《模式识别与机器学习 A》实验报告

实验题目：多项式拟合正弦函数

班级：2203601

学号：2022113416

姓名：刘子康

实验报告内容

一、实验目的

- 基于梯度下降法，学会使用高阶多项式函数拟合正弦函数；
- 掌握机器学习训练拟合原理（无惩罚项的损失函数）；
- 掌握加惩罚项（L2 范数）的损失函数优化、梯度下降法、理解过拟合、克服过拟合的方法（如加惩罚项、增加样本等）。

二、实验内容

生成一组某正弦函数上、添加了噪声的散点样本，尝试利用梯度下降法作为优化方法，使用高阶多项式函数拟合该正弦函数。比较不同数据量、不同超参数和不同多项式阶数下的拟合结果，解释现象原因，分析影响拟合效果的因素。

若模型出现过拟合现象，尝试通过添加和调整惩罚项、增加样本数量、调整超参数等方法克服过拟合问题，提高模型泛化能力。

三、实验环境

- 操作系统：Windows 11
- 编程语言：Python 3.10
- 第三方库：Numpy 1.23.4，Matplotlib 3.8.2
- IDE：Pycharm 2022 社区版

四、实验过程、结果及分析

4.1 实验原理

多项式函数拟合基于数学中的多项式逼近方法，它通过构造一个形如$f(x) = a_0 + a_1x + a_2x^2 \cdots + a_nx^n$的多项式函数，来逼近已知数据点集所代表的真实函数或关系，即使得这些数据点到该多项式函数的距离（或误差）之和最小。

梯度下降法是一种一阶迭代优化算法，其核心原理是利用负梯度方向作为搜索方向，因为在多元函数的某一点处，函数值沿着负梯度方向下降最快。当使用均方误差作为损失函数时，梯度下降法从某个初始点开始，通过迭代地计算损失函数关于模型参数的梯度，并按照梯度的反方向（即负梯度方向）更新参数，从而使得损失函数值逐渐减小，直至达到一个局部最小值或全局最小值。

4.2 实验过程

4.2.1 生成样本点

使用 Numpy 库的 linspace(-1, 1, num)函数生成(-1, 1)区间的 num 个 x 坐标点（num 由自己指定），再使用 np.sin(np.pi * x)函数生成对应的函数值序列 y，作为原始正弦函数$y = sin(\pi x)\ (-1, 1)$。对生成的 y 添加均值为 0、标准差为 0.005 的高斯噪声，得到 y_noise，作为多项式拟合的真实值。

```
    # 正弦函数
    x = np.linspace(-1, 1, sample_num)
    y = np.sin(x * np.pi)

    # 高斯噪声
    sigma = 0.05
    y_noise = y + np.random.normal(0, sigma, y.shape)
```

用上述方法，将训练集向两侧扩充 1.2 倍，作为测试集，用于测试拟合效果。

```
    # 测试集（训练集两侧扩充）
    x_test = np.linspace(-1.2, 1.2, int(sample_num * 1.2))
    y_test = np.sin(x_test * np.pi)
    y_test_noise = y_test + np.random.normal(0, sigma, y_test.shape)
    y_pred_all = []      # 不同阶数的拟合函数
```

4.2.2 定义多项式函数、损失函数和梯度计算函数

根据输入的最高阶数生成一个形如$y = ax^2 + bx + c$的多项式函数 polynomial()，返回带入参数 x 向量计算后的函数值 y。

```
# 多项式函数，degree为最高阶数
def polynomial(x, coef, degree):
    y = 0
    for i in range(degree + 1):
        y += coef[i] * x ** i
    return y
```

损失函数 loss()使用均方误差（$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$，$y_i$为第 i 个样本的真实值，$\hat{y}_i$为第 i 个样本的预测值），通过计算预测值和真实值之间的误差平方和的均值作为差异，来衡量模型的预测性能。均方误差越小，表示模型的预测越准确；均方误差越大，表示模型的预测误差越大。

```
# 损失函数MSE，flag表示是否加入正则化项
def loss(y_true, y_pred, flag=False):
    if flag:
        return np.mean((y_true - y_pred) ** 2) + np.sum(lamda * coef ** 2)
    return np.mean((y_true - y_pred) ** 2)
```

每次迭代过程的下降梯度通过损失函数对系数向量 coef 求偏导得到（$grad = -2 * \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i}) * x^i$），梯度计算函数为 gradient()。

```
# 计算梯度，flag表示是否加入正则化项
def gradient(x, y_true, y_pred, degree, flag=False):
    grad = []
    if flag:
        for i in range(degree + 1):
            grad.append(-2 * np.sum((y_true - y_pred) * x ** i) / len(x) + 2 * lamda * coef[i])
    else:
        for i in range(degree + 1):
            grad.append(-2 * np.sum((y_true - y_pred) * x ** i) / len(x))
    return np.array(grad)
```

