Assignment #2 -- Regression

題目

1. Use the linear model $y = 2x + \varepsilon$ with zero-mean Gaussian noise $\varepsilon \sim N(0, 1)$ to generate 15 data points with (equal spacing) $x \in [-3, 3]$.

2. Perform Linear Regression. Show the fitting plot, the training error, and the five-fold cross-validation errors.

3. Perform Polynomial Regression with degree 5, 10 and 14, respectively. For each case, show the fitting plot, the training error, and the five-fold cross-validation errors. (Hint: Arrange the polynomial regression equation as follows and solve the model parameter vector w.)

$$y = \begin{bmatrix} x_1^5 & x_1^4 & x_1^3 & x_1^2 & x_1^1 & 1 \\ x_2^5 & x_2^4 & x_2^3 & x_2^2 & x_2^1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{15}^5 & x_{15}^4 & x_{15}^3 & x_{15}^2 & x_{15}^1 & 1 \end{bmatrix} \bullet \begin{bmatrix} w_5 \\ w_4 \\ w_3 \\ w_2 \\ w_1 \\ w_0 \end{bmatrix}$$

4. Change the model to $y = \sin(2\pi x) + \varepsilon$ with the noise $\varepsilon \sim N(0, 0.04)$ and (equal spacing) $x \in [0, 1]$. Then repeat those stated in 2) and 3). Compare the results with linear/polynomial regression on different datasets.

5. Following 4), perform polynomial regression with degree 14 by varying the number of training data points $m = 10, 80, 320$. Show the five-fold cross-validation errors and the fitting plots. Compare the results to those in 4).

6. Following 4), perform polynomial regression of degree 14 via **regularization**. Compare the results by setting $\lambda = 0, 0.001/m, 1/m, 1000/m$, where $m = 15$ is the number of data points (with $x = 0, 1/(m-1), 2/(m-1), \ldots, 1$). Show the five-fold cross-validation errors and the fitting plots.

Note:
- The assignment should be implemented by Python.
- You need to hand in the python code and the report.
- In your report, it should contain: (請以中文撰寫)
- **Execution description**: steps how to execute your codes.
- **Experimental results**: As specified in the assignment.
- **Conclusion**: The observation from your results.
- **Discussion**: The questions or the difficulties you met during the implementation.
- Assignment format
    - Zip all your files into a single one and upload it to the E-Course2 website.
- Please format the file name as: Student ID_proj1_verNo, ex：602410143_proj1_v1 • No copy! Late policy applies.

我的答案

如何執行 (Execution Description)

- 用 python 將檔案執行：

```
python3 regression_code.py
```

實驗結果 (Experimental Results)

- 題目 1 實驗結果：
    - 使用以下程式碼產生 15 個點：

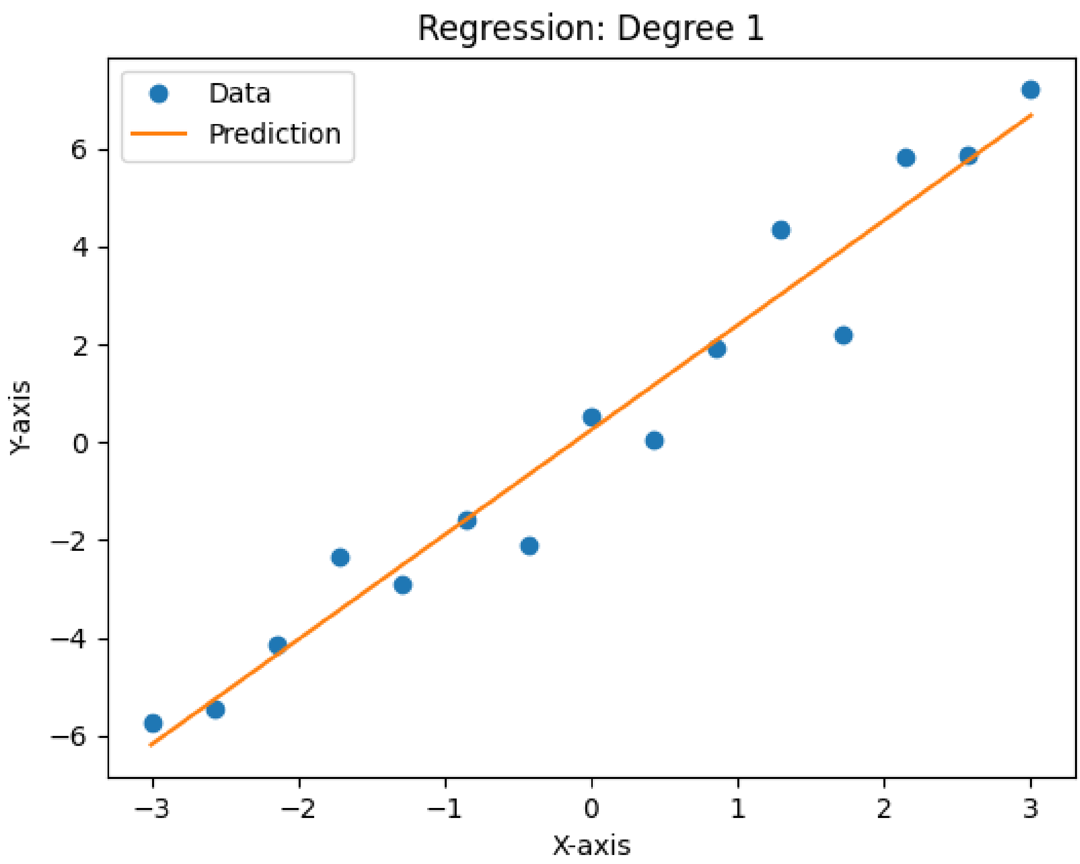
```
# y = 2 * x + epsilon

n_points = 15
x_min, x_max = -3, 3
x_init = torch.linspace(x_min, x_max, n_points)
epsilon = torch.randn(n_points)
y_init = 2 * x_init + epsilon

print("x_init:", x_init)
print("y_init:", y_init)
```

    - 結果截圖如下：

```
x_init: tensor([-3.0000, -2.5714, -2.1429, -1.7143, -1.2857, -0.8571, -0.4286,  0.0000,
         0.4286,  0.8571,  1.2857,  1.7143,  2.1429,  2.5714,  3.0000])
y_init: tensor([-5.7266, -5.4377, -4.1348, -2.3594, -2.9223, -1.5806, -2.0920,  0.5303,
         0.0399,  1.9234,  4.3453,  2.2039,  5.8454,  5.8555,  7.2106])
```

