Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
Московский государственный технический университет имени Н.Э. Баумана

Лабораторная работа №6
по курсу «Численные методы»
«Метод наискорейшего спуска поиска минимума функций многих переменных»

Выполнил:
студент группы ИУ9-62Б
Головкин Дмитрий

Проверила:
Домрачева А.Б.

Москва, 2023

Цель:

Анализ метода наискорейшего спуска поиска минимума функций многих переменных.

Постановка задачи:

Дано: Функция нескольких переменных $f(x_1, x_2, ..., x_n)$ и некоторое начальное приближение $X^0 = (x_1^0, ..., x_n^0)$.

Найти: Минимум функции нескольких переменных с заданной точностью.

Тестовый пример:

В качестве тестового примера были предложены следующие входные данные:

$$f(X) = e^{x_1} + (x_1 + x_2)^2, \quad X^0 = (1, 1).$$

Описание методов:

Пусть мы имеем некоторое приближение к минимуму $X^k = (x_1^k, ..., x_n^k)$. Рассмотрим функцию одной переменной

$$\phi_k(t) = f(x_1^k - t\frac{\partial f}{\partial x_1}(X^k), ..., x_n^k - t\frac{\partial f}{\partial x_n}(X^k)) = f(X^k - tgradf(X^k)),$$

где вектор $gradf(X^k) = (\frac{\partial f}{\partial x_1}(X^k), ..., \frac{\partial f}{\partial x_n}(X^k))$ - градиент функции f в точке X^k.

Обозначим точку минимума функции $\phi_k(t)$ через t^*. Тогда имеем следующую форму

$$X^{k+1} = X^k - t^* gradf(X^k).$$

Процесс поиска минимума продолжаем до тех пор, пока $||gradf(X^k)|| = \max_{1 \le i \le n} |\frac{\partial f}{\partial x_i}(X^k)|$ не станет меньше допустимой погрешности ϵ.

В большинстве случаев точно искать минимум функции $\phi_k(t)$ не нужно и достаточно ограничиться лишь одним приближением. Тогда особенно простым будет вид итерации в двумерном случае

$$(x_{k+1}, y_{k+1}) = (x_k - t^k\frac{\partial f}{\partial x}, y_k - t^k\frac{\partial f}{\partial y}),$$

где $t^k = \frac{\phi_k'(0)}{\phi_k''(0)}$, $\quad \phi_k'(0) = -(\frac{\partial f}{\partial x})^2 - (\frac{\partial f}{\partial y})^2$, $\quad \phi_k''(0) = \frac{\partial f^2}{\partial x^2}(\frac{\partial f}{\partial x})^2 + 2\frac{\partial f^2}{\partial x \partial y}\frac{\partial f}{\partial x}\frac{\partial f}{\partial y} + \frac{\partial f^2}{\partial y^2}(\frac{\partial f}{\partial y})^2$, все производные беруться в точке (x_k, y_k).

Практическая реализация: Листинг 1. Метод наискорейшего спуска поиска минимума функций двух переменных

```
#!python
# -*- coding: utf-8 -*-

import matplotlib
import numpy as np
from matplotlib import pyplot as plt
```

```
from matplotlib.colors import ListedColormap

plt.rcParams["figure.figsize"] = (10,10)
plt.figure(figsize=(20, 20))
random_seed = 42

def func(x):
    return np.exp(x[0]) + (x[0] + x[1])**2

np.random.seed(random_seed)
w_0 = np.asarray([1,1])
w = w_0.copy()
w_list = [w.copy()]
eps = 1e-3
k = 0
max_der = 1

while max_der > eps:
    k += 1
    der1_x = np.exp(w[0]) + 2*(w[0] + w[1])
    der2_xx = np.exp(w[0]) + 2
    der1_y = 2*(w[0] + w[1])
    der2_yy = 2
    der2_xy = 2
    max_der = max(der1_x, der1_y)
    phi1_r = -(der1_x)**2 - (der1_y)**2
    phi2_r = der2_xx*(der1_x)**2 + 2*der2_xy*der1_x*der1_y + der2_yy*(der1_y)*
    lr = -phi1_r / phi2_r
    #print(der1_y.astype(np.float64), der1_x)
    w = w - lr * np.asarray([der1_x.astype(np.float64), der1_y.astype(np.float
    w_list.append(w.copy())

print(k, 'Num steps')
w_list = np.array(w_list)
w

def plot_convergence_2d(func, steps, ax, xlim, ylim, cmap="viridis", title=""):

    ax.set_title(title, fontsize=20, fontweight="bold")
```

```
    xrange = np.linspace(*xlim, 100)
    yrange = np.linspace(*ylim, 100)
    grid = np.meshgrid(xrange, yrange)
    X, Y = grid
    fvalues = np.array([func(x) for x in
        np.dstack(grid).reshape(-1, 2)]).reshape((xrange.size, yrange.size))
    ax.pcolormesh(xrange, yrange, fvalues, cmap=cmap, alpha=0.8)
    CS = ax.contour(xrange, yrange, fvalues)
    ax.clabel(CS, CS.levels, inline=True)
    arrow_kwargs = dict(linestyle="--", color="black", alpha=0.8)
    for i, _ in enumerate(steps):
        if i + 1 < len(steps):
            ax.arrow(
                *steps[i],
                *(steps[i+1] - steps[i]),
                **arrow_kwargs
            )
    n = len(steps)
    color_list = [(i / n, 0, 0, 1 - i / n) for i in range(n)]
    ax.scatter(steps[:, 0], steps[:, 1], c=color_list, zorder=10)
    ax.scatter(steps[-1, 0], steps[-1, 1],
               color="red", label=f"estimate = {np.round(steps[-1], 2)}")
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    ax.set_ylabel("$y$")
    ax.set_xlabel("$x$")
    ax.legend(fontsize=16)

fig, axes = plt.subplots(figsize=(10, 10), squeeze=False)
fig.suptitle("Steepest Descent 2D", fontsize=25, fontweight="bold")
plot_convergence_2d(func, w_list, axes[np.unravel_index(0, shape=axes.shape)]
```

Результаты:

В результате работы программы (Листинг 1), был вычислен приближенно с точность $\epsilon = 0.001$ минимум заданной функции. Приближенным минимумом является точка [-5.68862217, 5.68983688]. Алгоритм сошелся за достаточно большое число шагов - 587. Это связанно с тем, что функция имеет область с очень слабо изменяющимся градиентом, и глобального минимума не существует, так как из вида функции ясно, что минимум будет в точке $[-\infty, \infty]$. Таким образом, если увеличивать точность, вычисленная алгоритмом точка будет приближаться к

точке $[-\infty, \infty]$.

Выводы:

В ходе выполнения лабораторной работы был рассмотрен метод наискорейшего спуска для поиска минимума функции двух переменных. Для этого метода была написана реализация на языке программирования Python. Анализируя результаты полученной программы, можно заметить, что работа метода сильно зависит от начальных условий, а также от вида функции, которую необходимо минимизировать. Так, если у функциии достаточно маленький градиент, то может потребоваться заметно большее число операций для достижения необходимой точность, чем если бы у функции был четко выраженный минимум. Также немаловажен и выбор начальной точки, так как при неудачном выборе алгоритм моджет сойтись к локальному, а не глобальному минимуму.