4.2.3 训练过程

设置迭代次数 epochs 为 10000，确定生成数据量，对于不同的多项式阶数，通过多项式函数 polynomial()计算预测值 y_fit，之后通过 loss()函数和 gradient()函数计算当前 loss 值和下降梯度，并进入下一次迭代。

可以选择有惩罚项或无惩罚项，若选择有惩罚项，则在计算 loss 和梯度时加入正则化

项，loss 为均方误差加上$\lambda * \|coef\|_2^2$，其中 λ 为惩罚项系数，$\|coef\|_2^2$为系数向量的 L2 范数的平方；梯度为对系数向量 coef 的偏导数加上$2 * \lambda * coef$。

```
    # 梯度下降法求解系数
    epochs = 10000  # 迭代次数
    lr = 0.1        # 学习率
    flag = False    # 是否加入惩罚项
    lamda = 0.003   # 惩罚项系数
    y_fit = np.zeros(sample_num)

    for degree in degrees:  # 不同阶数的拟合
        coef = np.zeros(degree + 1)
        for epoch in range(epochs):
            y_fit = polynomial(x, coef, degree)                    # 多项式拟合函数
            loss_value = loss(y_noise, y_fit, flag)                # 计算loss值
            coef -= lr * gradient(x, y_noise, y_fit, degree, flag)  # 梯度下降

            if epoch % 2000 == 0:
                print(f"Epoch {epoch}, Loss: {loss_value}")

        y_pred = polynomial(x_test, coef, degree)
        y_pred_all.append(y_pred)
```

4.2.4 输出结果并绘图

当拟合完成后，对于不同的多项式阶数，分别输出拟合函数和最终误差。

```
    print(f"阶数为{degree}时，拟合函数为：y = {coef[degree]:.3f}x^{degree}", end=' ')
    for i in range(degree-1, 1, -1):
        if coef[i] < 0:
            print(f"- {-coef[i]:.3f}x^{i}", end=' ')
        else:
            print(f"+ {coef[i]:.3f}x^{i}", end=' ')
    if coef[-2] < 0:
        print(f"- {-coef[-2]:.3f}x", end=' ')
    else:
        print(f"+ {coef[-2]:.3f}x", end=' ')
    if coef[-1] < 0:
        print(f"- {-coef[-1]:.3f}")
    else:
        print(f"+ {coef[-1]:.3f}")
    print(f"最终误差为：{loss(y_noise, y_fit, flag):.6f}")
    print("-----------------------------------------------------------------------------------")
```

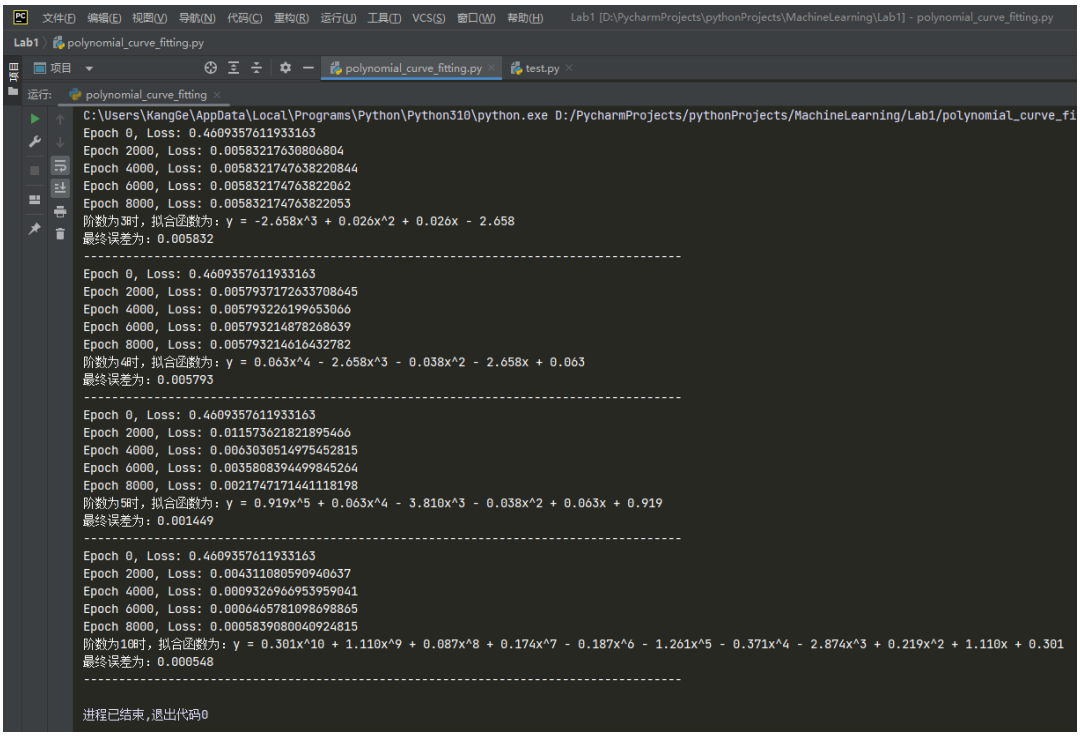
利用训练集向两侧扩充 1.2 倍后的样本作为测试集，使用 Matplotlib 库的 pyplot 模块绘制图像，将拟合函数与原始函数和加噪声后的样本点进行对比，分析拟合效果，判断是否发生过拟合现象。

