Оптимизация

Цель занятия

В результате обучения на этой неделе:

- вы повторите функции и их свойства;
- вспомните, как находить производные функции, что такое экстремумы и критические точки функции;
- узнаете, что такое градиент функции;
- познакомитесь с методами оптимизации;
- научитесь решать задачи оптимизации градиентными методами.

План занятия

Используемые термины

Функция — отображение аргумента *x* в значение *y*.

Область определения — множество значений аргумента, для которых определена функция.

Область значений — множество значений *f(x)* для всех значений аргумента *x*.

Конспект занятия

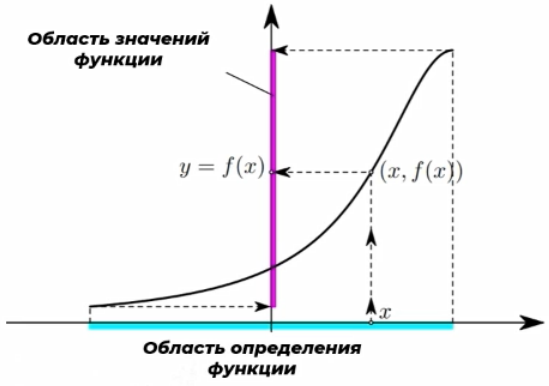
1. Производная и ее применение

Определение функции

Функция задает отображение аргумента x в значение y.

Область определения — множество значений аргумента, для которых определена функция.

Область значений — множество значений f(x) для всех значений аргумента x.



Функции могут быть:

- скалярные — отображение в число;
- векторные — отображение в вектор.

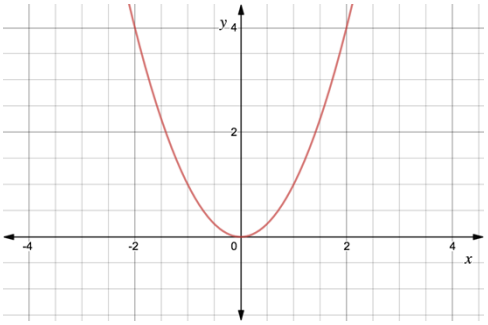
Результатом воздействия функции на какой-либо объект может быть что угодно — тензор, матрица, граф и др.

Как правило, мы будем работать с линейными пространствами, где объектами являются векторы. Поэтому чаще всего функции у нас будут скалярные или векторные (скаляр — одномерный вектор).

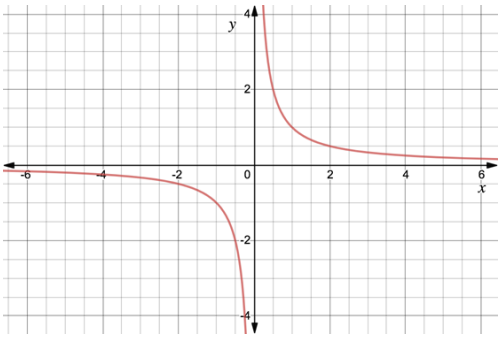
Свойства функции

Непрерывность:

- Непрерывной будем называть функцию, график которой мог бы быть проведен без отрыва пера (это неформальное определение).
- Также можно отметить, что если непрерывная функция принимает два значения, то она также принимает все значения между ними.



Непрерывная функция, парабола x^2. *Источник*



Функция, не являющаяся непрерывной, гипербола $\frac{1}{x}$. *У нее разрыв на нуле. Источник*

Теорема о срединной точке, о промежуточном значении. Если у нас есть две точки, в которых непрерывная функция принимает значения A и B, то все значения между A и B она также должна «пробегать» при переходе через аргументы от A до B.

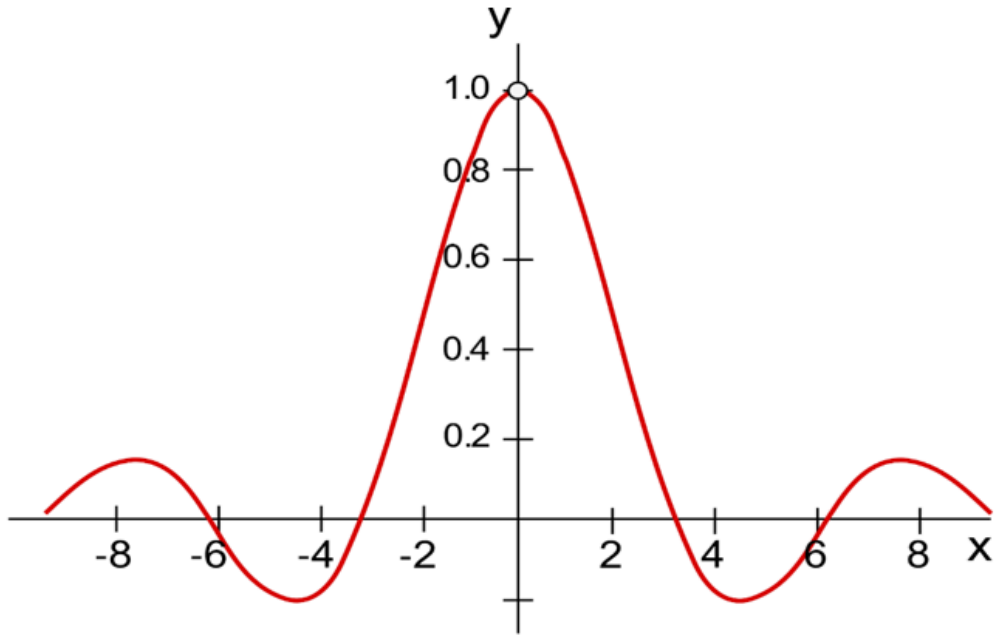
Формальное определение непрерывной функции. Функция $f(x)$ называется непрерывной для всех x_0 из области определения, если

$$\lim_{x \to x_0} f(x) = f(x_0).$$

Если в каждой точке предел функции совпадает с ее значением в этой точке.

Задание. Повторить определение предела функции по Коши и по Гейне.

Пример. $f(x) = \frac{\sin x}{x}$



Функция $f(x) = \frac{\sin x}{x}$ не определена в $x_0 = 0$, но $\lim\limits_{x \to 0} \frac{\sin x}{x} = 1$. То есть

$f(x) = \begin{cases} \frac{\sin x}{x}, & x \neq 0 \\ 1, & x = 0 \end{cases}$ является непрерывной функцией.

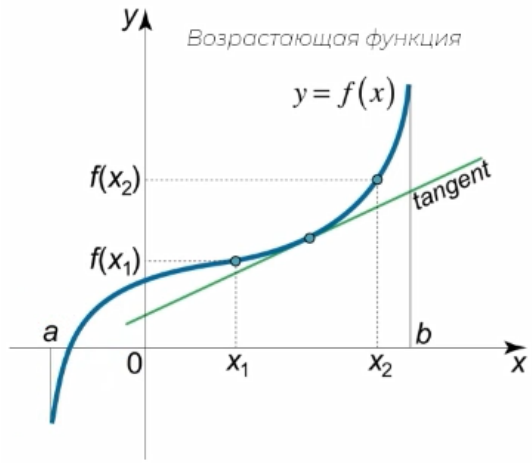
NB. Как найти предел функции? Достаточно найти расходящуюся подпоследовательность в значении функции. Тогда сама функция не имеет предела.

Если есть непрерывная функция, значит, можно задать непрерывное отображение либо на ограниченной области определения, на ограниченном домене, либо на всей области действительных чисел *R*. Функция может считаться непрерывной только на определенном отрезке.

Возрастание / убывание. Функция может быть непрерывной, но при этом она может вести себя монотонно и не монотонно.

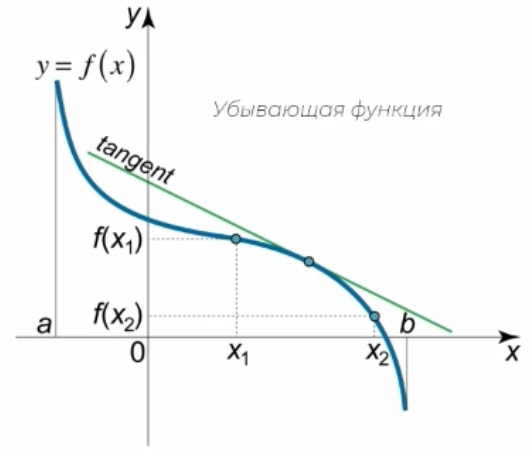
Монотонной функцию называют, если она возрастает / убывает монотонно.

