**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра дослідження операцій

**Кваліфікаційна робота**

**на здобуття ступеня бакалавра**

за спеціальністю 113 Прикладна математика

на тему:

**ЗАСТОСУВАННЯ МЕТОДІВ ГРАДІЄНТНОГО СПУСКУ У АНАЛІЗІ ДАНИХ**

Виконав студент 4-го курсу

Майстренко Олександр Сергійович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Науковий керівник:

доцент, кандидат фіз.-мат. наук

Якимів Роман Ярославович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Роботу розглянуто й допущено до захисту на засіданні кафедри дослідження операцій

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 р.,

протокол № \_\_\_\_\_

Завідувач кафедри

О. М. Іксанов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2021

**ЗМІСТ**

|  |  |  |
| --- | --- | --- |
|  | ВСТУП | 3 |
|  | РОЗДІЛ 1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ | 4 |
|  | 1.1 Вступ | 4 |
|  | 1.2 Загальний алгоритм градієнтного методу | 4 |
|  | 1.3 Метод найшвидшого спуску | 8 |
|  | 1.4 Метод областей довіри | 10 |
|  | РОЗДІЛ 2 МЕТОДИ ГРАДІЄНТНОГО СПУСКУ У РІЗНИХ МОДЕЛЯХ АНАЛІЗУ ДАНИХ | 12 |
|  | 2.1 Градієнтний спуск у лінійній регресії з однією змінною | 12 |
|  | 2.2 Градієнтний спуск у лінійній регресії з декількома змінними | 13 |
|  | 2.3 Градієнтний спуск у поліноміальній рідж-регресії | 14 |
|  | ВИСНОВОК | 17 |
|  | ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ | 18 |
|  | Додаток А (*лістинг програми лінійної регресії з однією змінною)* | 19 |
|  | Додаток Б (*лістинг програми лінійної регресії з декількома змінними)* | 31 |
|  | Додаток В (*лістинг програми поліноміальної рідж-регресії*) | 42 |

**ВСТУП**

Градієнтні методи належать до наближених числових методів розв'язування задач нелінійного програмування, оскільки дають точний розв'язок за нескінченне і лише в окремих випадках за скінченне число кроків. З їх використанням можна розв'язувати будь-яку задачу нелінійного програмування, знаходячи, як правило, лише локальний екстремум. Тому застосування цих методів дає найбільший ефект для розв'язування задач випуклого програмування, де локальний екстремум є одночасно і глобальним.

Також градієнтні методи мають широке застосування в аналазі даних, зокрема в регресійному аналізі. При побудові регресійної моделі використовується так звана функція витрат, яка може бути ефективно мінімізована саме методом градієнтного спуску. Для дослідження використання градієнтних методів в аналазі даних я обрав сучасні датасети з даними, пов’язаними з поширенням COVID-19 в Україні, тому робота є актуальною: отримані результати можуть бути використані для подальшого аналізу.

**РОЗДІЛ 1**

**КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

* 1. **Вступ**

Градієнтні методи належать до наближених числових методів розв’язування задач нелінійного програмування, оскільки дають точний розв’язок за нескінченне і лише в окремих випадках за скінченне число кроків. З їх використанням можна розв’язувати будь-яку задачу нелінійного програмування, знаходячи, як правило, лише локальний екстремум. Тому застосування цих методів дає найбільший ефект для розв’язування задач випуклого програмування, де локальний екстремум є одночасно і глобальним.

* 1. **Загальний алгоритм градієнтного методу**

Всі алгоритми оптимізації на основі градієнтних методів можна описати наступним чином. Ми починаємо з числа ітерацій та початкової точки

1. *Тест на збіжність.* Якщо умови збіжності виконуються у початковій точці, тоді ми можемо зупинитися і - це рішення.
2. *Обчислення напрямку пошуку.* Обчислюємо вектор , який визначає напрямок у n-мірному просторі уздовж якого ми будемо шукати.
3. *Обчислення довжини кроку.* Знаходимо додатний скаляр такий, що
4. *Оновлення змінних.* Присвоюємо і повертаємося до кроку 1.

У цьому типі алгоритмів є дві підзадачі для кожної великої ітерації: обчислення напрямку пошуку та знаходження розміру кроку (керованого ). Різниця між різними типами градієнтних алгоритмів полягає у методі, який використовується для обчислення напрямку пошуку.

Нехай є функція , де – -мірний вектор

Градієнт цієї функції задається частковими похідними кожної з незалежних змінних відповідно,

Вищі похідні функцій з декількома змінними визначаються, як у випадку з однією змінною, але кількість компонентів градієнта збільшується в n разів для кожного диференціювання.

Хоча градієнт функції від змінних є -мірним вектором, "друга похідна" -змінної функції визначається частковими похідними (похідними перших часткових похідних відносно змінних):

Якщо часткові похідні , та неперервні і – однозначна, тоді існує та . Таким чином часткові похідні другого порядку можуть бути записані у вигляді матриці Гесе:

яка містить незалежних елементів.

Якщо квадратична, матриця Гесе буде константою і функція може бути записана так:

Як і в випадку з однією змінною, умови оптимальності можна отримати з розкладу рядів Тейлора по :

де , – скаляр, – -мірний вектор.

Для того щоб був локальним мінімумом, для будь-якого вектора має існувати скінчене таке, що , тобто існує окіл, в якому ця нерівність справджується. Якщо ця умова задовольняється, тоді і перший та другий доданки мають бути більше або рівні 0.

Як і в випадку однієї змінної, і з тієї ж причини, ми починаємо з розгляду члену першого порядку. Оскільки є довільним вектором, а може бути додатним або від’ємним, кожна складова вектора градієнта повинна дорівнювати нулю.

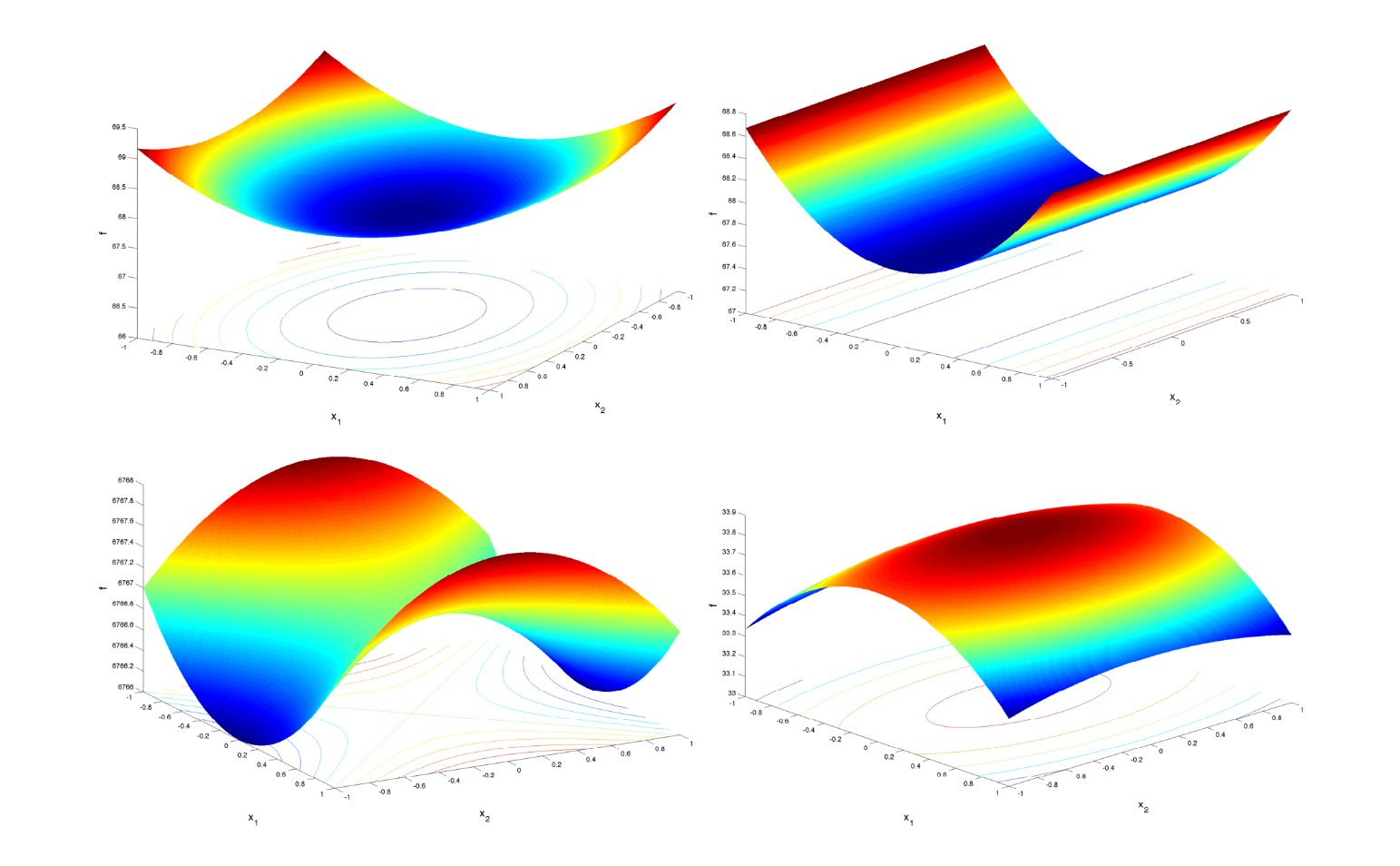
Тепер ми повинні розглянути член другого порядку, . Для того, щоб цей термін був невід’ємним, повинна бути додатно напіввизначеною, а за неперервністю, матриця Гесе також повинен бути додатно напіввизначеним.