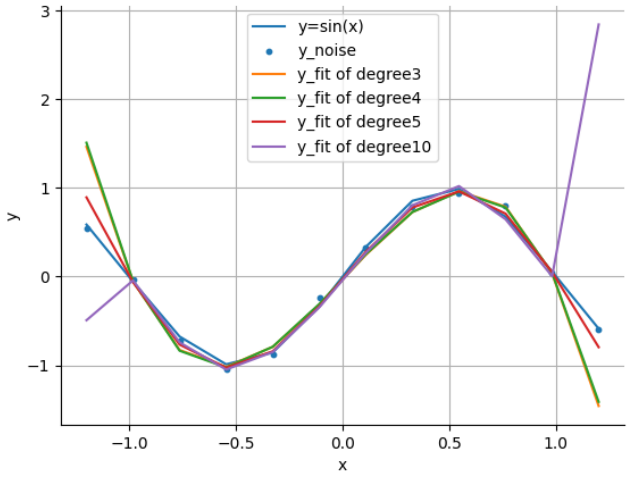
```
    # 绘图
    fig, ax = plt.subplots()
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)

    plt.plot(x_test, y_test, label='y=sin(x)')                 # 原始函数
    plt.scatter(x_test, y_test_noise, s=10, label='y_noise')   # 加入高斯噪声的样本点
    for degree, y_pred in zip(degrees, y_pred_all):            # 不同阶数的拟合函数
        plt.plot(x_test, y_pred, label='y_fit of degree' + str(degree))
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.show()
```

