ps1

July 25, 2024

CS 7180 MLSA Linear Regression and Gradient Descent Due: July 23rd, 2024 Joshua Nougaret

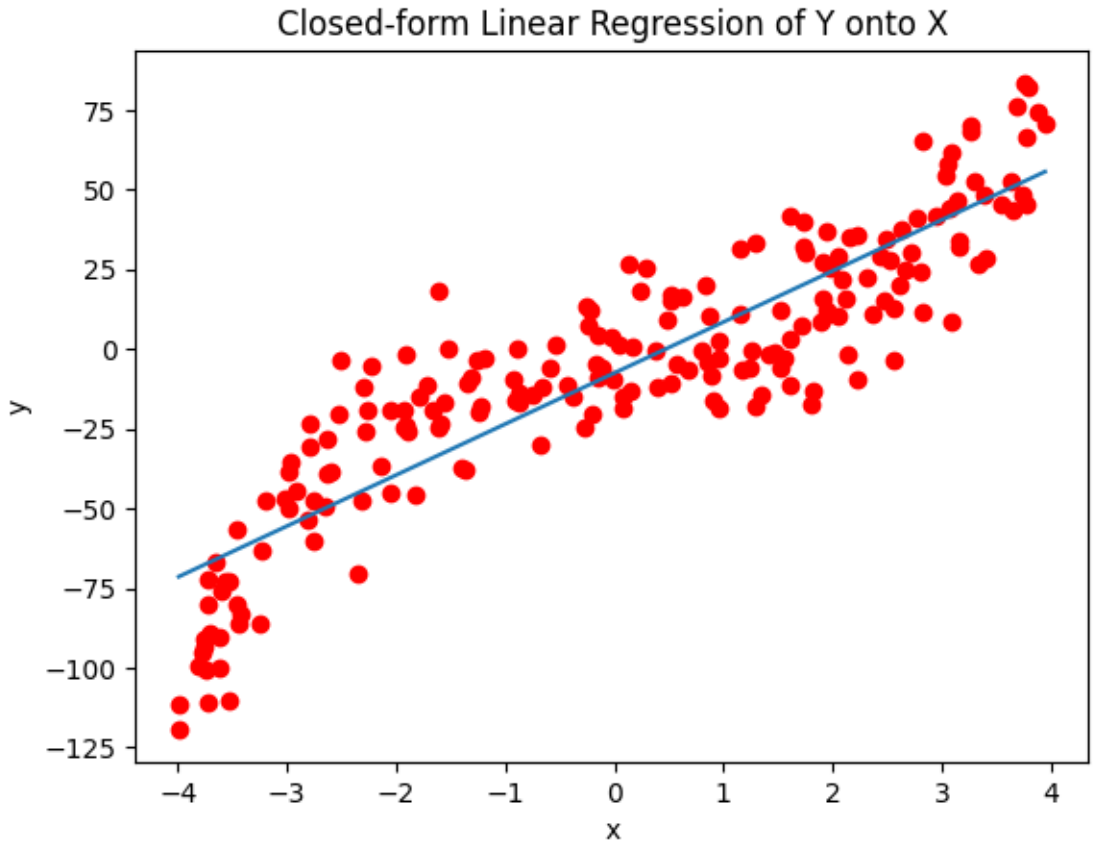
1. Write a python code whose input is a training dataset (x1, y1), ..., (xN , yN) and its output is the weight vector in the model y = T (x) for a non-linear mapping (). Consider mean squared error as loss function.[25 Points] Implement two cases:

- i) closed-form solution, and
- ii) using stochastic gradient descent on mini-batches of size m.

Exploratory solution to practice linear regression (not part of assignment):

[41]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Download the train data and apply closed-form solution on train data
train_path = "HW-2-Data/train_data.csv"
train = pd.read_csv(train_path)

x_train = train["x"]
y_train = train["y"]
mean_x = np.average(x_train)
mean_y = np.average(y_train)

a = 0
b = 0
for i in range(len(x_train)):
    a = a + (x_train[i] - mean_x)*(y_train[i] - mean_y)
    b = b + (x_train[i] - mean_x)**2

beta_1 = a / b
beta_0 = mean_y - beta_1 * mean_x

# with learned weights, plot linear regression line over training data.
x = np.linspace(min(x_train), max(x_train), len(x_train))
y = beta_0 + beta_1 * x

plt.scatter(x=x_train, y=y_train, color='red')
plt.plot(x, y)
```

```
plt.title("Closed-form Linear Regression of Y onto X")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

