Лабораторная работа №1	*2024*
Методы нулевого порядка	**Артемьев Иван Вадимович**
	Никитин Михаил Алексеевич
	Павлов Евгений Андреевич

Цель работы: Реализация изученных методов нулевого порядка для поиска минимумов функций на языке программирования Python и исследование полученных результатов

Инструментарий: Python 3.12 - библиотеки: NumPy, SciPy, MatPlotLib

Постановка задачи:

Реализуйте и исследуйте на эффективность следующие методы:

1. Метод градиентного спуска с постоянным шагом (learning rate);
2. Любой метод одномерного поиска и градиентный спуск на его основе;
3. Метод Нелдера-Мида. При этом используйте готовую реализацию в Python библиотеке scipy.optimize. Изучите возможности библиотеки scipy.optimize.
4. Метод покоординатного спуска

Содержание исследования:

Для исследования подберите 2-3 квадратичные функции двух переменных, на которых эффективность методов будет явно отличаться; Сравните методы на каждой из этих функциях:

1. Исследуйте сходимость и сравните эффективность методов на выбранных функциях, с учетом количества итераций и количества

вычислений значений минимизируемой функции и ее градиентов, в зависимости от желаемой точности;
2. Исследуйте работу методов в зависимости от выбора начальной точки;
3. В каждом случае иллюстрируйте примеры. Нарисуйте графики рассматриваемых функций (3D), нарисуйте графики с линиями уровня и траекториями методов (2D, в области задания). Вычисленные значения оформите в виде сравнительных таблиц.

Описание используемых методов и их реализация:

$f(x,\ y)$: $\mathbb{R}^2 \to \mathbb{R}$

Задача оптимизации: $f(x,\ y) \to min$

1. *Метод градиентного спуска с постоянным шагом:*

Основная идея метода заключается в том, чтобы идти в направлении наискорейшего спуска, а это направление задаётся антиградиентом $-\nabla f$:

$$x^{[j+1]} = x^{[j]} - \alpha \cdot \nabla f(x^{[j]})$$

в данной формуле $x^{[j]}$ - это j-ое значение вектора аргументов функции, α - шаг градиентного спуска (чем α больше, тем и алгоритм работает быстрее, однако при слишком больших значениях - алгоритм расходится) - также это число называется *learning rate.*

В итоге получается следующий алгоритм

a. Задается начальное приближение x^0 и точность расчета ε
b. Рассчитывают $x^{[j+1]} = x^{[j]} - \alpha \cdot \nabla f(x^{[j]})$
c. Проверяют условие остановки:

i. $\left|x^{[j+1]} - x^{[j]}\right| > \varepsilon \Rightarrow \quad j = j + 1$ и возвращение к шагу *b.*

ii. Иначе $x = x^{[j+1]}$ и остановка.

Реализация данного метода в нашем репозитории представлена в файле *gradient.py*

2. Метод градиентного спуска с изменяемым шагом:

Данный метод является модификацией описанного выше метода градиентного спуска, которая заключается в исправлении проблемы с тем, что алгоритм может не сойтись при некоторых значениях α - *learning rate.* На каждом шаге алгоритма выбирается новое значение α с помощью метода дихотомии для функции $g(step) = f(x^{[j]} - step \cdot \nabla f(x^{[j]}))$ (обратим внимание, что в данном случае $x^{[j]}$ и $\nabla f(x^{[j]})$ - фиксированные числа, то есть g - функция одной переменной и к ней действительно применим данный метод):

A. Задаётся начальный интервал *(0, learning_rate)*;
B. Убедиться, что на концах функция g имеет разный знак;
C. Повторять
    a. выбрать внутри интервала точку *mid*;
    b. сравнить знак функции в точке mid со знаком функции в одном из концов;
        i. если совпадает, то переместить этот конец интервала в точку *mid*,
        ii. иначе переместить в точку mid другой конец интервала;

пока не будет достигнута нужная точность.

Благодаря данной модификации - метод градиентного спуска больше не будет расходиться, так как значение α будет уменьшаться.

Реализация метода дихотомии представлена в файле *d1_methods.py*

3. Метод Нелдера-Мида

Алгоритм заключается в формировании треугольника и последующего его деформирования в направлении минимума, посредством трех операций:

1) Отражение

2) Растяжение

3) Сжатие

На первом шаге выбираем три случайные точки и формируем треугольник. Вычисляем значение функции в каждой его вершине. Сортируем точки по значениям функции в этих точках.

