Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования
Прикладная математика и информатика

Методы оптимизации

Отчет по лабораторной работе №1

⟨Собрано 17 марта 2023 г.⟩

Работу выполнили:
Бактурин Савелий Филиппович M32331
Вереня Андрей Тарасович M32331
Сотников Максим Владимирович M32331

Преподаватель:
Свинцов Михаил

Задача 1

Постановка задачи

Реализуйте градиентный спуск с постоянным шагом (learning rate).

Решение

Поймем, сначала, что мы хотим добиться: мы хотели бы найти направление наискорейшего спуска с некоторой точки к минимуму на заданной плоскостью функцией f. Однако, при решении этой задачи возникает проблема с производительностью нахождения $\operatorname{argmin} f$ за счет появления тех или иных накладных расходов на подсчет не целочисленных значения, а также проблемой с нахождением такого шага λ, что наш алгоритм не «застрянет» в бесконечном поиске интересующей точки.

Введем обозначения, пусть α – есть некоторая константа, порядка 10^{-3}, $x_i = \{x_i^0, x_i^1, \ldots, x_i^{n-1}\}$ – некоторая координата в n-мерном пространстве, p_i – наше текущее направление.

Теперь рассмотрим идею *градиентного спуска*: оптимизацию нахождения необходимого минимум за k шагов мы будем осуществлять шаги в n-мерном пространстве в направлении, задаваемый как антиградиент функции f в точке, задаваемая предыдущем шагом, то есть

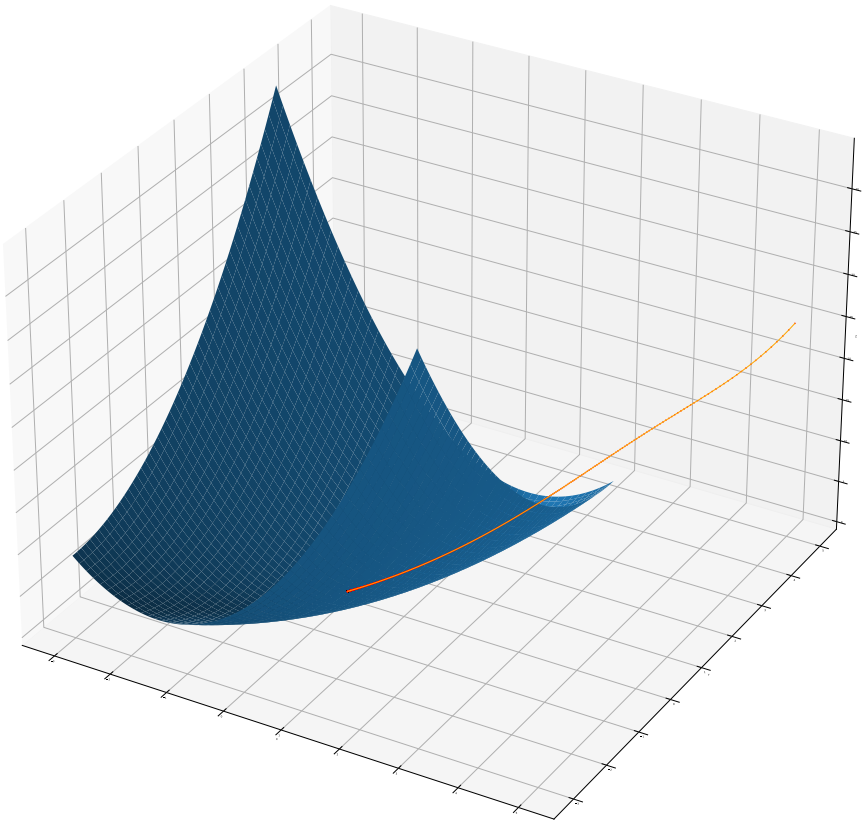
$$x_{i+1} = x_i - \alpha \cdot \nabla f(x_i),$$

где x_0 будет задаваться некоторым множеством $\text{INIT} = \{x_0^0, x_0^1, \ldots, x_0^{n-1}\}$ – то есть точка, от которой мы собираемся двигаться.

