Лабораторна робота №4

ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи регресії даних у машинному навчанні.

Хід роботи

Репозиторій GITHUB: https://github.com/AnatoliiYarmolenko/SMI_1S4C

Завдання 2.1. Створення регресора однієї змінної

Виконаємо наступну програму. Вона будує лінійну регресійну модель на основі одного параметра, навчає її на 80% даних, перевіряє точність на решті, будує графік реальних і прогнозованих значень, обчислює показники якості та зберігає модель у файл для подальшого використання:

```
# Імпорт необхідних бібліотек
import pickle
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt

# ----------------------------
# 1. Завантаження вхідних даних
# ----------------------------
input_file = 'Lab_4/data_singlevar_regr.txt'

# Файл має формат CSV з комами
data = np.loadtxt(input_file, delimiter=',')

# Поділ на X (вхідна змінна) і y (вихідна змінна)
X, y = data[:, :-1], data[:, -1]

# ----------------------------
# 2. Розбиття даних на навчальні та тестові
# ----------------------------
num_training = int(0.8 * len(X))
num_test = len(X) - num_training
```

					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.25.121.31.000 – Лр4		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Ярмоленко А.М.			Звіт з лабораторної роботи №4	Літ.	Арк. 1 / Аркушів 21
Перевір.		Маєвський О.В.					
Реценз.						ФІКТ, гр. ІПЗ-22-4	
Н. Контр.							
Зав.каф.		Вакалюк Т.А.					

```
X_train, y_train = X[:num_training], y[:num_training]
X_test, y_test = X[num_training:], y[num_training:]

# ----------------------------
# 3. Створення та навчання моделі
# ----------------------------
regressor = linear_model.LinearRegression()
regressor.fit(X_train, y_train)

# ----------------------------
# 4. Прогнозування результатів
# ----------------------------
y_test_pred = regressor.predict(X_test)

# ----------------------------
# 5. Побудова графіка
# ----------------------------
plt.scatter(X_test, y_test, color='green', label='Справжні дані')
plt.plot(X_test, y_test_pred, color='black', linewidth=3, label='Прогноз моделі')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Лінійна регресія (одна змінна)')
plt.show()

# ----------------------------
# 6. Оцінка якості моделі
# ----------------------------
print("Linear regressor performance:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_test_pred),
2))
print("Explained variance score =", round(sm.explained_variance_score(y_test,
y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))

# ----------------------------
# 7. Збереження моделі у файл
# ----------------------------
output_model_file = 'model.pkl'
with open(output_model_file, 'wb') as f:
    pickle.dump(regressor, f)

print("\nМодель збережено у файл:", output_model_file)

# ----------------------------
# 8. Завантаження моделі з файлу
# ----------------------------
with open(output_model_file, 'rb') as f:
    regressor_model = pickle.load(f)
```

```
# Перевірка завантаженої моделі
y_test_pred_new = regressor_model.predict(X_test)
print("\nNew mean absolute error =", round(sm.mean_absolute_error(y_test,
y_test_pred_new), 2))
```

В результаті отримаємо наступний графік та вивід в консоль:

```
SMI_1S4C/Lab_4/LR_4_task_1.py"
Linear regressor performance:
Mean absolute error = 0.59
Mean squared error = 0.49
Median absolute error = 0.51
Explained variance score = 0.86
R2 score = 0.86

Модель збережено у файл: model.pkl

New mean absolute error = 0.59
```

Рис. 1 Консольний вивід

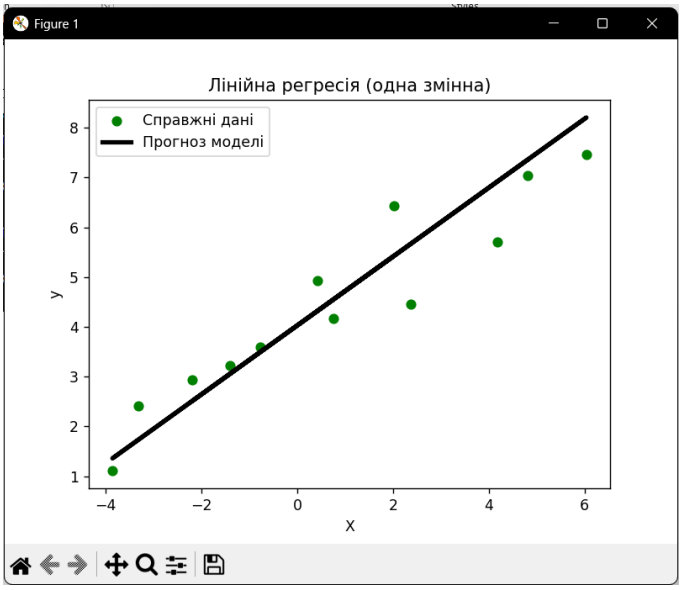

Рис. 2 Графічний вивід

Отримані результати демонструють, що модель досить точно відображає залежність між змінними: лінія регресії добре наближує точки вибірки, а значення коефіцієнта детермінації близьке до 1, що свідчить про високу якість апроксимації. Невеликі значення середньої та медіанної абсолютної похибки вказують на те, що прогноз моделі узгоджується з реальними даними, а отже, побудована регресійна модель є ефективною для опису цієї залежності.

