Методы оптимизации, лекции

Глава 1

О предмете

Слова оптимизация просисходит от слова *optimus* — поиск наилучшего решения.

Чем эта дисциплина занимается?

Поиск минимума или максимума какой-либо функции. $f(x) \to \min(\max)$.

Например, $f(x)$ – стоимость, которую мы хотим минимизировать.

Обычно мы бдуем рассматривать функции, действующие из множества в числа, $f : A \to \mathbb{R}$.

Поиск решения задачи, где у нас оптимизируется несколько параметро – многокритериальная оптимизация.

Во многокритериальную оптимизацию сильно углубляться мы не будем.

Методы математической оптимизации также называют математическим программированием (программмирование $\equiv$ поиск оптимального плана).

1.1 Какиого вида может быть функция f?

1.1.1 Линейная

Пусть $f(x) = \varphi \cdot x$.

Как правило, есть дополлнительные ограничения $A \cdot x = b$ (A – матрица, b – вектор). $x_i \geqslant 0$.

Пример 1 (Транспортная задача)**.** Есть n складов, m магазинов как эффективнее огранизовать логистику?

Вообще, бывают задачи, где ограничения есть и где их нет.

1.1.2 Квадратичная функция

Пусть $f(x) = \varphi \cdot x + x^T \cdot \theta \cdot x$.

Простоейшая задача – линейная регрессия. При помощи линейной функции покрыть множество точек, так чтобы сумма квадратов отклонений была минимальным.

Так как квадратичная функция выпукла, то ответ, обычно, бывает один.

В более сложных случаях глобавльных минимумов может не быть и придется искать локальные минимумы.

1.1.3 Нельнейная функция

$f(X) \to \min, f(X) \to \mathbb{R}$. Иногда, функция может быть дискретной. Например, в задачах динамического программирования.

Пример 2. Пусть есть окружность с радиусом R, нужно вписать в него прямоугольник со сторонами a и b.

$f = a \cdot b$. Ограничение $\sqrt{a^2 + b^2} = 2R$.

Пример 3. Обчуение различных моделей машинного обучения.

1.2 Методы решения

Иногда можно решить аналитически или в явном виде.

Но, это получается отнюдь не всегда.

Определение 1 (Итеррационные методы решения)**.** Позволяют на каждом своем шаге как-то уточнять результаты решения. И таким образом можно получить, возможно, не точное решение, но достаточно близкое к нему.

Пусть есть некоторая фунция $f : \mathbb{R}^n \to \mathbb{R}$. Методы решения бывают детерминимрованными и стахастическими.

Один из методов решения – **метод Монте-Карло** (генерируем случайные решения, проверяем, что они подходят, выбираем среди них оптимальный). В этом методе надо вычислять только значние функции.

Еще один метод – **метод иммитации отжига**. Мы находимся в точке, выбираем еще одну точку. Сморим, там значение меньше или больше в зависимости от этого получаем вроятность, что мы туда перейем.

Классификация методов по порядкам:

- Нулевого порядка – считаем только значние функции;
- Первого парядка – используем градиенты изменения функции;
- Второго порядка – используем вторые производные.

Еще один метод нулевого порядка – **метод Нелдера – Мида** (похоже на симпликс-метод, но для большего множества задач).

И еще один метод нулевого порядка – **эволюционный метод или генетические алгоритмы**. Использование подходов монте-карло с методами эволюции – скрещеванием, матацией и отбором.

Методы **первого порядка** используют производные или градиенты. Из математичсекого анализа известно, что градиент показывает направление наибольшего роста фнукции и, следуя туда, можно найти какой-то экстремум.

Самый простейший метод – градиентный спуск. Берем точку начального приближения, считаем в ней градиент и шагаем по направлению градиента. Такой метод сходися. На некоторых классах функции дает достаточно неплохой отевет.

Метод наискорейшего спуска. Считаем градиент в одной точке и там, где в том направлении, где градиент минимальный. Ищем в том направлении минимум и переходим туда.

В методах **второго порядка** используются матрица вторых производных – гессиан. Ингда читать вторые производные достаточно накладно. Поэтому, такие методы применяются достаточнно редко и для специфических задач с небольшим количством параметров.

Один из таких методов – метод ньютона.

Есть **квази–ньютоновские методы**. Они приближают матрицу вторых производных и это позволяет не считать матрицу явно. BFGS.

