# Отчёт по лабораторной работе №1

# Методы нулевого порядка

## Введение

В данной лабораторной работе были исследованы методы оптимизации: метод градиентного спуска, метод одномерного поиска с использованием золотого сечения и метод Нелдера-Мида. Целью работы было сравнить эффективность этих методов на различных функциях и проанализировать их сходимость.

## Описание используемых методов:

**Метод градиентного спуска.**
Метод градиентного спуска (Gradient Descent) — это метод поиска минимума функции, который использует информацию о градиенте функции для определения направления движения. Основное уравнение метода:

**x_new = x_cur - eta * nabla f(x_cur)**

, где eta — шаг или learning rate, nabla f(x) — градиент функции.

**Метод одномерного поиска с использованием золотого сечения.**
Этот метод использует принцип золотого сечения для минимизации функции одной переменной. На каждом шаге определяется новый интервал, в котором находится минимум функции.

**Метод Нелдера-Мида.**
Метод Нелдера-Мида (Nelder-Mead) — это метод оптимизации, который не требует вычисления производных и используется для минимизации многомерных нелинейных функций. Он работает, манипулируя простыми многоугольниками в пространстве параметров.

## Исследование

**f(x, y) = x^2 + y^2**

- Метод градиентного спуска показал быструю сходимость и высокую точность.
- Метод золотого сечения продемонстрировал хорошую производительность, но потребовал больше итераций.

**f(x, y) = x^2 + 100 * y^2**

- Из-за большого числа обусловленности (отношение максимального и минимального собственных значений гессиана), метод градиентного спуска испытывал трудности с сходимостью по переменной **y**.
- Появлялись предупреждения об переполнении:

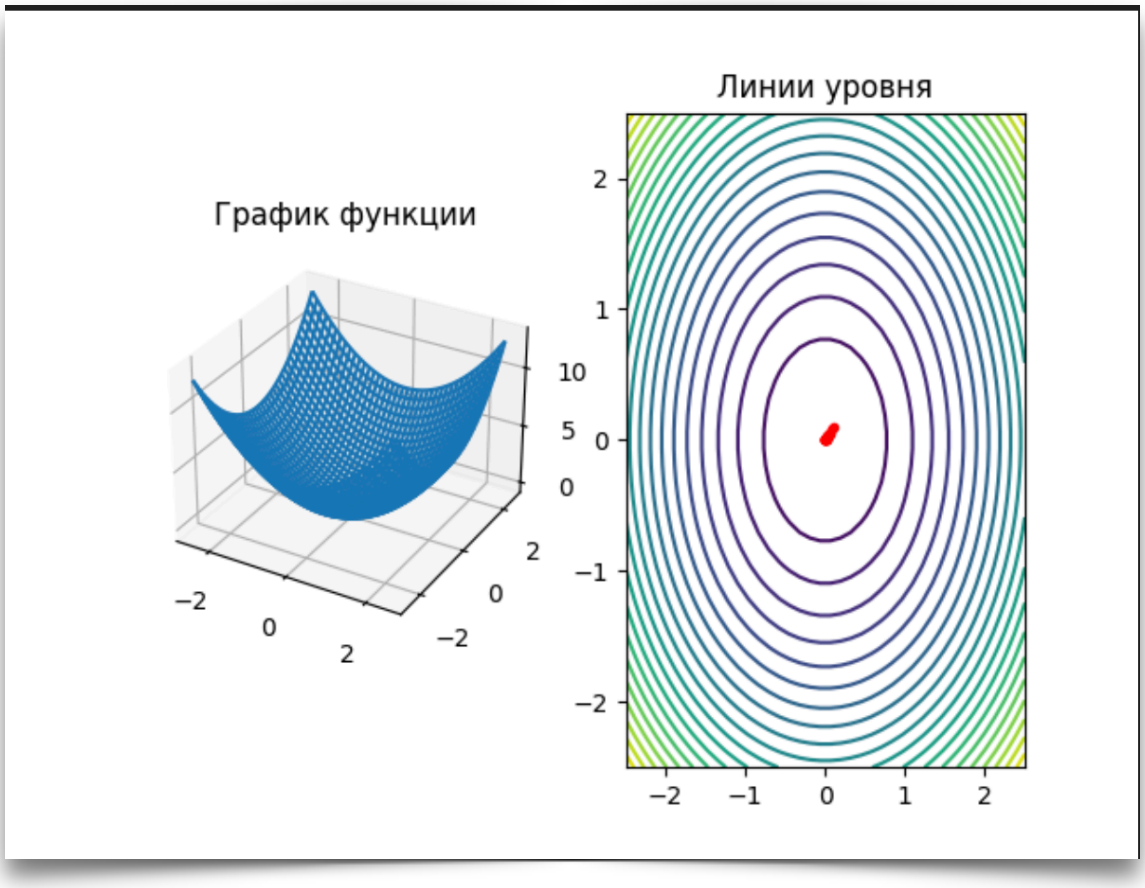
*RuntimeWarning: overflow encountered in scalar multiply*
*return x ** 2 + 100 * y ** 2*
*RuntimeWarning: overflow encountered in scalar power*
*return x ** 2 + 100 * y ** 2*