Завдання 2.2. Передбачення за допомогою регресії однієї змінної (2 варіант)

Аналогічно представимо програму, яка зчитує дві колонки даних із файлу data_regr_2.txt, навчає лінійну регресійну модель за допомогою LinearRegression, обчислює коефіцієнти рівняння прямої, проводить прогнозування, оцінює точність моделі за метриками R^2 та MSE, а також візуалізує результати.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# ----------------------------
# 1. Завантаження вхідних даних
# ----------------------------
input_file = 'Lab_4/data_regr_2.txt'

# Файл має формат CSV з комами
data = np.loadtxt(input_file, delimiter=',')

X, y = data[:, :-1], data[:, -1]

# Побудова моделі лінійної регресії
model = LinearRegression()
model.fit(X, y)

# Отримання параметрів моделі
a = model.coef_[0]
b = model.intercept_

# Прогноз
y_pred = model.predict(X)
# Оцінка якості
r2 = r2_score(y, y_pred)
mse = mean_squared_error(y, y_pred)
# Вивід результатів
print(f"Рівняння регресії: y = {a:.3f}x + {b:.3f}")
print(f"Коефіцієнт детермінації R²: {r2:.3f}")
print(f"Середньоквадратична помилка (MSE): {mse:.3f}")
# Побудова графіка
plt.scatter(X, y, color='blue', label='Дані')
plt.plot(X, y_pred, color='red', linewidth=2, label='Регресійна пряма')
plt.title('Лінійна регресія (одна змінна)')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```

В результаті отримаємо наступні графіки та вивід в консоль даних:

```
Рівняння регресії: y = 0.048x + 2.761
Коефіцієнт детермінації R²: 0.002
Середньоквадратична помилка (MSE): 13.605
```

Рис. 3 Вивід у консоль

Побудована модель лінійної регресії дозволяє наближено описати залежність між змінними у наборі даних. Якщо коефіцієнт детермінації R^2 близький до 1, модель добре пояснює варіацію цільової змінної. У разі нижчого значення R^2 — залежність між ознакою та цільовою змінною є слабшою або нелінійною. Отже, отримана модель може використовуватись для базових передбачень, але для більшої точності варто перевірити можливість застосування нелінійної регресії або поліноміальної моделі.

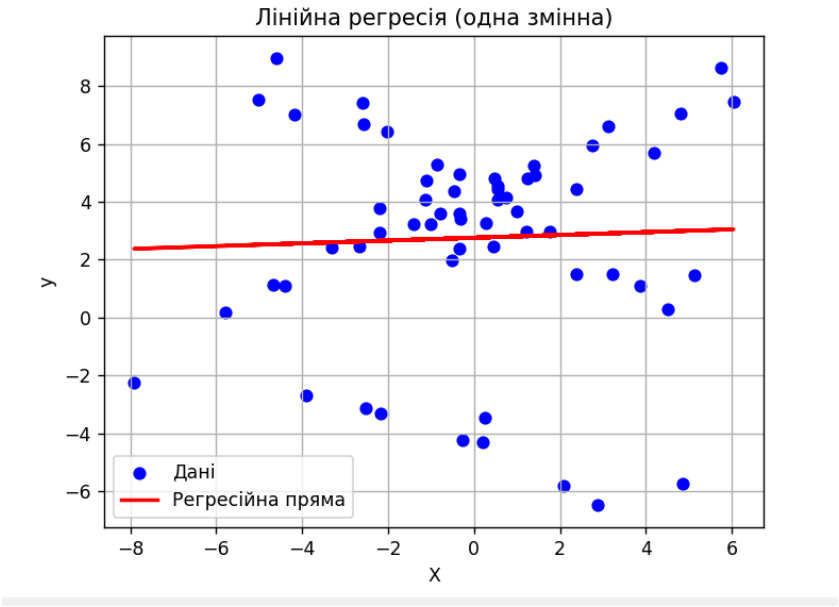

Рис. 4 Графічний вивід

Завдання 2.3. Створення багатовимірного регресора

Наступна програма зчитує багатовимірні дані з файлу data_multivar_regr.txt, розділяє їх на навчальний і тестовий набори, будує модель лінійної регресії, оцінює її якість за п’ятьма метриками, а також створює поліноміальну регресійну модель 10-го ступеня для покращення точності передбачень. Потім вона порівнює результати обох моделей на прикладі вибіркової точки:

```
# ----------------------------------------------------------
# Багатовимірна лінійна та поліноміальна регресія
# ----------------------------------------------------------

import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
from sklearn.preprocessing import PolynomialFeatures

# ----------------------------------------------------------
# 1. Завантаження вхідних даних
# ----------------------------------------------------------
input_file = 'Lab_4/data_multivar_regr.txt'  # файл із даними
data = np.loadtxt(input_file, delimiter=',')

# Поділ на вхідні змінні (X) і вихідну (y)
X, y = data[:, :-1], data[:, -1]

# ----------------------------------------------------------
# 2. Розбиття даних на навчальні та тестові
# ----------------------------------------------------------
num_training = int(0.8 * len(X))
num_test = len(X) - num_training

X_train, y_train = X[:num_training], y[:num_training]
X_test, y_test = X[num_training:], y[num_training:]

# ----------------------------------------------------------
# 3. Лінійна регресія
# ----------------------------------------------------------
linear_regressor = linear_model.LinearRegression()
linear_regressor.fit(X_train, y_train)

# Прогнозування
y_test_pred = linear_regressor.predict(X_test)

# ----------------------------------------------------------
# 4. Оцінка якості лінійного регресора
# ----------------------------------------------------------
print("Linear Regressor performance:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_test_pred),
2))
print("Explained variance score =", round(sm.explained_variance_score(y_test,
y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))

# ----------------------------------------------------------
# 5. Поліноміальна регресія (ступінь 10)
# ----------------------------------------------------------
```

```
polynomial = PolynomialFeatures(degree=10)
X_train_transformed = polynomial.fit_transform(X_train)

poly_linear_model = linear_model.LinearRegression()
poly_linear_model.fit(X_train_transformed, y_train)

# ---------------------------------------------------------
# 6. Перевірка на вибірковій точці
# ---------------------------------------------------------
datapoint = [[7.75, 6.35, 5.56]]
poly_datapoint = polynomial.fit_transform(datapoint)

print("\nLinear regression prediction:", linear_regressor.predict(datapoint))
print("Polynomial regression prediction:", poly_linear_model.predict(poly_datapoint))
```