На следующем шаге находим середину отрезка, точками которого являются две минимальные точки треугольника.

Далее применяем операцию отражения для наибольшей точки относительно середины отрезка.

Пытаемся найти точку со значением функции меньше, чем наибольшее значение функции в текущем треугольнике, применяя операции сжатия/растяжения.

Строим новый треугольник с найденной точкой.

Алгоритм останавливается, когда площадь треугольника достигла определенной точности.

Мы использовали реализацию этого метода из библиотеки *scipy*

4. Метод покоординатного спуска

Основная идея метода - идти параллельно осям координат в направлении уменьшения значения функции.

На первом шаге выбираем стартовую точку и начальную длину шага(step).

Начинаем идти по x координате. Смотрим значения функции в точках (x+step, y) и (x-step, y). Идём в ту сторону, где значение функции меньше текущего (в нашем решении, если оба значения меньше, идем в точку (x+step, y).

Если в обеих сторонах значения функции больше текущего, переходим к y координате и выполняем аналогичные действия для неё. Затем возвращаемся к x.

Если подряд по обеим координатам мы не нашли меньшего значения, уменьшаем step в два раза и продолжаем идти.

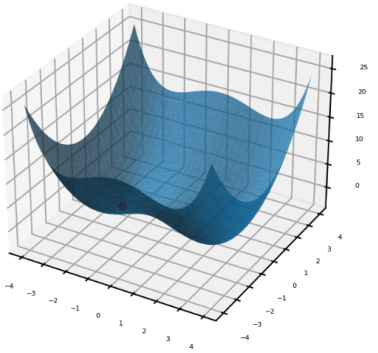
Завершаем алгоритм, когда step становится < eps

Метод реализован в файле *coordinate_descent.py*

Результаты исследования:

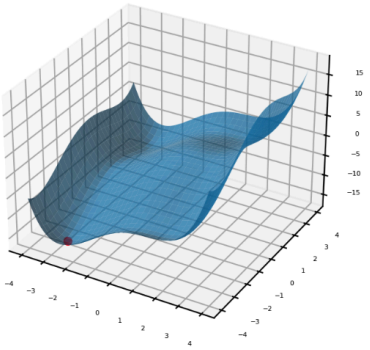
Мы провели исследование всех методов на N функциях:

1. $f_1(x, y) = y^2 - x^2 + \frac{x^4}{10}$



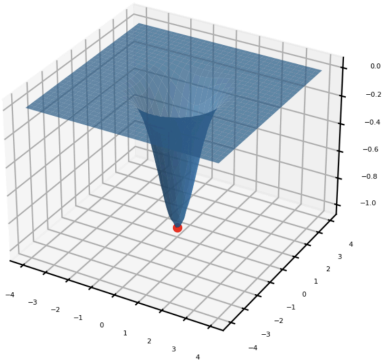
Данная функция весьма проста - в ней всего два локальных минимума.

2. $f_2(x, y) = -y^2 - x^2 + \frac{x^4}{10} + \frac{y^4}{20} + y + 2x$



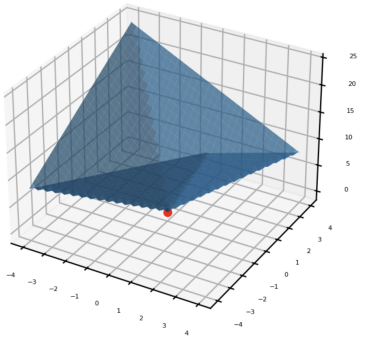
В данной функции - целых четыре локальных минимума, причем три из них - это достаточно маленькие “впадины”, а четвертая - самая глубокая - глобальный максимум.

3. $f_3(x, y) = -e^{-x^2-y^2}$



В данной функции всего один локальный - он же глобальный минимум. Однако интерес этой функции заключается в том, что на большинстве значений (x, y) - те, что находятся за окружностью с центром в (0, 0) и радиусом 10 - функция имеет очень маленький(по длине) градиент.

