МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерні науки та технології

(найменування кафедри)

КУРСОВИЙ ПРОЕКТ (РОБОТА)

з Теорія Оптимальних Рішень

(назва дисципліни)

на тему:

Порівняння Ефективності Інтервальних та Точкових Методів Оптимізації для Функцій з Однією Змінною

Студента Бояра Єгора курсу Третього групи КНТ-811
спеціальності Системного аналізу
освітня програма (спеціалізація) _____

(ПРІЗВИЩЕ та ініціали)

Керівник _____

(посада, вчене звання, науковий ступінь, ПРІЗВИЩЕ та ініціали)

Кількість балів: _____
Члени комісії: _____ _____

(підпис) *(Ім'я та ПРІЗВИЩЕ)*

_____ _____

(підпис) *(Ім'я та ПРІЗВИЩЕ)*

_____ _____

(підпис) *(Ім'я та ПРІЗВИЩЕ)*

202___

1 Завдання

Порівняти ефективність різних методів оптимізацій на функціяїх однієї змінної.

2 Реферат

Курсовий проект: ПЗ: 47 с; 38 рис; 3 додатків; 0 таблиць; 11 джерел
Об'єкт дослідження - інтервальні методи, метод випадкового пошуку, метод ньютона, метод градієнтного спуску
Предмет дослідження - порівняння методів оптимізацій
Метод дослідження - аналітичний/статистичний.
Ключові слова - функції однієї змінної, оптимізація, Python

3 Зміст

4 Вступ

Оптимізація — це ключовий інструмент у математиці та інженерії, що дозволяє вирішувати різноманітні практичні завдання, від мінімізації витрат до оптимізації процесів. Ця курсова робота фокусується на порівнянні ефективності інтервальних та точкових методів оптимізації для функцій однієї змінної, використовуючи такі методи як оптимізація золотим відношенням, метод Фібоначчі, бісекція, метод Ньютона, градієнтний спуск та випадковий пошук для визначення їхньої придатності у різних сценаріях оптимізації.

Для реалізації проекту було обрано мову програмування Python через її можливості застосування принципів об'єктно-орієнтованого програмування, що було критично для створення класів оптимізації.

У ході роботи були використані такі бібліотеки:

- `sympy` — для символьних обчислень, в тому числі обчислення похідних, необхідних для методів Ньютона та градієнтного спуску.
- `numpy` — для високоефективних наукових обчислень.
- `pandas` — для обробки та аналізу даних, особливо зручно при роботі з табличними даними.
- `scipy` — використовується для різноманітних наукових обчислень, включаючи статистичний аналіз.
- `statsmodels` — для проведення статистичного моделювання та тестування гіпотез.

Важливо зазначити, що продуктивність методів Ньютона та градієнтного спуску залежить від якості реалізації функцій та обчислення похідних у бібліотеці `sympy`. Недоліки у цих аспектах можуть суттєво вплинути на швидкість виконання цих методів, як це було виявлено під час тестування.

Ця робота включає візуалізації результатів оптимізації, а також оцінку ефективності кожного методу за допомогою великого набору тестових функцій. Результати порівняльного аналізу допоможуть визначити, які методи оптимальні для конкретних видів завдань, і як можна покращити загальну продуктивність процесів оптимізації.

5 Основна частина

5.1 Теорія

5.1.1 Оптимізація методом золотого перетину

Оптимізація методом золотого перетину полягає у знаходженні мінімуму унімодальної функції в заданому інтервалі. Алгоритм починається з визначення інтервалу, в якому очікується мінімум. На кожній ітерації алгоритм обирає дві точки в межах інтервалу, використовуючи золотий перетин, обчислює значення функції в цих точках і вибирає новий інтервал в залежності від порівняння цих значень.

Якщо значення функції у точці x_1 менше, ніж у точці x_2 ($f(x_1) < f(x_2)$), то новий правий край інтервалу стає точкою x_2, а нова ліва точка обчислюється за формулою:

$$a' = x_1 = a + (1 - \phi)(b - a)$$

Де a і b - початковий і кінцевий край інтервалу відповідно, а ϕ - золотий перетин, що дорівнює $\frac{\sqrt{5}-1}{2}$.

Якщо значення функції у точці x_1 не менше, ніж у точці x_2 ($f(x_1) \geq f(x_2)$), то новий лівий край інтервалу стає точкою x_1, а нова права точка обчислюється за формулою:

$$b' = x_2 = a + \phi(b - a)$$

5.1.2 Оптимізація методом Фібоначчі

Оптимізація методом Фібоначчі - це метод для знаходження мінімуму функції в заданому інтервалі. Алгоритм використовує послідовність чисел Фібоначчі для визначення нових точок в межах інтервалу на кожній ітерації. За допомогою цих точок обчислюються значення функції, і інтервал звужується відповідно до результатів порівняння.

Якщо значення функції у точці x_1 менше, ніж у точці x_2 ($f(x_1) < f(x_2)$), то:

$$b = x_2$$

$$x_2 = x_1$$

$$x_1 = a + \frac{F_{n-2}}{F_n}(b - a)$$

Де a і b - початковий і кінцевий край інтервалу відповідно, а F_n - n-тий елемент послідовності чисел Фібоначчі.

У випадку, коли значення функції у точці x_1 не менше, ніж у точці x_2 ($f(x_1) \geq f(x_2)$), то:

$$a = x_1$$

$$x_1 = x_2$$

$$x_2 = a + \frac{F_{n-1}}{F_n}(b - a)$$

5.1.3 Метод бісекції

Метод бісекції - це числовий метод для знаходження кореня неперервної функції. У контексті пошуку мінімуму цей метод можна використовувати для знаходження мінімуму функції в заданому інтервалі. Алгоритм починається з оцінки значення функції в середині інтервалу, і зменшує інтервал до тих пір, поки не буде досягнуто необхідної точності.

Нехай c - середина інтервалу $[a, b]$, тоді:

$$c = \frac{a + b}{2}$$

Якщо $f(c - \delta) < f(c + \delta)$, то:

$$b = c$$

Інакше, якщо $f(c - \delta) \geq f(c + \delta)$, то:

$$a = c$$

де δ - відстань від середини інтервалу, на якій обчислюється значення функції.

5.1.4 Метод Ньютона

Метод Ньютона - це ітеративний числовий метод для знаходження кореня функції, використовуючи її похідні. У контексті пошуку мінімуму функції цей метод можна використовувати для знаходження локального мінімуму.

Нехай $f(x)$ - функція, а $f'(x)$ та $f''(x)$ її перша та друга похідні відповідно. Починаючи з початкового наближення x_k, на кожній ітерації метод обчислює нове наближення x_{k+1} за формулою:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Алгоритм продовжує ітерації досягнення необхідної точності або до досягнення максимальної кількості ітерацій.

5.1.5 Метод градієнта

Метод градієнта (градієнтний спуск) - це ітеративний метод оптимізації функції, який використовує градієнт для знаходження локального мінімуму.

Нехай $f(x)$ - функція, а $\nabla f(x)$ - її градієнт. Починаючи з початкового наближення x_k, на кожній ітерації метод оновлює наближення за формулою:

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

де α - крок зміни, який може бути сталим або змінюватися на кожній ітерації. Метод продовжує ітерації до досягнення необхідної точності або до досягнення максимальної кількості ітерацій.

5.1.6 Метод випадкового пошуку

Метод випадкового пошуку - це евристичний метод оптимізації функції, який полягає у випадковому виборі точок у просторі параметрів функції з метою знаходження мінімуму або максимуму.

Нехай $f(x)$ - функція, яку ми хочемо оптимізувати, а x_k - поточне наближення до оптимального значення. На кожній ітерації метод обчислює нове наближення x_{k+1}, зміщуючи поточне значення на випадкову величину.

Алгоритм може бути представлений наступним чином:

1. Почати з початкового наближення x_k.
2. Обчислити значення функції $f(x_k)$.
3. Випадково змінити значення x_k.
4. Порівняти значення функції для нового x_k з попереднім значенням.
5. Якщо нове значення є кращим, прийняти його як поточне x_k.
6. Повторити кроки 2-5 до досягнення критерію зупинки (наприклад, до досягнення певної кількості ітерацій або до досягнення заданої точності).

Цей процес триває до досягнення заданої точності або до досягнення максимальної кількості ітерацій. Результатом є значення, приблизно оптимальне для функції, його функційне значення та кількість проведених ітерацій.

5.2 Імплементація

5.2.1 Огляд реалізації

Проект складається з кількох файлів Python, кожен з яких містить класи та методи для різних технік оптимізації, аналізу даних та побудови графіків. Нижче наведено короткий огляд кожного файлу:

- `IntervalOptimizationMethods.py`: Цей файл містить класи та методи для методів оптимізації інтервалу, таких як Оптимізація золотим відношенням, Оптимізація числами Фібоначчі та Оптимізація методом бісекції.
- `PointOptimizationMethods.py`: Тут ви знайдете класи та методи для точкових методів оптимізації, зокрема Метод Ньютона, Градієнтний спуск та Випадковий пошук.
- `Graph_plotting.py`: Цей файл включає функції для читання та обробки даних з CSV-файлу, обчислення середніх часів та побудови стовпчикових діаграм, гістограм та бокс-плотів.
- `multi_optimization.py`: Цей скрипт оркеструє процес оптимізації за допомогою як точкових, так і інтервальних методів, зберігаючи результати у CSV-файлі. Також результати агрегуються та зберігаються у іншому CSV-файлі.
- `Data_analysis.py`: Містить функції для завантаження та передобробки даних, виконання статистичних тестів, бутстрепу та побудови гістограм та бокс-плотів.

5.2.2 Блок-Схеми

На Рис. 1 ви можете побачити блок-схему, яка пояснює мою імплементацію методу Ньютона.

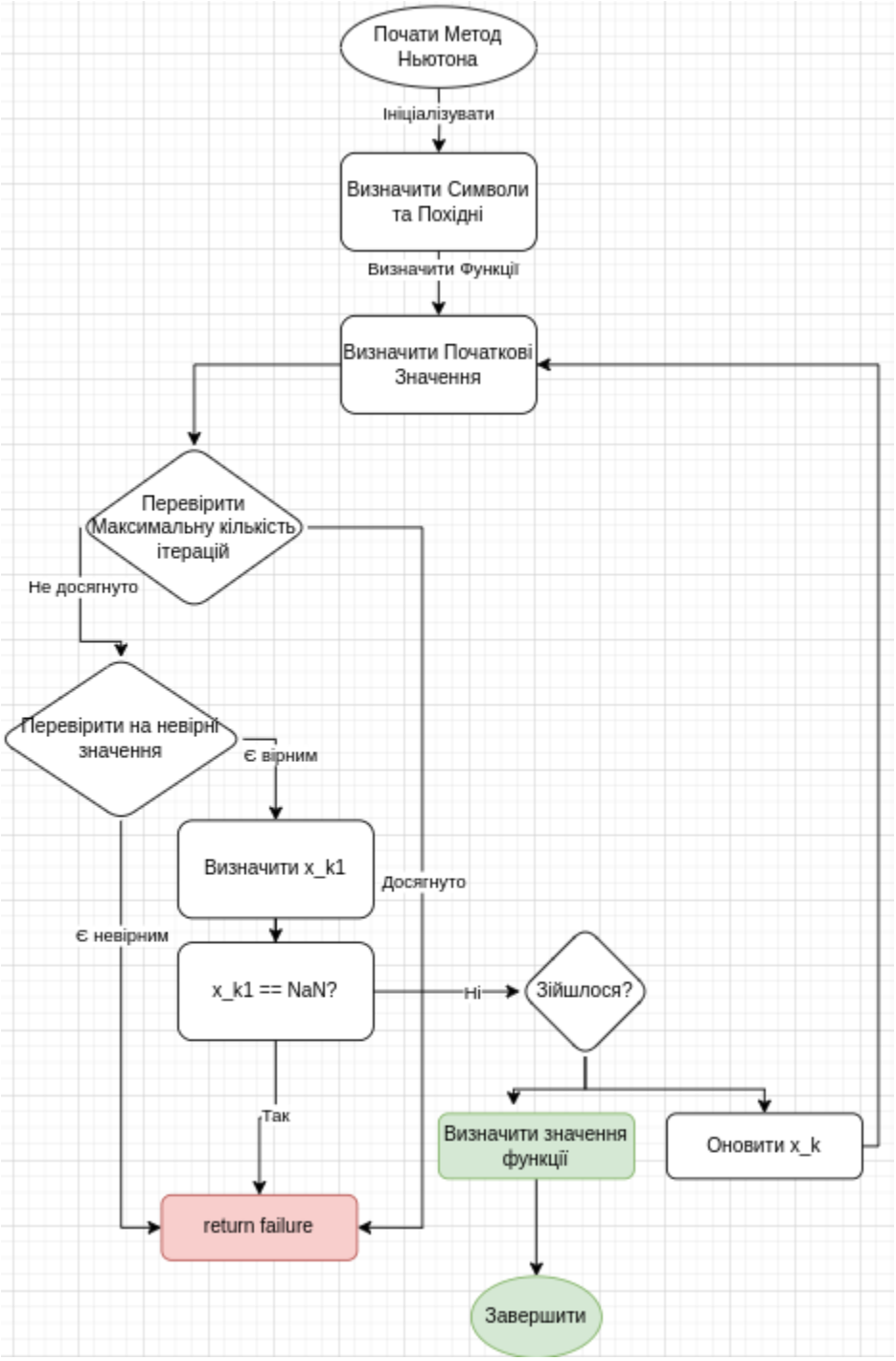

Рис. 1: Ньютон Блок-Схема

На Рис. 2 ви можете побачити блок-схему, яка пояснює мою імплементацію методу градієнту.

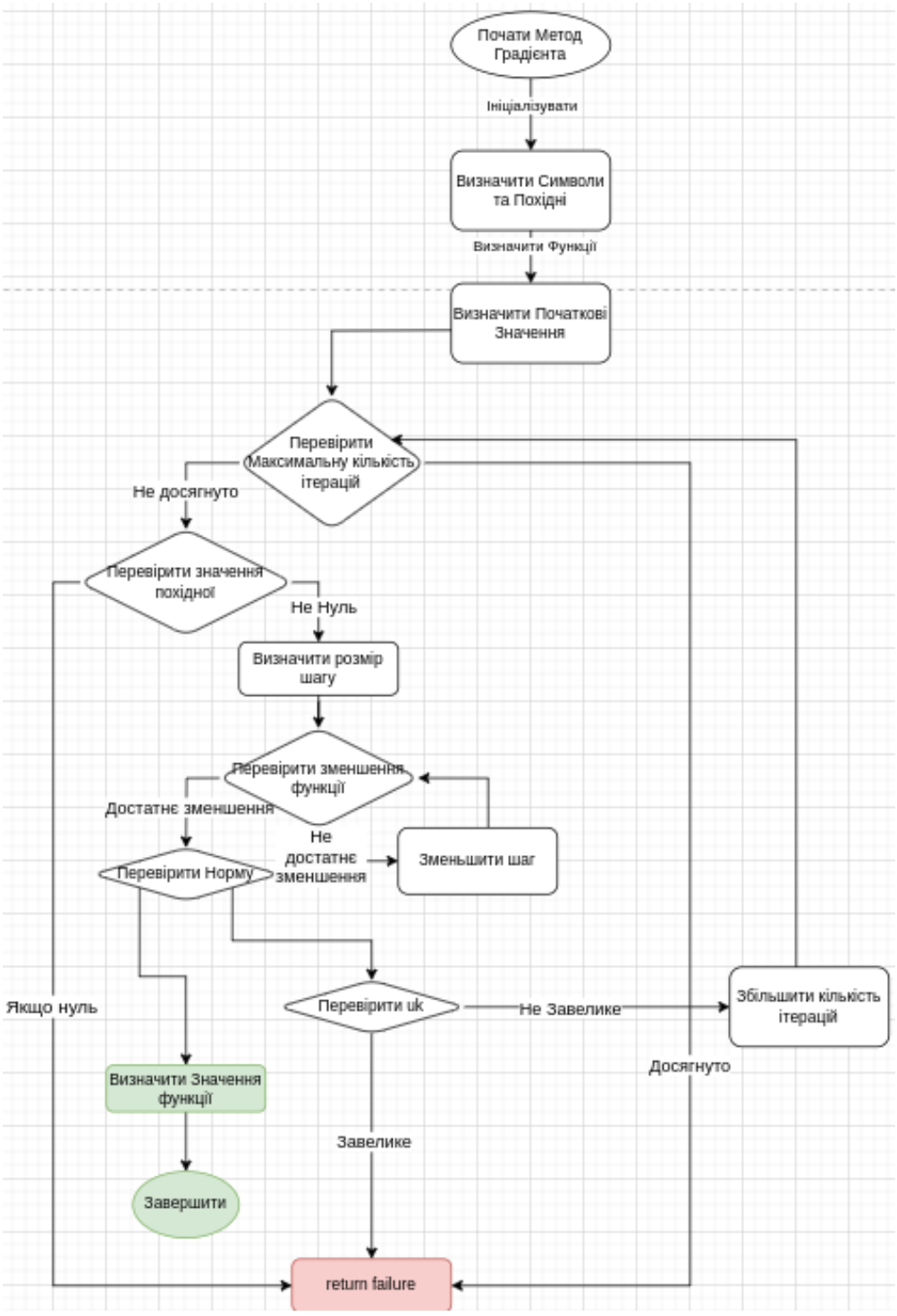

Рис. 2: Градієнт Блок-Схема

На Рис. 1 ви можете побачити блок-схему, яка пояснює мою імплементацію методу випадкового пошуку.