*3*. **Метод Нелдера-Мида**

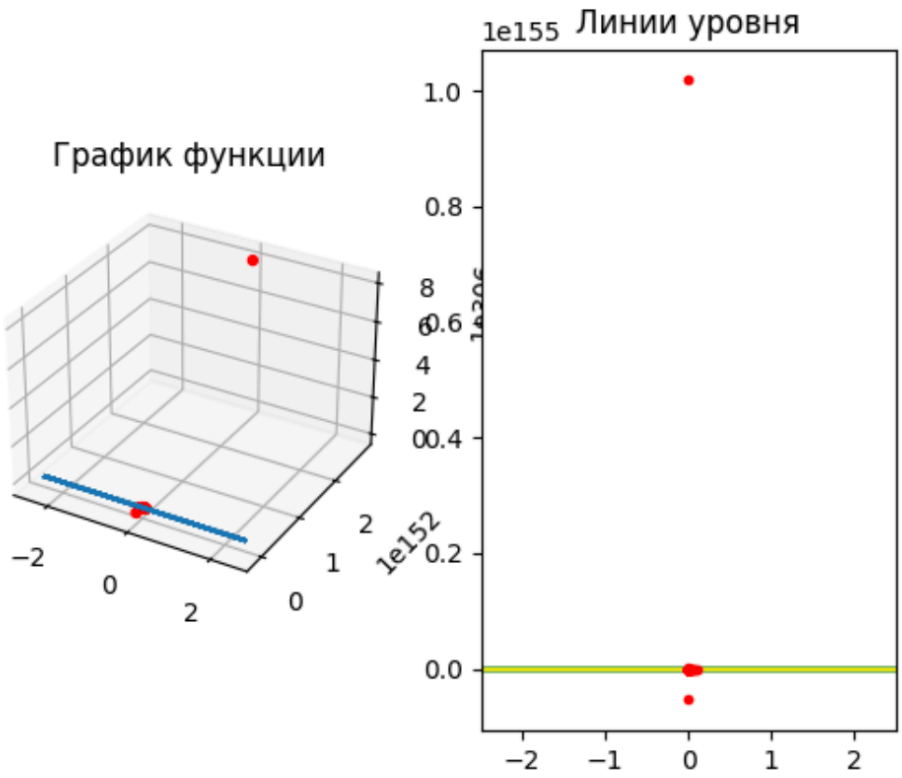
- Метод продемонстрировал стабильную работу на всех тестируемых функциях, однако время выполнения было значительно больше по сравнению с методами, использующими информацию о градиенте.

## Графики и анализ

На рисунке 1 приведены результаты работы методов на функции x^2 + y^2. Видно, что метод градиентного спуска нашёл минимум за несколько шагов. Метод золотого сечения также продемонстрировал хорошую сходимость, но потребовал больше времени на выполнение.




На рисунке 2 показаны результаты работы методов на функции ( x^2 + 100 * y^2 ). Заметно, что из-за разного масштаба переменных метод градиентного спуска испытывал трудности.




## Дополнительное задание 1

### Метод одномерного поиска и покоординатный спуск

Для выполнения дополнительного задания 1 был выбран метод покоординатного спуска, который сочетался с методом одномерного поиска золотого сечения. Метод покоординатного спуска позволяет эффективно минимизировать функции, оптимизируя каждую переменную отдельно, что особенно полезно для функций с разными масштабами переменных.

