המכללה האקדמית להנדסה בראודה בכרמיאל

המחלקה להנדסת חשמל ואלקטרוניקה

מערכות לומדות ולמידה עמוקה (31245)

Lab 3 report

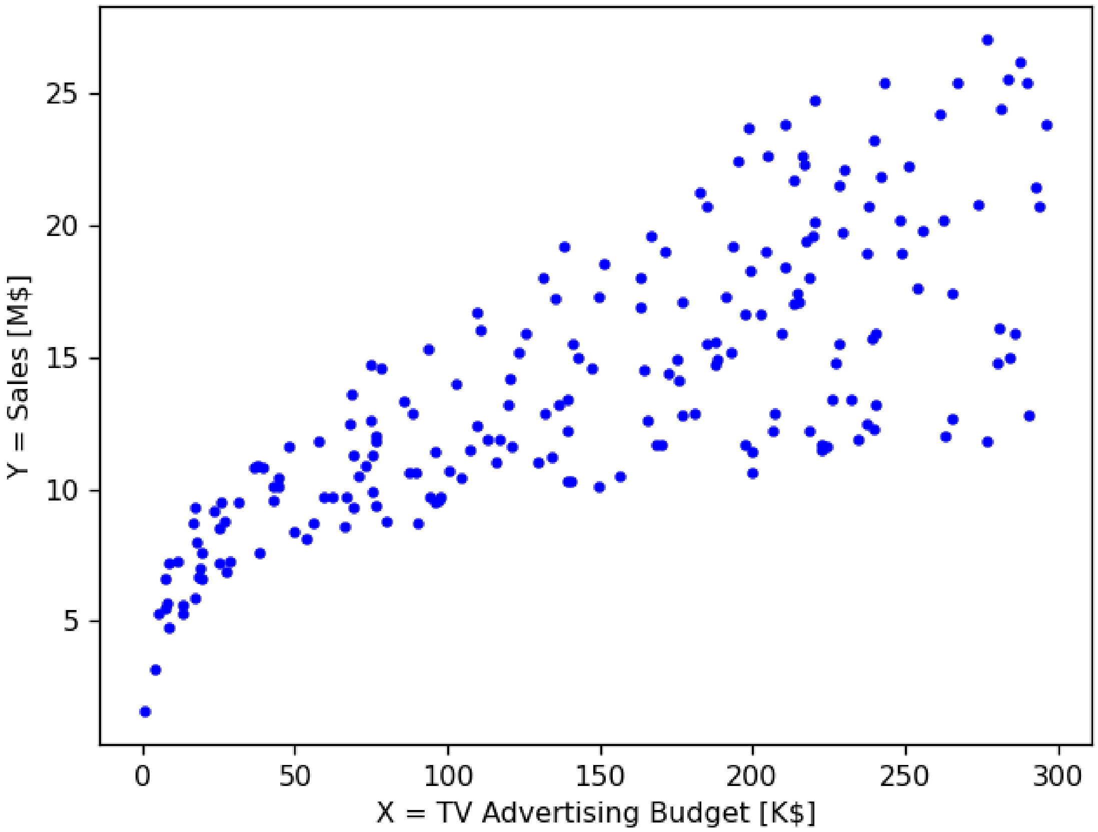
פרנסיס עבוד

Prepared for: Dr. Amer Adler

Date: 08/04/2025

יש לכתוב קוד פייתון למימוש רגרסיה לינארית עבור דוגמת ה- ADVERTISING

```python
# -*- coding: utf-8 -*-
"""
Created on 08/04/2025

@author: Francis Aboud
"""

import numpy as np
import matplotlib.pyplot as plt
from numpy import genfromtxt

# Load data
my_data = genfromtxt('advertising.csv', delimiter=',')

# Plot data
X = my_data[1:201, 1:4]
Y = my_data[1:201, 4:5]
plt.plot(X[:, 0], Y, 'b.')
plt.xlabel('X = TV Advertising Budget [K$]')
plt.ylabel('Y = Sales [M$]')
```



Ex.1:

רגרסיה לינארית עם משתנה יחיד (פרסום ב- TV), והחישוב באמצעות חישוב PSEUDO-INVERSE. יש להציג גרף של תוצאת הרגרסיה, יחד עם נקודות סט האימון (מסומנות ב-X), עבור שני החישובים.