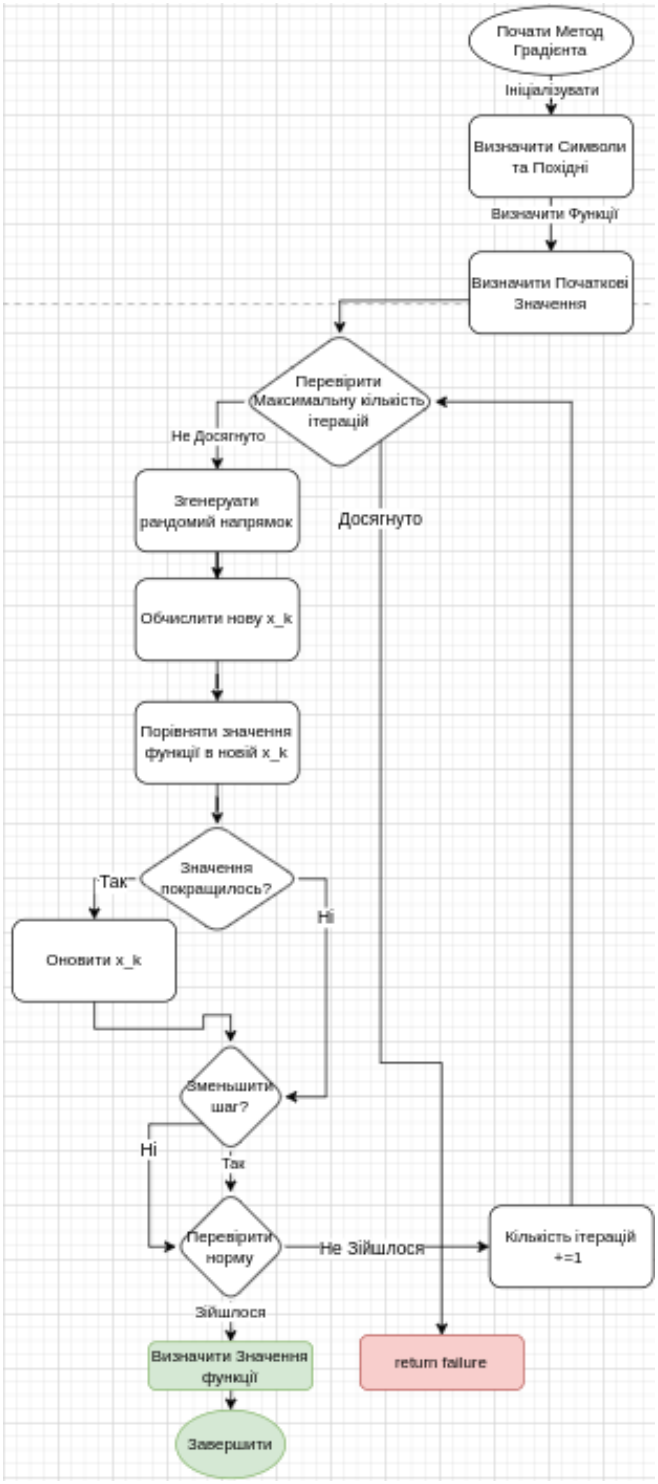

Рис. 3: Рандом Блок-Схема

Нарешті, Рис. 4 представляє блок-схему, яка показує імплементацію всіх "інтервальних"методів.

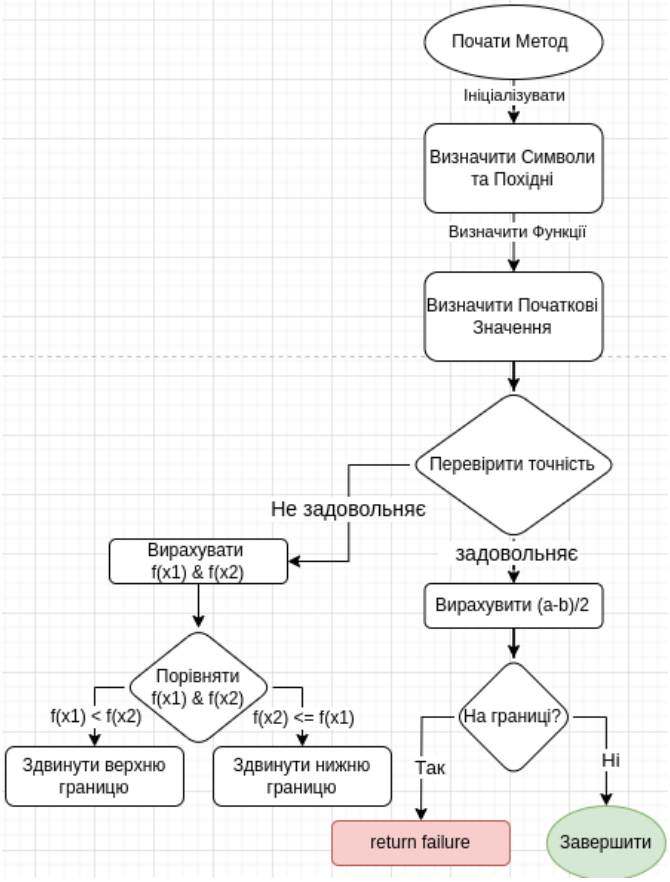

Рис. 4: Інтревальні методи

5.3 Створення даних

Для порівняння методів оптимізацій був створен лист функцій якій можна побачити на Рис 5.

```
def define_functions():
    x = sp.symbols('x')
    return {
        'Quadratic 1': x ** 2 - 8 * x + 8,
        'Quadratic 2': x ** 2 + 4 * x + 4,
        'Quadratic 3': 3 * x ** 2 - 3 * x + 1,
        'Quadratic 4': x ** 2 + 2 * x - 1,
        'Quadratic 5': 0.5 * x ** 2 - 5 * x + 12,
        'Cubic 1': x ** 3 - 3 * x - 1,
        'Cubic 2': -x ** 3 + 6 * x ** 2 - 9 * x + 4,
        'Cubic 3': 2 * x ** 3 - 6 * x ** 2 + 4 * x,
        'Cubic 4': x ** 3 + x ** 2 - 4 * x - 4,
        'Cubic 5': 4 * x ** 3 - 12 * x ** 2 + 9 * x - 2,
        'Quartic 1': x ** 4 - 4 * x ** 3 + 6 * x ** 2 - 4 * x + 1,
        'Quartic 2': -2 * x ** 4 + 8 * x ** 3 - 12 * x ** 2 + 8 * x - 2,
        'Quartic 3': 0.5 * x ** 4 - x ** 3 - 3.5 * x ** 2 + 2 * x + 10,
        'Quartic 4': x ** 4 + 2 * x ** 3 - 13 * x ** 2 + 14 * x - 24,
        'Quartic 5': 3 * x ** 4 - 6 * x ** 3 + 3 * x ** 2 - 6 * x + 2,
        'Exponential 1': sp.exp(x ** 2),
        'Exponential 2': 2 ** x - x ** 2,
        'Exponential 3': 2 ** x * 9 * x ** 2,
        'Logarithmic 1': sp.log(sp.exp(x ** 2) + x + 1),
        'Logarithmic 2': x * sp.log(x) - x ** 0.5,
        'Logarithmic 3': x * sp.log(x) + x ** 2
    }
```

Рис. 5: Функції

Потім, в циклі на Рис 6., кожен метод мав оптимізувати ці функції з заданою точністю.

```
def perform_optimizations(test_functions, initial_intervals, initial_points, precisions, max_iterations):
    interval_results, point_results = {}, {}
    for precision in precisions:
        for name, func in test_functions.items():
            interval_results.setdefault(name, {})
            point_results.setdefault(name, {})
            run_interval_optimizations(func, interval_results[name], initial_intervals, precision)
            run_point_optimizations(func, point_results[name], initial_points, precision, max_iterations)
    return interval_results, point_results
```

Рис. 6: Цикл оптимізацій

В кінці, всі результати заипсуються в файл формату CSV. Скріпт який це робить можна побачити на Рис. 7

```python
def save_optimization_results(all_interval_results, all_point_results, filename='optimization_results.csv'):
    with open(filename, 'w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(
            ['Optimization Type', 'Function Name', 'Parameter', 'Method', 'Optimal x', 'Function Value', 'Iterations',
             'Result', 'Time', 'Precision'])
        for precision, interval_results in all_interval_results.items():
            for func_name, results_by_name in interval_results.items():
                for param, methods in results_by_name.items():
                    for method, results in methods.items():
                        writer.writerow(['Interval', func_name, param, method] + list(results) + [precision])
        for precision, point_results in all_point_results.items():
            for func_name, results_by_name in point_results.items():
                for param, methods in results_by_name.items():
                    for method, results in methods.items():
                        writer.writerow(['Point', func_name, param, method] + list(results) + [precision])
```

Рис. 7: Запис даних

5.4 Аналіз даних

Після отримання даних було проведено їх аналіз.

5.4.1 Візуалізація даних

Візуалізація є ключовим компонентом нашого дослідження, оскільки вона дозволяє наглядно порівняти результати різних методів оптимізації за різними типами математичних функцій та рівнями точності. Завдяки візуалізаціям легко ідентифікувати тенденції, варіабельність та ефективність кожного методу. У проекті використано такі типи візуалізацій:

- Стовпчикові діаграми: демонструють середній час виконання оптимізаційних методів, розподілений за типами функцій та рівнями точності.
- Гістограми: відображають розподіл часу виконання оптимізації для всіх методів.
- Боксплоти: використовуються для порівняння розмаху та центральної тенденції часів виконання між різними методами.
- Точкові діаграми: були використані щоб визначити кореляцію між кількістю ітерацій та точністю оптимізації

Стовпчикові діаграми

На Рис. 8 представлена діаграма, що показує середні часи виконання для всіх методів оптимізацій, агреговані за типами функцій.

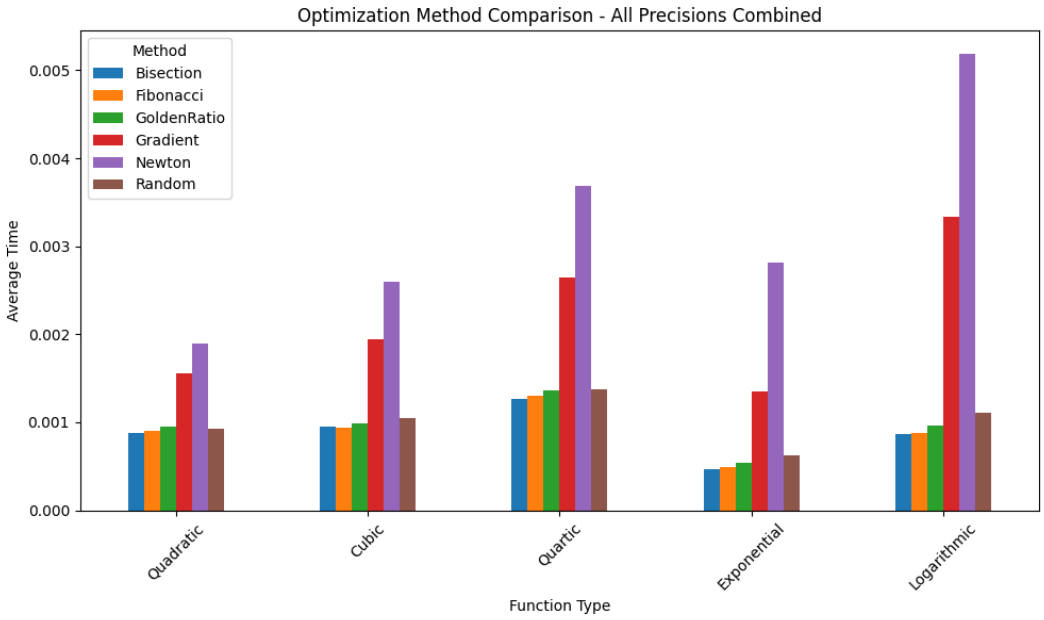

Рис. 8: Середній час — усі точності

Також було створено діаграми для кожного рівня точності, які показують, як час оптимізації змінюється зі збільшенням вимог до точності.

Гістограми

Рис. 9 демонструє розподіл часів виконання методів оптимізації за усіма рівнями точності. З такої гістограми можна оцінити стабільність методу та його варіативність, що корисно для статистичного аналізу.

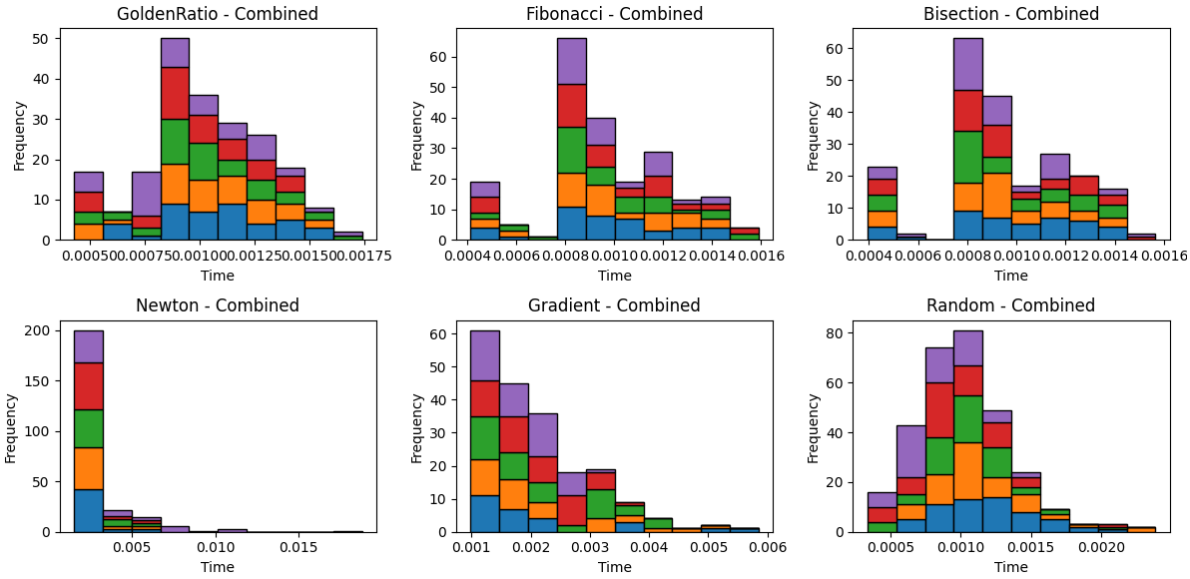

Рис. 9: Розподіл часів — усі точності

Гістограми для кожного рівня точності демонструють подібні тенденції: методи Ньютона та градієнтного спуску зазвичай мають скошені розподіли.

Боксплоти

На Рис. 10 представлено боксплот для різних методів оптимізації за усіма рівнями точності. З цього графіку можна побачити, що методи нульового порядку мають меншу дисперсію, а їх середній час виконання приблизно однаковий. У той же час, градієнтний метод і метод Ньютона показують схожий час виконання і дисперсію між собою, але їх час виконання та дисперсія виглядають більшим, ніж у методів нульового порядку.

