《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实验

学号：

姓名：

实验报告内容

1．实验目的

本实验的目的是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)。本实验要求使用高阶多项式函数拟合正弦函数曲线，并用梯度下降法求解最优解。同时，要用不同数据量，不同超参数，不同的多项式阶数，比较实验效果，并用实验数据解释过拟合现象。

2．实验内容

1. 生成数据，加入噪声；
2. 用高阶多项式函数拟合曲线（建议正弦函数曲线）；
3. 优化方法求解最优解（梯度下降）；
4. 用你得到的实验数据，解释过拟合。
5. 用不同数据量，不同超参数，不同的多项式阶数，比较实验效果。

3．实验环境

Windows 10 ；python 3.9.7；jupyter notebook 6.4.12

4．实验过程、结果及分析（包括代码截图、运行结果截图及必要的理论支撑等）

4.1 实验原理

本实验的内容是使用多项式函数$f(x)=\sum_{i=0}^{n} a_i x^i$来拟合正弦函数$y=\sin(x)$，其中$n$是多项式的阶数，$a_i$是多项式的系数。为了生成数据，我们在区间[0,1]上均匀采样m个点，并在正弦函数值上加入一定程度的噪声。我们用均方误差（MSE）作为损失函数，即

$$L(a_0, a_1, \dots, a_n)=\frac{1}{m}\sum_{i=1}^{m}(f(x_i)-y_i)^2 \qquad (1)$$

我们的目标是找到一组多项式系数$a_0, a_1, ..., a_n$，使得损失函数最小。为了防止过拟合，我们可以在损失函数中加入一个惩罚项（L2 范数），即

$$L(a_0, a_1, ..., a_n) = \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2 + \lambda\sum_{i=0}^{n} a_i^2 \qquad (2)$$

其中λ是一个超参数，用来控制惩罚项的强度。我们用梯度下降法来求解最优解，即每次更新多项式系数为

$$a_i = a_i - \alpha\frac{\partial L}{\partial a_i} \qquad (3)$$

其中α是另一个超参数，称为学习率，用来控制更新的步长。梯度下降法的终止条件可以是达到最大迭代次数或者损失函数小于某个阈值。

4.2 实验过程

1. 导入必要的库。

```
# 导入必要的库
import numpy as np
import matplotlib.pyplot as plt
```

2. 定义生成数据的函数，输入为数据量和噪声程度，输出为特征矩阵和标签向量。

```
# 定义正弦函数
def sin_func(x):
    return np.sin(2 * np.pi * x)
```

3. 定义多项式函数和损失函数（有无惩罚项），输入为特征矩阵、标签向量、多项式系数和超参数，输出为损失值。

```
# 定义多项式函数
def poly_func(w, x):
    n = len(w)
    y = 0
    for i in range(n):
        y += w[i] * (x ** i)
    return y

# 定义损失函数（均方误差）
def loss_func(y_true, y_pred):
    return 0.5 * np.mean((y_true - y_pred) ** 2)

# 定义带惩罚项的损失函数（L2 范数）
def loss_func_reg(y_true, y_pred, w, lam):
    return loss_func(y_true, y_pred) + 0.5 * lam * np.sum(w ** 2)
```

