**Методы принятия оптимальных решений**

**Лабораторная работа №1**

**Выполнил студент группы ИС-23**

**Поляков Никита Валериевич**

**Оглавление**

[**Введение 3**](#_Toc211429476)

[**Исходные данные 4**](#_Toc211429477)

[**Программа реализации 5**](#_Toc211429478)

[**Заключение 14**](#_Toc211429479)

# **Введение**

Целью данной лабораторной работы является знакомство с классическими методами одномерного поиска минимума унимодальной функции — методом дихотомии и методом золотого сечения, а также сравнение их эффективности по числу вычислений функции, необходимых для достижения заданной точности.

# **Исходные данные**

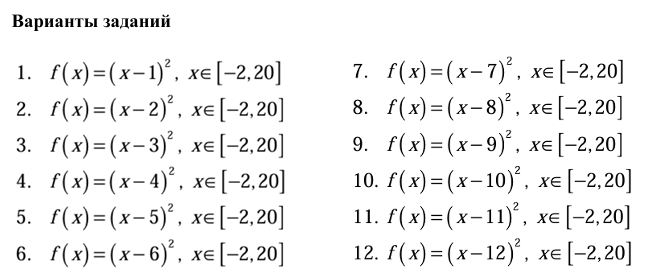
**Цель работы:** ознакомиться с методами одномерного поиска, используемыми в многомерных методах минимизации функций n переменных. Сравнить различные алгоритмы по эффективности на тестовых примерах.

**Порядок выполнения работы**.

Разработать ПО, реализующее методы дихотомии и золотого сечения. Результаты работы программы выводить в виде таблицы, где должны быть отражены границы и длины интервалов на каждой итерации (формат таблице см. ниже).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | x1 | x2 | f(x1) | f(x2) | ai | bi | bi - ai | (bi-1 - ai-1)/  (bi - ai) |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| … |  |  |  |  |  |  |  |  |

Провести сравнение методов дихотомии и золотого сечения по числу вычислений функции для достижения заданной точности ε от 10−1 до 10−7. Построить график зависимости количества вычислений функции от десятичного логарифма задаваемой точности ε.



# **Программа реализации**

Программа реализована на языке Python — высокоуровневом, интерпретируемом языке с мощной поддержкой научных вычислений. Для решения задачи использованы следующие библиотеки:

NumPy — для математических операций (в частности, вычисления константы золотого сечения);

Pandas — для структурированной обработки и форматирования табличных данных;

Matplotlib — для построения графика зависимости числа вычислений функции от заданной точности;

OpenPyXL — для записи результатов непосредственно в файл формата Excel (.xlsx), включая вставку графика как изображения;

OS — для управления временными файлами (удаление временного изображения после вставки в Excel).

Программа является универсальной: для адаптации под любой вариант задания достаточно изменить параметры в блоке VARIANT — целевую функцию, начальный отрезок [a, b] и истинное значение минимума.

На каждой итерации фиксируются:

длина интервала и коэффициент сжатия (отношение длины предыдущего интервала к текущему).

Сбор статистикиПодсчитывается общее количество вычислений функции для каждого метода при каждой точности.

В консоль выводятся: сводная таблица сравнения числа вычислений; детальные таблицы шагов для ε = 10⁻⁴ (по одному на метод); краткий вывод преимуществе метода золотого сечения. В файл lab1\_excel.xlsx сохраняются: лист «Сравнение и вывод» с таблицей эффективности и текстовым заключением; листы «Дихотомия\_1e-4» и «Золотое\_сечение\_1e-4» с полными итерационными таблицами; отдельный лист «График» с изображением зависимости числа вычислений от точности в логарифмическом масштабе. Особенности реализации: В методе золотого сечения используется константа k = 2 – φ ≈ 0.382, что обеспечивает оптимальное сжатие интервала. При каждом шаге метод золотого сечения переиспользует одно значение функции из предыдущей итерации, что снижает общее число вычислений по сравнению с дихотомией. Все числовые значения точности (ε) форматируются в читаемый десятичный вид (например, 1e-3 → 0.001). Временный график сохраняется как temp\_plot.png, вставляется в Excel и затем удаляется, чтобы не засорять рабочую директорию. Ширина столбцов в Excel автоматически настраивается для удобного отображения данных.

