

Yel Lin ID 6520242

Assignment 6: Gradient Descent for Linear Regression

Question 1: Linear Regression Model ug Maximum Likelihood Estimation

Step 1.1 : Plotting Training and Testing Data

- Plotting the relationship between the training data and testing data on the same graph

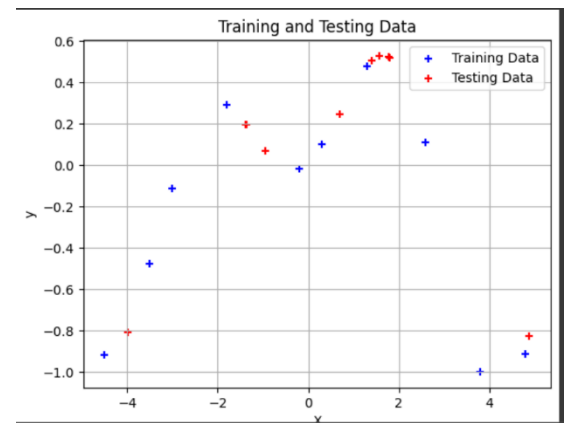
```
!pip install numpy
X_train = np.array([-1.5, -1.2, -1.0, -0.8, -0.5, -0.3, 0.1, 0.4, 0.7, 1.0, 1.3, 1.6, 1.9, 2.2, 2.5, 2.8, 3.1, 3.4, 3.7, 4.0]).reshape(-1,1)
y_train = np.array([0.04286391, 0.5962279, 0.9973792, 1.3613361, 1.6295937, 1.8647683, 2.0599184, 2.2187586, 2.3465532, 2.4394145, 2.502721, 2.5321871, 2.5250075, 2.4804254, 2.4005864, 2.2835931, 2.137907, 1.9720324, 1.7845338, 1.5740154]).reshape(-1,1)

X_test = np.array([0.85, 0.98, 1.12, 1.26, 1.4, 1.5, 1.58, 1.67, 1.74, 1.81]).reshape(-1,1)
y_test = np.array([0.9973792, 1.3613361, 1.6295937, 1.8647683, 2.0599184, 2.2187586, 2.3465532, 2.4394145, 2.502721, 2.5321871]).reshape(-1,1)

# Training Training Data (X_train, y_train)
plt.figure()
plt.scatter(X_train, y_train, marker='x', label='Training Data')

# Training Testing Data (X_test, y_test)
plt.scatter(X_test, y_test, marker='x', label='Testing Data')

plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Training and Testing Data')
plt.show()
```



1.2 Polynomial Feature Transformation

```
#step 1.2
def poly_features(X, K):
    # return np.hstack([X**i for i in range(K + 1)])
    X= X.flatten()
    N= X.shape[0]
    Phi= np.zeros((N,K+1))
    for k in range (K+1):
        Phi[:,k]= X**k
    return Phi
```

The purpose and the effect of this transformation:

Polynomial transformation enables linear models to fit nonlinear data by adding higher-degree features. It improves flexibility but risks underfitting with low degrees or overfitting with high degrees. The right degree balances accuracy and complexity for better generalization.

1.3 Fitting the Model Using Maximum Likelihood

```
Assignment 6.ipynb - Colab
colab.research.google.com/drive/1QAUGP32YsrharmXkNnHFLF3hsQuD_9#scrollTo=ExGgpx1YBUz

Assignment 6.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
In [4]:
# Step 1.1
# Polynomial degree
K = 5

# Transform both training and testing data
Phi_train = poly_features(X_train, K)
Phi_test = poly_features(X_test, K)

# Compute MLE coefficients: theta_MLE = (Phi^T Phi)^(-1) Phi^T y
theta_mle = np.linalg.inv(Phi_train.T @ Phi_train) @ Phi_train.T @ y
print("theta_mle:\n", theta_mle)

# Generate a range of x values from -8 to 8 for plotting the model's predictions
x_plot = np.linspace(-8, 8, 100).reshape(-1, 1)
Phi_plot = poly_features(x_plot, K)
y_plot = Phi_plot @ theta_mle

# Plotting
plt.figure(figsize=(8,6))
plt.scatter(X_train, y_train, color='blue', marker='x', label='Training Data')
plt.scatter(X_test, y_test, color='red', marker='x', label='Testing Data')

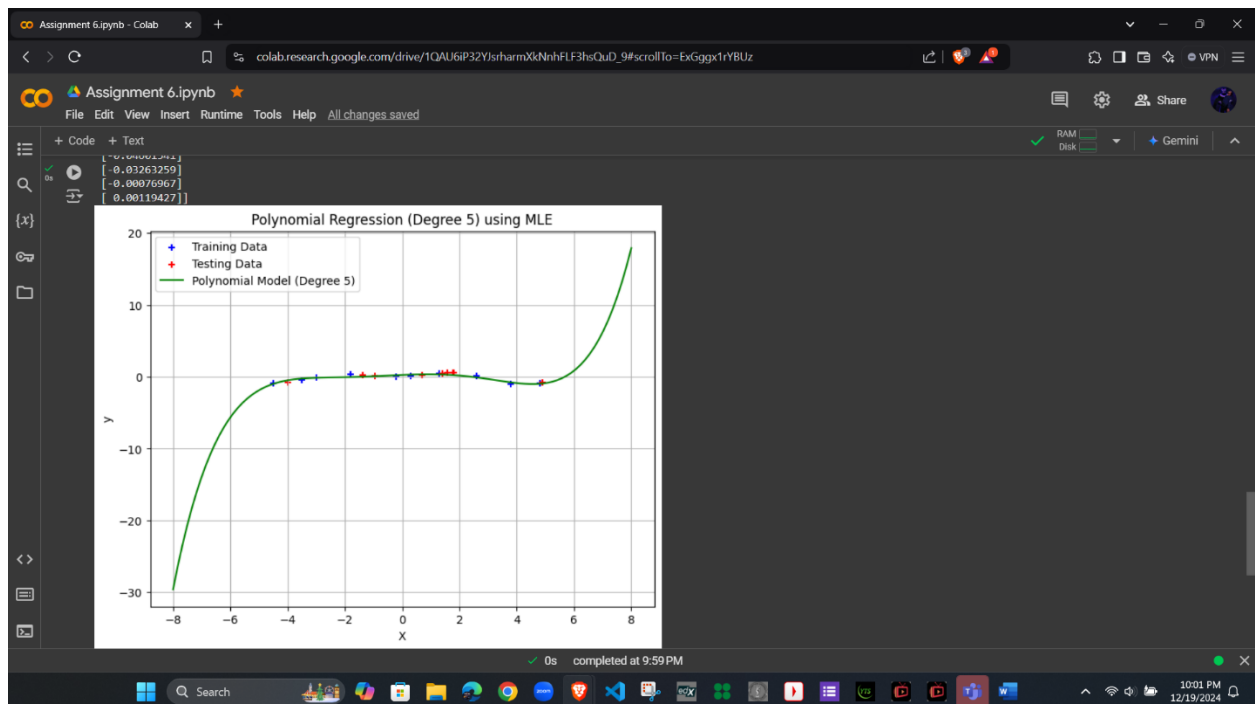
# Plot the polynomial fit
plt.plot(x_plot, y_plot, color='green', label='Polynomial Model (Degree 5)')

plt.title('Polynomial Regression (Degree 5) using MLE')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()

0s completed at 9:59 PM
```