Итого, псевдо-алгоритм для этой задачи выглядит следующим образом:

```
function f(x):
        /*implementation defined*/

function ∇f(x):
        return [f(x)∂/∂x^0, f(x)∂/∂x^1, ..., f(x)∂/∂x^{n-1}]

function main:
        x_0 ← INIT
        α ← const
        forall i ∈ [1, k] do
                x_i ← x_{i-1} − α · ∇f(x_{i-1})
```

Разберем пример. Мы хотим найти методом градиентного спуска приближенный $\operatorname*{argmin}_{\substack{x \in X \\ y \in Y}} f(x, y)$ функции $f(x, y) = x^2 - (x - y)^2$. Его направлением-антиградиентом будет $p_i = -\nabla f(x, y) = \{(-1) \cdot (2 \cdot x + 2 \cdot (x - y)), (-1) \cdot (-2 \cdot (x - y))\}$. В исходном коде `points` представляет из себя шаги, проделываемые алгоритмом от стартового состояния x_0 до некоторого приближенного x_k – являющийся минимумом. Наконец, представим всему миру полученную картину.



Задача 2

Постановка задачи

Реализуйте метод одномерного поиска (метод дихотомии, метод Фибоначчи, метод золотого сечения) и градиентный спуск на его основе.

Решение

Представим себе задачу: мы хотим на непрерывной функции, которая обязательно сходится к некоторому минимуму M, найти на некотором интервале её корень. Решая эту задачу простым способом, мы бы могли уйти в долгий, бесконечный или даже бесполезный процесс нахождения приближенного корня заданной функции $f(x)$, ведь на момент поиска мы не знаем: с какой точностью брать значение (то есть, шаг

нашего поиска), в какой окрестности лежит корень на интервале (то есть, к чему нам следует сходится, чтобы успешно найти то, что мы ищем). Для этого мы применим метод *дихотомии*, или же *методом деления отрезка на две части*.

В общем случае, этот метод описывается так – посмотрим на текущий (может быть начальный, может быть измененный на некотором шаге) интервал $[l, r]$, выберем середину данного отрезка x и сравним со знаком функции в одном из концов: при совпадении, мы перемещаем один конец интервала на точку x, в ином случае – другой. Отличие начинаются там, где мы решаем подзадачи вида поиска *экстремума функции многих переменных*:

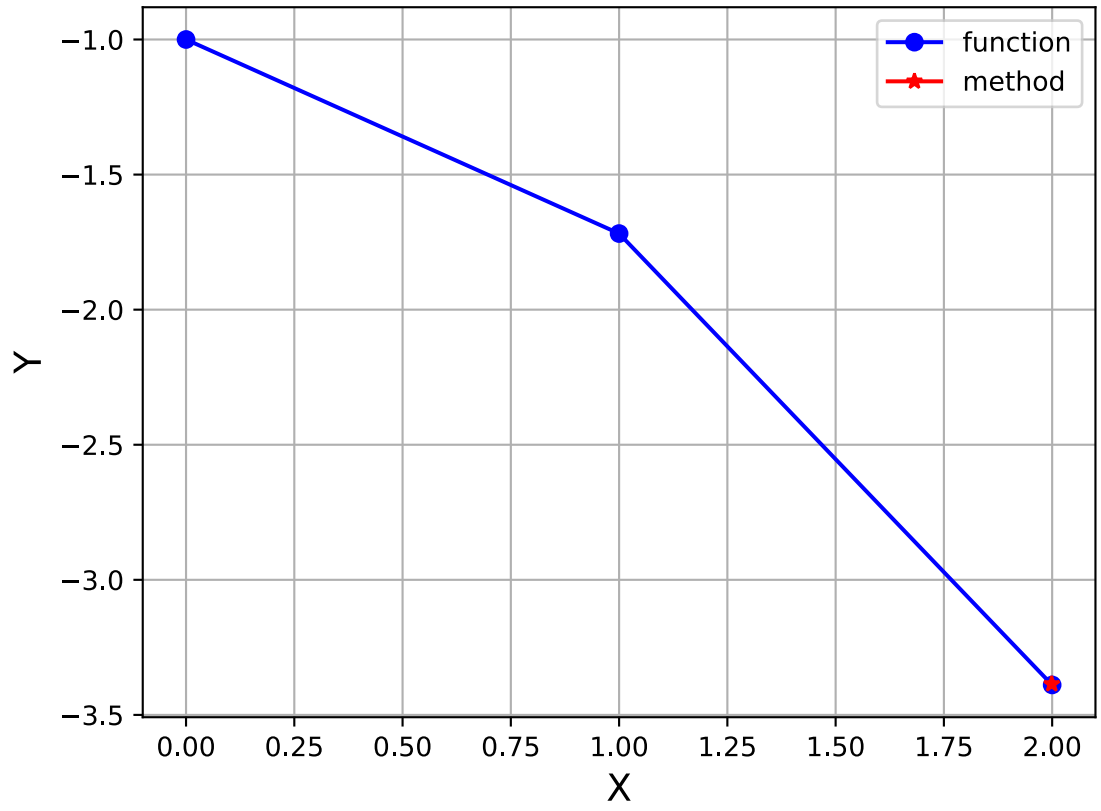
$$\eth = \underset{\eth \in [l,r]}{\operatorname{argmin}} f(x)$$