*Необхідна умова*(для локального мінімуму):

– додатно напіввизначена

*Достатня умова:*

– додатно визначена



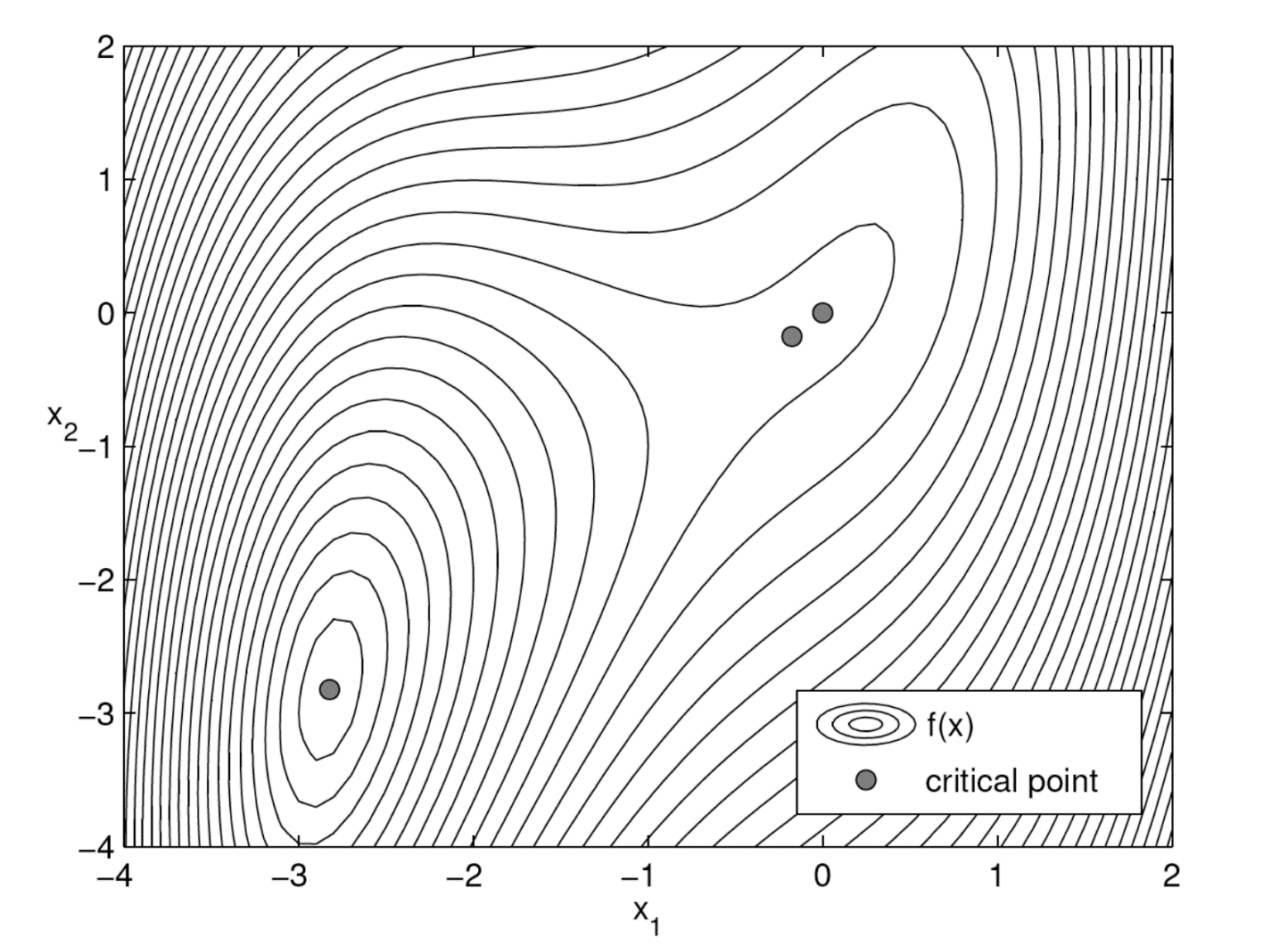
Мал. 1. Приклади критичних точок

Приклад 1.Критичні точки функції

Нехай є функція . Знайти усі стаціонарні точки та визначити їх тип

Розв’язок: розв’яжемо , отримаємо такі три розв’язки:

Щоб встановити тип точки, ми повинні визначити, чи є матриця Гесе додатно визначениою і порівняти значення функції в точках.



Мал. 2. Критичні точки функції

* 1. **Метод найшвидшого спуску**

Метод найшвидшого спуску використовує вектор градієнта в кожній точці як напрямок пошуку для кожної ітерації. Як згадувалося раніше, вектор градієнта ортогональний площині, дотичній до ізоповерхні функції.

Вектор градієнта в точці, , є також напрямком максимальної швидкості зміни (максимального збільшення) функції в цій точці. Ця швидкість змін задана нормою, .

Алгоритм найшвидшого спуску:

1. Обираємо початкову точку та параметри збіжності .
2. Обчислюємо . Якщо , тоді зупиняємося. Інакше, обчислюємо нормалізований напрям пошуку
3. Виконуємо лінійний пошук щоб знайти довжину кроку у напрямку .
4. Перевизначаємо поточну точку, .
5. Обчислюємо . Якщо умова задовольняється за дві успішні ітерації, тоді зупиняємося.

Інакше, та повертаємося до кроку 2.

Отже, нерівність є перевіркою успішного зменшення . – це абсолютне допустиме відхилення зміни значення функції(зазвичай маленьке ) та – відносне допустиме відхилення(зазвичай рівне 0.01).

