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

Методы оптимизации

Отчет по лабораторной работе №1

⟨Собрано 27 марта 2023 г.⟩

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

Преподаватель:
Казанков Владислав Константинович

Задача 1

Постановка задачи

Реализуйте градиентный спуск с постоянным шагом (learning rate).

Решение

Поймем, изначальное, чего мы хотим добиться: мы хотели бы найти приближенный минимум на плоскости у заданной непрерывной функции f. Однако, при наивном 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 всецело такого алгоритма, который бы с точность до некоторого изменения ε не «застрянет» в бесконечном поиске интересующей точки.

Для таких целей самым простым, но действующим метод является *градиентный спуск с постоянным шагом*. Введем обозначения, пусть λ – 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: 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

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

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.

В качестве промежуточного итога предоставим псевдо-алгоритм для решения этой задачи:

function $f(x)$:
    $/*implementation\ defined*/$

function $\nabla f(x)$:
    return $\left[f(x)\dfrac{\partial}{\partial x^0},\ \ f(x)\dfrac{\partial}{\partial x^1},\ \ \ldots,\ \ f(x)\dfrac{\partial}{\partial x^{n-1}}\right]$

function $gradient(f(x))$:
    $x_0 \leftarrow$ INIT
    $\lambda \leftarrow$ const
    $\forall i \in [1, k]$:
        $x_i \leftarrow x_{i-1} - \lambda \cdot \nabla f(x_{i-1})$

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).

Простое решение мы могли бы с легкостью заменить на тот же градиентный спуск с постоянным шагом. Однако, даже с ним мы иногда можем проиграть не только по точности приближенного минимума (если мы ищем минимум за определенное k число шагов), но и вовсе уйти в бесконечный цикл (если мы ищем минимум, пока не будет выполнен критерий $|x_i - x_{i-1}| \leqslant \varepsilon$, где $x_i = \{x_i^0,\ x_i^1,\ \ldots,\ x_i^{n-1}\}$ – некоторая координата в n-мерном пространстве). По жизни чаще всего случается и другая не менее важная задача: мы хотим найти минимум как можно быстрее и скорее, причем, с точностью до предельно малого ε, в отличие от честного ожидания градиентного спуска.

Для решения такой столь непосильной задачи мы воспользуемся *методом дихотомии*, или же, как его еще называют, *методом деления отрезка на две части.*

Но для начала мы посмотрим на работу в одномерном пространстве. 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.

Обозначим за ε – как некоторая окрестность минимума, $l, r \in \mathbb{R}$, задана $f(x) : \mathbb{R} \to \mathbb{R}$. Тогда, в качестве промежуточного итога предоставим псевдо-алгоритм для решения этой задачи:

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

function dichotomy2d(l, r, ε):
        x ← l
        ∀ ∞:
                m ← (x + r) / 2
                f₁ ← f(m + ε)
                f₂ ← f(m − ε)

                if |f₁ − f₂| < ε then
```

```
                r ← m
        else
                x ← m
```

Теперь рассмотрим иной, более общий случай, n-мерный. Порассуждаем, в чем была проблема градиентного спуска: в описанном выше алгоритме цикл хода по направлению антиградиента (то есть, наискорейшего спуска) ничего не мог сделать в тех случаях, когда до приближенного минимума остается сделать не один гигантский шаг в *learning rate* величину, а поменьше, в точности до некоторого малого ε.

Для исправления столь шальной ситуации, когда алгоритм, который работает не на количество шагов, а – на критерий, то есть когда время ожидания отклика программы потребует века, мы будем либо уменьшать шаг в только тех случаях, когда на заданном шаге есть пренеприятный шпион в виде локального минимум на это отрезке, либо не изменять, то есть когда локального минимума нет, а значит нам незачем как-то останавливаться на достигнутом и куда-то сворачивать. Для этого введем специальную функцию $g : [0, 1]$, которая будет возвращать *scale* нашего шага, причем, так как мы ищем лишь окрестность желаемой точки, то возвращаемое значение будет также учитывать поданный нам ε. Обозначим за $x_i = \{x_i^0,\ x_i^1,\ \ldots,\ x_i^{n-1}\}$ – координата точки в n-мерном пространстве, $f(x)$ – заданная функция, $\nabla f(x)$ – градиент функции. Идея – изначально мы посчитаем $f(x_{i-1})$ и $\nabla f(x_{i-1})$, передадим их в функцию для подсчета *scale*, а дальше что есть сил мы будем делать это много раз:

(1) Положим $m \leftarrow \dfrac{l+r}{2}$ и $\alpha \leftarrow m \pm \varepsilon$.

(2) В качестве a и b разность поданного сверху $f(x_{i-1})$ и произведения $\nabla f(x_{i-1})$ и α.

(3) Рассмотрим два случая: в том случае, если $a < b$, то это значит, что ..., однако в ином же случае – .... Таким образом, в первом – мы смещаем левую границу, а в втором – правую.

В качестве промежуточного итога предоставим псевдо-алгоритм для решения этой задачи:

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

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

function scale(p_1,  p_2):
        l, r ← 0, 1
        ∀ ∞:
                m ← (l + r)/2
                α ← λ · (m ± ε)
                a, b ← p_1 − p_2 · α
```

```
            if $a < b$ then
                $l \leftarrow m$
            else
                $r \leftarrow m$

            if $|l - r| \leqslant \varepsilon$ then
                break

function gradient_dichotomy:
        $x_0 \leftarrow$ INIT
        $\lambda \leftarrow$ const
        $\forall\ i \in [1, k]$:
            $x_i \leftarrow x_{i-1} - \lambda \cdot$ scale$(f(x_{i-1}), \nabla f(x_{i-1}))$
```

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

Для начала поймем, что такое *число обусловленности*. По своей сущности, это нечто, что может показать насколько измениться значение функции при небольшом изменении аргумента. Для нахождения такого числа и, в следствии, нахождения вектора чисел, которые будут являться коэффициентами квадратичной сгенерированной функции. Для решении такой задачи мы могли воспользоваться правильными методами такими как *теорема о сингулярном разложении*: возьмем некоторую матрицу A, возьмем его после разложении образовавшийся диагональную матрицу, тогда его (матрицы A) число обусловленности будет равно отношению максимального по модулю и минимального по модулю собственных чисел выбранной матрицы. Проблемой столь мощного инструментария заключается в том, что генерация подобной матрицы наивным методом может занимать немереное время, ибо асимптотику спектрального разложения, которое и является основным в сингулярном, никто предугадать не может.

Тогда приходит идея менее безболезненная, но более радикальная: скажем, что наша матрица изначально была подана диагональной, а значит наша генерации функции сводится к вычислению n-чисел на диагонали матрицы.

Итак, пусть дано число обусловленности k, положим $\mathrm{MAX} = k \cdot \mathrm{MIN}$ – максимальное по модулю значение собственного числа матрицы, а в качестве минимального возьмем случайное число из ограниченного операционной системой диапазоном, например $[0, 2^{\log_2 X} - 1]$. Тогда, наконец, все остальные элементы следует брать из диапазона $[\mathrm{MIN} + 1, \mathrm{MIN} \cdot k)$.

В качестве промежуточного итога предоставим псевдо-алгоритм для решения этой задачи:

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

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

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

Дополнительное задание

Постановка задачи

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

Решение