theta_mle:

```
[[ 0.24879002]
 [ 0.16821157]
 [-0.04601541]
 [-0.03263259]
 [-0.00076967]
 [ 0.00119427]]
```



1.4 Model Evaluation

```
Assignment 6.ipynb - Colab
colab.research.google.com/drive/1QAU6IP32YjsrhamXkNnhfUF3hsQuD_9#scrollTo=m7FnVaS7hH-8

Assignment 6.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

In 1:
def rmse(y_true, y_pred):
    return np.sqrt(np.mean((y_true - y_pred)**2))

max_degree = 16
train_rmse_values = []
test_rmse_values = []

for d in range(max_degree):
    # Generate polynomial features for both training and testing sets
    Phi_train = poly_features(X, d)
    Phi_test = poly_features(X_test, d)

    # Compute theta_MLS = (Phi^T Phi)^(-1) Phi^T y
    # Handle the case where Phi is just a column of ones
    theta_MLS = np.linalg.lstsq(Phi_train.T @ Phi_train, Phi_train.T @ y)

    # Predictions
    y_pred_train = Phi_train @ theta_MLS
    y_pred_test = Phi_test @ theta_MLS

    # Compute RMSE for train and test
    train_rmse = rmse(y, y_pred_train)
    test_rmse = rmse(y_test, y_pred_test)

    train_rmse_values.append(train_rmse)
    test_rmse_values.append(test_rmse)

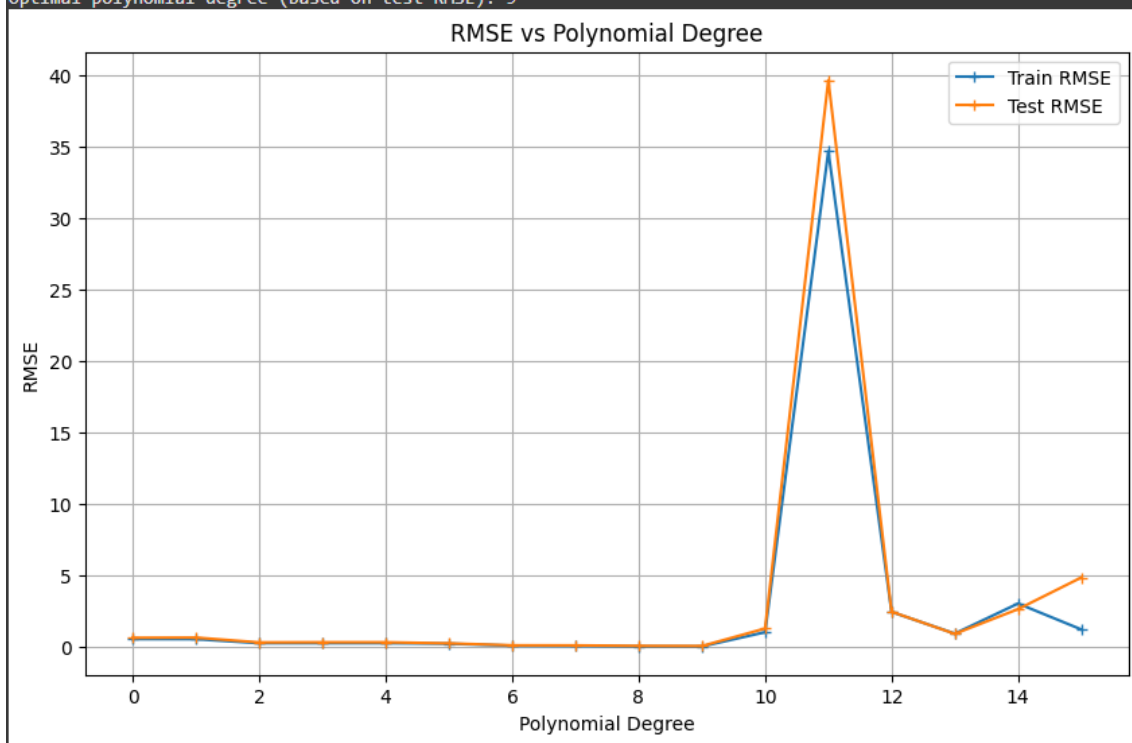
# Determine optimal polynomial degree based on minimum test RMSE
optimal_degree = np.argmin(test_rmse_values)

print("Train RMSE values:", train_rmse_values)
print("Test RMSE values:", test_rmse_values)
print("Optimal polynomial degree (based on test RMSE): (optimal_degree)")

# Plotting the RMSE for training and testing sets
plt.figure(figsize=(10,6))
plt.plot(range(max_degree), train_rmse_values, marker='x', label='Train RMSE')
plt.plot(range(max_degree), test_rmse_values, marker='x', label='Test RMSE')
plt.xlabel('Polynomial Degree')
plt.ylabel('RMSE')

Out 1:
completed at 10:28 PM
```

Train RMSE values: [0.5147699543846374, 0.5117157870391207, 0.22972426509507113, 0.22879771630326973, 0.228795
Test RMSE values: [0.6099911018527294, 0.6216251909396072, 0.28492811777747346, 0.29486416063656634, 0.2949815
Optimal polynomial degree (based on test RMSE): 9