В результате выполнения программы генерируются: две подробные таблицы итераций (для ε = 10⁻⁴); сводная таблица сравнения эффективности методов по всем точностям; график, наглядно демонстрирующий, что метод золотого сечения требует меньше вычислений функции при той же точности, благодаря более эффективной стратегии сокращения интервала.

Все выходные файлы обновляются автоматически при каждом запуске, гарантируя актуальность результатов.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import openpyxl

from openpyxl.drawing.image import Image as ExcelImage

from openpyxl.utils.dataframe import dataframe\_to\_rows

import os

# === НАСТРОЙКИ ===

VARIANT = {

    "function": lambda x: (x - 12) \*\* 2,

    "a": -2,

    "b": 20,

    "true\_min\_x": 12,

}

EPSILONS = [10\*\*(-i) for i in range(1, 8)]  # [0.1, 0.01, ..., 1e-7]

EXCEL\_FILE = "lab1\_excel.xlsx"

PLOT\_FILE = "temp\_plot.png"

def metod\_dihotomii(f, a, b, eps):

    delta = eps / 3.0

    shagi = []

    chislo\_vychisleniy = 0

    iteratsiya = 1

    while b - a > eps:

        mid = (a + b) / 2

        x1, x2 = mid - delta, mid + delta

        f1, f2 = f(x1), f(x2)

        chislo\_vychisleniy += 2

        shagi.append({"i": iteratsiya, "x1": x1, "x2": x2, "f(x1)": f1, "f(x2)": f2, "a\_i": a, "b\_i": b, "b\_i - a\_i": b - a})

        if f1 < f2:

            b = x2

        else:

            a = x1

        iteratsiya += 1

    return (a + b) / 2, chislo\_vychisleniy, shagi

def metod\_zolotogo\_secheniya(f, a, b, eps):

    k = 2 - (1 + np.sqrt(5)) / 2

    x1 = a + k \* (b - a)

    x2 = b - k \* (b - a)

    f1, f2 = f(x1), f(x2)

    chislo\_vychisleniy = 2

    shagi = []

    iteratsiya = 1

    while b - a > eps:

        shagi.append({"i": iteratsiya, "x1": x1, "x2": x2, "f(x1)": f1, "f(x2)": f2, "a\_i": a, "b\_i": b, "b\_i - a\_i": b - a})

        if f1 < f2:

            b, x2, f2 = x2, x1, f1

            x1 = a + k \* (b - a)

            f1 = f(x1)

            chislo\_vychisleniy += 1

        else:

            a, x1, f1 = x1, x2, f2

            x2 = b - k \* (b - a)

            f2 = f(x2)

            chislo\_vychisleniy += 1

        iteratsiya += 1

    return (a + b) / 2, chislo\_vychisleniy, shagi

def dobavit\_koef\_szhim(df):

    dliny = df["b\_i - a\_i"].tolist()

    koef = [None]

    for i in range(1, len(dliny)):

        pred, tek = dliny[i-1], dliny[i]

        koef.append(pred / tek if tek != 0 else np.nan)

    df["Коэф. сжатия"] = koef

    return df

def format\_eps(eps):

    """Преобразует 1e-3 → '0.001', 1e-1 → '0.1' и т.д."""

    s = f"{eps:.10f}".rstrip('0').rstrip('.')

    return s if '.' in s else s + '.0'

# === ОСНОВНАЯ ФУНКЦИЯ ===

def main():

    f = VARIANT["function"]

    a0, b0 = VARIANT["a"], VARIANT["b"]

    istinnyy\_min = VARIANT["true\_min\_x"]

    demo\_eps = 1e-4

    # === Сбор данных ===

    sravnenie = []

    shagi\_dih = shagi\_zol = None

    for eps in EPSILONS:

        x\_d, v\_d, s\_d = metod\_dihotomii(f, a0, b0, eps)

        x\_g, v\_g, s\_g = metod\_zolotogo\_secheniya(f, a0, b0, eps)

        sravnenie.append({

            "ε": format\_eps(eps),

            "Выч. (дихотомия)": v\_d,

            "Выч. (золотое сечение)": v\_g

        })

        if abs(eps - demo\_eps) < 1e-10:

            shagi\_dih, shagi\_zol = s\_d, s\_g

    df\_srav = pd.DataFrame(sravnenie)