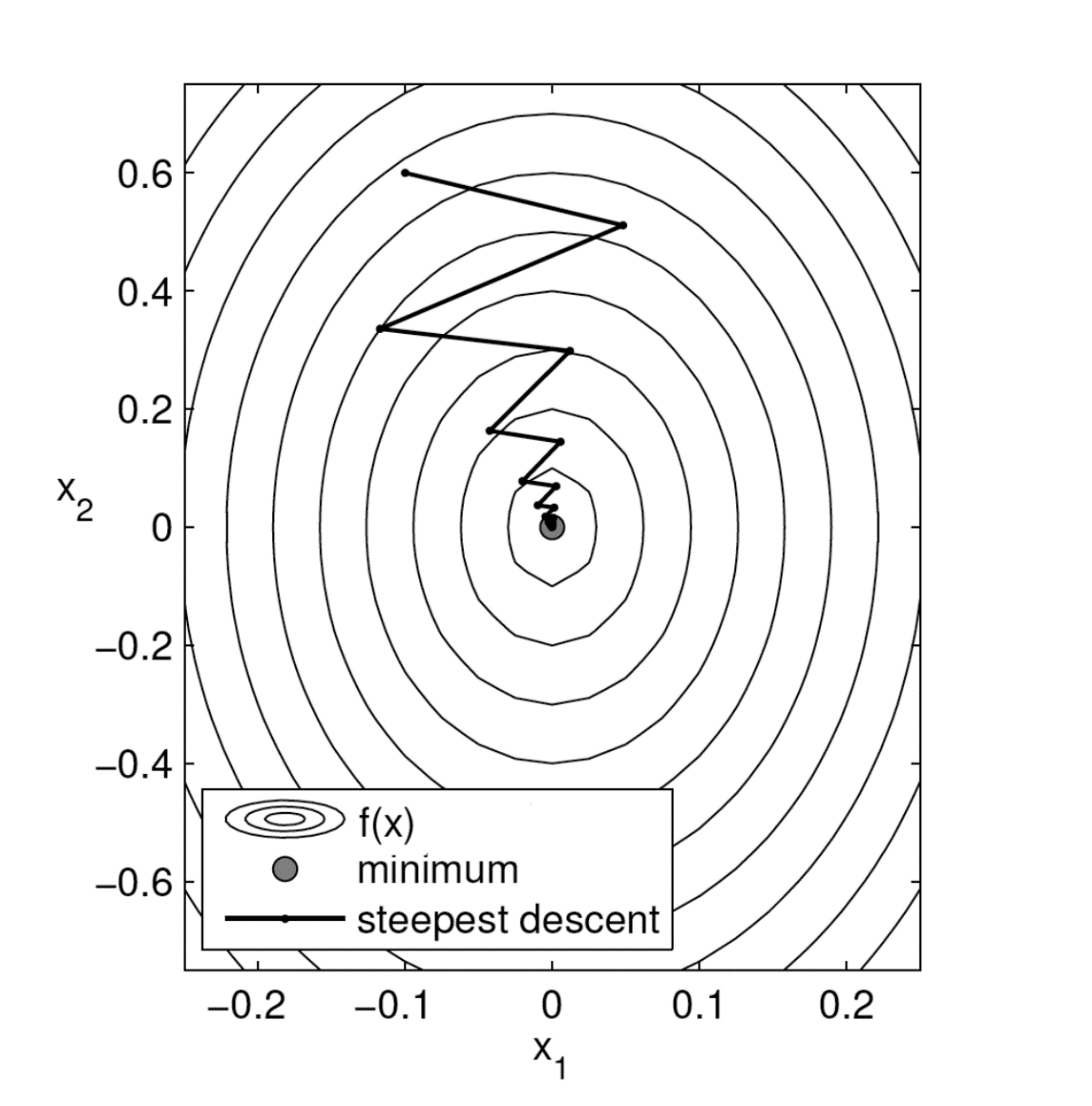
Якщо використовувати лінійний пошук, напрямок найшвидшого спуску на кожній новій ітерації буде ортогональним до попереднього:

Отже, метод рухається “зигзагами” у просторі, що не є дуже ефективним. Хоча протягом перших кількох ітерацій може спостерігатися суттєве зменшення, після цього метод, як правило, дуже повільний. Зокрема, хоча алгоритм гарантовано збігається, він може зайняти нескінченну кількість ітерацій. Швидкість збіжності лінійна.

Для найшвидшого спуску та інших градієнтних методів, які не дають чітко масштабованих напрямків пошуку, нам потрібно використовувати іншу інформацію, щоб вгадати довжину кроку.

Одна з стратегій полягає в припущенні, що зміна першого порядку в буде такою ж, як та, яка була отримана на попередньому кроці:

Приклад 2.Використання методу найшвидшого спуску для

Мал. 3. Шлях розв’язку методом найшвидшого спуску 

* 1. **Метод областей довіри**

Область довіри або методи “обмеженого кроку” - це інший підхід до розв’язку, що виникає внаслідок матриці Гесе, яка не є додатно визначеною або вкрай нелінійною функцією.

Один із способів інтерпретувати ці проблеми - сказати, що вони виникають через те, що ми виходимо за межі області, для якої квадратичне наближення має місце. Таким чином, ми можемо подолати ці труднощі, мінімізуючи квадратичну функцію в межах області навколо , в межах якої ми довіряємо квадратній моделі.

Алгоритм областей довіри:

1. Обираємо початкову точку та параметр збіжності та початковий розмір області довіри .
2. Обчислюємо . Якщо , тоді зупиняємося. Інакше, продовжуємо.
3. Обчислюємо та мінімізуємо:
4. Обчислюємо та відношення, яке вимірює точність квадратичної моделі
5. Обчислюємо розмір нової області довіри таким чином:
6. Визначаємо нову точку:
7. Прирівнюємо та повертаємося до кроку 2.

Початкове значення зазвичай 1.

**РОЗДІЛ 2**

**МЕТОДИ ГРАДІЄНТНОГО СПУСКУ**

**У РІЗНИХ МОДЕЛЯХ АНАЛІЗУ ДАНИХ**

* 1. **Градієнтний спуск у лінійній регресії з однією змінною**

Для демонстрації використання методу градієнтного спуску у лінійній регресії з однією змінною я використав датасет із щоденною статистикою COVID-19 в Україні з квітня 2020р. (лістинг програми див. у додатку А)

Перш за все я центрував та стандартизував дані, віднявши від кожного значення незалежної змінної її середнє значення та потім поділивши на стандартне відхилення.

Побудуємо таку модель для кожного значення :

Також маємо таку функцію витрат:

Основні кроки побудови навчального алгоритму будуть такі:

1. Визначити структуру моделі(наприклад кількість незалежних змінних)
2. Ініціалізувати параметри моделі
3. Виконати основний цикл
   1. Обчислити значення функції вартості
   2. Обчислити градієнт
   3. Оновити параметри моделі

Задамо таке , що , та знайдемо для нього значення функції витрат

Знайдемо градієнт функції витрат:

Мета – знайти значення та , при яких значення функції витрат найменше. Для параметра правило спуску буде виглядати так: , де – швидкість навчання.

* 1. **Градієнтний спуск у лінійній регресії з декількома змінними**

Для демонстрації використання методу градієнтного спуску у лінійній регресії з однією змінною я віикористав датасет із щоденною статистикою COVID-19 в Україні з квітня 2020р. (лістинг програми див. у додатку Б)

Спочатку центруємо та стандартизуємо дані, віднявши від кожного значення незалежної змінної її середнє значення та потім поділивши на стандартне відхилення.

Побудуємо таку модель для кожного значення :

Також маємо таку функцію витрат:

Основні кроки побудови навчального алгоритму будуть такі:

1. Визначити структуру моделі(наприклад кількість незалежних змінних)
2. Ініціалізувати параметри моделі
3. Виконати основний цикл
   1. Обчислити значення функції витрат
   2. Обчислити градієнт
   3. Оновити параметри моделі

Задамо таке , що , та знайдемо для нього значення функції витрат

Знайдемо градієнт функції витрат:

Мета – знайти значення та , при яких значення функції витрат буде найменше. Для параметра правило спуску буде виглядати так: , де – розмір кроку.

* 1. **Градієнтний спуск у поліноміальній рідж-регресії**

Для демонстрації використання методу градієнтного спуску у поліноміальній рідж-регресії я використав датасет із щоденною статистикою госпіталізацій пацієнтів

COVID-19 в Україні з 5 липня 2020р по 5 листопада 2020р. (лістинг програми див. у додатку В)

Перш за все, будемо розглядати кількість госпіталізацій як функцію від дня року. Тому календарні дати за п’ять місяців треба нормалізувати, тобто звести до значень на відрізку від 0 до 1.

Основна ідея полліноміальної регресії – комбінації незалежних змінних із заданою степінню. Нехай є степінь та . Тоді вектор незалежних змінних буде виглядати так:

Додамо одиницю для того щоб разом рахувати й випадкову похибку. Отже, буде виглядати так:

Тоді маємо таку прогнозуючу функцію:

Застосовуючи поліном надто високого степіня можно дуже просто перенавчити модель. Основна техніка проти перенавчання називається регуляризацією. Функція витрат буде виглядати так:

де – регуляризація, а – Евклідова норма.