В этом случае на последнем шаге мы смотрим не на знак, а на значение функции $f(x \pm \varepsilon)$ и также здесь смотрим на то, что мы хотим найти: если минимум, то перебрасываем правый конец в рассмотренную точку x, в ином случае – левый.

Пусть ε – некоторая окрестность минимума/максимума; $l, r \in \mathbb{R}$; задана $f(x) : \mathbb{R} \to \mathbb{R}$. Тогда, псевдо-алгоритм для задачи реализации метода одномерного поиска выглядит следующим образом:

```
function f(x):
        /*implementation defined*/

function find(l, r, ε, isMin):
        x ← l
        forall ∞ do
                m ← (x + r) / 2
                f1 ← f(m + ε)
                f2 ← f(m − ε)

                if isMin then
                        if |f1 − f2| < ε then
                                r ← m
                        else
                                x ← m
                then
                        if |f1 − f2| < ε then
                                x ← m
                        else
                                r ← m

                if |x − r| ⩽ ε then
                        break
        return x
```

Рассмотрим пример одномерного поиска. Возьмем функцию $f(x) = x^2 - e^x$ и промежуток $[0, 2]$. Также, к качестве окрестности точки x' (найденная) $\varepsilon = 0.001$. Тогда, по нашему алгоритму выходит вот такой график:



Здесь синим цветом указана функция $f(x)$ на промежутке $[0, 2]$, а красным – найденный экстремум (в данном случае, минимум) функции, которая приблизительно равна $x' \approx 1.9990234375$.

Теперь рассмотрим задачу немного другого вида. Скажем, что у нас функция $find$ теперь ищет только минимум на заданном промежутке. Тогда, рассмотрим координату $x_i = \{x_i^0, x_i^1, \ldots, x_i^{n-1}\}$ в n-мерном пространстве, удовлетворяющее свойству:

$$x_i \in \underset{\substack{x \in X \\ y \in Y}}{\operatorname{argmin}} f(x_i)$$

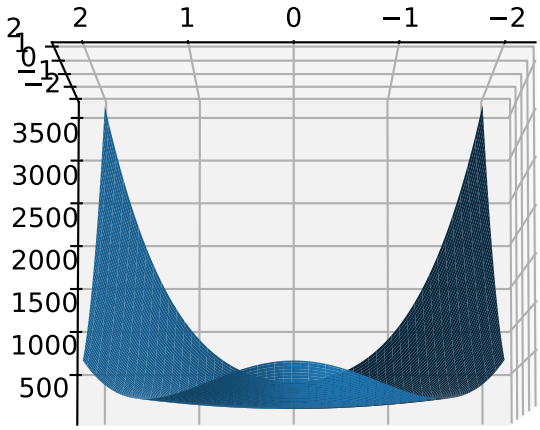
Задача: реализовать градиентный спуск на основе метода дихотомии. Как уже было сказано ранее, данный метод ищет на отрезке непрерывной функции, которая к чему-то на данном интервале сходится. Рассмотрим интервал $[x_0, x_i]$ – это путь, по которому мы передвигаемся по направлению антиградиента функции $f(x_i)$ с некоторым постоянным шагом α. В чем проблема такой незамысловатой инженерной мысли? В том, что мы можем двигаться чрезвычайно медленно и не очень правильно: дело в том, что в алгоритме мы указываем число шагов, за которые мы бы очень хотели дойти до минимума, что, справедливо, не факт, что мы дойдем до него (хотя направление вполне можем иметь верное). Для исправления подключается метод дихотомии и говорит: давайте на каждом шаге мы будем выбирать такую точку x_i, где значение функции меньше всего из рассматриваемого отрезка (которое мы изначально зададим как $[l,\ r] : \exists [l',\ r']$ на котором лежит минимум) и уже зная аргумент, куда потенциально приведет нас эта точка, пойдем туда по направлению

анти-градиента. Таким образом, мы значительно сокращаем количество итераций, как раз за счет того, что мы на каждом шаге пытаемся найти минимум.