Train RMSE values: [0.5147699543846374,
0.5117157870391207, 0.22972426509507113,
0.22879771630326973, 0.22879573735413525,
0.19502327062655744, 0.05342435077718177,
0.03818517735253735, 0.003910516742283594,
5.538327228678245e-10, 0.999183995488885,
34.68901085509442, 2.4195160890609984,
0.9029673413927789, 3.0081770554024114,
1.1934142035010373]

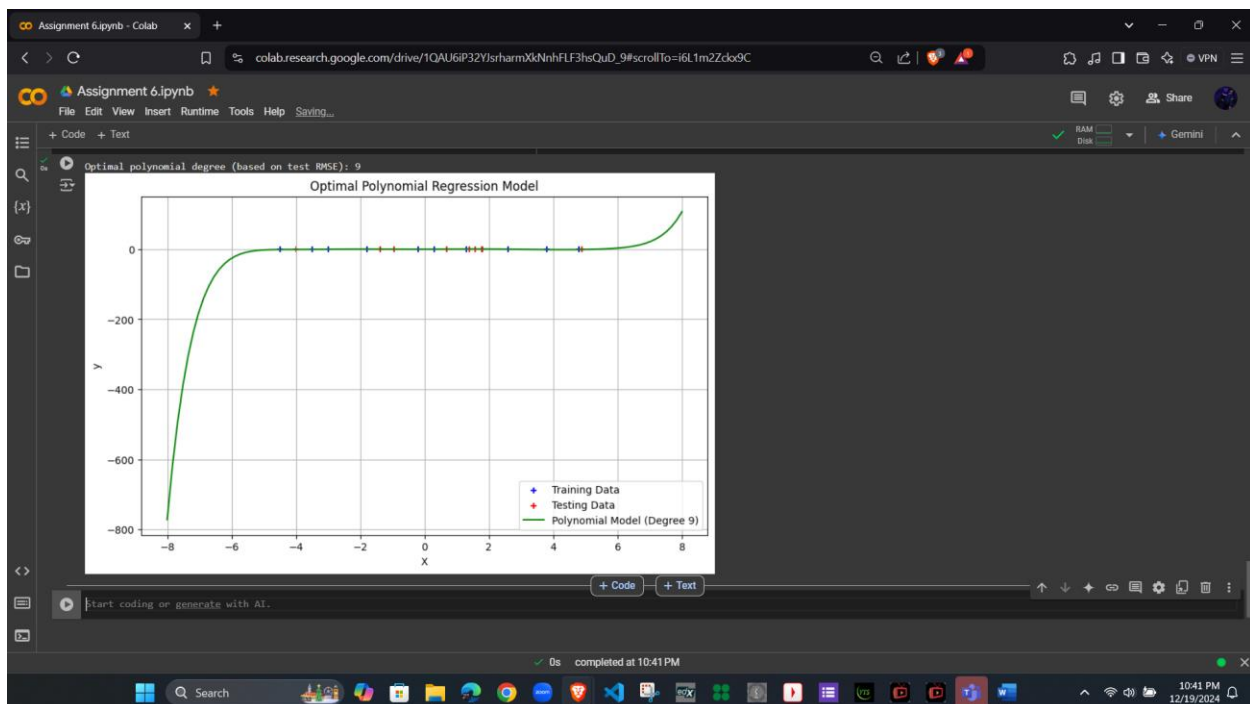
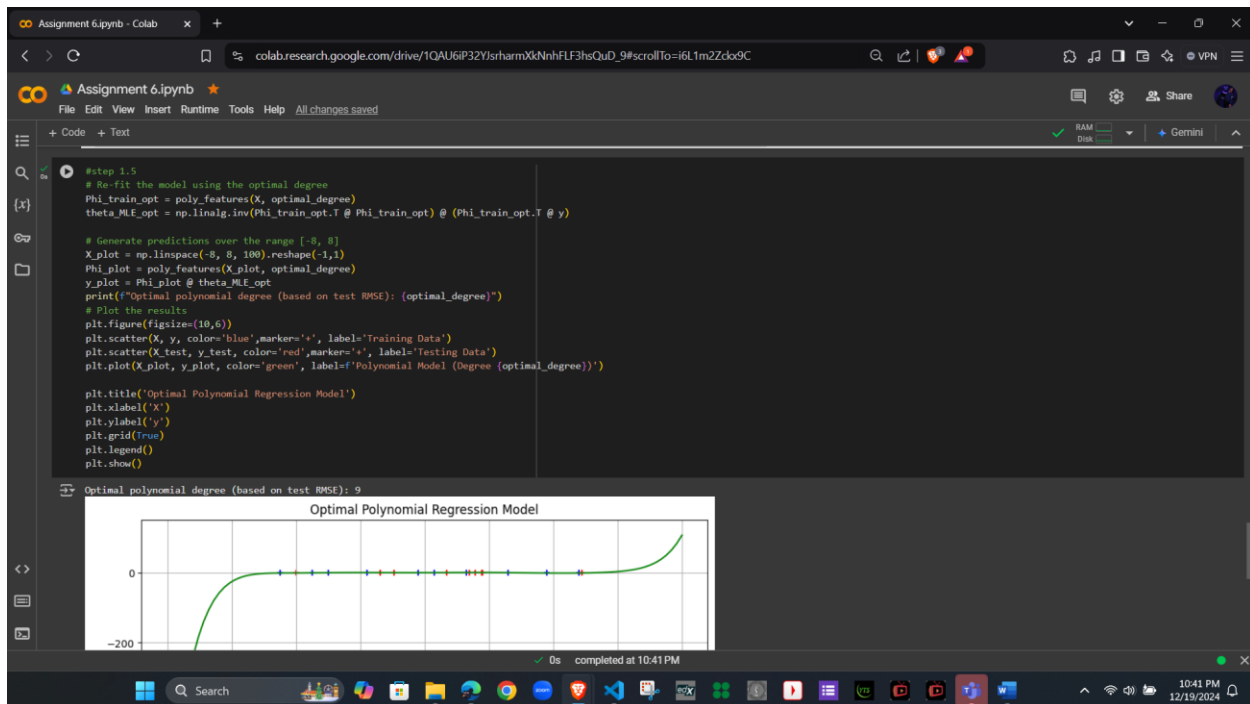
Final Train RMSE values: 1.1934142035010373