4. $f_4(x, y) = |x + y| + 3|y - x|$

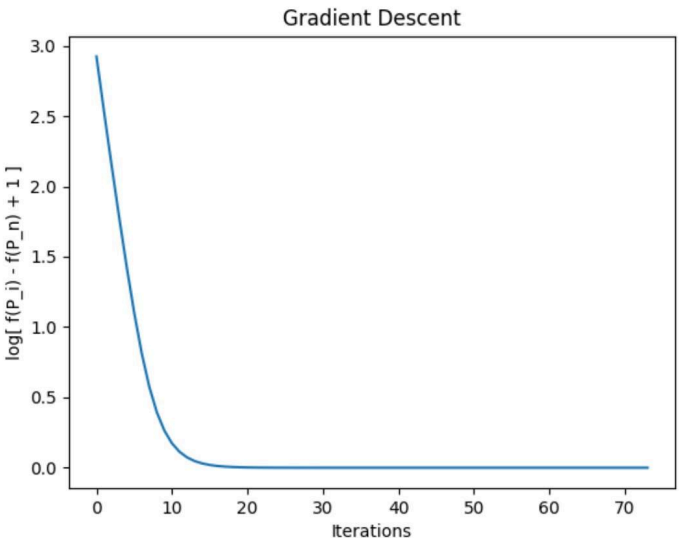


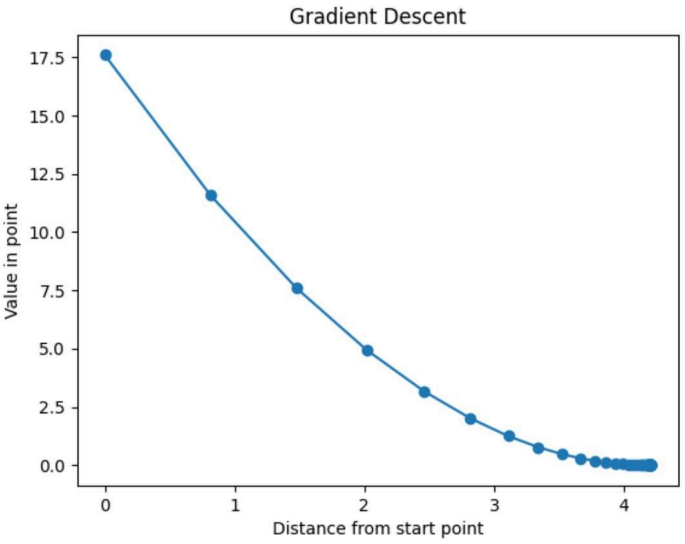
У данной функции - всего один минимум, но ее особенность заключается в том, что она не гладкая.

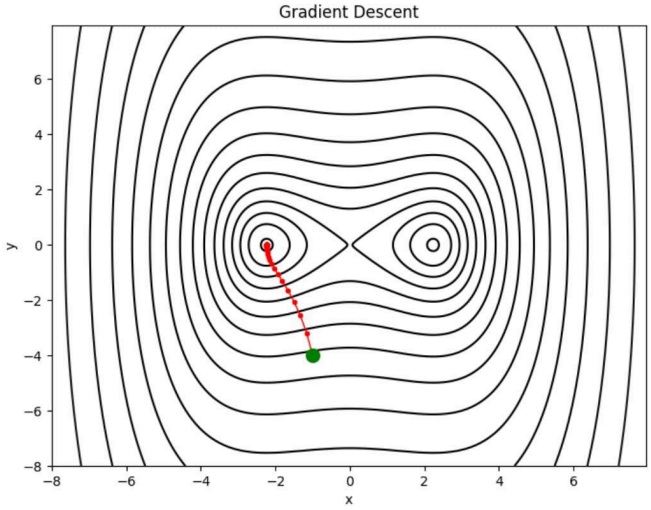
1. *<u>Метод градиентного спуска с постоянным шагом:</u>*

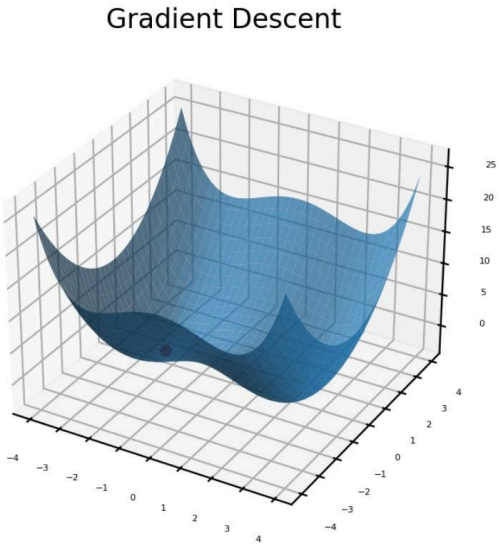
   a. *Пример работы(функция $f_1(x, y) = y^2 - x^2 + \frac{x^4}{10}$):*

Метод сходится к минимуму за ≈ 70 итераций, при этом на каждой итерации функция подсчитывается 4 раза в нашей реализации - для нахождения градиента в точке. Значит необходимо $\approx 70 \bullet 4 = 280$ вычислений исследуемой функции.



