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 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:

* F₀ = 1
* F₁ = 1
* Fₙ = Fₙ₋₁ + Fₙ₋₂ для n ≥ 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 Fₙ > (b - a) / tol.
   * Вычисляются числа Фибоначчи от F₀ до Fₙ.
2. Вычисление внутренних точек:
   * x1 = a + (Fₙ₋₂ / Fₙ) \* (b - a)
   * x2 = a + (Fₙ₋₁ / Fₙ) \* (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 + (Fᵢ / Fᵢ₊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 + (Fᵢ₋₁ / Fᵢ₊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:

* 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 Fₙ > (b - a) / tol.

В нашем случае: (b - a) / tol = (-1 - (-2)) / 0.1 = 1 / 0.1 = 10

* 1. Вычисляем числа Фибоначчи:
     + F₀ = 1
     + F₁ = 1
     + F₂ = 2
     + F₃ = 3
     + F₄ = 5
     + F₅ = 8
     + F₆ = 13

Поскольку F₆ = 13 > 10, то n = 6. Это означает, что для достижения точности 0.1 по x нам потребовалось бы 6 итераций.

* 1. Ручной расчет:
     + Инициализация:
       - a = -2
       - b = -1
       - n = 3 (поскольку мы ограничиваемся двумя итерациями + 1 начальное вычисление; F₃=2)
       - F₀ = 1, F₁ = 1, F₂ = 2, F₃ = 3
     + Вычисление x₁ и x₂:
       - x₁ = a + (Fₙ₋₂ / Fₙ) \* (b - a) = -2 + (F₁ / F₃) \* (-1 - (-2)) = -2 + (1 / 3) \* 1 = -2 + 1/3 = -1.666... ≈ -1.667
       - x₂ = a + (Fₙ₋₁ / Fₙ) \* (b - a) = -2 + (F₂ / F₃) \* (-1 - (-2)) = -2 + (2 / 3) \* 1 = -2 + 2/3 = -1.333... ≈ -1.333
       - f(x₁) = f(-1.667) ≈ (-1.667 + 1) \* e^(cos(-1.667)) ≈ -0.667 \* e^(-0.096) ≈ -0.667 \* 0.908 ≈ -0.606
       - f(x₂) = f(-1.333) ≈ (-1.333 + 1) \* e^(cos(-1.333)) ≈ -0.333 \* e^(0.236) ≈ -0.333 \* 1.266 ≈ -0.422
     + Итерация 1:
       - Сравниваем: f(x₁) ≈ -0.606 < f(x₂) ≈ -0.422. Значит, минимум на интервале [a, x₂].
       - Обновляем:
         1. b = x₂ ≈ -1.333
         2. x₂ = x₁ ≈ -1.667
         3. f(x₂) = f(x₁)
         4. x₁ = a + (Fᵢ₋₁ / Fᵢ₊₁) \* (b - a), где i = n - 2 = 1. Значит, x₁ = -2 + (F₀ / F₂) \* (-1.333 - (-2)) = -2 + (1/2)\*0.667= -2 + 0.3335 = -1.6665 ≈ -1.667
         5. f(x₁) = (-1.667+1) \* e^(cos(-1.667) ≈ -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 x₁:

x₁ = x₀ - f'(x₀) / f''(x₀) ≈ -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 x₂:

x₂ = x₁ - f'(x₁) / f''(x₁) ≈ -1.827 - 0.151 / 0.739 ≈ -1.827 - 0.204 ≈ -2.031

**Результаты работы программы**

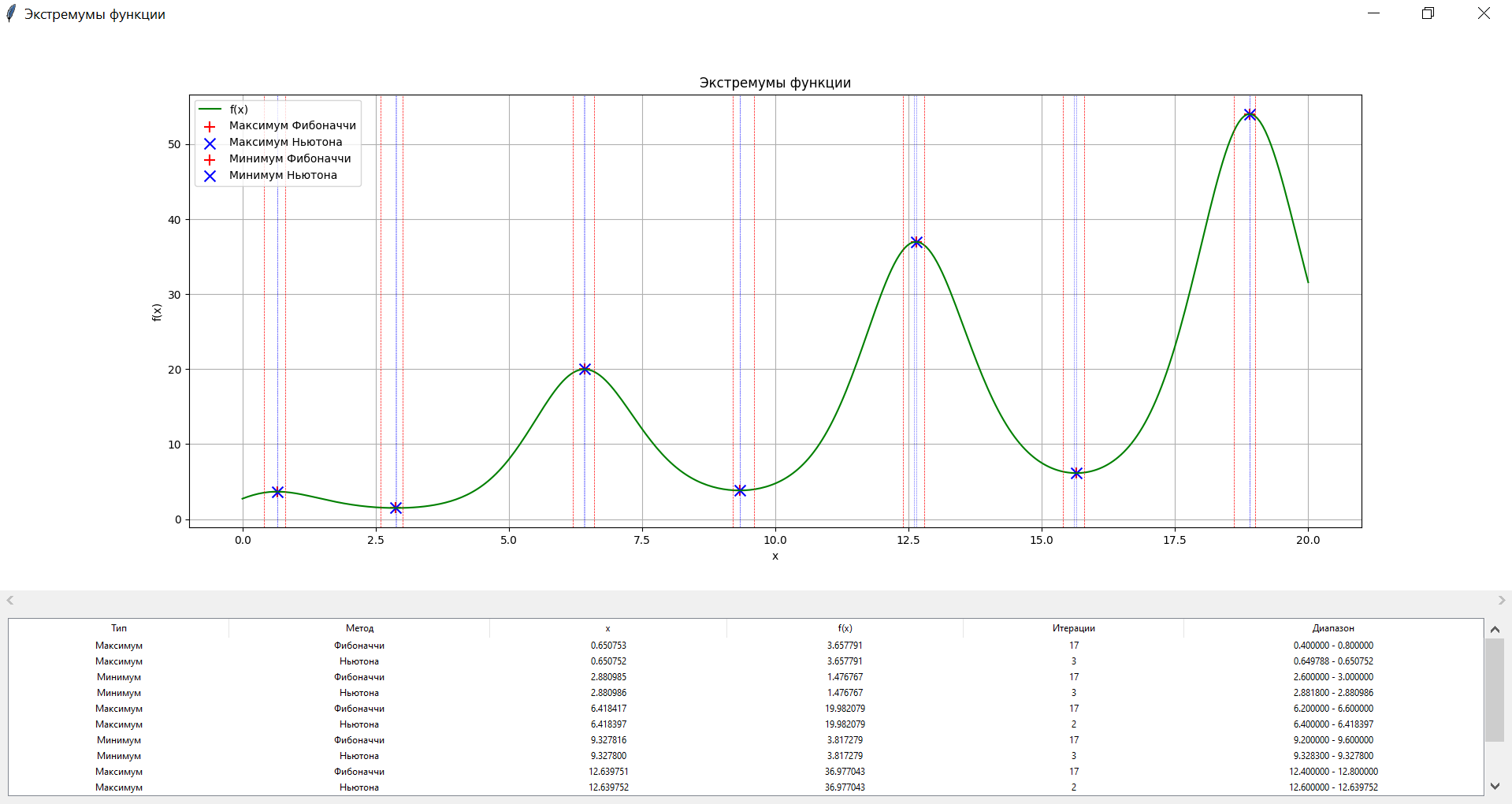
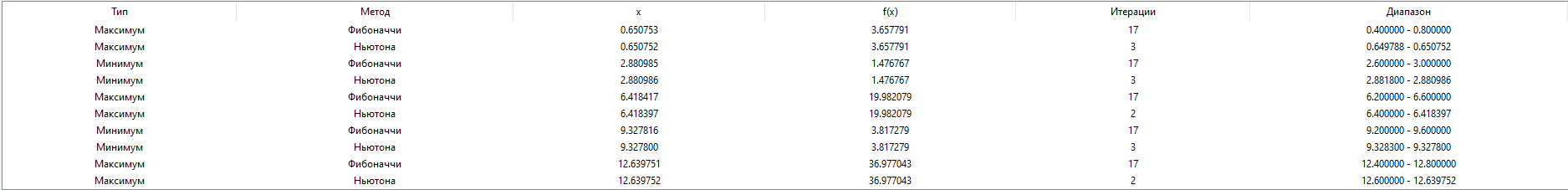
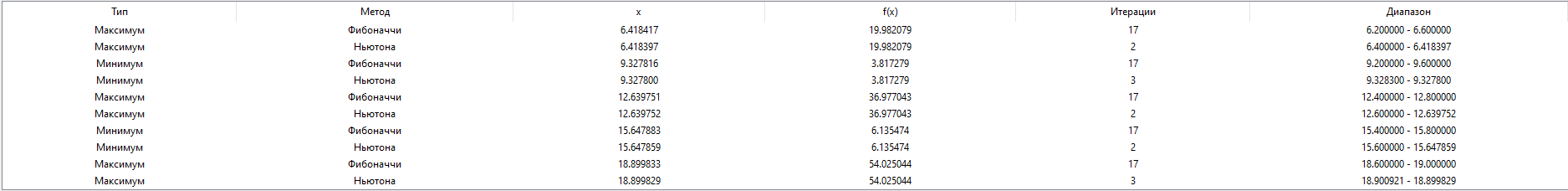


Таблица 1. Результаты поиска экстремумов методами Фибоначчи и Ньютона (касательных)





**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\_values[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\_values[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, interval\_b, tol=tol)  x\_newton, iter\_newton, rng\_newton = tangent\_method\_extrema(interval\_a, interval\_b, df, ddf, tol=tol)  elif type\_ == 'min':  x\_fib, iter\_fib, rng\_fib = fibonacci\_min(interval\_a, interval\_b, tol=tol)  x\_newton, iter\_newton, rng\_newton = tangent\_method\_extrema(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", linestyle=':', linewidth=0.5)  ax.axvline(x=rng\_newton\_clipped[1], color="blue", linestyle=':', 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, expand=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, command=tree.yview)  tree.configure(yscroll=scrollbar.set)  scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  hscrollbar = ttk.Scrollbar(root, orient=tk.HORIZONTAL, command=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() |