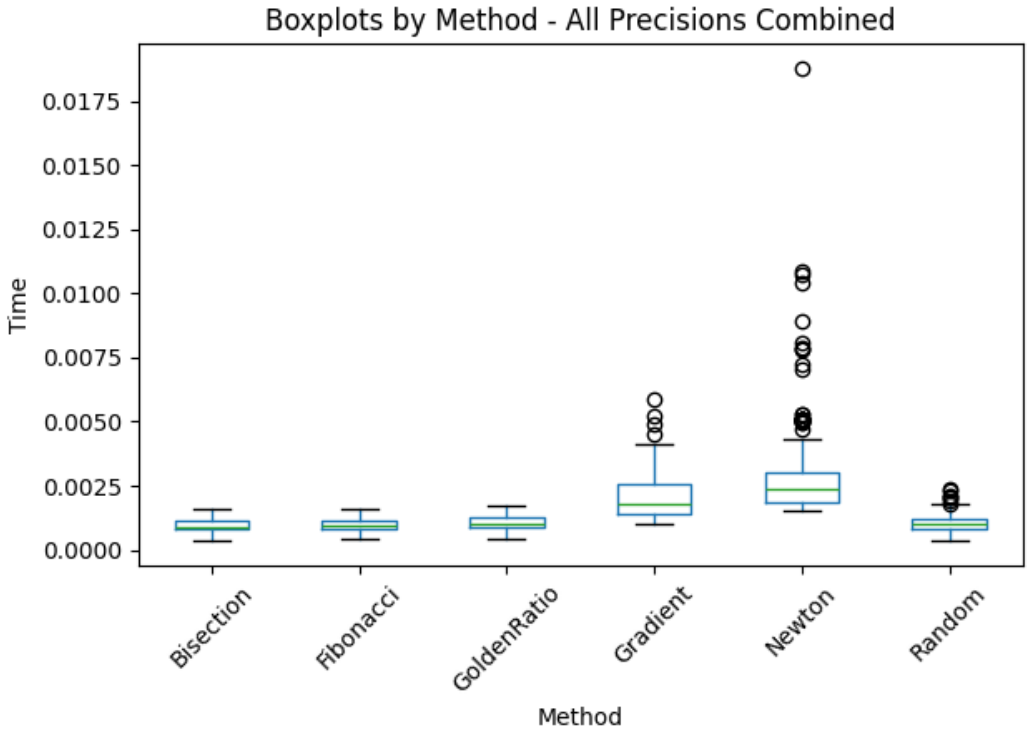

Рис. 10: Боксплот — усі точності

Точкові діаграми

На точкових діаграмах (див. Рис. 33-38 в додатках) показано час виконання оптимізації залежно від кількості ітерацій. Якщо в інтервальних методах та методу нульвого порядку ця залежність доволі очевидна, то в інших методах ми не можемо сказати те саме. Це може свідчити про те, що в точкових методах оптимізації точність залежить від більш ніж одного фактора.

5.4.2 Гіпотеза

На основі спостережень можна висунути гіпотезу, що середній час оптимізації у методів першого та другого порядку є меншим, ніж у методів нульового порядку. Також можна гіпотезувати, що в інтервальних методах та матоду нульового порядку оптимізації кількість ітерацій є ключовим фактором, який впливає на точність, тоді як у точкових методах, ймовірно, не спостерігається така ж картина.

5.4.3 Тест Гіпотез

Тест даних

Спочатку було проведено аналіз даних, використовуючи формальні тести для

перевірки на нормальність та гомогенність дисперсій. Дані були перевірені на нормальність за допомогою тесту Шапіро-Вілка для кожного методу та на гомогенність дисперсій за допомогою тесту Бартлетта. На Рис. 11 приведен скріпт котрий був використан

```
def perform_shapiro_tests(data, unique_methods):
    for i, method_data in enumerate(data, 1):
        stat, p_value = stats.shapiro(method_data)
        print(f"Shapiro-Wilk test for method {unique_methods[i - 1]}: Statistic: {stat}, p-value: {p_value}")
        print(
            "Data is normally distributed (fail to reject H0)\n" if p_value > 0.05 else "Data is not normally distributed (reject H0)\n")

1 usage  Yehor Boiar
def check_variances_homogeneity(data):
    stat, p_value = stats.bartlett(*data)
    print(f"Bartlett's test for homogeneity of variances: Statistic: {stat}, p-value: {p_value}")
    print(
        "Variances are homogeneous (fail to reject H0)" if p_value > 0.05 else "Variances are not homogeneous (reject H0)")
```

Рис. 11: Тест скріпт

Отримані наступні результати:

- **Тест Шапіро-Вілка:**
  - Метод Бісекції: статистика = 0.9392, p-значення = 0.0244. Дані не є нормально розподіленими (нульову гіпотезу відхилено).
  - Метод Фібоначчі: статистика = 0.9474, p-значення = 0.0520. Дані є нормально розподіленими (нульову гіпотезу не відхилено).
  - Метод Золотого перетину: статистика = 0.9602, p-значення = 0.1504. Дані є нормально розподіленими (нульову гіпотезу не відхилено).
  - Метод Градієнтного спуску: статистика = 0.5526, p-значення = 7.121e-11. Дані не є нормально розподіленими (нульову гіпотезу відхилено).
  - Метод Ньютона: статистика = 0.7325, p-значення = 5.233e-08. Дані не є нормально розподіленими (нульову гіпотезу відхилено).
  - Метод Випадкового пошуку: статистика = 0.9191, p-значення = 0.0008. Дані не є нормально розподіленими (нульову гіпотезу відхилено).
- **Тест Бартлетта на гомогенність дисперсій:** статистика = 551.576, p-значення = 5.841e-117. Дисперсії не є гомогенними (нульову гіпотезу відхилено).

(Вивод в консоль скрипту якій це робив можна побачити в додатках)

Статистичний тест гіпотези

Оскільки наші дані не відповідали припущенням ANOVA про нормальність та гомогенність дисперсій, було застосовано непараметричний тест Крускала-Валліса, за яким послідував тест Games-Howell для детального порівняння груп. Результати цих тестів представлено на Рис. 12.

```
Kruskal-Wallis Test: H-statistic = 1138.8695021479248, p-value = 5.10861374591917e-244
Games-Howell Post-Hoc Test Results:
         group1       group2  meandiff   p-adj   lower   upper  reject
0     Bisection    Fibonacci    0.0000  1.0000 -0.0034  0.0034   False
1     Bisection  GoldenRatio    0.0001  1.0000 -0.0033  0.0035   False
2     Bisection     Gradient    0.0109  0.0000  0.0075  0.0143    True
3     Bisection       Newton    0.0017  0.6940 -0.0017  0.0052   False
4     Bisection       Random    0.0002  1.0000 -0.0033  0.0036   False
5     Fibonacci  GoldenRatio    0.0001  1.0000 -0.0033  0.0035   False
6     Fibonacci     Gradient    0.0108  0.0000  0.0074  0.0143    True
7     Fibonacci       Newton    0.0017  0.7014 -0.0017  0.0051   False
8     Fibonacci       Random    0.0001  1.0000 -0.0033  0.0036   False
9   GoldenRatio     Gradient    0.0108  0.0000  0.0074  0.0142    True
10  GoldenRatio       Newton    0.0016  0.7399 -0.0018  0.0051   False
11  GoldenRatio       Random    0.0001  1.0000 -0.0033  0.0035   False
12     Gradient       Newton   -0.0091  0.0000 -0.0125 -0.0057    True
13     Gradient       Random   -0.0107  0.0000 -0.0141 -0.0073    True
14       Newton       Random   -0.0016  0.7717 -0.0050  0.0018   False
```

Рис. 12: Стат. Тести

Висновки з результатів тесту Games-Howell:

Метод градієнтного спуску демонструє значні статистичні відмінності в часі виконання порівняно з методами Бісекції, Фібоначчі, Золотого перетину, Ньютона та Випадкового пошуку, що підтверджує його низку ефективність. Водночас, значні відмінності відсутні між більшістю інших методів, що може свідчити про їх схожість у певних умовах виконання.

Кореляційний аналіз

Також був проведений кореляційний аналіз для визначення залежності між кількістю ітерацій та точністю оптимізації. Оскільки дані не відповідали нормальному розподілу, для обчислення кореляції використовувався тест Спірмена. В додатках буде наведений скрипт, який був використаний. Отримані результати кореляційного аналізу:

- Метод Золотого перетину: Кореляція = -0.9820
- Метод Фібоначчі: Кореляція = -0.9820

- Метод Бісекції: Кореляція = -0.9820
- Метод Ньютона: Кореляція = -0.1308
- Метод Градієнтного спуску: Кореляція = 0.0372
- Метод Випадкового пошуку: Кореляція = -0.8257

Висновки з результатів кореляційного аналізу

Кореляційний аналіз показує сильну негативну кореляцію між кількістю ітерацій та точністю для методів Золотого перетину, Фібоначчі та Бісекції, що свідчить про те, що зі збільшенням кількості ітерацій точність оптимізації збільшується. Для методу Ньютона та градієнтного спуску кореляція значно слабша, що підтверджує нашу гіпотезу про залежність точності в цих методах від інших факторів. Метод випадкового пошуку також показав значну негативну кореляцію, що свідчить про те, що кількість ітерацій впливає на точність, проте з меншою ефективністю порівняно з інтервальними методами.

6 Висновки

Цей курсовий проект досліджував ефективність різних методів оптимізації для функцій з однією змінною. Завдяки аналізу виконання кожного методу та статистичному порівнянню за допомогою тесту Крускала-Валліса та пост-хок аналізу Games-Howell ми зробили наступні ключові висновки:

- **Низька швидкість градієнтного методу:** На відміну від очікувань, градієнтний метод показав найнижчу швидкість серед усіх тестованих методів. Це ставить під сумнів його придатність для задач, де важлива швидкість обчислень.

- **Підозра щодо методу Ньютона:** Метод Ньютона, хоч і не показав найнижчі швидкості в статистичних тестах, базуючись на графіках, цей метод може бути потенційно повільнішим за градієнтний метод.

- **Складність виконання методу Ньютона та градієнтного методу:** Метод Ньютона та метод градієнта використовують похідні. Їх обчислення може бути проблематичним, адже ліміту в заданій точці може не існувати, або ліміт може досягати нескінченності. Також, в методі Ньютона, ми можемо стикнутися з діленням на нуль, що ще додає проблематичності.

- **Надійність інших методів:** Відсутність значущих статистичних відмінностей між іншими методами свідчить про їх надійність та схожість у ефективності при оптимізації різних типів функцій.

- **Метод випадкового пошуку:** Цей метод показав значну негативну кореляцію між кількістю ітерацій та точністю, що свідчить про його залежність від кількості ітерацій. Проте метод випадкового пошуку є менш ефективним порівняно з інтервальними методами, хоча він може бути корисним у випадках, коли інші методи не працюють.

З урахуванням виявлених недоліків у швидкості методів градієнта та Ньютона, рекомендуємо використовувати для оптимізації функцій інтервальні методи або методи нульового порядку. Ці методи не тільки демонструють кращу швидкість, але й менше залежать від складності обчислень, яка може суттєво варіюватися залежно від властивостей функції.

7 Список використаної літератури

1. Burden, R. L., Faires, J. D. *Numerical Analysis.* — Brooks/Cole, 2010. — 888 p.

2. Nocedal, J., Wright, S. J. *Numerical Optimization.* — Springer, 2006. — 664 p.

3. Kiefer, J., Wolfowitz, J. *Stochastic Estimation of the Maximum of a Regression Function.* — The Annals of Mathematical Statistics, 1952. — 23(3): 462-466.

4. Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. *Numerical Recipes: The Art of Scientific Computing.* — Cambridge University Press, 2007. — 1235 p.

5. Sympy Development Team. *SymPy: Python library for symbolic mathematics.* — URL: https://www.sympy.org/

6. Oliphant, T. E. *A Guide to NumPy.* — USA: Trelgol Publishing, 2006. — 278 p.

7. McKinney, W. *Data Structures for Statistical Computing in Python.* — Proceedings of the 9th Python in Science Conference, 2010.

8. Virtanen, P. et al. *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.* — Nature Methods, 2020.

9. Seabold, S., Perktold, J. *Statsmodels: Econometric and Statistical Modeling with Python.* — Proceedings of the 9th Python in Science Conference, 2010.