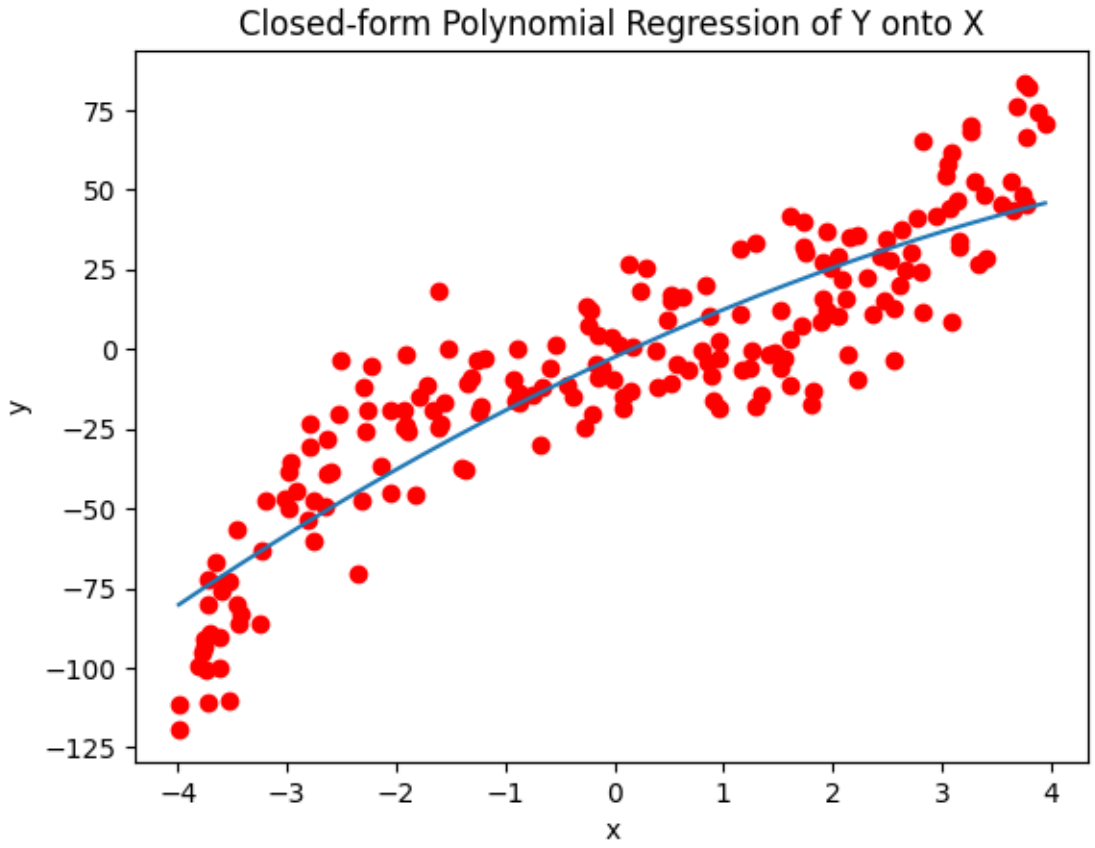


1.(a) Download the train data and apply closed-form solution on train data and with learned weights, plot regression line over training data. [2 Points]

Solution for Problem 1.(a):