4.3 实验结果及分析

4.3.1 样本量为 10

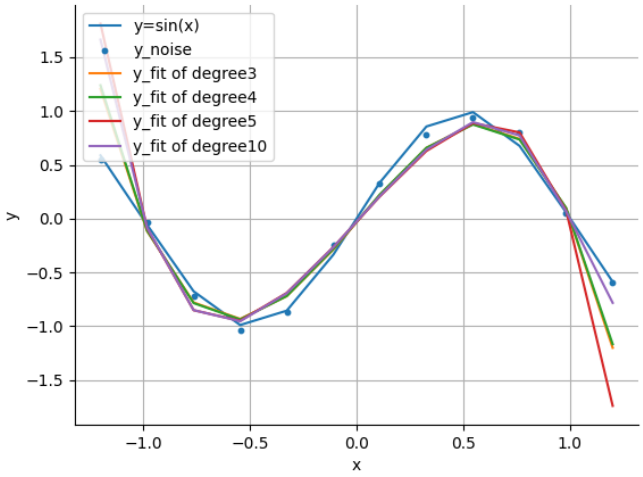
- 无惩罚项





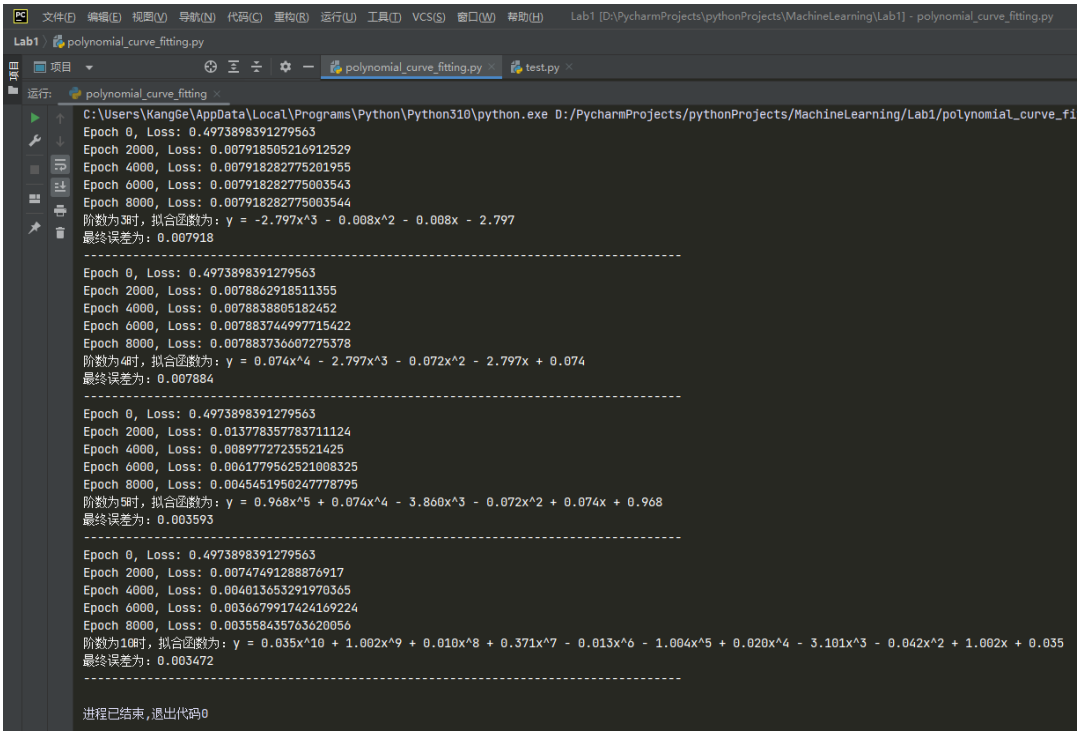
- 有惩罚项

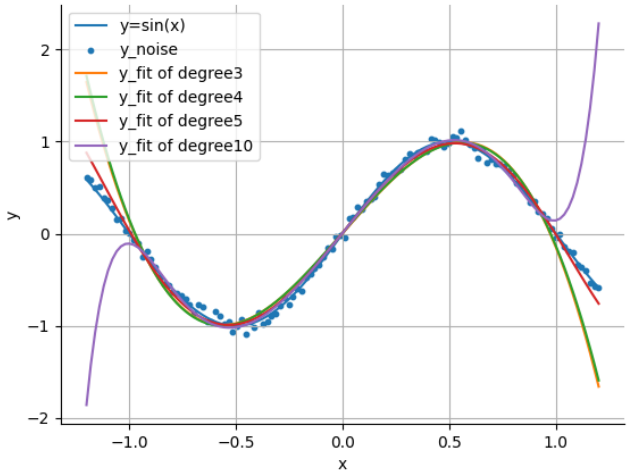
```
C:\Users\KangGe\AppData\Local\Programs\Python\Python310\python.exe D:/PycharmProjects/pythonProjects/MachineLearning/Lab1/polynomial_curve_fi
Epoch 0, Loss: 0.4609357611933163
Epoch 2000, Loss: 0.04287990591781299
Epoch 4000, Loss: 0.04287990579485871
Epoch 6000, Loss: 0.04287990579485871 6
Epoch 8000, Loss: 0.04287990579485871 6
阶数为3时，拟合函数为：y = -2.333x^3 + 0.025x^2 + 0.025x - 2.333
最终误差为：0.042880
---------------------------------------------------------------------------------
Epoch 0, Loss: 0.4609357611933163
Epoch 2000, Loss: 0.042848383126114234
Epoch 4000, Loss: 0.04284835538145096
Epoch 6000, Loss: 0.0428483553236709
Epoch 8000, Loss: 0.042848355323550025
阶数为4时，拟合函数为：y = 0.045x^4 - 2.333x^3 - 0.019x^2 - 2.333x + 0.045
最终误差为：0.042848
---------------------------------------------------------------------------------
Epoch 0, Loss: 0.4609357611933163
Epoch 2000, Loss: 0.0371455233786697
Epoch 4000, Loss: 0.0369422829631235
Epoch 6000, Loss: 0.03693277111895693
Epoch 8000, Loss: 0.03693232590040983
阶数为5时，拟合函数为：y = -0.771x^5 + 0.045x^4 - 1.433x^3 - 0.019x^2 + 0.045x - 0.771
最终误差为：0.036932
---------------------------------------------------------------------------------
Epoch 0, Loss: 0.4609357611933163
Epoch 2000, Loss: 0.036439401215959046
Epoch 4000, Loss: 0.036268123661543514
Epoch 6000, Loss: 0.036266889574523606
Epoch 8000, Loss: 0.036266869606152534
阶数为10时，拟合函数为：y = 0.090x^10 + 0.298x^9 + 0.036x^8 - 0.201x^7 - 0.033x^6 - 0.895x^5 - 0.088x^4 - 1.422x^3 + 0.028x^2 + 0.298x + 0.090
最终误差为：0.036267
---------------------------------------------------------------------------------

进程已结束,退出代码0
```