10. Hunter, J. D. *Matplotlib: A 2D Graphics Environment.* — Computing in Science & Engineering, 2007. — 9(3): 90-95.

11. Waskom, M. et al. *seaborn: Statistical Data Visualization.* — Journal of Open Source Software, 2021. — 6(60), 3021.

8 Додатки

8.1 Код

Весь код ви зможете побачити на цій сторінці GitHub

8.2 Графіки

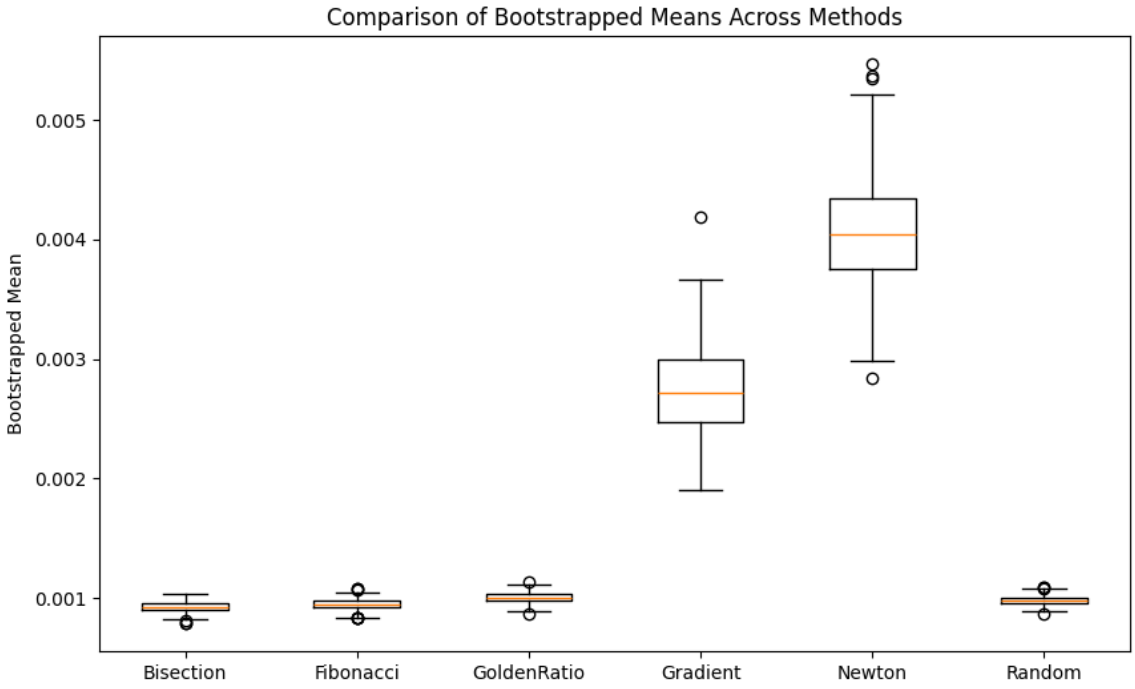

Рис. 13: Enter Caption

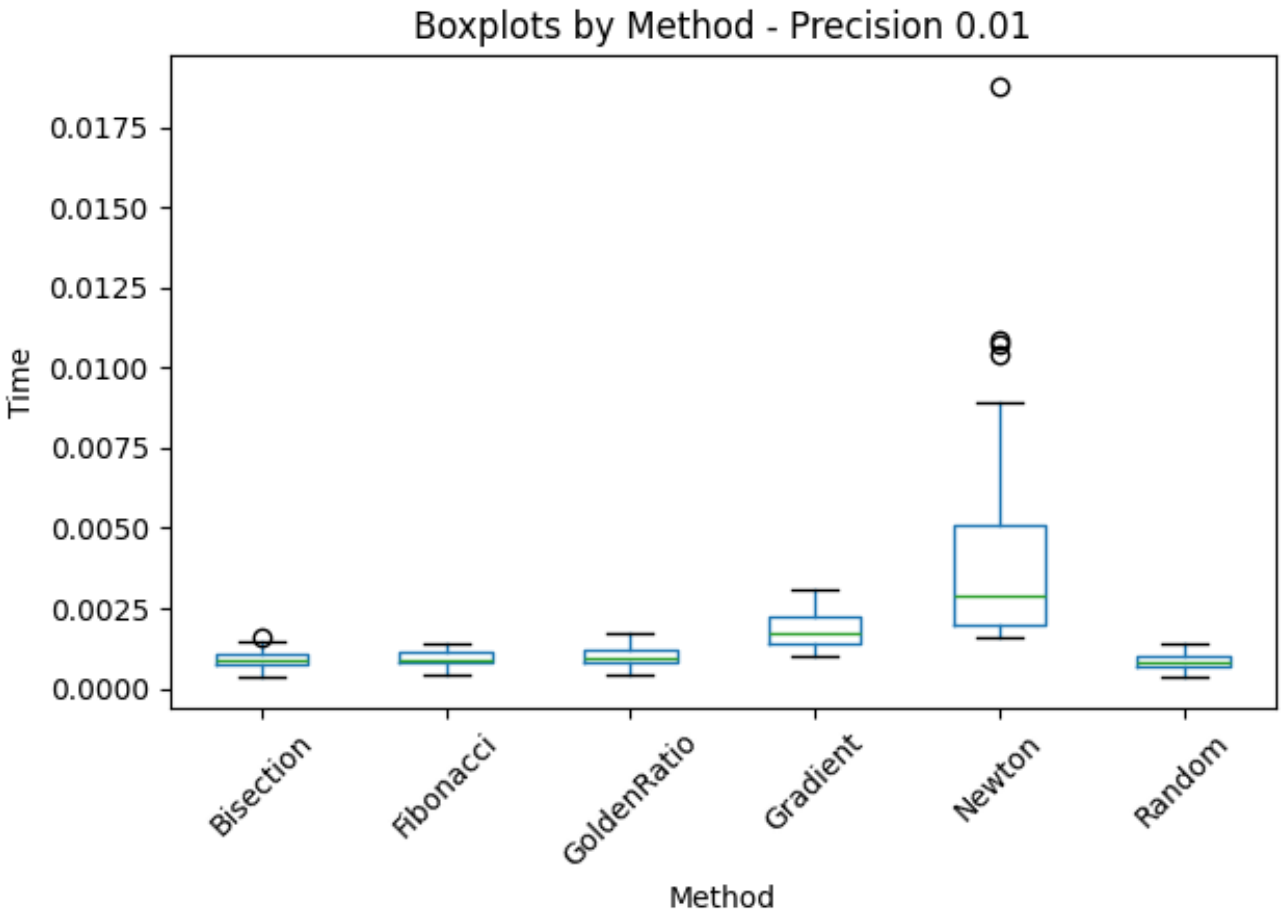

Рис. 14: Enter Caption

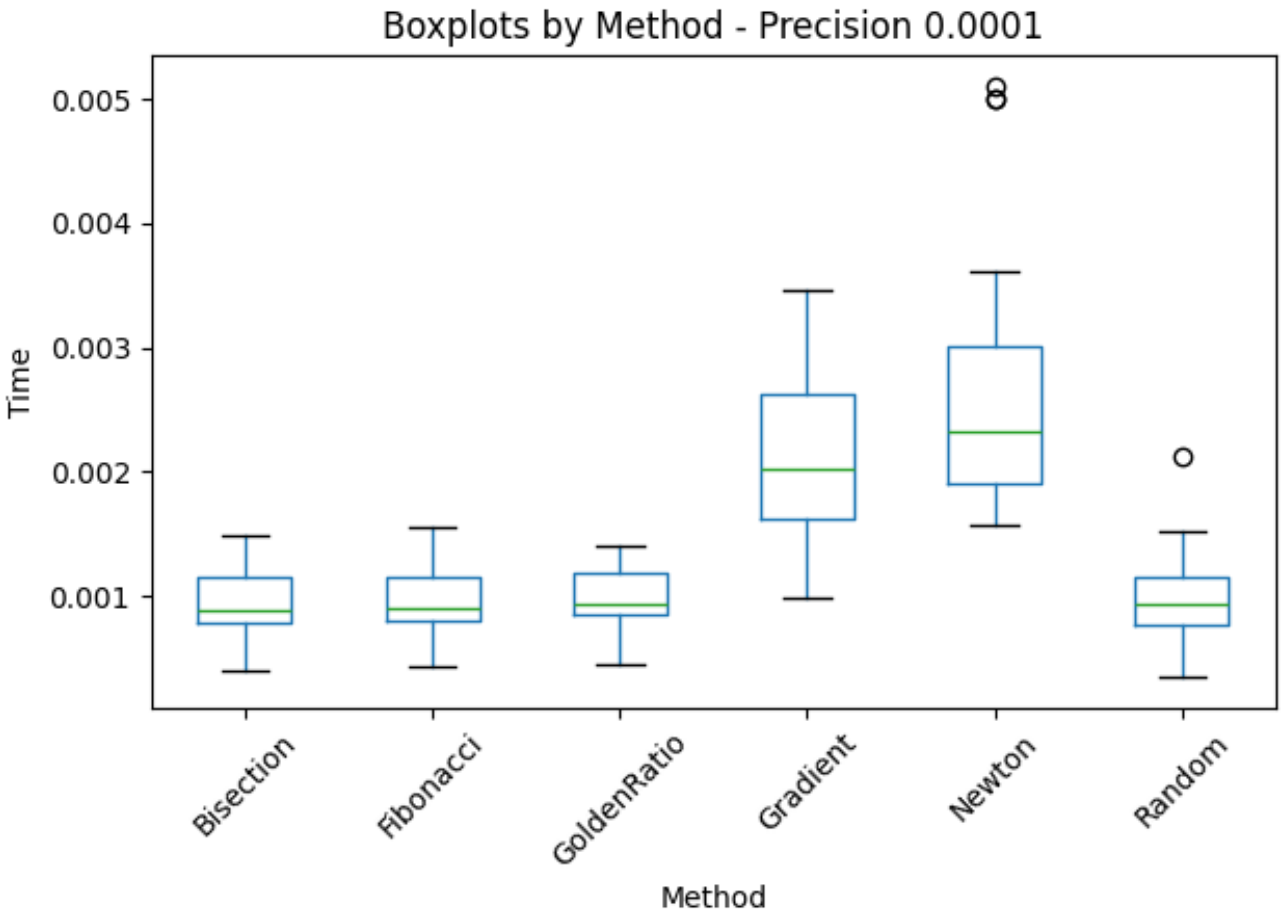