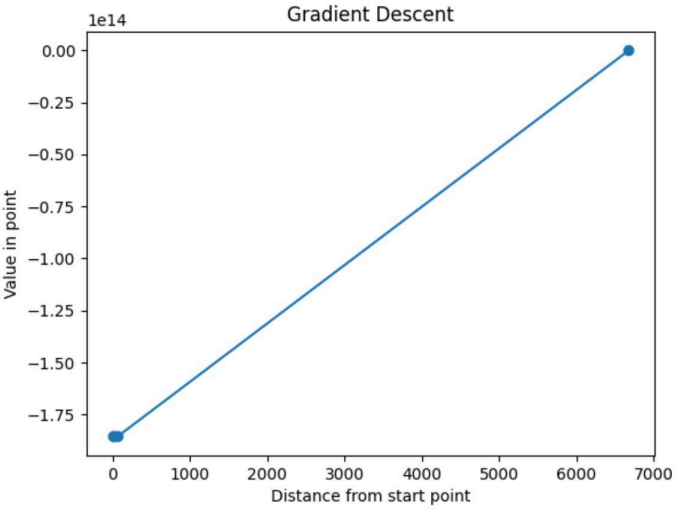






b. Пример неудачной работы(функция $f_1(x, y) = y^2 - x^2 + \frac{x^4}{10}$*):*

Как и было выше упомянуто в описании данного метода - он может расходиться, если взять слишком большой *learning rate,* в данном примере мы вместо 0.1, решили выставить шаг 0.2 - и метод ни разу не сошелся за 15 запусков из различных рандомных точек. В данном примере уже на третьей точке расстояние до стартовой точки становится больше 6000 - дальше смысла смотреть уже нет - будет только больше.

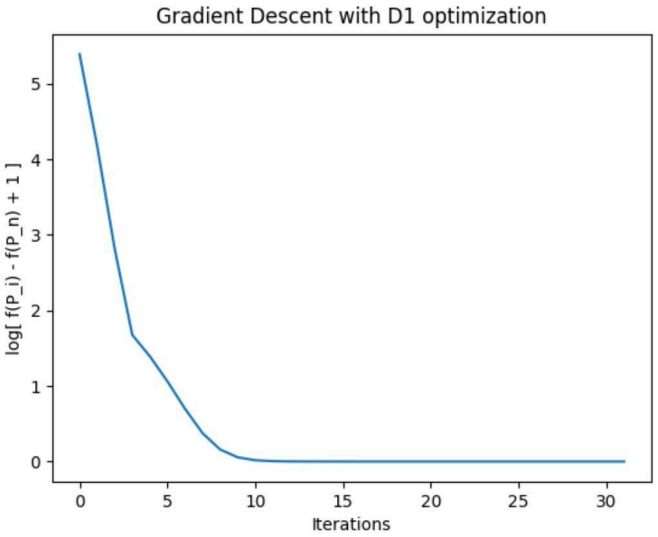


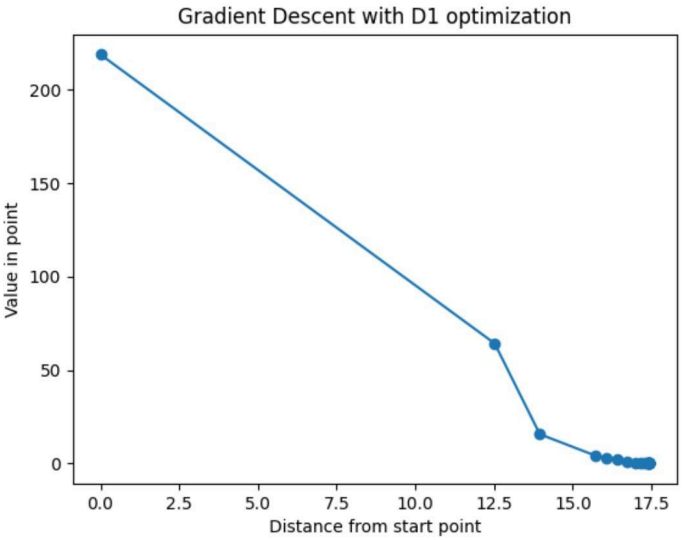
2. *<u>Метод градиентного спуска с модификацией(метод дихотомии):</u>*