Test RMSE values: [0.6099911018527294,
0.6216251909396072, 0.28492811777747346,
0.29486416063656634, 0.29498152071402095,
0.21742862799598514, 0.08213402882291138,
0.07763539925156139, 0.03924484661505437,
0.030724661923518273, 1.2579162752970154,
39.66507331006011, 2.435318758318806,
0.8844692052280537, 2.61375681037945, 4.830158928594374]

Final Test RMSE values: 4.830158928594374

Optimal polynomial degree (based on test RMSE): 9

1.5 Selecting the Best Model



Optimal polynomial degree (based on test RMSE): 9

Question 2: Gradient Descent for Polynomial Regression

2.1 Mathematical Derivation

$$J(\theta) = (y - \Phi^T \theta)^2$$

Step 1: Introduce a temporary variable

$$h = \Phi^T \theta$$

$$J(\theta) = (y - h)^2$$

Step 2: Differentiate with respect to h

$$dJ/dh = 2(y - h)(-1) = -2(y - h) = 2(h - y)$$

Step 3: Differentiate h with respect to θ

$$h = \Phi^T \theta = \sum_i \phi_i \theta_i$$

$$\partial h / \partial \theta = \Phi$$

Step 4: Apply the chain rule

$$\nabla_{\theta} J(\theta) = dJ/dh \cdot \partial h / \partial \theta$$

$$dJ/dh = 2(h - y)$$

$$\partial h / \partial \theta = \Phi$$

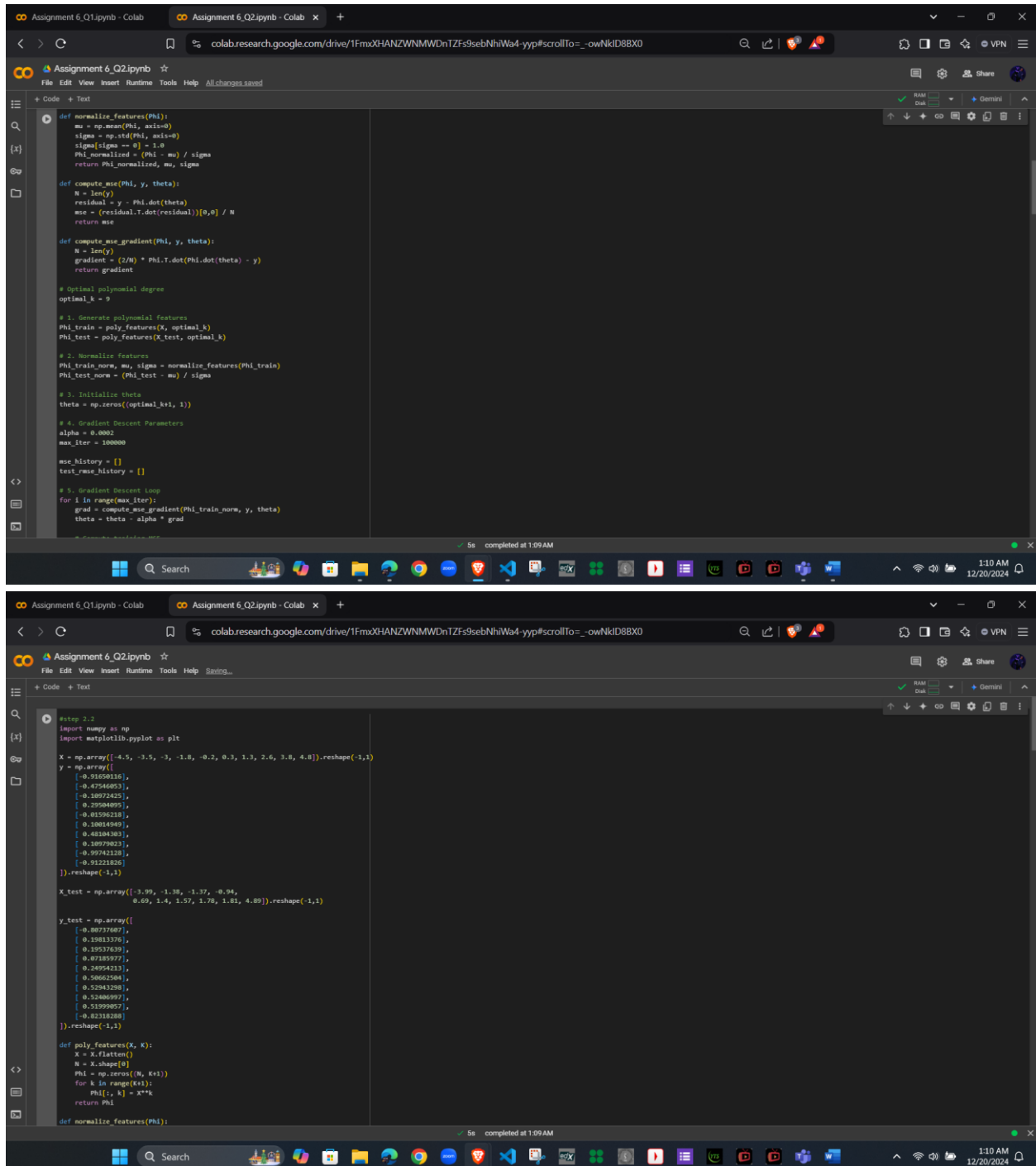
$$\text{Thus, } \nabla_{\theta} J(\theta) = 2(h - y)\Phi$$

Step 5: Substitute $h = \Phi^T \theta$

$$\nabla_{\theta} J(\theta) = 2(\Phi^T \theta - y)\Phi$$

$$\text{Final answer: } \nabla_{\theta} J(\theta) = 2(\Phi^T \theta - y)\Phi$$

2.2 Feature Transformation and Gradient Descent Implementation



The image displays two screenshots of a Google Colab notebook titled "Assignment 6_Q2.ipynb". The notebook is open in a web browser, showing the code editor and the output area.