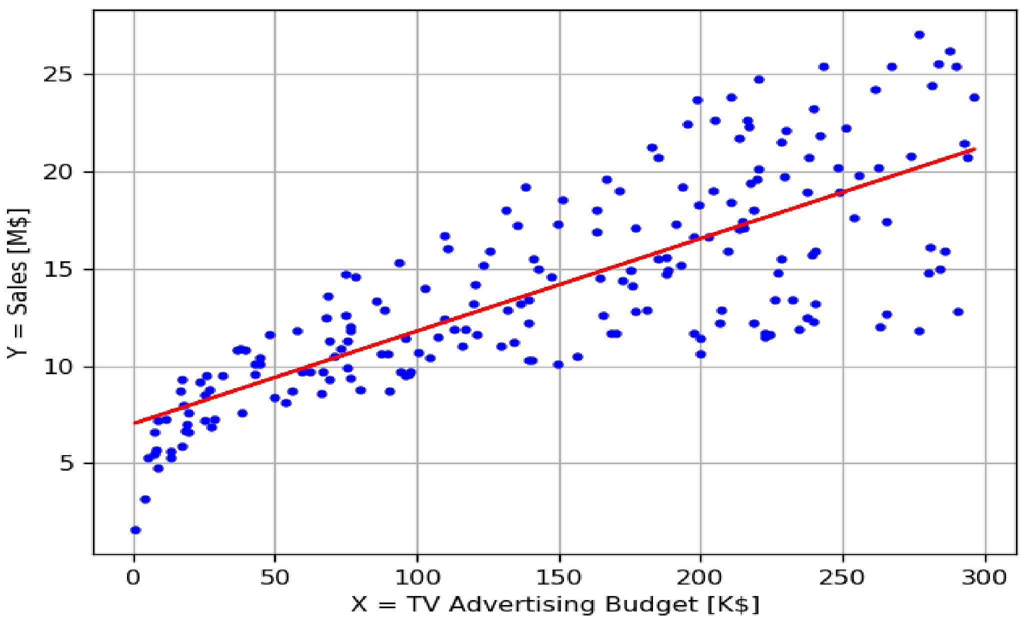
Solution:

Ex1.1: Direct Calculation

```python
"""
--- EX1: Direct Calculation ---
"""
# Calculate means
mean_sales = np.mean(Y)
mean_tv_budget = np.mean(X[:, 0])

# Calculate variance and covariance
variance_tv_budget = np.sum((X[:, 0] - mean_tv_budget) ** 2)
covariance_tv_sales = np.dot((X[:, 0] - mean_tv_budget).T, (Y - mean_sales))

# Calculate regression coefficients
slope_tv_sales = covariance_tv_sales / variance_tv_budget
intercept_tv_sales = mean_sales - slope_tv_sales * mean_tv_budget

# Predicted values
predicted_sales = intercept_tv_sales + slope_tv_sales * X[:, 0]

# Plot regression line
plt.plot(X[:, 0], predicted_sales, 'r-')
plt.grid()
plt.show()

print("Intercept (W0) = ", intercept_tv_sales)
print("Slope (W1) = ", slope_tv_sales)
```

Output:

```
Intercept (W0) =  [7.03259355]
Slope (W1) =  [0.04753664]
```



Ex1.2: Pseudo-Inverse (Single Feature)

```python
"""
--- EX1: Pseudo-Inverse (Single Feature) ---
"""
# Add bias term to feature matrix
tv_feature_matrix = np.column_stack((np.ones(X.shape[0]), X[:, 0]))

# Calculate weights using pseudo-inverse
weights_single_feature = np.matmul(np.linalg.pinv(tv_feature_matrix), Y)

# Predicted values
predicted_sales_pseudo = np.matmul(tv_feature_matrix, weights_single_feature)

# Plot regression line
plt.plot(X[:, 0], predicted_sales_pseudo, 'g-')

print("Pseudo-Inverse (Single Feature):")
print("Weights = ", weights_single_feature)
```