Итак, пусть $x_i = \{x_i^0,\ x_i^1,\ \ldots, x_i^{n-1}\}$ – i-ая координата в n-мерном пространстве; задана функция $f(x_i) : \mathbb{R}^n \to \mathbb{R}$; координаты $\texttt{INIT} = \{x_0^0,\ x_0^1,\ \ldots,\ x_0^{n-1}\}$ — начальная точка, от которой нам следует двигаться; число шагов k, шаг α и окрестность ε. Тогда, псевдо-алгоритм для этой задачи выглядит следующим образом:

```
function f(x):
        /*implementation defined*/

function ∇f(x):
        return [f(x)∂/∂x^0,  f(x)∂/∂x^1,  ...,  f(x)∂/∂x^(n-1)]

function find(a, ε):
        x ← a
        forall ∞ do
                m ← (x + r)/2
                f1 ← f(m + ε)
                f2 ← f(m − ε)

                if |f1 − f2| < ε then
                        x ← m
                else
                        r ← m

                if |x − r| ⩽ ε then
                        break
        return x

function main:
        x0 ← INIT
        α ← const
        ε ← const
        forall i ∈ [1, k] do
                xi ← x0 − α · ∇f(x0)
        x0 ← find(x0,  r,  ε)
```

Рассмотрим пример использования алгоритма. Пусть $f(x, y)$ – это функция Розенброка, или $(1 - x)^2 + 100 \cdot (y - x^2)^2$. Данная функция является отличным примером для проверки правильности алгоритма, засчет того, что минимум $f(x, y) = 0$ в точке $(1,\ 1)$.



Задача 3

Постановка задачи

Проанализируйте траекторию градиентного спуска на примере квадратичных функций. Для этого придумайте две-три квадратичные функции от двух переменных, на которых работа методов будет отличаться.

Решение

Задача 4

Постановка задачи

Для каждой функции:

(a) исследуйте сходимость градиентного спуска с постоянным шагом, сравните полученные результаты для выбранных функций;

(b) сравните эффективность градиентного спуска с использованием одномерного поиска с точки зрения количества вычислений минимизируемой функции и ее градиентов;

(c) исследуйте работу методов в зависимости от выбора начальной точки;

(d) исследуйте влияние нормализации (scaling) на сходимость на примере масштабирования осей плохо обусловленной функции;

(e) в каждом случае нарисуйте графики с линиями уровня и траекториями методов;

Решение

Задача 5

Постановка задачи

Реализуйте генератор случайных квадратичных функций n переменных с числом обусловленности k.

Решение

Изначально поймем, что такое *число обусловленности*. По своей сущности, это нечто, что может показать насколько может измениться значение функции при небольшом изменении аргумента. Для нахождения такого числа и, в следствии, нахождения некоторого вектора чисел, которые будут являться коэффициентами квадратичной формы, существует несколько способов: через матричный нормы – это исходит напрямую из рассматриваемого линейного уравнения вида $\mathfrak{A}x = \mathbf{b}$, где $\mathfrak{A}$ – линейный оператор, $\mathbf{b}$ – вектор и x – переменная. Такой способ подошел бы нам, если б мы знали заранее нормы матриц. Из-за чего нам понадобится более сильное средство, именуемое как *сингулярное разложение*, а именно: возьмем некоторую матрицу A, тогда его число обусловленности будет равно отношению максимального и минимального из диагональных элементов; остальные же элементы на диагональной части матрицы будут коэффициентами разложения квадратичной формы.

Итак мы хотим сгенерировать матрицу такую, что $k = \frac{\max_{\forall i \in [0,n]} x_{i,i \in [0,n]}}{\min_{\forall i} x_{i,i}}$. Для этого мы получим максимальный элемент $\text{MAX} = k \cdot \text{MIN}$, а в качестве минимального возьмем случайное число из ограниченного операционной системой диапазоном, например $[0, 2^{64} - 1]$. Тогда как все остальные элементы следует брать из диапазона $[\text{MIN} + 1, \text{MIN} \cdot k)$.