То есть у возрастающей функции каждое следующее значение строго больше предыдущего:



Возрастающая функция $f(x_2) > f(x_1)\ \forall x_2 > x_1 \in [a; b]$. *Источник*

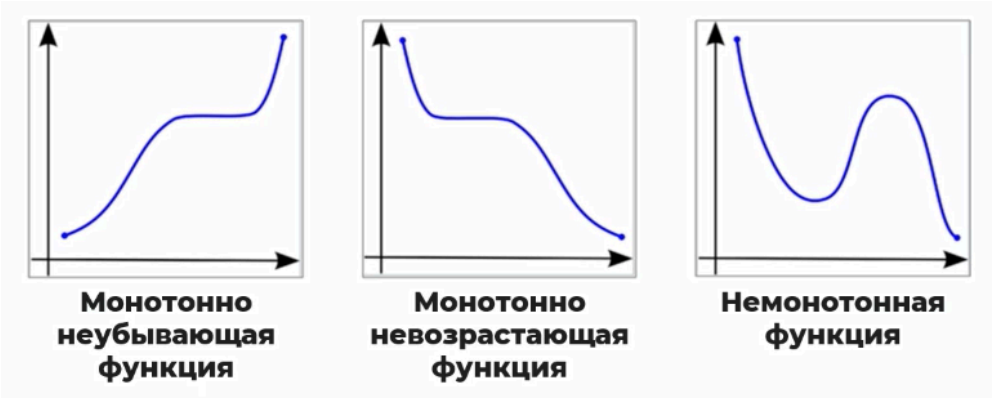
А у убывающей — строго меньше:



Убывающая функция $f(x_2) < f(x_1)\ \forall x_2 < x_1 \in [a; b]$. *Источник*

Про монотонно возрастающую / убывающую функцию можно сказать через понятие их производной.

Монотонность. Монотонная функция (не)убывает или (не)возрастает на всей области определения.



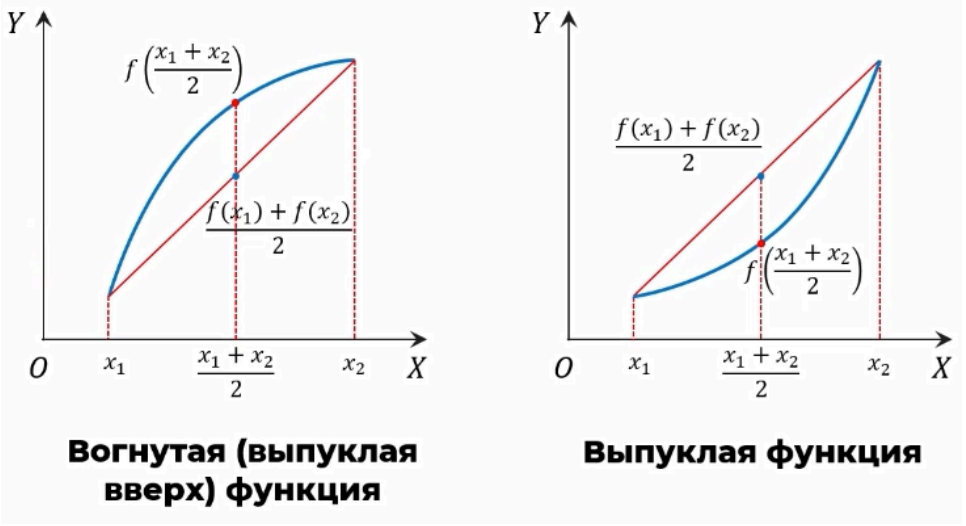
Монотонные функции. [Источник]

Самая большая проблема для нас — немонотонные функции. Если функция немонотонная, она не является выпуклой (вверх или вниз), и тогда градиентный метод оптимизации нам не гарантирует нахождение глобального экстремума. А это проблема задачи оптимизации. Практически все задачи машинного обучения сводятся к оптимизационной задаче.

Выпуклость. Выпуклой (вверх или вниз) функция $f(x)$ называется, если

$f\left(\frac{x_1+x_2}{2}\right) \geq \frac{f(x_1)+f(x_2)}{2}$ — выпуклая вверх. (1)

$f\left(\frac{x_1+x_2}{2}\right) \leq \frac{f(x_1)+f(x_2)}{2}$ — выпуклая вниз. (2)

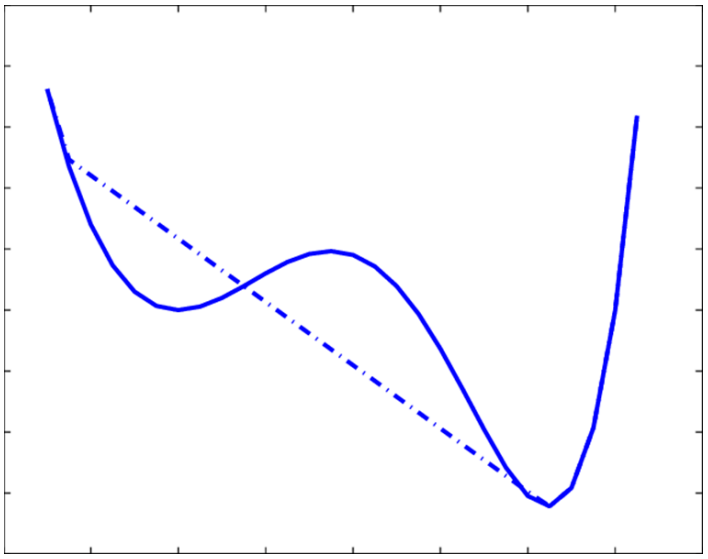


Выпуклые вверх и вниз функции. [Источник]

Если в формулах (1) или (2) выполняется равенство, значит, эта функция прямая. Задает некоторое линейное отображение, и тогда говорить про выпуклость или вогнутость не имеет смысла, производная у функции — константа.

В разных источниках выпуклую и вогнутую функции называют по-разному.

В невыпуклой функции несколько локальных экстремумов:



На рисунке видно, что у этой невыпуклой функции есть глобальный минимум.

Производная

Градиент — это направление наискорейшего возрастания / убывания функции. В общем случае градиент — это вектор из частных производных по всем частям аргумента (если аргумент векторный).

Производная определяется как:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x)-f(x)}{\Delta x} \quad (3)$$

Свойство производной. Производная функции f в точке x указывает, насколько сильно изменится f при небольшом изменении x на величину Δx:

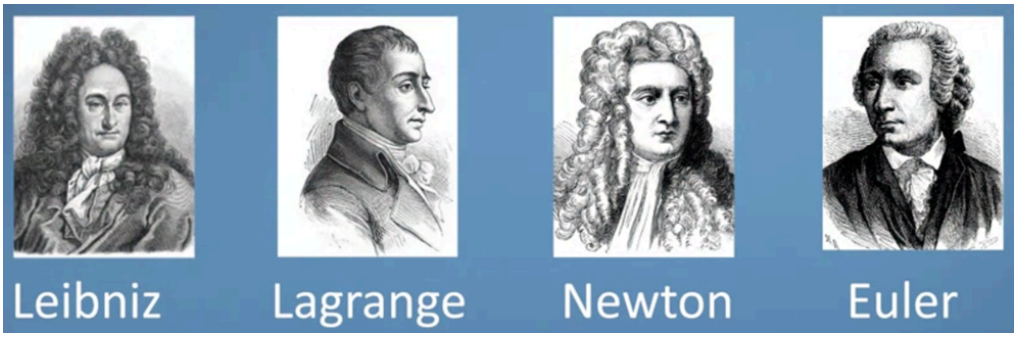
$$f(x + \Delta x) \approx f(x) + \Delta x \cdot f'(x) \quad (4)$$

Формула (4) очень похожа на ряд Тейлора, и в некоторой окрестности мы можем описывать изменение функции через ее производную.

Пример:

$$\left(\frac{1}{x}\right)' = \lim_{\Delta x \to 0} \frac{\frac{1}{x+\Delta x}-\frac{1}{x}}{\Delta x} = \lim_{\Delta x \to 0} \frac{-\Delta x}{\Delta x \cdot x(x+\Delta x)} = \lim_{\Delta x \to 0} \frac{-1}{x^2+x\Delta x} = -\frac{1}{x^2}$$

Различные формы записи производной:



$f'(x)$ $= f'_x(x)$ $= \frac{d}{dx} f(x)$ $= \frac{\partial}{\partial x} f(x)$

Производные некоторых функций:

$(c)' = 0 \ \ (c = const),$ $(x^\alpha)' = \alpha x^{\alpha-1},$

$(e^x)' = e^x,$ $(a^x)' = a^x \ln a,$

$(\ln x)' = \frac{1}{x},$ $(\log_a x)' = \frac{1}{x \ln a},$

$(\sin x)' = \cos x,$ $(\cos x)' = -\sin x,$

$(\mathrm{tg}\, x)' = \frac{1}{\cos^2 x},$ $(\mathrm{ctg}\, x)' = -\frac{1}{\sin^2 x},$

$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}},$ $(arccos\ x)' = -\frac{1}{\sqrt{1-x^2}},$

$(\mathrm{arctg}\, x)' = \frac{1}{1+x^2},$ $(arcctg\ x)' = -\frac{1}{1+x^2},$

При подсчете производной следует пользоваться табличными значениями производных некоторых функций, а также правилами взятия производной.

Есть множество автоматически дифференцирующих библиотек, движков. Нейронные сети по факту являются огромными дифференцируемыми функциями. Они оптимизируются подсчетом производной.