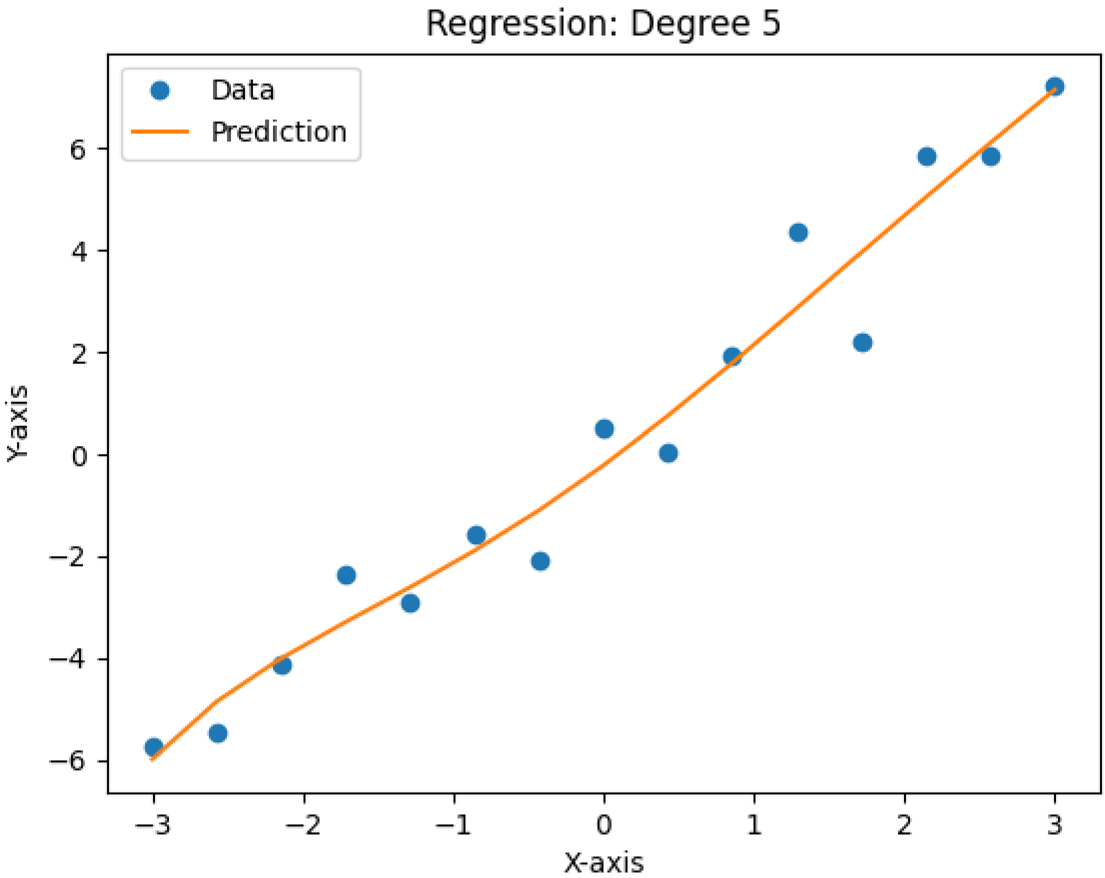
- 題目 2 實驗結果：

- 我實作 `poly_regression` 這個 function 來產生 Linear Regression (Degree : 1) 的圖，並且計算出 train loss 與 average kfold cross validation loss 。
- 結果截圖如下：

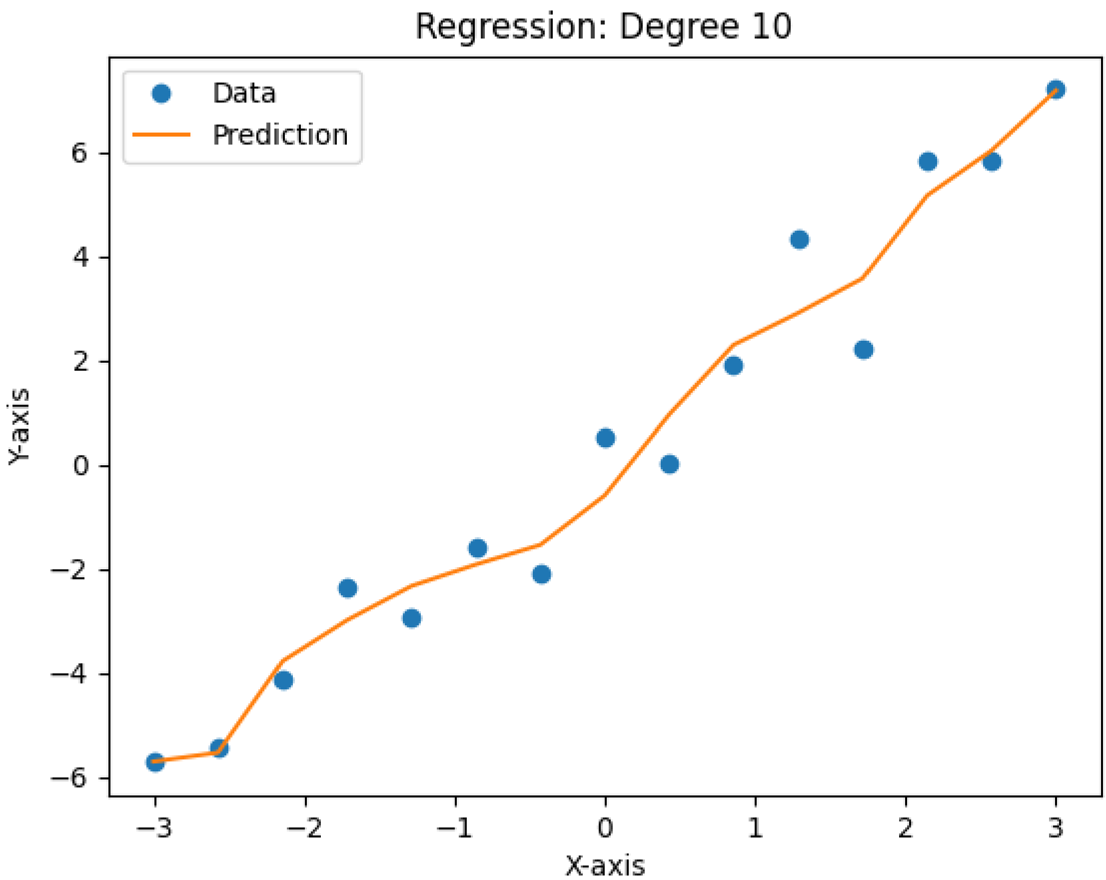



```
DEGREE --> 1

train_mse_loss: 0.7361408472061157
kfold_avg_loss: 1.1283553004264832
```

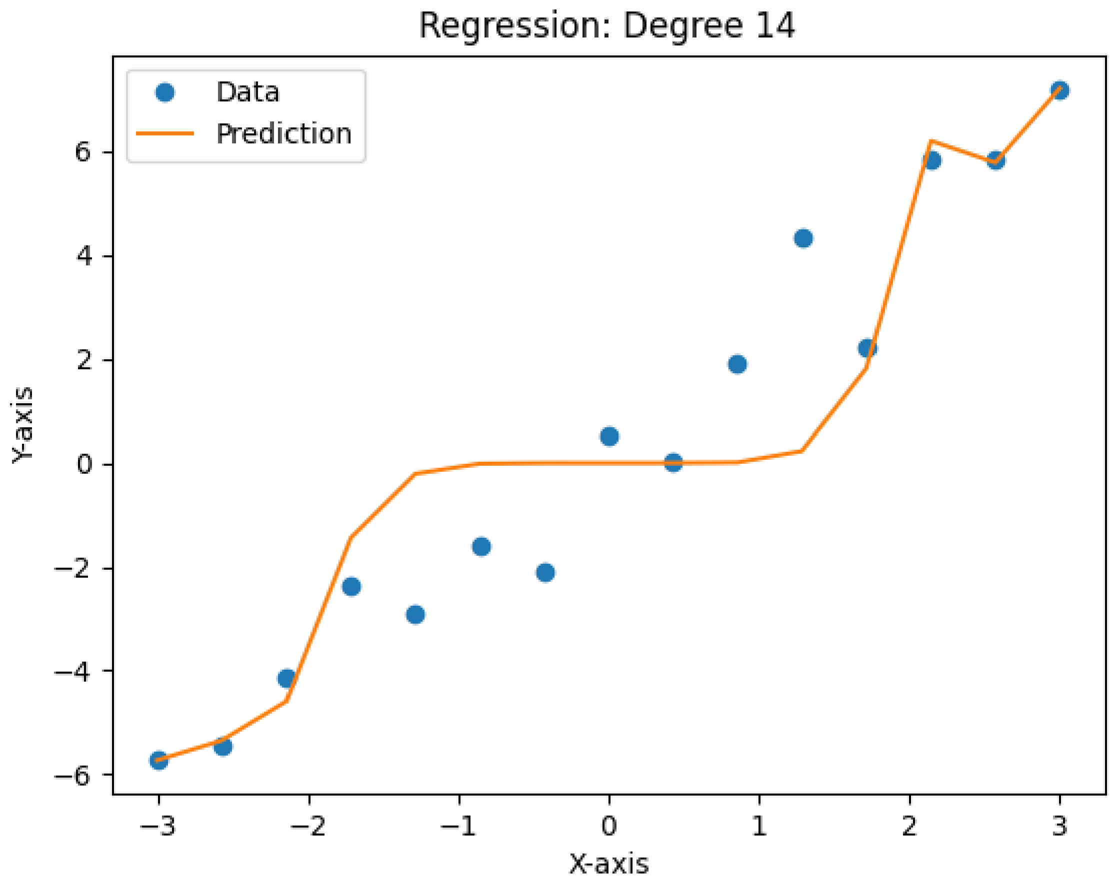
- 題目 3 實驗結果：
  - 一樣是透過我所實作的 `poly_regression` 進行運算，指是給定特定的次方 (5, 10, 14) 。
  - 結果截圖如下：