Итого, псевдо-алгоритм для этой задачи выглядит следующим образом:

```
function random(l, r):
        return randomized R ∈ [l, ..., r)

function main(n, k):
        MIN ← random(0, 2^64 − 1)
        MAX ← MIN · k
        q ← [MIN, MAX, x_0 ..., x_{n−3}], ∀x_i = 0
        forall i ∈ [2, n] do
                q_i ← random(MIN + 1, MAX)
```

Для примера мы рассмотрим задачу. Необходимо сгенерировать квадратичную функцию с $n = 10$ переменными и $k = 5$ числом обусловленности. Воспользуемся заготовленной программой, которая выводит функцию в TEXвиде, и получим, как один из результатов вот такой:

$$\begin{aligned} f(x_0, x_1, \ldots, x_9) = {} & 55881652088902977 \cdot x_0^2 + 279408260444514885 \cdot x_1^2 \\ & + 168040993445098276 \cdot x_2^2 + 98341599851142922 \cdot x_3^2 \\ & + 126881448626083312 \cdot x_4^2 + 112063718763017541 \cdot x_5^2 \\ & + 167683450954662177 \cdot x_6^2 + 104734621927684905 \cdot x_7^2 \\ & + 196100745993640030 \cdot x_8^2 + 199634427735860479 \cdot x_9^2 \end{aligned}$$

Задача 6 и 7

Постановка задачи

Исследуйте зависимость числа итераций $T(n, k)$, необходимых градиентному спуску для сходимости в зависимости от размерности пространства $2 \leqslant n \leqslant 10^3$ и числа обусловленности оптимизируемой функции $1 \leqslant k \leqslant 10^3$.

Решение

Для решения этой задачи нам понадобится решение *первой задачи*, а именно метод, используемый там. Кратко напомним, что такое метод градиентного спуска. Пусть у нас задана точка $x_i = \{x_i^0,\ x_i^1,\ \ldots,\ x_i^{n-1}\}$ – координата в n-мерном пространстве – тогда, чтобы найти приближенно какой-то минимум (ведь, их бывает по жизни несколько) двигаясь по направлению, противоположный наибыстрейшему подъему, с неким α – константой, значащий шаг в нашем `for`-е, и какое-то предельное число шагов k. Формулой для всего этого, как мы знаем, была

$$x_i = x_{i-1} - \alpha \cdot \nabla f(x_{i-1}),$$

где $x_0 = \text{INIT} = \{x_0^0,\ x_0^1,\ \ldots,\ x_0^{n-1}\}$ – начальная точка, от которой нам бы хотелось найти минимум.

Для исследования, мы зададим для $\forall\langle n, k\rangle$ некоторое число шагов, которое является предельным. Кроме того, мы можем воспользоваться критерием остановки ??? для понимания, есть ли нам еще смысл идти дальше, или мы можем остаться с нынешними значениями (наше значение измениться не более, чем на ε).

Итак, пусть $\hat{k}$ – максимальное количество шагов, допустимые; n, k – входные данные; $x_i = \{x_i^0,\ x_i^1,\ \ldots,\ x_i^{n-1}\}$ – i-ая координата в n-мерном пространстве; $\text{INIT} = \{x_0^0,\ x_0^1,\ \ldots,\ x_0^{n-1}\}$ – начальная координата, с которой нам следует двигаться; $f(x)$ – сгенерированная функция с n переменными и числом k обусловленности; $\nabla f(x)$ – градиент сгенерированной функции; $g(x)$ – функция проверки критерия остановки; α – шаг. Тогда, псевдо-алгоритм для этой задачи выглядит следующим образом:

```
function f(x):
        /*implementation defined*/
```

```
function ∇f(x):
        return [f(x)∂/∂x^0,  f(x)∂/∂x^1,  ...,  f(x)∂/∂x^{n-1}]

function random(l, r):
        return randomized R ∈ [l, ..., r)

function get(n, k):
        MIN ← random(0, 2^64 − 1)
        MAX ← MIN · k
        q ← [MIN,  MAX,  x_0  ...,  x_{n-3}],  ∀x_i = 0
        forall i ∈ [2, n − 1] do
                q_i ← random(MIN + 1,  MAX)
        return Q : L → K  ⟺  Σ_{i=0}^{n-1} q_i · x^i_{i,  i}

function main:
        forall ⟨n′,  k′⟩ ∈ {⟨n,  k⟩}:
                x_0 ← INIT
                α ← const
                forall i ∈ [1, k̂] do
                        if g(x_{i-1}) then
                                break
                        x_i ← x_{i-1} − α · ∇f(x_{i-1})
```

Дополнительное задание

Постановка задачи

Реализуйте одномерный поиск с учетом условий Вольфе и исследуйте его эффективность. Сравните полученные результаты с реализованными ранее методами.

Решение