

**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

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 (<i>лістинг програми лінійної регресії з однією змінною</i>)	19
Додаток Б (<i>лістинг програми лінійної регресії з декількома змінними</i>)	31
Додаток В (<i>лістинг програми поліноміальної рідж-регресії</i>)	42

ВСТУП

Градiєнтнi методи належать до наближених числових методiв розв'язування задач нелiнiйного програмування, оскiльки дають точний розв'язок за нескiнченне i лише в окремих випадках за скiнченне число крокiв. З їх використанням можна розв'язувати будь-яку задачу нелiнiйного програмування, знаходячи, як правило, лише локальний екстремум. Тому застосування цих методiв дає найбільший ефект для розв'язування задач випуклого програмування, де локальний екстремум є одночасно i глобальним.

Також градiєнтнi методи мають широке застосування в аналізі даних, зокрема в регресiйному аналізі. При побудові регресiйної моделі використовується так звана функція витрат, яка може бути ефективно мінімізована саме методом градiєнтного спуску. Для дослідження використання градiєнтних методiв в аналізі 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

1.1 Вступ

Градiєнтні методи належать до наближених числових методiв розв'язування задач нелiнійного програмування, оскiльки дають точний розв'язок за нескiнченне i лише в окремих випадках за скiнченне число крокiв. З їх використанням можна розв'язувати будь-яку задачу нелiнійного програмування, знаходячи, як правило, лише локальний екстремум. Тому застосування цих методiв дає найбільший ефект для розв'язування задач випуклого програмування, де локальний екстремум є одночасно i глобальним.

1.2 Загальний алгоритм градієнтного методу

Всі алгоритми оптимізації на основі градієнтних методів можна описати наступним чином. Ми починаємо з числа ітерацій $k = 0$ та початкової точки

1. *Тест на збіжність.* Якщо умови збіжності виконуються у початковій точці, тоді ми можемо зупинитися і x_k - це рішення.
2. *Обчислення напрямку пошуку.* Обчислюємо вектор p_k , який визначає напрямок у n -мірному просторі уздовж якого ми будемо шукати.
3. *Обчислення довжини кроку.* Знаходимо додатний скаляр α_k такий, що $f(x_k + \alpha_k p_k) < f(x_k)$
4. *Оновлення змінних.* Присвоюємо $x_{k+1} = x_k + \alpha_k p_k, k = k + 1$ і повертаємося до кроку 1.

$$\Delta x_k = \alpha_k p_k$$

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

Нехай є функція $f(x)$, де x – n -мірний вектор $x = [x_1, x_2, \dots, x_n]^T$

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

$$\nabla f(x) \equiv g(x) \equiv \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

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 n змінних є n -мірним вектором, "друга похідна" n -змінної функції визначається n^2 частковими похідними (похідними n перших часткових похідних відносно n змінних):

$$\frac{\partial^2 f}{\partial x_i \partial x_j}, i \neq j \text{ та } \frac{\partial^2 f}{\partial x_i^2}, i = j$$

Якщо часткові похідні $\frac{\partial f}{\partial x_i}$, $\frac{\partial f}{\partial x_j}$ та $\frac{\partial^2 f}{\partial x_i \partial x_j}$ неперервні і f – однозначна, тоді $\frac{\partial^2 f}{\partial x_j \partial x_i}$ існує та $\frac{\partial^2 f}{\partial x_j \partial x_i} = \frac{\partial^2 f}{\partial x_i \partial x_j}$. Таким чином часткові похідні другого порядку можуть бути записані у вигляді матриці Гесе:

$$\nabla^2 f(x) \equiv H(x) \equiv \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \dots & \frac{\partial^2 f}{\partial^2 x_n} \end{bmatrix}$$

яка містить $n(n + 1)/2$ незалежних елементів.

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

$$f(x) = \frac{1}{2}x^T Hx + g^T x + \alpha.$$

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

$$f(x^* + \varepsilon p) = f(x^*) + \varepsilon p^T g(x^*) + \frac{1}{2} \varepsilon^2 p^T H(x^* + \varepsilon \theta p)p,$$

де $0 \leq \theta \leq 1$, ε – скаляр, p – n -мірний вектор.

Для того щоб x^* був локальним мінімумом, для будь-якого вектора p має існувати скінчене ε таке, що $f(x^* + \varepsilon p) \geq f(x^*)$, тобто існує окіл, в якому ця нерівність справджується. Якщо ця умова задовольняється, тоді $f(x^* + \varepsilon p) - f(x^*) \geq 0$ 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 p є довільним вектором, а ε може бути додатним або від'ємним, кожна складова вектора градієнта $g(x^*)$ повинна дорівнювати нулю.

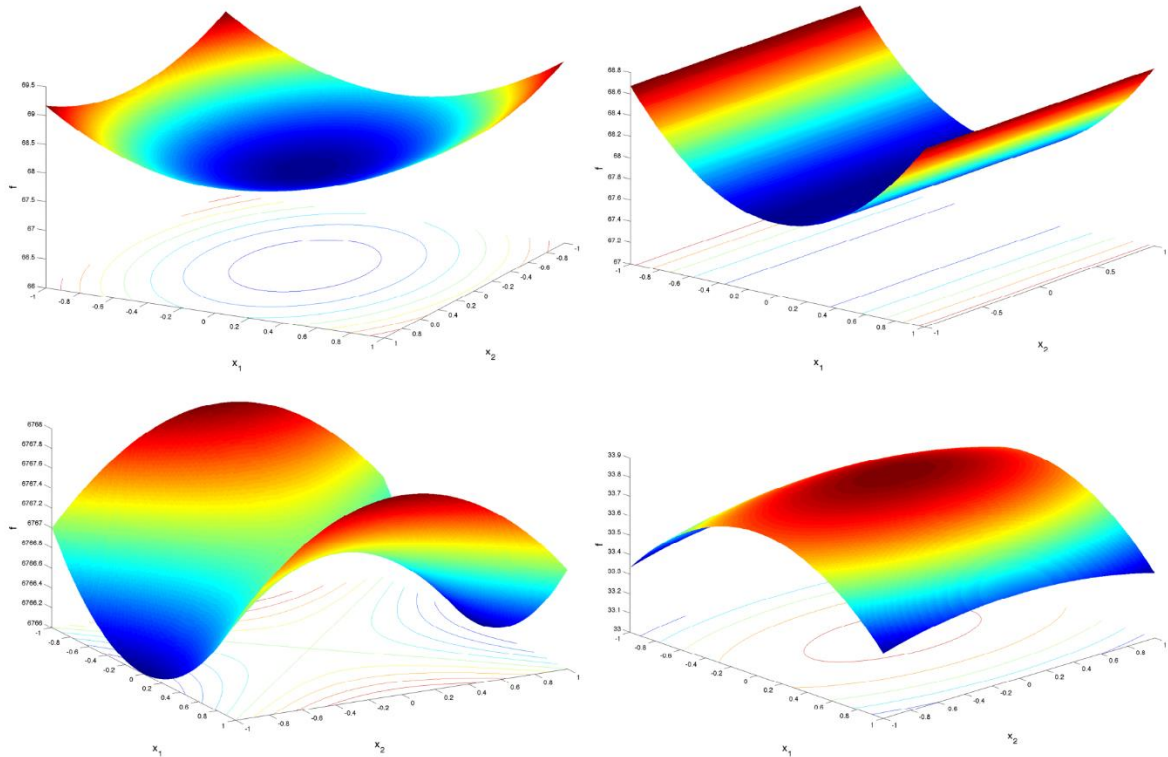
Тепер ми повинні розглянути член другого порядку, $\varepsilon^2 p^T H(x^* + \varepsilon \theta p)p$. Для того, щоб цей термін був невід'ємним, $H(x^* + \varepsilon \theta p)$ повинна бути додатно напіввизначеною, а за неперервністю, матриця Гесе $H(x^*)$ також повинен бути додатно напіввизначеним.

