Матанализ 3
Теория оптимизации

Денис Волк
Senior Data Scientist @ KPMG

- Математик, кандидат наук, МГУ
- Специальность: динамические системы и случайные процессы
- Работал в университетах Триеста, Рима, Стокгольма
- Автор 14 научных статей в международных журналах

Денис Волк
Senior Data Scientist @ KPMG

Когда градиент не помогает

Пусть есть алгоритм с параметрами $\alpha_1, ..., \alpha_N$

Задача: подобрать параметры так, чтобы алгоритм давал лучший результат.

Можно записать в виде оптимизационной задачи

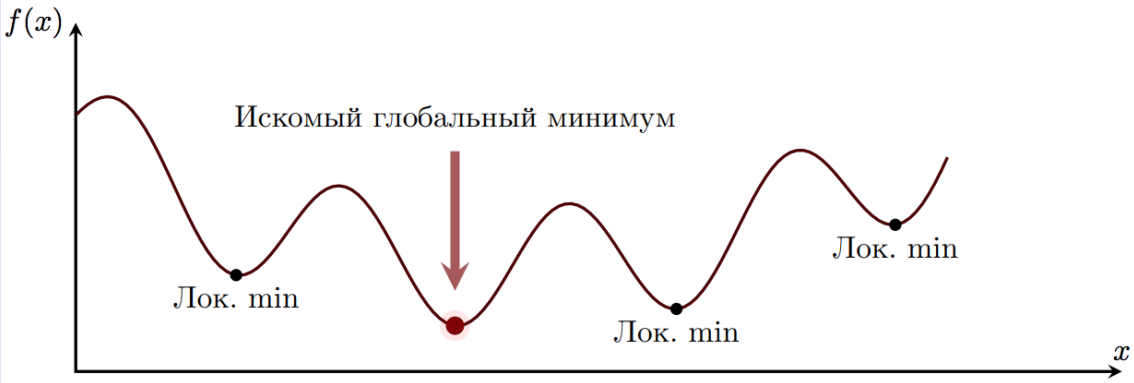
$$Q(\alpha_1, ..., \alpha_N) \to \max_{\alpha_1, ..., \alpha_N}$$

Вычисление градиента в этом случае зачастую
невозможно в принципе или крайне непрактично:

- Функция может быть негладкой
- Требуется N раз вычислять значение функции на каждом шаге

Проблема локальных экстремумов

Другая проблема градиентных методов — проблема локальных минимумов.



Градиентный спуск, попав на дно локального минимума, где градиент также равен нулю, там и остается. Глобальный минимум так и остается не найденным.

Решить эти проблемы позволяют методы **случайного поиска экстремумов**.

Общая идея этих методов заключается в намеренном введении элемента случайности.

Градиентный спуск (почти) без градиента

Хотим: $f(\vec{x}) \to \min_{\vec{x}}$

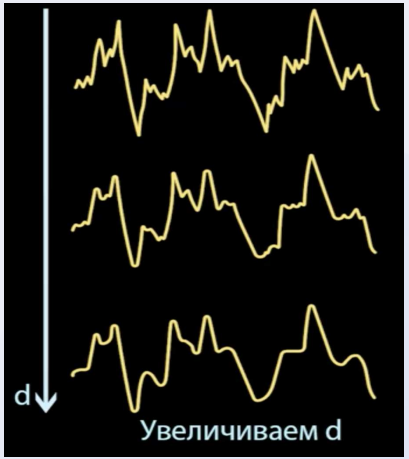
Сначала выбирается $\vec{u}$ (случайный вектор $\vec{u}$ равномерно распределен по сфере)

Затем вычисляем численную оценку антипроизводной по направлению $\dfrac{f(\vec{x}) - f(\vec{x} + d\vec{u})}{d}$

Сдвигаем точку в направлении $\vec{u}$ пропорционально антипроизводной по направлению.