Regression: Degree 5
Data
Prediction
Y-axis
X-axis
6
4
2
0
−2
−4
−6
−3
−2
−1
0
1
2
3

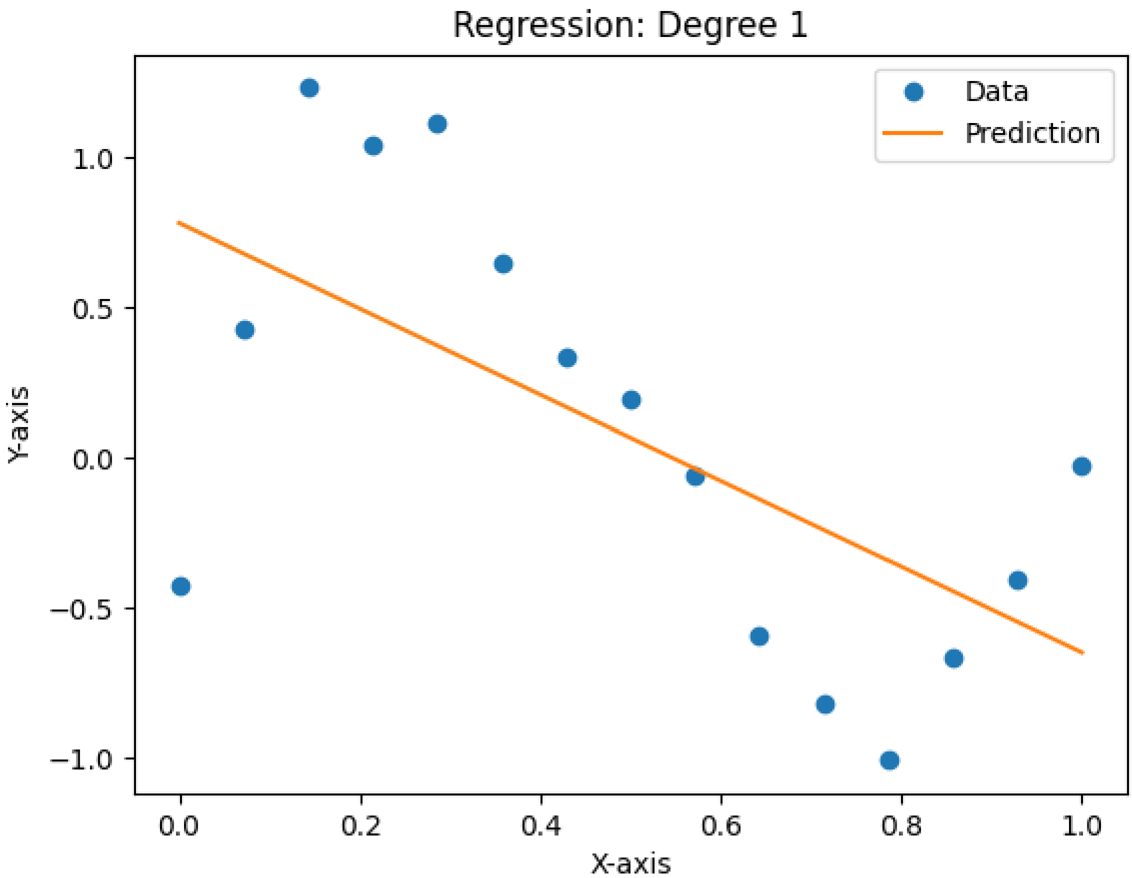


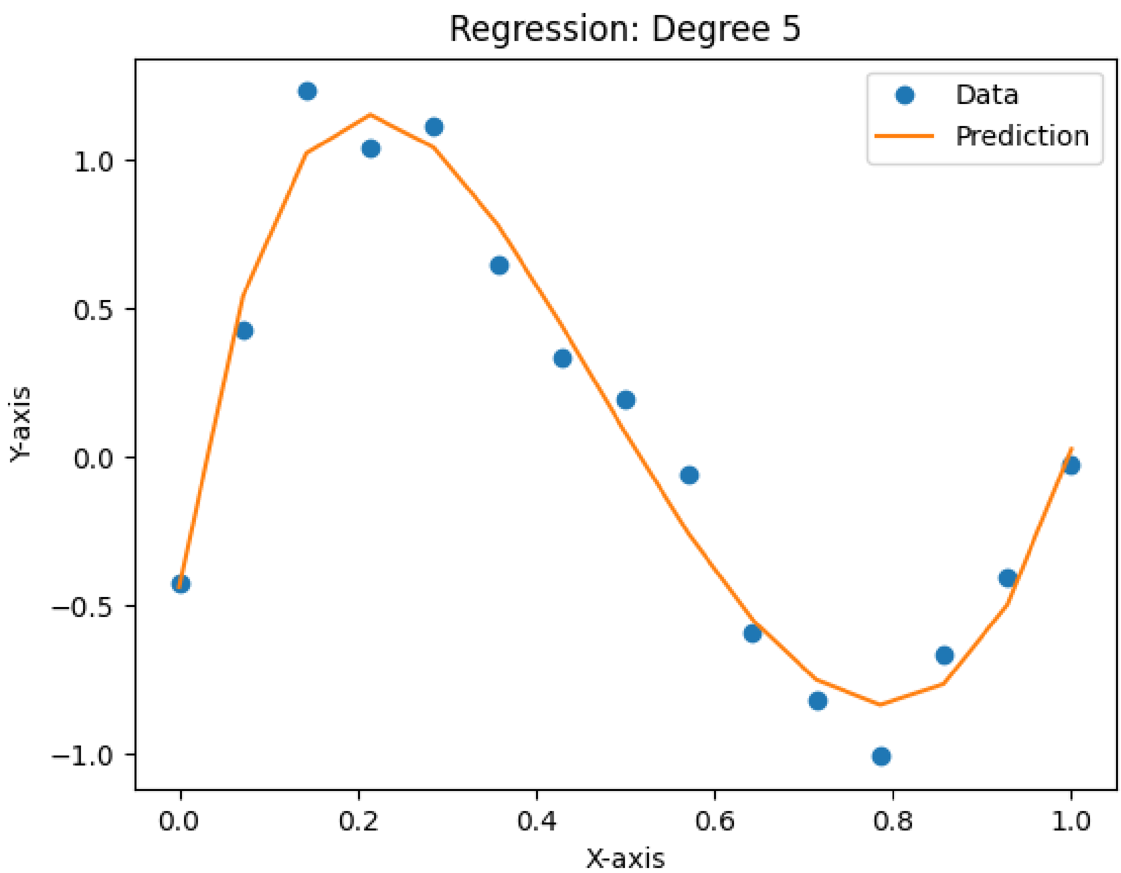
Regression: Degree 10
Data
Prediction
Y-axis
X-axis
6
4
2
0
−2
−4
−6
−3
−2
−1
0
1
2
3

