Домашнее задание №3.
Градиентный спуск (практическая часть)

Это практическое домашнее задание. Для его выполнения нужно будет реализовать нужные методы, а затем поставить эксперименты и сделать выводы. Отчет об экспериментах принимается в формате `Jupyter notebook`. Также нужно прислать код реализованных вами методов (файлы `oracles.py` и `methods.py`). В этом задании можно набрать 10 баллов.

Требования к отчету:

- На графиках должны быть подписаны оси, сами графики – проименованы. Если на одном графике несколько линий, используйте легенду, чтобы их различать. Проверяющему, который смотрит ваш отчет, должно быть понятно, что изображено на графике, даже если он не видел кода.
- Пояснения к экспериментам и выводы оформляйте с помощью `markdown`, а не в виде комментариев к коду.

1 Методы и целевые функции

1.1 Градиентный спуск

Рассмотрим задачу безусловной выпуклой оптимизации

$$f(x) \to \min_{x \in \mathbb{R}^n}, \tag{1}$$

где $f(x)$ – гладкая функция. Будем решать эту задачу с помощью итеративного метода

$$x_{k+1} = x_k + \alpha_k d_k.$$

Здесь d_k – это *направление поиска*, а α_k – *размер шага*. Схема метода может быть представлена в следующем виде.

В случае градиентного спуска направление поиска задается как $d_k = -\nabla f(x_k)$.

1.2 Линейный поиск

Как выбрать размер шага? Можно задать постоянный шаг, а можно пользоваться правилами **Армихо** или **Вульфа**. Рассмотрим функцию

$$\varphi_k(\alpha) = f(x_k + \alpha d_k), \ \alpha \geq 0.$$

Алгоритм 1 Общая схема итеративного метода оптимизации

Вход: Начальное приближение x_0, максимальное число итераций N.

1: **for** $k = 0, \dots, N$ **do**
2:     *Вызов оракула*: вычислить $f(x), \nabla f(x)$ и т.д.
3:     *Критерий остановки*: если выполнен критерий остановки, то выйти.
4:     *Вычисление направления*: вычислить направление поиска d_k.
5:     *Линейный поиск*: найти подходящую длину шага α_k.
6:     *Шаг*: $x_{k+1} = x_k + \alpha_k d_k$.
7: **end for**

Выход: Точка x_k.

Заметим, что это скалярная функция, производная которой дается выражением

$$\varphi_k'(\alpha) = \langle \nabla f(x_k + \alpha d_k), d_k \rangle .$$

Условием Армихо для размера шага α называется выполнение неравенства

$$\varphi_k(\alpha) \le \varphi_k(0) + c_1 \alpha \varphi_k'(0),$$

где $c_1 \in (0, 0.5)$ – некоторая константа. В случае градиентного спуска $\varphi_k'(0) = \langle \nabla f(x_k), -\nabla f(x_k) \rangle = -\left\| \nabla f(x_k) \right\|_2^2 \le 0$, и выполнение условия Армихо гарантирует $\varphi_k(\gamma) \le \varphi_k(0)$, т.е. функция $f(x)$ будет нестрого убывать на каждом шаге.

Для нахождения α, соответствующего условию Армихо, используют следующий алгоритм (метод дробления шага, или backtracking).

Алгоритм 2 Одномерный поиск по Армихо (backtracking)

Вход: Начальное приближение α_k^0, константа $c_1 \in (0, 0.5)$.

1: $\alpha := \alpha_k^0$
2: **while** $\varphi_k(\alpha) > \varphi_k(0) + c_1 \alpha \varphi_k'(0)$ **do**
3:     $\alpha := \frac{\alpha}{2}$
4: **end while**

Выход: Размер шага α.

В качестве начального приближения можно брать $\alpha_k^0 = 1$, либо использовать *адаптивный* способ. В последнем случае метод запоминает найденный на предыдущей итерации размер шага α_k и начитает поиск с $\alpha_{k+1}^0 = 2\alpha_k$.

Условия Вульфа накладывают более жёсткие ограничения на размер шага:

$$\begin{cases} \varphi_k(\alpha) \le \varphi_k(0) + c_1 \alpha \varphi_k'(0) \\ |\varphi_k'(\alpha)| \le c_2 |\varphi_k'(0)|. \end{cases}$$

Здесь $c_1 \in (0, 0.5)$, $c_2 \in (c_1, 1)$. Процедуру поиска по Вульфу *не нужно реализовывать самостоятельно*: воспользуйтесь функцией `scalar_search_wolfe2` из библиотеки `scipy.optimize.linesearch`.