1.3 Почему Python?

Python на данный момент является стандартом как для научного программирования, так и для, в частности, машинного обучения.

С одной сторны, питон очень простой. С другой стороны, на питоне есть большое количество библиотек.

Да, Python очень медленный. Однако есть библиотеки, например numpy, которые используют оптимальные оптимизации и засчет этого будет большой выигрыш.

1.3.1 Библиотеки для python, без которых жить будет сложно

NumPy

– самая важная библиотека. Позволяет работать с многомерными числовыми массивами и выполнять операции над ними.

Зачастую нужно будет сформулировать свой алгоритм так, чтобы он выражался в виде операций над массивами.

Оффициальный сайт numpy.org/doc/stable.

Broadcasting. В nympy особенное предобразование массивов.

Если умножить число на массив, число неявно превратится в массив и после этого произойдет поэлементное умножение. То же самое есть с массивами разных размерностей.

SciPy

Библиотека для научных вычислений на Python. В ней есть много уже готовых алгоритмов, в том числе методов оптимизации.

Сайт scipy.github.io.

MatPlotLib

Библиотека для построения различных визуализаций, графиков и т.д.

Сайт matplotlib.org

Pandas

Библиотека для работы с таблицами и табличными данными.

Сайт pandas.pydata.org

1.3.2 Как работать с python

Есть такое понятие, на **jupiter notebook**. Это интерактивные блокноты в которых можно писать на python или других языках, писать текст с математическими формулами и делать раличные визулизации.

Редакторы можно использовать различные. Можно писать в VS Code или PyCharm, еще есть Google Cloud.

Массивы в python

```
    a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    a[-1] # 9
    a[2:2] # [3, 4]
    a[::-1] # [9, 8, 7, 6, 5, 4, 3, 2, 2, 1]
```

1.3.3 Испольщование numpy

```
    import numpy as np
    a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
    a * 2 # [2, 4, 6, 8, 10, 12, 14, 16, 18]
    [1, 2, 3] + [4, 5, 6] # [1, 2, 3, 4, 5, 6]
```

```
    a = np.arrange(0, 10)
    a # array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    a + 10 # array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    a ** 2 # array([0, 1, 4, 9, 16, 25, 36, 49, 64, 81])
    np.sin(a) # sin foreach elements
    np.sum(a) # sum of elements
    a[a > 3] # [4, 5, 6, 7, 8, 9]
    a.shape() # (10,)
    a.reshape(2, 5) # [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]
    a.reshape(2, 5).T # [[0, 5], [1, 6], [2, 7], [3, 8], [4, 9]]
```

```
    a.reshape(2, 5) * [-1, 2] # [[0, 2], [2, 6], [4, 10], [6, 14], [8, 18]]

    b = np.array([[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]])
    b[:, 0] # [0, 2, 4, 6, 8] -- first column
    b[0, :] # [0, 1] -- first line
    b[:2, :2] # [[0, 1, [2, 3]]

    np.array([1, 2, 3, 4, 5])[:, np.newaxis]
```

Графики

```
    x = n.linspace(0, 10, 100)
    plt.plot(x, x) # graphic of f: [0, 10] to [0, 10], f(x) = x
    plt.plot(x, np.sin(x)) # graphic of f(x) = sin(x)

    x = n.linspace(0, 10, 1000)
    plt.plot(x, np.sin(x ** 2))
    plt.grid()
    plt.xlabel("axis x")
    plt.title("$\sin x\^2$")
```

Глава 2

Градиентный спуск

Пусть нам дана функция f. Найдем минимум функции.

Величина **learning rate** – скорость обучения. Сами итеррации можно назвать эпохами.

Градиентный спуск может быть стахастическим. В этом случае градиент можно считать не целиком, выбирая только часть каких-то параметров.

Иногда можно иметь шаг не константный, а подбирать его каждый раз при помощи какого-то метода.

Градиент можно вычислять при помощи численных методов. Например, через центральную разность

$h = \varepsilon$, $f'_k(X) = \dfrac{f(X + h \cdot (0, \ldots, 1_k, \ldots, 0)) - f(X - h \cdot (0, \ldots, 1_k, \ldots, 0))}{2h}$.

Еще один важный вопрос – масштабирование. Удобно искать решения, если расстояния при приближении по ращличным осям примерно равны. Хуже, когда есть вытянутые овраги.