Величина смещения зависит от выражения функции в точке $\vec{x} + d\vec{u}$.
В среднем смещение происходит по антиградиенту сглаженной функции.

d — параметр сглаживания

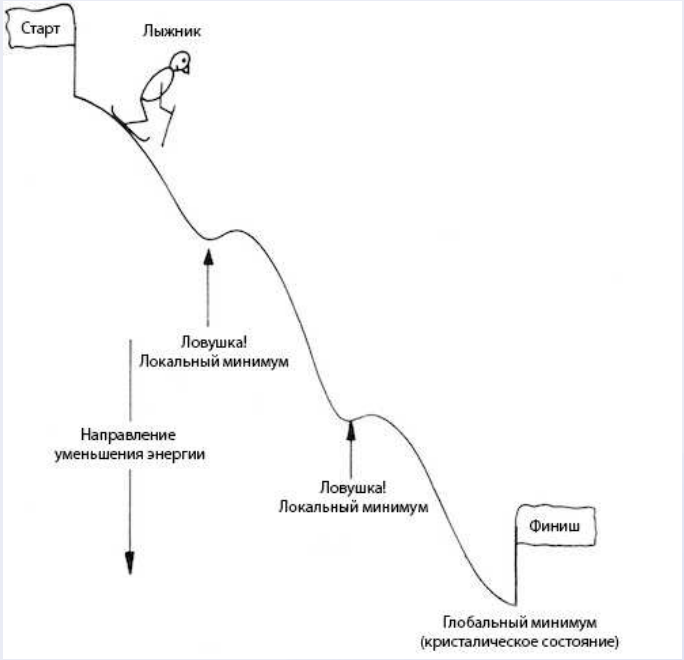


Когда много локальных минимумов

Имитация отжига (simulated annealing)

При застывании металл приходит в состояние с минимальной энергией.

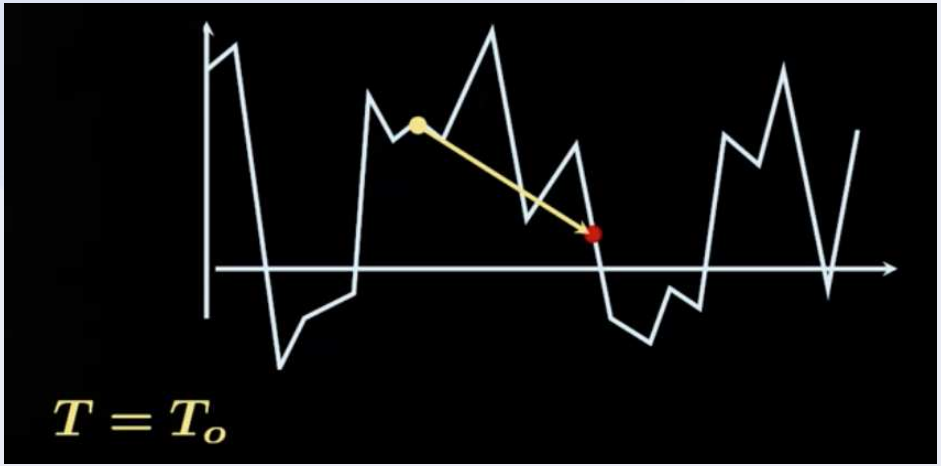
Вводим параметр **T**, который имеет смысл температуры, и в начальный момент присваиваем ему значение $\mathbf{T_0}$. Набор переменных, по которым происходит оптимизация, будет обозначаться как **x**.