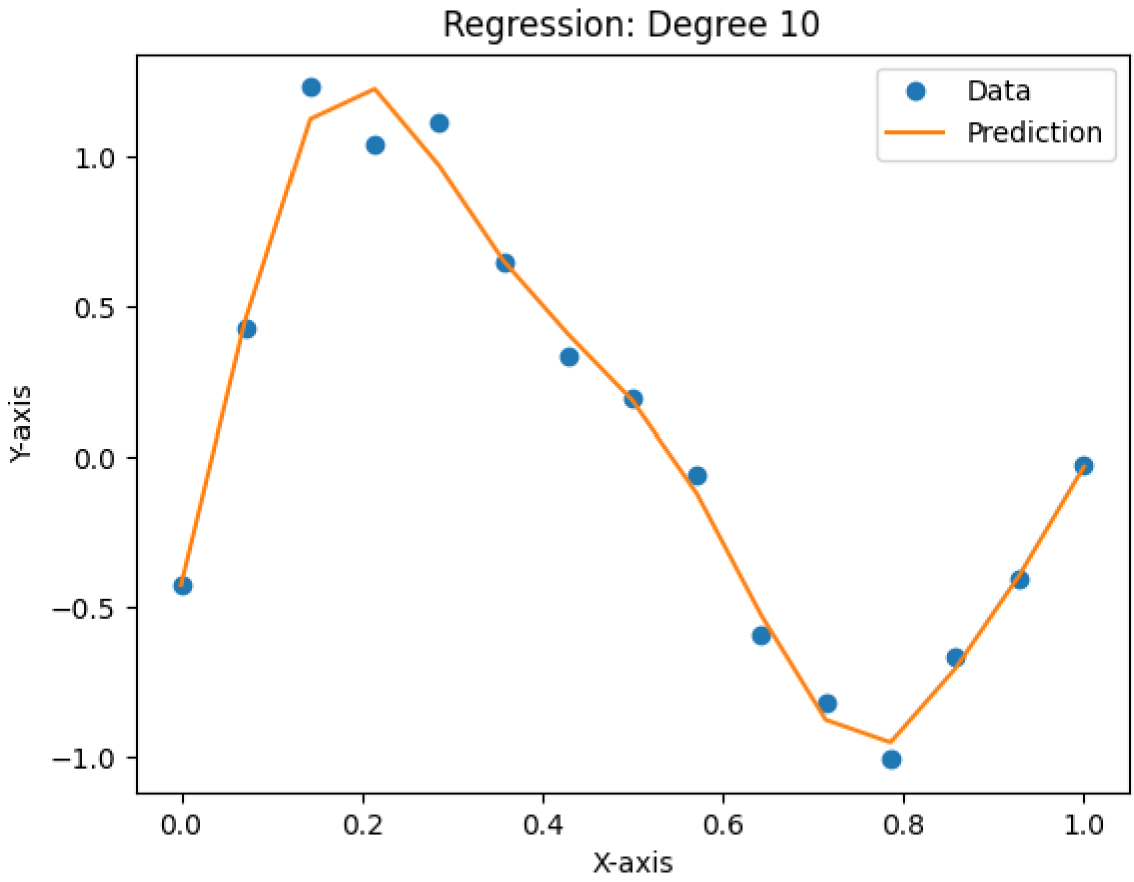



```
DEGREE --> 5
train_mse_loss: 0.6356481313705444
kfold_avg_loss: 329.4580878019333
DEGREE --> 10
train_mse_loss: 0.5307279825210571
kfold_avg_loss: 2368322.752342892
DEGREE --> 14
train_mse_loss: 2.4299495220184326
kfold_avg_loss: 570395113.5250143
```

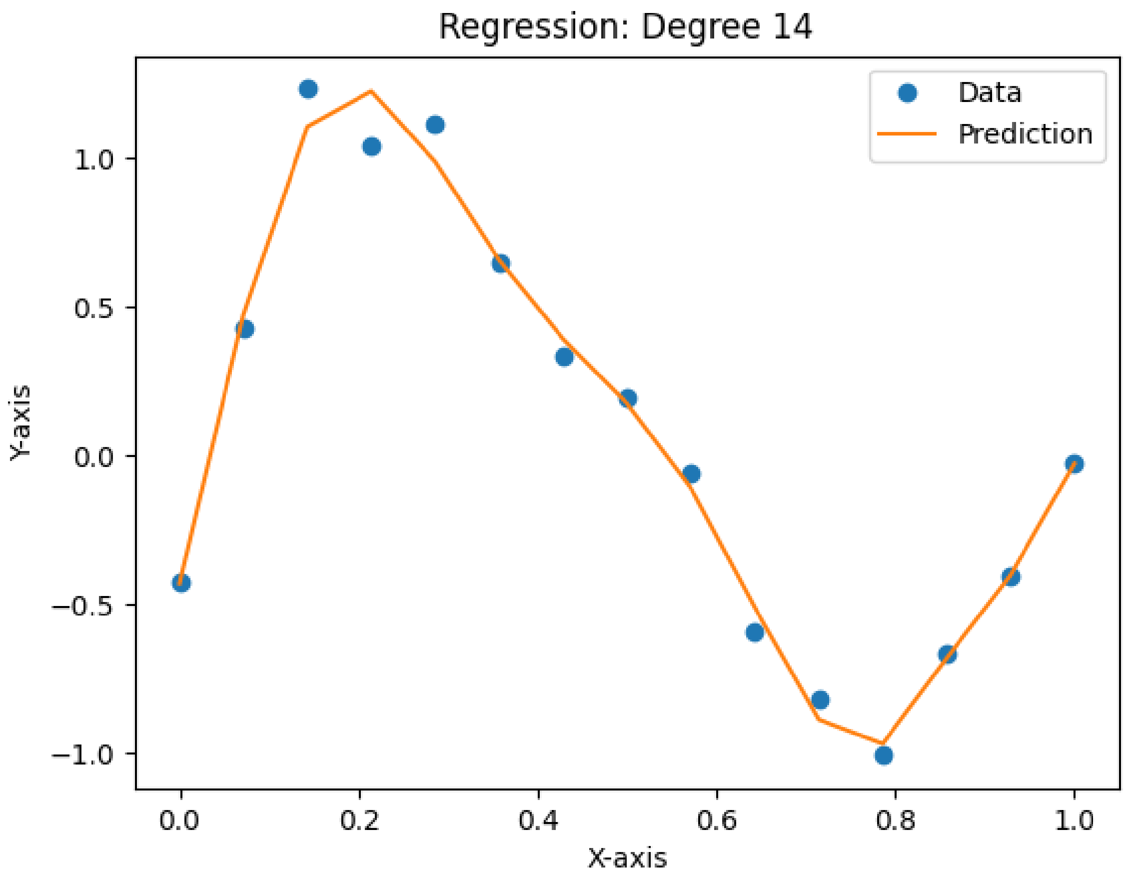
- 題目 4 實驗結果：
  - 重新用新的公式產生出 15 個點，並且一樣做 regression ，次方數我用 1, 5, 10, 14 。
  - 結果截圖如下：


Regression: Degree 1
Data
Prediction
1.0
0.5
0.0
−0.5
−1.0
Y-axis
0.0
0.2
0.4
0.6
0.8
1.0
X-axis


Regression: Degree 5
Data
Prediction
1.0
0.5
0.0
−0.5
−1.0
Y-axis
0.0
0.2
0.4
0.6
0.8
1.0
X-axis



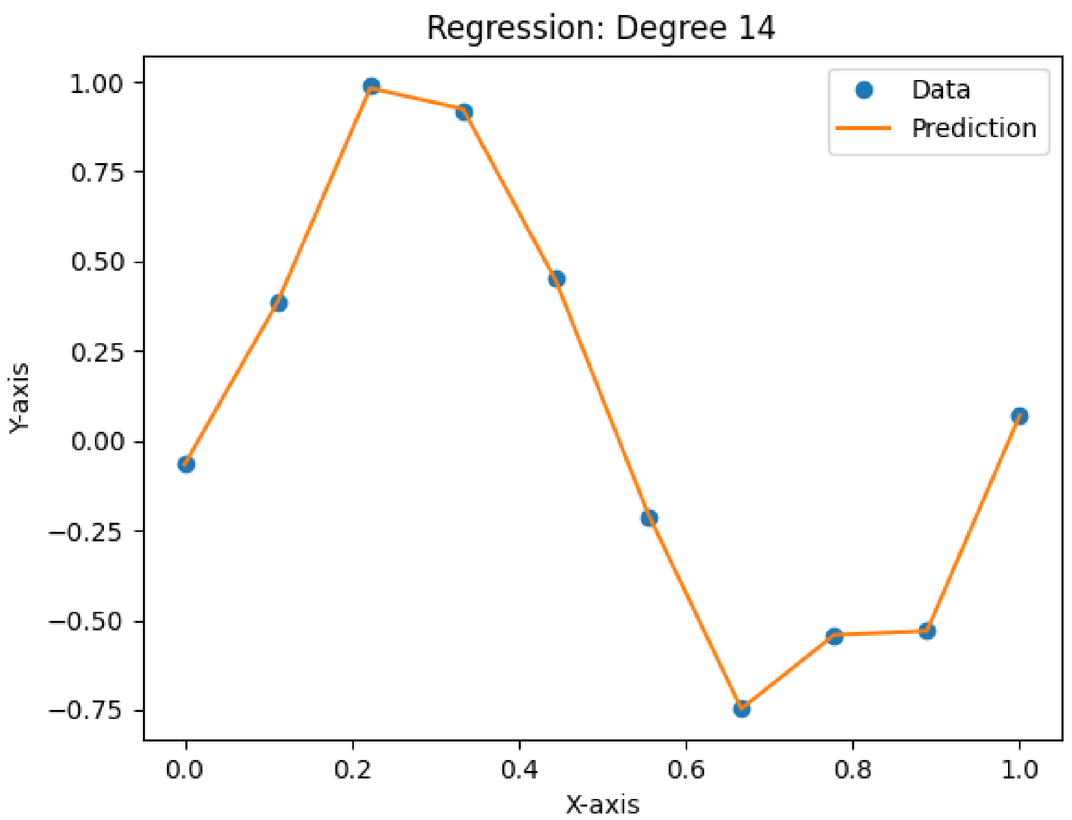
Regression: Degree 10
Data
Prediction
1.0
0.5
0.0
−0.5
−1.0
Y-axis
0.0
0.2
0.4
0.6
0.8
1.0
X-axis