**Исследование**

Метод покоординатного спуска (Coordinate Descent) оптимизирует функцию последовательно по каждой из переменных. Это позволяет избежать переполнений и улучшить сходимость для плохо обусловленных функций.

**f_1(x, y) = x^2 + y^2**

- Целевая функция для проверки сходимости и эффективности методов на простой квадратичной функции.

**f_2(x, y) = x^2 + 1000 * y^2**

- Целевая функция с высоким числом обусловленности для проверки устойчивости методов.

**Реализация**

Метод покоординатного спуска был реализован в файле coordinate_descent.py и использовался для оптимизации функций f_1 и f_2 . В качестве метода одномерного поиска применялся метод золотого сечения, реализованный в golden_section_search.py.

**Результаты**

$$f\_1(x, y) = x^2 + y^2$$

- Метод покоординатного спуска показал быструю сходимость и точность при минимизации функции. Оптимальные значения были достигнуты за небольшое количество итераций.

$$f\_2(x, y) = x^2 + 1000 * y^2$$

- Метод покоординатного спуска показал лучшую устойчивость к разнице в масштабах переменных и избежал переполнения по переменной **y**. Время выполнения было увеличено из-за необходимости адаптивного выбора шага.

## Дополнительное задание 2

### Исследование сложных случаев

**Исследование на функциях n переменных**

$$f(x, y, z) = x^3 * y^3 * z^3$$

Функция высокой размерности для проверки сходимости методов на многомерных задачах. Метод градиентного спуска был использован для оптимизации функции f с использованием файла gradient_descent_multiple_variable.py. Использовался шаг eta = 0.1.

**Результаты:**

- Метод градиентного спуска показал хорошую сходимость и высокую точность при минимизации функции с тремя переменными. Оптимальные значения были достигнуты за разумное количество итераций, несмотря на высокую размерность пространства.

**Исследование плохо обусловленных функций**

$$f\_bad(x, y) = x^3 + y^3 + \sqrt{x^4 + y^4}$$

- Функция с высоким числом обусловленности для проверки устойчивости методов к переполнению.

**Результаты**

- Метод градиентного спуска столкнулся с проблемами из-за высокой обусловленности функции, что привело к переполнению. Время выполнения было увеличено из-за необходимости адаптивного выбора шага.

## Таблицы

Общая таблица входных данных при learning rate = 0.99

| | Method | Function | Learning Rate | Initial Point |
|---|---|---|---|---|
| 0 | Gradient Descent | x^2 + y^2 | 0.99 | (0.1, 0.1) |
| 1 | Gradient Descent | x^2 + 100 * y^2 | 0.99 | (0.1, 0.1) |
| 2 | Gradient Descent | x^2 + y^2 | Golden section search | (0.1, 0.1) |
| 3 | Gradient Descent | x^2 + 100 * y^2 | Golden section search | (0.1, 0.1) |
| 4 | Scipy Minimize | x^2 + y^2 | Nelder-Mead | (0.1, 0.1) |
| 5 | Scipy Minimize | x^2 + 1000 * y^2 | Nelder-Mead | (0.1, 0.1) |
| 6 | Coordinate Descent | x^2 + y^2 | Golden section search | (0.1, 0.1) |
| 7 | Coordinate Descent | x^2 + 1000 * y^2 | Golden section search | (0.1, 0.1) |
| 8 | Gradient Descent | x^3 * y^3 * z^3 | 0.1 | (1, 1, 1) |
| 9 | Gradient Descent | x**3 + y**3 + np.sqrt(x**4 + y**4) | Golden section search | (0.1, 0.1) |
| 10 | Gradient Descent | x^2 * y^2 * log(x^2 + y^2) with noise | Golden section search | (0.1, 0.1) |