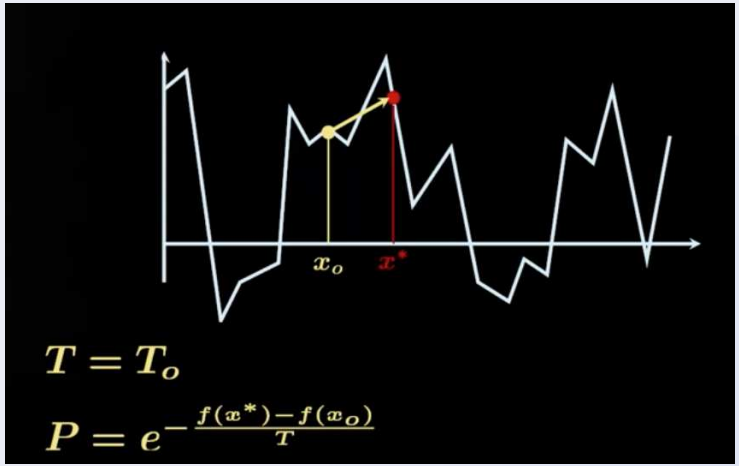


https://www.pvsm.ru/matematika/53102

Метод имитации отжига

1. Начальное приближение — произвольная точка **x**.
2. Выбираем x^* случайно из множества соседних состояний.
3. Если значение функции меньше, то переходим в x^*
4. Если значение функции больше, то переходим в x^* с вероятностью $P = e^{-\frac{f(x^*)-f(x)}{T}}$
5. Уменьшаем значение температуры **T**



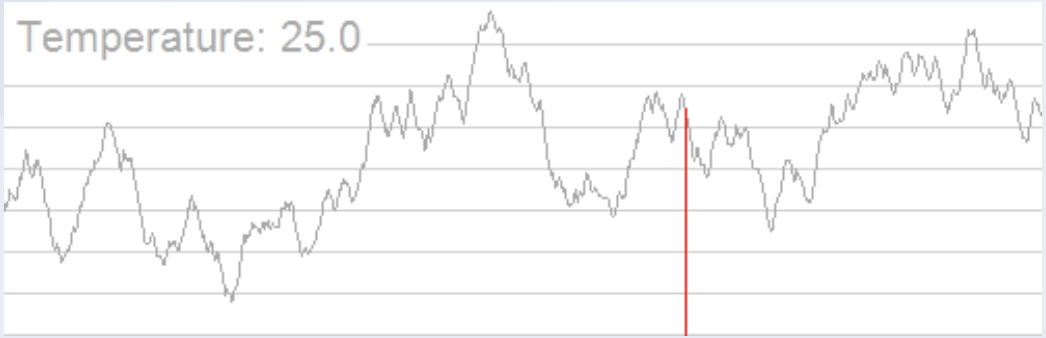


https://www.coursera.org/learn/mathematics-and-python

Метод имитации отжига

Благодаря возможности перехода в состояния с большим значением функции мы не застреваем в локальных минимумах, а исследуем всё пространство.

Постепенно с уменьшением температуры **T** уменьшается и вероятность переходов в состояния с большим значением функции. Таким образом в конце имитации отжига в качестве **x** оказывается искомый глобальный минимум.



поиск максимума

Работает даже для негладких функций!

https://en.wikipedia.org/wiki/Simulated_annealing

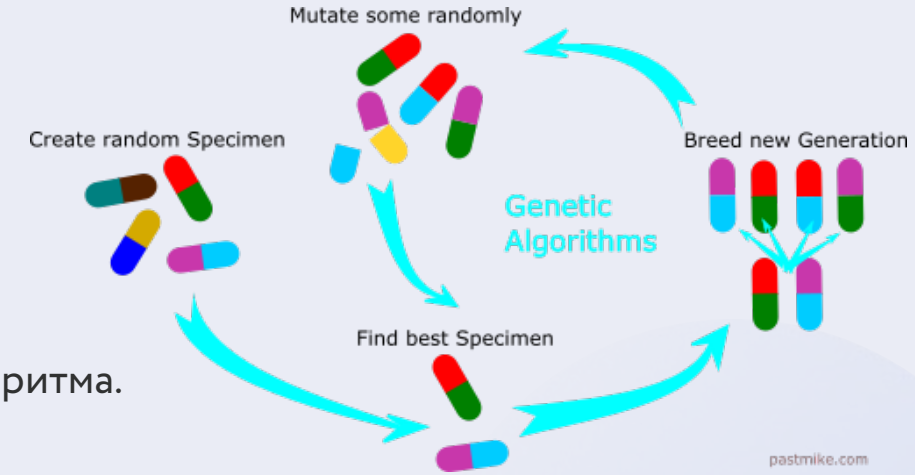
Генетические алгоритмы

Моделируют процесс естественного отбора в ходе эволюции

Стадии:

1. генерации популяции
2. отбора
3. скрещивания
4. мутаций

Порядок стадий зависит от конкретного алгоритма.