[42]:
```
N = len(x_train)
X_train = np.ones((N, 3))
X_train[:, 1] = x_train
X_train[:, 2] = x_train**2

# Solve for theta using closed-form solution
XT_X = np.dot(X_train.T, X_train)
XT_y = np.dot(X_train.T, y_train)
theta = np.linalg.solve(XT_X, XT_y)
```

```
beta_0, beta_1, beta_2 = theta

# with learned weights, plot polynomial regression line over training data
x = np.linspace(min(x_train), max(x_train), len(x_train))
y = beta_0 + beta_1 * x + beta_2 * x**2

plt.scatter(x=x_train, y=y_train, color='red')
plt.plot(x, y)
plt.title("Closed-form Polynomial Regression of Y onto X")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



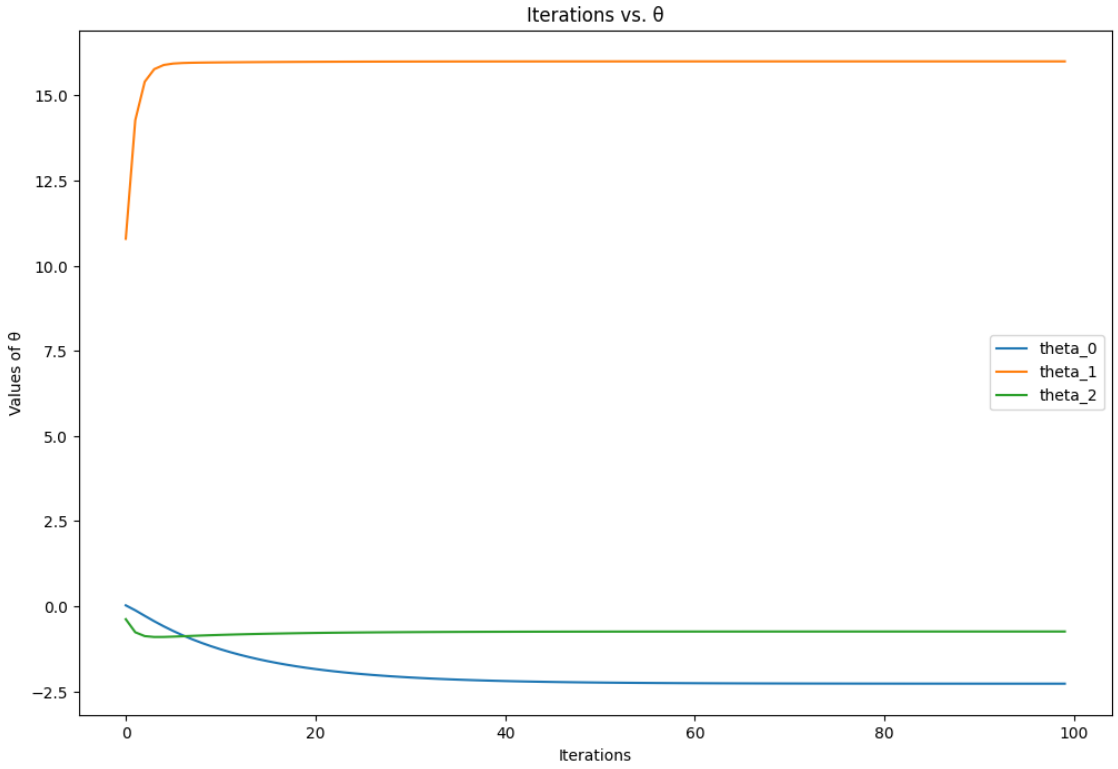
1.(b) Apply gradient descent with mini-batch of size 10. Plot iterations vs. . Change the mini-batch size and report its impact on learned weights. [3 Points]

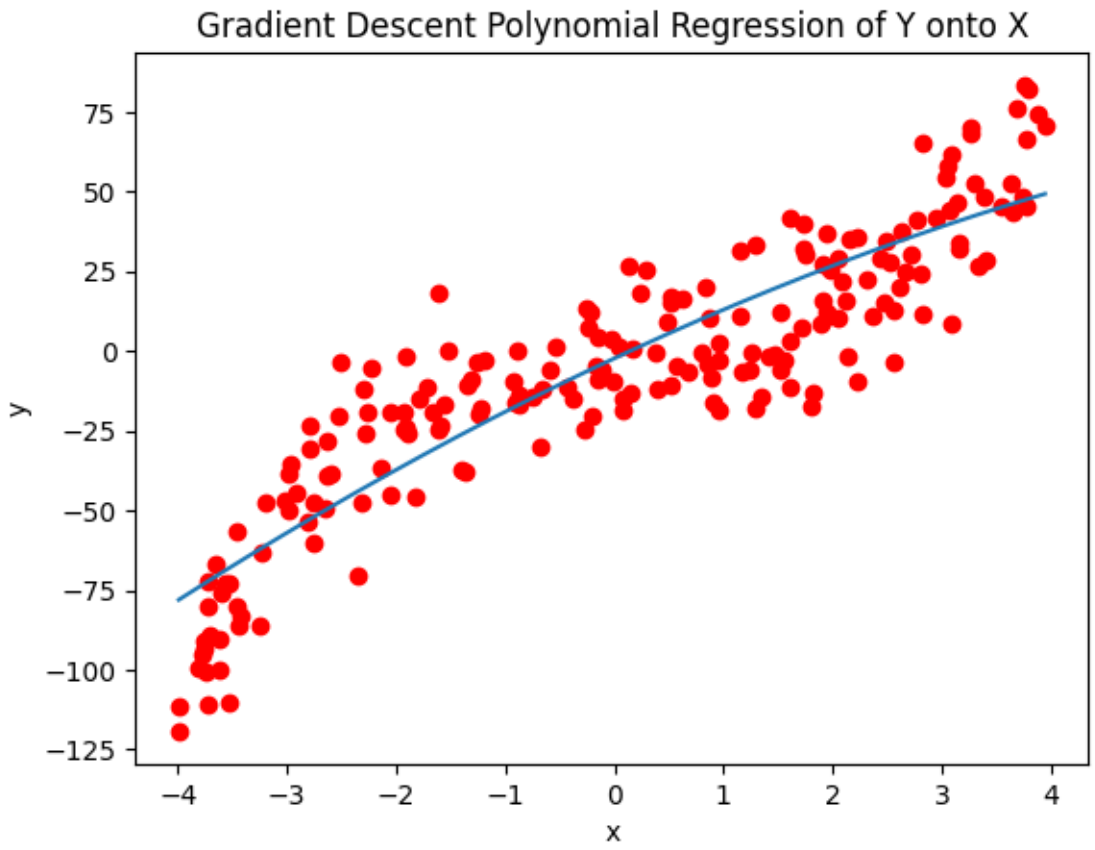
Solution for Problem 1.(b):

[43]:
```
X_train = np.ones((N, 3))
X_train[:, 1] = x_train
```