Top Screenshot: The code defines functions for feature normalization and MSE computation, and implements the gradient descent algorithm.

```
def normalize_features(Phi):
    mu = np.mean(Phi, axis=0)
    sigma = np.std(Phi, axis=0)
    sigma[sigma == 0] = 1.0
    Phi_normalized = (Phi - mu) / sigma
    return Phi_normalized, mu, sigma

def compute_mse(Phi, y, theta):
    N = len(y)
    residual = y - Phi.dot(theta)
    mse = (residual.T.dot(residual))[0,0] / N
    return mse

def compute_mse_gradient(Phi, y, theta):
    N = len(y)
    gradient = (2/N) * Phi.T.dot(Phi.dot(theta) - y)
    return gradient

# Optimal polynomial degree
optimal_k = 9

# 1. Generate polynomial features
Phi_train = poly_features(X, optimal_k)
Phi_test = poly_features(X_test, optimal_k)

# 2. Normalize features
Phi_train_norm, mu, sigma = normalize_features(Phi_train)
Phi_test_norm = (Phi_test - mu) / sigma

# 3. Initialize theta
theta = np.zeros((optimal_k+1, 1))

# 4. Gradient Descent Parameters
alpha = 0.0002
max_iter = 100000

mse_history = []
test_mse_history = []

# 5. Gradient Descent loop
for i in range(max_iter):
    grad = compute_mse_gradient(Phi_train_norm, y, theta)
    theta = theta - alpha * grad
```

Bottom Screenshot: The code defines the polynomial features function and the data sets.

```
def poly_features(X, K):
    X = X.flatten()
    N = X.shape[0]
    Phi = np.zeros((N, K+1))
    for k in range(K+1):
        Phi[:, k] = X**k
    return Phi

def normalize_features(Phi):
    mu = np.mean(Phi, axis=0)
    sigma = np.std(Phi, axis=0)
    sigma[sigma == 0] = 1.0
    Phi_normalized = (Phi - mu) / sigma
    return Phi_normalized, mu, sigma

# Step 2.2
import numpy as np
import matplotlib.pyplot as plt

X = np.array([-4.5, -3.5, -3, -1.8, -0.2, 0.3, 1.3, 2.6, 3.8, 4.8]).reshape(-1,1)
y = np.array([
    -0.91650116,
    -0.47546053,
    -0.18972425,
    0.29504009,
    -0.01596218,
    0.10014949,
    0.48384303,
    0.18879623,
    -0.99742128,
    -0.91221826
]).reshape(-1,1)

X_test = np.array([-3.99, -1.38, -1.37, -0.94,
    0.69, 1.4, 1.57, 1.78, 1.81, 4.89]).reshape(-1,1)

y_test = np.array([
    -0.80737607,
    0.19013176,
    0.10537439,
    0.07185977,
    0.24954233,
    0.50622041,
    0.52943298,
    0.52880997,
    0.51990057,
    -0.82131808
]).reshape(-1,1)

def poly_features(X, K):
    X = X.flatten()
    N = X.shape[0]
    Phi = np.zeros((N, K+1))
    for k in range(K+1):
        Phi[:, k] = X**k
    return Phi

def normalize_features(Phi):
    mu = np.mean(Phi, axis=0)
    sigma = np.std(Phi, axis=0)
    sigma[sigma == 0] = 1.0
    Phi_normalized = (Phi - mu) / sigma
    return Phi_normalized, mu, sigma
```


Assignment 6_Q1.ipynb - Colab Assignment 6_Q2.ipynb - Colab +

colab.research.google.com/drive/1FmxXHANZWNMWDnTZfs9sebNhiWa4-yp#scrollTo=-owNkID8BX0

Assignment 6_Q2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
# 5. Gradient Descent Loop
for i in range(max_iter):
    grad = compute_mse_gradient(Phi_train_norm, y, theta)
    theta = theta - alpha * grad

# Compute training MSE
train_mse = compute_mse(Phi_train_norm, y, theta)
mse_history.append(train_mse)

# Compute test RMSE to track progress
test_mse = compute_mse(Phi_test_norm, y_test, theta)
test_rmse = np.sqrt(test_mse)
test_rmse_history.append(test_rmse)

# Optional dynamic adjustment of learning rate
# Every 10,000 iterations, check if improvement is negligible
if i % 10000 == 0:
    if (mse_history[-10000] - train_mse) < 1e-8:
        alpha = alpha * 0.5

print("Final training MSE:", mse_history[-1])
print("Final test RMSE:", test_rmse_history[-1])

plt.figure(figsize=(10,6))
plt.plot(mse_history)
plt.title("Training MSE over iterations")
plt.xlabel("Iteration")
plt.ylabel("MSE")
plt.grid(True)
plt.show()

plt.figure(figsize=(10,6))
plt.plot(test_rmse_history)
plt.title("Test RMSE over iterations")
plt.xlabel("Iteration")
plt.ylabel("RMSE")
plt.grid(True)
plt.show()
```

Final training MSE: 0.09141959694267199
Final test RMSE: 0.24725352784834527

Training MSE over iterations

completed at 1:09 AM

Assignment 6_Q1.ipynb - Colab Assignment 6_Q2.ipynb - Colab +

colab.research.google.com/drive/1FmxXHANZWNMWDnTZfs9sebNhiWa4-yp#scrollTo=-owNkID8BX0

Assignment 6_Q2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
plt.grid(True)
plt.show()

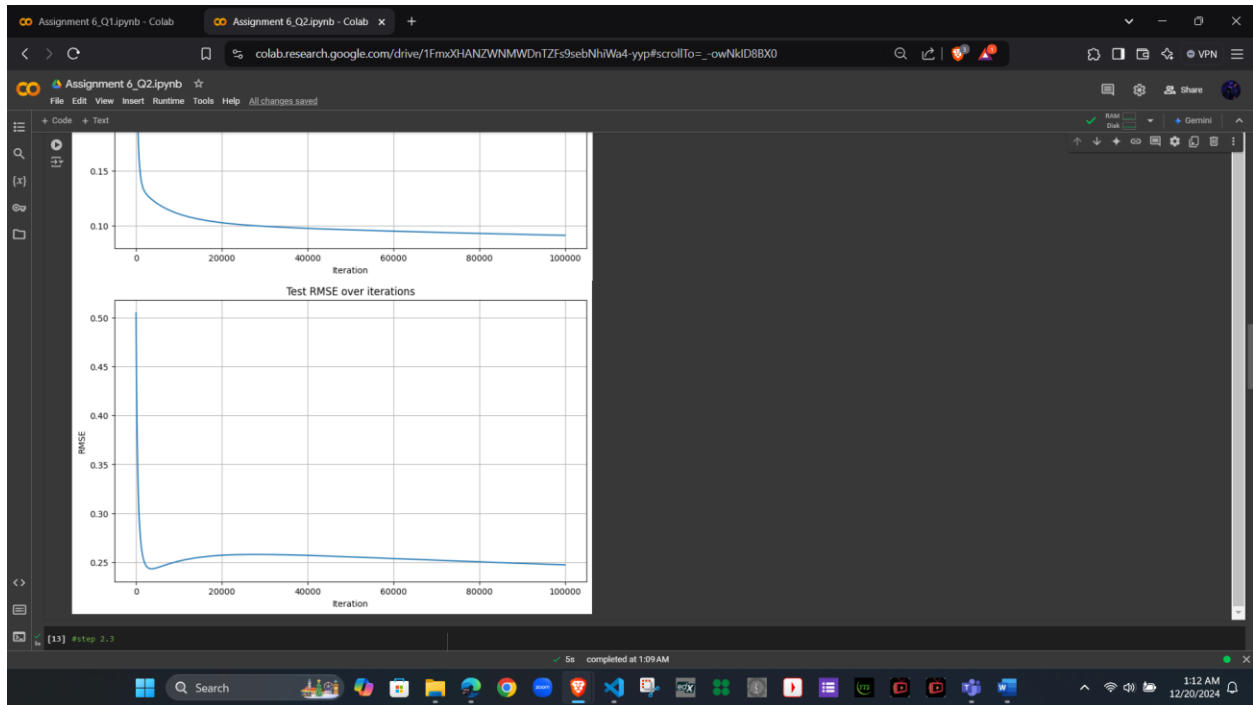
plt.figure(figsize=(10,6))
plt.plot(test_rmse_history)
plt.title("Test RMSE over iterations")
plt.xlabel("Iteration")
plt.ylabel("RMSE")
plt.grid(True)
plt.show()
```