За результатами виконання видно, що поліноміальний регресор забезпечує точніше передбачення, значення якого ближче до очікуваного (близько 41.35), ніж звичайна лінійна регресія. Це свідчить, що додавання поліноміальних ознак дозволяє моделі краще враховувати нелінійні залежності між змінними, підвищуючи точність прогнозу.

```
Linear Regressor performance:
Mean absolute error = 3.58
Mean squared error = 20.31
Median absolute error = 2.99
Explained variance score = 0.86
R2 score = 0.86

Linear regression prediction: [36.05286276]
Polynomial regression prediction: [41.08245747]
```

Рис. 5 Вивід у консоль

Завдання 2.4. Регресія багатьох змінних

Програма завантажує набір даних diabetes із sklearn.datasets, який містить 10 змінних і показник прогресування діабету. Дані поділяються на навчальну та тестову вибірки (по 50%). Створюється модель лінійної регресії, яка навчається на тренувальних даних, прогнозує результати для тестових даних, а потім обчислюються показники якості — R^2, MAE та MSE:

```
# ---------------------------------------------------------
# Лінійна регресія на наборі даних про діабет
# ---------------------------------------------------------
```

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split

# ----------------------------------------------------------
# 1. Завантаження даних
# ----------------------------------------------------------
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

# ----------------------------------------------------------
# 2. Розбиття на навчальну та тестову вибірки
# ----------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=0)

# ----------------------------------------------------------
# 3. Створення та навчання моделі лінійної регресії
# ----------------------------------------------------------
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# ----------------------------------------------------------
# 4. Прогнозування результатів
# ----------------------------------------------------------
y_pred = regr.predict(X_test)

# ----------------------------------------------------------
# 5. Оцінка якості моделі
# ----------------------------------------------------------
print("Коефіцієнти регресії:", regr.coef_)
print("Вільний член (intercept):", regr.intercept_)
print("R2 score:", round(r2_score(y_test, y_pred), 3))
print("Mean Absolute Error (MAE):", round(mean_absolute_error(y_test, y_pred), 3))
print("Mean Squared Error (MSE):", round(mean_squared_error(y_test, y_pred), 3))

# ----------------------------------------------------------
# 6. Побудова графіка
# ----------------------------------------------------------
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred, edgecolors=(0, 0, 0), label='Передбачені значення')
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=3, label='Ідеальна
відповідність')
ax.set_xlabel('Виміряно')
ax.set_ylabel('Передбачено')
ax.set_title('Порівняння справжніх і передбачених значень (Діабет)')
ax.legend()
plt.show()
```

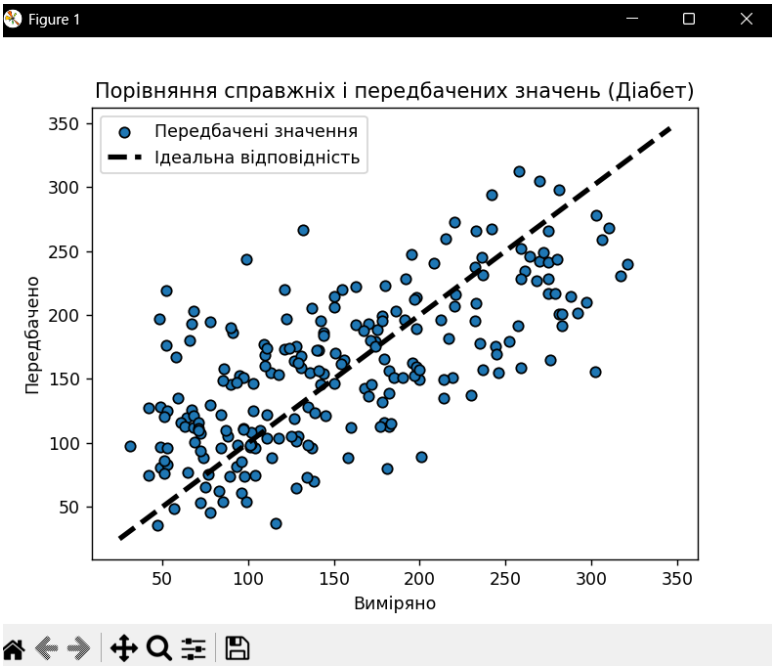

Рис. 6 Графік

Модель лінійної регресії демонструє помірну точність на даних про діабет. Коефіцієнт детермінації R^2 зазвичай становить близько 0.47, що означає, що модель пояснює приблизно половину варіації результатів. Значення MAE та MSE показують середній рівень відхилення прогнозів від реальних даних. На графіку видно, що передбачення моделі розташовані доволі близько до ідеальної прямої, отже, лінійна регресія є прийнятним базовим методом для цього набору даних, хоча для кращих результатів можуть знадобитися складніші моделі.