Означимо градієнт:

Також у програмі було додано функцію polynomial\_features для утворення комбінацій. Наприклад, для degree = 3 та незалежних змінних отримаємо:

Середньоквадратичну помилку було обчислено за формулою:

## **ВИСНОВОК**

Питання аналізу даних та роботи з ними є надзвичайно актуальним, особливо в сьогоденні. Зокрема під час таких глобальних явищ, таких як пандемія COVID-19, дуже важливо збирати інформацію, дані та опрацьовувати їх для правдивої оцінки поточної ситуації та подальшого прийняття рішень.

У роботі було висвітлено як використувуються методи градієнтного спуску при аналізі даних, наведено приклади розв’язку та аналізу задач нелінійного програмування градієнтним методом та розроблено програми різноманітних регресій, що використовують метод градієнтного спуску(див. у додатки А, Б, В).

## **ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ**

1. J. Nocedal and S. J. Wright. Numerical Optimization. Springer, 2nd edition, 2006.

2. Грас Дж. Data Science. Наука о данных с нуля: Пер. С англ. – СПб.: БХВ-Петербург, 2017.-336.: ил.

3. О`Нил Кэти, Шатт Рэйчел Data Science. Инсайдерская информация для новичков. Включая язык R. – СПб.: Питер, 2019.-368 с.: ил.

4. Барінський А.Ф. і ін. Математичне програмування та дослідження операцій. – Л.: Інтелект- Захід, 2008. – 588с.: іл.

5. Барінський А.Ф. і ін. Математичне програмування. – Л.: Інтелект-Захід, 2004. – 448с.: іл.

6. Зайченко Ю.П. Дослідження операцій. – К.: ДМК Пресс, 2006. – 576с.: іл.

7. Долженков В.А., Колесников Ю.В. Microsoft® Excel 2000. – СПб.: БХВ-Петербург, 2001. – 1088с.

8. Кудрявцев Е.М. Mathcad 2000 Pro. – М.: ДМК Пресс, 2001. – 576с.

9. Таха Хэмди А. Введение в исследование операций, 6-е издание: Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 912с.: ил.

10. Наконечний С. І., Савіна С. С. Математичне програмування: Навч. посіб. — К.: КНЕУ, 2003. — 452 с.

11. https://www.kaggle.com/vbmokin/covid19-in-ukraine-daily-data?select=COVID-19-in-Ukraine-from-April.csv

12. https://www.kaggle.com/vbmokin/covid19-in-ukraine-daily-data?select=hospitalizations\_number\_06\_12.csv

## **Додаток А (*лістинг програми лінійної регресії з однією змінною*):**

# Майстренко Олександр ДО-4, 2021

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Завантаження даних

def load\_data():

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('COVID-19-in-Ukraine-from-April.csv', usecols=['n\_confirmed', 'n\_deaths'])

x = data['n\_confirmed']

y = data['n\_deaths']

train\_set\_x, test\_set\_x, train\_set\_y, test\_set\_y = train\_test\_split(x, y, test\_size=0.33, random\_state=42)

return train\_set\_x, test\_set\_x, train\_set\_y, test\_set\_y

train\_set\_x, test\_set\_x, train\_set\_y, test\_set\_y = load\_data()

m\_train = train\_set\_x.shape

m\_test = test\_set\_x.shape

print("Кількість тренувальних прикладів: m\_train = " + str(m\_train))

print("Кількість тестових прикладів: m\_test = " + str(m\_test))

# Тренувальний сет даних

xmax = max(max(train\_set\_x), max(test\_set\_x))

plt.scatter(train\_set\_x, train\_set\_y)

plt.xlim([-0.05\*xmax, xmax\*1.05])

plt.title("Тренувальний сет")

plt.xlabel("Нові зафіксовані випадки")

plt.ylabel("Нові смерті")

plt.show()

# Тестовий сет даних

plt.scatter(test\_set\_x, test\_set\_y)

plt.xlim([-0.05\*xmax, xmax\*1.05])

plt.title("Тестовий сет")

plt.xlabel("Нові зафіксовані випадки")

plt.ylabel("Нові смерті")

plt.show()

mean = np.concatenate([train\_set\_x, test\_set\_x]).mean()

std = np.concatenate([train\_set\_x, test\_set\_x]).std()

train\_set\_x = (train\_set\_x - mean) / std

test\_set\_x = (test\_set\_x - mean) / std

# Тренувальний сет даних (після стандартизації)

plt.scatter(train\_set\_x, train\_set\_y)

plt.title("Тренувальний сет (після стандартизації)")

plt.xlabel("Нові зафіксовані випадки")

plt.ylabel("Нові смерті")

plt.show()

# Тестовий сет даних (після стандартизації)

plt.scatter(test\_set\_x, test\_set\_y)

plt.title("Тестовий сет (після стандартизації)")

plt.xlabel("Нові зафіксовані випадки")

plt.ylabel("Нові смерті")

plt.show()

def initialize\_with\_zeros():

"""

Ця функція ініціалізує theta та b як нулі.

Повертає:

theta -- ініціалізований скалярний параметр

b -- ініціалізований скаляр (відповідає випадковій похибці)

"""

theta = 0

b = 0

return theta, b

theta, b = initialize\_with\_zeros()

def propagate(theta, b, X, Y):

"""

Імплементує функцію витрат та її градієнт для подальшого градієнтного спуску

Аргументи:

theta -- параметр, скаляр

b -- випадкова похибка, скаляр

X -- вектор значень незалежної змінної розміру (кількість прикладів, )

Y -- значення залежної змінної (кількість прикладів, )

Повертає:

cost -- функція витрат для лінійної регресії

dt -- градієнт по theta, тієї самої розмірності, що й theta

db -- градієнт по b, тієї самої розмірності, що й b

"""

m = X.shape[0]

H = theta \* X + b # підставляємо поточні theta та b

cost = np.dot(H - Y, H - Y) / (2 \* m) # рахуємо значення функції витрат

dt = np.dot(X, H - Y) / m

db = np.sum(H - Y) / m

cost = np.squeeze(cost)

grads = {"dt": dt,

"db": db}

return grads, cost

def optimize(theta, b, X, Y, num\_iterations, learning\_rate, print\_cost=False):

"""

Ця функція оптимізує theta та b за допомогою алгоритму градієнтного спуску

Аргументи:

theta -- параметр, скаляр

b -- випадкова похибка, скаляр

X -- вектор значень незалежної змінної розміру (кількість прикладів, )

Y -- значення залежної змінної (кількість прикладів, )

num\_iterations -- кількість ітерацій для оптимізуючого циклу

learning\_rate – розмір кроку для оновлення градієнтного спуску

print\_cost -- встановлюється на True для того щоб надрукувати витрати кожні 100 ітерацій

Повертає:

params -- dictionary з ваговими коефіцієнтами theta та b

grads -- dictionary з градієнтами вагових коефіцієнтів та випадкової похибки із урахуванням функції витрат

costs -- list усіх значень функції витрат протягом оптимізації.