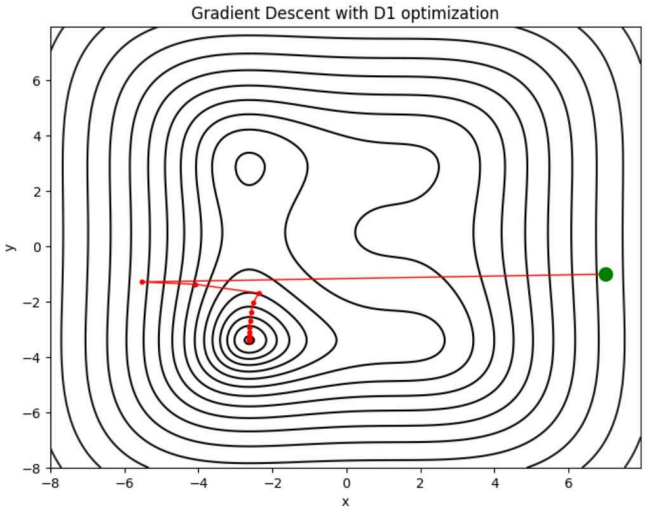
a. Пример работы (функция $f_2(x, y) = -y^2 - x^2 + \frac{x^4}{10} + \frac{y^4}{20} + y + 2x$*):*

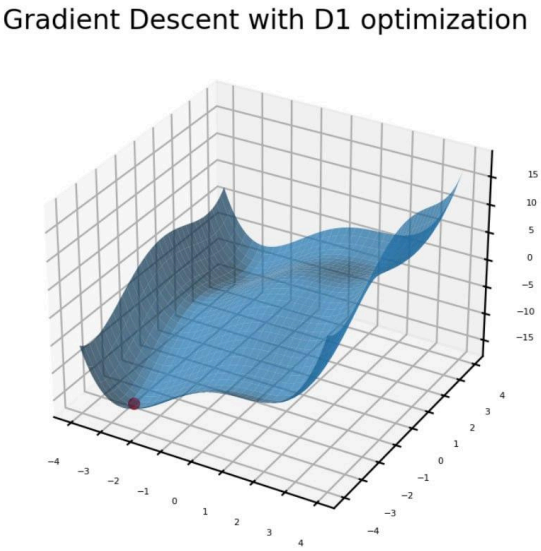
Метод сходится к минимуму за ≈ 30 итераций, при этом на каждой итерации функция подсчитывается 4 раза - для нахождения градиента в точке. И еще $\approx 30 \bullet 2$ раз в методе дихотомии (примерно 2 раза за итерацию необходимо

посчитать значение функции, всего итераций $\approx log_2 \frac{learning\ rate}{epsilon} \approx 30$ Значит необходимо $\approx 30 \bullet (30 \bullet 2 + 4) \approx 2000$ вычислений исследуемой функции.





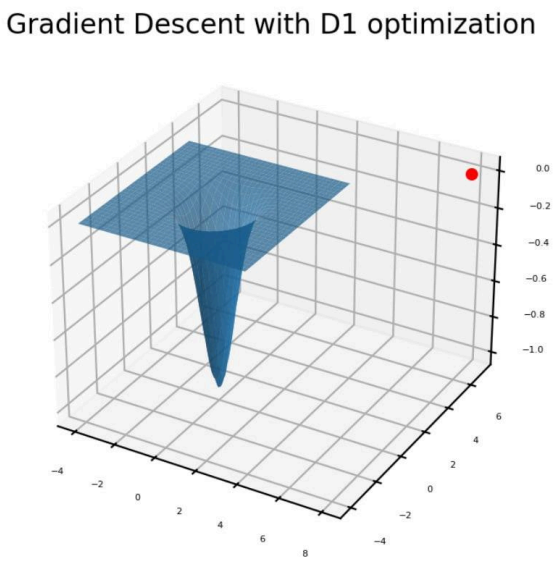




b. Пример неудачной работы(функция $f_3(x, y) = -e^{-x^2-y^2}$*):*

В отличие от предыдущего метода - в этом невозможно неверно выбрать *learning rate,* так как он изменяется в процессе выполнения данного алгоритма. Но у любого метода использующего градиентный спуск, возникнет проблема, если он попадет в точку, откуда в направлении антиградиента, следующая точка будет

