```
import numpy as np
import scipy.linalg
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) # 10x1 vector, N=5, D=1
y = np.array([
    [-1.11362822],
    [-1.24394281],
    [-0.91157385],
    [0.67067171],
    [1.24891634],
    [0.7776148],
    [-0.62067303],
    [-1.41641754],
    [-0.30383694],
    [0.92323755]
]).reshape(-1,1) # 10x1 vector
def f(X):
  return np.cos(X) + 0.2*np.random.normal(size=(X.shape))
X_data= np.linspace(-4,4,20).reshape(-1,1)
y_data=f(X_data)

# Define the maximum likelihood estimation function for calculating
theta
def max_lik_estimate(X, y):
    # Calculate the inverse of X transposed times X
    inverse_term = np.linalg.inv(X.T @ X)
    # Multiply the inverse by X transposed, and then by y to compute
theta maximum likelihood
    theta_ml = inverse_term @ (X.T @ y)
    return theta_ml
# Calculate theta using the training data
theta_ml = max_lik_estimate(X, y)
print("Theta:", theta_ml)
```

```
Theta: [[0.10642665]]
```

```
def predict_with_estimate(Xtest, theta):
    # Return predictions as dot product of test inputs and theta
    prediction = Xtest @ theta
    return prediction
Xtest = np.linspace(-5, 5, 100).reshape(-1,1)
ml_prediction = predict_with_estimate(Xtest, theta_ml)

print("="*10)
N, D = X.shape
X_aug = np.concatenate((np.ones((N, 1)), X), axis=1)
print("X_aug = ")
print(X_aug)
```

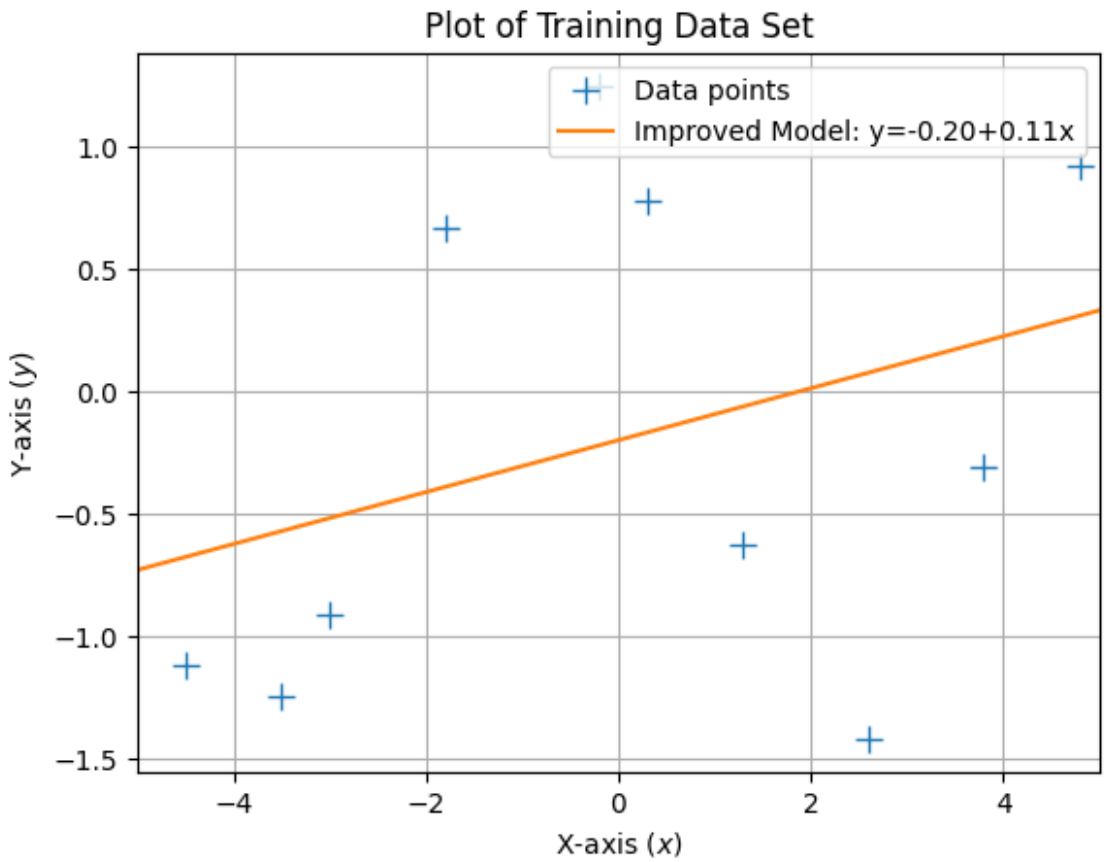
```
print("="*10)
# Recalculation
def max_lik_estimate_with_bias(X_aug, y):
    # Calculate the inverse of X_aug transposed times X_aug
    inverse_term = np.linalg.inv(X_aug.T @ X_aug)
    # Multiply the inverse by X_aug transposed, and then by y to
compute theta maximum likelihood
    theta_ml_bias = inverse_term @ (X_aug.T @ y)
    return theta_ml_bias

theta_ml_bias = max_lik_estimate_with_bias(X_aug, y)
print("Theta with bias term (Two Values):", theta_ml_bias)
```

```
==========
X_aug =
[[ 1.  -4.5]
 [ 1.  -3.5]
 [ 1.  -3. ]
 [ 1.  -1.8]
 [ 1.  -0.2]
 [ 1.   0.3]
 [ 1.   1.3]
 [ 1.   2.6]
 [ 1.   3.8]
 [ 1.   4.8]]
==========
Theta with bias term (Two Values): [[-0.19684334]
 [ 0.10599307]]
```

```
Xtest_aug = np.concatenate((np.ones((Xtest.shape[0], 1)), Xtest),
axis=1)
ml_prediction_bias = predict_with_estimate(Xtest_aug, theta_ml_bias)

plt.figure()
plt.plot(X, y, '+', markersize=10, label='Data points')
plt.plot(Xtest, ml_prediction_bias, label=f'Improved Model:
y={theta_ml_bias[0][0]:.2f}+{theta_ml_bias[1][0]:.2f}x')
plt.xlabel("X-axis ($x$)")
plt.ylabel("Y-axis ($y$)")
plt.title("Plot of Training Data Set")
plt.xlim([-5, 5])
plt.legend()
plt.grid(True)
plt.show()
```




```
def nonlinear_features_maximum_likelihood(X, K):
    X=X.flatten()
    n_samples = X.shape[0]
    X_poly = np.zeros((n_samples, K + 1))
    for i in range(K + 1):
        X_poly[:, i] = X**i
    return X_poly


K = 2
X_poly = nonlinear_features_maximum_likelihood(X, K)


theta_ml_poly = max_lik_estimate(X_poly, y)


Xtest_poly = nonlinear_features_maximum_likelihood(Xtest, K)
ml_prediction_poly = predict_with_estimate(Xtest_poly, theta_ml_poly)
Xtest=np.linspace(-6,6,100).reshape(-1,1)

plt.figure()
```

```
plt.plot(X, y, '+', markersize=10, label='Data points')

plt.plot(Xtest, ml_prediction_poly, label=f'Polynomial Model (K {K})')


plt.xlabel("X-axis ($x$)")
plt.ylabel("Y-axis ($y$)")
plt.title("Plot of Training Data with Polynomial Feature Model")
plt.xlim([-5, 5])
plt.legend()
plt.grid(True)
plt.show()
```