```
X_train[:, 2] = x_train**2

def compute_mse(X, y, theta):
    N = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    mse = (1 / (2 * N)) * np.dot(errors.T, errors)
    return mse

def gradient_descent(X, y, theta, learning_rate, iterations, batch_size):
    m = len(y)
    theta_iterations = np.zeros((iterations, len(theta)))
    mse_iterations = np.zeros(iterations)

    for i in range(iterations):
        for j in range(0, m, batch_size):
            X_batch = X[j:j+batch_size]
            y_batch = y[j:j+batch_size]

            predictions = X_batch.dot(theta)
            errors = predictions - y_batch
            gradient = (1 / batch_size) * X_batch.T.dot(errors)
            theta = theta - learning_rate * gradient

        mse_iterations[i] = compute_mse(X, y, theta)
        theta_iterations[i, :] = theta

    return theta, mse_iterations, theta_iterations

theta = np.zeros(3)
learning_rate = 0.01
iterations = 100
batch_size = 10

theta_final, mse_iterations, theta_iterations = gradient_descent(X_train,␣
 ↪y_train, theta, learning_rate, iterations, batch_size)

# Plot iterations vs. theta
plt.figure(figsize=(12, 8))
plt.plot(range(iterations), theta_iterations[:, 0], label='theta_0')
plt.plot(range(iterations), theta_iterations[:, 1], label='theta_1')
plt.plot(range(iterations), theta_iterations[:, 2], label='theta_2')
plt.xlabel('Iterations')
plt.ylabel('Values of  ')
plt.title('Iterations vs.  ')
plt.legend()
plt.show()
```

```
# Plot the training data and the polynomial regression line
x = np.linspace(min(x_train), max(x_train), len(x_train))
y = theta_final[0] + theta_final[1] * x + theta_final[2] * x**2

plt.scatter(x=x_train, y=y_train, color='red')
plt.plot(x, y)
plt.title("Gradient Descent Polynomial Regression of Y onto X")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

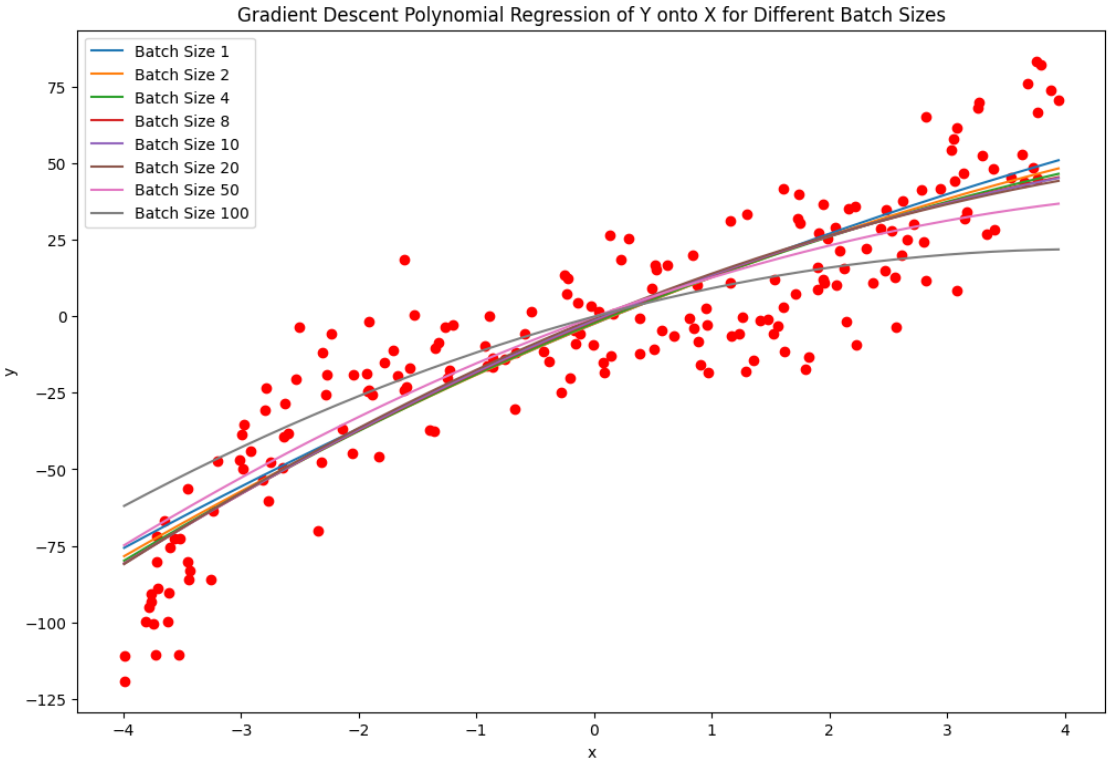




The effects of varying the batch size are shown below. As the batch size is varied, the values of theta change slightly and the MSE also reduces. Batch sizes of 50 and 100 clearly have larger MSE values.