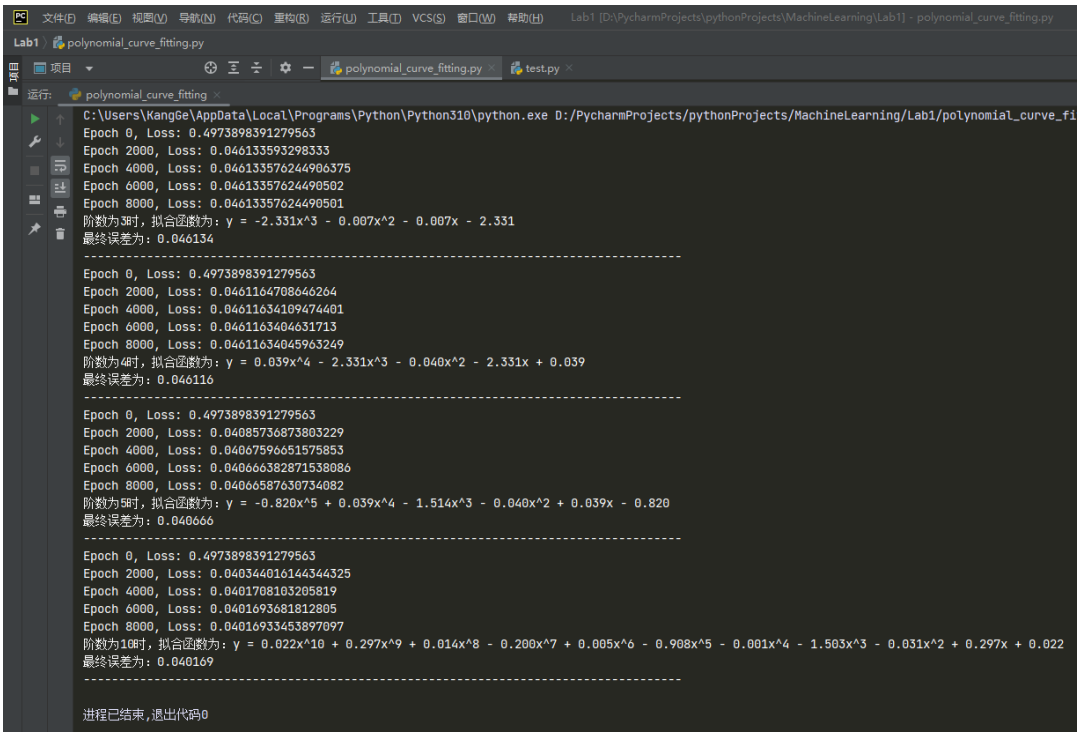
4.3.2 样本量为 100

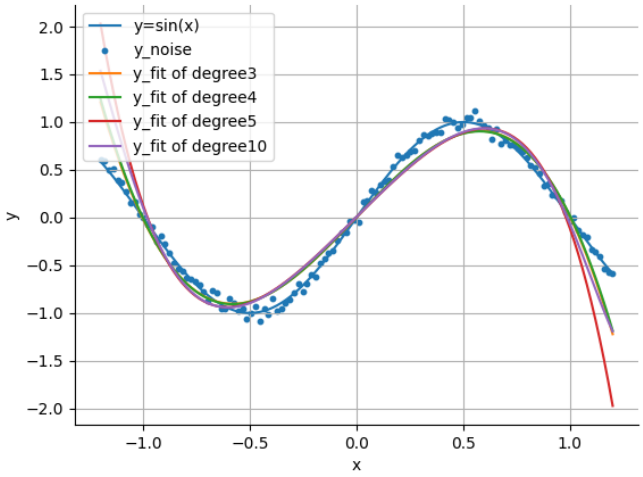
- 无惩罚项





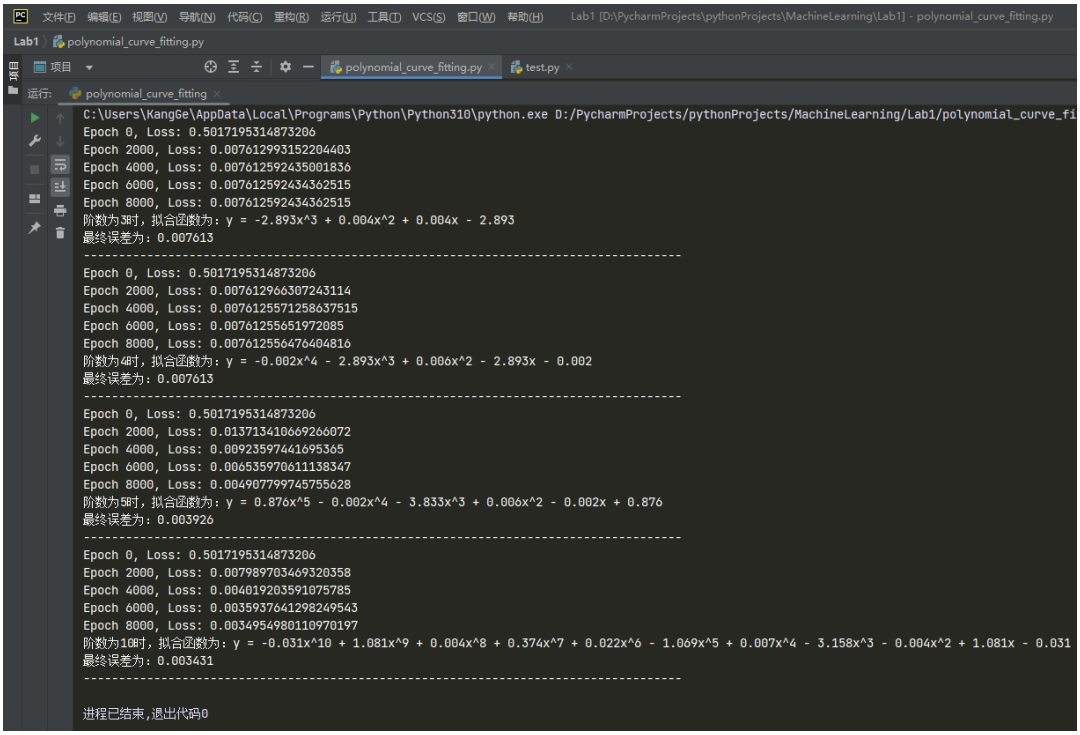
- 有惩罚项

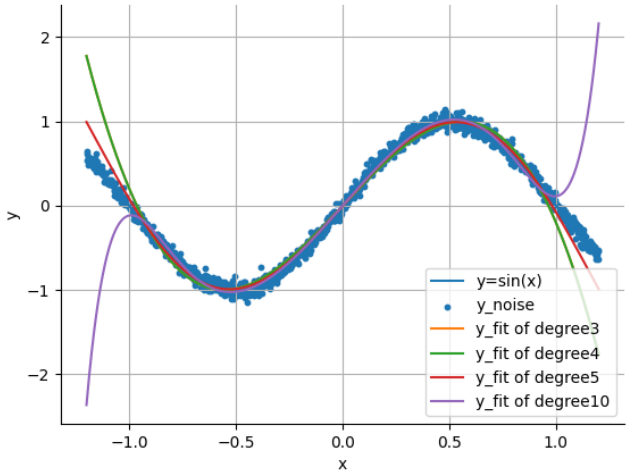

C:\Users\KangGe\AppData\Local\Programs\Python\Python310\python.exe D:/PycharmProjects/pythonProjects/MachineLearning/Lab1/polynomial_curve_fi
Epoch 0, Loss: 0.4973898391279563
Epoch 2000, Loss: 0.046133593298333
Epoch 4000, Loss: 0.04613357624496375
Epoch 6000, Loss: 0.04613357624490502
Epoch 8000, Loss: 0.04613357624490501
阶数为3时，拟合函数为：y = -2.331x^3 - 0.007x^2 - 0.007x - 2.331
最终误差为：0.046134
Epoch 0, Loss: 0.4973898391279563
Epoch 2000, Loss: 0.04611647086462 64
Epoch 4000, Loss: 0.04611634109474401
Epoch 6000, Loss: 0.0461163404631713
Epoch 8000, Loss: 0.04611634045963249
阶数为4时，拟合函数为：y = 0.039x^4 - 2.331x^3 - 0.040x^2 - 2.331x + 0.039
最终误差为：0.046116
Epoch 0, Loss: 0.4973898391279563
Epoch 2000, Loss: 0.04085736873803229
Epoch 4000, Loss: 0.04067596651575853
Epoch 6000, Loss: 0.04066638287153808 6
Epoch 8000, Loss: 0.04066587630734082
阶数为5时，拟合函数为：y = -0.820x^5 + 0.039x^4 - 1.514x^3 - 0.040x^2 + 0.039x - 0.820
最终误差为：0.040666
Epoch 0, Loss: 0.4973898391279563
Epoch 2000, Loss: 0.040344016144344325
Epoch 4000, Loss: 0.0401708103205819
Epoch 6000, Loss: 0.0401693681812805
Epoch 8000, Loss: 0.04016933453897097
阶数为10时，拟合函数为：y = 0.022x^10 + 0.297x^9 + 0.014x^8 - 0.200x^7 + 0.005x^6 - 0.908x^5 - 0.001x^4 - 1.503x^3 - 0.031x^2 + 0.297x + 0.022
最终误差为：0.040169
进程已结束,退出代码0


y=sin(x)
y_noise
y_fit of degree3
y_fit of degree4
y_fit of degree5
y_fit of degree10
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−1.0
−0.5
0.0
0.5
1.0
x