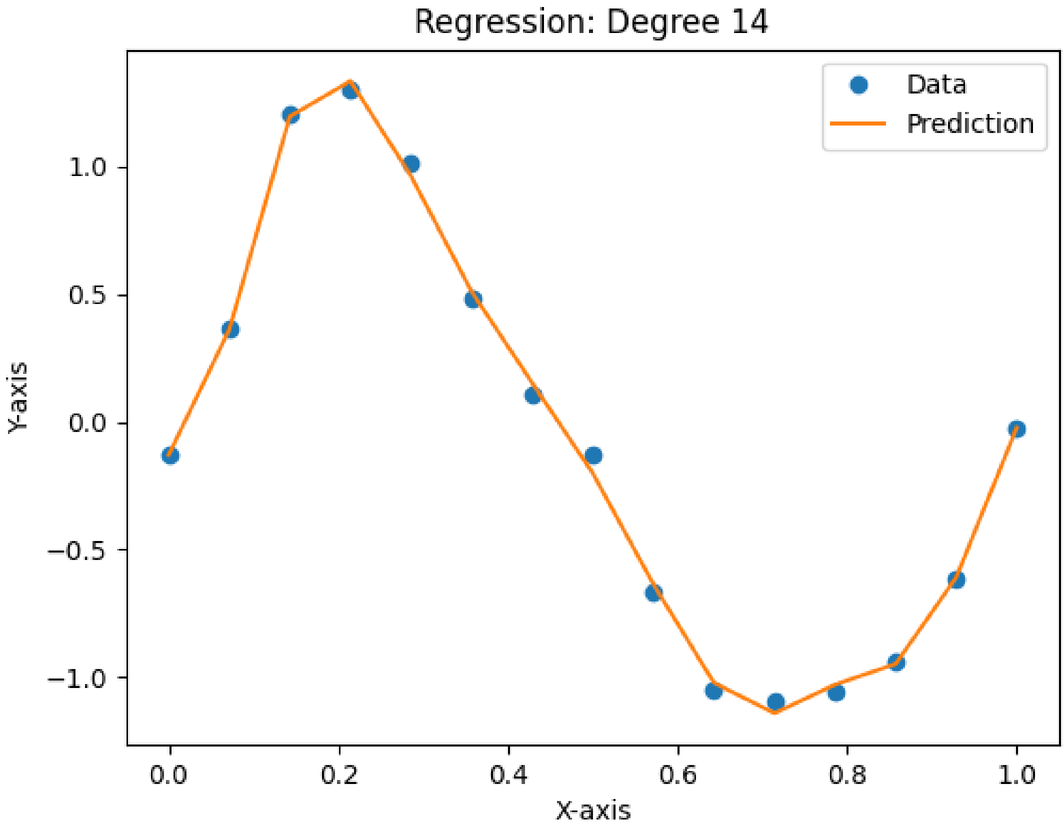
Regression: Degree 14
Data
Prediction
1.0
0.5
0.0
−0.5
−1.0
Y-axis
0.0
0.2
0.4
0.6
0.8
1.0
X-axis

```
x_init: tensor([0.0000, 0.0714, 0.1429, 0.2143, 0.2857, 0.3571, 0.4286, 0.5000, 0.5714,
        0.6429, 0.7143, 0.7857, 0.8571, 0.9286, 1.0000])
y_init: tensor([-0.4213,  0.4292,  1.2339,  1.0439,  1.1186,  0.6512,  0.3388,  0.1952,
        -0.0584, -0.5905, -0.8162, -1.0063, -0.6652, -0.4039, -0.0249])
DEGREE --> 1
train_mse_loss: 0.2964556813240051
kfold_avg_loss: 0.5869197141379118
DEGREE --> 5
train_mse_loss: 0.014215162955224514
kfold_avg_loss: 1.1550547793507575
DEGREE --> 10
train_mse_loss: 0.0059426468797028065
kfold_avg_loss: 893.1593061511405
DEGREE --> 14
train_mse_loss: 0.0058064451441168785
kfold_avg_loss: 8949.376399468816
```

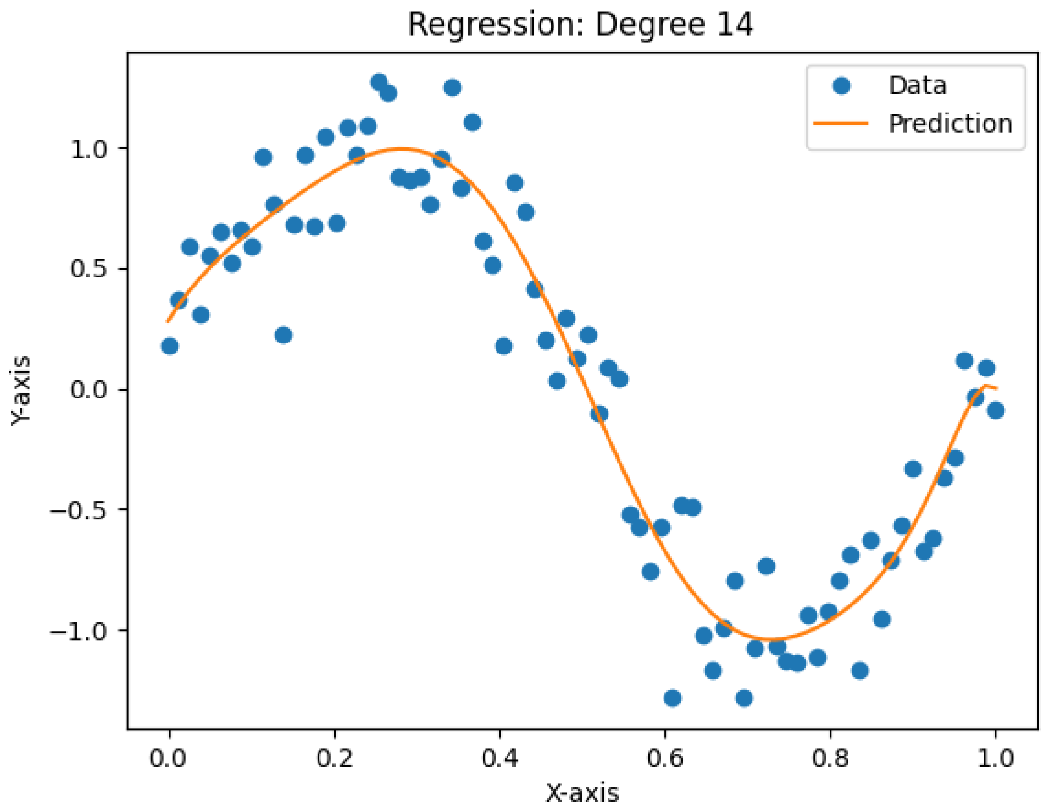
- 題目 5 實驗結果：
  - 產生對於不同資料個數所訓練出來的結果，這邊使用 14 次多向式。
  - 結果截圖如下：
    - 10 組資料：