Правила взятия производной

Производная суммы:

$$[u(x) + v(x)]' = u'(x) + v'(x)$$

Пример.

$$(x^2 + x^3)' = 2x + 3x^2$$

Производная произведения:

$$[u(x) \cdot v(x)]' = u'(x) \cdot v(x) + u(x) \cdot v'(x)$$

Пример.

$$(xe^x)' = 1 \cdot e^x + x \cdot e^x$$

$$\left(\frac{1-x}{x}\right)' = (1 - x) \cdot \frac{1}{x} = -\frac{1}{x} - \frac{1-x}{x^2}$$

Производная сложной функции:

$$f(g(x))' = f'(g(x)) \cdot g'(x)$$

Другая форма записи:

$$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$$

Пример.

$$\left(\frac{1}{1-x}\right)' = -\frac{1}{(1-x)^2} \cdot (1 - x)' = \frac{1}{(1-x)^2}$$

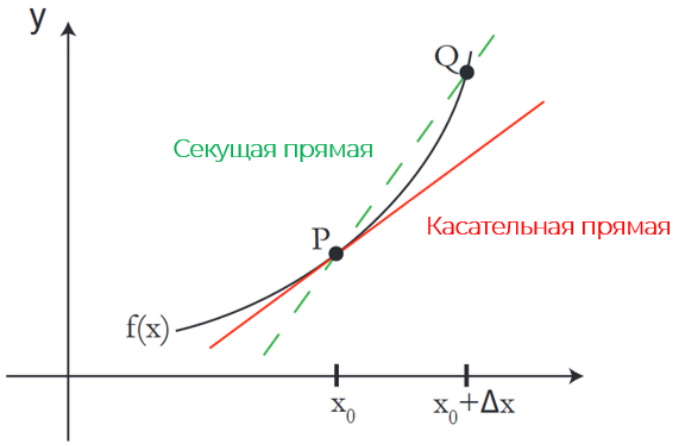
$$\left(e^{x^2}\right)' = e^{x^2} \cdot \left(x^2\right)' = e^{x^2} \cdot 2x$$

Производная отношения:

$$\left(\frac{u(x)}{v(x)}\right)' = \left(u(x) \cdot \frac{1}{v(x)}\right)' = u'(x) \cdot \frac{1}{v(x)} - u(x) \cdot \frac{1}{(v(x))^2} \cdot v'(x) = \frac{u'(x)v(x) - u(x)v'(x)}{(v(x))^2}$$

Геометрический смысл производной

Геометрический смысл производной — это касательная к графику функции в точке взятия производной.

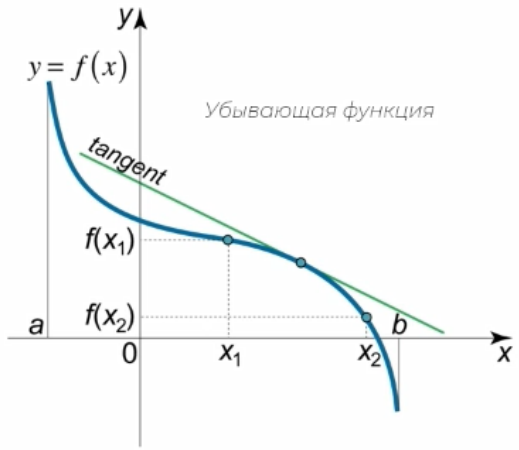


На рисунке представлена функция выпуклая, возрастающая. Ее производная будет положительная.

Анализ функции с помощью производной

Вернемся к возрастающей функции $f(x_2) > f(x_1)\ \forall x_2 > x_1 \in [a; b]$:

И к убывающей функции $f(x_2) < f(x_1)\ \forall x_2 < x_1 \in [a; b]$:



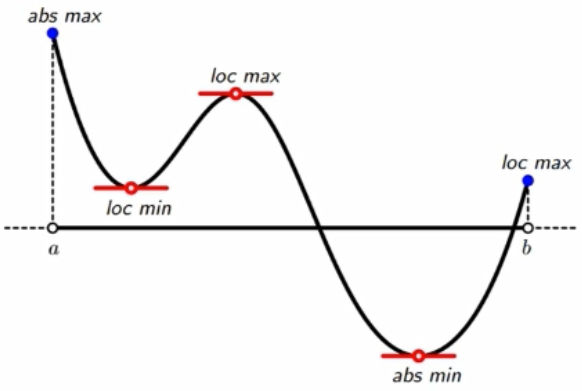
Из первого рисунка получаем $f'(x) > 0 \ \forall x \in [a, b]$.

Из второго получаем $f'(x) < 0 \ \forall x \in [a, b]$.

Непрерывно дифференцируемой функция является, если у нее в каждой точке существует производная.

Экстремум функции

Функция $f(x)$ достигает **локального минимума (максимума)** — экстремума — в точке x_0, если $f(x_0)$ принимает наименьшее (наибольшее) значение в окрестности x_0.



Пример функции с локальными и глобальными экстремумами

Функция $f(x)$ достигает **глобального минимума (максимума)** в точке x_0, если $f(x_0)$ является наименьшим (наибольшим) значением $f(x)$.

Критические точки

Стационарной точкой функции $f(x)$ называют точку x_0, если $f'(x_0) = 0$

Критической точкой функции $f(x)$ называют точку x_0, если:

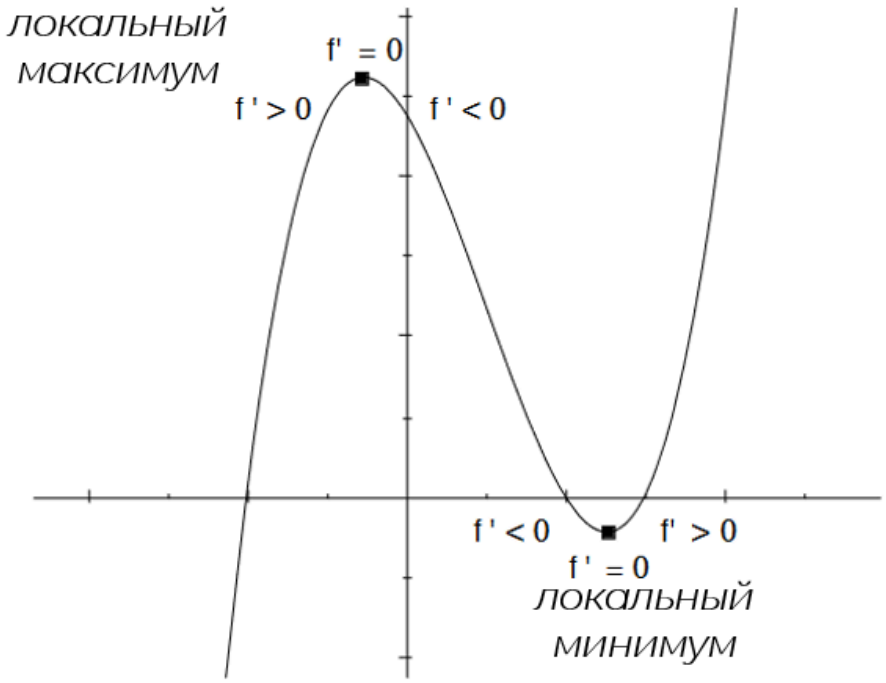
- либо $f'(x_0) = 0$ (x_0 является стационарной точкой);
- либо $f'(x_0)$ не определена.

Анализ критических точек

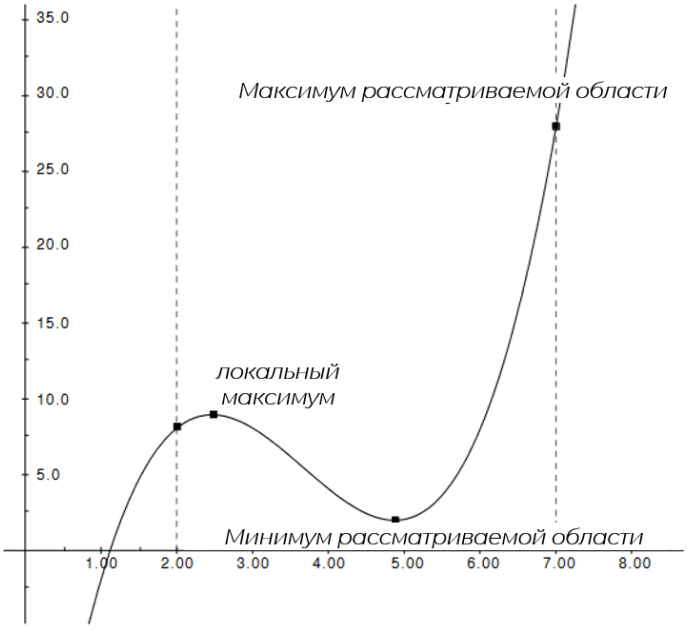
Пусть x_0 является критической точкой $f(x)$.

Если $f'(x) < 0$ при $x < x_0$ и $f'(x) > 0$ при $x > x_0$, то x_0 — локальный минимум.

Если $f'(x) > 0$ при $x < x_0$ и $f'(x) < 0$ при $x > x_0$, то x_0 — локальный максимум.



Важно! Не забывайте про края области определения. Там производная не совсем определяется, но могут достигаться экстремумы:



2. Градиентная оптимизация

Введем понятие градиента. Пусть у нас задана некоторая непрерывно дифференцируемая функция $f(x)$.

$x \in R^n$ — некоторый действительный n-мерный вектор.

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Пусть $f(x) = y \in R$ — функция, которая отображает вектор в число, то есть **функционал**.