Рис. 15: Enter Caption

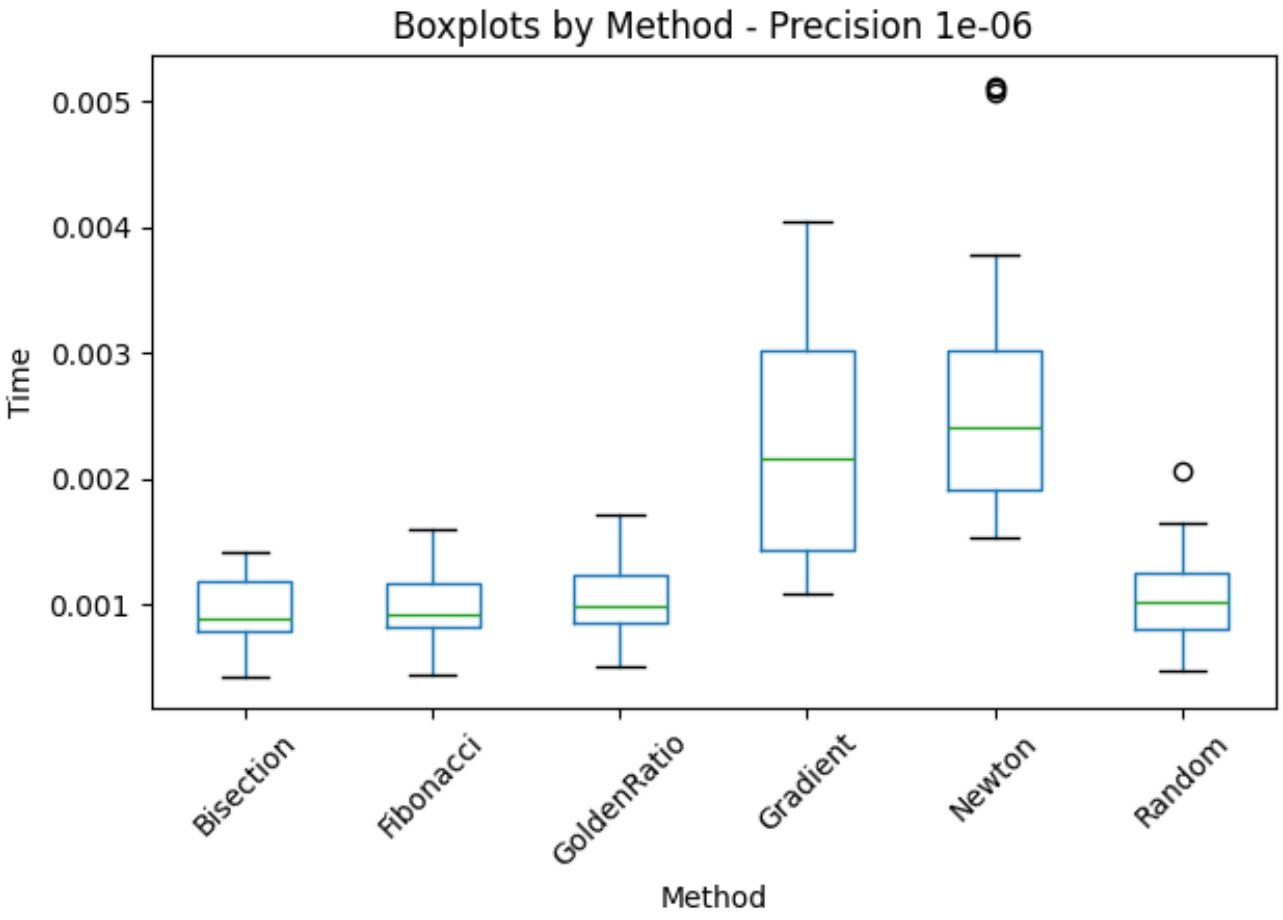

Рис. 16: Enter Caption

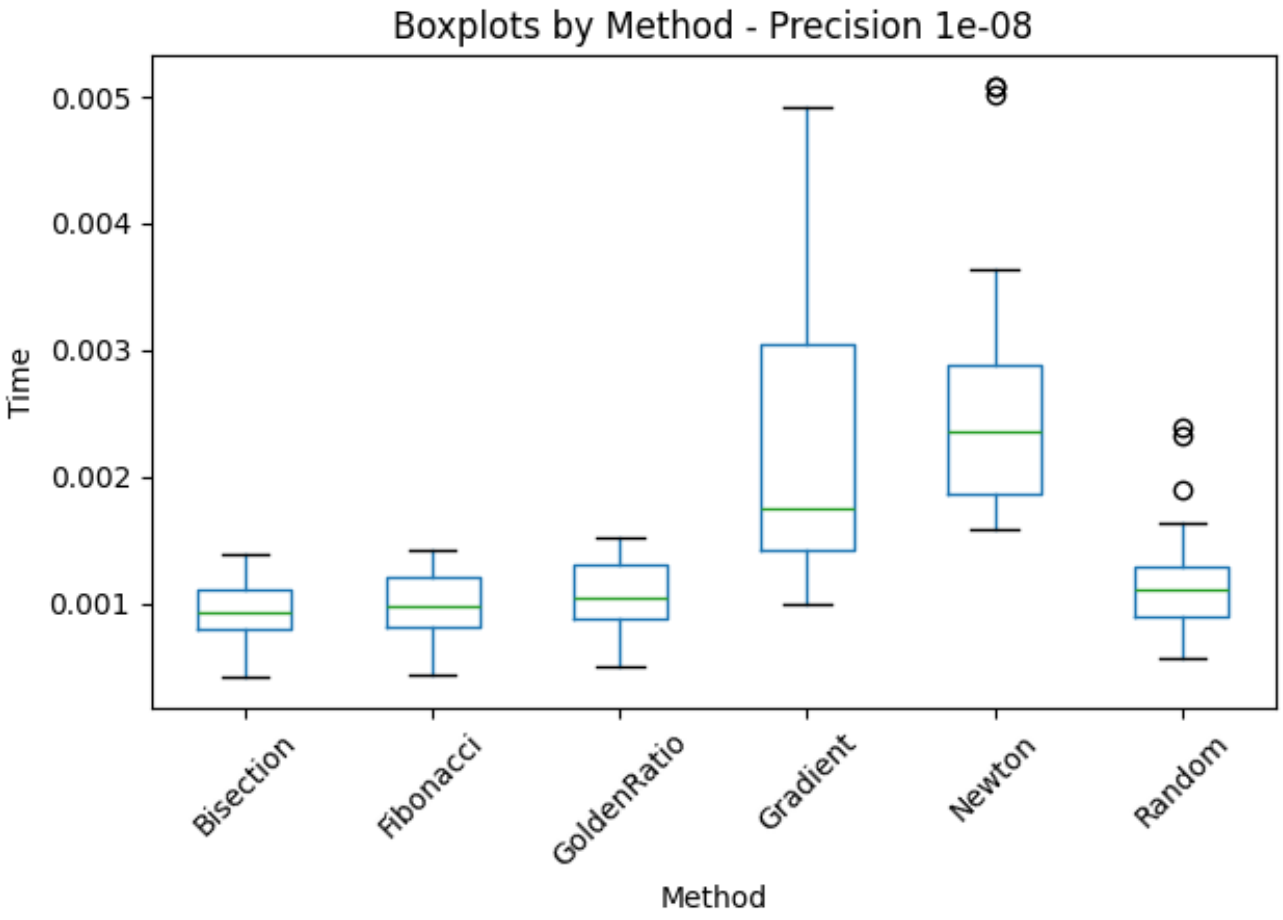

Рис. 17: Enter Caption

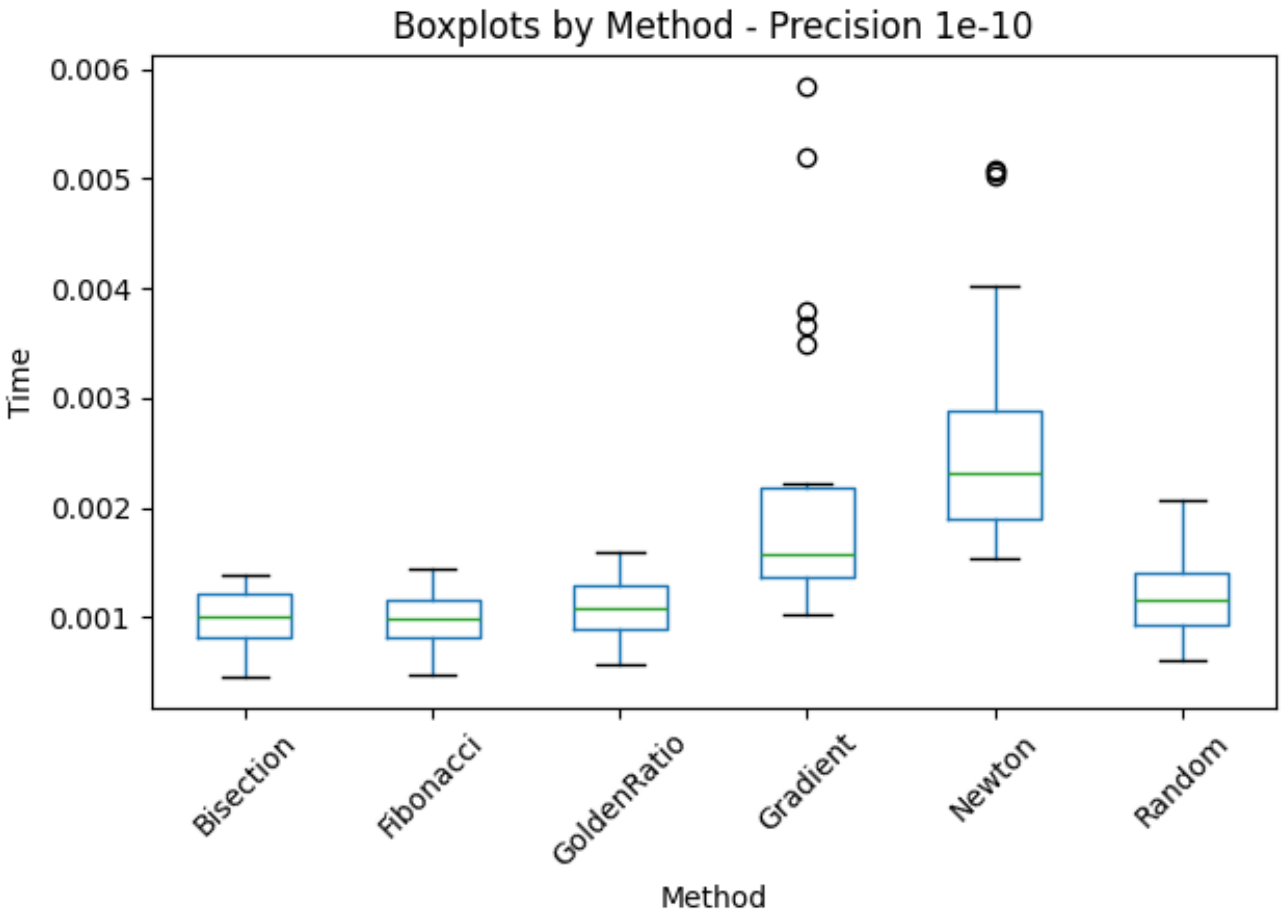

Рис. 18: Enter Caption

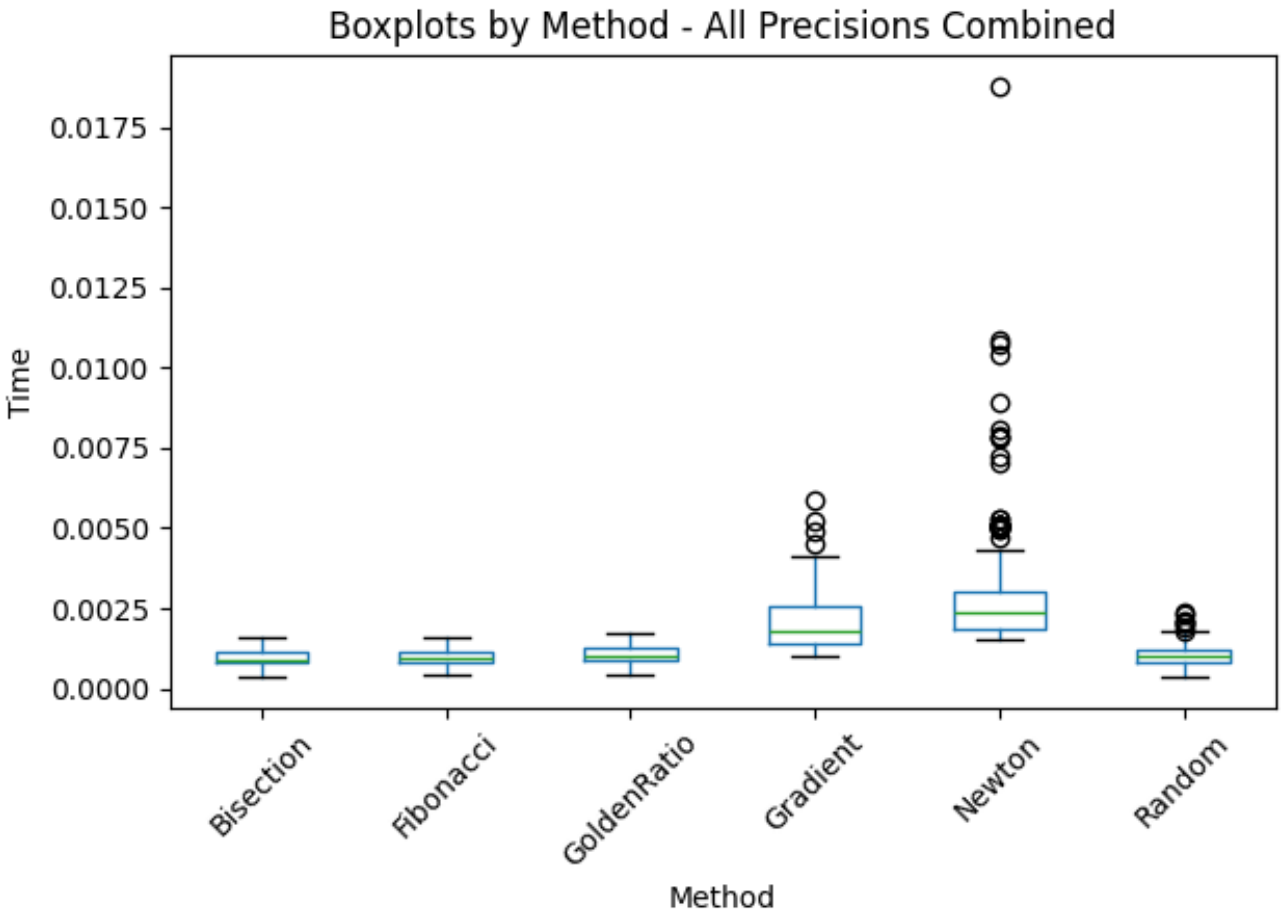

Рис. 19: Enter Caption

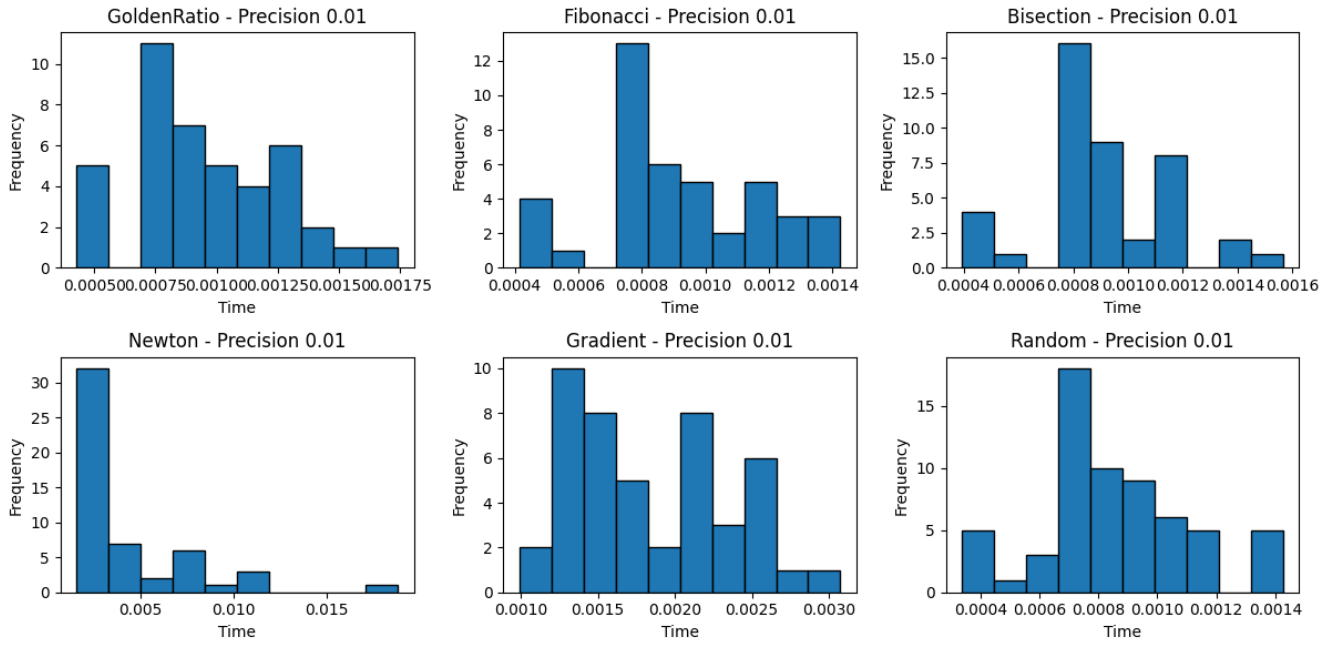

Рис. 20: Enter Caption

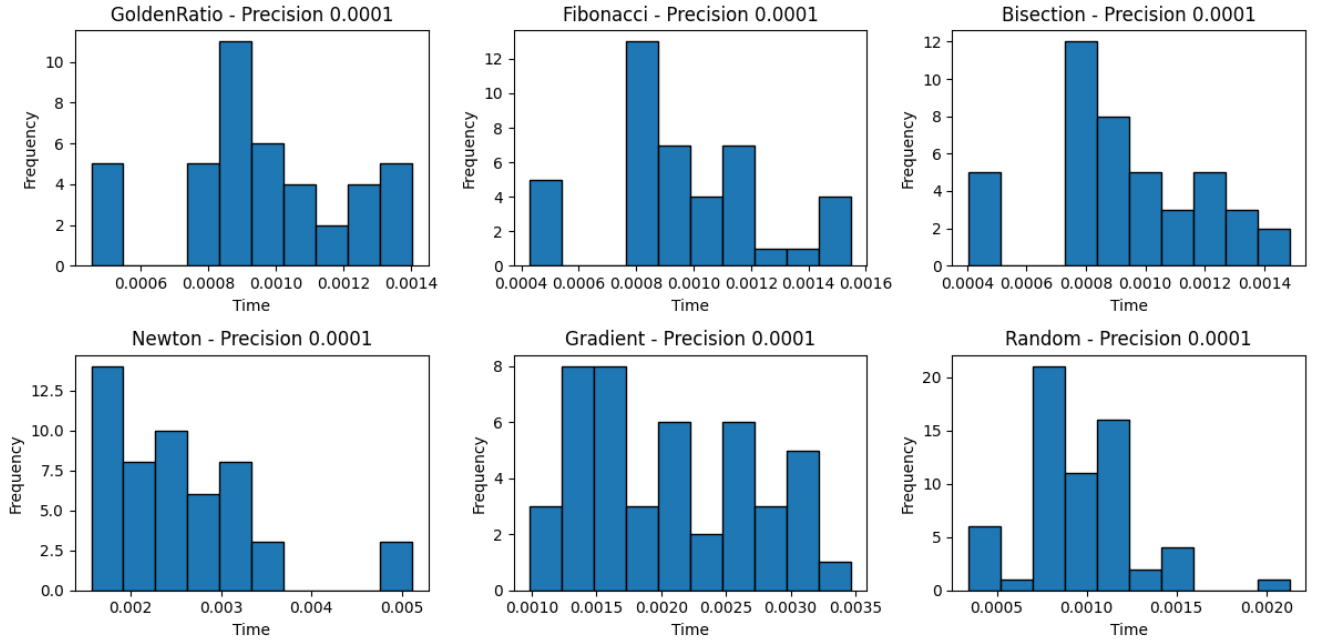

Рис. 21: Enter Caption

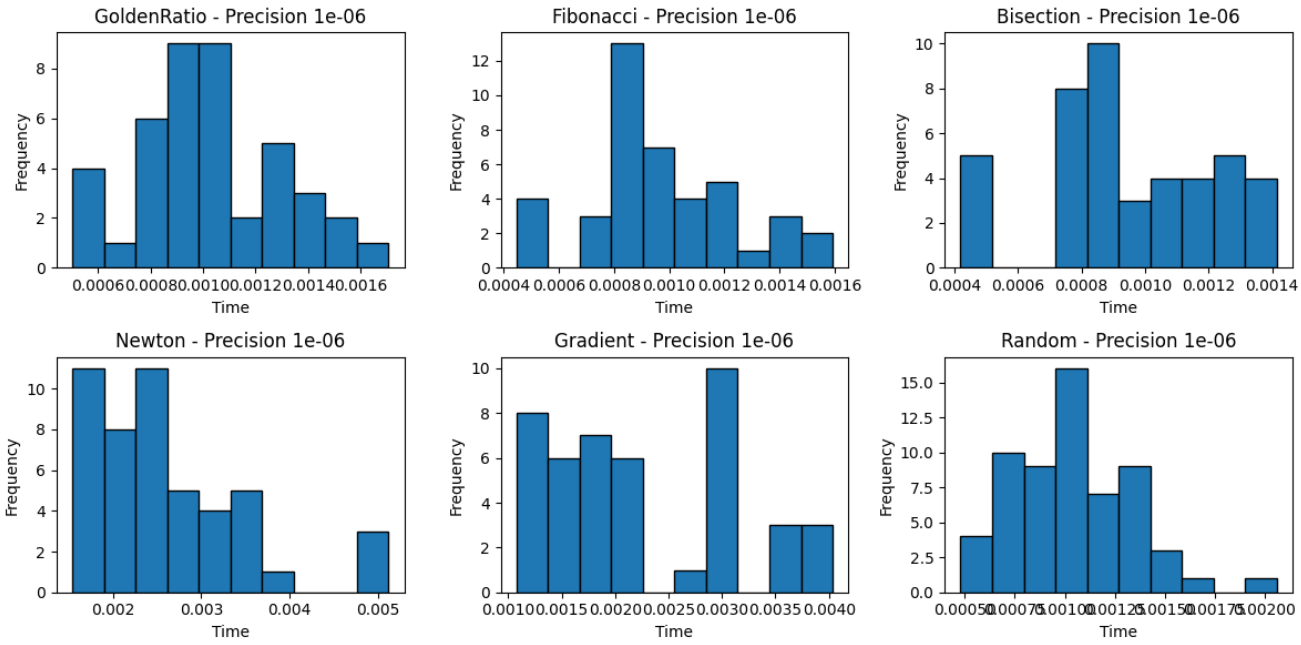

Рис. 22: Enter Caption

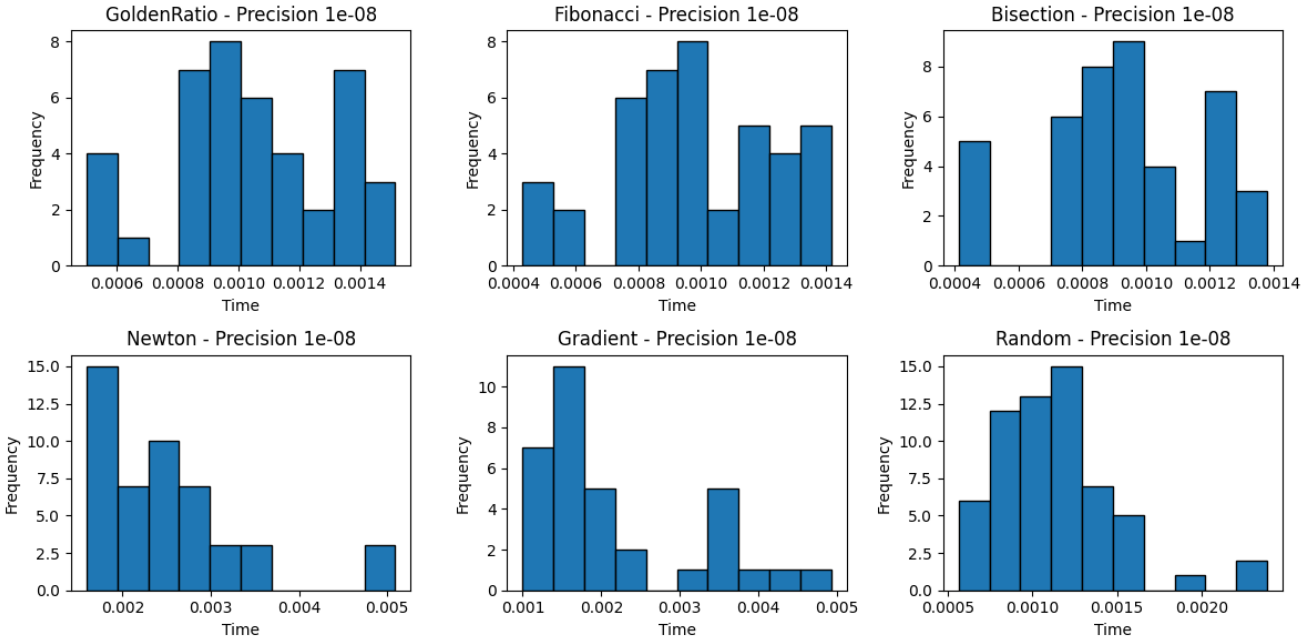