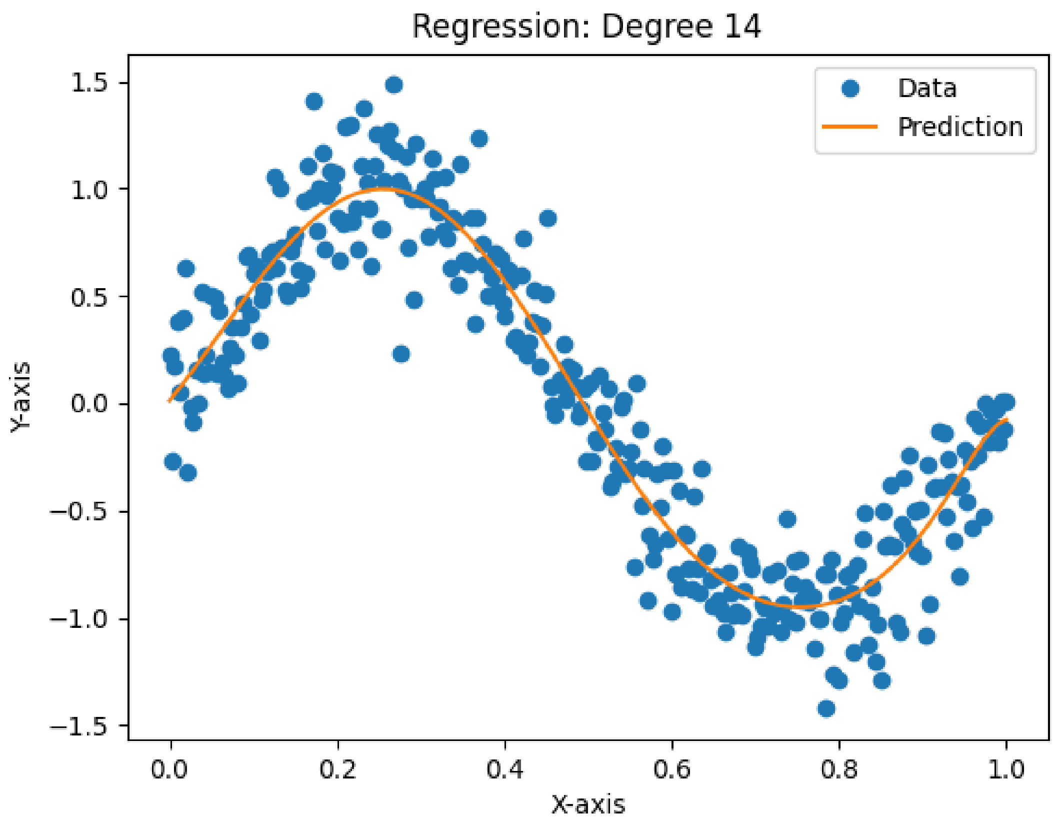
    - 15 組資料：



- 80 組資料：



- 320 組資料：

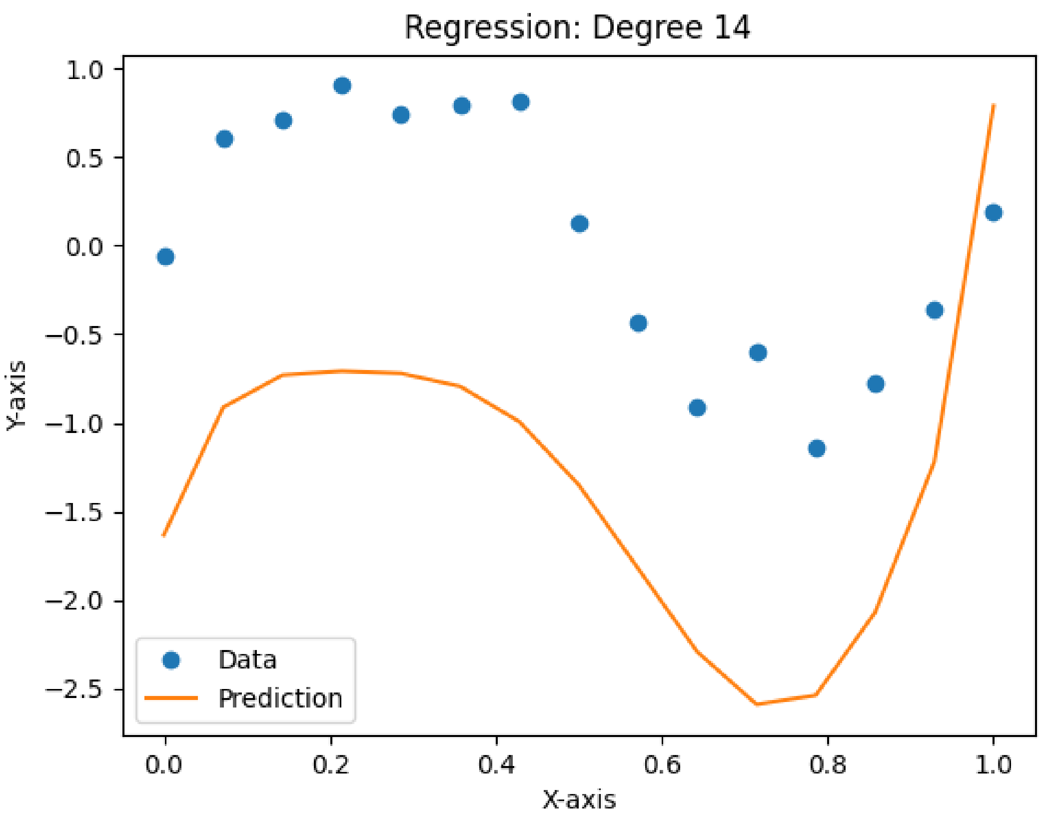


- **綜合數據：**

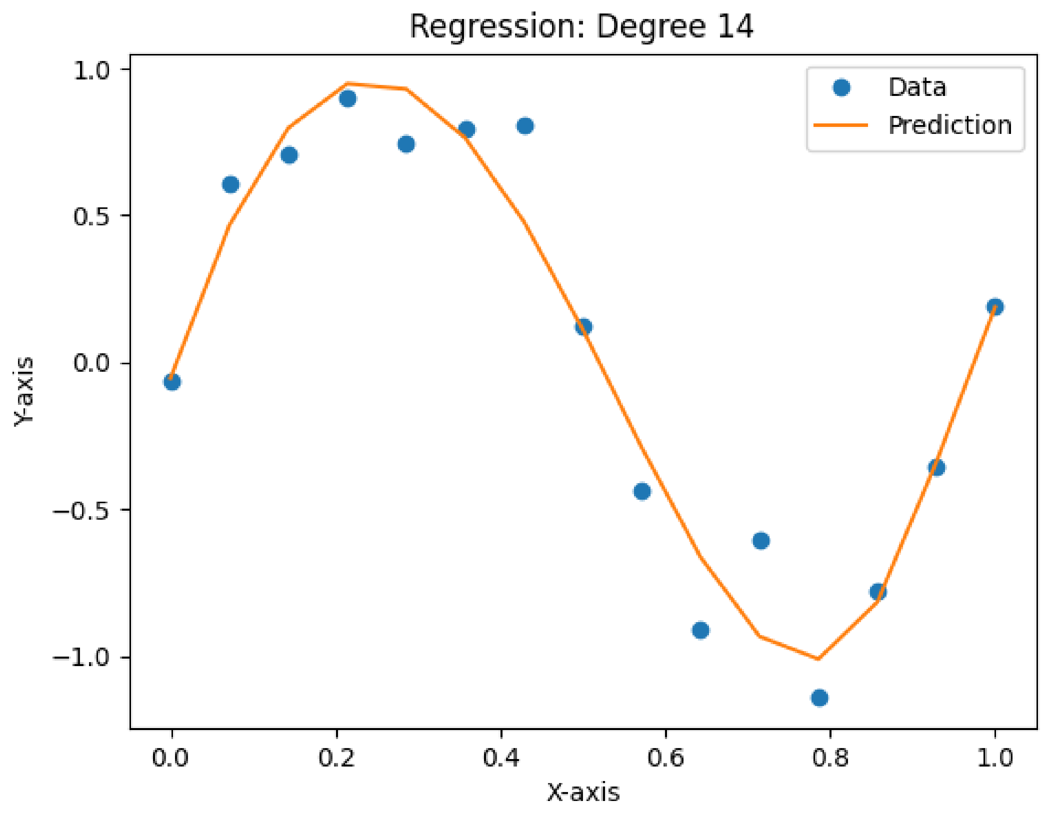
```
Total data point = 10
DEGREE --> 14
train_mse_loss: 2.323156877537258e-05
kfold_avg_loss: 5714.505616638065
Total data point = 15
DEGREE --> 14
train_mse_loss: 0.0011635350529104471
kfold_avg_loss: 98.68086846759543
Total data point = 80
DEGREE --> 14
train_mse_loss: 0.03996386379003525
kfold_avg_loss: 1.1314146004617214
Total data point = 320
DEGREE --> 14
train_mse_loss: 0.04302976652979851
kfold_avg_loss: 340.71082664839923
```