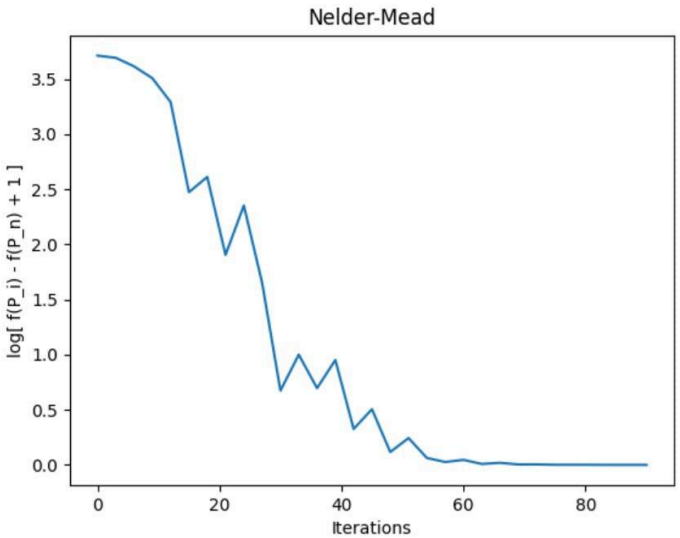
очень близко (на расстоянии меньше *epsilon*), то алгоритм завершится, но эта точка может быть не минимум, как в функции f_3 - у нее почти на всей плоскости значение градиента очень мало, из-за чего как раз таки любой градиентный спуск и валится.