Что значит непрерывно дифференцируемая функция? Функция $g(x) = \frac{df}{dx}$ должна быть непрерывной на R^n.

$x \in \chi \subset R^n$, χ — подмножество R^n**.** Тогда функция $f(x)$ должна быть непрерывно дифференцируема на всем χ.

Градиент

Градиент функции — это вектор частных производных функционала по каждой из координат его аргумента:

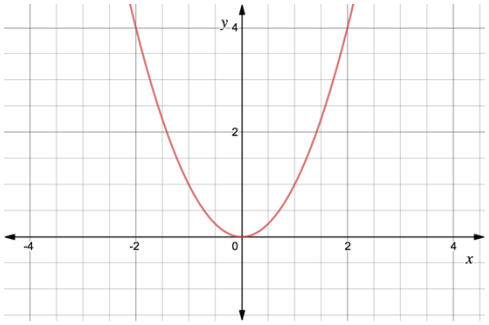
$$grad\ f(x) = grad_x f(x) = \nabla_x f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Этот вектор будет куда-то направлен и иметь какую-то величину.

Свойства градиента

Градиент — это направление наискорейшего возрастания функции.

Рассмотрим параболу $f(x) = x^2$:



$\frac{\partial f}{\partial x} = 2x$ — одномерный случай.

Слева от нуля у нас $2x < 0$, справа от нуля $2x > 0$. То есть слева от нуля нам нужно двигаться влево, чтобы скорей расти. Двигаясь вправо, мы получим экстремум. Справа от нуля все наоборот.

Чем больше норма вектора

$$\|\nabla_x f(x)\|,$$

тем быстрее будет функция расти в направлении градиента.

Градиентная оптимизация

В большинстве случаев мы будем пользоваться градиентной оптимизацией первого порядка.

Пусть у нас есть какая-то поверхность (в общем случае гиперповерхность) функции потерь. У нас есть некоторая модель, которая задает отображение:

$$f_w: x \to y.$$

w — параметр, от которого зависит функция f.

Функция потерь задает отображение:

$L: y \times y \to R$.

y — истинное значение,

$\hat{y}$ — предсказание.

Пусть f зависит от параметра w. Тогда:

$$L(y, \hat{y}) = L\big(y, f_w(x)\big) \to \min_w$$

$$w_0 \sim N(0;\ \sigma^2)$$

Для каждого следующего w мы можем ввести правило

$$w_{t+1} = w_t - \alpha \nabla_w L\big(y, f_w(x)\big) \tag{5}$$

α — градиентный шаг, learning rate. Показывает, насколько быстро мы должны двигаться в данном направлении.

Правило (5) будем использовать повсеместно с линейными моделями, с логистической регрессией, с задачей для SVM, с нейронными сетями.

Градиенты второго порядка используются редко. Чем больше в градиенте параметров, тем больше будет частных производных по этим параметрам, тем сложнее считать.

Все функции потерь всегда должны возвращать число. Поэтому всегда говорим про оптимизацию единственного функционала. А многокритериальная оптимизация, когда множество подзадач, остается в особых ситуациях или сводится к задаче оптимизации с ограничениями.

3. Условная оптимизация

Пусть у нас ограничения будут формата равенства. Пусть $f_w: x \to y$.

У нас также есть функция потерь: $Q: y \times y \to R$.

$Q(y, \hat{y})=\varepsilon$

Наша задача — минимизировать функцию Q:

$$Q\big(y, f_w(x)\big) = Q(y, \hat{y}) \to \min_w$$

Но при этом у нас есть ограничение $I_i(w) = 0,\ i = 1, k$. То есть у нас k ограничений.

Пример. Допустим, у нас есть выборка в 10 объектов. Какие-то объекты синие, какие-то зеленые:

Мы бы хотели посчитать энтропию этой выборки: $H = -\sum_k p_k log\ p_k$.

$p_{\text{синих}} = 0,6,\ p_{\text{зеленых}} = 0,4.$

Мы бы хотели максимизировать энтропию:

$$H \to \max_p$$

И при этом у нас есть ограничение: мы знаем, что $\sum_{i=1}^{k} p_i = 1$. Мера всех возможных исходов должна суммарно быть равна 1: $\forall i\ p_i \in [0;\ 1]$.

Перепишем в чуть другом формате наше ограничение: $1 - \sum_i p_i = 0$.

Теперь можем ввести **функцию Лагранжа** (мы используем **метод множителей Лагранжа**):

$L(p, \lambda) = H + \lambda(1 - \sum_i p_i)$ – это наш функционал.

$$L(p, \lambda) = -\sum_k p_k log\ p_k + \lambda(1 - \sum_i p_i) \qquad (6)$$

Функция Лагранжа обладает любопытным свойством: экстремумы функции Лагранжа совпадают с экстремумами изначальной функции. Мы найдем точки-претенденты на оптимум нашей функции, поскольку равенство нулю производной является необходимым, но недостаточным условием для экстремума.

У нашего логарифма нет никакого основания, поскольку энтропию мы можем оптимизировать независимо от основания.

Продифференцируем (6):

$$\frac{\partial L}{\partial p_i} = -\log p_i - 1 - \lambda(0-1) == -\log p_i - 1 + \lambda = 0$$

$log\ p_i = \lambda - 1\ \forall i$

Второе ограничение:

$$\frac{\partial L}{\partial \lambda} = 0$$

$$\sum_{i=1}^{k} p_i = 1$$

Следовательно, $p_i = e^{\lambda-1}$.

$$\sum_{i=1}^{k} e^{\lambda-1} = 1 = ke^{\lambda-1},\ e^{\lambda-1} = \frac{1}{k} = p_i.$$

Точка экстремума:

$\lambda = log\frac{1}{k} = -\ log\ k.$

$\frac{1}{k}$ — экстремум. Значит, для данной точки экстремума мы нашли точку максимума.

Мы решили задачу условной оптимизации и учли ограничения.

4. Решение задачи оптимизации градиентными методами

Метод градиентной оптимизации — практически единственный метод, доступный для огромного класса задач, где оптимизируемый функционал является дифференцируемой функцией. При этом градиентные методы работают достаточно быстро по сравнению с другими и дают вполне неплохое решение.

Можно в Jupiter Notebook работать локально или запустить в облаке:

```
'''

If you are using Google Colab, uncomment the next line to download
`utils_02.py`.

'''

# !wget
```

```
https://raw.githubusercontent.com/girafe-ai/ml-course/22f_basic/week0_02_linear_reg/utils_02.py
```

Сделаем все необходимые import:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import warnings
warnings.filterwarnings("ignore")


random_seed = 45


matplotlib.rcParams.update({'font.size': 16})
```

Градиентный спуск

Предполагаем, что у нас есть некоторые значения наших параметров на текущий момент t. Мы имеем некоторый функционал Q, который хотим минимизировать. И есть шаг спуска η — learning rate:

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla Q\left(w^{(t)}\right)$$

Будем рассматривать на примере линейной регрессии:

$$\hat{Y} = Xw$$

И минимизировать среднеквадратичную ошибку MSE:

$$Q(Y,\ X,\ w) = MSE(Y,\ Xw) = ||Y - Xw||_2^2 = \sum_i \left(y_i - x_i^T w\right)^2$$

Линейная операция явно дифференцируема, MSE тоже дифференцируема, поэтому можем посчитать градиент нашей функции потерь по параметру:

$$\nabla Q(w) = -2X^T Y + 2X^T Xw = 2X^T(Xw - Y)$$

Здесь можно заметить алгоритмическую сложность вычисления градиента $O(pN)$, где p — размерность пространства, N — количество объектов.

Можно сделать еще проще, если использовать стохастический градиентный спуск. В этом случае используется подвыборка, и тогда сложность уменьшается до $O(pK)$, $K << N$.

Решение задачи

Попробуем решить задачу с помощью градиентного метода.

Инициализируем данные:

```
n_features = 2

n_objects = 300

batch_size = 10

num_steps = 43

np.random.seed(random_seed)


# Let it be the *true* weights vector

w_true = np.random.normal(size=(n_features, ))


X = np.random.uniform(-5, 5, (n_objects, n_features))