Необхідна умова(для локального мінімуму):

$$\|g(x^*)\| = 0 \text{ та } H(x^*) - \text{додатно напіввизначена}$$

Достатня умова:

$$\|g(x^*)\| = 0 \text{ та } H(x^*) - \text{додатно визначена}$$



Мал. 1. Приклади критичних точок

Приклад 1. Критичні точки функції

Нехай є функція $f(x) = 1.5x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + 0.5x_1^4$. Знайти усі стаціонарні точки та визначити їх тип

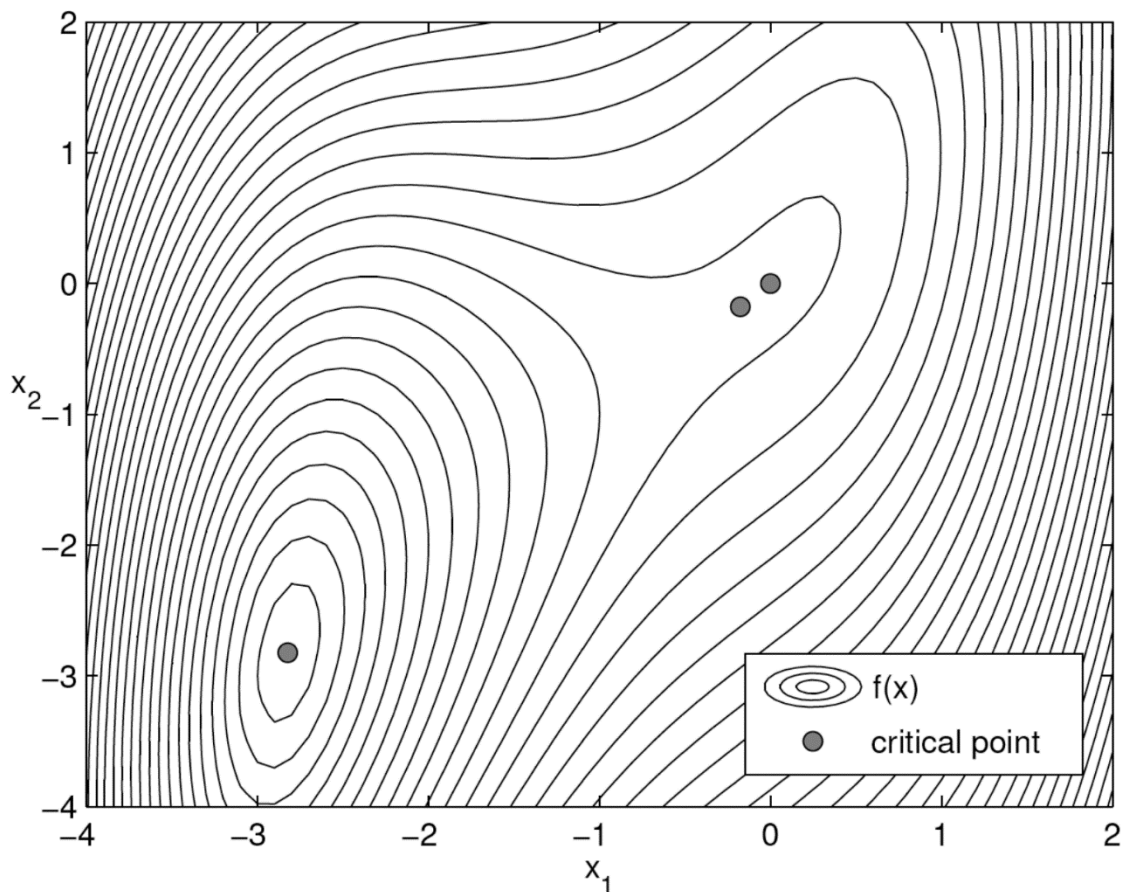
Розв'язок: розв'яжемо $\nabla f(x) = 0$, отримаємо такі три розв'язки:

$(0, 0)$ – локальний мінімум

$1/2(-3 - \sqrt{7}, -3 - \sqrt{7})$ – глобальний мінімум

$1/2(-3 + \sqrt{7}, -3 + \sqrt{7})$ – сідлова точка

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



Мал. 2. Критичні точки функції $f(x) = 1.5x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + 0.5x_1^4$

1.3 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, $g(x_k)$, є також напрямком максимальної швидкості зміни (максимального збільшення) функції в цій точці. Ця швидкість змін задана нормою, $\|g(x_k)\|$.

Алгоритм найшвидшого спуску:

1. Обираємо початкову точку x_0 та параметри збіжності ε_g , ε_a та ε_r .

2. Обчислюємо $g(x_k) \equiv \nabla f(x_k)$. Якщо $\|g(x_k)\| \leq \varepsilon_g$, тоді зупиняємося.

Інакше, обчислюємо нормалізований напрям пошуку

$$p_k = -g(x_k)/\|g(x_k)\|$$

3. Виконуємо лінійний пошук щоб знайти довжину кроку α_k у напрямку p_k .

4. Перевизначаємо поточну точку, $x_{k+1} = x_k + \alpha p_k$.

5. Обчислюємо $f(x_{k+1})$. Якщо умова $|f(x_{k+1}) - f(x_k)| \leq \varepsilon_a + \varepsilon_r |f(x_k)|$ задовольняється за дві успішні ітерації, тоді зупиняємося.

Інакше, $k = k + 1, x_{k+1} = x_k + 1$ та повертаємося до кроку 2.

Отже, нерівність $|f(x_{k+1}) - f(x_k)| \leq \varepsilon_a + \varepsilon_r |f(x_k)|$ є перевіркою успішного зменшення f . ε_a – це абсолютне допустиме відхилення зміни значення функції(зазвичай маленьке $\approx 10^{-6}$) та ε_r – відносне допустиме відхилення(зазвичай рівне 0.01).

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

$$\frac{df(x_{k+1})}{d\alpha} = 0 \Rightarrow \frac{\partial f(x_{k+1})}{\partial x_{k+1}} \frac{\partial x_{k+1}}{\partial \alpha} = 0 \Rightarrow \nabla^T f(x_{k+1}) p_k = 0 \Rightarrow -g^T(x_{k+1}) g(x_k) = 0$$

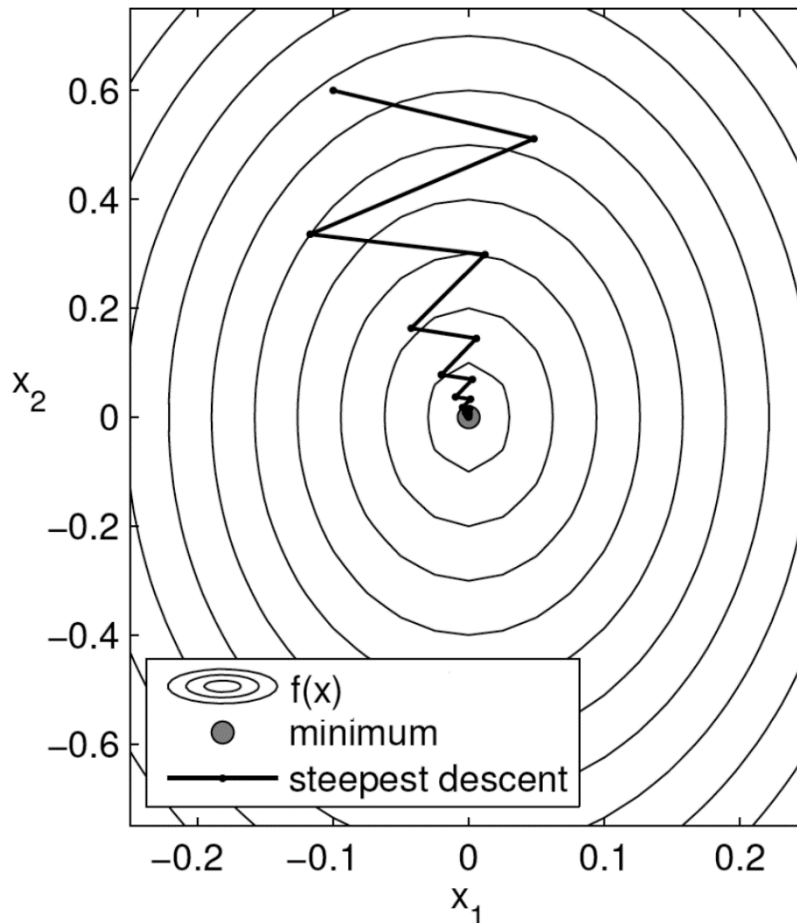
Отже, метод рухається “зигзагами” у просторі, що не є дуже ефективним. Хоча протягом перших кількох ітерацій може спостерігатися суттєве зменшення, після цього метод, як правило, дуже повільний. Зокрема, хоча алгоритм гарантовано збігається, він може зайняти нескінченну кількість ітерацій. Швидкість 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 x_k буде такою ж, як та, яка була отримана на попередньому кроці:

$$\bar{\alpha} g_k^T p_k = \alpha_{k-1} g_{k-1}^T p_{k-1} \rightarrow \bar{\alpha} = \alpha_{k-1} \frac{g_{k-1}^T p_{k-1}}{g_k^T p_k}$$

Приклад 2. Використання методу найшвидшого спуску для $f(x_1, x_2) = 1 - e^{-(10x_1^2 + x_2^2)}$



Мал. 3. Шлях розв'язку методом найшвидшого спуску

1.4 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 x_k , в межах якої ми довіряємо квадратній моделі.

Алгоритм областей довіри:

1. Обираємо початкову точку x_0 та параметр збіжності ε_g та початковий розмір області довіри h_0 .
2. Обчислюємо $g(x_k) \equiv \nabla f(x_k)$. Якщо $\|g(x_k)\| \leq \varepsilon_g$, тоді зупиняємося. Інакше, продовжуємо.
3. Обчислюємо $H(x_k) \equiv \nabla^2 f(x_k)$ та мінімізуємо:

$$q(s_k) = f(x_k) + g(x_k)^T s_k + \frac{1}{2} s_k^T H(x_k) s_k; \quad -h_k \leq s_k \leq h_k, i = \overline{1, n}$$

4. Обчислюємо $f(x_k + s_k)$ та відношення, яке вимірює точність квадратичної моделі

$$r_k = \frac{\Delta f}{\Delta q} = \frac{f(x_k) - f(x_k + s_k)}{f(x_k) - q(s_k)}$$

5. Обчислюємо розмір нової області довіри таким чином:

$$h_{k+1} = \frac{\|s_k\|}{4} \text{ якщо } r_k < 0.25$$

$$h_{k+1} = 2h_k \text{ якщо } r_k > 0.75 \text{ та } h_k = \|s_k\|$$

$$h_{k+1} = h_k \text{ в усіх інших випадках}$$

6. Визначаємо нову точку:

$$x_{k+1} = x_k \text{ якщо } r_k \leq 0$$

$$x_{k+1} = x_k + s_k \text{ в усіх інших випадках}$$

7. Прирівнюємо $k = k + 1$ та повертаємося до кроку 2.

Початкове значення h зазвичай 1.

РОЗДІЛ 2

МЕТОДИ ГРАДІЄНТНОГО СПУСКУ У РІЗНИХ МОДЕЛЯХ АНАЛІЗУ ДАНИХ

2.1 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.

$$X_{\text{ст}} = \frac{X - \mu}{\sigma}$$

Побудуємо таку модель для кожного значення $x^{(i)}$:

$$h^{(i)} = \theta x^{(i)} + b$$

Також маємо таку функцію витрат:

$$J(\theta, b) = \frac{1}{2m} \sum_{i=1}^m (h^{(i)} - y^{(i)})^2$$

Основні кроки побудови навчального алгоритму будуть такі:

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

Задамо таке H , що $H = (\theta X + b) = (h^{(1)}, h^{(2)}, \dots, h^{(m-1)}, h^{(m)})$, та знайдемо для нього значення функції витрат $J(\theta, b) = \frac{1}{2m} \sum_{i=1}^m (h^{(i)} - y^{(i)})^2$

Знайдемо градієнт функції витрат:

$$\frac{\partial J}{\partial \theta} = \frac{1}{m} X(H - Y)^T$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (h^{(i)} - y^{(i)})$$

Мета – знайти значення θ та b , при яких значення функції витрат найменше. Для параметра θ правило спуску буде виглядати так: $\theta_{\text{н}} = \theta - \alpha \partial J$, де α – швидкість навчання.

2.2 Градієнтний спуск у лінійній регресії з декількома змінними

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 Б)

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

$$X_{\text{ст}} = \frac{X - \mu}{\sigma}$$

Побудуємо таку модель для кожного значення $x^{(i)}$:

$$h^{(i)} = w^T x^{(i)} + b$$

Також маємо таку функцію витрат:

$$J(\theta, b) = \frac{1}{2m} \sum_{i=1}^m (h^{(i)} - y^{(i)})^2$$

Основні кроки побудови навчального алгоритму будуть такі:

- 4) Визначити структуру моделі(наприклад кількість незалежних змінних)
- 5) Ініціалізувати параметри моделі
- 6) Виконати основний цикл
 - a) Обчислити значення функції витрат
 - b) Обчислити градієнт
 - c) Оновити параметри моделі

Задамо таке H , що $H = (w^T X + b) = (h^{(1)}, h^{(2)}, \dots, h^{(m-1)}, h^{(m)})$, та знайдемо для нього значення функції витрат $J = \frac{1}{2m} \sum_{i=1}^m (h^{(i)} - y^{(i)})^2$

Знайдемо градієнт функції витрат:

$$\frac{\partial J}{\partial w} = \frac{1}{m} X(H - Y)^T$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (h^{(i)} - y^{(i)})$$

Мета – знайти значення w та b , при яких значення функції витрат буде найменше. Для параметра θ правило спуску буде виглядати так: $\theta_n = \theta - \alpha \partial J$, де α – розмір кроку.

2.3 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 відрізьку від 0 до 1.

Основна ідея поліноміальної регресії – комбінації незалежних змінних із заданою степінню. Нехай ϵ степінь $degree = 3$ та $x^{(i)} = (x_1)$. Тоді вектор незалежних змінних буде виглядати так:

$$x^{(i)} = ((x_1^{(i)})^0 = 1, x_1^{(i)}, (x_1^{(i)})^2, (x_1^{(i)})^3)$$

Додамо одиницю для того щоб разом рахувати й випадкову похибку. Отже, $x^{(i)}$ буде виглядати так:

$$x^{(i)} = (1, (x_1^{(i)})^0 = 1, x_1^{(i)}, (x_1^{(i)})^2, (x_1^{(i)})^3)$$

Тоді маємо таку прогнозуючу функцію: $h^{(i)} = w^T x^{(i)}$

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

$$J = \frac{1}{2} \sum_{i=1}^m (h^{(i)} - y^{(i)})^2 + \frac{1}{2} \lambda \|w\|_2^2,$$

де λ – регуляризація, а $\|w\|_2$ – Евклідова норма.