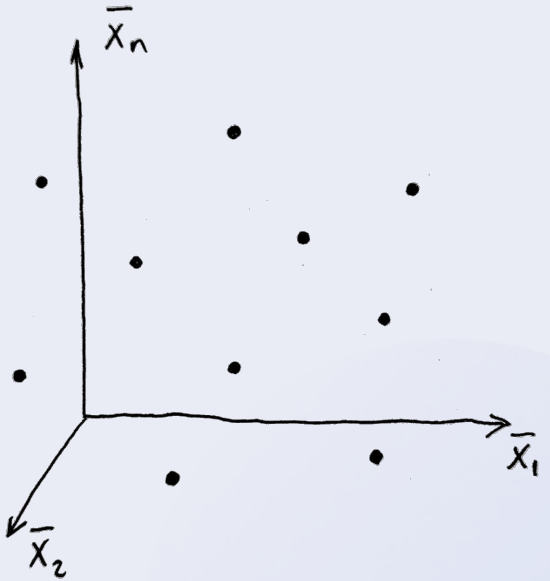
Алгоритм дифференциальной эволюции

Популяция — множество векторов-кандидатов в $\mathbb{R}^n$

Начальная популяция — **N** случайных векторов.

На каждой следующей итерации алгоритм генерирует новое поколение векторов, комбинируя векторы предыдущего поколения.

Результат — самый лучший вектор из финальной популяции.



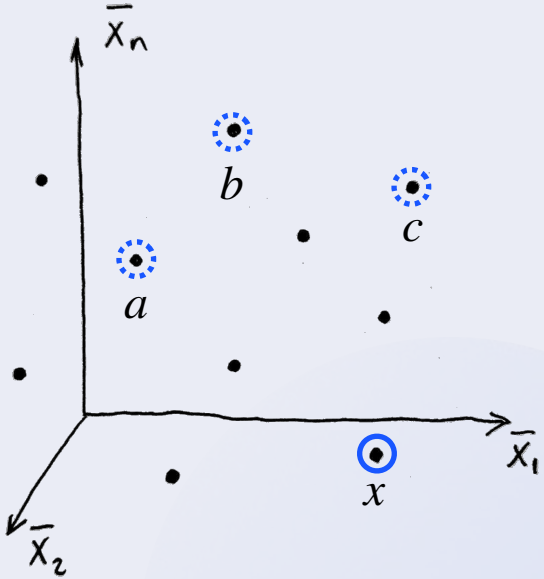
Исходная популяция **N** векторов в $\mathbb{R}^n$

Алгоритм дифференциальной эволюции

На каждой итерации для каждого вектора **x** случайно выбираются 3 другие вектора **a, b, c**.

На основе этих векторов генерируется так называемый **мутантный** вектор **m** (**F** — параметр):

$$\bar{m} = (m_i), \quad m_i = a_i + F \cdot (b_i - c_i)$$



На **стадии скрещивания** каждая координата $\mathbf{x}_i$ исходного вектора с некоторой вероятностью замещается соответствующей координатой $\mathbf{m}_i$ мутантного вектора. Получившийся вектор **y** называется пробным.

На **стадии отбора**, если **f(y) ⩽ f(x)**, то заменяем **x** на **y**.

Алгоритм дифференциальной эволюции

- В случае **бинарных** векторов можно определить мутацию следующим образом: с вероятностью **p** менять 0->1 и 1->0 в исходных векторах.

- Часто, чтобы увеличить эффективность алгоритма, он запускается одновременно на нескольких независимых популяциях.

- Эффективность метода зависит от выбора операторов мутации и скрещивания для каждого конкретного типа задач.

Метод Нелдера-Мида

Метод Нелдера-Мида, или метод деформируемого многогранника, применяется для нахождения решения задачи оптимизации вещественных функций многих переменных. На каждой итерации для функции от **n** переменных требуется вычислить значение функции в **n+1** точке.

Метод прост в реализации и полезен на практике, но для него не существует теории сходимости — алгоритм может расходиться даже на гладких функциях.

Используется по умолчанию в функции **minimize** из **scipy.optimize**.