- 題目 6 實驗結果：

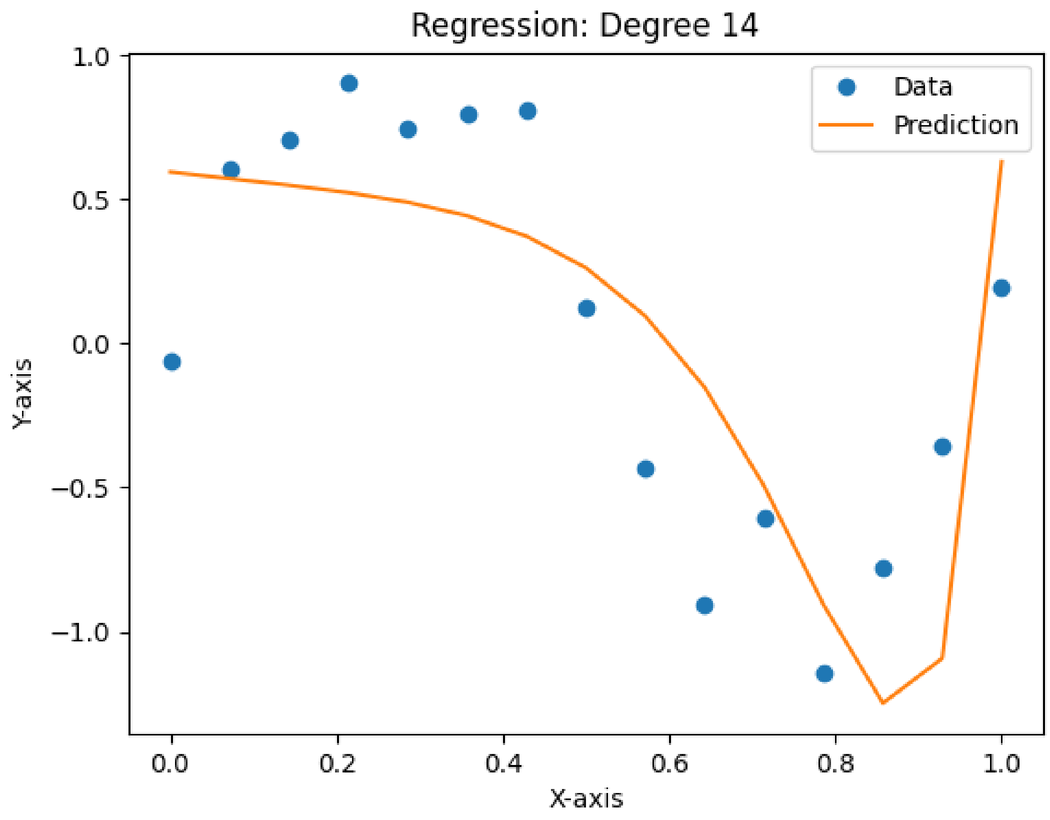
- 一樣使用 15 組資料，且使用 14 次多項式來做 regression ，不過這次加上了 regularization ，並且嘗試採用不同的 λ 來比對。
- 結果截圖如下：
  - $\lambda = 0$ ：



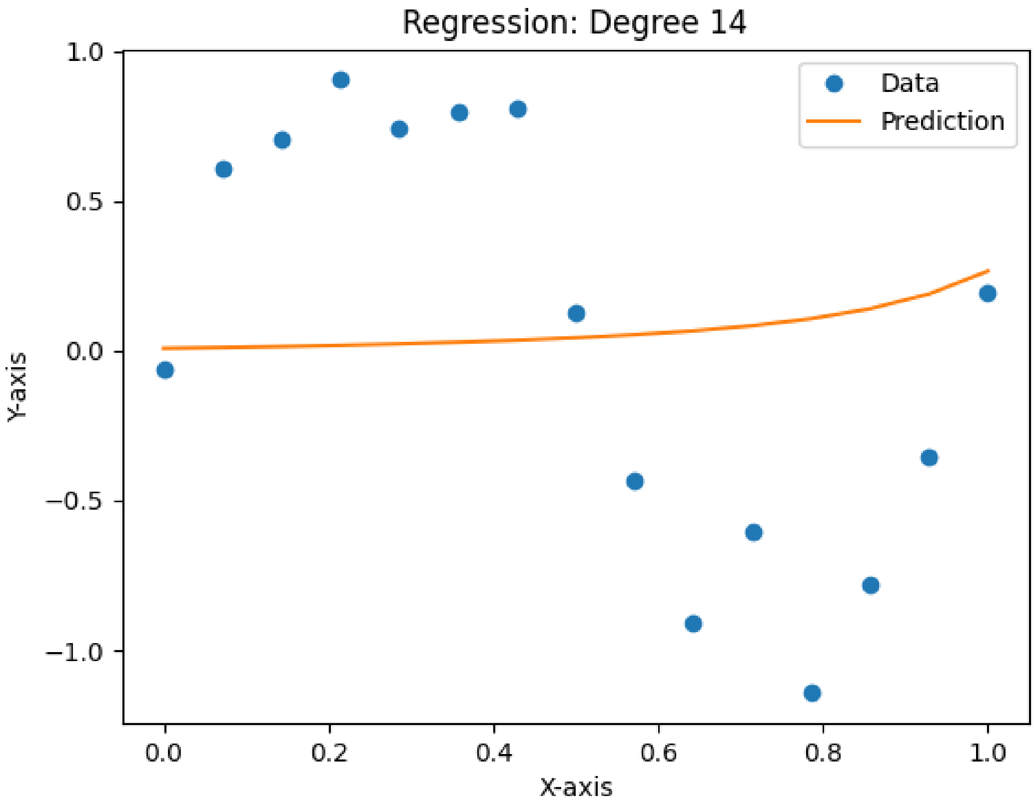
  - $\lambda = 0.001/15$ ：



- $\lambda = 1/15$ ：



- $\lambda = 1000/15$ ：



- **綜合數據：**

```
Lambda: 0
DEGREE --> 14
train_mse_loss: 2.1365323066711426
--> Kfold average MSE Loss : 35423394.4100524
Lambda: 6.666666666666667e-05
DEGREE --> 14
train_mse_loss: 0.025632511824369 43
--> Kfold average MSE Loss : 8.260862489044666
Lambda: 0.06666666666666667
DEGREE --> 14
train_mse_loss: 0.1919802874326706
--> Kfold average MSE Loss : 16.5090371966362
Lambda: 66.66666666666667
DEGREE --> 14
train_mse_loss: 0.5142942070960999
--> Kfold average MSE Loss : 0.4201230525970459
```

結果觀察 (Conclusion)

- 題目 1 結果觀察：
  - 15 筆 sample data 。
- 題目 2 結果觀察：

    - 一條合適的直線。

- 題目 3 結果觀察：
    - 次數越高，模型越複雜， train loss 降低，但是 kfold cross validation loss 會增高。
- 題目 4 結果觀察：
    - 這個模型比起第 1 題的模型，在使用高次多像式 (5, 10, 14) 時 train loss 又會更低。
    - kfold cross validation loss 一樣隨著模型複雜越來越大，但是又比第 1 題的模型還要小，這可能是因為這題產生數據的模型較複雜。
- 題目 5 結果觀察：
    - 10 個點的時候 train loss 很小。
    - 80 個點的時候 kfold cross validation loss 相較於其他實驗來說很小。
    - 隨著點的數量上升， kfold cross validation loss 的值下降，但是又在點的數量為 320 時上升。
- 題目 6 結果觀察：
    - 隨著 λ 的增大，曲線會越接近平滑。
    - 使用 λ 之後， kfold cross validation loss 的值在使用某些特定的 λ 值會降低。

相關討論 (Disscussion)