1.3 Критерий остановки

Не обязательно выполнять заранее фиксированное число итераций для нахождения решения с заданной точностью ε. В идеале, можно было бы завершать алгоритм, как только выполнится условие $f(x_k) - f^* \leq \varepsilon$. Проблема в том, что величина f^* не известна. Поэтому вместо невязки по функции используют норму градиента.

В этом задании используйте следующий критерий:

$$\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2} \leq \varepsilon. \tag{2}$$

Этот критерий задает *относительную точность* решения благодаря нормировке на $\|\nabla f(x_0)\|_2^2$.

1.4 Квадратичная функция

Рассмотрим матрицу $A \in \mathbb{S}^n_{++}$ и вектор $b \in \mathbb{R}^n$. зададим функцию

$$f(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle. \tag{3}$$

Её число обусловленности зависит от матрицы A и равняется $\kappa = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$. В задании вам предстоит исследовать поведение градиентного спуска в зависимости от числа обусловленности.

Сгенерировать случайную квадратичную задачу с заданным κ можно, например, так: взять случайные числа $\lambda_1, \ldots, \lambda_n \in [1, \kappa]$, так что $\min\limits_i \lambda_i = 1$, $\max\limits_i \lambda_i = \kappa$, и положить $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. Элементы вектора b можно взять произвольными, они на обусловленность не влияют.

В случае двумерной функции ($x \in \mathbb{R}^2$) можно взять ортогональную матрицу вида

$$S = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix}$$

и задать $A = S \cdot \mathrm{diag}(1, \kappa) \cdot S^\top$. Получится "повернутая" квадратичная функция.

2 Задание

Скачайте код из репозитория `https://github.com/alexrogozin12/made_opt/tree/master/homework_3`. В нём содержатся заготовки методов, которые вам предстоит реализовать.

2.1 Методы и оракулы (4 балла)

1. Реализуйте недостающие методы `.func` и `.grad` в оракуле `QuadraticOracle` в модуле `oracles.py`.

2. Реализуйте методы одномерного поиска, соответствующие правилу Армихо и Вульфа: класс `LineSearchTool` в модуле `methods.py`. Для правила Армихо см. алгоритм 2, а для правила Вульфа воспользуйтесь функцией `scalar_search_wolfe2` из библиотеки `scipy.optimize.linesearch`. У этой библиотечной функции есть одна проблема: метод может не сойтись и вернуть None. Если это произошло, запустите backtracking (алгоритм 2).

3. Реализуйте метод градиентного спуска: класс `GradientDescent` в модуле `methods.py`. Алгоритм должен поддерживать постоянную длину шага, а также одномерный поиск по Армихо и Вульфу. Сделайте это, используя `LineSearchTool` из предыдущего пункта задания. В случае поиска по Армихо пользуйтесь **адаптивным подбором шага** (см. п.1.2).

   В ходе работы алгоритм должен также сохранять историю в поле `.hist`. Обратите внимание на формат этого поля, приведенный в шапке метода `.run`.

2.2 Траектория градиентного спуска на квадратичной функции. (3 балла)

Задайте две-три двумерные квадратичные функции с разными числами обусловленности. Запустите на них GD с различными стратегиями выбора шага, изобразите на графиках траектории методов и линии уровня функции. Для рисования линий уровня воспользуйтесь функцией `plot_trajectory`, а для траекторий методов – функцией `plot_levels` из файла `plot_trajectory_2d`.

Постарайтесь ответить на вопрос: как зависит поведение методов от числа обусловленности, от начальной точки, от стратегии выбора длины шага?

2.3 Зависимость числа итераций градиентного спуска от числа обусловленности и размерности пространства (3 балла)

Исследуйте, как зависит число итераций, необходимое GD для сходимости, от

- Числа обусловленности целевой функции κ;
- Размерности пространства n.

Для данных параметров n и κ сгенерируйте случайную квадратичную задачу размерности n с числом обусловленности κ. Это можно сделать, как описано в пункте 1.4. На этой задаче запустите метод градиентного спуска с вашей любимой стратегией выбора шага и измерьте число итераций $N(n, \kappa)$, необходимое для достижения точности ε.

Фиксируя n и варьируя κ, постройте график $N(n, \kappa)$ против κ. Так как квадратичные задачи генерируются случайно, проведите этот эксперимент несколько раз и получите семейство кривых $T(n)$. Изобразите эти кривые на одном графике.

Постройте такие семейства кривых для разных n (можно перебирать n по логарифмической сетке, например, $n \in [10, 100, 1000]$). Получится несколько наборов кривых: часть красных (для $n = 10$), часть зелёных (для $n = 100$), часть синих (для $n = 1000$) и т.д.

Сделайте выводы. Как полученные результаты согласуются с теорией, т.е. с теоретическими оценками на число итераций GD?