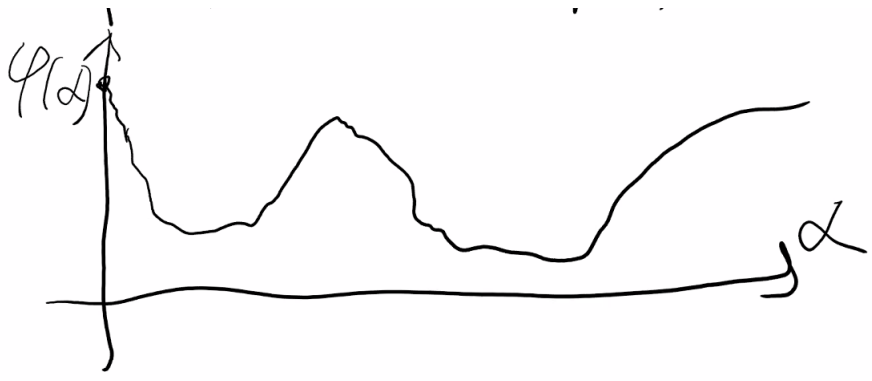
Число обусловенности (condition number). Абсолютное число обучловенности:

$$\lim_{\varepsilon \to 0} \sup_{\|\delta x\| \leqslant \varepsilon} \frac{\|\delta f(x)\|}{\|\delta x\|}.$$

2.1 Одномерные методы оптимизации

Пусть у нас есть функция $f(x)$.

Пусть мы выбрали некоторое напроавление p. Тогда для оптимизации мы можем рассматривать функцию $\varphi(\alpha) = f(x_k + \alpha p_k), \alpha > 0$.



Наша задача — найти некоторый локальный минимум этой функции.

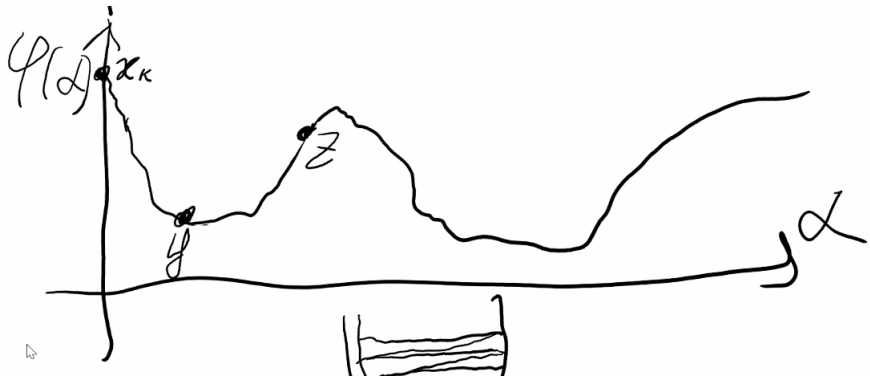
Встает два вопроса:

- Как найти этот минимум?
- Как точно искать этот минимум?

Если вычислять очень точно, мы потратим очень много на это ресурсов. Если искать не точно, есть риск потерять сходимость функции.

Какие бывают методы поиска локального минимума в этой задаче?

Пусть нам известна точка x_k, мы хотим найти еще две точки x, y, в одной из которых (y) функция будт меньше, а в другой (z), больше чем в той (y).



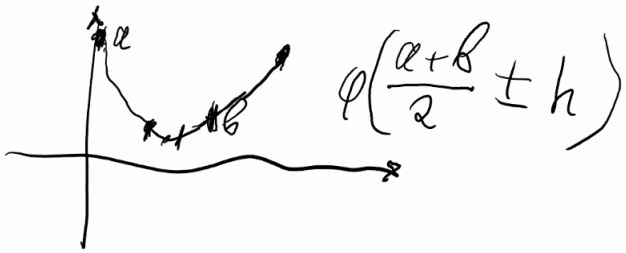
Как правило здесь используют довольно примитивные подходы.

У нас убывает функция, давайте возьмем некоторый шаг α (обчно < 1) и посчитаем $\varphi(\alpha)$.

Если там фунция меньше, чем $\varphi(0)$, то мы нашли y. Как найти z. Давайте воскольконибудь увеличим шаг α, например, в два раза и посчитаем там $\varphi(2\alpha)$. Там может быть $\varphi(2\alpha) > \varphi(\alpha)$, то $z = 2\alpha$. Иначе, путь $y = 2\alpha$, а z стоит попробовать найти еще раз.