- 實作的過程中發現有許多重複的程式碼，故我把一些重複地方的寫成 function 重複利用，縮短冗長程式碼，增加 debug 效率。

程式碼 (Code)

```
import torch
from torch import nn
import numpy as np
import math
import matplotlib.pyplot as plt
import code
# code.interact(local=locals())
from sklearn.model_selection import KFold

def poly_train_mse_loss(y_init, y_hat):
    loss_fn = nn.MSELoss()
    train_err = loss_fn(y_init, y_hat)
    # print("Train error (MSE loss) :", train_err.item())
    return train_err.item()

def poly_predict_y_hat(x_values, w_lin):
    y_hat = torch.matmul(x_values, w_lin)
    return y_hat
```

```python
def poly_train_w(x_values, y_values):
    x_pinv = torch.linalg.pinv(x_values)
    w_lin = torch.matmul(x_pinv, y_values)
    # print("w_lin:", w_lin)
    return w_lin

def poly_kfold_cross_validation(current_degree, n_splits, x_values, y_values,
loss_fn):
    # poly KFOLD cross validation
    kfold = KFold(n_splits=n_splits)
    kfold_loss_list = []
    xy_values = torch.cat((x_values, y_values), dim=1)
    for fold_i, (train_ids, val_ids) in enumerate(kfold.split(xy_values)):

        # print("Fold Info:")
        # print(fold_i, (train_ids, val_ids))

        x_values_train = xy_values[:, :current_degree+1][train_ids]
        y_values_train = xy_values[:, current_degree+1][train_ids].unsqueeze(1)

        x_values_val = xy_values[:, :current_degree+1][val_ids]
        y_values_val = xy_values[:, current_degree+1][val_ids].unsqueeze(1)

        # w
        w_lin = poly_train_w(x_values_train, y_values_train)

        # predict
        y_hat = poly_predict_y_hat(x_values_val, w_lin)

        # loss
        kfold_loss = loss_fn(y_values_val, y_hat)
        kfold_loss_list.append(kfold_loss)

    return sum(kfold_loss_list) / len(kfold_loss_list)

def poly_regression_plot(current_degree, x_init, y_init, y_hat):
    # Plot dots
    plt.plot(x_init, y_init, 'o', label='Data')

    # Plot line
    plt.plot(x_init, y_hat, '-', label='Prediction')

    # Add legend
    plt.legend()

    # Add labels and title
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Regression: Degree ' + str(current_degree))

    # Display the plot
    plt.show()

def poly_regression(current_degree, x_init, y_init, n_splits, loss_fn):
```

```
    x_values = torch.ones(n_points, current_degree + 1)
    for i in range(current_degree):
        x_values[:, i] = x_init ** (current_degree - i)
    y_values = y_init.unsqueeze(1)

    x_pinv = torch.linalg.pinv(x_values)
    w_lin = torch.matmul(x_pinv, y_values)
    # print("w_lin:", w_lin)

    y_hat = poly_predict_y_hat(x_values, w_lin)

    poly_regression_plot(current_degree, x_init, y_init, y_hat)

    train_mse_loss = loss_fn(y_values, y_hat)
    # print("--> MSE Loss : " + str(train_err))

    kfold_avg_loss = poly_kfold_cross_validation(
        current_degree=current_degree,
        n_splits=n_splits,
        x_values=x_values,
        y_values=y_values,
        loss_fn=loss_fn
    )
    # print("--> Kfold average MSE Loss : " + str(kfold_avg_loss))
    return train_mse_loss, kfold_avg_loss

def poly_regu(degree, lambd, x_values, y_values):
    A = torch.matmul(x_values.transpose(0, 1), x_values)
    B = torch.inverse(A - lambd * torch.eye(degree+1))
    w_reg = torch.matmul(torch.matmul(B, x_values.transpose(0, 1)), y_values)
    return w_reg

# y = 2 * x + epsilon

n_points = 15
x_min, x_max = -3, 3
x_init = torch.linspace(x_min, x_max, n_points)
epsilon = torch.randn(n_points)
y_init = 2 * x_init + epsilon

print("x_init:", x_init)
print("y_init:", y_init)


degree = [1, 5, 10, 14]
n_splits_list = [5, 5, 5, 5]

for d in range(len(degree)):
    print("DEGREE --> " + str(degree[d]))
    train_mse_loss, kfold_avg_loss = poly_regression(
        degree[d],
        x_init,
        y_init,
```

```
        n_splits_list[d],
        poly_train_mse_loss
    )
    print("train_mse_loss:", train_mse_loss)
    print("kfold_avg_loss:", kfold_avg_loss)

# y = sin(2*pi) + epsilon

n_points = 15
x_min, x_max = 0, 1
x_init = torch.linspace(x_min, x_max, n_points)
epsilon = torch.randn(n_points) * math.sqrt(0.04)
y_init = torch.sin(2 * math.pi * x_init) + epsilon

print("x_init:", x_init)
print("y_init:", y_init)

degree = [1, 5, 10, 14]
n_splits_list = [5, 5, 5, 5]

for d in range(len(degree)):
    print("DEGREE --> " + str(degree[d]))
    train_mse_loss, kfold_avg_loss = poly_regression(
        degree[d],
        x_init,
        y_init,
        n_splits_list[d],
        poly_train_mse_loss
    )
    print("train_mse_loss:", train_mse_loss)
    print("kfold_avg_loss:", kfold_avg_loss)

# varying y = sin(2*pi) + epsilon 's data point
# m(n_points) = 10, 15, 80, 320

degree = [14]
n_splits_list = [5]
n_points_list = [10, 15, 80, 320]
for i in range(len(n_points_list)):

    print("Total data point = " + str(n_points_list[i]))

    n_points = n_points_list[i]
    x_min, x_max = 0, 1
    x_init = torch.linspace(x_min, x_max, n_points)
    epsilon = torch.randn(n_points) * math.sqrt(0.04)
    y_init = torch.sin(2 * math.pi * x_init) + epsilon

    # print("x_init:", x_init)
    # print("y_init:", y_init)

    for d in range(len(degree)):
        print("DEGREE --> " + str(degree[d]))
        train_mse_loss, kfold_avg_loss = poly_regression(
```

```
            degree[d],
            x_init,
            y_init,
            n_splits_list[d],
            poly_train_mse_loss
        )
        print("train_mse_loss:", train_mse_loss)
        print("kfold_avg_loss:", kfold_avg_loss)

# regu

n_points = 15
x_min, x_max = 0, 1
x_init = torch.linspace(x_min, x_max, n_points)
epsilon = torch.randn(n_points) * math.sqrt(0.04)
y_init = torch.sin(2 * math.pi * x_init) + epsilon

degree = [14]
n_splits_list = [5]
lambda_list = [0, 0.001 / n_points, 1 / n_points, 1000 / n_points]

for l in range(len(lambda_list)):

    print("Lambda:", lambda_list[l])

    for d in range(len(degree)):

        print("DEGREE --> " + str(degree[d]))

        current_degree = degree[d]

        x_values = torch.ones(n_points, current_degree + 1)
        for i in range(current_degree):
            x_values[:, i] = x_init ** (current_degree - i)
        y_values = y_init.unsqueeze(1)

        w_reg = poly_regu(current_degree, lambda_list[l], x_values, y_values)

        # x_pinv = torch.linalg.pinv(x_values)
        # w_lin = torch.matmul(x_pinv, y_values)
        # print("w_lin:", w_lin)

        y_hat = poly_predict_y_hat(x_values, w_reg)

        poly_regression_plot(current_degree, x_init, y_init, y_hat)

        train_mse_loss = poly_train_mse_loss(y_values, y_hat)
        print("train_mse_loss:", train_mse_loss)

        # poly KFOLD cross validation
        kfold = KFold(n_splits=n_splits_list[d])
        kfold_loss_list = []
        xy_values = torch.cat((x_values, y_values), dim=1)
        for fold_i, (train_ids, val_ids) in enumerate(kfold.split(xy_values)):
```

```
            # print("Fold Info:")
            # print(fold_i, (train_ids, val_ids))

            x_values_train = xy_values[:, :current_degree+1][train_ids]
            y_values_train = xy_values[:, current_degree+1]
[train_ids].unsqueeze(1)

            x_values_val = xy_values[:, :current_degree+1][val_ids]
            y_values_val = xy_values[:, current_degree+1][val_ids].unsqueeze(1)

            # w
            w_reg = poly_regu(current_degree, lambda_list[l], x_values_train,
y_values_train)

            # predict
            y_hat = poly_predict_y_hat(x_values_val, w_reg)

            # loss
            kfold_loss = poly_train_mse_loss(y_values_val, y_hat)
            kfold_loss_list.append(kfold_loss)

        print("--> Kfold average MSE Loss : " + str(sum(kfold_loss_list) /
len(kfold_loss_list)))
```