Означимо градієнт:

$$\begin{aligned} X &= (x^{(1)}, x^{(2)}, \dots, x^{(m-1)}, x^{(m)}) \\ H &= w^T X = (h^{(1)}, h^{(2)}, \dots, h^{(m-1)}, h^{(m)}) \\ \frac{\partial J}{\partial w} &= X(H - Y)^T + \lambda w \end{aligned}$$

Також у програмі було додано функцію `polynomial_features` для утворення комбінацій. Наприклад, для $degree = 3$ та незалежних змінних (x_1, x_2, x_3) отримаємо:

$$\rightarrow (1, x_1, x_2, x_3, x_1^2, x_1 x_2, x_1 x_3, x_2^2, x_2 x_3, x_3^2, x_1^3, x_1^2 x_2, x_1^2 x_3, x_1 x_2^2, x_1 x_2 x_3, x_1 x_3^2, x_2^3, x_2^2 x_3, x_2 x_3^2, x_3^3)$$

Середньоквадратичну помилку було обчислено за формулою:

$$MSE = \frac{1}{m} \sum_{i=1}^m (h^{(i)} - y^{(i)})^2$$

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 використовуються 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 Д0-4, 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

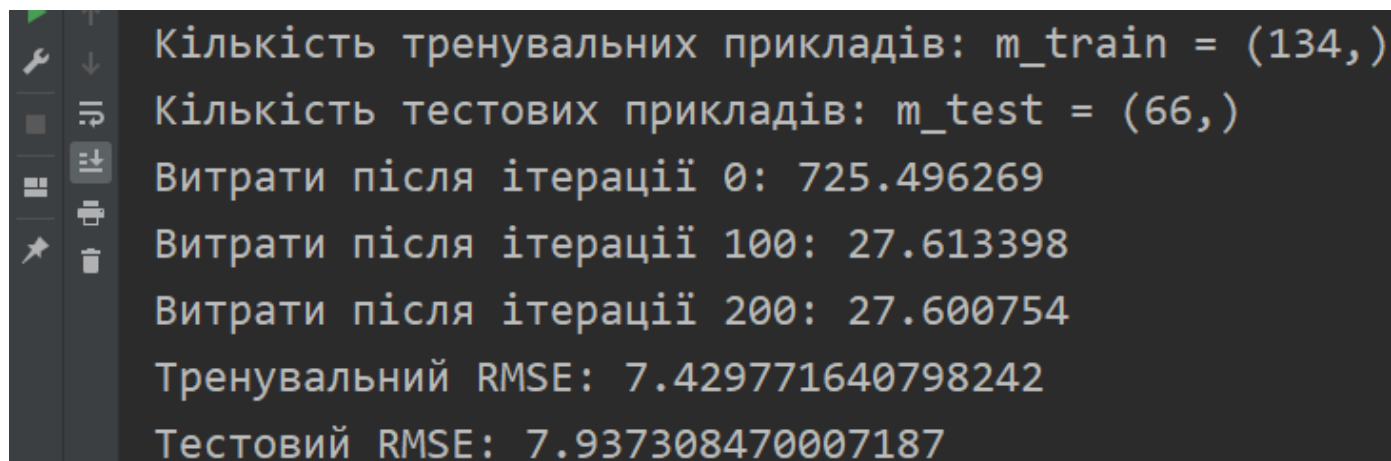
d = model(train_set_x, train_set_y, test_set_x, test_set_y,
num_iterations=300, learning_rate=0.05, print_cost=True)