Что делать, если $\varphi(\alpha) > \varphi(0)$. Тогда возьмем $\alpha/2$ и попробуем сделать то же самое.

Допустим, мы нашли некоторый интервал, на котором есть минимум. То есть есть три точки, в средней из которых функция меньше. Один из способов дальшнейшего исслежования функции — **метод дихатомии**. Возьмем центр этого интервала и изучать поведение функции справа и слева от этой точки: $\varphi\left(\frac{a+b}{2} \pm h\right)$. h — выбирается $\pm$ имперически.



Дальше мы справниваем знаение функции слева и справа ($\pm h$) и переносим ближайшую крайнюю точку в точку с большим значением. На рисунке выше мы перенесли b в точку $\frac{a+b}{2} + h$.

Плюсы такого метода: простой метод (по модулю предположения, что функция унимодальна), считать просто.

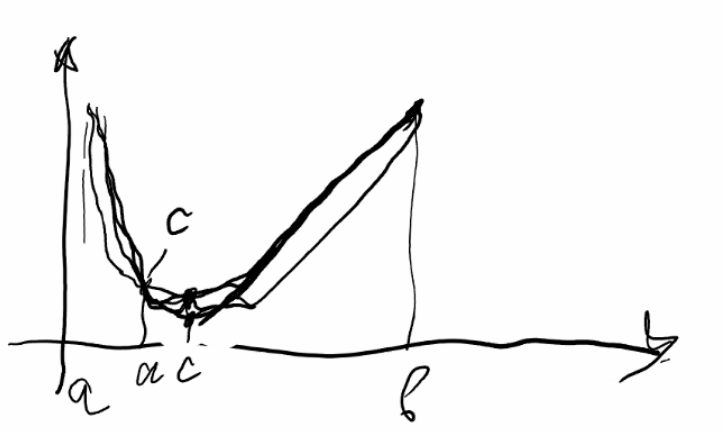
Минусы: не очень сильно эффективен (на каждый шаг два вычисления функции).

Более эффективный метод: метод золотого сечения (метод Фибоначчи).

Еще один из простых, но при некоторых предположениях достаочно эффективный — **метод полиномиальной Аппроксимации**. Предположим, что функция очень близка к какому-то полиному. Самое просто — квадратичному полиному (параболе).

Пусть у нас есть три точки, проведем через них параболу. Теперь аналичтически найдем точку минимума этой параболы и в этой точке посмотрим явное повдение нашей функции.

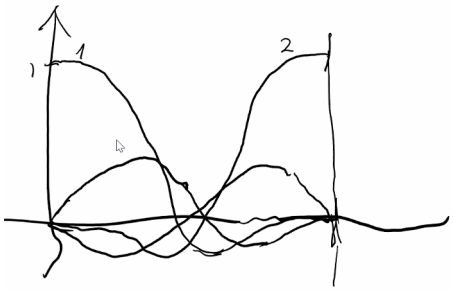
Дальше, так же, как в Дихатомии, сдвиним рассматриваемый отрезок фунции и повторим наши действия.



Плюсы: если функция хорошая (достаточно гладкая, вблизи минимума хорошо аппроксимируется вблизи минимума).
Минусы: если функция плохая, будет работать очень не точно.

Еще можно аппроксимировать с использованием **производных**. Пусть в точках a и b мы знаем не только значение φ, но и значение производной φ'.

Для аппроксимации есть эрмитовы полиномы, их 4:



Возьмем: $\varphi(a)E_1 + \varphi(b)E_2 + \varphi'(a)E_3 + \varphi'(b)E_4$.

Плюсы и минусы те же. Кроме того, функция φ должна быть достаточно гладкой, что бы мы могли вычисялять ее производные.

Метод Ньютона (метод касательных). Строим касательные (производные), ищем их пересеченеи с нулем ($\varphi' = 0$). Найденные точки (нули) и есть ответ.

Плюсы: если функция квадратичная (или другая хорошая) будет очень быстро сходиться.
Минусы: нужны вторые производные.

Метод секущих (хорд). Так же, как в методе Ньютона, ищем нули.

