МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт Информационных Технологий

Кафедра МПО ЭВМ

Дисциплина «Математические методы решения задач искусственного интеллекта»

Лабораторная работа №3

«Исследование численных методов оптимизации»

Исполнитель:

студент группы 1ПИб-02-3оп-22

Подтелков Владислав Владимирович

Руководитель:

Юдина Ольга Вадимовна

2024 год

Реализовать программно метод градиентного спуска. Реализация должна предполагать останов при достижении заданного количества шагов, останов по величине ɛ, регулировать величину шага, задание начальной точки.

Провести исследование влияния параметров метода на результат (количество итераций, величина параметров, точность, задание начальной точки). Для этой цели могут быть использованы приведенные задачи и предложенные вами.

Результаты отразить в отчете.

Код программы

import numpy as np

def numerical\_gradient(func, x, h=1e-5):

    """

    Функция для численного вычисления градиента методом конечных разностей.

    func: функция для которой нужно вычислить градиент

    x: точка, в которой вычисляется градиент

    h: малое значение для численного дифференцирования

    """

    grad = np.zeros\_like(x)  # Инициализация градиента

    for i in range(len(x)):

        x\_h1 = x.copy()

        x\_h2 = x.copy()

        x\_h1[i] += h  # Увеличиваем значение по i-й переменной

        x\_h2[i] -= h  # Уменьшаем значение по i-й переменной

        grad[i] = (func(x\_h1) - func(x\_h2)) / (2 \* h)  # Вычисляем частную производную

    return grad

def gradient\_descent(func, x0, eps1, eps2, m, t):

    """

    Реализация метода градиентного спуска.

    func: функция для минимизации

    x0: начальная точка (numpy массив)

    eps1: точность по градиенту

    eps2: точность по изменениям функции и точки

    m: максимальное количество итераций

    t: начальный шаг

    """

    x = np.array(x0, dtype=float)

    k = 0

    while k < m:

        grad = numerical\_gradient(func, x)  # Вызов функции для вычисления градиента

        grad\_norm = np.linalg.norm(grad)

        # Проверка критерия остановки по градиенту

        if grad\_norm < eps1:

            break

        # Обновляем точку

        x\_new = x - t \* grad

        # Проверка критериев остановки по изменениям

        if np.linalg.norm(x\_new - x) < eps2 and abs(func(x\_new) - func(x)) < eps2:

            x = x\_new

            break

        x = x\_new

        k += 1

    return x, k

# Функция для проверки

def f(x):

    return x[0]\*\*3 - x[0]\*x[1] + x[1]\*\*2 - 2\*x[0] + 3\*x[1] - 4 # Задание для проверки № 1

    # return (x[1]-x[0]\*\*2)\*\*2 + (1-x[0])\*\*2 # Задание для проверки № 2

    # return (x[1]\*\*2+x[0]\*\*2-1)\*\*2+(x[0]+x[1]-1)\*\*2 # Задание для исследования № 1

    # return (x[0]\*\*2+x[1]-11)\*\*2+(x[0]+x[1]\*\*2-7)\*\*2 # Задание для исследования № 2

    # return 4\*(x[0]-5)\*\*2 + (x[1]-6)\*\*2 # Задания для исследования № 3-5

x0 = [0, 0]

eps1 = 0.1

eps2 = 0.1

m = 10

t = 0.1

result, iterations = gradient\_descent(f, x0, eps1, eps2, m, t)

print("Результат:", result)

Описание программы

1. Определение вспомогательной функции для численного вычисления градиента (numerical\_gradient). Используется метод конечных разностей для вычисления градиента в точке. Градиент вычисляется по каждой переменной. Добавляется небольшое значение h для оценки производной вперед и назад. Вычисляется центральная разностная производная:
2. Определение основной функции градиентного спуска (gradient\_descent). Функция предназначена для нахождения минимума заданной функции func. Основные входные параметры:

* x0 — начальная точка.
* eps1 — критерий остановки по норме градиента.
* eps2 — критерий остановки по изменениям точки и функции.
* m — максимальное количество итераций.
* t — шаг.

1. Инициализация начальных параметров для градиентного спуска. Устанавливается начальная точка х равной х0. Счетчик итераций k устанавливается в 0.
2. Цикл градиентного спуска. Выполняется до достижения одного из условий остановки или превышения максимального числа итераций:

* Вычисление градиента с использованием numerical\_gradient.
* Проверка критерия остановки по градиенту: если норма градиента ∥grad∥ меньше eps1, алгоритм завершает работу.
* Обновление точки: Новая точка вычисляется как
* Проверка критерия остановки по изменениям: если изменения точки и значения функции меньше eps2, алгоритм завершает работу.
* Обновление текущей точки x и увеличение счетчика итераций k.