```
Коефіцієнти регресії: [ -20.4047621  -265.88518066  564.65086437  325.56226865 -692.16120333
  395.55720874   23.49659361  116.36402337  843.94613929   12.71856131]
Вільний член (intercept): 154.3589285280134
R2 score: 0.438
Mean Absolute Error (MAE): 44.801
Mean Squared Error (MSE): 3075.331
```

Рис. 7 Вивід консолі

Завдання 2.5. Самостійна побудова регресії (2 варіант)

Програма генерує 100 випадкових точок згідно з рівнянням $y = 0.6 \cdot X^2 + X + 2 +$ шум, після чого навчає лінійну і поліноміальну (2-го ступеня) регресії. Обидві моделі оцінюються за метриками R^2, MAE і MSE. Результати та графіки порівнюються, щоб показати перевагу поліноміальної регресії для нелінійних даних.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# ------------------------------------------------------
# 1. Генерація випадкових даних
# ------------------------------------------------------
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.6 * X**2 + X + 2 + np.random.randn(m, 1)  # модель з шумом

# ------------------------------------------------------
# 2. Лінійна регресія
# ------------------------------------------------------
lin_reg = LinearRegression()
lin_reg.fit(X, y)
y_lin_pred = lin_reg.predict(X)

print("\nЛінійна регресія:")
print("Коефіцієнт:", lin_reg.coef_)
print("Вільний член:", lin_reg.intercept_)
print("R2 =", round(r2_score(y, y_lin_pred), 3))
print("MAE =", round(mean_absolute_error(y, y_lin_pred), 3))
print("MSE =", round(mean_squared_error(y, y_lin_pred), 3))

# ------------------------------------------------------
# 3. Поліноміальна регресія (ступінь 2)
# ------------------------------------------------------
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)

print("\nЗначення X[0]:", X[0])
print("Відповідне X_poly[0]:", X_poly[0])

lin_reg_poly = LinearRegression()
lin_reg_poly.fit(X_poly, y)
y_poly_pred = lin_reg_poly.predict(X_poly)

print("\nПоліноміальна регресія (ступінь 2):")
print("Вільний член:", lin_reg_poly.intercept_)
print("Коефіцієнти:", lin_reg_poly.coef_)
print("R2 =", round(r2_score(y, y_poly_pred), 3))
print("MAE =", round(mean_absolute_error(y, y_poly_pred), 3))
print("MSE =", round(mean_squared_error(y, y_poly_pred), 3))

# ------------------------------------------------------
# 4. Побудова графіків
# ------------------------------------------------------
X_new = np.linspace(-3, 3, 100).reshape(100, 1)
```

```
y_lin_new = lin_reg.predict(X_new)
X_new_poly = poly_features.transform(X_new)
y_poly_new = lin_reg_poly.predict(X_new_poly)

plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Вихідні дані')
plt.plot(X_new, y_lin_new, color='red', linewidth=2, label='Лінійна регресія')
plt.plot(X_new, y_poly_new, color='green', linewidth=2, label='Поліноміальна регресія
(2 ступінь)')
plt.title('Порівняння лінійної та поліноміальної регресії')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()

# ------------------------------------------------------------
# 5. Математичні рівняння моделей
# ------------------------------------------------------------
print("\nМодельні дані (справжнє рівняння): y = 0.6 * X^2 + X + 2 + шум")
print("Отримана модель лінійної регресії: y = {:.3f} * X +
{:.3f}".format(float(lin_reg.coef_), float(lin_reg.intercept_)))
print("Отримана модель поліноміальної регресії: y = {:.4f} * X^2 + {:.4f} * X +
{:.4f}".format(
    lin_reg_poly.coef_[0,1], lin_reg_poly.coef_[0,0],
float(lin_reg_poly.intercept_)))
```

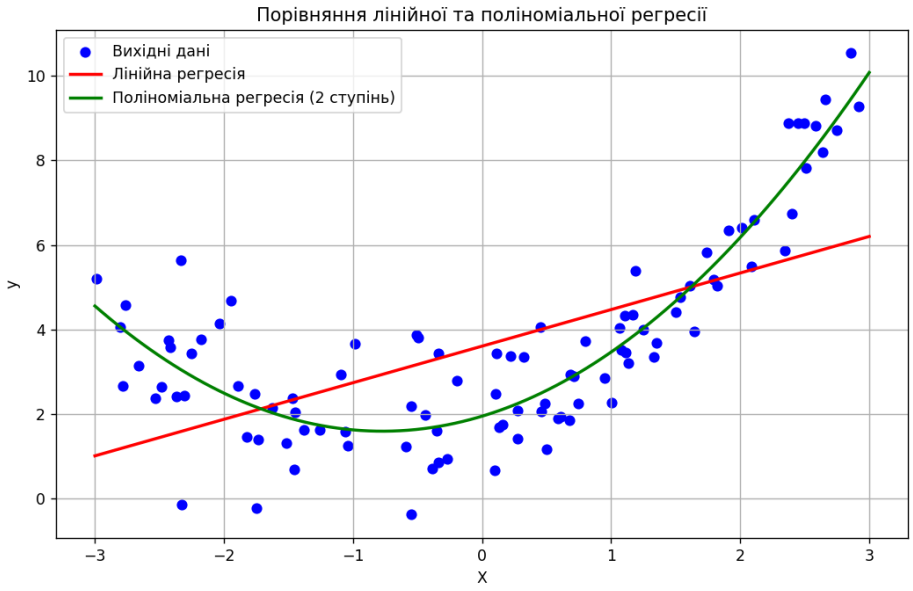

Рис. 8 Графік порівняння

Відповідно наша модель математичного рівняння: $y = 0.6 * X^2 + X + 2 +$ шум.