[44]:
```
X_train = np.ones((N, 3))
X_train[:, 1] = x_train
X_train[:, 2] = x_train**2

def compute_mse(X, y, theta):
    N = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    mse = (1 / (2 * N)) * np.dot(errors.T, errors)
    return mse

learning_rate = 0.001
iterations = 100

batch_sizes = [1, 2, 4, 8, 10, 20, 50, 100]
```

```
# Plotting setup
plt.figure(figsize=(12, 8))

for batch_size in batch_sizes:
    theta = np.zeros(3)
    theta_iterations = np.zeros((iterations, len(theta)))
    mse_iterations = np.zeros(iterations)

    for i in range(iterations):
        for j in range(0, N, batch_size):
            X_batch = X_train[j:j+batch_size]
            y_batch = y_train[j:j+batch_size]

            predictions = X_batch.dot(theta)
            errors = predictions - y_batch
            gradient = (1 / batch_size) * X_batch.T.dot(errors)
            theta = theta - learning_rate * gradient

        mse_iterations[i] = compute_mse(X_train, y_train, theta)
        theta_iterations[i, :] = theta

    # Plot the polynomial regression line for each batch size
    x = np.linspace(min(x_train), max(x_train), 200)
    y = theta[0] + theta[1] * x + theta[2] * x**2
    plt.plot(x, y, label=f'Batch Size {batch_size}')

# Plot the training data
plt.scatter(x=x_train, y=y_train, color='red')
plt.title("Gradient Descent Polynomial Regression of Y onto X for Different␣
 ↪Batch Sizes")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```



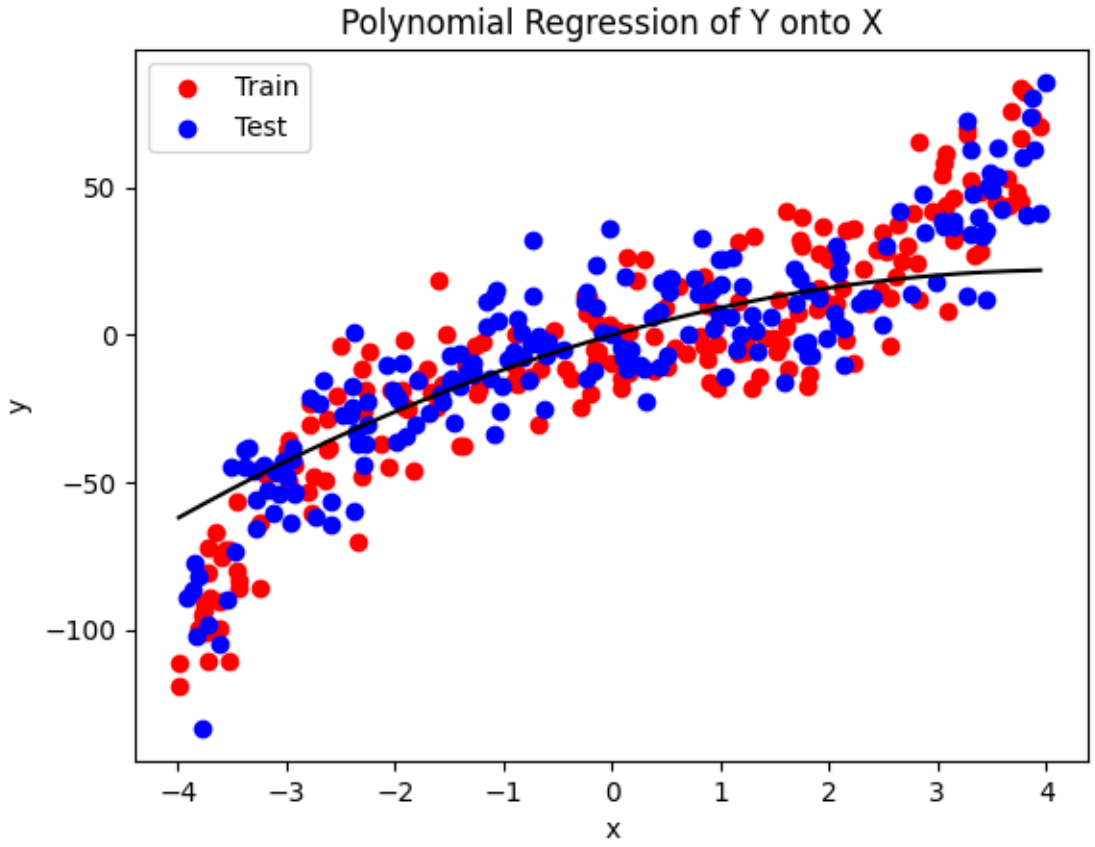
Write a predict function that takes a new test sample (xtst,ytst), and learned weights , and outputs prediction yˆ. Report train and test error. [2 Points]

Solution for Problem 1.(c):

[45]:
```
def predict(X, theta):
    return X.dot(theta)