Результаты работы и гиперпараметры

| Optimal Point | Function Value | Iterations | Execution Time |
|---|---|---|---|
| (-0.0034257419087928617, -0.0034257419087928617) | 2.347142e-05 | 167 | 0.000270 |
| (-0.02480842668343483, -2.0805090302657592e+157) | inf | 69 | 0.001215 |
| (1.6099624430392533e-15, 1.6099624430392533e-15) | 5.183958e-30 | 2 | 0.000077 |
| (9.263681513643177e-06, 9.263668355432861e-06) | 8.667371e-09 | 4 | 0.000139 |
| (-1.552281980928567e-09, 5.504060914521181e-09) | 3.270427e-17 | 61 | 113.000000 |
| (4.650769607971084e-11, -7.20870340531329e-10) | 5.196757e-17 | 70 | 131.000000 |
| (1.3931831859930762e-18, -3.732536918432672e-10) | 1.393183e-19 | 2 | 0.000089 |
| (9.466615411015083e-06, 9.466612170847069e-06) | 9.051291e-09 | 4 | 0.000175 |
| (0.33490149885113785, 0.3349014988511379, 0.3349014988511376) | 5.299731e-05 | 1000 | 0.002333 |
| (-1.3633238326486213e-09, -1.3633238326486213e-09) | 2.628531e-18 | 2 | 0.000146 |
| (4.65524463517063e-05, 4.65524463517063e-05) | -3.059638e-07 | 2 | 0.001001 |

Общая таблица входных данных при learning rate = 0.1

| | Method | Function | Learning Rate | Initial Point |
|---|---|---|---|---|
| 0 | Gradient Descent | x^2 + y^2 | 0.1 | (0.1, 0.1) |
| 1 | Gradient Descent | x^2 + 100 * y^2 | 0.1 | (0.1, 0.1) |
| 2 | Gradient Descent | x^2 + y^2 | Golden section search | (0.1, 0.1) |
| 3 | Gradient Descent | x^2 + 100 * y^2 | Golden section search | (0.1, 0.1) |
| 4 | Scipy Minimize | x^2 + y^2 | Nelder-Mead | (0.1, 0.1) |
| 5 | Scipy Minimize | x^2 + 1000 * y^2 | Nelder-Mead | (0.1, 0.1) |
| 6 | Coordinate Descent | x^2 + y^2 | Golden section search | (0.1, 0.1) |
| 7 | Coordinate Descent | x^2 + 1000 * y^2 | Golden section search | (0.1, 0.1) |
| 8 | Gradient Descent | x^3 * y^3 * z^3 | 0.1 | (1, 1, 1) |
| 9 | Gradient Descent | x**3 + y**3 + np.sqrt(x**4 + y**4) | Golden section search | (0.1, 0.1) |
| 10 | Gradient Descent | x^2 * y^2 * log(x^2 + y^2) with noise | Golden section search | (0.1, 0.1) |

Результаты работы и гиперпараметры

| Optimal Point | Function Value | Iterations | Execution Time |
|---|---|---|---|
| (0.0009223372036854776, 0.0009223372036854776) | 1.701412e-06 | 21 | 0.000045 |
| (1.5030672529752546e-13, 1.0184490707224479e+155) | inf | 122 | 0.001382 |
| (1.6099624430392533e-15, 1.6099624430392533e-15) | 5.183958e-30 | 2 | 0.000079 |
| (9.263681513643177e-06, 9.263668355432861e-06) | 8.667371e-09 | 4 | 0.000141 |
| (-1.552281980928567e-09, 5.504060914521181e-09) | 3.270427e-17 | 61 | 113.000000 |
| (4.650769607971084e-11, -7.20870340531329e-10) | 5.196757e-17 | 70 | 131.000000 |
| (1.3931831859930762e-18, -3.732536918432672e-10) | 1.393183e-19 | 2 | 0.000089 |
| (9.466615411015083e-06, 9.466612170847069e-06) | 9.051291e-09 | 4 | 0.000175 |
| (0.33490149885113785, 0.3349014988511379, 0.3349014988511376) | 5.299731e-05 | 1000 | 0.002360 |
| (-1.3633238326486213e-09, -1.3633238326486213e-09) | 2.628531e-18 | 2 | 0.000148 |
| (0.00029250418033015734, 0.00029250418033015734) | -1.524902e-07 | 3 | 0.001355 |

## Реализация

Код был написан на Python и включал следующие файлы:

- main.py — основной файл для запуска экспериментов.
- gradient_descent.py — реализация метода градиентного спуска.
- golden_section_search.py — реализация метода золотого сечения.
- coordinate_descent.py — реализация метода покоординатного спуска.
- plot_graphs.py — скрипт для построения графиков.
- gradient_descent_multiple_variable.py — метод градиентного спуска для многомерных функций.