

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. $f(x) \rightarrow \min(\max)$.

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

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

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 (программирование \equiv поиск оптимального плана).

1.1 Какого вида может быть функция f ?

1.1.1 Линейная

Пусть $f(x) = \varphi \cdot x$.

Как правило, есть дополнительные ограничения $A \cdot x = b$ (A — матрица, b — вектор). $x_i \geq 0$.

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

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

1.1.2 Квадратичная функция

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

Простейшая 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 глобальных минимумов может не быть и придется искать локальные минимумы.

1.1.3 Нельнейная функция

$f(X) \rightarrow \min, f(X) \rightarrow \mathbb{R}$. 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. Обучение различных моделей машинного обучения.

1.2 Методы решения

Иногда можно решить аналитически или в явном виде.

Но, это получается отнюдь не всегда.

Определение 1 (Итерационные 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 функция $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Методы решения бывают детерминированными и стохастическими.

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

Еще один метод – **метод имитации отжига**. Мы находимся в точке, выбираем еще одну точку. Смотрим, там значение меньше или больше в зависимости от этого получаем вероятность, что мы туда перейдем.

Классификация методов по порядкам:

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

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

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

Методы **первого порядка** используют производные или градиенты. Из математического анализа известно, что градиент показывает направление наибольшего роста функции 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 сходится. На некоторых классах функции дает достаточно неплохой ответ.

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. Иногда считать вторые производные достаточно накладно. Поэтому, такие методы применяются достаточно редко и для специфических задач с небольшим количеством 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 стороны, питон очень простой. С другой стороны, на питоне есть большое количество библиотек.

Да, Python очень медленный. Однако есть библиотеки, например numru, которые используют оптимальные оптимизации и за счет 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.

Официальный сайт numpy.org/doc/stable.

Broadcasting. В numpy особенное преобразование 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 **jupyter notebook**. Это интерактивные блокноты в которых можно писать на python или других языках, писать текст с математическими формулами и делать различные визуализации.

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

Массивы в python

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

1.3.3 Использование numpy

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

1 a = np.arange(0, 10)
2 a # array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
3 a + 10 # array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
4 a ** 2 # array([0, 1, 4, 9, 16, 25, 36, 49, 64, 81])
5 np.sin(a) # sin foreach elements
6 np.sum(a) # sum of elements
7 a[a > 3] # [4, 5, 6, 7, 8, 9]
8 a.shape() # (10,)
9 a.reshape(2, 5) # [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]
10 a.reshape(2, 5).T # [[0, 5], [1, 6], [2, 7], [3, 8], [4, 9]]
```

```

11 a.reshape(2, 5) * [-1, 2] # [[0, 2], [2, 6], [4, 10], [6, 14], [8, 18]]
12
13 b = np.array([[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]])
14 b[:, 0] # [0, 2, 4, 6, 8] -- first column
15 b[0, :] # [0, 1] -- first line
16 b[:2, :2] # [[0, 1], [2, 3]]
17
18 np.array([1, 2, 3, 4, 5])[:, np.newaxis]

```

Графики