# For different scales of features. In case of 3 features the code is equal
to the commented line below

# X *= np.arange([1, 3, 5])[None, :]

# X *= (np.arange(n_features) * 2 + 1)[np.newaxis, :]


# Here comes the *true* target vector

Y = X.dot(w_true) + np.random.normal(0, 1, n_objects)
```

Запускаем градиентный спуск:

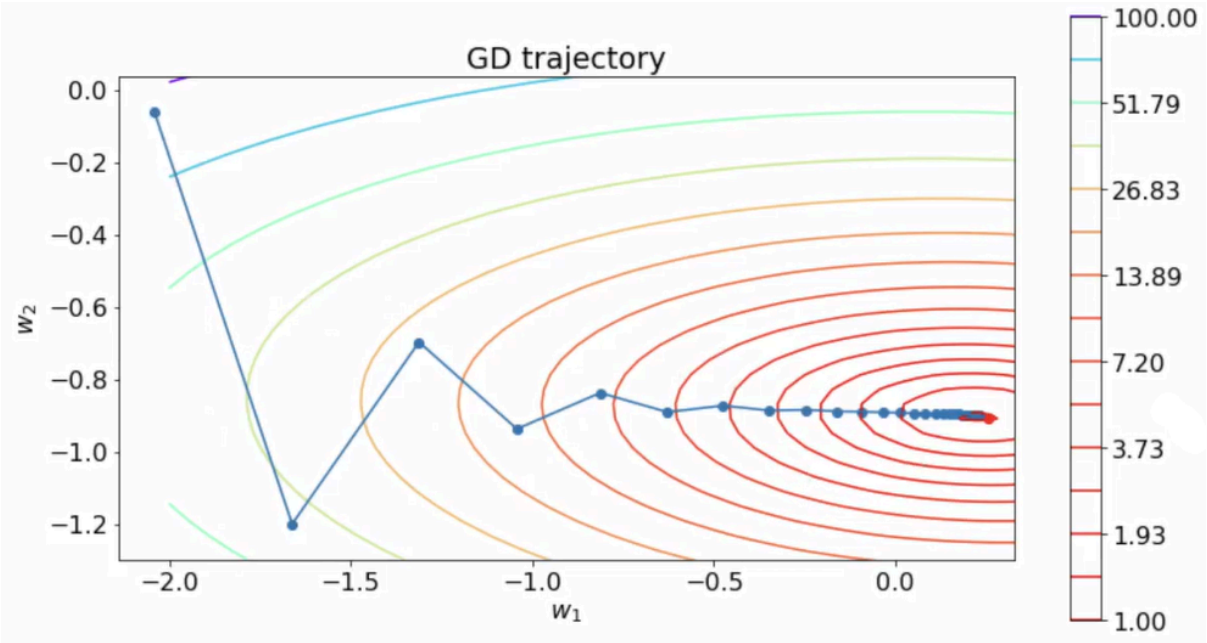
```
np.random.seed(random_seed)
w_0 = np.random.uniform(-2, 2, n_features)-0.5
w = w_0.copy()
w_list = [w.copy()]
step_size = 1e-2

for i in range(num_steps):
      w -= step_size * 2. * np.dot(X.T, (X.dot(w) - Y)) / Y.size # YOUR
      CODE HERE
      w_list.append(w.copy())
w_list = np.array(w_list)
```

Рисуем результат:

```
# compute level set
A, B = np.meshgrid(np.linspace(-2, 2, 100), np.linspace(-2, 2, 100))

levels = np.empty_like(A)
for i in range(A.shape[0]):
      for j in range(A.shape[1]):
            w_tmp = np.array([A[i, j], B[i, j]])
            levels[i, j] = np.mean(np.power(np.dot(X, w_tmp) - Y, 2))

plt.figure(figsize=(13, 9))
plt.title('GD trajectory')
plt.xlabel('$w_1$')
plt.ylabel('$w_2$')
plt.xlim(w_list[:, 0].min() - 0.1, w_list[:, 0].max() + 0.1)
plt.ylim(w_list[:, 1].min() - 0.1, w_list[:, 1].max() + 0.1)
plt.gca().set_aspect('equal')
```

```
# visualize the level set

CS = plt.contour(A, B, levels, levels=np.logspace(0, 2, num=15),
cmap=plt.cm.rainbow_r)

CB = plt.colorbar(CS, shrink=0.8, extend='both')


# visualize trajectory

plt.scatter(w_true[0], w_true[1], c='r')

plt.scatter(w_list[:, 0], w_list[:, 1])

plt.plot(w_list[:, 0}, w_list[:, 1])


plt.show()
```

Получаем функцию потерь:



На рисунке представлены разноцветные линии уровня. У нас эквипотенциальная поверхность, вдоль нее значение функции потерь одинаково.

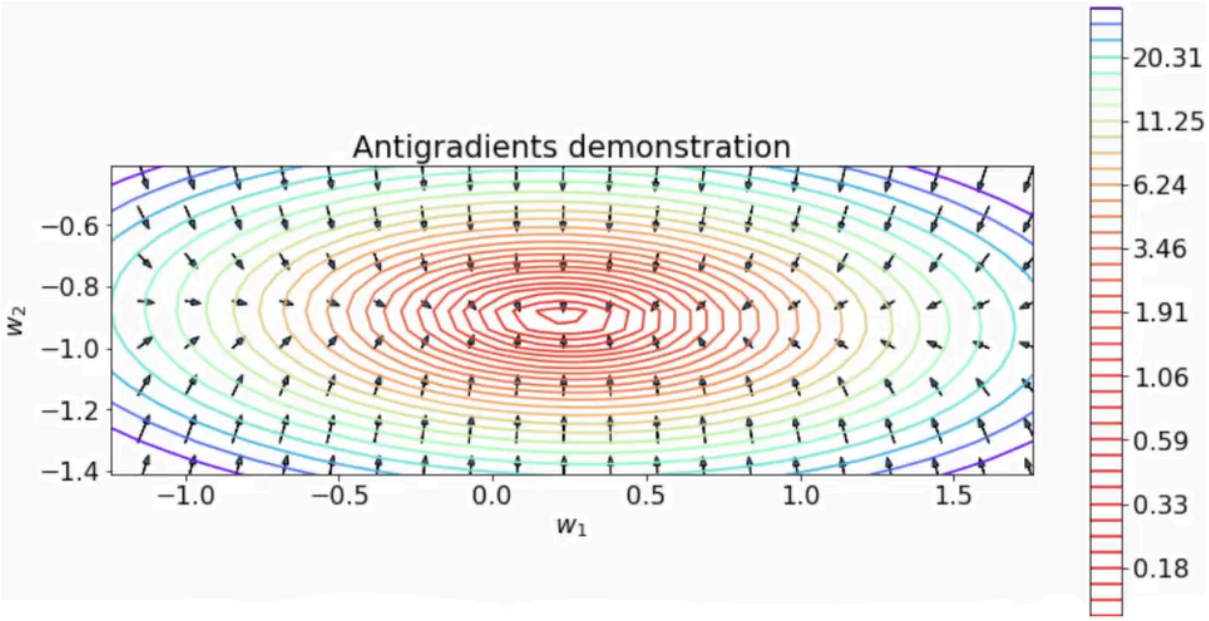
Код отрисовки графика достаточно нетривиальный, поскольку нарисовали все уровни поверхности и в каждой точке градиент. Но для наглядности это полезно.

Градиент всегда ортогонален линиям уровня. Провизуализируем это:

```
# compute level set

A, B = np.meshgrid(np.linspace(-3, 3, 100), np.linspace(-3, 3, 100))

A_mini, B_mini = np.meshgrid(np.linspace(-3, 3, 40), np.linspace(-3, 3,
40))


levels = np.empty_like(A)

for i in range(A.shape[0]):

      for j in range(A.shape[1]):

            w_tmp = np.array([A[i, j], B[i, j]])

            levels[i, j] = np.mean(np.power(np.dot(X, w_tmp) - Y, 2))


# visualize the level set

plt.figure(figsize=(13, 9))

CS = plt.contour(A, B, levels, levels=np.logspace(-1, 1.5, num=40),
cmap=plt.cm.rainbow_r)

CB = plt.colorbar(CS, shrink=0.8, extend='both')


# visualize the gradients

gradients = np.empty_like(A_mini)

for i in range(A_mini.shape[0]):

      for j in range(A_mini.shape[1]):

            w_tmp = np.array([A_mini[i, j], B_mini[i, j]])

            antigrad = -2 * 1e-3 * np.dot(X.T, np.dot(X, w_tmp) - Y) /
            Y.shape[0]

            plt.arrow(A_mini[i, j], B_mini[i, j], antigrad[0],
            antigrad[1], head_width=0.02)


plt.title('Antigradients demonstration')

plt.xlabel(r'$w_1$')