Метод Нелдера-Мида

1) случайно выбираем **n+1** точку, образующие симплекс:

Рис. 2: Отрезок (1-симплекс)

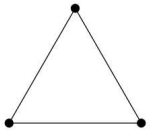


Рис. 3: Треугольник (2-симплекс)

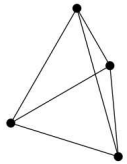
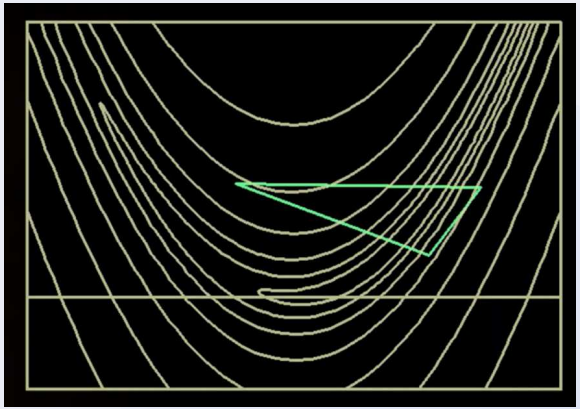


Рис. 4: Тетраэдр (3-симплекс)

2) сортируем значения функции в вершинах:

$$f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1})$$

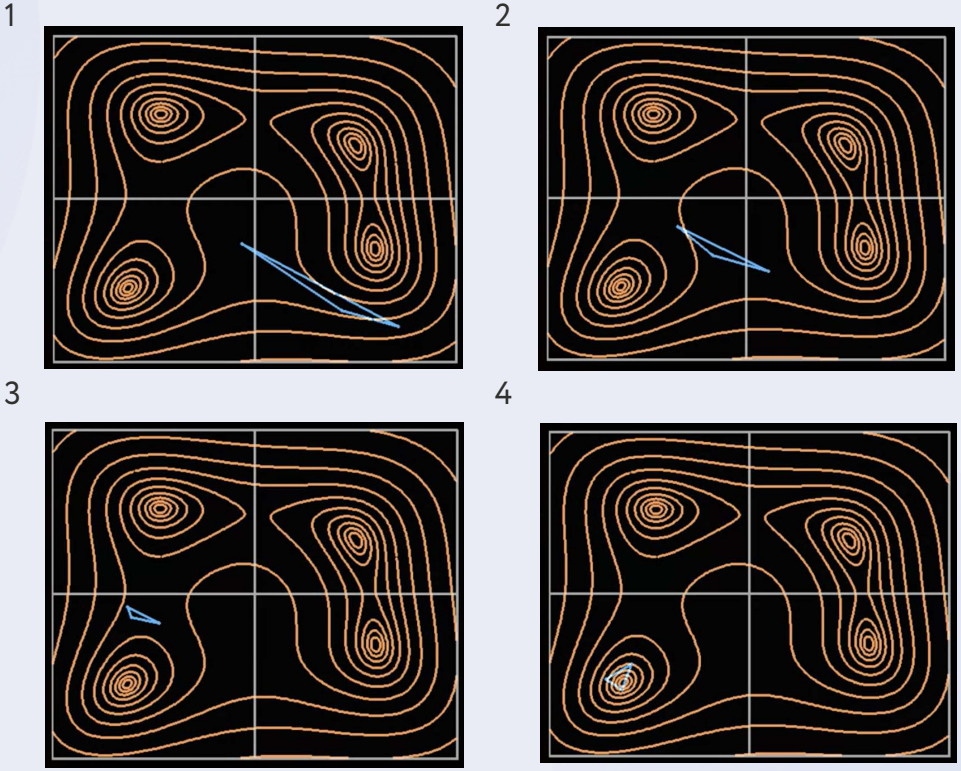
За шаг хотим уменьшить значение функции хотя бы в самой плохой точке $\mathbf{x_{n+1}}$.



Метод Нелдера-Мида

3) деформируем (отражение, сжатие, растяжение, глобальное сжатие симплекса) симплекс так, чтобы он “подползал” к минимуму функции.

В конце **проверка сходимости:** что симплекс стал достаточно маленьким.



Summary

Узнали:

1. Когда градиент не помогает
2. Метод имитации отжига
3. Алгоритм дифференциальной эволюции
4. Метод Нелдера-Мида