"""

costs = []

for i in range(num\_iterations):

# Обчислення функції витрат та градієнту

grads, cost = propagate(theta, b, X, Y)

# Отримуємо похідні з градієнтів

dt = grads["dt"]

db = grads["db"]

# правило спуску

theta -= learning\_rate \* dt

b -= learning\_rate \* db

# Записуємо витрати

if i % 100 == 0:

costs.append(cost)

# Друкуємо витрати кожні 100 ітерацій

if print\_cost and i % 100 == 0:

print("Вартість після ітерації %i: %f" % (i, cost))

params = {"theta": theta,

"b": b}

grads = {"dt": dt,

"db": db}

return params, grads, costs

def predict(theta, b, X):

"""

Прогнозує використовуючи отримані параметри лінійної регресії (theta, b)

Аргументи:

theta -- параметр, скаляр

b -- випадкова похибка, скаляр

X -- вектор значень незалежної змінної розміру (кількість прикладів, )

Повертає:

Y\_prediction -- numpy array (вектор), що містить усі прогнози для прикладів в X

"""

# Обчислюємо вектор "Y\_prediction" прогнозуючи кількість нових смертей

Y\_prediction = theta \* X + b

return Y\_prediction

def model(X\_train, Y\_train, X\_test, Y\_test, num\_iterations=2000, learning\_rate=0.5, print\_cost=False):

"""

Будує модель лінійної регресії, використовуючи функції, що були імплементовані раніше

Аргументи:

X\_train -- тренувальний сет представлений numpy array розміром (m\_train, )

Y\_train -- тренувальні значення представлені numpy array (вектором) розміру (m\_train, )

X\_test -- тестовий сет представлений numpy array розміром (m\_test, )

Y\_test -- тестові значення представлені numpy array (вектором) розміру (m\_test, )

num\_iterations -- параметр, що позначає кількість ітерацій для оптимізації параметрів

learning\_rate -- параметр, що позначає розмір кроку для правила спуску у optimize()

print\_cost -- встановлюється на True для того щоб надрукувати витрати кожні 100 ітерацій

Повертає:

d -- dictionary, що містить інформацію про модель

"""

# ініціалізуємо параметри нулями

theta, b = initialize\_with\_zeros()

# Градієнтний спуск

parameters, grads, costs = optimize(theta, b, X\_train, Y\_train, num\_iterations, learning\_rate, print\_cost)

# Отримуємо значення theta та b з dictionary

theta = parameters["theta"]

b = parameters["b"]

# Прогнозуємо значення на тестовому та тренувальному сетах Y\_prediction\_test = predict(theta, b, X\_test)

Y\_prediction\_train = predict(theta, b, X\_train)

# Виводимо помилки тестового та тренувального сетів

print("Тренувальний RMSE: {} ".format(np.sqrt(np.mean((Y\_prediction\_train - Y\_train) \*\* 2))))

print("Тестовий RMSE: {} ".format(np.sqrt(np.mean((Y\_prediction\_test - Y\_test) \*\* 2))))

d = {"costs": costs,

"Y\_prediction\_test": Y\_prediction\_test,

"Y\_prediction\_train": Y\_prediction\_train,

"theta": theta,

"b": b,

"learning\_rate": learning\_rate,

"num\_iterations": num\_iterations}

return d

d = model(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, num\_iterations=300, learning\_rate=0.05, print\_cost=True)

# Тренувальний сет даних

plt.title("Тренувальний сет")

plt.scatter(train\_set\_x, train\_set\_y)

x = np.array([min(train\_set\_x), max(train\_set\_x)])

theta = d["theta"]

b = d["b"]

y = theta \* x + b

plt.plot(x, y, color='orange')

plt.xlabel("Нові зафіксовані випадки")

plt.ylabel("Нові смерті")

plt.show()

# Тестовий сет даних

plt.title("Тестовий сет")

plt.scatter(test\_set\_x, test\_set\_y)

x = np.array([min(test\_set\_x), max(test\_set\_x)])

theta = d["theta"]

b = d["b"]

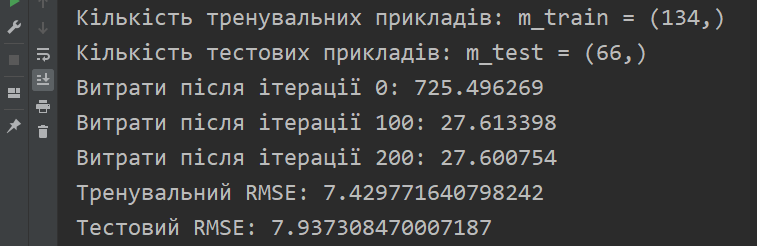
y = theta \* x + b

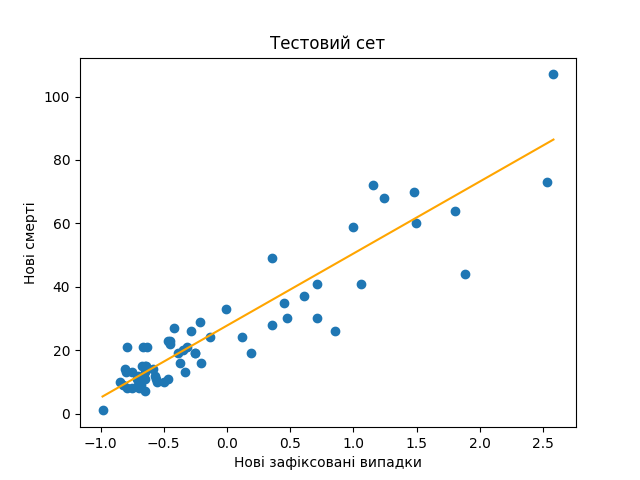
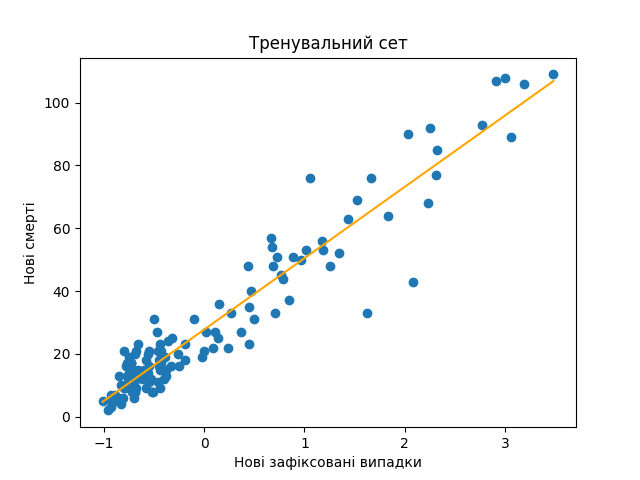
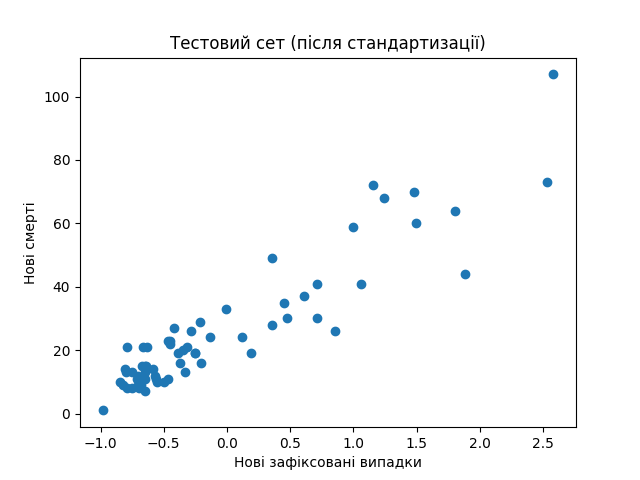
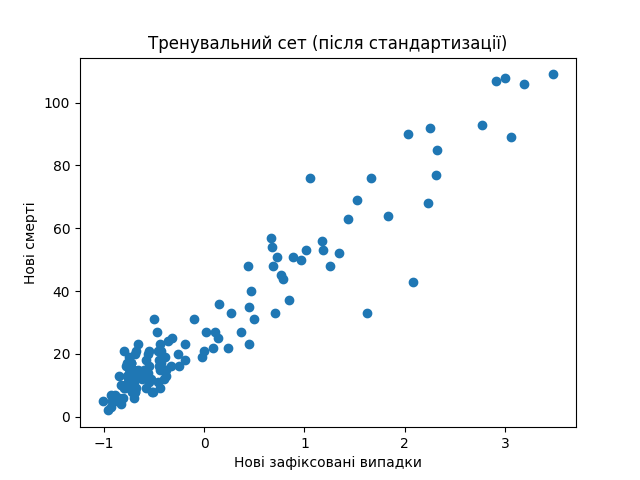
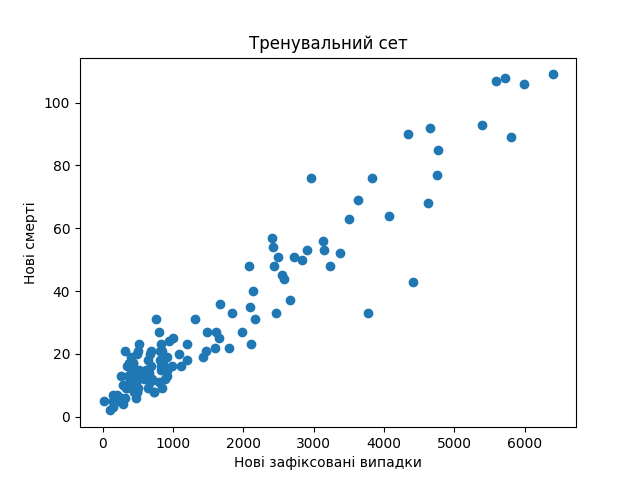
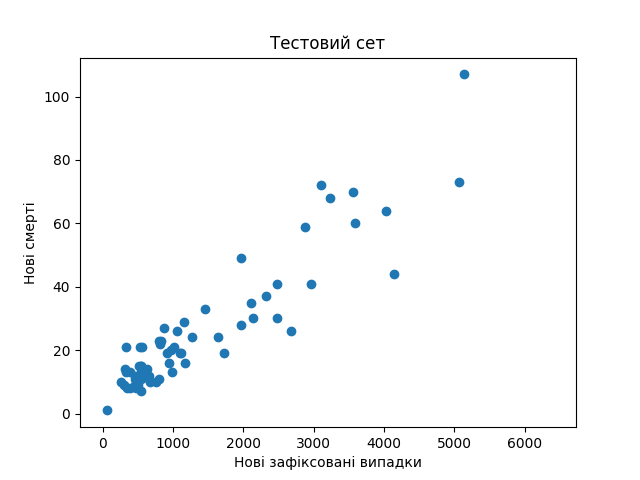
plt.plot(x, y, color='orange')