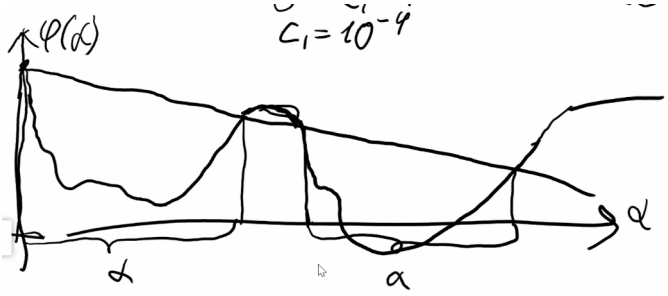
2.1.1 Условия Вольфе

Условия выбора следующей точки для исследования ее на минимум.

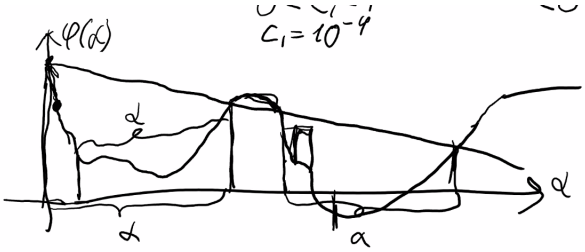
1. $f(x_k + \alpha p_k) \leqslant f(x_k) + c_1 \alpha \nabla f(x_k) p_k$,
   где $0 < c_1 < 1$, $c_1 = 10^{-4}$.

2. $\nabla f(x_k + \alpha_k p_k) \cdot p_k \geqslant c_2 \nabla f(x_k) \cdot p_k$.

Первое условие накладывает вограничения на выбираемое α.



Первого условия не достаточно, так как можно взять $\alpha = \varepsilon$ и это условие там будет выполняться.
Второе условие запершает делать делать слишком маленькие шаги и шагать туда, где градиент продолжает достаточно быстро убывать



$$\nabla f(x_k + \alpha_k p_k) \cdot p_k \geqslant c_2 \nabla f(x_k) \cdot p_k \quad (2)$$
$$c_1 < c_2 < 1 \qquad \nabla f(x_k) \cdot p_k < 0$$
$$c_2 = 0.9$$

Одномерная оптимизация — некоторый итерративный процесс, в котором мы ищем некоторый локальный минимум и потом пытаемся его улучшить. Возникает вопрос, а в какой момент нужно остановиться? Мы можем добиваться точности один знак после запятой или десять знаков после запятой. И так и так, мы можем в итоге прийти к минимуму многомерной функции.

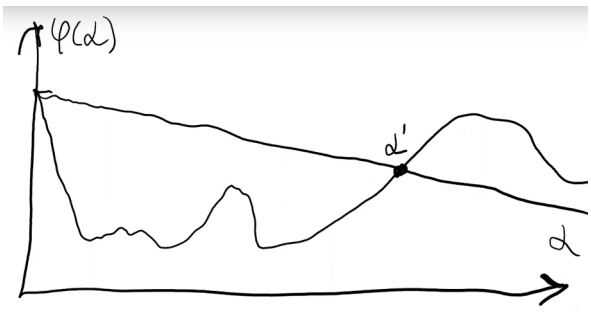
Ответ на возникший вопрос — условия Вольфе. Как только они стали выполняться, мы можем прикрать одномерную опитимизацию.

Вспомним условия Вольфе:

1. $f(x_k + \alpha p_k) \leqslant f(x_k) + c_1 \alpha \nabla f(x_k) p_k$,
2. $\nabla f(c_k + \alpha_k p_k) \cdot p_k \geqslant c_2 \nabla f(x_k) \cdot p_k$.

Где, c_1, c_2 — некоторые параметры, удовлетворяющие условию $0 < c_1 < c_2 < 1$.

Доказательство корректности условий Вольфе. Запишем правую часть первого условия в виде функции и получим прямую, которая идет из начальной точки x_k и убвает. В некоторой точке α' она пересекается с функцией. Для a' верно $f(x_k + \alpha' p_k) = f(x_k) + \alpha' c_1 \nabla f(x_k) \cdot p_k)$.



Заметим, что первое условие выполнятется всюду, где $\alpha < \alpha'$. Почему где-то на этом интервале выполнится второе условие?

$$f(x_k + \alpha' p_k) - f(x_k) = \alpha' \nabla f(p_k + \alpha'' p_k) p_k, \quad \text{где } \alpha'' \text{ — некоторая неизвестная нам точка.}$$

Тогда,

$$\nabla f(x_k + \alpha'' p_k) \cdot p_k = c_1 \nabla f(x_k) \cdot p_k > c_2 \nabla f(x_k) p_k.$$

Таким образом, α'' удовлетворяет второму условию Вольфе. □

Введем обозначение $\cos\Theta = \dfrac{-\nabla f(c_k) \cdot p_k}{\|\nabla f(x_k)\| \|p_k\}}$.