# 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"]

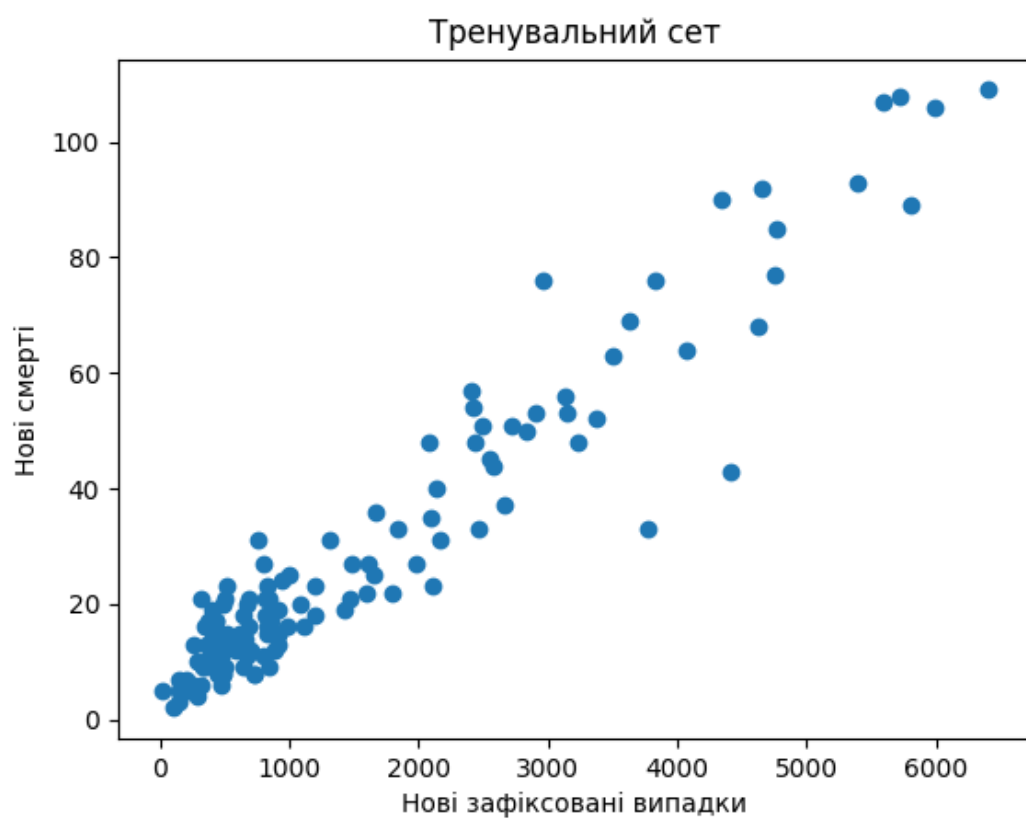
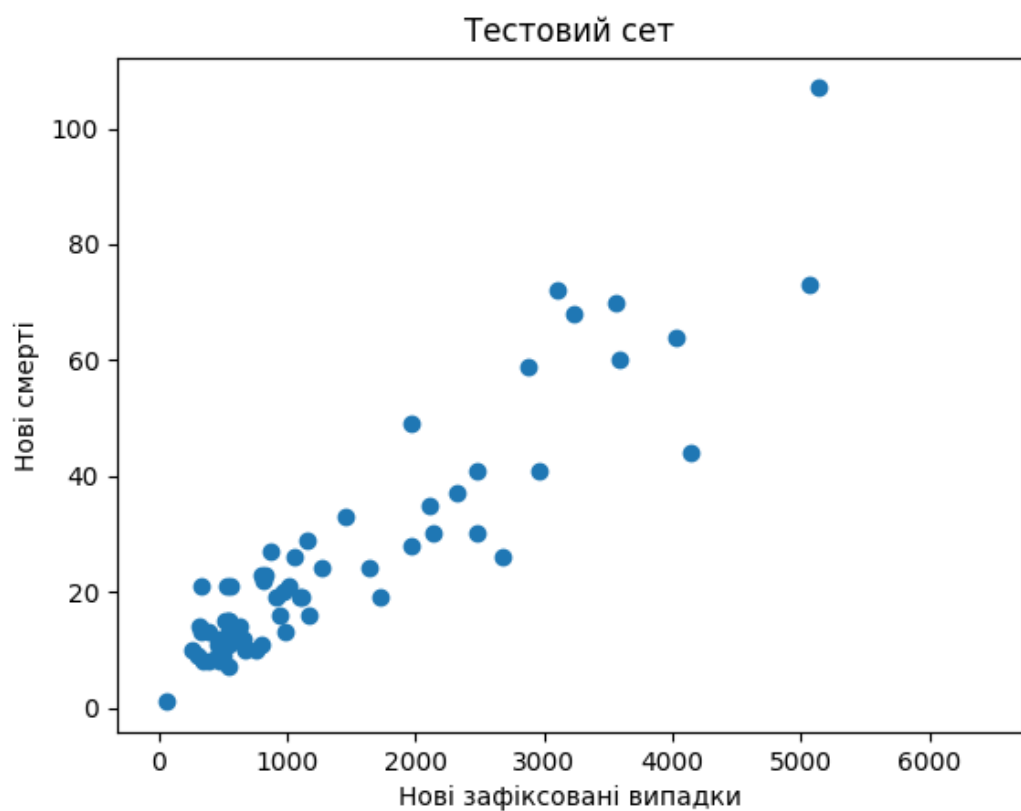
```

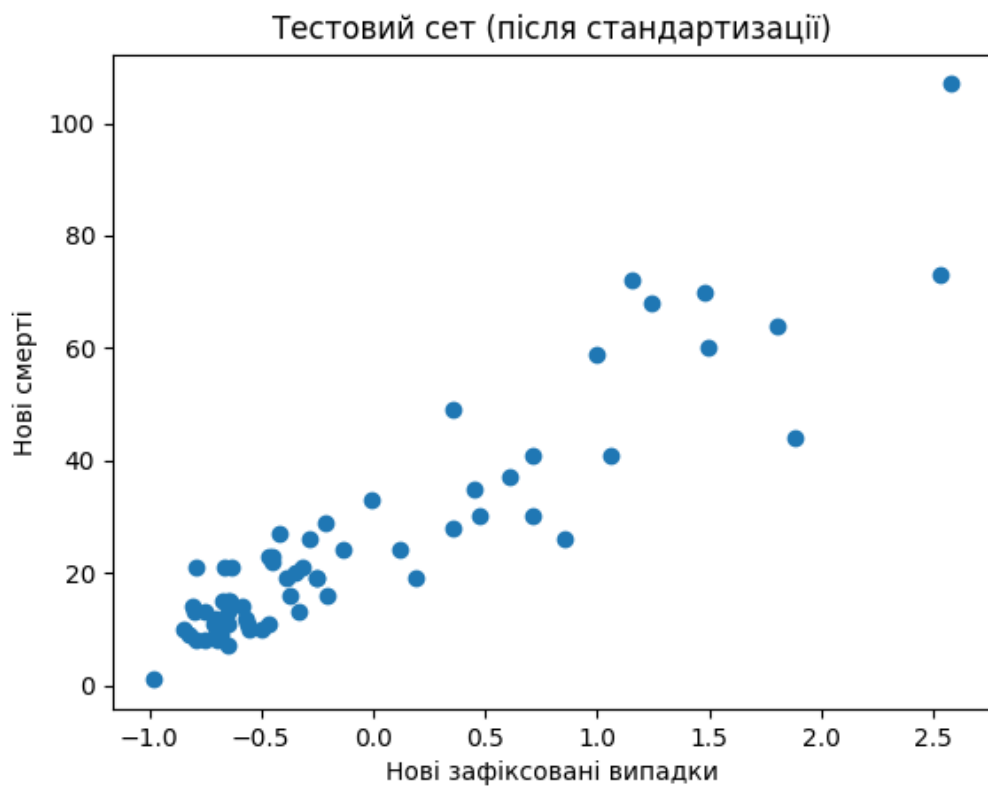
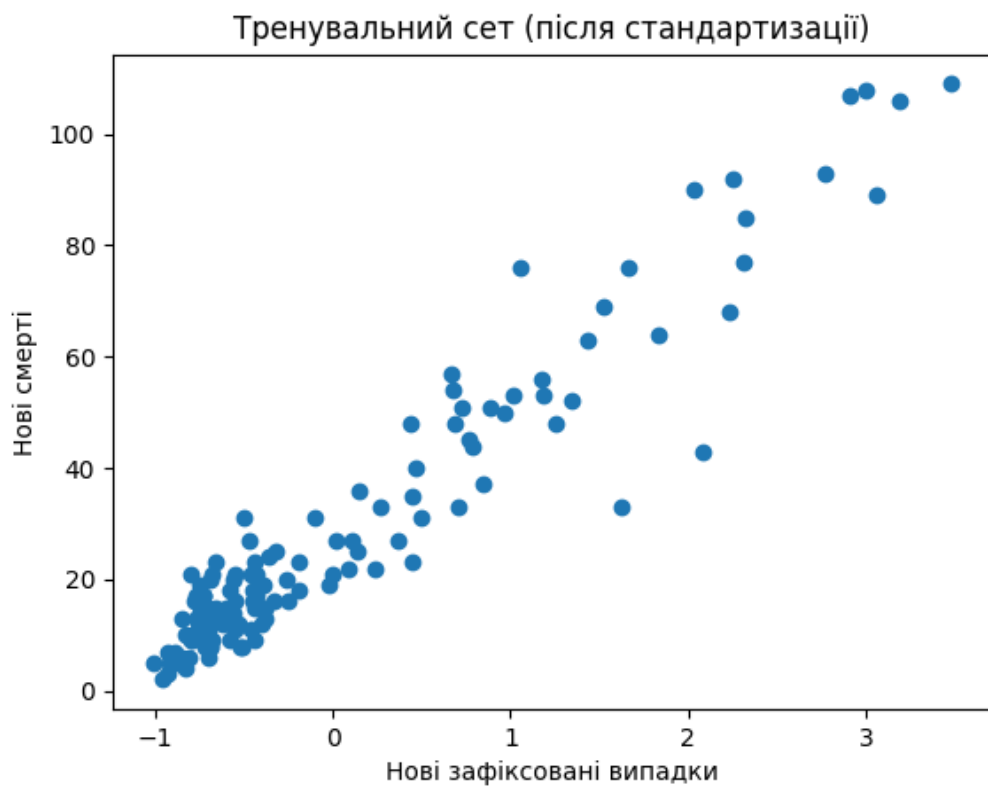
```
b = d["b"]
y = theta * x + b
plt.plot(x, y, color='orange')
plt.xlabel("Нові зафіксовані випадки")
plt.ylabel("Нові смерті")
plt.show()