plt.xlabel("Нові зафіксовані випадки")

plt.ylabel("Нові смерті")

plt.show()





## **Додаток Б (*лістинг програми лінійної регресії з декількома змінними*):**

# Майстренко Олександр ДО-4, 2021

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Завантаження даних

def load\_data():

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('COVID-19-in-Ukraine-from-April.csv', usecols=['n\_confirmed', 'n\_deaths', 'n\_recovered'])

x = data[['n\_confirmed', 'n\_recovered']].to\_numpy()

y = data['n\_deaths'].to\_numpy()

train\_set\_x, test\_set\_x, train\_set\_y, test\_set\_y = train\_test\_split(x, y, test\_size=0.33, random\_state=42)

train\_set\_y = train\_set\_y.reshape((1, train\_set\_y.shape[0]))

test\_set\_y = test\_set\_y.reshape((1, test\_set\_y.shape[0]))

return train\_set\_x.T, train\_set\_y, test\_set\_x.T, test\_set\_y, data

train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, visualization\_set = load\_data()

m\_train = train\_set\_x.shape[1]

m\_test = test\_set\_x.shape[1]

print("Кількість тренувальних прикладів: m\_train = " + str(m\_train))

print("Кількість тестових прикладів: m\_test = " + str(m\_test))

plt.hist(visualization\_set['n\_deaths'])

plt.xlabel("Нові смерті")

plt.ylabel("Кількість")

plt.tight\_layout()

plt.show()

plt.scatter(visualization\_set['n\_confirmed'], visualization\_set['n\_deaths'])

plt.xlabel("Нові зафіксовані випадки")

plt.ylabel("Нові смерті")

plt.show()

plt.scatter(visualization\_set['n\_recovered'], visualization\_set['n\_deaths'])

plt.xlabel("Нові одужання")

plt.ylabel("Нові смерті")

plt.show()

all\_set\_x = np.concatenate([train\_set\_x, test\_set\_x], axis=1)

mean = all\_set\_x.mean(axis=1, keepdims=True)

std = all\_set\_x.std(axis=1, keepdims=True)

train\_set\_x = (train\_set\_x - mean) / std

test\_set\_x = (test\_set\_x - mean) / std

def initialize\_with\_zeros(dim):

"""

Ця функція створює вектор нулів розмірністю (dim, 1) для w ті ініціалізує b як 0.

Аргумент:

dim -- розмірність вектора w (або кількість незалежних змінних у цьому випадку)

Повертає:

w -- ініціалізований вектор розмірністю (dim, 1)

b -- ініціалізований скаляр (відповідає випадковій похибці)

"""

w = np.zeros([dim, 1])

b = 0

return w, b

def propagate(w, b, X, Y):

"""

Імплементує функцію витрат та її градієнт для подальшого градієнтного спуску

Arguments:

w -- вагові коефіцієнти, numpy array розміру (кількість полів, 1)

b -- випадкова похибка, скаляр

X -- матриця значень незалежних змінних розміру (кількість полів, кількість прикладів)

Y -- значення залежної змінної (1, кількість прикладів)

Повертає:

cost -- функція витрат для лінійної регресії

dw -- градієнт по w, тієї самої розмірності, що й w

db -- градієнт по b, тієї самої розмірності, що й b

"""

m = X.shape[1]

H = np.dot(w.T, X) + b # підставляємо поточні w та b

cost = np.dot((H - Y)[0], (H - Y)[0]) / (2 \* m) # рахуємо значення функції витрат

dw = np.dot(X, (H - Y).T) / m

db = np.sum(H - Y) / m

cost = np.squeeze(cost)

grads = {"dw": dw,

"db": db}

return grads, cost

def optimize(w, b, X, Y, num\_iterations, learning\_rate, print\_cost=False):

"""

Ця функція оптимізує w та b за допомогою алгоритму градієнтного спуску

Arguments:

w -- вагові коефіцієнти, numpy array розміру (кількість полів, 1)

b -- випадкова похибка, скаляр

X -- матриця значень незалежних змінних розміру (кількість полів, кількість прикладів)

Y -- значення залежної змінної (1, кількість прикладів)

num\_iterations -- кількість ітерацій для оптимізуючого циклу

learning\_rate -- розмір кроку для оновлення градієнтного спуску

print\_cost -- встановлюється на True для того щоб надрукувати витрати кожні 100 ітерацій

Повертає:

params -- dictionary з ваговими коефіцієнтами theta та b

grads -- dictionary з градієнтами вагових коефіцієнтів та випадкової похибки із урахуванням функції витрат

costs -- list усіх значень функції витрат протягом оптимізації, це знадобидться для того,

щоб побудувати криву навчання.

"""

costs = []

for i in range(num\_iterations):

# Обчислення функції витрат та градієнту

grads, cost = propagate(w, b, X, Y)

# Отримуємо похідні з градієнтів

dw = grads["dw"]

db = grads["db"]

# правило спуску

w -= learning\_rate \* dw

b -= learning\_rate \* db

# Записуємо витрати

if i % 100 == 0:

costs.append(cost)

# Друкуємо витрати кожні 100 ітерацій

if print\_cost and i % 100 == 0:

print("Витрати після ітерації %i: %f" % (i, cost))

params = {"w": w,

"b": b}

grads = {"dw": dw,

"db": db}

return params, grads, costs

def predict(w, b, X):

"""

Прогнозує використовуючи отримані параметри лінійної регресії (w, b)

Аргументи:

w -- вагові коефіцієнти, numpy array розміру (кількість прикладів, 1)

b -- випадкова похибка, скаляр

X -- дані розміру (кількість полів, кількість прикладів)

Повертає:

H -- numpy array (вектор), що містить усі прогнози для прикладів в X

"""

m = X.shape[1]

# Обчислюємо вектор "H"

H = np.dot(w.T, X) + b

return H

def model(X\_train, Y\_train, X\_test, Y\_test, num\_iterations=2000, learning\_rate=0.5, print\_cost=False):

"""

Будує модель лінійної регресії, використовуючи функції, що були імплементовані раніше

Аргументи:

X\_train -- тренувальний сет представлений numpy array розміром (кількість полів, m\_train)

Y\_train -- тренувальні значення представлені numpy array (вектором) розміру (1, m\_train)

X\_test -- тестовий сет представлений numpy array розміром (кількість полів, m\_test)