1. Возврат результата. После выхода из цикла возвращается найденная точка x и количество итераций k.
2. Задание целевой функции (f). Целевая функция определяет, какую задачу минимизации решает алгоритм.
3. Запуск градиентного спуска. Определяются начальные параметры. Вызывается функция gradient\_descent для вычисления результата.
4. Вывод результата. На экран выводится найденная точка (минимум функции).

Проверка работы программы

Для начала проверим работу программы на 2 функциях, решения для которых мы знаем.

Результат вычислений: [0.50028344; -1.24954101] ≈ [0.5; -1.25] = (), что совпадает с решением по условию



Результат вычислений: [0.99082563; 0.97790963] ≈ [1; 1], что совпадает с решением по условию



По результатам проверки мы можем сделать вывод, что программа работает корректно и правильно проводит расчеты.

Исследование влияния параметров на результат

Исследуем влияние следующих параметров на результаты:

* Начальная точка (x0)
* Точность (ɛ1, ɛ2)
* Максимальное количество шагов (M)
* Величина шага (t)

В качестве значений по умолчанию будем брать (если в задании не указано иное):

* x0 = (0; 0)
* ɛ1, ɛ2 = 10-4
* M = 100
* t = 0,01

Для проведения исследования будем поочередно менять по одному из этих параметров, чтобы отследить изменение результатов

Первая начальная точка по условию . Остальные параметры не заданы, т. е. берем значения по умолчанию.

Результат при параметрах по умолчанию: [-0.02362282; 1.00526322] ≈ (-0,02; 1)



Попробуем уменьшить значение точности до ɛ1, ɛ2 = 10-6



Результат остался без изменений.

Попробуем, наоборот, выбрать более «грубое» значение точности ɛ1, ɛ2 = 10-2



Результат стал менее точным. Следовательно, точность вычислений уменьшается с увеличением ее значения.

Вернем значение точности по умолчанию (ɛ1, ɛ2 = 10-4), но уменьшим количество шагов до М = 10:



Следовательно, из-за уменьшения количества шагов уменьшается точность результата – вычисления не проведены до конца

Наоборот, увеличим количество шагов до М = 200:



Результат остался таким же, как и при М = 100. Следовательно, 100 шагов вполне достаточно для нахождения минимума.

Вернем значение М = 100 и попробуем изменить величину шага. Для начала увеличим ее до t = 0,1:



Результат сильно отклоняется от реальных значений.

Теперь попробуем уменьшить шаг до t = 0.001:



Результат также отклоняется, но в данном случае уже не так сильно.

Это объясняется тем, что при малом значении шага вычисления будут проводится медленнее, и максимального количества шагов (М) может быть недостаточно – т. е. вычисления проведены не до конца.

При большом значении шага алгоритм может «перепрыгнуть» через минимум. Из-за этого результат будет некорректным.

Поэтому значения шага должно быть оптимальным – не слишком большим, чтобы не «перепрыгнуть» минимум, и не слишком маленьким, чтобы обеспечить нормальную скорость вычислений.

Проверим результаты вычисления для второй начальной точки по условию . Заметим, что здесь аргументы x1 и x2 поменяны местами.

Посчитаем со значениями по умолчанию:



Здесь аргументы также поменялись местами.

Такие результаты можно объяснить структурой функции и правилом «от перестановки мест слагаемых сумма не меняется»:



Проводить вычисления для разных значений параметров здесь не имеет смысла, т. к. они будут аналогичны результатам при за тем лишь исключением, что аргументы в результате поменяются местами.

По условию заданы:

* x0 = (0; 0)
* ɛ1, ɛ2 = 0,1

Для количества и длины шагов возьмем значения по умолчанию.

В результате вычислений получаем [2.96898325; 2.06884055] ≈ (2.97; 2.07)



Уменьшим количество шагов до М = 10:



Значение стало менее точным

Увеличим количество шагов до М = 200:



Значение совпало с результатом при М = 100. Следовательно, 100 шагов вновь оказалось достаточно для нахождения точного результата.

Вернем значение М = 100 и увеличим длину шага до t = 0,1:



Алгоритм вновь «перепрыгнул» минимум и выдал некорректные значения

Уменьшим длину шага до t = 0,001:



Результат немного отклоняется от результата при t = 0,01

Результаты исследования функции №2 подтверждают выводы, сделанные при исследовании функции №1.