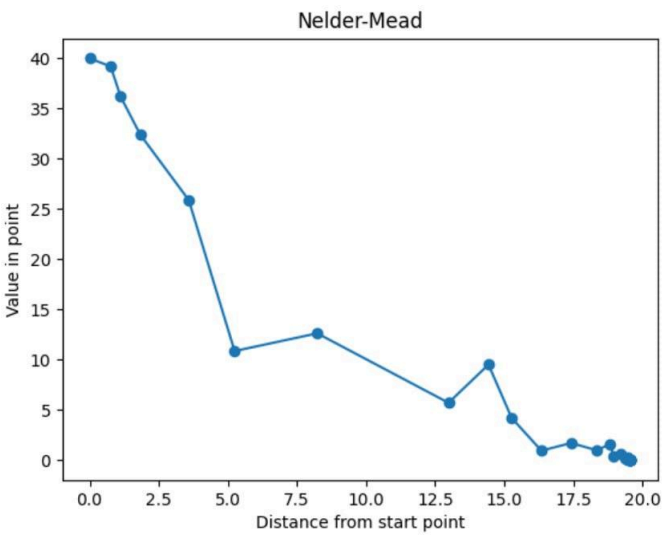


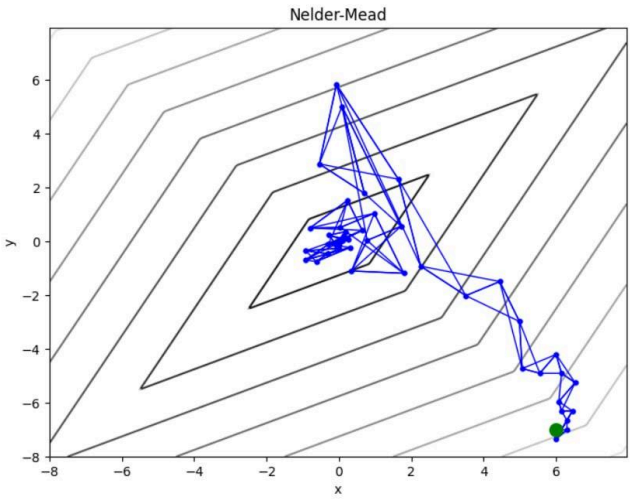
3. *Метод Нелдера-Мида*

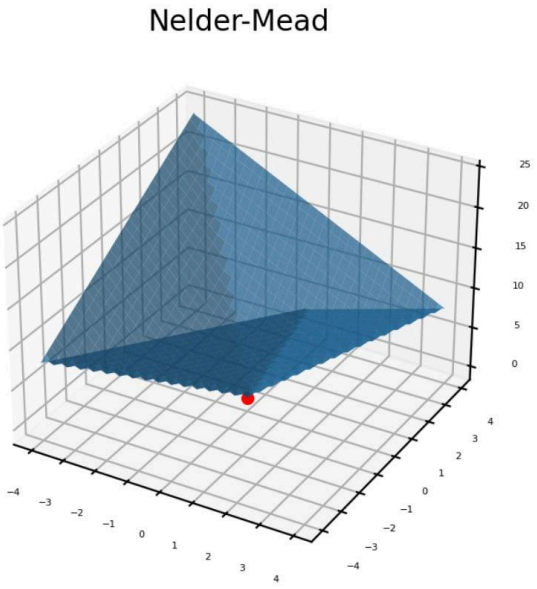
   a. *Пример работы (функция* $f_4(x, y) = |x + y| + 3|y - x|$*)*

Метод сходится к минимуму за ~30 итераций, при этом производя порядка 90 вычислений исследуемой функции, при этом не совершается никаких сложных вычислений. Достаточно неплохой результат относительно других исследуемых методов. Заметим что демонстрируемая функция не гладкая, при этом метод работает корректно, вообще говоря, данный алгоритм правильно вычислял минимум на всех исследованных функциях, что опять же выгодно выделяет его на фоне остальных методов.