Y\_test -- тестові значення представлені numpy array (вектором) розміру (1, m\_test)

num\_iterations -- параметр, що позначає кількість ітерацій для оптимізації параметрів

learning\_rate -- параметр, що позначає розмір кроку для правила спуску у optimize()

print\_cost -- встановлюється на True для того щоб надрукувати витрати кожні 100 ітерацій

Повертає:

d -- dictionary, що містить інформацію про модель

"""

# ініціалізуємо параметри нулями

w, b = initialize\_with\_zeros(X\_train.shape[0])

# Градієнтний спуск

parameters, grads, costs = optimize(w, b, X\_train, Y\_train, num\_iterations, learning\_rate, print\_cost)

# Отримуємо значення w та b з dictionary

w = parameters["w"]

b = parameters["b"]

# Прогнозуємо значення на тестовому та тренувальному сетах

Y\_prediction\_test = predict(w, b, X\_test)

Y\_prediction\_train = predict(w, b, X\_train)

# Виводимо помилки тестового та тренувального сетів

print("Тренувальний RMSE: {} ".format(np.sqrt(np.mean((Y\_prediction\_train - Y\_train) \*\* 2))))

print("Тестовий RMSE: {} ".format(np.sqrt(np.mean((Y\_prediction\_test - Y\_test) \*\* 2))))

d = {"costs": costs,

"Y\_prediction\_test": Y\_prediction\_test,

"Y\_prediction\_train": Y\_prediction\_train,

"w": w,

"b": b,

"learning\_rate": learning\_rate,

"num\_iterations": num\_iterations}

return d

d = model(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, num\_iterations=3000, learning\_rate=0.05, print\_cost=True)

# Тренувальний сет даних

plt.title("Тренувальний сет")

plt.scatter(train\_set\_y, d["Y\_prediction\_train"])

plt.plot([0, 100], [0, 100], "--k")

plt.xlabel("Реальні нові смерті")

plt.ylabel("Спрогнозовані нові смерті")

plt.show()

# Тестовий сет даних

plt.title("Тестовий сет")

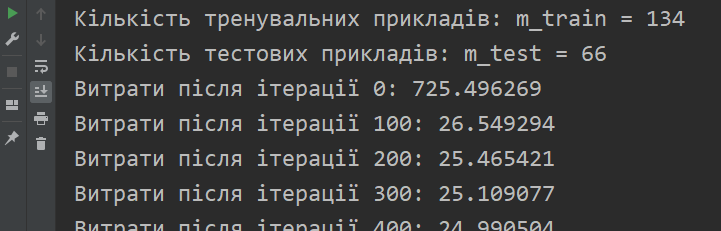
plt.scatter(test\_set\_y, d["Y\_prediction\_test"])

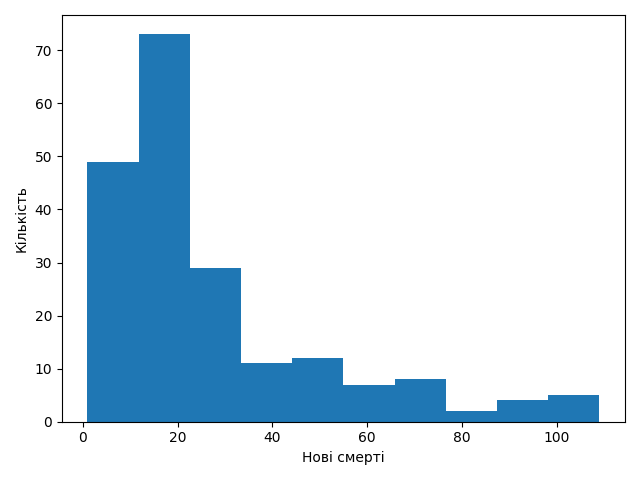
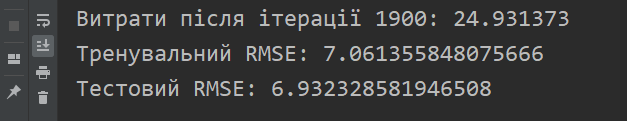
plt.plot([0, 100], [0, 100], "--k")

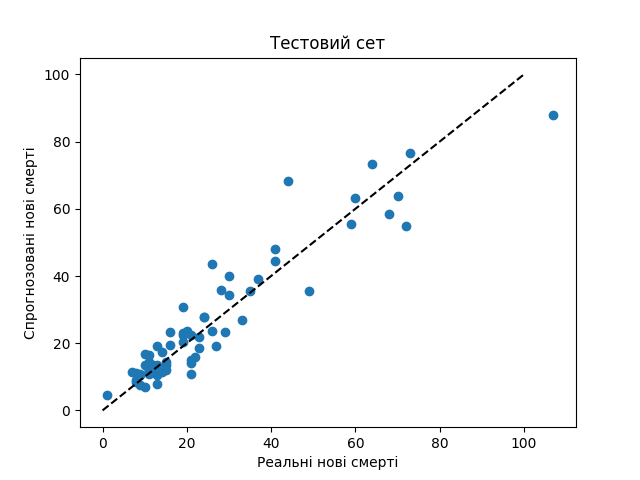
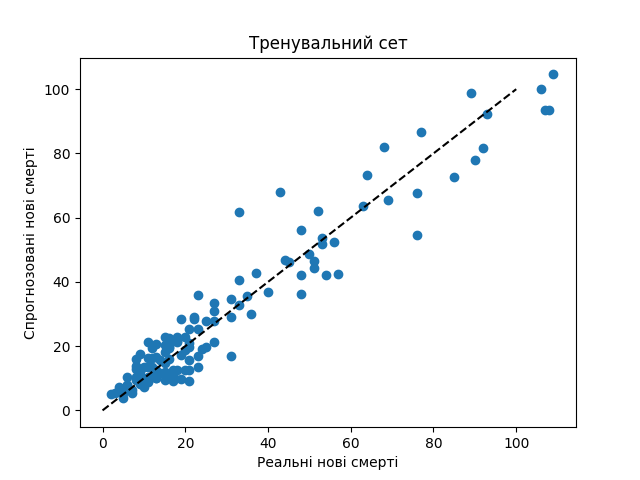
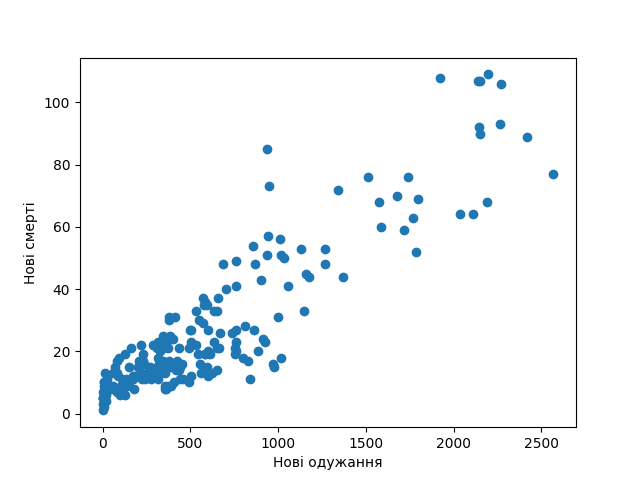
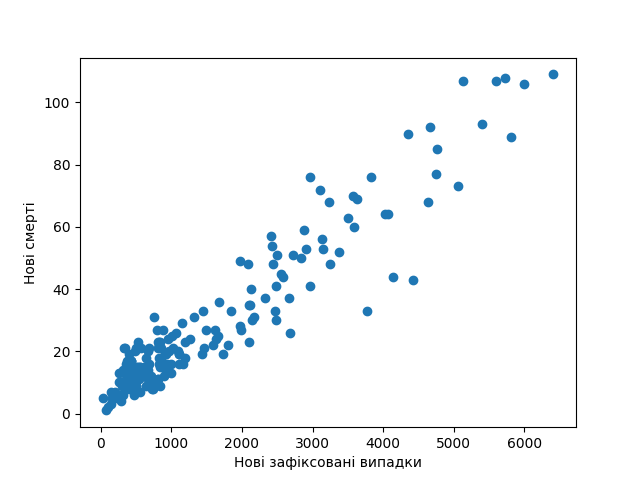
plt.xlabel("Реальні нові смерті")

plt.ylabel("Спрогнозовані нові смерті")

plt.show()