Final training MSE: 0.09141959694267199
Final test RMSE: 0.24725352784834527

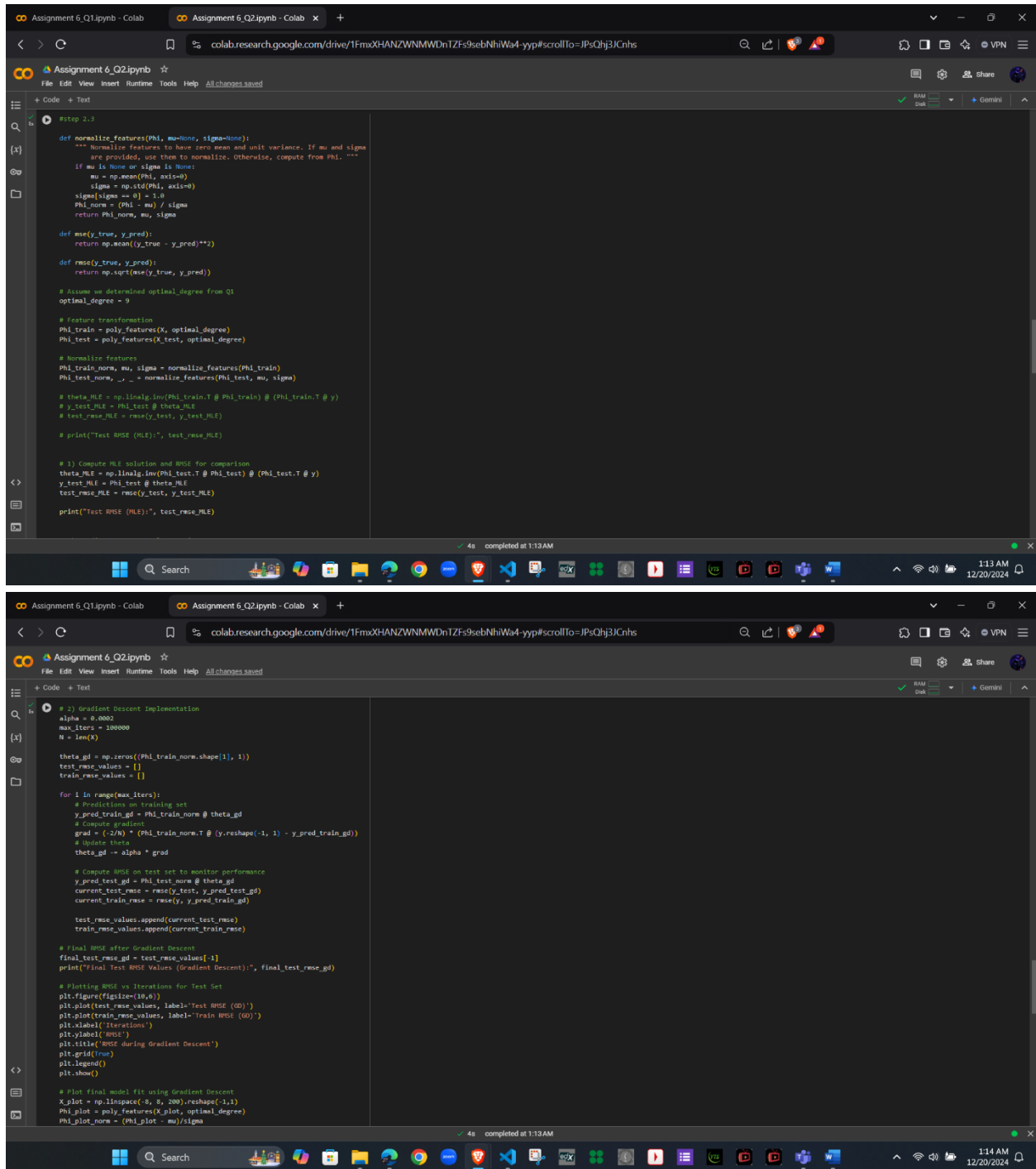
Training MSE over iterations

Test RMSE over iterations

completed at 1:09 AM



2.3 Model Evaluation and Comparison



The image displays two screenshots of a Google Colab notebook, showing the implementation of a polynomial regression model and its evaluation using Gradient Descent.

