность).
    - Сойтись к другому экстремуму (не к ближайшему).
    - Осциллировать (колебаться вокруг экстремума).
  - Метод Ньютона требует существования и непрерывности первой и второй производных в окрестности экстремума. Если вторая производная равна нулю в точке экстремума, сходимость может быть медленнее, чем квадратичная.
- **Вычислительная сложность:**
  - Метод Ньютона требует существования и непрерывности первой и второй производных в окрестности экстремума. Если вторая производная равна нулю в точке экстремума, сходимость может быть медленнее, чем квадратичная.
  - Метод Ньютона требует вычисления значения функции, её первой и второй производных на каждой итерации. Численное дифференцирование упрощает задачу, но все равно делает метод Ньютона более вычислительно сложным. Если производные можно вычислить аналитически, сложность уменьшается.

Вывод: Для данной функции и заданных условий метод Ньютона показал более высокую скорость сходимости, чем метод Фибоначчи, при условии, что он сошелся. Это связано с квадратичной сходимостью метода Ньютона вблизи экстремума. Однако метод Ньютона менее надежен и более требователен к вычислительным ресурсам (из-за вычисления производных). Метод Фибоначчи, хотя и сходится медленнее, является более надежным и менее требовательным к функции (требует только унимодальности на интервале и не требует вычисления производных). Выбор метода зависит от конкретной задачи, требований к точности, скорости и надежности, а также от доступности информации о производных. Предварительное сканирование для поиска интервалов унимодальности является важным шагом для обоих методов, если функция не является унимодальной на всем интервале.

Приложение (Листинг программы)

```
import numpy as np
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import ttk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

def f(x):
    return (x + 1) * np.exp(np.cos(x))

def df(x, h=1e-6):
    return (f(x + h) - f(x - h)) / (2 * h)

def ddf(x, h=1e-6):
    return (df(x + h) - df(x - h)) / (2 * h)

def fibonacci_min(a, b, tol=1e-6, max_iter=100):
    fib = [1, 1]
    while fib[-1] < (b - a) / tol:
        fib.append(fib[-1] + fib[-2])
    n = len(fib) - 1

    initial_a, initial_b = a, b
    x1 = a + (fib[n - 2] / fib[n]) * (b - a)
    x2 = a + (fib[n - 1] / fib[n]) * (b - a)
    fx1 = f(x1)
    fx2 = f(x2)
    intervals = 1

    for i in range(n - 2, 0, -1):
        if fx1 < fx2:
            b = x2
            x2 = x1
            fx2 = fx1
            x1 = a + (fib[i - 1] / fib[i + 1]) * (b - a)
            fx1 = f(x1)
        else:
            a = x1
            x1 = x2
            fx1 = fx2
            x2 = a + (fib[i] / fib[i + 1]) * (b - a)
            fx2 = f(x2)
        intervals += 1
        if (b - a) < tol:
            break
    return (a + b) / 2, intervals, (initial_a, initial_b)

def fibonacci_max(a, b, tol=1e-6, max_iter=100):
    fib = [1, 1]
    while fib[-1] < (b - a) / tol:
        fib.append(fib[-1] + fib[-2])
    n = len(fib) - 1

    initial_a, initial_b = a, b
    x1 = a + (fib[n - 2] / fib[n]) * (b - a)
    x2 = a + (fib[n - 1] / fib[n]) * (b - a)
    fx1 = f(x1)
    fx2 = f(x2)
    intervals = 1

    for i in range(n - 2, 0, -1):
        if fx1 > fx2:
            b = x2
            x2 = x1
```

```
            fx2 = fx1
            x1 = a + (fib[i - 1] / fib[i + 1]) * (b - a)
            fx1 = f(x1)
        else:
            a = x1
            x1 = x2
            fx1 = fx2
            x2 = a + (fib[i] / fib[i + 1]) * (b - a)
            fx2 = f(x2)
        intervals += 1
        if (b - a) < tol:
            break
    return (a + b) / 2, intervals, (initial_a, initial_b)

def tangent_method_extrema(a, b, dfx, d2fx, tol=1e-6, max_iter=100):
    initial_a, initial_b = a, b
    x0 = (a + b) / 2
    intervals = 0
    prev_x = x0

    for _ in range(max_iter):
        df_x0 = dfx(x0)
        ddf_x0 = d2fx(x0)

        if abs(ddf_x0) < 1e-12 or not np.isfinite(ddf_x0) or not
np.isfinite(df_x0):
            return None, intervals, (initial_a, initial_b)

        x1 = x0 - df_x0 / ddf_x0
        x1 = np.clip(x1, a, b)
        intervals += 1

        if abs(x1 - x0) < tol:
            return x1, intervals, (prev_x, x1)

        prev_x = x0
        x0 = x1

    return x0, intervals, (prev_x, x0)

def scan_for_all_extrema(a, b, h):
    x_values = np.arange(a, b + h, h)
    y_values = f(x_values)
    extrema_intervals = []

    for i in range(1, len(x_values) - 1):
        if y_values[i] < y_values[i - 1] and y_values[i] < y_val-
ues[i + 1]:
            extrema_intervals.append((x_values[i - 1], x_values[i +
1], 'min'))
        elif y_values[i] > y_values[i - 1] and y_values[i] > y_val-
ues[i + 1]:
            extrema_intervals.append((x_values[i - 1], x_values[i +
1], 'max'))
    return extrema_intervals

def find_extrema(a, b, h_scan, tol):
    extrema_intervals = scan_for_all_extrema(a, b, h_scan)
    results = []

    for interval_a, interval_b, type_ in extrema_intervals:
        x_fib = y_fib = iter_fib = rng_fib = None
        x_newton = y_newton = iter_newton = rng_newton = None

        if type_ == 'max':
```

```
            x_fib, iter_fib, rng_fib = fibonacci_max(interval_a, in-
terval_b, tol=tol)
            x_newton, iter_newton, rng_newton = tangent_method_ex-
trema(interval_a, interval_b, df, ddf, tol=tol)
        elif type_ == 'min':