Рис. 23: Enter Caption

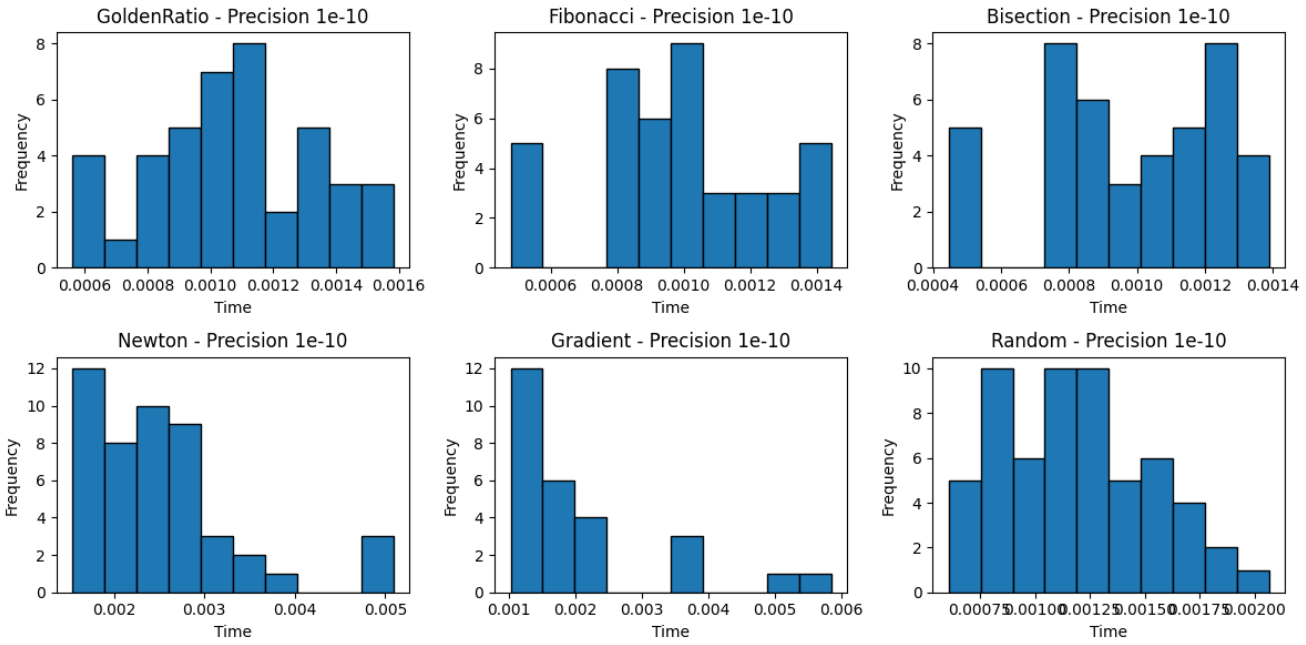

Рис. 24: Enter Caption

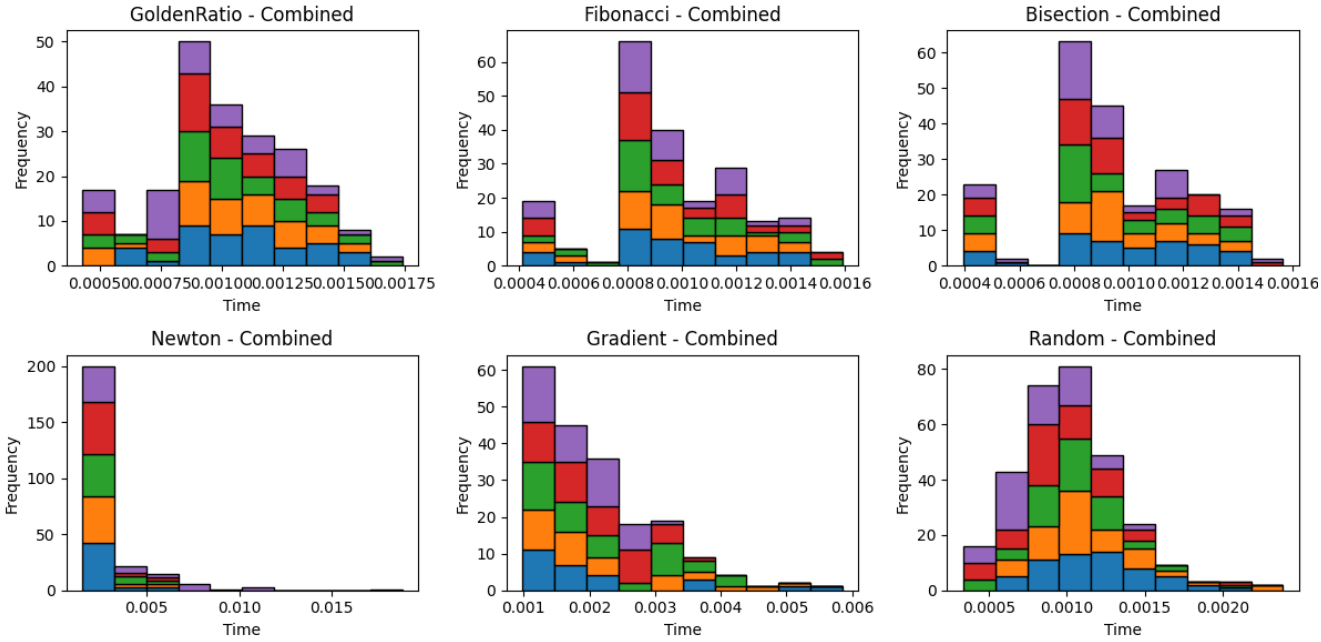

Рис. 25: Enter Caption

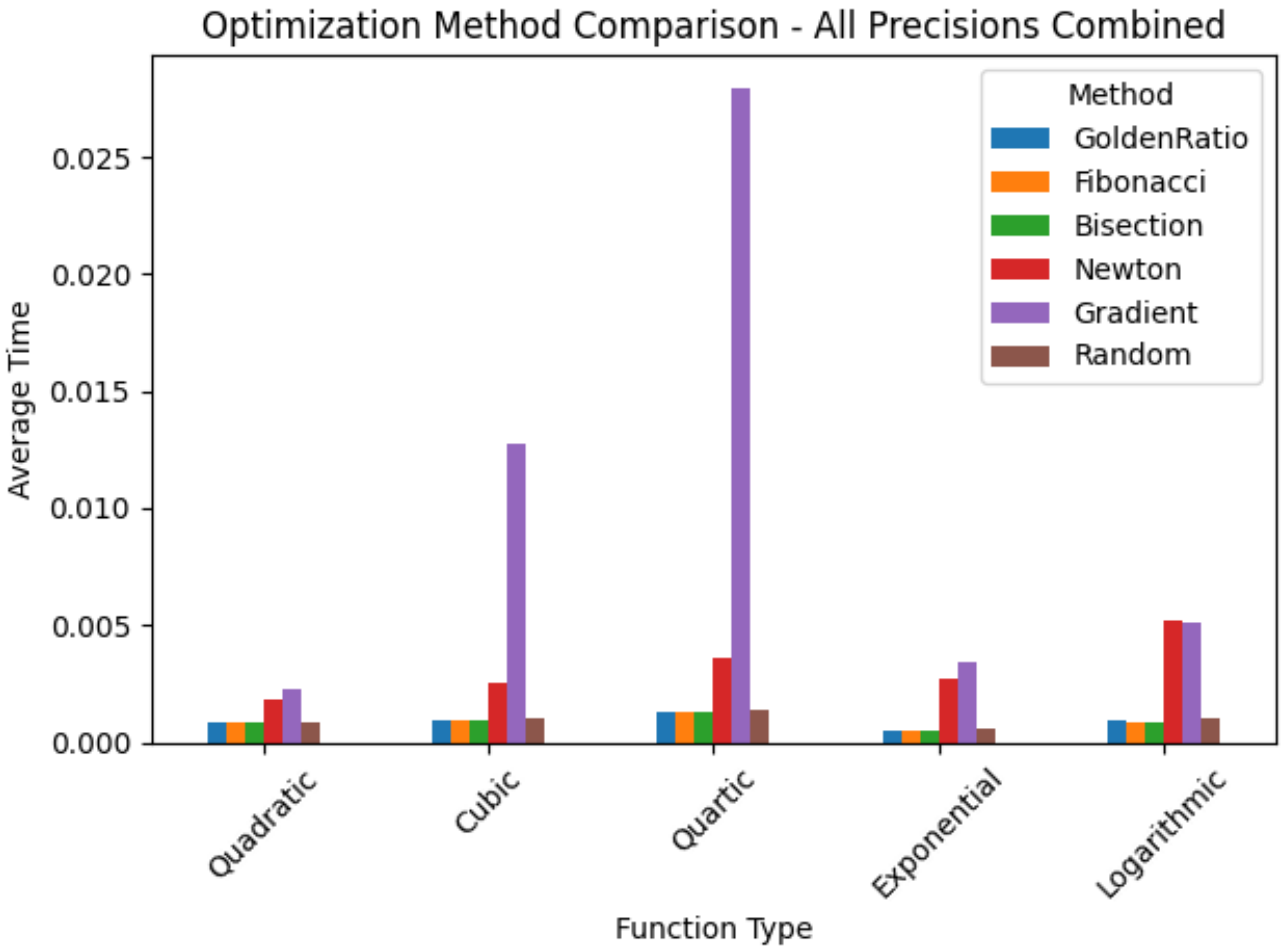

Рис. 26: Enter Caption

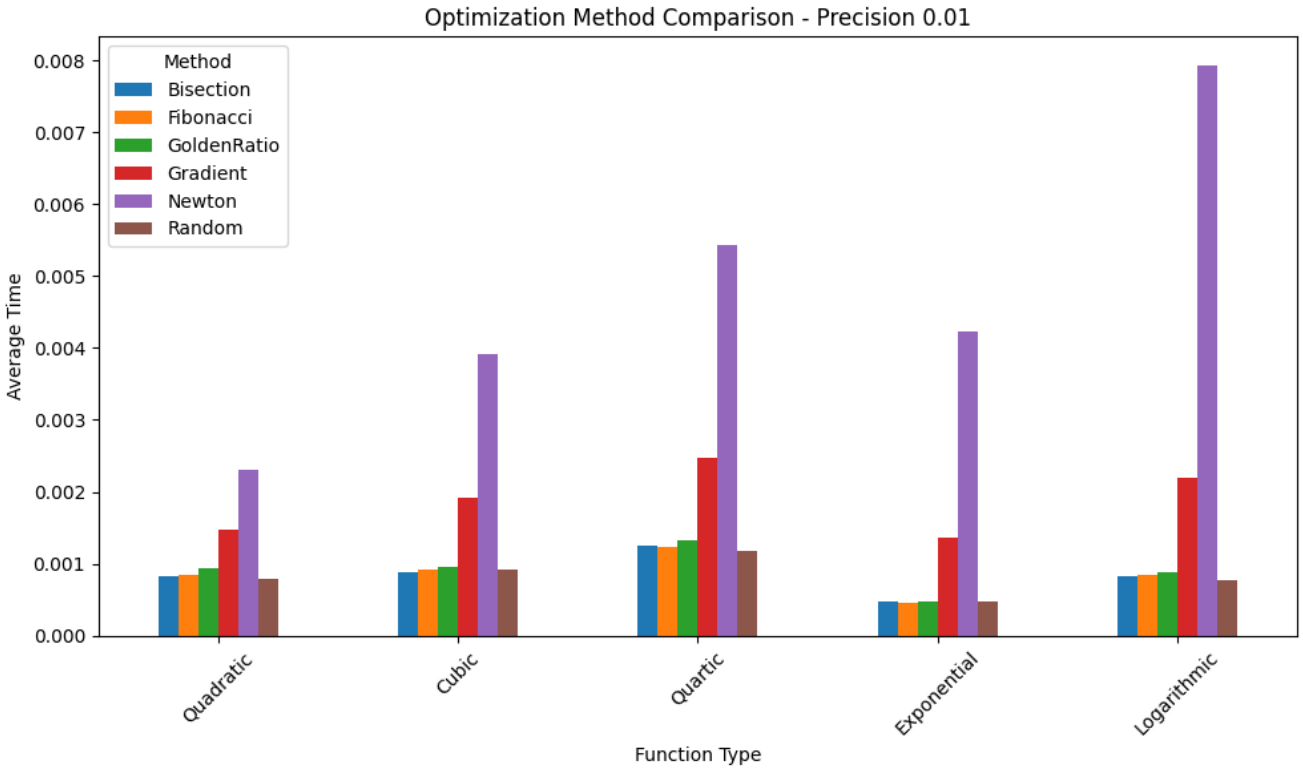

Рис. 27: Enter Caption

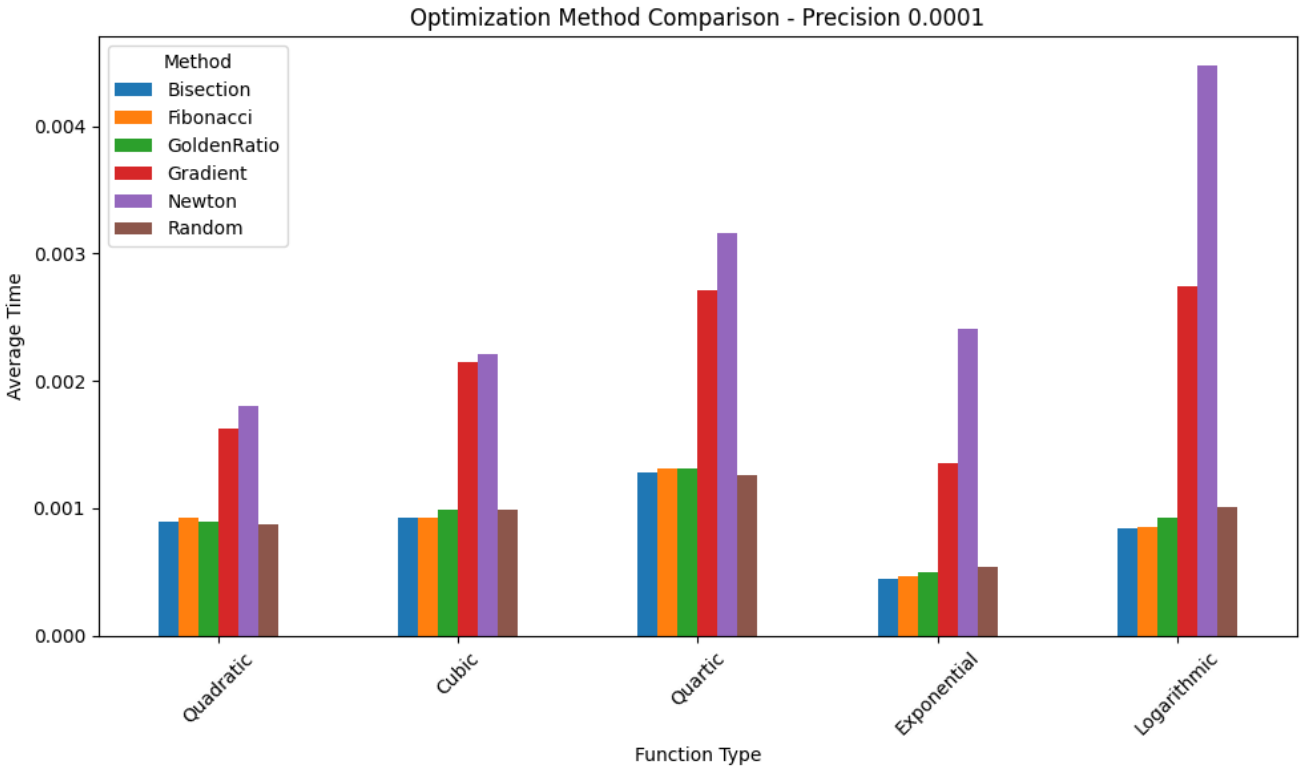

Рис. 28: Enter Caption

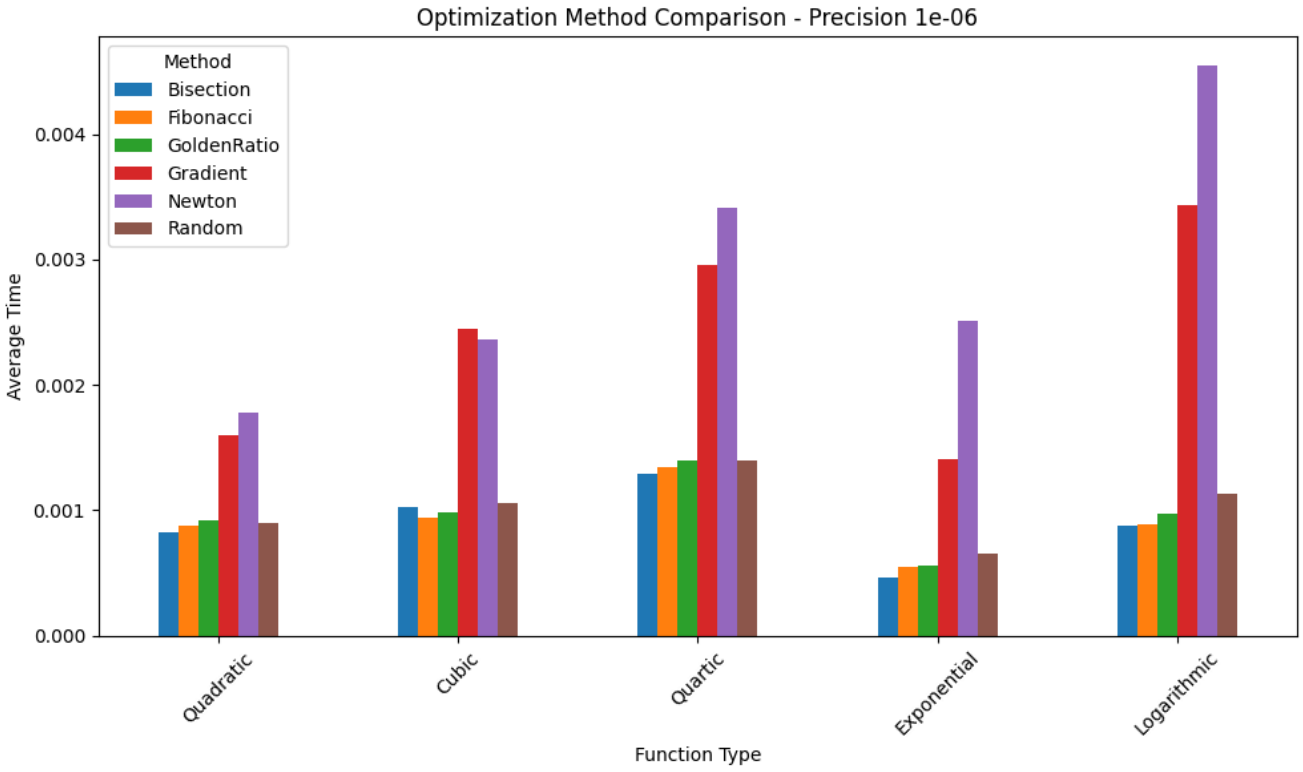

Рис. 29: Enter Caption

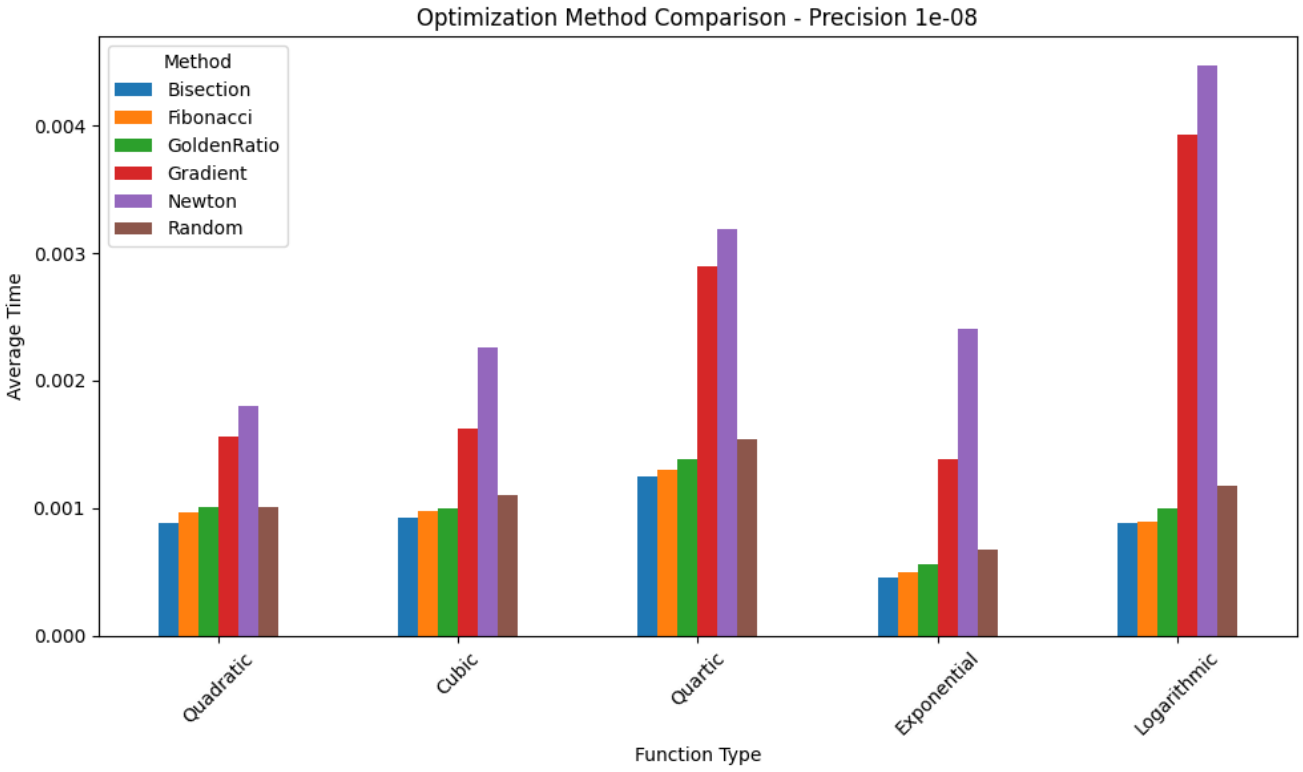

Рис. 30: Enter Caption

[H]