Отримана модель математичного рівняння: $y = 0.56 * X^2 + 0.86 * X + 2.18$.

```
Лінійна регресія:
Коефіцієнт: [[0.82742004]]
Вільний член: [3.92832594]
R2 = 0.394
MAE = 1.494
MSE = 3.27

Значення X[0]: [-0.59113783]
Відповідне X_poly[0]: [-0.59113783  0.34944394]

Поліноміальна регресія (ступінь 2):
Вільний член: [2.17689998]
Коефіцієнти: [[0.85746611 0.55802852]]
R2 = 0.812
MAE = 0.813
MSE = 1.012
```

Рис. 9 Вивід у консоль

```
Отримана модель поліноміальної регресії: y = 0.5580 * X^2 + 0.8575 * X + 2.1769
```

Рис. 10 Вивід отриманого рівняння

Лінійна регресія не здатна точно описати залежність між X і y, оскільки дані мають нелінійний (квадратичний) характер. Поліноміальна регресія другого ступеня забезпечує значно вищий коефіцієнт детермінації R^2, а її коефіцієнти дуже близькі до модельних (близько 0.6 для X^2, 1 для X, 2 для константи). Це підтверджує, що модель навчена правильно та відтворює справжню закономірність даних.

Завдання 2.6. Побудова кривих навчання

Програма генерує набір даних на основі квадратичної функції, до якої додається випадковий шум. Далі для трьох моделей — лінійної, поліноміальної 2-го ступеня та поліноміальної 10-го ступеня — будується крива навчання.

Функція plot_learning_curves() поступово збільшує розмір навчального набору, кожного разу обчислюючи середньоквадратичну помилку (RMSE) на навчальних і перевірочних даних. Це дозволяє візуально оцінити, як модель навчається і узагальнює дані.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# --- 1. Генерація даних (з попереднього завдання) ---
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.4 * X ** 2 + X + 4 + np.random.randn(m, 1)

# --- 2. Функція для побудови кривих навчання ---
def plot_learning_curves(model, X, y, title):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
    train_errors, val_errors = [], []

    # Навчання на все більших підмножинах даних
    for m in range(5, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train[:m], y_train_predict))
        val_errors.append(mean_squared_error(y_val, y_val_predict))

    # Побудова графіка
    plt.plot(np.sqrt(train_errors), "r-", linewidth=2, label="Навчальний набір")
    plt.plot(np.sqrt(val_errors), "b-", linewidth=2, label="Перевірочний набір")
    plt.xlabel("Розмір навчального набору")
    plt.ylabel("RMSE")
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.show()

# --- 3. Лінійна модель ---
lin_reg = LinearRegression()
plot_learning_curves(lin_reg, X, y, "Криві навчання — лінійна регресія")

# --- 4. Поліноміальна модель 10-го ступеня ---
poly_reg_10 = Pipeline([
    ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
    ("lin_reg", LinearRegression())
])
plot_learning_curves(poly_reg_10, X, y, "Криві навчання — поліноміальна регресія
(ступінь 10)")