```

1 x = np.linspace(0, 10, 100)
2 plt.plot(x, x) # graphic of f: [0, 10] to [0, 10], f(x) = x
3 plt.plot(x, np.sin(x)) # graphic of f(x) = sin(x)
4
5 x = np.linspace(0, 10, 1000)
6 plt.plot(x, np.sin(x ** 2))
7 plt.grid()
8 plt.xlabel("axis x")
9 plt.title("$\sin x^2$")

```

Глава 2

Градиентный спуск

Пусть нам дана функция f . Найдем минимум функции.

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

Градиентный спуск может быть стохастическим. 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

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

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

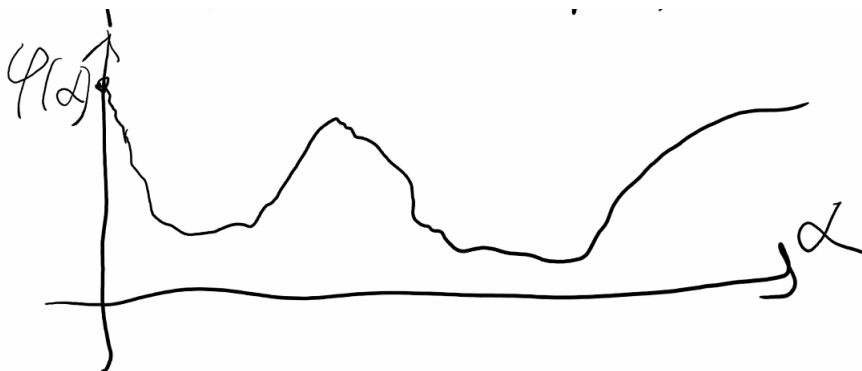
Число обусловленности (condition number). Абсолютное число обусловленности:

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

2.0.1 Одномерные методы оптимизации

Пусть у нас есть функция $f(x)$.

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



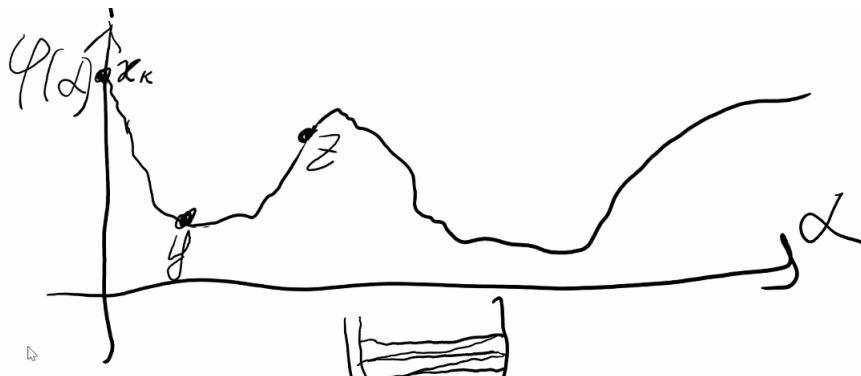
Наша задача — найти некоторый локальный минимум этой функции. Встает два вопроса:

- 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 будет 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 (обычно < 1) и посчитаем $\varphi(\alpha)$.

Если там функция меньше, чем $\varphi(0)$, то мы нашли y . Как найти z . Давайте восколько-нибудь 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, пусть $y = 2\alpha$, 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 дальнейшего исследования функции — **метод дихотомии**. 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 сравниваем значение 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.

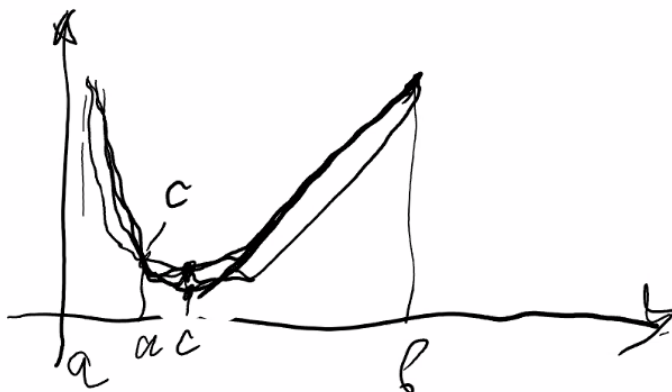
Минусы: не очень сильно эффективен (на каждый шаг два вычисления функции).

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

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

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

Дальше, так же, как в Дихотомии, сдвинем рассматриваемый отрезок функции и повторим наши действия.

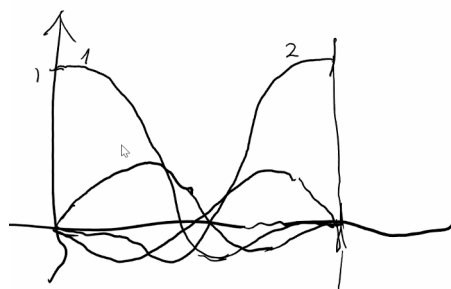


Плюсы: если функция хорошая (достаточно гладкая, вблизи минимума хорошо аппроксимируется вблизи минимума).

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 вычислять ее производные.

Метод Ньютона (метод касательных). Строим касательные (производные), ищем их пересечения 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.

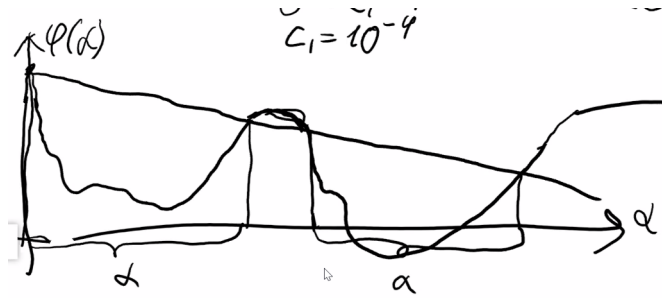
Метод секущих (хорд). Так же, как в методе Ньютона, ищем нули.

2.0.2 Условия Вольфе

Условия выбора следующей точки для исследования ее на минимум.

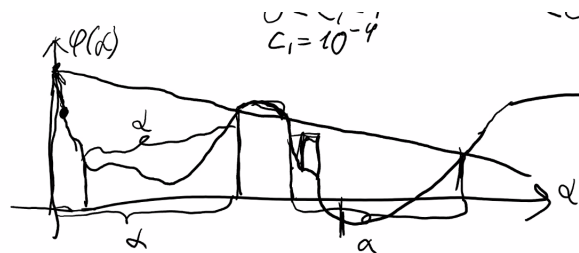
- $\varphi(\alpha) = f(x_k + \alpha p_k)$
- $f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k) p_k$.
 $0 < c_1 < 1, c_1 = 10^{-4}$.

Первое условие накладывает ограничения на выбираемое α .



Первого условия не достаточно, так как можно взять $\alpha = \varepsilon$ и это условие там будет выполняться.

Второе условие запрещает делать слишком маленькие шаги и шагать туда, где градиент продолжает достаточно быстро убывать



$$\nabla f(x_k + \alpha_k p_k) \cdot p_k \geq c_2 \nabla f(x_k) \cdot p_k \quad (2)$$

$$c_1 < c_2 < 1 \quad c_2 = 0.9$$

В целом, есть доказательство корректности условий Вольфа.