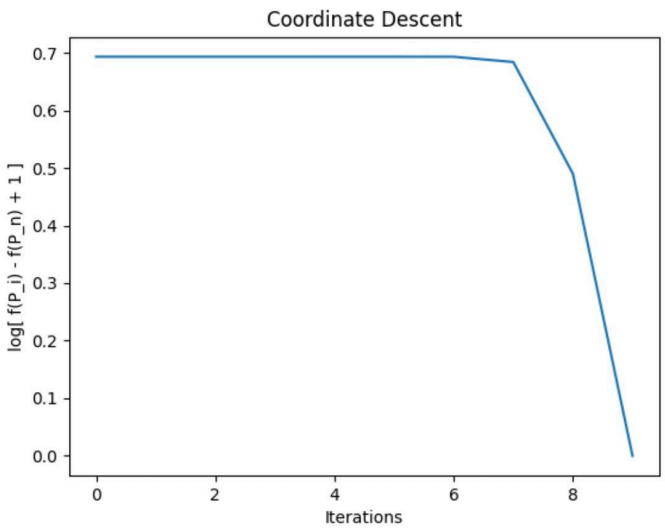


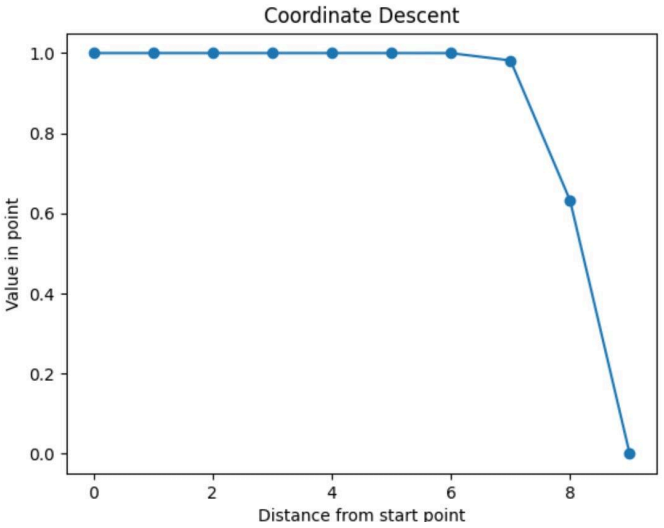


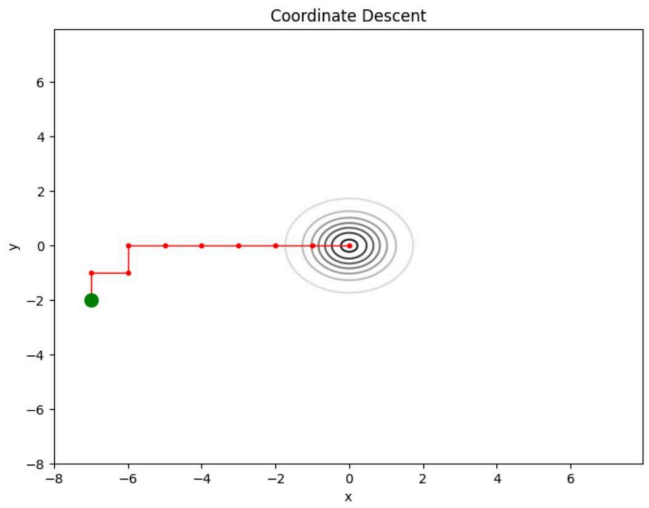
4. *Метод покоординатного спуска*

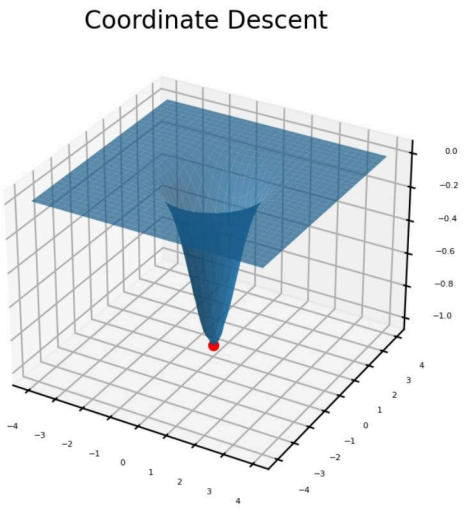
a. Пример работы (функция $f_3(x, y) = -e^{-x^2-y^2}$)

Метод правильно вычисляет минимум функции (с чем не справляются функции опирающиеся на градиент), при этом он совершает всего 9 итераций и порядка 36 вычислений, вообще говоря метод достаточно быстро сходится к инфинуму на простых функциях.



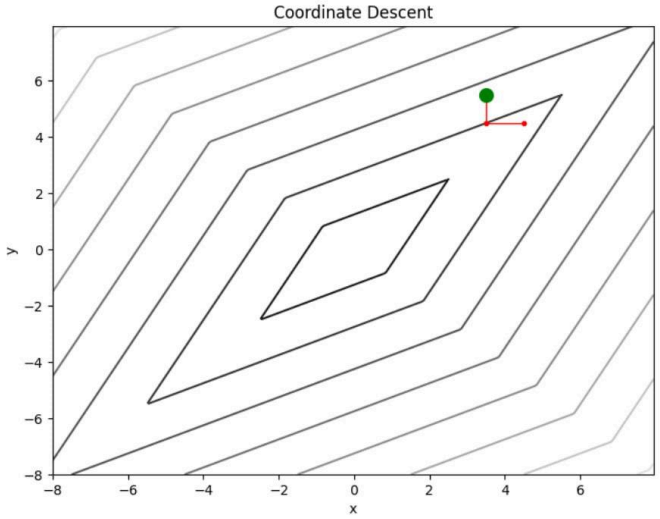






b. Пример неудачной работы (функция $f_4(x, y) = |x + y| + 3|y - x|$*)*

У данного метода есть проблемы с негладкими функциями, как видим он может упереться в нестационарную точку.



Анализ сходимости:

	f_1	f_2	f_3	f_4
Градиентный спуск	+/-	+	-	+
Градиентный спуск с дихотомией	+	+	-	+
Нелдер-Мид	+	+	+	+
Покоординатный спуск	+	+	+	-

Анализ результатов:

Метод	**Преимущества +**	**Недостатки —**
Градиентный спуск с постоянным шагом	• Прост в реализации • Мало вычислений	• Легко расходится • Неустойчив к функциям с маленьким градиентом
Градиентный спуск с использованием метода дихотомии	• Есть теория сходимости • Достаточно быстр (при быстровычисляемых функциях)	• Неустойчив к функциям с маленьким градиентом • Большое число вычислений
Метод Нелдера-Мида	• Совершает мало вычислений • Хорошо сходится	• Отсутствие теории сходимости
Метод покоординатного спуска	• Прост в реализации • Совершает мало вычислений	• Может застрять в нестационарной точке

Ссылка на репозиторий с кодом:

https://github.com/Sedromun/lab1-MetOpt