Output:

```
Pseudo-Inverse (Single Feature):
Weights =  [[7.03259355]
 [0.04753664]]
```

Ex.2:

רגרסיה ליניארית עם שלושת המשתנים, והחישוב באמצעות PSEUDO-INVERSE וגם באמצעות GRADIENT DESCENT. הראו שהתקבלו תוצאות כמעט זהות בשני אופני החישוב, עבור מקדמי הרגרסיה.

Solution:

Ex2.1: Pseudo-Inverse (All Features)

```python
"""
--- EX2: Pseudo-Inverse (All Features) ---
"""
# Add bias term to feature matrix
all_features_matrix = np.column_stack((np.ones(X.shape[0]), X))

# Calculate weights using pseudo-inverse
weights_all_features = np.matmul(np.linalg.pinv(all_features_matrix), Y)

print("Pseudo-Inverse (All Features):")
print("Weights = ", weights_all_features)
```

Output: Gradient Descent (All Features)

```
Pseudo-Inverse (All Features):
Weights =  [[ 2.93888937e+00]
 [ 4.57646455e-02]
 [ 1.88530017e-01]
 [-1.03749304e-03]]
```

Ex2.2:

```python
"""
--- EX2: Gradient Descent (All Features) ---
"""
# Initialize parameters
learning_rate = 2e-7
weights_gradient_descent = np.random.rand(4, 1)  # Random initialization for weights

# Gradient descent loop
for iteration in range(1, 1000):
    gradient = np.matmul(all_features_matrix.T, np.matmul(all_features_matrix, weights_gradient_descent) - Y)
    weights_gradient_descent -= learning_rate * gradient

print("Gradient Descent:")
print("Optimal Weights = ", weights_gradient_descent)
```

Output:

```
Gradient Descent:
Optimal Weights =  [[0.53477881]
 [0.05233454]
 [0.21597003]
 [0.01364216]]
```

Ex3:

השוו בין תוצאת ה- MSE של סעיף 1 ו-2.

Solution:

```python
"""
--- MSE Comparison Between EX1 and EX2 ---
"""
# Predicted values for EX2 (All Features)
predicted_sales_all_features = np.matmul(all_features_matrix, weights_all_features)

# Calculate MSE for EX1: Pseudo-Inverse (Single Feature)
mse_single_feature = np.mean((Y - predicted_sales_pseudo) ** 2)

# Calculate MSE for EX2: Pseudo-Inverse (All Features)
mse_all_features = np.mean((Y - predicted_sales_all_features) ** 2)

# Compare MSEs
print("\nMean Squared Error (MSE) Comparison:")
print(f"EX1: Pseudo-Inverse (Single Feature) MSE: {mse_single_feature}")
print(f"EX2: Pseudo-Inverse (All Features) MSE: {mse_all_features}")

if mse_single_feature < mse_all_features:
    print("EX1: Pseudo-Inverse (Single Feature) has a lower MSE.")
elif mse_single_feature > mse_all_features:
    print("EX2: Pseudo-Inverse (All Features) has a lower MSE.")
else:
    print("Both methods have the same MSE.")
```

Output:

```
Mean Squared Error (MSE) Comparison:
EX1: Pseudo-Inverse (Single Feature) MSE: 10.512652915656757
EX2: Pseudo-Inverse (All Features) MSE: 2.784126314510936
EX2: Pseudo-Inverse (All Features) has a lower MSE.
```

בהשוואת שגיאת הריבועים הממוצעת (MSE) בין שתי השיטות, התקבלו התוצאות הבאות:

- פסאודו-אינברס (משתנה יחיד): MSE = 10.512652915656757
- פסאודו-אינברס (כל המשתנים): MSE = 2.784126314510936

מסקנה: לשיטה EX2: פסאודו-אינברס (כל המשתנים) יש שגיאה ממוצעת נמוכה יותר.