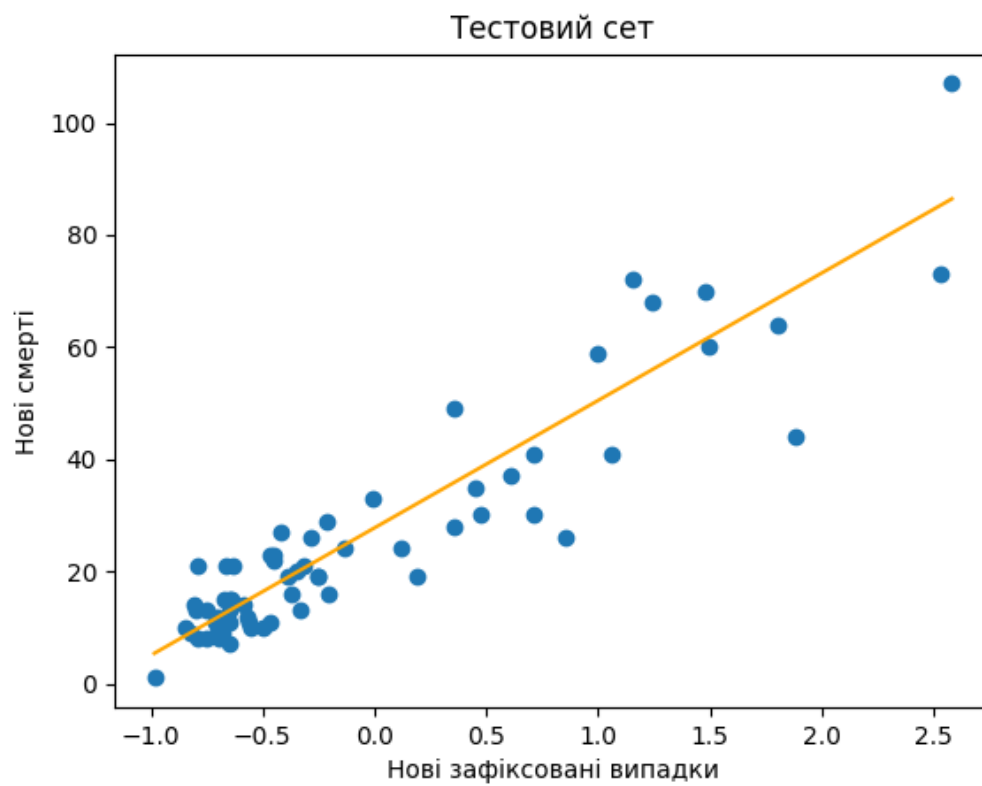
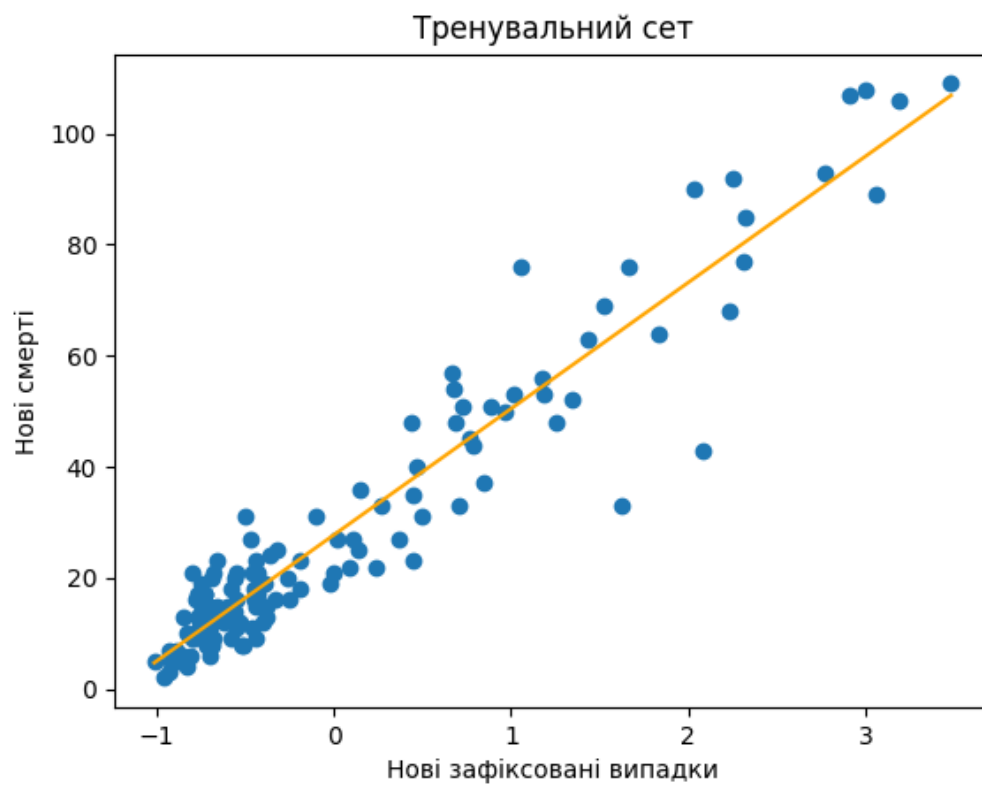
# Тестовий сет даних
plt.title("Тестовий сет")
plt.scatter(test_set_x, test_set_y)
x = np.array([min(test_set_x), max(test_set_x)])
theta = d["theta"]
b = d["b"]
y = theta * x + b
plt.plot(x, y, color='orange')
plt.xlabel("Нові зафіксовані випадки")
plt.ylabel("Нові смерті")
plt.show()
```

A terminal window with a dark background and light-colored text. On the left side, there is a vertical toolbar with icons for a wrench, a downward arrow, a refresh symbol, a list, a print icon, and a trash can. The terminal output displays the following metrics:

```
Кількість тренувальних прикладів: m_train = (134,)
Кількість тестових прикладів: m_test = (66,)
Витрати після ітерації 0: 725.496269
Витрати після ітерації 100: 27.613398
Витрати після ітерації 200: 27.600754
Тренувальний RMSE: 7.429771640798242
Тестовий RMSE: 7.937308470007187
```







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