plt.ylabel(r'$w_2$')
```

```
plt.xlim((w_true[0] - 1.5, w_true[0] + 1.5))
plt.ylim((w_true[1] - .5, w_true[1] + .5))
plt.gca().set_aspect('equal')
plt.show()
```

Как выглядит градиент в каждой точке:



У нас антиградиент, поскольку мы минимизируем функцию ошибки. Конечно, он тоже ортогонален линиям уровня, как и градиент, но направлен в другую сторону.

Почему градиент ортогонален линиям уровня? Мы хотим, чтобы как можно скорее изменилась функция ошибки. Если будем «идти» не перпендикулярно поверхности, то у нас будут две составляющие — проекции на плоскость и на перпендикуляр к ней. Шаг вдоль поверхности не приведет ни к каким изменениям, потому что поверхность эквипотенциальная. Значит, остается перпендикуляр, направленный в сторону изменения.

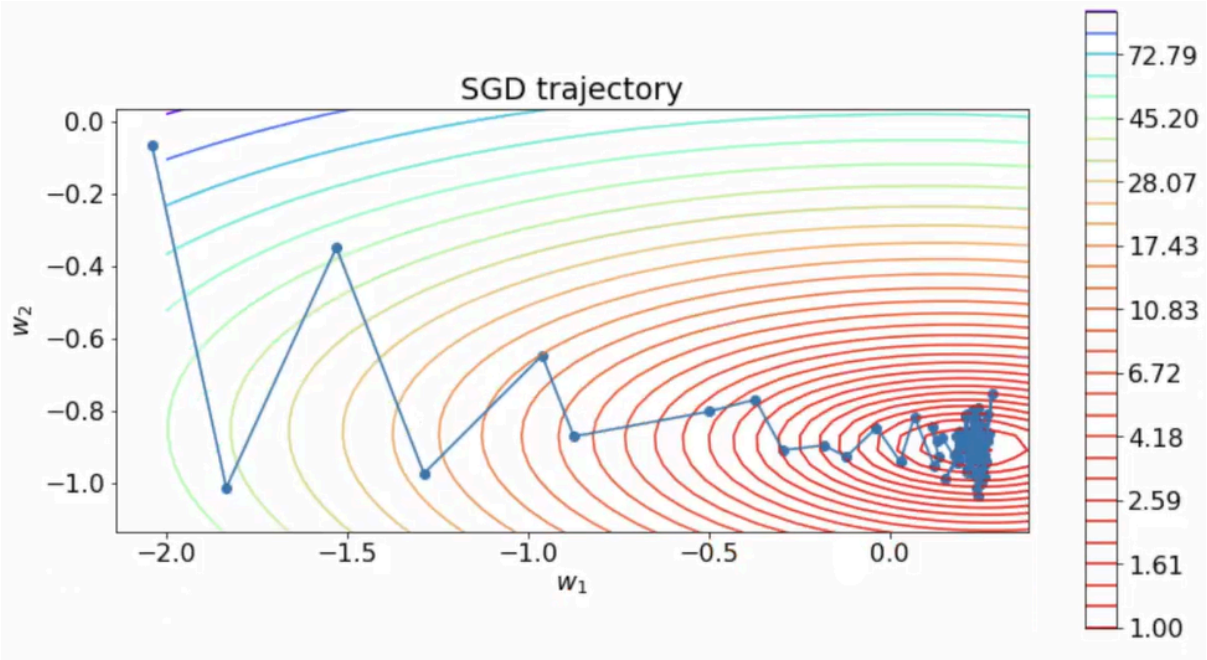
Стохастический градиентный спуск

При градиентном спуске мы в каждой точке считаем градиент. Что делать, если очень много объектов? Тогда можем попробовать работать со **стохастическим градиентным спуском**, и каждый раз выбирать случайное подмножество объектов:

```
np.random.seed(random_seed)

batch_size = 10

w = w_0.copy()

w_history_list = [w.copy()]

lr = 1e-2


for i in range(2*num_steps):

      sample_indices = np.random.randint(0, n_objects, batch_size) # YOUR
      CODE HERE

      X_batch = X[sample_indices, :]

      Y_batch = Y[sample_indices]

      w -= 2 * lr * np.dot(X_batch.T, X_batch @ w - Y_batch) / batch_size
      #YOUR CODE HERE

      w_history_list.append(w.copy())

w_history_list = np.array(w_history_list)
```

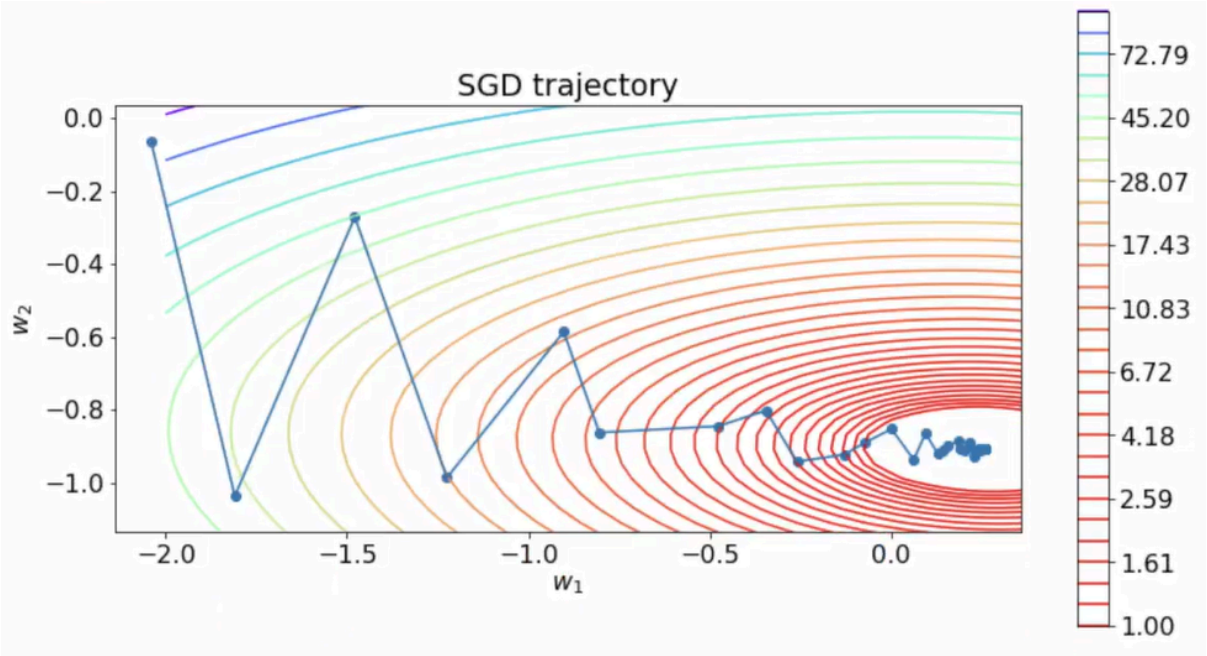
```
# compute level set

A, B = np.meshgrid(np.linspace(-2, 2, 100), np.linspace(-2, 2, 100))


levels = np.empty_like(A)

for i in range(A.shape[0]):

      for j in range(A.shape[1]):

            w_tmp = np.array([A[i, j], B[i, j]])

            levels[i, j] = np.mean(np.power(np.dot(X, w_tmp) - Y, 2))
```

```
plt.figure(figsize(figsize=(13, 9))

plt.title('SGD trajectory')

plt.xlabel(r'$w_1$')

plt.ylabel(r'$w_2$')

plt.xlim((w_history_list[:, 0].min() - 0.1, w_history_list[:, 0].max() +
0.1))

plt.ylim((w_history_list[:, 1].min() - 0.1, w_history_list[:, 1].max() +
0.1))

plt.gca().set_aspect('equal')


# visualize the level set

CS = plt.contour(A, B, levels, levels=np.logspace(0, 25, num=30),
cmap=plt.cm.rainbow_r)

CB = plt.colorbar(CS, shrink=0.8, extend='both')


# visualize trajectory

plt.scatter(w_true[0], w_true[1], c='r')

plt.scatter(w_history_list[:, 0], w_history_list[:, 1])

plt.plot(w_history_list[:, 0}, w_history_list[:, 1])


plt.show()
```



Видим, что график выглядит более грубо, если сравнивать с первым. Около оптимума происходят колебания, градиент «не хотел» сходиться. Дело в том, что размер batch 10, и присутствует шум np.random.normal(0, 1, n_objects).

Если его убрать, то на всех объектах будет точно определен градиент. Шум случайный и центрированный — мы либо влево, либо вправо всегда сдвигаем значение нашей целевой функции.



Чем больше объектов мы учитываем, тем больше в среднем мы избавляемся от шума. Мы просто «угадываем» направление. Если увеличить размер batch, у нас будет

более плавная сходимость, тем более устойчивая оценка градиента. Но при этом менее качественная сходимость.

Условие Робинса-Монро

Мы решали задачу с фиксированным размером шага (learning rate, lr). По идее, чем ближе мы подходим к оптимуму, тем меньше градиент. Мы хотим точно в него сойтись.

Условие Робинса-Монро. Если ряд learning rate расходится (lr всегда положительный), то ряд из его квадратов должен сходиться:

$$\sum_{k=1}^{\infty} \eta_k = \infty, \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty.$$

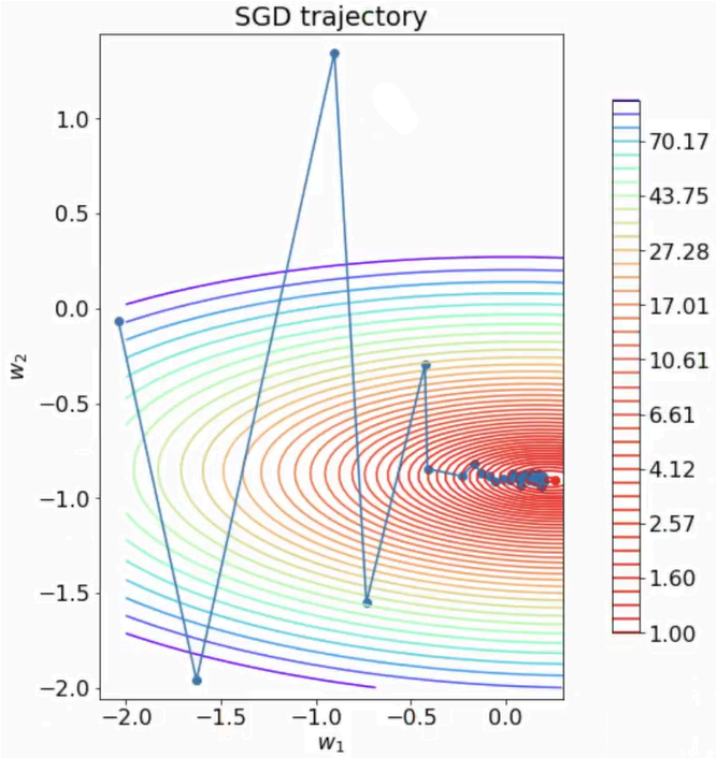
Пример. Ряд $\frac{1}{n^{0.51}}$ расходится, а ряд $\frac{1}{n^{1.02}}$ сходится:

```
np.random.seed(random_seed)

w = w_0.copy()

w_list = [w.copy()]

lr_0 = 0.02


for i in range(num_steps):

      lr = lr_0 / ((i+1) ** 0.51) # What should the power be? )

      sample_indices = np.random-randint(0, n_objects, batch_size) # YOUR
      CODE HERE

      X_batch = X[sample_indices, :]

      Y_batch = Y[sample_indices]

      w -= 2 * lr * np.dot(X_batch.T, X_batch @ w - Y_batch) / batch_size
      # YOUR CODE HERE


      w_list.append(w.copy())

w_list = np.array(w_list)
```

Рисуем график:

```
# compute level set
A, B = np.meshgrid(np.linspace(-2, 2, 100), np.linspace(-2, 2, 100))

levels = np.empty_like(A)
for i in range(A.shape[0]):
      for j in range(A.shape[1]):
            w_tmp = np.array([A[i, j], B[i, j]])
            levels[i, j] = np.mean(np.power(np.dot(X, w_tmp) - Y, 2))

plt.figure(figsize(figsize=(13, 9))
plt.title('SGD trajectory')
plt.xlabel(r'$w_1$')
plt.ylabel(r'$w_2$')
plt.xlim((w_history_list[:, 0].min() - 0.1, w_history_list[:, 0].max() +
0.1))
plt.ylim((w_history_list[:, 1].min() - 0.1, w_history_list[:, 1].max() +
0.1))
plt.gca().set_aspect('equal')

# visualize the level set
CS = plt.contour(A, B, levels, levels=np.logspace(0, 2, num=40),
cmap=plt.cm.rainbow_r)
CB = plt.colorbar(CS, shrink=0.8, extend='both')

# visualize trajectory
plt.scatter(w_true[0], w_true[1], c='r')
plt.scatter(w_history_list[:, 0], w_history_list[:, 1])
plt.plot(w_history_list[:, 0}, w_history_list[:, 1])

plt.show()
```

Результат:



Нам не всегда удается угадать правильный learning rate (lr). Когда мы задаем угасающий lr, мы ограничиваем нашу сходимость. Обычно на практике угасающий lr используют не так явно, обычно его заставляют осциллировать или падать только при определенных условиях.

Наши данные имеют «вытянутый» вид. Посмотрим на нашу матрицу X:

```
X.mean(axis=0)
```

Корень из дисперсии:

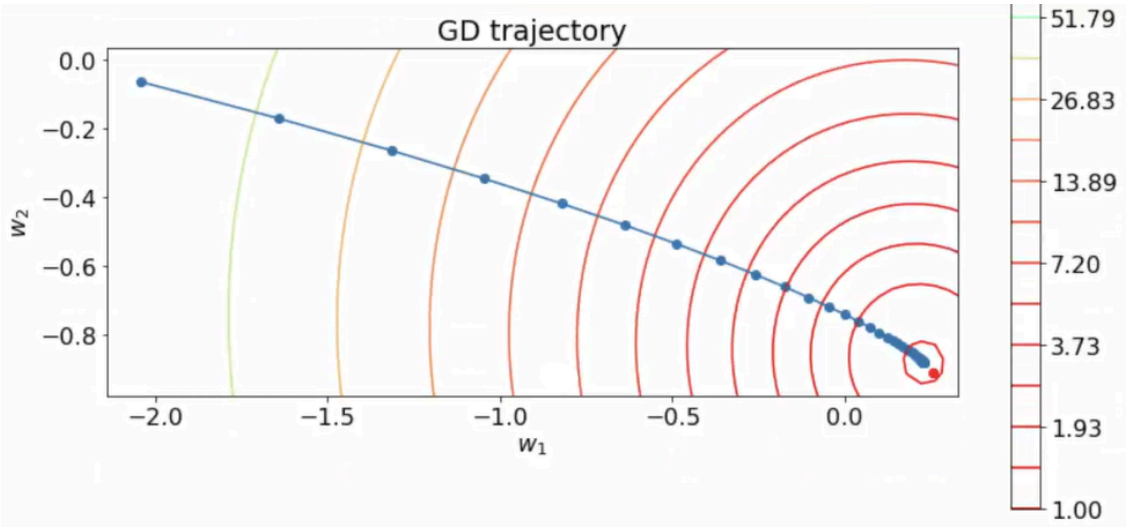
```
X.std(axis=0)
```

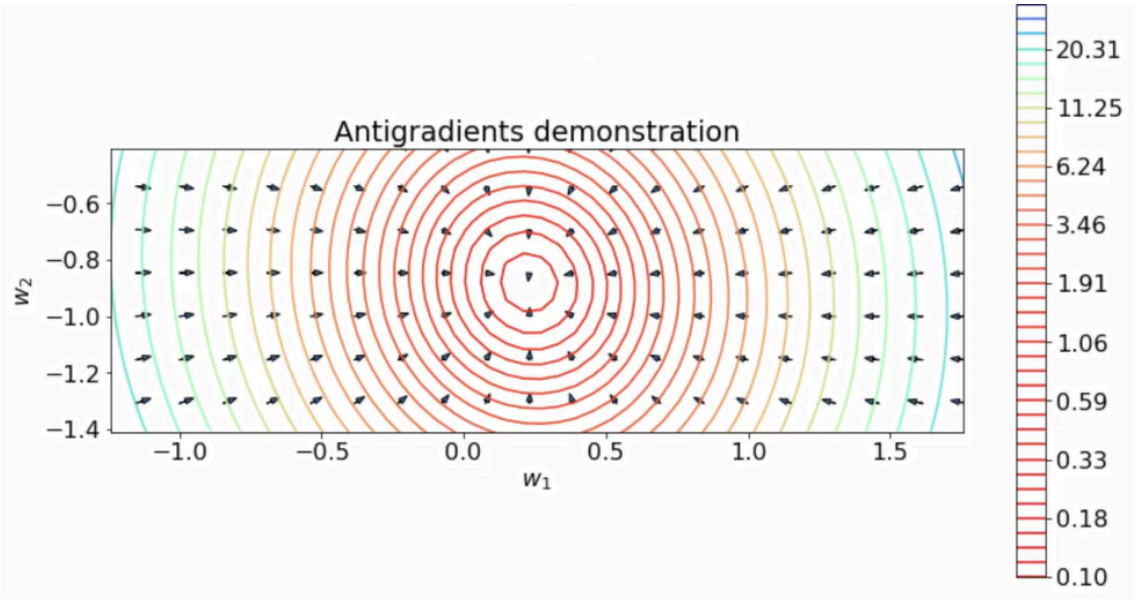
Отнормируем нашу матрицу, то есть проведем стандартизацию:

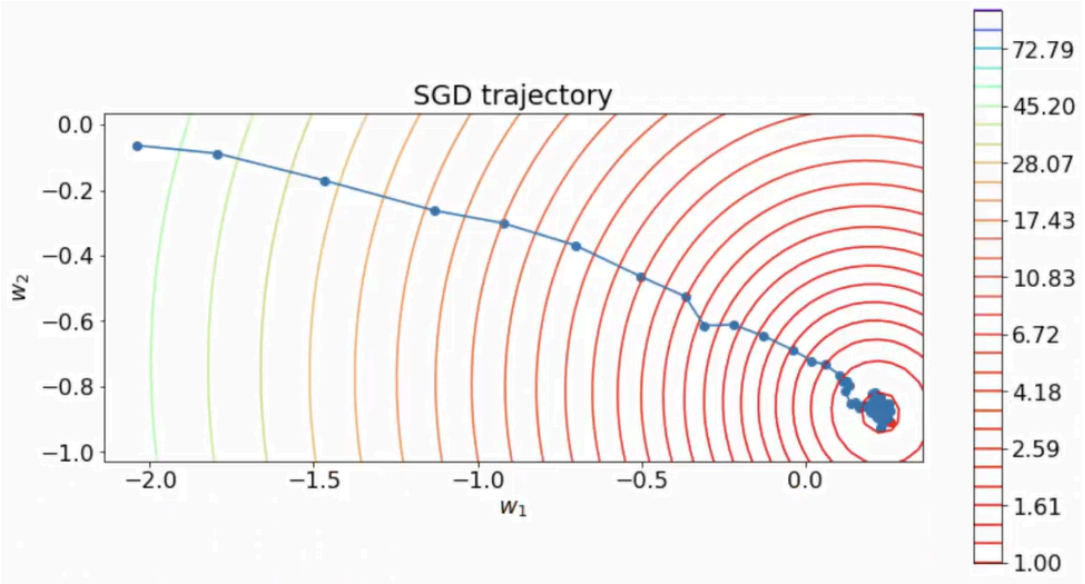
```
X_new =( X - X.mean(axis=0, keepdims=True)) / (X.std(axis=0,
keepdims=True) + 1e-6)
```