test_path = "HW-2-Data/test_data.csv"
test = pd.read_csv(test_path)

x_test = test["x"].values
y_test = test["y"].values

X_test = np.ones((N, 3))
X_test[:, 1] = x_test
X_test[:, 2] = x_test**2

y_train_pred = predict(X_train, theta_final)
y_test_pred = predict(X_test, theta_final)
train_mse = compute_mse(X_train, y_train, theta_final)
test_mse = compute_mse(X_test, y_test, theta_final)
```

```
print(f"Train MSE: {train_mse}")
print(f"Test MSE: {test_mse}")

plt.scatter(x=x_train, y=y_train, color='red', label='Train')
plt.scatter(x=x_test, y=y_test, color='blue', label='Test')
plt.plot(x, y, color='black')
plt.title("Polynomial Regression of Y onto X")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
Train MSE: 171.08715124944843
Test MSE: 161.52037671198454
```



2. Consider n degree of polynomials, (x) = [1, x, x2, ..., xn].

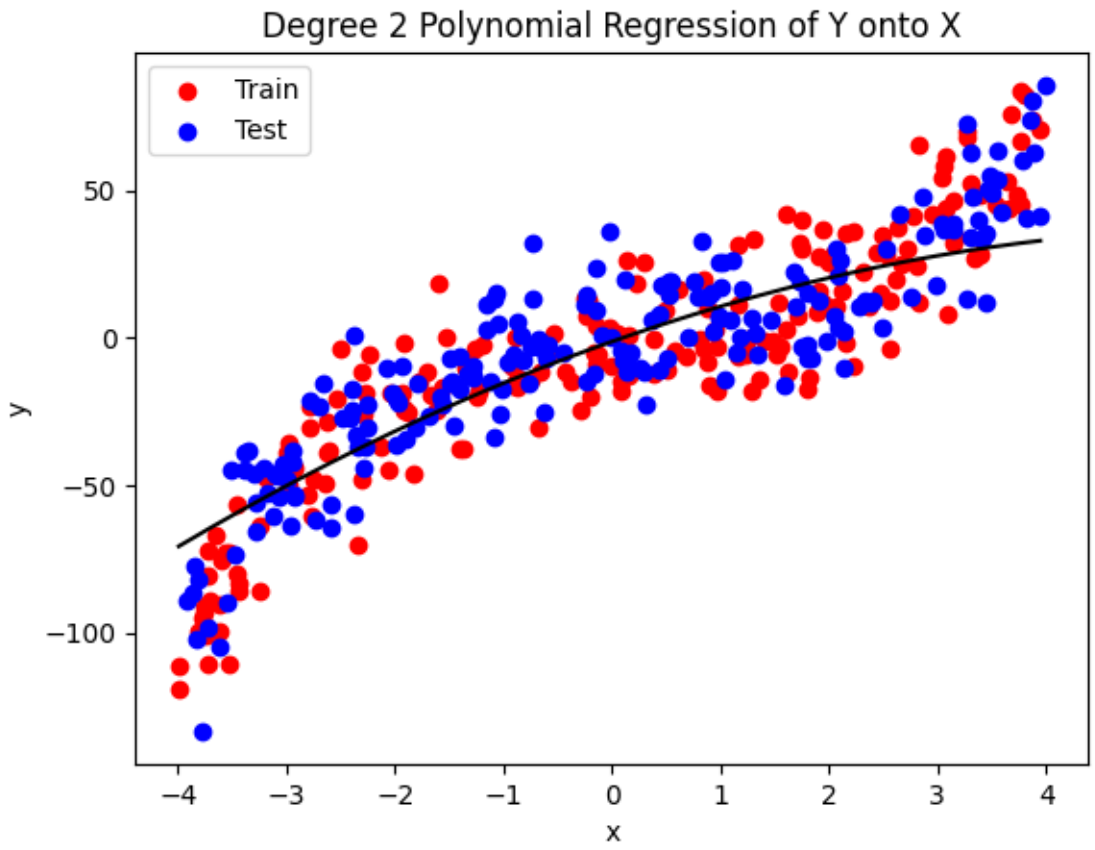
(a) Run the code again on training data to compute theta for n {2,3,5}, and report regression error (mean squared error) on train and test data for all n. [15 Points]

Solution for Problem 2.(a):

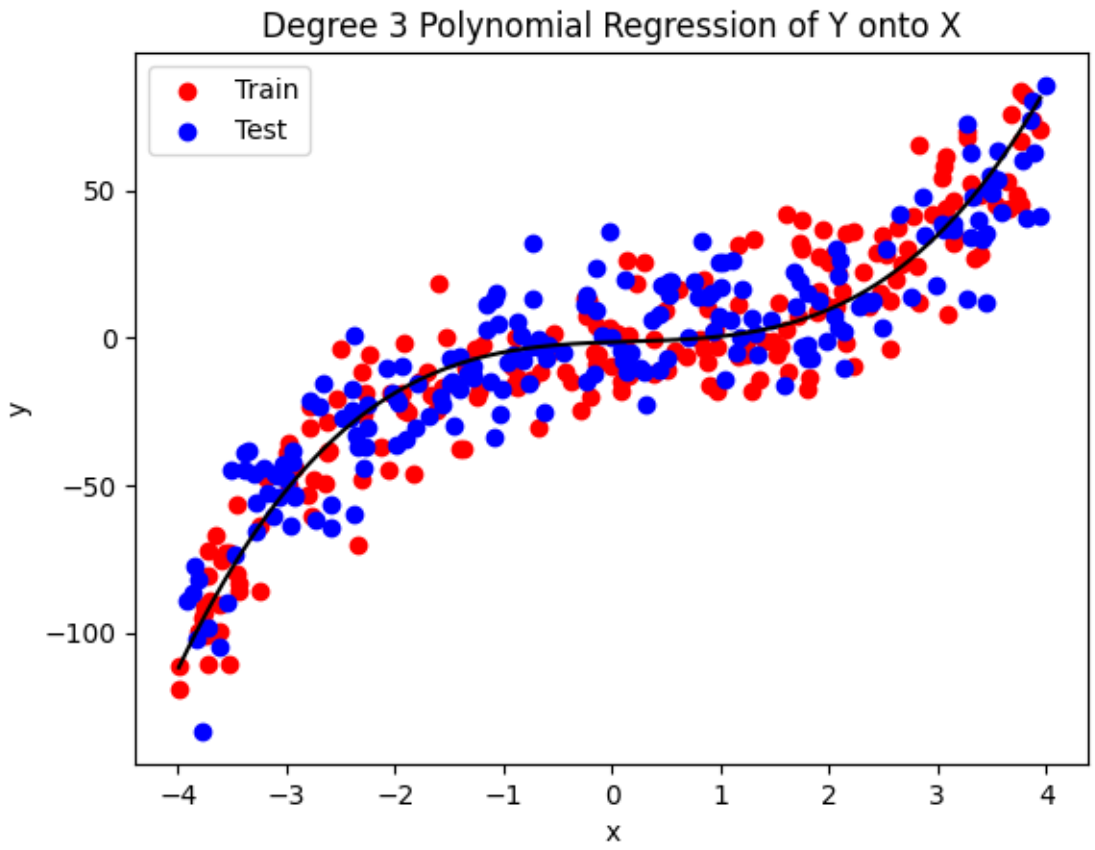
[46]:
```
test_path = "HW-2-Data/test_data.csv"
test = pd.read_csv(test_path)