```
# Майстренко Олександр Д0-4, 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 та 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 знадобиться 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")

```

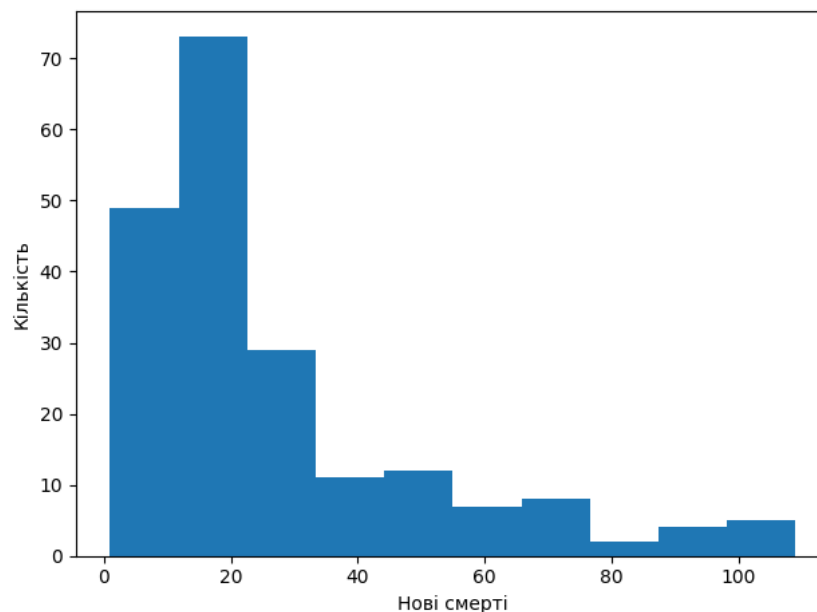
```
plt.scatter(train_set_y, d["Y_prediction_train"])
plt.plot([0, 100], [0, 100], "--k")
plt.xlabel("Реальні нові смерті")
plt.ylabel("Спрогнозовані нові смерті")
plt.show()
```

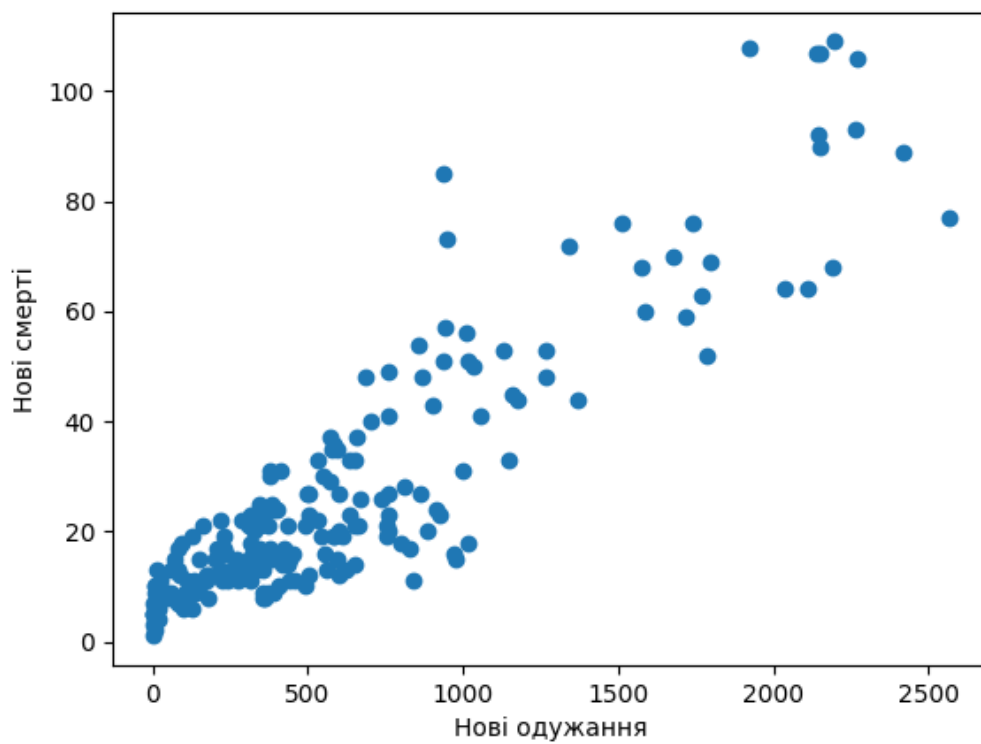
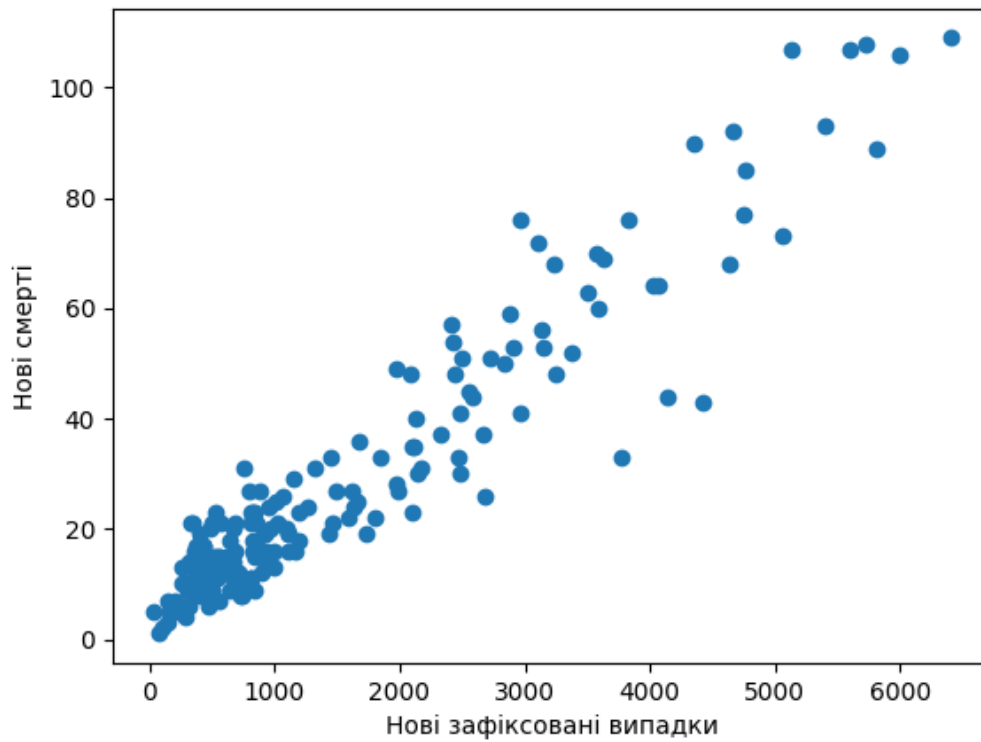
Тестовий сет даних

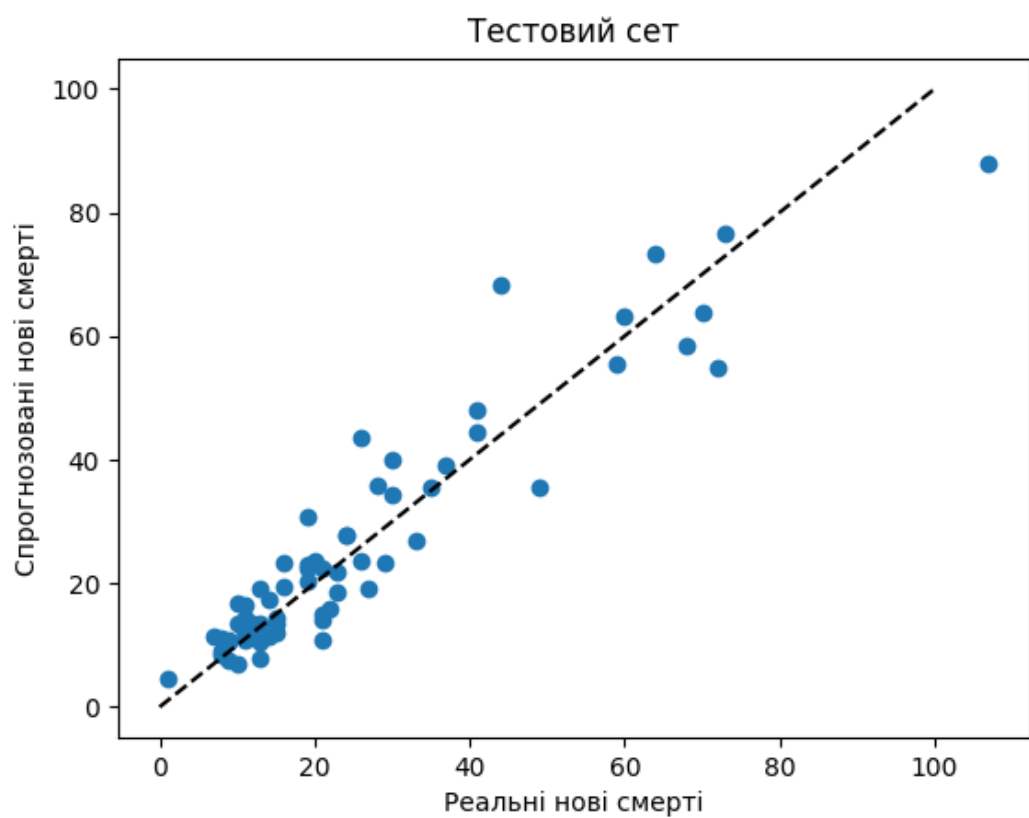
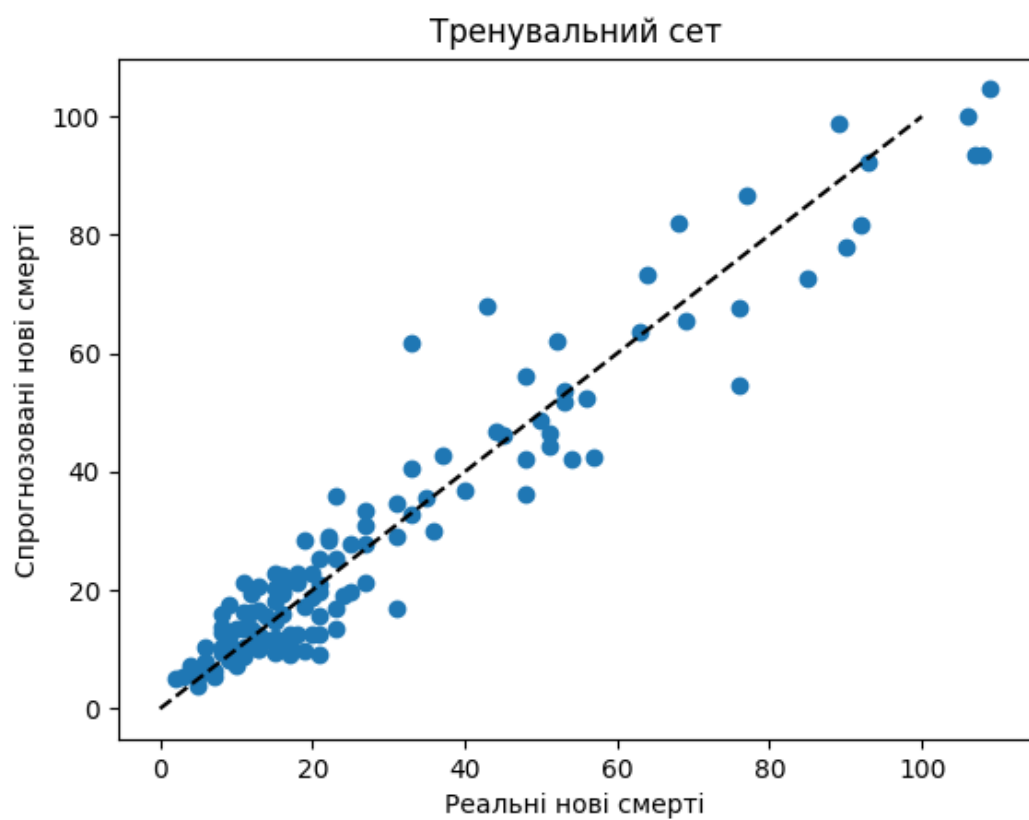
```
plt.title("Тестовий сет")
plt.scatter(test_set_y, d["Y_prediction_test"])
plt.plot([0, 100], [0, 100], "--k")
plt.xlabel("Реальні нові смерті")
plt.ylabel("Спрогнозовані нові смерті")
plt.show()
```