            x_fib, iter_fib, rng_fib = fibonacci_min(interval_a, in-
terval_b, tol=tol)
            x_newton, iter_newton, rng_newton = tangent_method_ex-
trema(interval_a, interval_b, df, ddf, tol=tol)

        results.append({
            'type': type_,
            'fibonacci': (x_fib, f(x_fib) if x_fib is not None else
None, iter_fib, rng_fib),
            'newton': (x_newton, f(x_newton) if x_newton is not None
else None, iter_newton, rng_newton)
        })

    return results

def plot_results(a, b, results):
    fig, ax = plt.subplots(figsize=(8, 5))

    x_graph = np.linspace(a, b, 400)
    y_graph = f(x_graph)
    ax.plot(x_graph, y_graph, label="f(x)", color="green")

    legend_added = {"fibonacci_max": False, "newton_max": False,
                    "fibonacci_min": False, "newton_min": False}

    for res in results:
        # Обработка Фибоначчи
        x_fib, y_fib, iter_fib, rng_fib = res["fibonacci"]
        if x_fib is not None:
            if res["type"] == "max":
                type_str = "Максимум"
            elif res["type"] == "min":
                type_str = "Минимум"
            else:
                type_str = "Экстремум"

            lab_fib = f'{type_str} Фибоначчи'
            if legend_added.get(f"fibonacci_{res['type']}"):
                lab_fib = None
            else:
                legend_added[f"fibonacci_{res['type']}"] = True
            ax.scatter(x_fib, y_fib, color="red", marker='+', s=100,
label=lab_fib)
            ax.axvline(x=rng_fib[0], color="red", linestyle='--',
linewidth=0.5)
            ax.axvline(x=rng_fib[1], color="red", linestyle='--',
linewidth=0.5)

        # Обработка Ньютона (Касательных)
        x_newton, y_newton, iter_newton, rng_newton = res["newton"]
        if x_newton is not None:
            if res["type"] == "max":
                type_str = "Максимум"
            elif res["type"] == "min":
                type_str = "Минимум"
            else:
                type_str = "Экстремум"

            lab_newton = f'{type_str} Ньютона'
            if legend_added.get(f"newton_{res['type']}"):
                lab_newton = None
```

```
            else:
                legend_added[f"newton_{res['type']}"] = True

            ax.scatter(x_newton, y_newton, color="blue", marker='x',
s=100, label=lab_newton)
            rng_newton_clipped = np.clip(rng_newton, a, b)
            ax.axvline(x=rng_newton_clipped[0], color="blue", lin-
estyle=':', linewidth=0.5)
            ax.axvline(x=rng_newton_clipped[1], color="blue", lin-
estyle=':', linewidth=0.5)

    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.set_title("Экстремумы функции")
    ax.grid(True)
    ax.legend()
    return fig

def run_app():
    a, b = 0, 20
    h_scan = 0.2
    tol = 1e-4

    results = find_extrema(a, b, h_scan, tol)

    root = tk.Tk()
    root.title("Экстремумы функции")
    root.geometry("1200x700")

    frame_fig = tk.Frame(root)
    frame_fig.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

    fig = plot_results(a, b, results)
    canvas = FigureCanvasTkAgg(fig, master=frame_fig)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, ex-
pand=True)

    frame_table = tk.Frame(root)
    frame_table.pack(side=tk.BOTTOM, fill=tk.X, padx=10, pady=10)

    tree = ttk.Treeview(frame_table, columns=('type', 'method', 'x',
'f(x)', 'iterations', 'range'), show='headings')
    tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

    tree.heading('type', text='Тип')
    tree.heading('method', text='Метод')
    tree.heading('x', text='x')
    tree.heading('f(x)', text='f(x)')
    tree.heading('iterations', text='Итерации')
    tree.heading('range', text='Диапазон')

    tree.column('type', width=100, anchor='center')
    tree.column('method', width=150, anchor='center')
    tree.column('x', width=120, anchor='center')
    tree.column('f(x)', width=120, anchor='center')
    tree.column('iterations', width=100, anchor='center')
    tree.column('range', width=200, anchor='center')

    for res in results:
        typ = res["type"]
        for method in ["fibonacci", "newton"]:
            x_val, y_val, iterations, rng = res[method]
            if x_val is None:
                x_str = "None"
                y_str = "None"
```

```
            else:
                x_str = f"{x_val:.6f}"
                y_str = f"{y_val:.6f}" if y_val is not None else
"None"

            if typ == "max":
                type_str = "Максимум"
            elif typ == "min":
                type_str = "Минимум"
            else:
                type_str = "Экстремум"

            if method == "fibonacci":
                method_str = "Фибоначчи"
            elif method == "newton":
                method_str = "Ньютона"
            else:
                method_str = "Неизвестный"

            range_str = f"{rng[0]:.6f} - {rng[1]:.6f}"
            tree.insert('', tk.END, values=(type_str, method_str,
x_str, y_str, iterations, range_str))

    scrollbar = ttk.Scrollbar(frame_table, orient=tk.VERTICAL, com-
mand=tree.yview)
    tree.configure(yscroll=scrollbar.set)
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

    hscrollbar = ttk.Scrollbar(root, orient=tk.HORIZONTAL, com-
mand=tree.xview)
    tree.configure(xscroll=hscrollbar.set)
    hscrollbar.pack(side=tk.BOTTOM, fill=tk.X)
    tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

    root.mainloop()

if __name__ == "__main__":
    run_app()
```