# --- 5. Поліноміальна модель 2-го ступеня ---
poly_reg_2 = Pipeline([
```

```
    ("poly_features", PolynomialFeatures(degree=2, include_bias=False)),
    ("lin_reg", LinearRegression())
])
plot_learning_curves(poly_reg_2, X, y, "Криві навчання — поліноміальна регресія
(ступінь 2)")
```

Рис. 11 Лінійна регресія

Лінійна регресія демонструє високу помилку на певних проміжках — це ознака недонавчання (high bias), оскільки модель занадто проста для відображення квадратичної залежності.

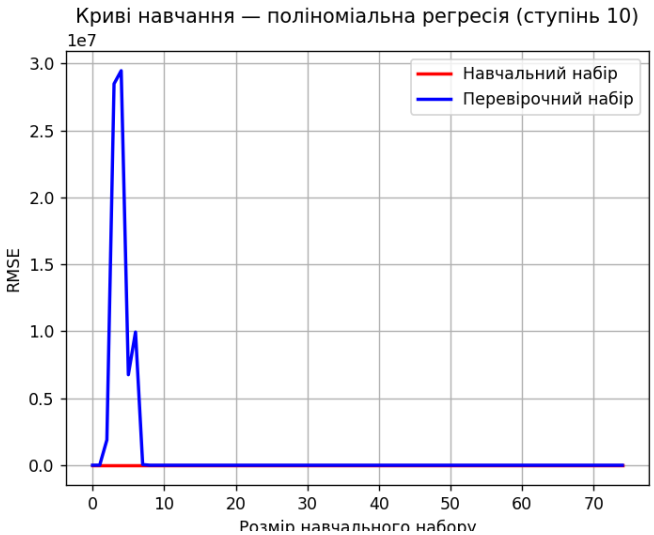

Рис. 12 Поліноміальна регресія 10 ступеня

Поліноміальна модель 10-го ступеня має дуже значну похибку на початкових даних — це ознака перенавчання (high variance). Така модель надто чутлива до шуму в даних.

Рис. 13 Поліноміальна регресія 2 ступеня

Поліноміальна модель 2-го ступеня показує оптимальний баланс між точністю на навчальних і перевірочних вибірках. Криві знаходяться близько одна до одної, що свідчить про гарне узагальнення моделі.

Отже, найкращі результати для даного набору даних демонструє поліноміальна регресія другого ступеня, яка забезпечує найкращий компроміс між зміщенням та дисперсією.

Завдання 2.7. Експериментально отримані N-значень величини Y при значеннях величини X. Відшукати параметри функції за методом найменших квадратів. Побудувати графіки, де в декартовій системі координат нанести експериментальні точки і графік апроксимуючої функції.

2							
	X	-1	-1	0	1	2	3
	Y	-1	0	1	1	3	5

Рис. 14 Варіант 2

Наша мета знайти лінійну функцію вигляду $y = \beta_0 + \beta_1 x$, яка найкраще "підходить" (апроксимує) точки даних. Відповідно β_0 це зрушення, а β_1 нахил нашої прямої.

Для знаходження прямої методом найменших квадратів нам необхідно знайти такі зрушення та нахил, при яких сума квадратів похибок між реальними точками та прогнозованими на прямій буде мінімальною. Відповідно до варіанту маємо наступну систему рівнянь:

$$-1 = \beta_0 - \beta_1$$
$$-1 = \beta_0$$
$$0 = \beta_0 + \beta_1$$
$$1 = \beta_0 + \beta_1$$
$$2 = \beta_0 + 3\beta_1$$
$$3 = \beta_0 + 5\beta_1$$

Знаходимо мінімум фіункції:

$$S(\beta_0, \beta_1) = [-1 - (\beta_0 - \beta_1)]^2 + [-1 - \beta_0]^2 + [-\beta_0 - \beta_1]^2 + [1 - (\beta_0 + \beta_1)]^2 + [2 - (\beta_0 - 3\beta_1)]^2 + [3 - (\beta_0 - 5\beta_1)]^2$$

Визначаємо мінімум через обчислення часткової похідної від $S(\beta_0, \beta_1)$ щодо β_0 і β_1 і прирівнюванням її до нуля:

$$\frac{dS}{d\beta_0} = 0 = 12\beta_0 + 8\beta_1 - 18$$
$$\frac{dS}{d\beta_1} = 0 = 8\beta_0 + 32\beta_1 - 46$$

Розв’язавши цю систему рівнянь отримаємо: $\beta_0 = 0.65, \beta_1 = 1.275$.

Перевіримо розв’язок розробивши програму для знаходження прямої та виведення графіку:

```python
import numpy as np
import matplotlib.pyplot as plt

# --- 1. Введення даних (Варіант 2) ---
# Експериментально отримані значення X та Y
x_data = np.array([-1, -1, 0, 1, 2, 3])
y_data = np.array([-1, 0, 1, 1, 3, 5])

print(f"Дані X: {x_data}")
print(f"Дані Y: {y_data}\n")

# --- 2. Обчислення сум для методу найменших квадратів ---
n = len(x_data)           # Кількість точок (n)
sum_x = np.sum(x_data)    # Сума X
```

```
sum_y = np.sum(y_data)     # Сума Y
sum_x2 = np.sum(x_data**2) # Сума X^2
sum_xy = np.sum(x_data * y_data) # Сума X*Y

print("--- Проміжні розрахунки ---")
print(f"Кількість точок (n): {n}")
print(f"Σx = {sum_x}")
print(f"Σy = {sum_y}")
print(f"Σx^2 = {sum_x2}")
print(f"Σxy = {sum_xy}\n")

# --- 3. Складання та розв'язання системи "нормальних рівнянь" ---
# Ми розв'язуємо систему Ax = B, де x = [b0, b1]
# A = [[n,    sum_x],
#      [sum_x, sum_x2]]
# B = [[sum_y],
#      [sum_xy]]

A = np.array([[n, sum_x],
              [sum_x, sum_x2]])

B = np.array([sum_y, sum_xy])

# Використовуємо np.linalg.solve для розв'язання системи
# beta - це вектор [b0, b1]
try:
    beta = np.linalg.solve(A, B)
    beta_0 = beta[0]
    beta_1 = beta[1]

    print("--- Результати (коефіцієнти) ---")
    print(f"Коефіцієнт β0 (зсув): {beta_0:.3f}")
    print(f"Коефіцієнт β1 (нахил): {beta_1:.3f}")
    print(f"\nРівняння апроксимуючої функції: y = {beta_0:.3f} + {beta_1:.3f}x")

    # --- 4. Побудова графіків ---
    # Створюємо набір точок для побудови лінії регресії
    # Беремо трохи ширший діапазон, ніж min і max x_data
    x_line = np.linspace(min(x_data) - 0.5, max(x_data) + 0.5, 100)
    # Розраховуємо y для цих точок за нашим рівнянням
    y_line = beta_0 + beta_1 * x_line

    # Налаштування графіка
    plt.figure(figsize=(10, 6))

    # 1. Графік експериментальних точок
    plt.scatter(x_data, y_data, color='red', s=100, label='Експериментальні точки (X,
Y)')