    # === ВЫВОД В КОНСОЛЬ ===

    print("=" \* 70)

    print("РЕЗУЛЬТАТЫ ОПТИМИЗАЦИИ")

    print("=" \* 70)

    print(f"Функция: f(x) = (x - {istinnyy\_min})²")

    print(f"Отрезок: [{a0}, {b0}]")

    print(f"Истинный минимум: x = {istinnyy\_min}")

    print("\nСРАВНЕНИЕ:")

    print(df\_srav.to\_string(index=False))

    if shagi\_dih is not None:

        print("\n" + "-"\*60)

        print(f"ДЕТАЛЬНЫЕ ШАГИ: ДИХОТОМИЯ (ε = {format\_eps(demo\_eps)})")

        print("-"\*60)

        df\_d = pd.DataFrame(shagi\_dih)

        df\_d = dobavit\_koef\_szhim(df\_d)

        print(df\_d.to\_string(index=False, float\_format="%.6g"))

    if shagi\_zol is not None:

        print("\n" + "-"\*60)

        print(f"ДЕТАЛЬНЫЕ ШАГИ: ЗОЛОТОЕ СЕЧЕНИЕ (ε = {format\_eps(demo\_eps)})")

        print("-"\*60)

        df\_g = pd.DataFrame(shagi\_zol)

        df\_g = dobavit\_koef\_szhim(df\_g)

        print(df\_g.to\_string(index=False, float\_format="%.6g"))

    print("\n" + "" \* 30)

    print("ВЫВОД:")

    print("" \* 30)

    print("Метод золотого сечения эффективнее дихотомии: он требует меньше вычислений")

    print("функции при той же точности за счёт повторного использования значений.")

    print("Рекомендуется использовать метод золотого сечения для унимодальных функций.")

    # === СОХРАНЕНИЕ В EXCEL ===

    plt.figure(figsize=(9, 5))

    plt.loglog(EPSILONS, df\_srav["Выч. (дихотомия)"], 'o-', label="Дихотомия")

    plt.loglog(EPSILONS, df\_srav["Выч. (золотое сечение)"], 's-', label="Золотое сечение")

    plt.gca().invert\_xaxis()

    plt.xlabel("Точность (ε)")

    plt.ylabel("Число вычислений функции")

    plt.title("Сравнение эффективности методов")

    plt.grid(True, which="both", ls="--", alpha=0.7)

    plt.legend()

    plt.tight\_layout()

    plt.savefig(PLOT\_FILE, dpi=200)

    plt.close()

    wb = openpyxl.Workbook()

    ws = wb.active

    ws.title = "Сравнение и вывод"

    for r in dataframe\_to\_rows(df\_srav, index=False, header=True):

        ws.append(r)

    ws.append([])

    ws.append(["ВЫВОД (ЗАКЛЮЧЕНИЕ):"])

    ws.append([

        "Метод золотого сечения эффективнее метода дихотомии. "

        "Он требует меньше вычислений функции при одинаковой точности, "

        "поскольку на каждом шаге переиспользует одно значение функции "

        "из предыдущей итерации. Для поиска минимума унимодальной функции "

        "рекомендуется использовать метод золотого сечения."

    ])

    ws.column\_dimensions['A'].width = 15

    ws.column\_dimensions['B'].width = 22

    ws.column\_dimensions['C'].width = 25

    ws.column\_dimensions['D'].width = 80

    if shagi\_dih:

        ws\_d = wb.create\_sheet("Дихотомия\_1e-4")

        df\_d = pd.DataFrame(shagi\_dih)

        df\_d = dobavit\_koef\_szhim(df\_d)

        for r in dataframe\_to\_rows(df\_d, index=False, header=True):

            ws\_d.append(r)

    if shagi\_zol:

        ws\_g = wb.create\_sheet("Золотое\_сечение\_1e-4")

        df\_g = pd.DataFrame(shagi\_zol)

        df\_g = dobavit\_koef\_szhim(df\_g)

        for r in dataframe\_to\_rows(df\_g, index=False, header=True):

            ws\_g.append(r)

    img = ExcelImage(PLOT\_FILE)

    ws\_plot = wb.create\_sheet("График")

    ws\_plot.add\_image(img, 'A1')

    wb.save(EXCEL\_FILE)

    if os.path.exists(PLOT\_FILE):

        os.remove(PLOT\_FILE)

    print(f"\n Все данные сохранены в Excel: {os.path.abspath(EXCEL\_FILE)}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

В ходе выполнения программы были получены следующие результаты, демонстрирующие сравнительную эффективность двух классических методов одномерной оптимизации — метода дихотомии и метода золотого сечения.