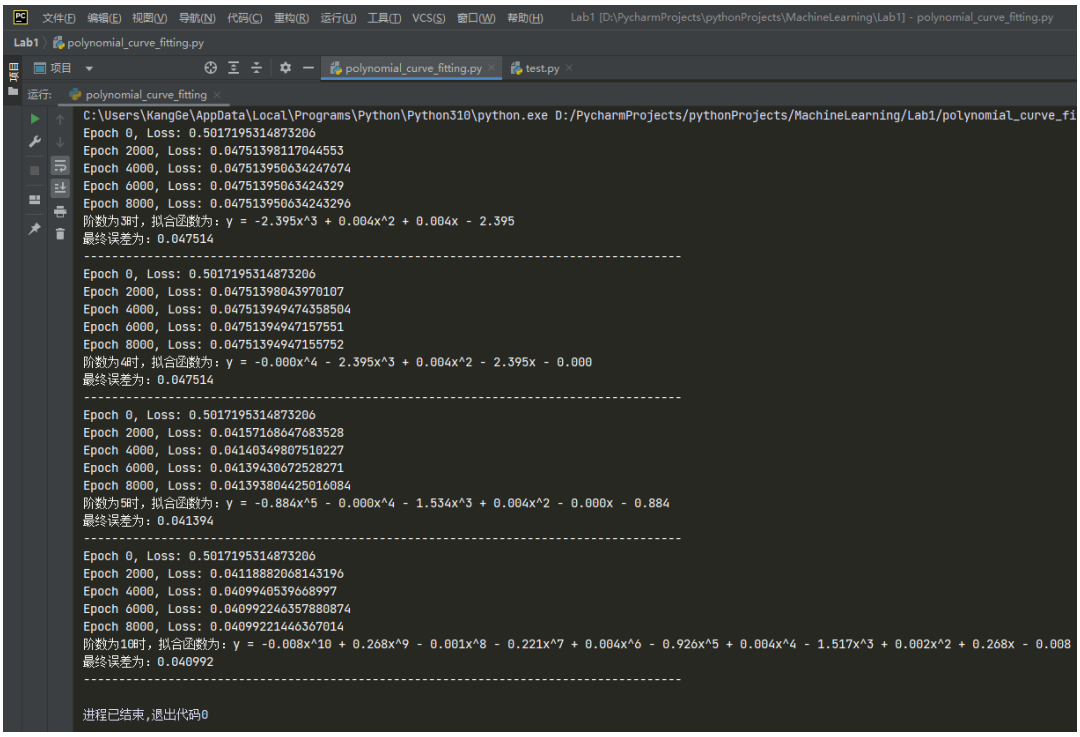
4.3.3 样本量为 1000

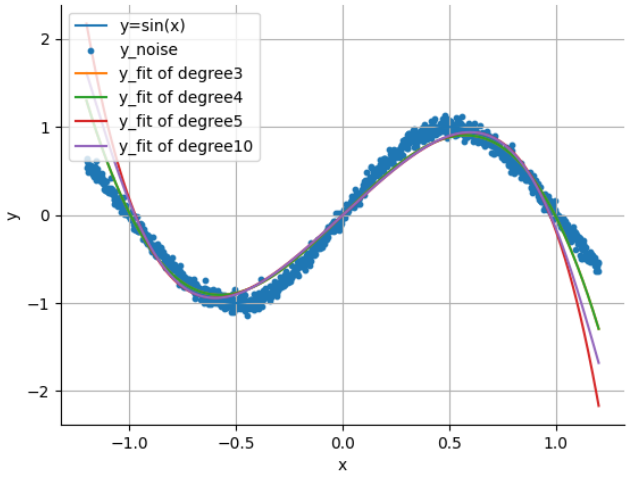
- 无惩罚项





• 有惩罚项





五、实验总体结论

在进行基于梯度下降法优化的多项式函数拟合局部正弦函数过程中，通过调整合适的数据量、学习率以及惩罚项系数等超参数，实现了较好的拟合效果。

过拟合指模型复杂度高于实际问题，模型在训练集上表现很好，但在测试集上却表现很差，即无法将从训练集学习到的特征泛化至测试集。根据实验结果，当使用高阶多项式进行拟合时（例如 10 阶），会出现明显过拟合现象，这时可以通过加入惩罚项或增加数据量来抑制和解决。实验中惩罚项使用 L2 范数是合理的，因为生成的噪声符合高斯分布，若高斯分布取到最大似然，同时 L2 损失函数也会最小。