    # 2. Графік апроксимуючої функції (лінії регресії)
```

```
    plt.plot(x_line, y_line, color='blue', label=f'Апроксимуюча функція\ny =
{beta_0:.3f} + {beta_1:.3f}x')

    # Додавання підписів та легенди
    plt.title('Лінійна регресія за методом найменших квадратів (Варіант 2)')
    plt.xlabel('Величина X')
    plt.ylabel('Величина Y')
    plt.legend()
    plt.grid(True)

    # Показати графік
    plt.show()

except np.linalg.LinAlgError:
    print("Помилка: неможливо розв'язати систему рівнянь. Можливо, матриця A є
сингулярною.")
```

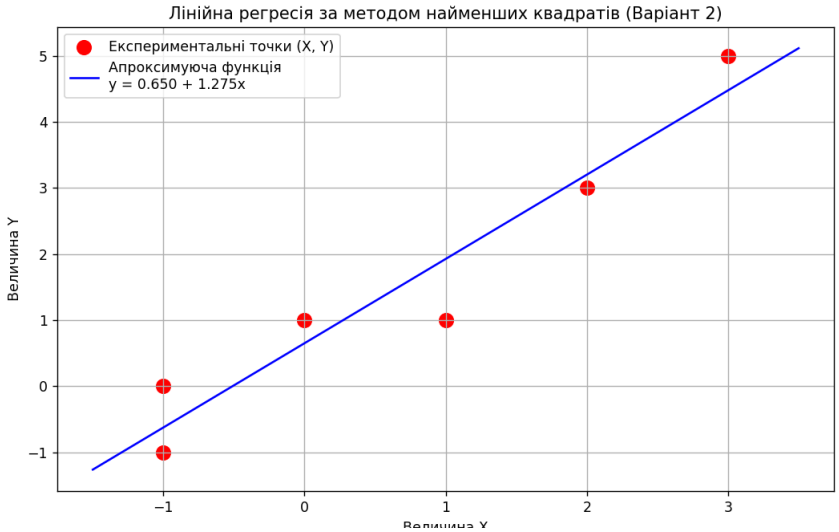

Рис. 15 Отриманий графік

```
Дані X: [-1 -1  0  1  2  3]
Дані Y: [-1  0  1  1  3  5]

--- Проміжні розрахунки ---
Кількість точок (n): 6
Σx = 4
Σy = 9
Σx^2 = 16
Σxy = 23

--- Результати (коефіцієнти) ---
Коефіцієнт β0 (зсув): 0.650
Коефіцієнт β1 (нахил): 1.275

Рівняння апроксимуючої функції: y = 0.650 + 1.275x
```

Рис. 16 Рівняння прямої

Оскільки розв’язок отриманий вручну та програмним метод співпадають, можна зробити висновок, що значення нахилу та зрушення визначені правильно.

Завдання 2.8. Виконати інтерполяцію функції, задану в табличній формі в п'яти точках (див. нижче). Розрахунки виконати в середовищі Python.

Вектори даних:

$$x := \begin{pmatrix} 0.1 \\ 0.3 \\ 0.4 \\ 0.6 \\ 0.7 \end{pmatrix} \qquad y := \begin{pmatrix} 3.2 \\ 3 \\ 1 \\ 1.8 \\ 1.9 \end{pmatrix}$$

Рис. 17 Значення змінних

Згідно поставлених кроків складаємо програму:

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

# --- Вхідні дані ---
# Вектори даних з Завдання № 3 [cite: 157-167]
x_data = np.array([0.1, 0.3, 0.4, 0.6, 0.7])
y_data = np.array([3.2, 3.0, 1.0, 1.8, 1.9])

N = len(x_data) # Кількість точок (N=5), отже степінь полінома N-1 = 4

print(f"Вхідні дані X: {x_data}")
print(f"Вхідні дані Y: {y_data}\n")

# --- Крок 1: Заповнення матриці X (Матриця Вандермонда) ---
# Створюємо матрицю X, де кожен рядок [1, x, x^2, x^3, x^4]
# (відповідно до теорії в лаб. роботі [cite: 148])
X = np.vander(x_data, N, increasing=True)
print("--- Крок 1: Матриця X (Вандермонда) ---")
print(X)
print("\n")

# --- Крок 2: Отримання коефіцієнтів інтерполяційного полінома ---
# Ми розв'язуємо систему лінійних рівнянь X * A = Y,
# де A - це вектор коефіцієнтів [a0, a1, a2, a3, a4]
# A = (X^-1) * Y
A = np.linalg.solve(X, y_data)
print("--- Крок 2: Коефіцієнти полінома [a0, a1, a2, a3, a4] ---")
print(A)
print("\n")

# --- Крок 3: Визначення функції полінома (степеню 4) ---
#
# Створюємо функцію полінома.
```

```
# np.poly1d очікує коефіцієнти від старшого степеня до молодшого,
# тому ми передаємо наш вектор A у зворотному порядку (A[::-1])
p = np.poly1d(A[::-1])
print("--- Крок 3: Інтерполяційний поліном P(x) ---")
print(p)
print("\n")

# --- Крок 4: Побудова графіка функції ---
# Створюємо 100 точок для плавної лінії графіка
x_plot = np.linspace(min(x_data), max(x_data), 100)
y_plot = p(x_plot)

plt.figure(figsize=(10, 6))
# Малюємо поліном
plt.plot(x_plot, y_plot, label='Інтерполяційний поліном P(x)')
# Малюємо вихідні точки, щоб переконатись, що графік проходить через них
plt.scatter(x_data, y_data, color='red', s=100, zorder=5,
            label='Експериментальні точки')
plt.title('Завдання 3: Інтерполяція поліномом 4-го степеня')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()

# --- Крок 5: Визначити значення функції в проміжних точках ---
x_intermediate_1 = 0.2
x_intermediate_2 = 0.5

y_at_0_2 = p(x_intermediate_1)
y_at_0_5 = p(x_intermediate_2)

print("--- Крок 5: Значення в проміжних точках ---")
print(f"Значення функції в точці x = {x_intermediate_1}:  P(0.2) = {y_at_0_2:.4f}")
print(f"Значення функції в точці x = {x_intermediate_2}:  P(0.5) = {y_at_0_5:.4f}")
```