x_test = test["x"].values
y_test = test["y"].values

def compute_mse(X, y, theta):
    N = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    mse = (1 / (2 * N)) * np.dot(errors.T, errors)
    return mse

def gradient_descent(X, y, theta, learning_rate, iterations, batch_size):
    N = len(y)
    theta_iterations = np.zeros((iterations, len(theta)))
    mse_iterations = np.zeros(iterations)

    for i in range(iterations):
        for j in range(0, N, batch_size):
            X_batch = X[j:j+batch_size]
            y_batch = y[j:j+batch_size]

            predictions = X_batch.dot(theta)
            errors = predictions - y_batch
            gradient = (1 / batch_size) * X_batch.T.dot(errors)
            theta = theta - learning_rate * gradient

        mse_iterations[i] = compute_mse(X, y, theta)
        theta_iterations[i, :] = theta

    return theta, mse_iterations, theta_iterations

def predict(X, theta):
    return X.dot(theta)

def create_matrix(x, degree, mean, range_x):
    m = len(x)
    X = np.ones((m, degree + 1))
    for i in range(1, degree + 1):
        X[:, i] = (x ** i - mean) / range_x
    return X

degrees = [2, 3, 5]
learning_rate = 0.001
iterations = 1000
batch_size = 10
```

```
# Feature normalization
mean = np.mean(x_train)
range_x = np.max(x_train) - np.min(x_train)

# Run gradient descent for each polynomial degree and report MSE
for degree in degrees:
    print(f"Degree {degree} polynomial:")