```
Кількість тренувальних прикладів: m_train = 134
Кількість тестових прикладів: m_test = 66
Витрати після ітерації 0: 725.496269
Витрати після ітерації 100: 26.549294
Витрати після ітерації 200: 25.465421
Витрати після ітерації 300: 25.109077
Витрати після ітерації 400: 24.990504
```

```
Витрати після ітерації 1900: 24.931373
Тренувальний RMSE: 7.061355848075666
Тестовий RMSE: 6.932328581946508
```







Додаток В (лістинг програми поліноміальної рідж-регресії):

```
# Майстренко Олександр Д0-4, 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')
```

```
# Візуалізація 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 --  $\frac{1}{2} * \alpha * \text{norm}(w)^2$ 
        """
        term = 1 / 2 * self.alpha * np.linalg.norm(w) ** 2
        return term

    def grad(self, w):
        """
        Обчислює похідну штрафу 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

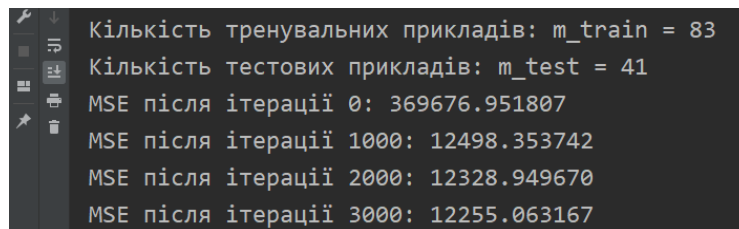
```



```

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 підж-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()

```

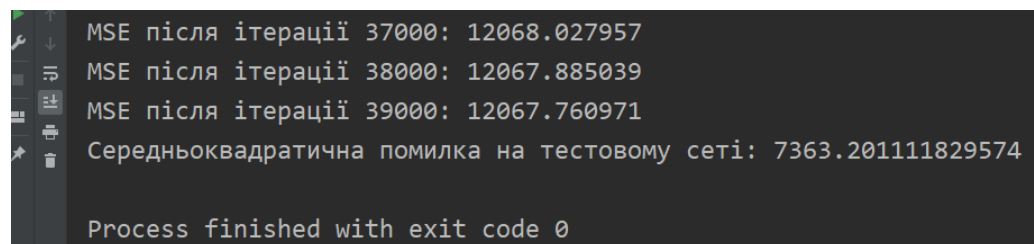


A terminal window with a dark background and light-colored text. On the left side, there is a vertical toolbar with icons for file operations (like open, save, copy, paste) and a search icon. The main area of the terminal displays the following text:

```

Кількість тренувальних прикладів: m_train = 83
Кількість тестових прикладів: m_test = 41
MSE після ітерації 0: 369676.951807
MSE після ітерації 1000: 12498.353742
MSE після ітерації 2000: 12328.949670
MSE після ітерації 3000: 12255.063167

```



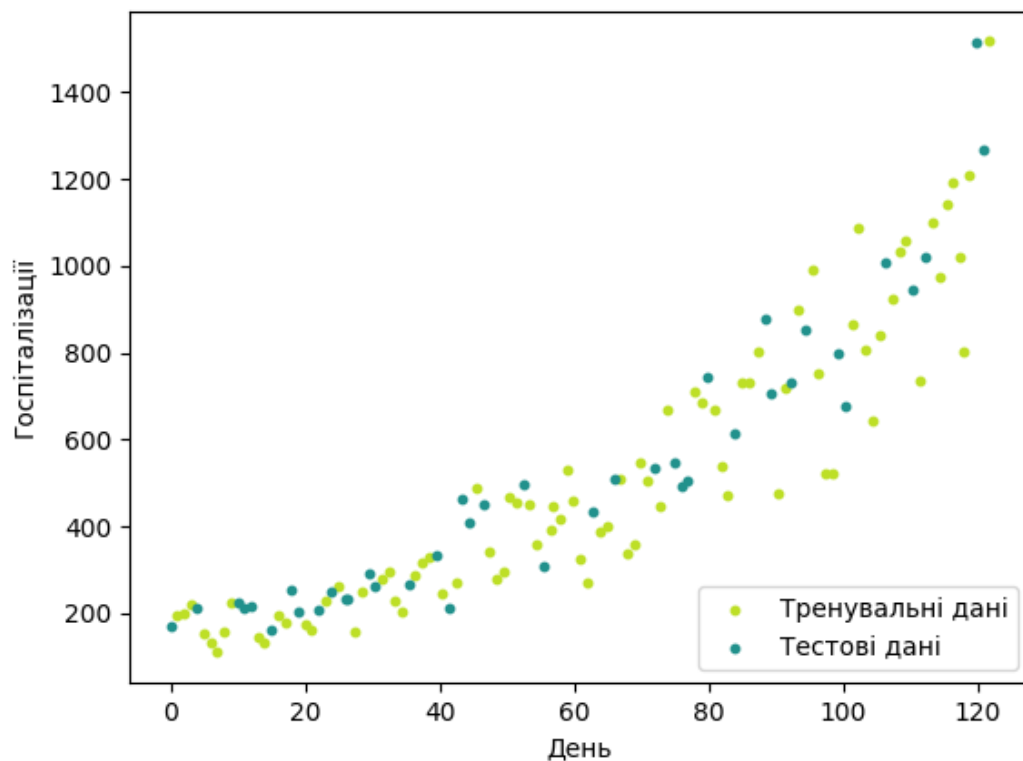
A terminal window with a dark background and light-colored text. On the left side, there is a vertical toolbar with icons for file operations (like open, save, copy, paste) and a search icon. The main area of the terminal displays the following text:

```

MSE після ітерації 37000: 12068.027957
MSE після ітерації 38000: 12067.885039
MSE після ітерації 39000: 12067.760971
Середньоквадратична помилка на тестовому сеті: 7363.201111829574

Process finished with exit code 0

```



Поліноміальна рідж-регресія

MSE: 7363.20