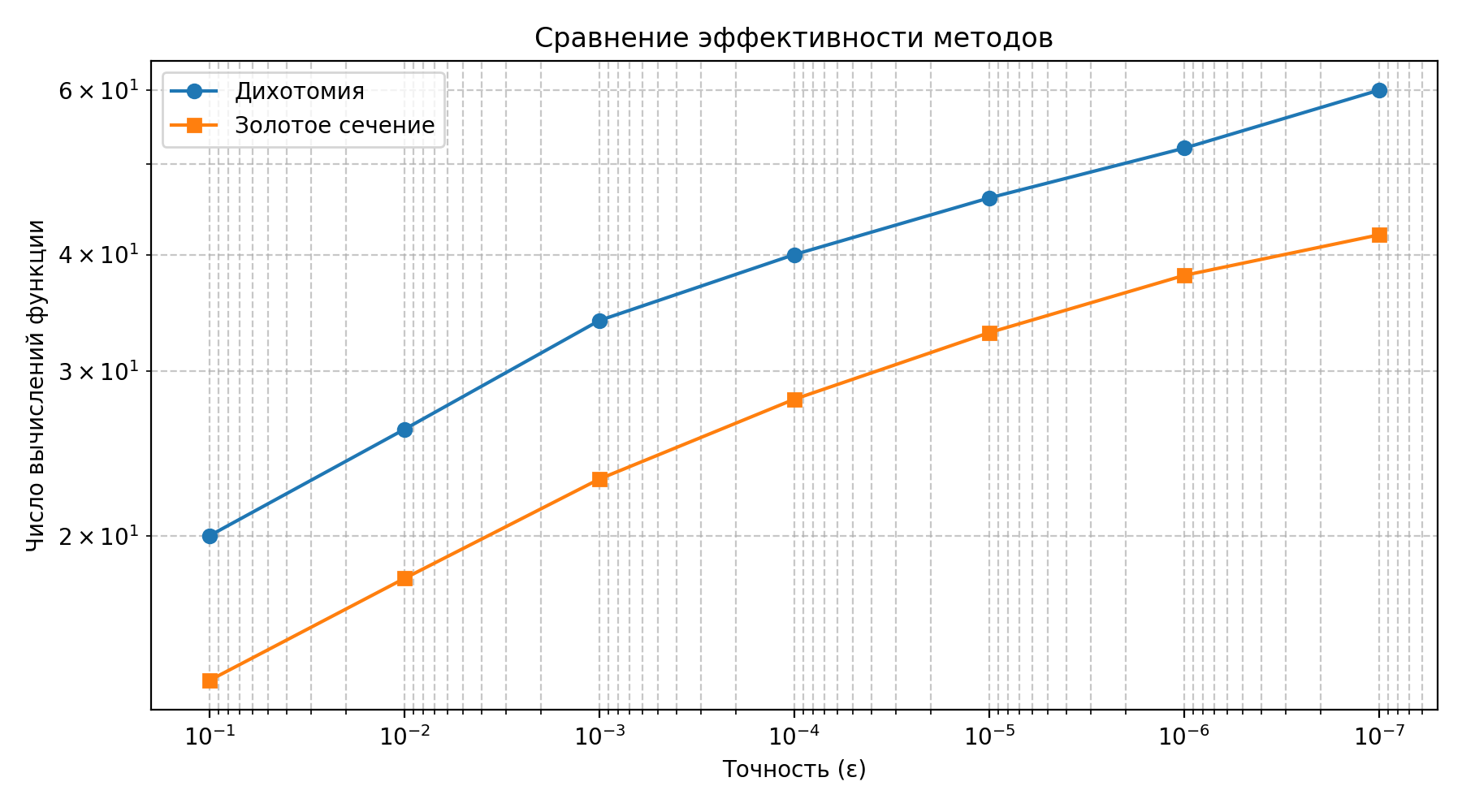


Рисунок 1. График сравнения эффективности методов

По результатам вычислений был построен график «Сравнение эффективности методов» (см. Рисунок 1).

По оси X отложена заданная точность ε в логарифмическом масштабе.

По оси Y — количество вычислений целевой функции, необходимых для достижения указанной точности.