```
X = X_new
```

Теперь на картинке у нас все центрально симметрично:







Сравнение стохастического градиентного спуска и обычного

Посмотрим на скорость сходимости при стохастическом градиентном спуске и обычном. Сделаем выборку посложнее:

```
# data generation
n_features = 50
n_objects = 1000
num_steps = 500
batch_size = 10


w_true = np.random.uniform(-2, 2, n_features)


X = np.random.uniform(-10, 10, (n_objects, n_features))
Y = X.dot(w_true) + np.random.normal(0, 5, n_objects)
```

Нарисовать такую выборку мы уже не сможем, поскольку не сможем изобразить 50-мерное пространство:

```
lr_sgd = 1e-3
lr_gd = 1e-3
```

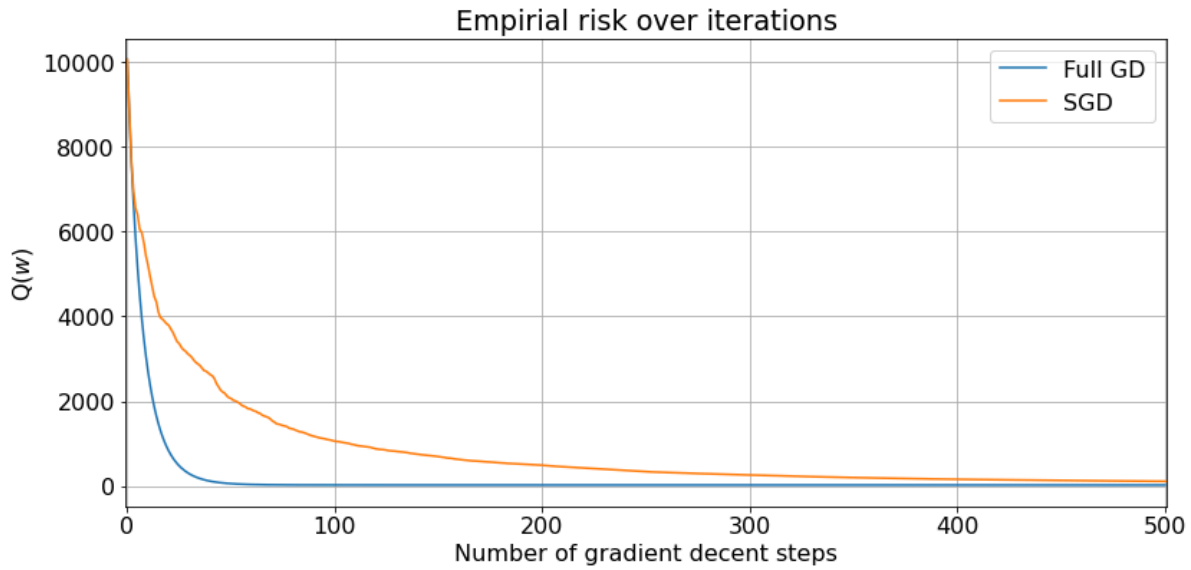
```
w_sgd = np.random.uniform(-4, 4, n_features)

w_gd - w_sgd.copy()

residuals_sgd = [np.mean(np.power(np.dot(X, w_sgd) - Y, 2))]

residuals_gd = [np.mean(np.power(np.dot(X, w_gd) - Y, 2))]


for i in range(num_steps):

      lr = lr_sgd / ((i+1) ** 0.51)

      sample = np.random.randint(n_objects, size=batch_size)

      w_sgd -= 2 * lr * np.dot(X[sample].T, np.dot(X[sample], w_sgd) -
      Y[sampl;e]) / batch_size

      residuals_sgd.append(np.mean(np.power(np.dot(X, w_sgd) - Y, 2)))


      w_gd -= 2 * lr_gd * np.dot(X.T, np.dot(X, w_gd) - Y) / Y.shape[0]

      residuals_gd.append(np.mean(np.power(np.dot(X, w_gd) - Y, 2)))


for i in range(num_steps*4):

      lr = lr_sgd / ((i+1) ** 0.51

      sample = np.random.randint(n_objects, size=batch_size)

      w_sgd -= 2 * lr * np.dot(X[sample].T, np.dot(X[sample], w_sgd) -
      Y[sample]) / batch_size

      residuals_sgd.append(np.mean(np.power(np.dot(X, w_sgd) - Y, 2)))
```

Мы используем усредненный градиент.

Посмотрим на результаты:

```
plt.figure(figsize=(13, 6))

plt.plot(np.arange(num_steps+1), residuals_gd, label='Full GD')

plt.plot(np.arange(num_steps*5+1), residuals_sgd, label='SGD')

plt.xlim((-1, (num_steps+1)))

plt.legend()

plt.xlabel('Number of gradient decent steps')
```

```
plt.ylabel(r'Q($w$)')
plt.grid()
plt.show
```



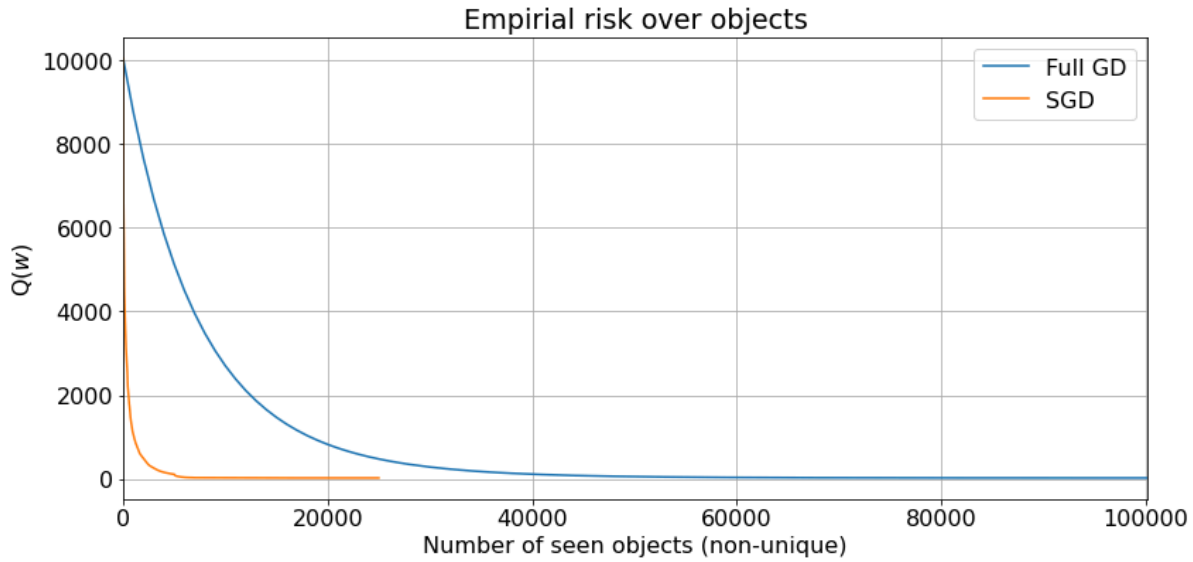
Видим, что градиентный спуск сходится меньше чем за 100 итераций. Стохастический градиентный спуск даже за 500 итераций не дошел до значения градиентного спуска.

Кажется, что градиентный спуск лучше стохастического.

Теперь построим график сходимости градиентного спуска и стохастического градиентного спуска от количества объектов, на которых считалась ошибка. Градиентный спуск на каждом шаге обрабатывает все объекты, а стохастический — только 10 объектов:

```
plt.figure(figsize=(13, 6))
plt.plot(np.arange(num_steps+1)*n_objects, residuals_gd, label='Full GD')
plt.plot(np.arange(num_steps*5+1)*batch_size, residuals_sgd, label='SGD')
plt.title('Empirial risk over objects')
plt.xlim((-1, (num_steps+1)*n_objects/5))
plt.legend()
plt.xlabel('Number of seen objects (non-unique)')
plt.ylabel(r'Q($w$)')
plt.grid()
```

```
plt.show()
```



Стохастический спуск сходится примерно на 1000 итераций. Обычному градиентному спуску нужно в 10 раз больше, чтобы сойтись. В стохастическом градиентном спуске шагов больше, но меньше вычислений.

Дополнительные материалы для самостоятельного изучения

1. https://www.expii.com/t/precalculus-concepts-continuous-vs-discontinuous-4767
2. https://math24.net/increasing-decreasing-functions.html
3. https://en.wikipedia.org/wiki/Monotonic_function
4. https://www.researchgate.net/figure/Judgment-of-concavity-and-convexity-a-Convex-curve-b-Concave-curve_fig3_339939083
5. http://www.conejousd.org/Portals/49/Departments/Math/Sansing/Math%20AnalyCP/Notes%203-6%20Critical%20Points%20and%20Extrema.pdf
6. https://www.sydney.edu.au/content/dam/students/documents/mathematics-learning-centre/introduction-to-differential-calculus.pdf
7. https://raw.githubusercontent.com/girafe-ai/ml-course/22f_basic/week0_02_linear_reg/utils_02.py