    X_train = create_matrix(x_train, degree, mean, range_x)
    X_test = create_matrix(x_test, degree, mean, range_x)

    theta = np.zeros(degree + 1)
    theta_final, mse_iterations, theta_iterations = gradient_descent(X_train,␣
 ↪y_train, theta, learning_rate, iterations, batch_size)

    y_train_pred = predict(X_train, theta_final)
    y_test_pred = predict(X_test, theta_final)

    train_mse = compute_mse(X_train, y_train, theta_final)
    test_mse = compute_mse(X_test, y_test, theta_final)

    print(f"Train MSE: {train_mse}")
    print(f"Test MSE: {test_mse}\n")

    x = np.linspace(min(x_train), max(x_train), len(x_train))
    X_plot = create_matrix(x, degree, mean, range_x)
    y_plot = predict(X_plot, theta_final)

    plt.scatter(x=x_train, y=y_train, color='red', label='Train')
    plt.scatter(x=x_test, y=y_test, color='blue', label='Test')
    plt.plot(x, y_plot, color='black')
    plt.title(f"Degree {degree} Polynomial Regression of Y onto X")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()
    plt.show()
```

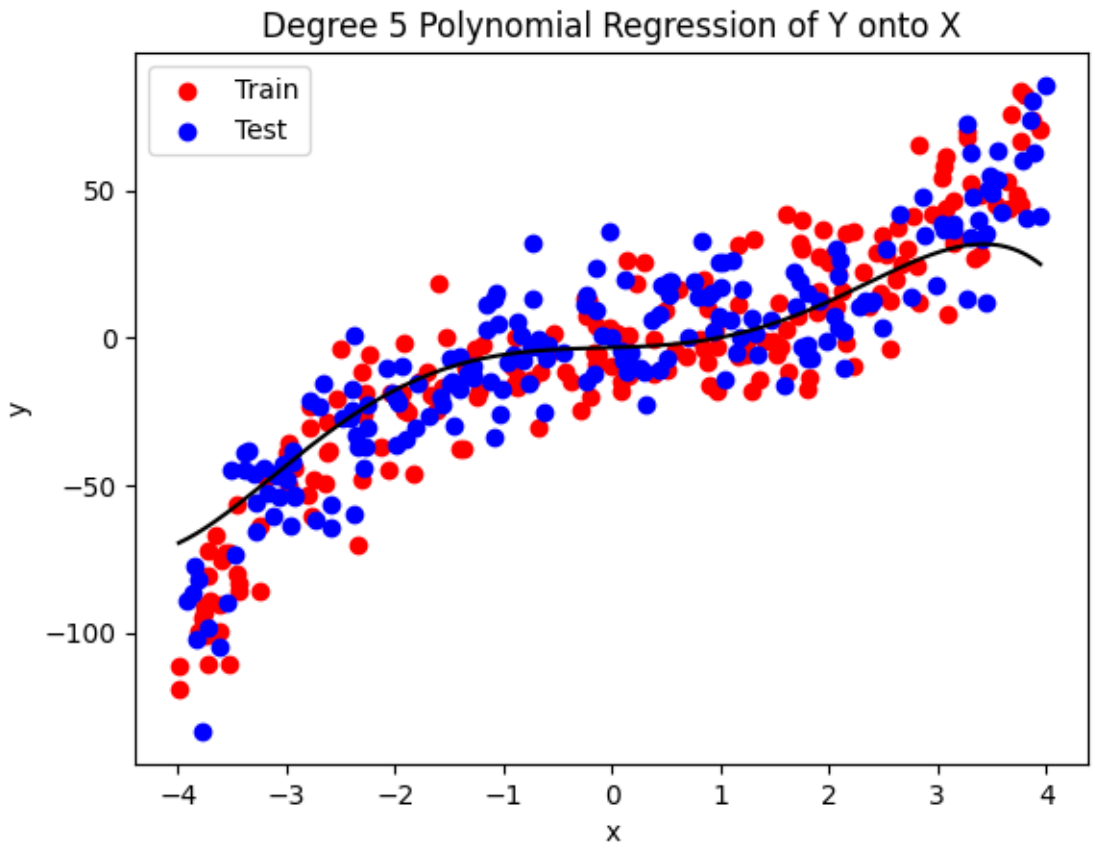
```
Degree 2 polynomial:
Train MSE: 191.69162742722503
Test MSE: 156.05464357342015
```




```
Degree 3 polynomial:
Train MSE: 113.61071881078004
Test MSE: 113.55013885123044
```




```
Degree 5 polynomial:
Train MSE: 194.13014332773173
Test MSE: 153.9518913371808
```



2.(b) Which value of n is the best fit for this data? Explain why. [4 Points]

The third degree polynomial appears to be the best fit for the provided data. It captures the trend and almost all of the variations in the data and does not seem to exhibit overfitting. The second degree polynomial is slightly underfit as seen in the plot as it does not capture the trend for the smallest and largest values of x. The degree 5 polynomial does not manage to fit the data well and also has high bias for the smallest and largest values of x. We can also compared each based on MSE and notice that the third degree polynomial has the lowest values for both train and test sets, showing low bias and extremely low variance for this dataset. If I had to guess, I would say that this dataset is synthetic and was created using a degree 3 polynomial, with 200 observations sampled using a Gaussian distribution.