Top Screenshot: Step 2.1 - Feature Normalization and MSE/RMSE Calculation

```
# Step 2.1

def normalize_features(Phi, mu=None, sigma=None):
    """ Normalize features to have zero mean and unit variance. If mu and sigma
    are provided, use them to normalize. Otherwise, compute from Phi. """
    if mu is None or sigma is None:
        mu = np.mean(Phi, axis=0)
        sigma = np.std(Phi, axis=0)
        sigma[sigma == 0] = 1.0
        Phi_norm = (Phi - mu) / sigma
        return Phi_norm, mu, sigma

def mse(y_true, y_pred):
    return np.mean((y_true - y_pred)**2)

def rmse(y_true, y_pred):
    return np.sqrt(mse(y_true, y_pred))

# Assume we determined optimal_degree from Q1
optimal_degree = 9

# Feature transformation
Phi_train = poly_features(X, optimal_degree)
Phi_test = poly_features(X_test, optimal_degree)

# Normalize features
Phi_train_norm, mu, sigma = normalize_features(Phi_train)
Phi_test_norm, _ = normalize_features(Phi_test, mu, sigma)

# theta_MLE = np.linalg.inv(Phi_train.T @ Phi_train) @ (Phi_train.T @ y)
# y_test_MLE = Phi_test @ theta_MLE
# test_rmse_MLE = rmse(y_test, y_test_MLE)
# print("Test RMSE (MLE):", test_rmse_MLE)

# 1) Compute MLE solution and RMSE for comparison
theta_MLE = np.linalg.inv(Phi_train.T @ Phi_train) @ (Phi_train.T @ y)
y_test_MLE = Phi_test @ theta_MLE
test_rmse_MLE = rmse(y_test, y_test_MLE)
print("Test RMSE (MLE):", test_rmse_MLE)
```