По условию заданы:

* x0 = (8; 9)
* ɛ1, ɛ2 = 0,1

M и t вновь берем по умолчанию.

В результате вычислений получаем [5.1371737; 7.4206466] ≈ (5.14; 7.42)



Уменьшим количество шагов до М = 10:



Видим отклонение от результата при параметрах по умолчанию – вычисления не завершены.

При М = 200 результаты совпадают с М = 100. 100 шагов вновь оказалось достаточно для точного результата:



Вернем М = 100 и увеличим t до 0,1:



В данном случае не произошло сильного отклонения – минимум не «перепрыгнут». Однако, результат все равно оказался менее точным.

При уменьшении до t = 0,001:



Также наблюдаются небольшие отклонения.

Немного подкорректируем вывод по влиянию длины шага с учетом полученных при исследовании функции № 3 результатов:

Чем короче шаг, тем выше точность вычислений и ниже скорость. При увеличении длины шага значение становится менее точным (а может быть и вообще «перепрыгнуто»), но при этом вычисления происходят быстрее. Поэтому важно подобрать оптимальное значение шага – чтобы и значение было точным, и скорость приемлемой.

1. Используем функцию из п. 3, но изменим условия по умолчанию:

* x0 = (8; 9)
* ɛ1, = 0,1
* t = 0,1

Здесь за ɛ2 по умолчанию примем 0,1 (уже посчитано выше) и будем отталкиваться от этого результата:



Уменьшим ɛ2 до 0,01:



Увеличим ɛ2 до 1:



При уменьшении ɛ2 значения оказались более точными.

Вернем ɛ2 = 0,1. Уменьшим М до 10:



Результаты сошлись с результатами при М = 100. Получается, что в данном случае вычисление произвелось менее чем за 10 шагов.

Еще раз уменьшим значение М до 5:



Здесь уже произошли отклонения. 5 шагов оказалось недостаточно.

Исходя из результатов исследования в пункте 4, мы можем сделать вывод, что при увеличении значения параметра точности (т. е. при более меньшей точности) увеличивается скорость вычисления, но уменьшается точность результата. Здесь также важно подобрать оптимальные значения – чтобы и точность, и скорость были высокими.

1. Возьмем функцию из п. 3

Значения по умолчанию:

* x0 = (8; 9)
* ɛ1, ɛ2 = 0,1
* t = 0,1
* M = 10

Здесь все значения заданы по условию. Исследовать изменения параметров здесь не будем, просто подсчитаем минимум: [5.00000031 6.32212255] ≈ (5; 6.32)



Вывод

В ходе выполнения лабораторной работы была написана программа, вычисляющая точки минимума методом конечных разностей.

Корректность работы программы была проверена на 2 функциях, точки минимума которых нам были известны.

Были проведены исследования влияния изменения различных параметров на основе 3 функций. Были сделаны следующие выводы:

* Чем выше количество шагов, за которые выполняется вычисление, тем ниже скорость этого вычисления
* Чем ниже значение параметра точности ɛ, тем выше точность результатов
* Чем ниже значение параметра точности ɛ, тем выше количество шагов, за которые находится результат (т. е., ниже скорость)
* Важно подобрать оптимальные значения точности – не слишком низкие, чтобы программа работала быстро, но и не слишком высокие, чтобы результаты были точными
* Максимальное количество шагов (М) напрямую на вычисления не влияет. Оно лишь останавливает программу при достижении этого количества шагов. При этом, если программа достигнет этого числа, вычисления будут не завершены, а результаты будут неточными. Однако, это поможет сократить время работы за счет жертвования точностью. В наших случаях все вычисления производились менее чем за 100 шагов.
* Чем короче длина шага t, тем точнее вычисления
* Чем короче длина шага t, тем выше количество шагов, за которые находится результат (т. е., ниже скорость)
* При слишком длинном шаге есть риск «перепрыгнуть» точку минимума. Тогда результаты будут не просто неточными, а вообще некорректными.
* Важно подобрать оптимальное значение длины шага – не слишком низкое, чтобы программа работала быстро, но и не слишком высокое, чтобы результаты были точными и корректными
* Если начальная точка находится близко к минимуму, алгоритм сойдётся быстрее, так как градиент будет небольшим и корректировки по направлению спуска будут минимальными.
* Если начальная точка далеко от минимума, алгоритм может потребовать больше итераций для достижения нужной точности.