然而，拟合结果显示加入惩罚项虽然抑制了过拟合问题，但也会使得拟合效果变差，需要合理调整惩罚项系数和多项式阶数，使拟合效果和防过拟合能力达到较好的平衡。

六、完整实验代码

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  np.random.seed(16)  # 设置随机数种子，便于结果复现
5.
6.  # 多项式函数，degree 为最高阶数
7.  def polynomial(x, coef, degree):
8.      y = 0
9.      for i in range(degree + 1):
10.         y += coef[i] * x ** i
11.     return y
12.
13. # 损失函数MSE，flag 表示是否加入正则化项
14. def loss(y_true, y_pred, flag=False):
15.     if flag:
16.         return np.mean((y_true - y_pred) ** 2) + np.sum(lamda * coef ** 2)
17.     return np.mean((y_true - y_pred) ** 2)
18.
19. # 计算梯度，flag 表示是否加入正则化项
20. def gradient(x, y_true, y_pred, degree, flag=False):
21.     grad = []
22.     if flag:
23.         for i in range(degree + 1):
24.             grad.append(-
    2 * np.sum((y_true - y_pred) * x ** i) / len(x) + 2 * lamda * coef[i])
25.     else:
26.         for i in range(degree + 1):
27.             grad.append(-2 * np.sum((y_true - y_pred) * x ** i) / len(x))
28.     return np.array(grad)
29.
30.
31. if __name__ == '__main__':
32.     # 样本量，阶数
33.     sample_num = 10
34.     degrees = [3, 4, 5, 10]
35.
36.     # 正弦函数
37.     x = np.linspace(-1, 1, sample_num)
38.     y = np.sin(x * np.pi)
39.
40.     # 高斯噪声
41.     sigma = 0.05
```

```
    y_noise = y + np.random.normal(0, sigma, y.shape)

    # 测试集（训练集两侧扩充）
    x_test = np.linspace(-1.2, 1.2, int(sample_num * 1.2))
    y_test = np.sin(x_test * np.pi)
    y_test_noise = y_test + np.random.normal(0, sigma, y_test.shape)
    y_pred_all = []     # 不同阶数的拟合函数

    # 梯度下降法求解系数
    epochs = 10000  # 迭代次数
    lr = 0.1        # 学习率
    flag = False    # 是否加入惩罚项
    lamda = 0.003   # 惩罚项系数
    y_fit = np.zeros(sample_num)

    for degree in degrees:  # 不同阶数的拟合
        coef = np.zeros(degree + 1)
        for epoch in range(epochs):
            y_fit = polynomial(x, coef, degree)          # 多项式拟合函数
            loss_value = loss(y_noise, y_fit, flag)      # 计算 loss 值
            coef -= lr * gradient(x, y_noise, y_fit, degree, flag)  # 梯度下降

            if epoch % 2000 == 0:
                print(f"Epoch {epoch}, Loss: {loss_value}")

        y_pred = polynomial(x_test, coef, degree)
        y_pred_all.append(y_pred)

        print(f"阶数为{degree}时，拟合函数为:
  y = {coef[degree]:.3f}x^{degree}", end=' ')
        for i in range(degree-1, 1, -1):
            if coef[i] < 0:
                print(f"- {-coef[i]:.3f}x^{i}", end=' ')
            else:
                print(f"+ {coef[i]:.3f}x^{i}", end=' ')
        if coef[-2] < 0:
            print(f"- {-coef[-2]:.3f}x", end=' ')
        else:
            print(f"+ {coef[-2]:.3f}x", end=' ')
        if coef[-1] < 0:
            print(f"- {-coef[-1]:.3f}")
        else:
            print(f"+ {coef[-1]:.3f}")
        print(f"最终误差为: {loss(y_noise, y_fit, flag):.6f}")
```

```
        print("--------------------------------------------------------------------
    ----------------------")

    # 绘图
    fig, ax = plt.subplots()
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)

    plt.plot(x_test, y_test, label='y=sin(x)')              # 原始函数
    plt.scatter(x_test, y_test_noise, s=10, label='y_noise')# 加高斯噪声的样本点
    for degree, y_pred in zip(degrees, y_pred_all):         # 不同阶数的拟合函数
        plt.plot(x_test, y_pred, label='y_fit of degree' + str(degree))
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
      plt.show()
```

七、参考文献

[1] 刘远超. 深度学习基础: 高等教育出版社, 2023.