## **Додаток В (*лістинг програми поліноміальної рідж-регресії*):**

# Майстренко Олександр ДО-4, 2021

from datetime import datetime

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Завантаження даних

def load\_data():

from sklearn.model\_selection import train\_test\_split

df = pd.read\_csv('hospitalizations\_number\_06\_12.csv', sep=';')

data = np.zeros((int(df.size/2.), 2))

for index, row in df.iterrows():

d = datetime.strptime(row['date'], '%d.%m.%Y')

data[index, 0] = (d.month \* 30.44 + d.day - 218.08) / 121.76

data[index, 1] = row['hospitalizations']

x = data[:, 0]

x = x.reshape((x.shape[0], 1))

y = data[:, 1]

train\_set\_x, test\_set\_x, train\_set\_y, test\_set\_y = train\_test\_split(x, y, test\_size=0.33, random\_state=42)

train\_set\_y = train\_set\_y.reshape((1, train\_set\_y.shape[0]))

test\_set\_y = test\_set\_y.reshape((1, test\_set\_y.shape[0]))

return train\_set\_x.T, test\_set\_x.T, train\_set\_y, test\_set\_y, x.T

train\_set\_x, test\_set\_x, train\_set\_y, test\_set\_y, full\_feature\_set\_for\_plot = load\_data()

m\_train = len(train\_set\_x[0])

m\_test = len(test\_set\_x[0])

print("Кількість тренувальних прикладів: m\_train = " + str(m\_train))

print("Кількість тестових прикладів: m\_test = " + str(m\_test))

# Палітра кольорів

cmap = plt.get\_cmap('viridis')

# Візуіалізація даних

m1 = plt.scatter(121.76 \* train\_set\_x, train\_set\_y, color=cmap(0.9), s=10)

m2 = plt.scatter(121.76 \* test\_set\_x, test\_set\_y, color=cmap(0.5), s=10)

plt.xlabel('День')

plt.ylabel('Госпіталізації')

plt.legend((m1, m2), ("Тренувальні дані", "Тестові дані"), loc='lower right')

plt.show()

def polynomial\_features(X, degree):

from itertools import combinations\_with\_replacement

# комбінації\_з\_повторами('ABC', 2) --> AA AB AC BB BC CC

n\_features, n\_samples = np.shape(X)

def index\_combinations(): # (1, 2) => [(1),(2),(1,1),(1,2),(2,2)]

combs = [combinations\_with\_replacement(range(n\_features), i) for i in range(0, degree + 1)]

# comb = [(),((1),(2)),((1,1),(1,2),(2,2))]

flat\_combs = [item for sublist in combs for item in sublist]

# flat\_combs = [(1),(2),(1,1),(1,2),(2,2)]

return flat\_combs

combinations = index\_combinations()

n\_output\_features = len(combinations)

X\_new = np.empty((n\_output\_features, n\_samples))

for i, index\_combs in enumerate(combinations):

X\_new[i, :] = np.prod(X[index\_combs, :], axis=0)

# if index\_combs == (1,2,3) => X\_new[:,i] = X[:,1] \* X[:,2] \* X[:,3]

return X\_new

def mean\_squared\_error(y\_true, y\_pred):

"""Повертає середньоквадратичну помилку між y\_true та y\_pred

Аргументи:

y\_true -- масив справжніх значень

y\_pred -- масив спрогнозованих значень

Повертає:

mse -- середньоквадратична помилка

"""

mse = (1 / len(y\_true.T)) \* np.sum((y\_true - y\_pred) \*\* 2)

return mse

class L2Regularization:

""" Регуляризація для рідж-регресії """

def \_\_init\_\_(self, alpha):

""" Встановлює alpha """

self.alpha = alpha

def \_\_call\_\_(self, w):

"""

Обчислює штраф l2 регуляризації

Аргументи:

w -- вагові коефіцієнти

Повертає:

term -- 1/2 \* alpha \* norm(w)^2

"""

term = 1 / 2 \* self.alpha \* np.linalg.norm(w) \*\* 2

return term

def grad(self, w):

"""

Обчислює похідну штрафа l2 регуляризації

Аргументи:

w -- вагові коефіцієнти

Повертає:

vector -- alpha \* w

"""

derivative = self.alpha \* w

return derivative

class PolynomialRidgeRegression(object):

"""

Параметри:

-----------

degree: int

Степінь полінома, на який буде перетворено незалежну змінну X

reg\_factor: float

Коефіцієнт який визначає кількість регуляризації та звуження незалежних змінних

n\_iterations: int

Кількість тренувальних ітерацій алгоритму

learning\_rate: float

Розмір кроку, який буде застосований при оновленні вагових коефіцієнтів

"""

def \_\_init\_\_(self, degree, reg\_factor, n\_iterations=3000, learning\_rate=0.01, print\_error=False):

self.degree = degree

self.regularization = L2Regularization(alpha=reg\_factor)

self.n\_iterations = n\_iterations

self.learning\_rate = learning\_rate

self.print\_error = print\_error

def initialize\_with\_zeros(self, n\_features):

"""

Ця функція створює вектор нулів формою (n\_features, 1)

Аргументи:

n\_features -- кількість незалежних змінних

"""

self.w = np.zeros((n\_features, 1))

def fit(self, X, Y):

# Генеруємо змінні полінома

X = polynomial\_features(X, self.degree)

# Вставляємо одиниці для вагових коефіцієнтів випадкової похибки

X = np.concatenate((np.ones((1, len(X[0]))), X), axis=0)

# Створюємо масив

self.initialize\_with\_zeros(n\_features=X.shape[0])

# Виконуємо градієнтний спуск для n\_iterations ітерацій

for i in range(self.n\_iterations):

# Прогнозуємо дані

H = self.w.T.dot(X)

# Градієнт штрафу l2

grad\_w = np.dot(X, (H - Y).T) + self.regularization.grad(self.w)

# Оновлюємо вагові коефіцієнти

self.w = self.w - self.learning\_rate \* grad\_w

if self.print\_error and i % 1000 == 0:

# Обчислюємо l2 штраф

mse = mean\_squared\_error(Y, H)

print("MSE після ітерації %i: %f" % (i, mse))

def predict(self, X):

# Генеруємо поля полінома

X = polynomial\_features(X, self.degree)

# Вставляємо одиниці для вагових коефіцієнтів випадкової похибки

X = np.concatenate((np.ones((1, len(X[0]))), X), axis=0)

# Прогнозуємо дані

y\_pred = self.w.T.dot(X)

return y\_pred

poly\_degree = 15

learning\_rate = 0.001

n\_iterations = 40000

reg\_factor = 0.1

model = PolynomialRidgeRegression(

degree=poly\_degree,

reg\_factor=reg\_factor,

learning\_rate=learning\_rate,

n\_iterations=n\_iterations,

print\_error=True

)

model.fit(train\_set\_x, train\_set\_y)

y\_predictions = model.predict(test\_set\_x)

mse = mean\_squared\_error(test\_set\_y, y\_predictions)

print("Середньоквадратична помилка на тестовому сеті: %s (фактор регуляризації: %s)" % (mse, reg\_factor))

# Прогнозуємо для усіх точок у наборі даних

y\_val = model.predict(full\_feature\_set\_for\_plot)

# Візуалізуємо результати

m1 = plt.scatter(121.76 \* train\_set\_x, train\_set\_y, color=cmap(0.9), s=10)

m2 = plt.scatter(121.76 \* test\_set\_x, test\_set\_y, color=cmap(0.5), s=10)

plt.plot(121.76 \* full\_feature\_set\_for\_plot.T, y\_val.T, color='black', linewidth=2, label="Прогноз")

plt.suptitle("Поліноміальна рідж-регресія")

plt.title("MSE: %.2f" % mse, fontsize=10)

plt.xlabel('День')

plt.ylabel('Госпіталізації')

plt.legend((m1, m2), ("Тренувальні дані", "Тестові дані"), loc='lower right')

plt.show()