Bottom Screenshot: Step 2.2 - Gradient Descent Implementation

```
# 2) Gradient Descent Implementation
alpha = 0.0001
max_iters = 100000
N = len(X)

theta_gd = np.zeros((Phi_train_norm.shape[1], 1))
test_rmse_values = []
train_rmse_values = []

for i in range(max_iters):
    # Predictions on training set
    y_pred_train_gd = Phi_train_norm @ theta_gd
    # Compute gradients
    grad = (-2/N) * (Phi_train_norm.T @ (y.reshape(-1, 1) - y_pred_train_gd))
    # Update theta
    theta_gd += alpha * grad

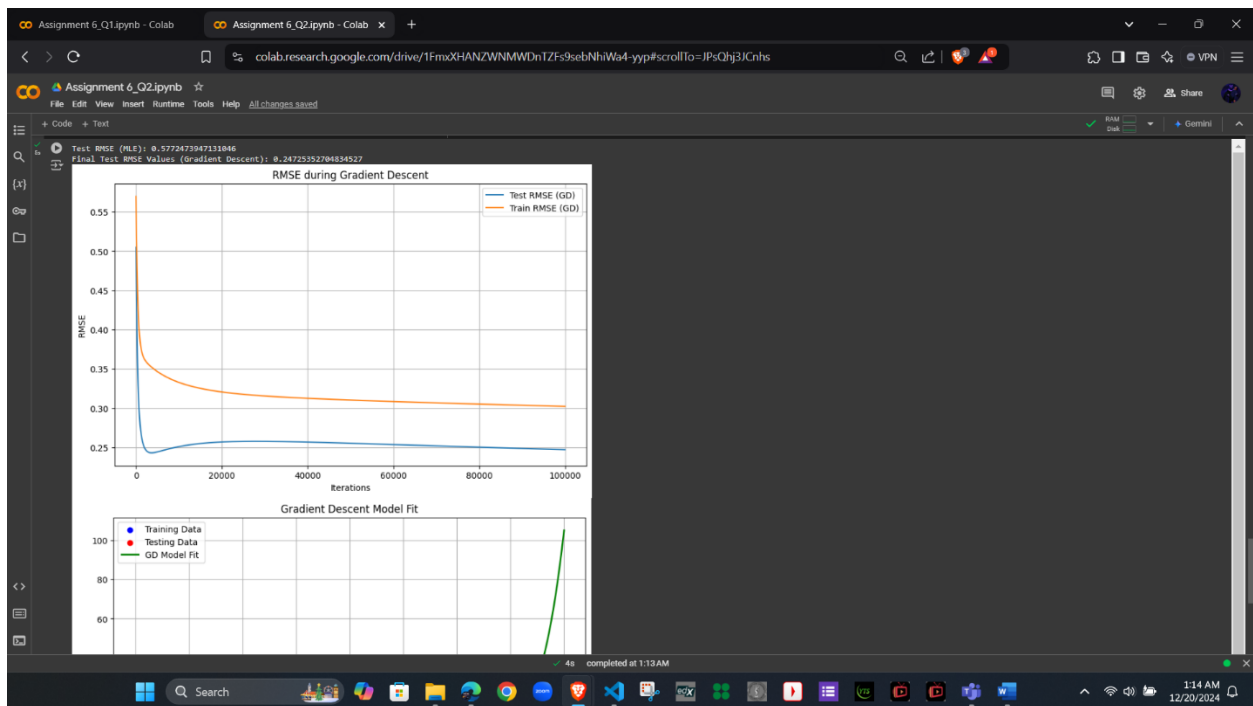
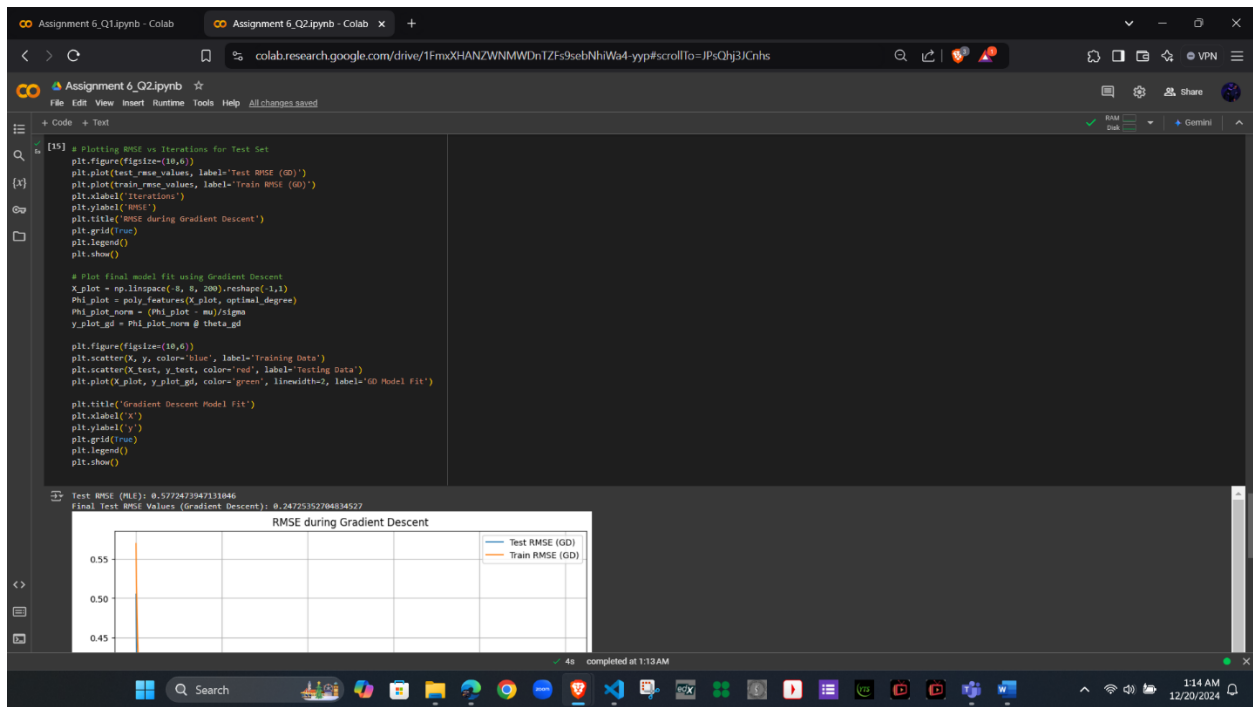
    # Compute RMSE on test set to monitor performance
    y_pred_test_gd = Phi_test_norm @ theta_gd
    current_test_rmse = rmse(y_test, y_pred_test_gd)
    current_train_rmse = rmse(y, y_pred_train_gd)

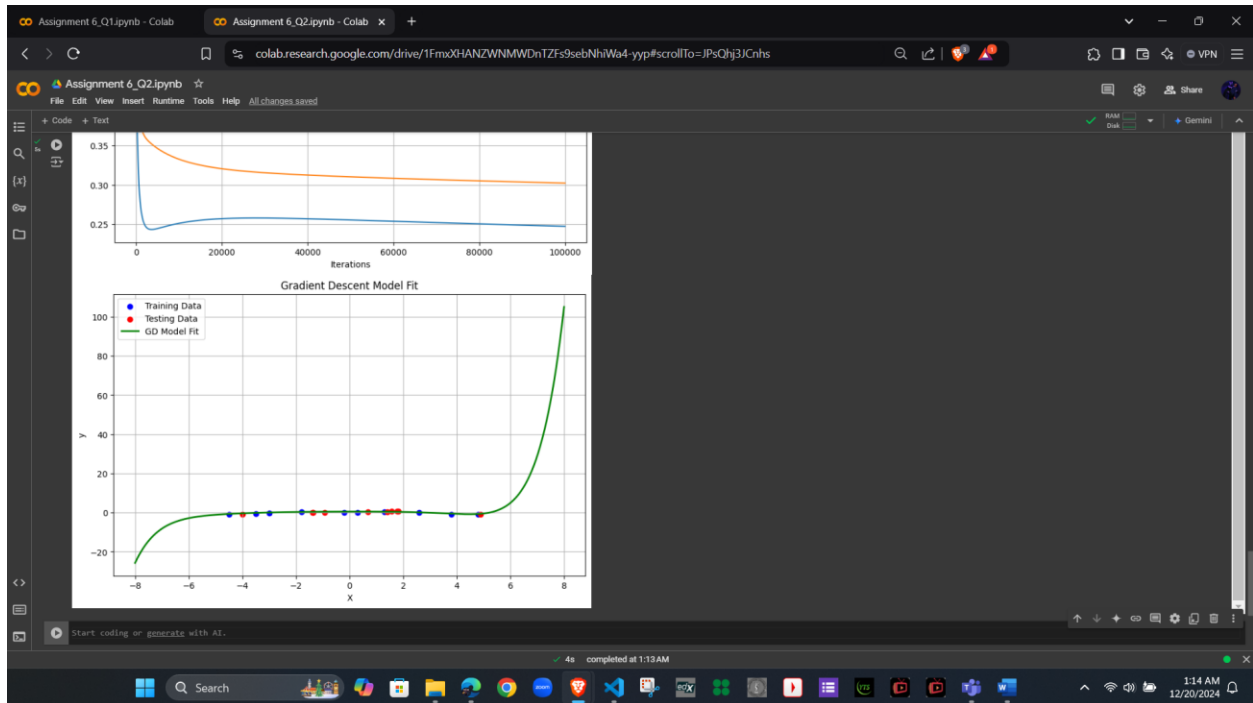
    test_rmse_values.append(current_test_rmse)
    train_rmse_values.append(current_train_rmse)

# Final RMSE after Gradient Descent
final_test_rmse_gd = test_rmse_values[-1]
print("Final Test RMSE Values (Gradient Descent):", final_test_rmse_gd)

# Plotting RMSE vs Iterations for Test Set
plt.figure(figsize=(10,6))
plt.plot(test_rmse_values, label="Test RMSE (GD)")
plt.plot(train_rmse_values, label="Train RMSE (GD)")
plt.xlabel('Iterations')
plt.ylabel('RMSE')
plt.title("RMSE during Gradient Descent")
plt.grid(True)
plt.legend()
plt.show()

# Plot final model fit using Gradient Descent
X_plot = np.linspace(-8, 8, 200).reshape(-1,1)
Phi_plot = poly_features(X_plot, optimal_degree)
Phi_plot_norm = (Phi_plot - mu) / sigma
```





Test RMSE (MLE): 0.5772473947131046

**Final Test RMSE Values (Gradient Descent):
0.24725352704834527**

Discussion:

- The training MSE curve should decrease over iterations, indicating successful convergence.
- The final GD solution should produce a test RMSE similar to the MLE solution.
- Gradient Descent convergence speed and final accuracy depend on the learning rate and normalization.
- Normalization and a sufficiently small learning rate (0.0002) help ensure smooth convergence.