Теорема 1. Если p_k — направление поиска, α_k удовлетворяет условиям Вольфе, f ограничена и непрерывно дифференцируема и градиент функции f является Липшица–непрерыным ($\|\nabla f - \nabla f(\overline{x})\| \leqslant L\|x - \overline{x}\|, \;\; L > 0$). Тогда будет выполняться следующее соотношение

$$\sum_{k \geqslant 0} \cos^2 \Theta_k \|\nabla f(x_k)\|^2 < +\infty.$$

Замечание 1. Если $\cos^2 > 0$, то последовательность градиентов должна стремится к 0.

Доказательство.

$$\underbrace{(\nabla f(x_{k+1}) - \nabla f(x_k)) \cdot p_k}_{\leqslant \alpha_k L \|p_k\|^2} \geqslant (c_2 - 1) \cdot \nabla f(x_k) \cdot p_k.$$

Выразим α_k:

$$\alpha_k \geqslant \frac{c_2 - 1}{L} \cdot \frac{\nabla f(x_k) \cdot p_k}{\|p_k\|^2}.$$

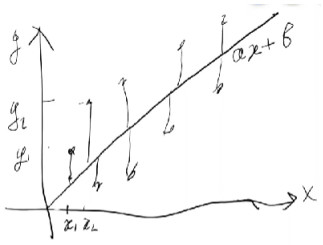
Скомбинируем условие на α с первым условием:

$$f(x_k) \leqslant f(x_k) - \underbrace{c_1 \frac{1 - c_2}{L} \cdot \frac{(\nabla f(x_k) \cdot p_k)^2}{\|p_k\|^2}}_{-c \cos^2 \Theta \cdot \|\nabla f(x_k)\|^2, \;\; c = \frac{c_1(1-c_2))}{L}} \implies f(x_{k+1}) \leqslant f(x_0) - c \sum_{j=0}^{k} \cos^2 \Theta \|\nabla f(x_j)\|^2.$$

Так как фукция была огранчиена, то ряд ограничен и сходится. □

2.2 Стахастический градентный спуск

Задача 1 (Линейная регрессия)**.** Пусть у нас есть некоторые точки на плоскости. Мы хотим провести некоторую прямую, чтобы минимизировать сумму квадратов отклонений.



Пусть есть функция $f(a, b) = \sum_{i=1}^{n} (ax_i + b - y_i)^2$. Хотим найти такие a, b, что f принемает минимальное значние.

Вообще, задачу можно решить аналитически.

Но если обощить ее на пространство более высокой размерности, аналитиечски решать будет сложно, численные методы лучше.

Можно переформулировать задачу в матричном виде:

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} x_{1,1} & \dots & x_{1,n} \\ \dots & \dots & \dots \\ x_{n,1} & \dots & x_{n,n}. \end{pmatrix}, \quad f = \|X\beta - Y\|^2 \implies \beta = \left(X^T X\right)^{-1} X^T Y.$$

Если размерность большая, это не сработает.

Определение 2. Пусть $f(x) = \sum\limits_j Q_j(x)$.

Стахастический градиентный спуск заключается в выборе одного случайного слогаемого в сумме, вычислении градиента этого слогаемого и спуск по этому градиенту.

+ Один шаг очень дешевый.
- Результат от одного шага достаточно сомнительный, поэтому говорить о сходимости данного метода не приходится.
- Найти точный минимум очнеь сложно.

У данного метода есть достаточно много модификаций.
Одна из наиболее популярных модификаций — **Mini Batch Gradient Descent**.

2.2.1 Mini Batch Gradient Descent

Мы можем проводить полный градиентный спуск, то есть считать граент по всем n слогаемым (тяжело вычислять). Можем проводить стахастичесий градиентный спуск и считать градиен по одному слогаемому (получается не точно). А можем выбрать, например 10 случайных слогаемых и посчитать по ним градиент.

В machine learning обыно выбирают не случайные слогаемые, а спрева перемешивают, а затем последовательно рассматривают первые k, вторые k и т.д. слогаемых.