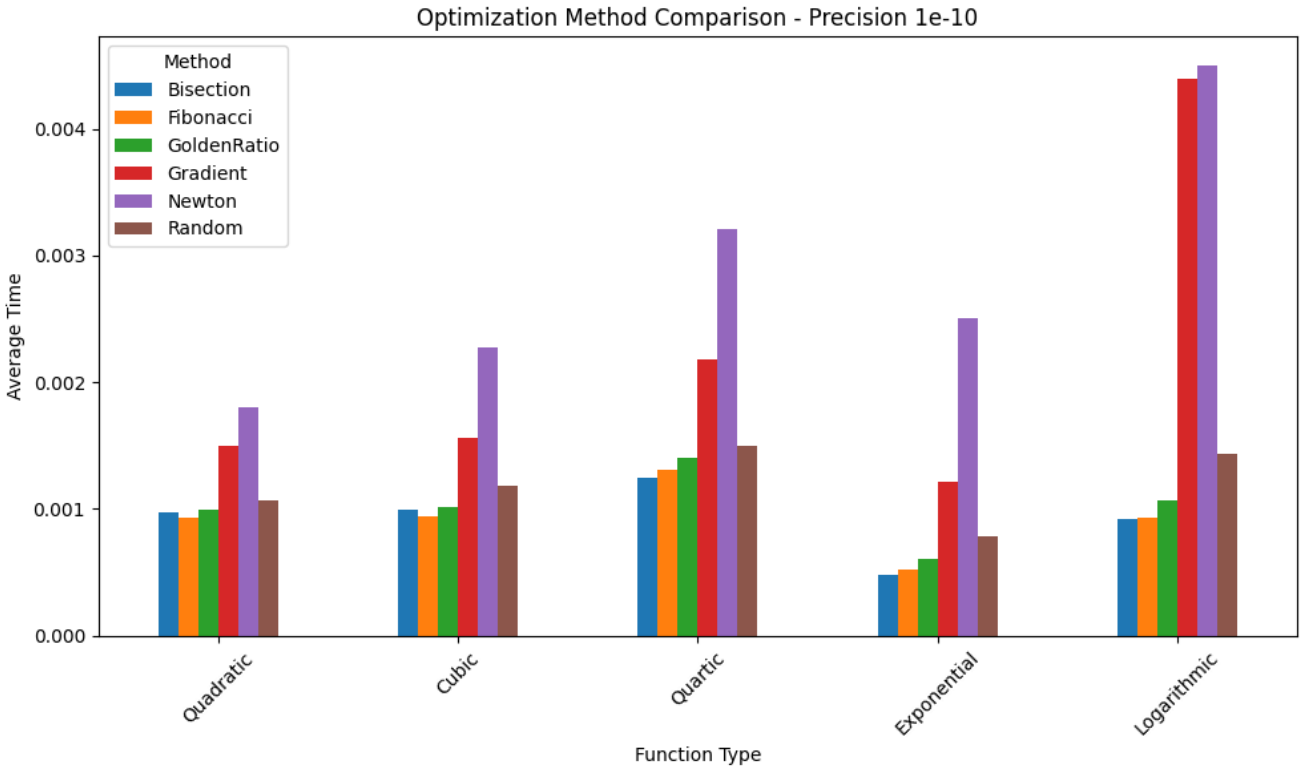

Рис. 31: Enter Caption

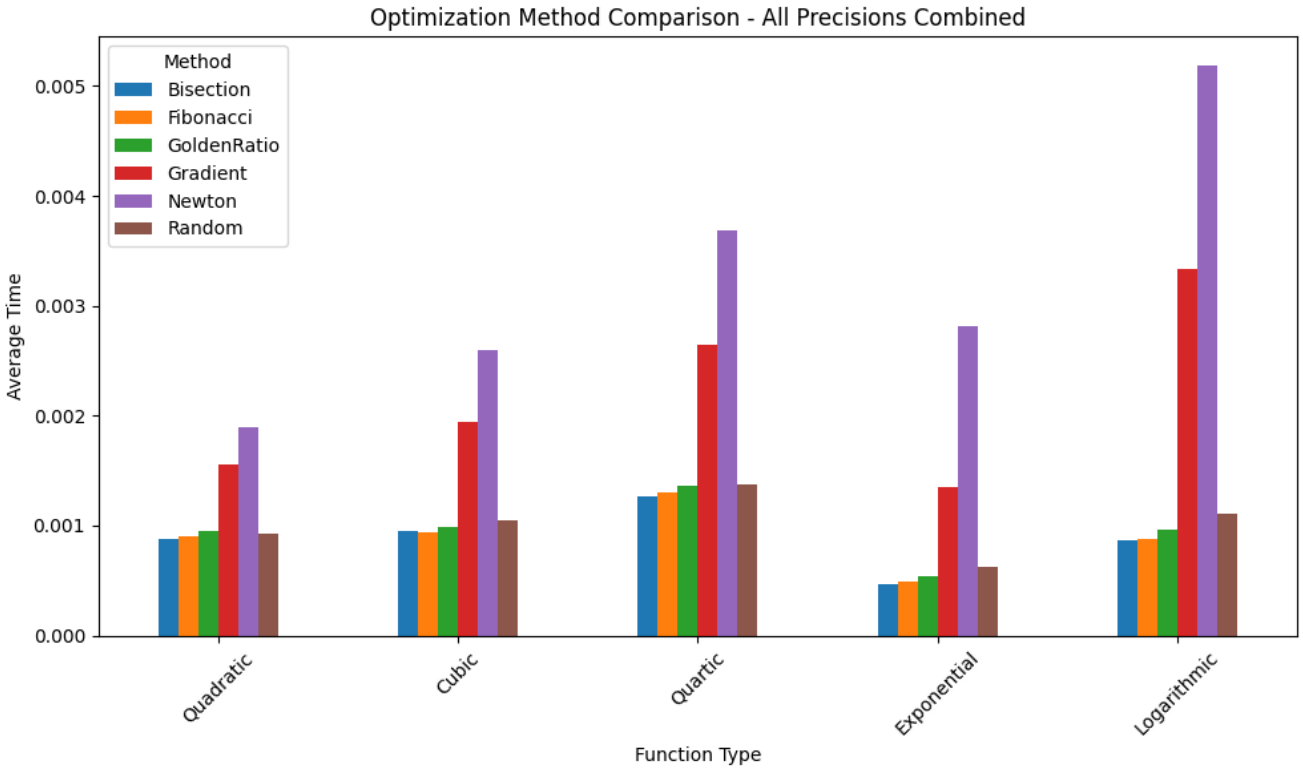

Рис. 32: Enter Caption

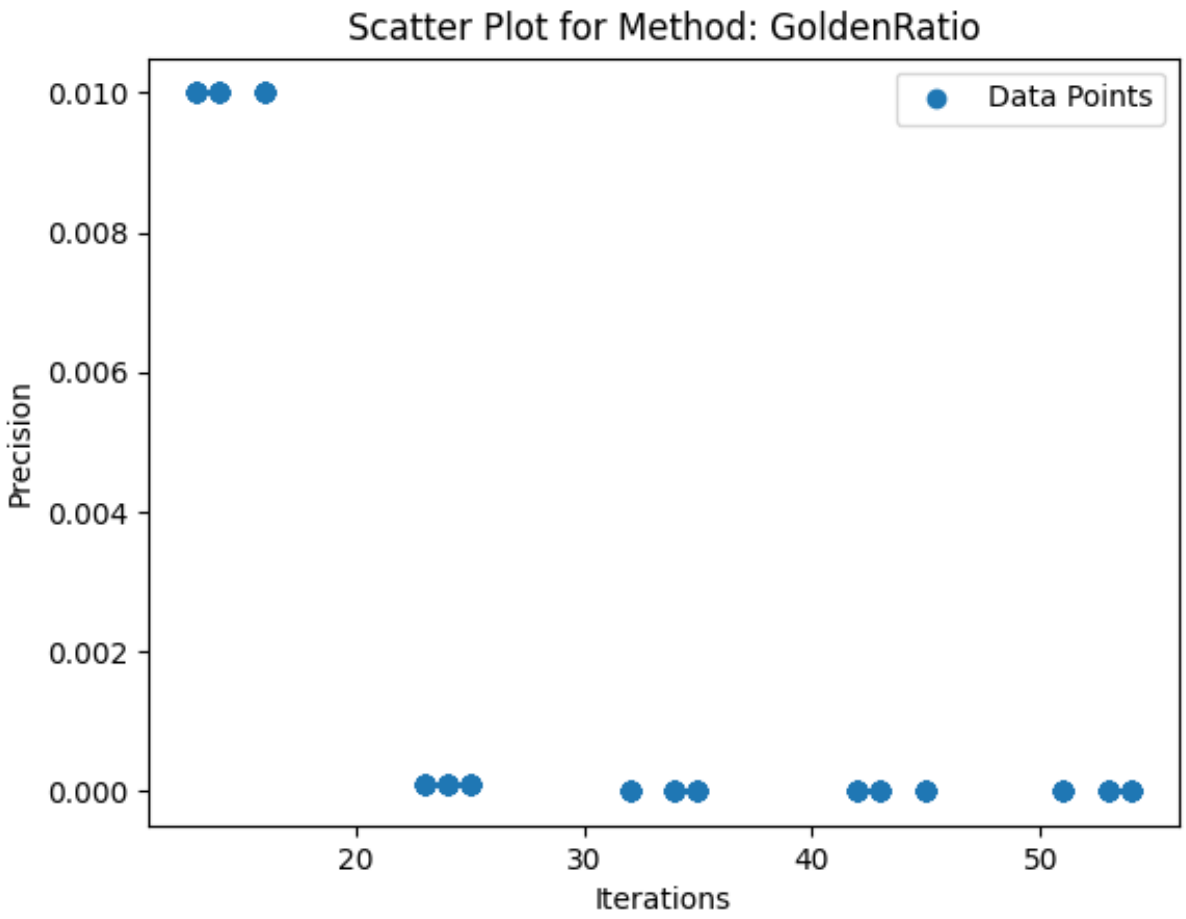

Рис. 33: Enter Caption

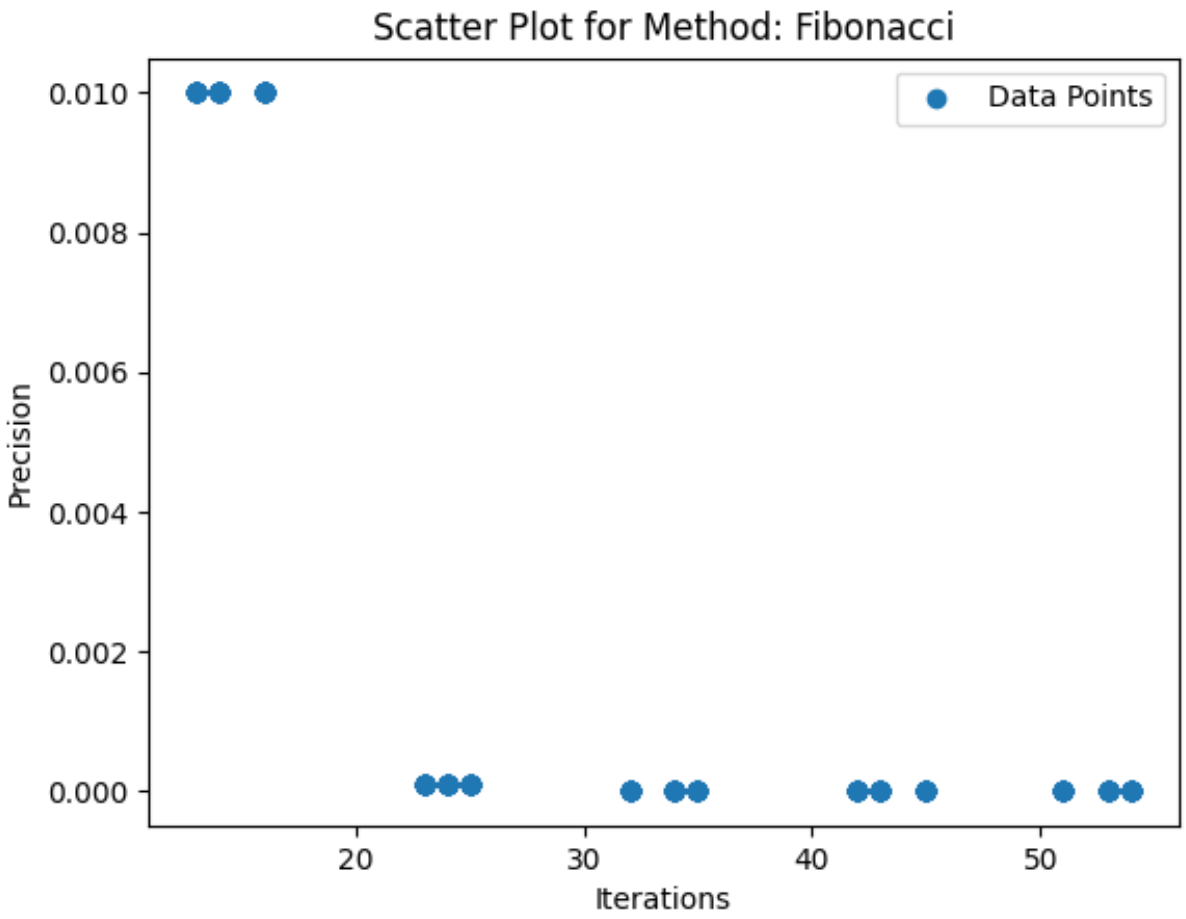

Рис. 34: Enter Caption

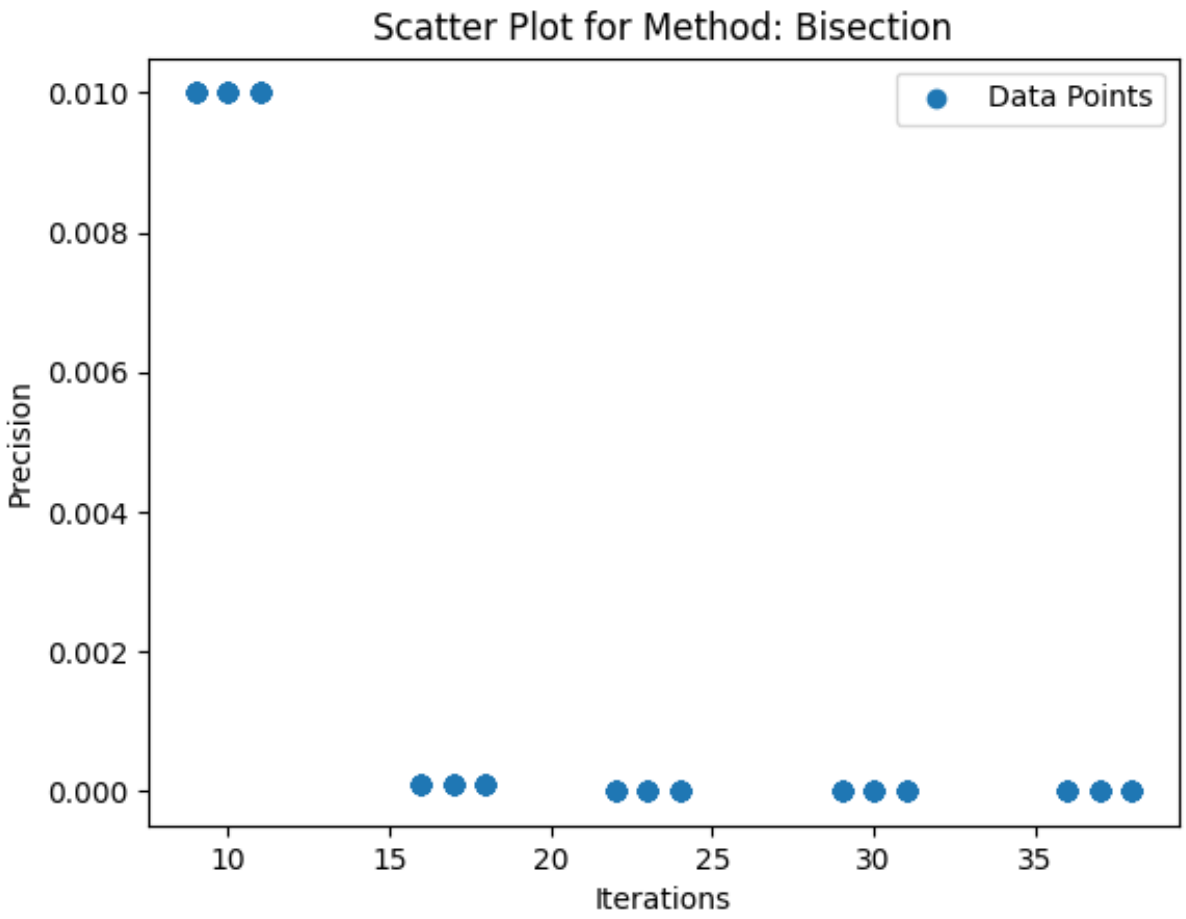

Рис. 35: Enter Caption

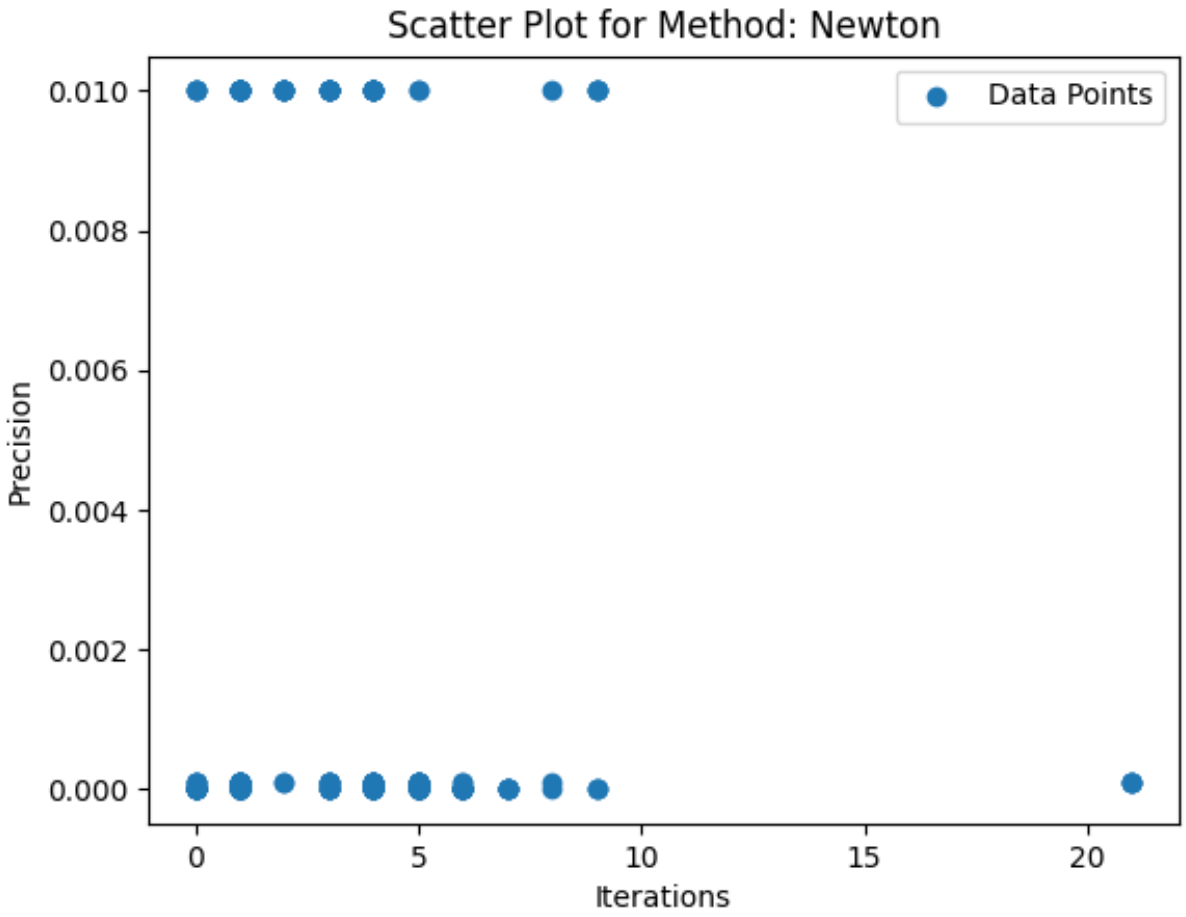

Рис. 36: Enter Caption

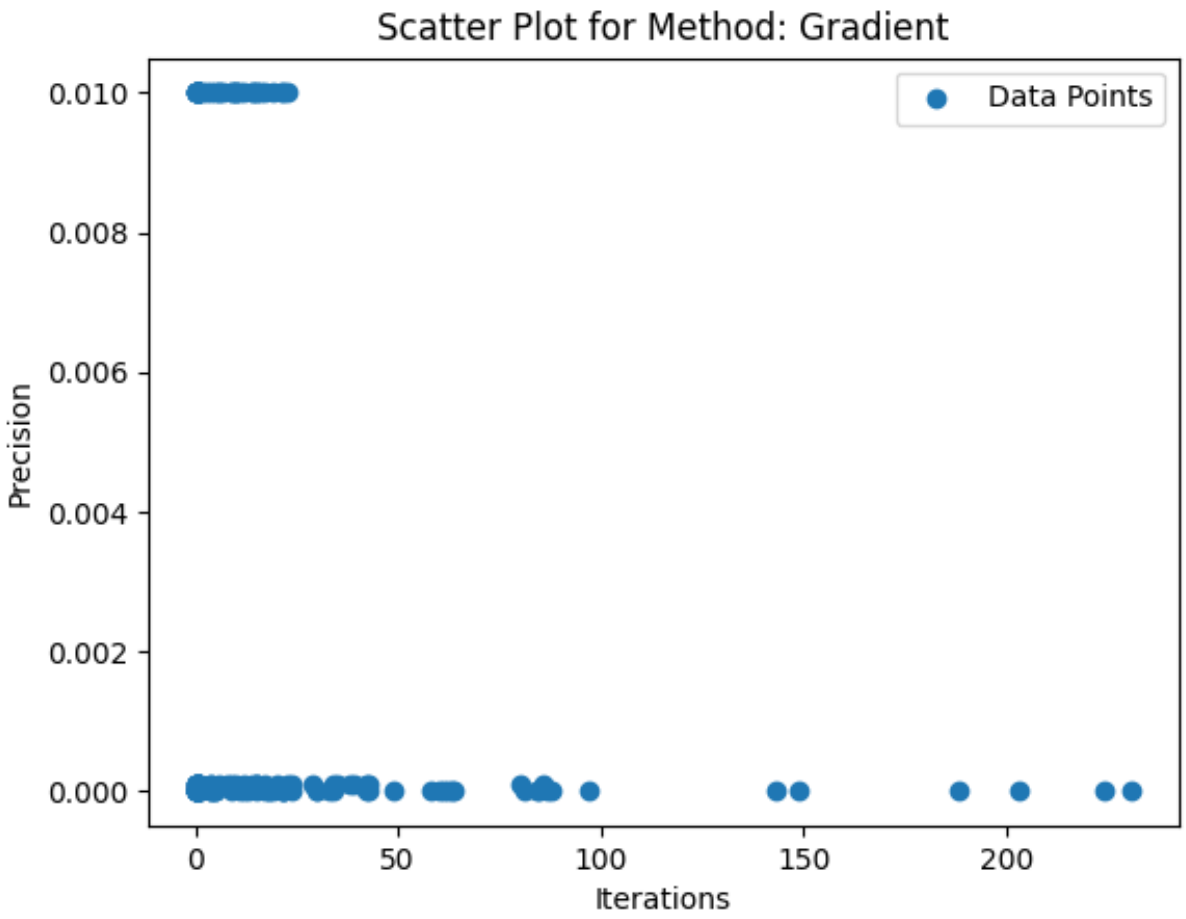

Рис. 37: Enter Caption

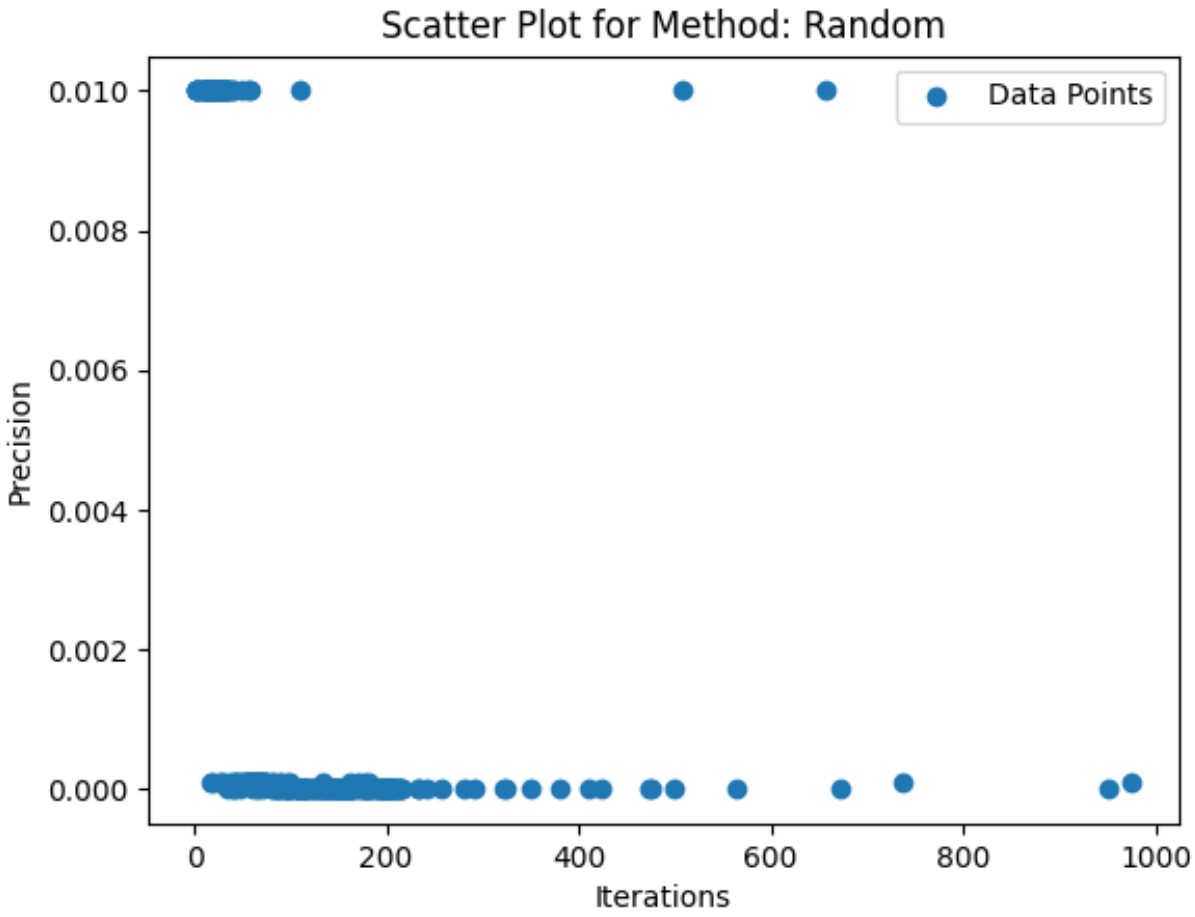

Рис. 38: Enter Caption

8.3 Виводи аналізу даних

Тест на нормальність та гомогенність дисперсій вивод:

Shapiro-Wilk test for method Bisection: Statistic: 0.9392315149307251, p-value: 0.024401120841503143 Data is not normally distributed (reject H0)

Shapiro-Wilk test for method Fibonacci: Statistic: 0.9473546743392944, p-value: 0.05195368826389313 Data is normally distributed (fail to reject H0)

Shapiro-Wilk test for method GoldenRatio: Statistic: 0.960222601890564, p-value: 0.1504192054271698 Data is normally distributed (fail to reject H0)

Shapiro-Wilk test for method Gradient: Statistic: 0.5525614023208618, p-value: 7.121234851803493e-11 Data is not normally distributed (reject H0)

Shapiro-Wilk test for method Newton: Statistic: 0.7325495481491089, p-value: 5.232516286923783e-08 Data is not normally distributed (reject H0)

Shapiro-Wilk test for method Random: Statistic: 0.9191308617591858, p-value: 0.0007903799414634705 Data is not normally distributed (reject H0)

Bartlett's test for homogeneity of variances: Statistic: 551.5756091566902, p-value: 5.840696577488205e-117 Variances are not homogeneous (reject H0)

Process finished with exit code 0

Тест на корреляцію - вивод:

Method: GoldenRatio, Correlation: -0.9820028733646522 Method: Fibonacci, Correlation: -0.9820028733646522 Method: Bisection, Correlation: -0.9820085172659275 Method: Newton, Correlation: -0.13083501729791458 Method: Gradient, Correlation: 0.037227906059889 Method: Random, Correlation: -0.82569308300009