question5.py

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

# Load the data from the provided CSV file.
data = pd.read_csv("HW2_nonlinear_data.csv")
# Load the first (input/feature) column into the vector X.
X = data.iloc[:, 0].values  # First column as input (feature)
# Load the second (output/target) colulmn into the vector Y.
Y = data.iloc[:, 1].values  # Second column as output (target)

# Reshape X for computation.  This transforms X from the initial
# one-dimensional array of n values into a 2D array having n rows
# and 1 column.  The first parameter value of -1 causes numpy to
# infer the number of rows based on the size of X.  The second
# parameter gives the number of columns as 1.
X = X.reshape(-1, 1)

# A cubic regression uses an expression of the form:
#     Y = aX^3 + bX^2 + cX + d
# The coefficient values a, b, c, and d are the values that will
# be learned through the training.

# Initialize the coefficients to zero.
a = b = c = d = 0

# Set the learning rate and number of epochs to the values given in
# the problem statement.  They can be adjusted, but start here.
learning_rate = 1e-6
epochs = 10000

# Count the number of input values.
n = len(X)

# Calculate the gradient descent.
for e in range(epochs):
    # Calcuate the predicted value
    Y_pred = (a*(X**3)) + (b*(X**2)) + (c*X) + d
    # Calculate the error relative to ground truth.
    error = Y_pred.flatten() - Y
    # Use the error value to calculate the MSE.
    mse = np.mean(error**2)

    # Update the coefficients using gradient descent.

    # Compute gradients, which are the partial derivatives
    # with respect to a, b, c, d).
```

```python
    # The flatten() method takes the 2D array created by reshape()
    # and returns 1D array that is needed for numpy element-wise operations.
    da = (2/n) * np.sum(error * X.flatten()**3)
    db = (2/n) * np.sum(error * X.flatten()**2)
    dc = (2/n) * np.sum(error * X.flatten())
    dd = (2/n) * np.sum(error)

    # Update the coefficients using the learning rate and the gradients.
    a -= learning_rate * da
    b -= learning_rate * db
    c -= learning_rate * dc
    d -= learning_rate * dd

    # Print the MSE value every 1000 epochs to see changes.
    if e % 1000 == 0:
        print(f"Epoch {e}: MSE = {mse}")

# Generate the final predictions using the trained coefficients.
Y_pred = (a*(X**3)) + (b*(X**2)) + (c*X) + d

# Plot the results
# Generate a scatter plot of the ground-truth data.
plt.scatter(X, Y, color="blue", label="Actual Data")
# Overlay the predicted values as red points.
plt.scatter(X, Y_pred, color="red", label="Fitted Curve")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Non-linear (Cubic) Regression")
plt.show()

# Print final trained coefficients.
print(f"Final coefficients:\n     a={a}\n     b={b}\n     c={c}\n     d={d}")
```