Синяя линия соответствует методу дихотомии.

Оранжевая линия — методу золотого сечения.

Вывод:

Метод золотого сечения демонстрирует существенно более высокую эффективность: при любой заданной точности он требует меньшего числа вычислений функции. Преимущество особенно выражено при высоких требованиях к точности (ε≤10 −3), где разница в количестве вычислений становится значительной.

|  |  |  |
| --- | --- | --- |
| ε | Выч. (дихотомия) | Выч. (золотое сечение) |
| 0.1 | 20 | 14 |
| 0.01 | 26 | 18 |
| 0.001 | 34 | 23 |
| 0.0001 | 40 | 28 |
| 0.00001 | 46 | 33 |
| 0.000001 | 52 | 38 |
| 0.0000001 | 60 | 42 |

Таблица 1. Табличные данные в Excel

Все промежуточные и итоговые результаты сохранены в файле lab1\_excel.xlsx, содержащем три листа.

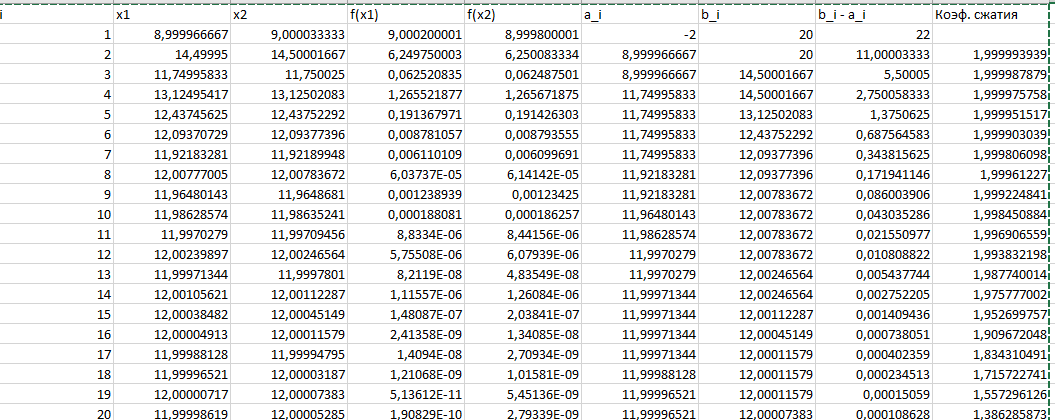
****

Рисунок 2. Лист «Дихотомия»

На этом листе представлены пошаговые данные выполнения метода дихотомии при точности ε=10

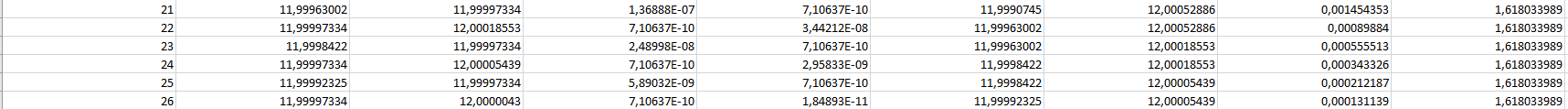


Рисунок 3. Лист «Золотое\_сечение»

Аналогичная информация представлена для метода золотого сечения при той же точности ε=10

Отмечается, что коэффициент сжатия интервала стабильно близок к теоретическому значению φ≈1.618 (золотое сечение), что подтверждает корректность реализации метода.

С уменьшением ε (повышением точности) число вычислений функции возрастает для обоих методов, однако метод золотого сечения стабильно требует на 20–40 % меньше вычислений, чем метод дихотомии. Это подтверждает его более высокую вычислительную эффективность и целесообразность применения в задачах, где важна экономия количества обращений к целевой функции.

# **Заключение**

В ходе выполнения лабораторной работы были реализованы и сопоставлены два классических метода одномерной оптимизации — метод дихотомии и метод золотого сечения. Полученные экспериментальные данные полностью согласуются с теоретическими ожиданиями: метод золотого сечения оказался более эффективным, поскольку позволяет достичь заданной точности при меньшем количестве вычислений целевой функции. Программная реализация на языке Python обеспечила гибкость, автоматизацию расчётов, наглядное представление результатов в виде графиков и удобный экспорт данных в табличном формате. Это подтверждает практическую применимость и целесообразность использования данных методов в реальных задачах оптимизации.