В результаті отримаємо наступний графік та значення:

Рис. 18 Графік функції

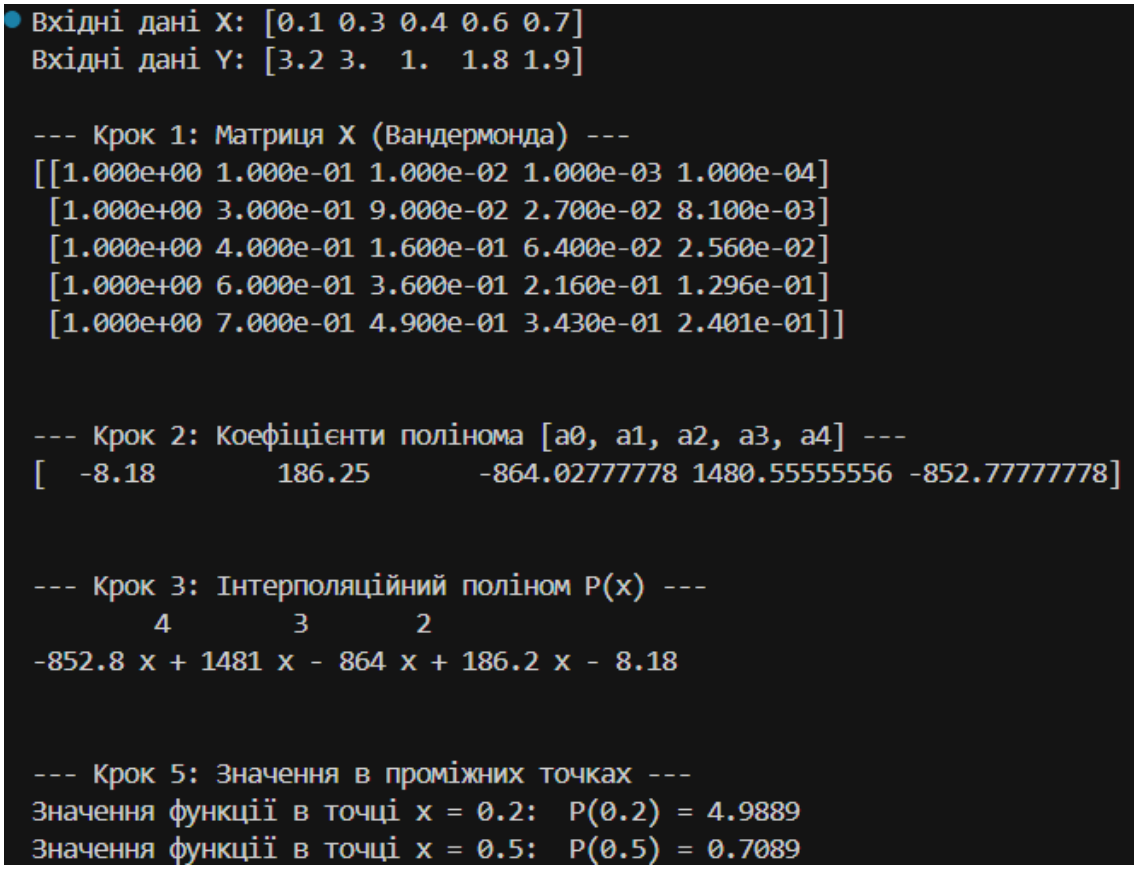

Рис. 19 Отримані значенні

У ході виконання коду було успішно виконано поліноміальну інтерполяцію для заданого набору даних. Шляхом розв'язання системи лінійних рівнянь на основі матриці Вандермонда було знайдено коефіцієнти та побудовано інтерполяційний поліном 4-го степеня. Ця функція, яка за визначенням точно проходить через всі вузлові точки, була використана для обчислення значень у проміжних точках, де P(0.2) склало 4.9889, а P(0.5) = 0.7089, що й демонструє практичне застосування методу для оцінки значень між відомими даними.

Висновок: Під час лабораторної роботи ми побудували лінійну та поліноміальні моделі регресії на різних даних. Лінійна модель показала недонавчання, поліноміальна 10-го ступеня — перенавчання, а поліноміальна 2-го ступеня забезпечила оптимальний баланс між точністю та узагальненням. Криві навчання та графіки наочно продемонстрували вплив складності моделі на помилки навчання та перевірки. Також було виконано завдання з інтерполяції та методу найменших квадратів, реалізовано відповідні алгоритми.