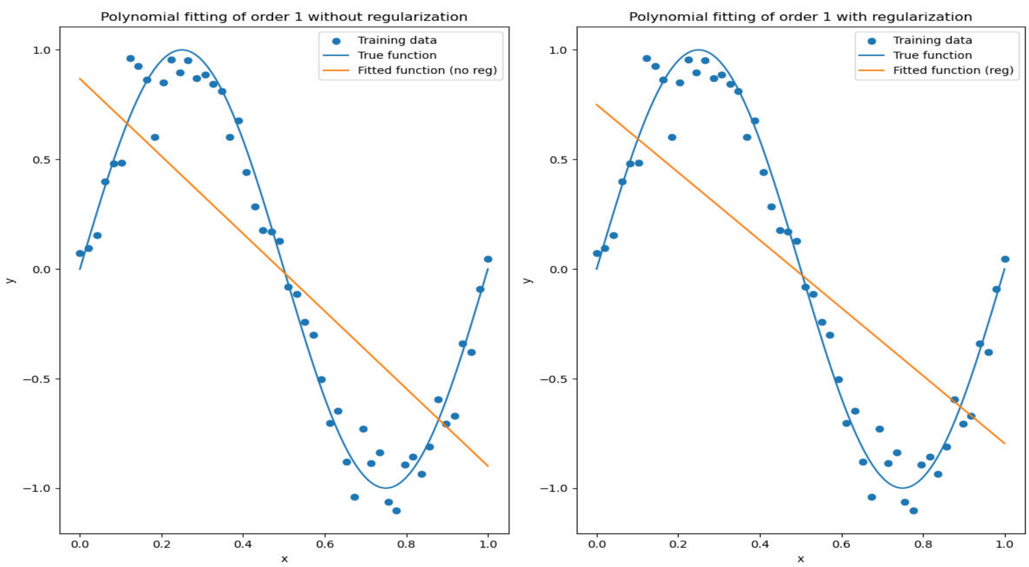
4. 定义梯度下降法求解最优解的函数，输入为特征矩阵、标签向量、多项式阶数、超参数和终止条件，输出为最优多项式系数和损失值。

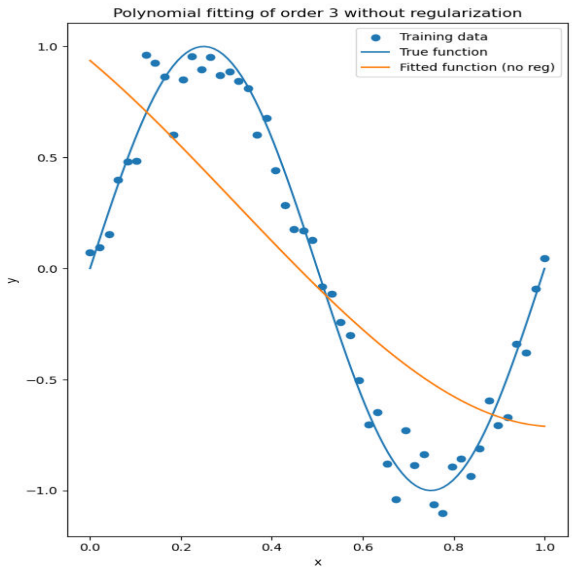
```
# 定义梯度下降法
def gradient_descent(x, y, w, lr, epochs, lam=0):
    n = len(w)
    m = len(x)
    loss_list = [] # 存储损失函数值
    for i in range(epochs):
        # 计算预测值
        y_pred = poly_func(w, x)
        # 计算损失值
        loss = loss_func_reg(y, y_pred, w, lam)
        loss_list.append(loss)
        # 计算梯度
        grad = np.zeros(n)
        for j in range(n):
            grad[j] = np.mean((y_pred - y) * (x ** j)) + lam * w[j]
        # 更新权重
        w = w - lr * grad
    return w, loss_list
```

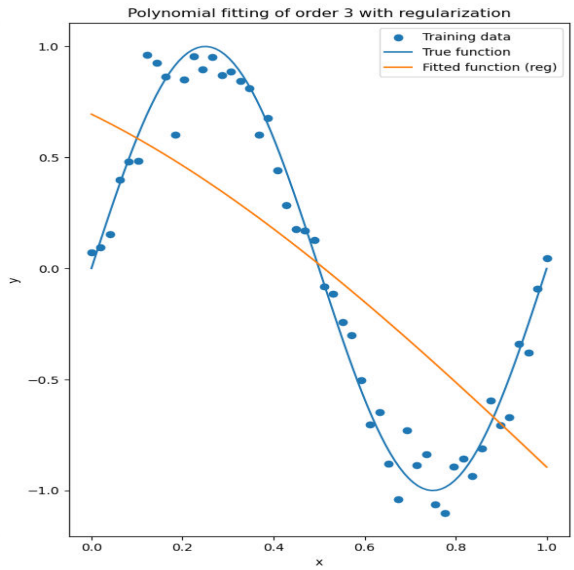
5. 用不同数据量，不同超参数，不同的多项式阶数，调用上述函数，进行实验，并观察实验结果。绘制拟合曲线和损失曲线的函数，输入为特征矩阵、标签向量、多项式系数和损失值，输出为图形。

4.3 实验结果及分析

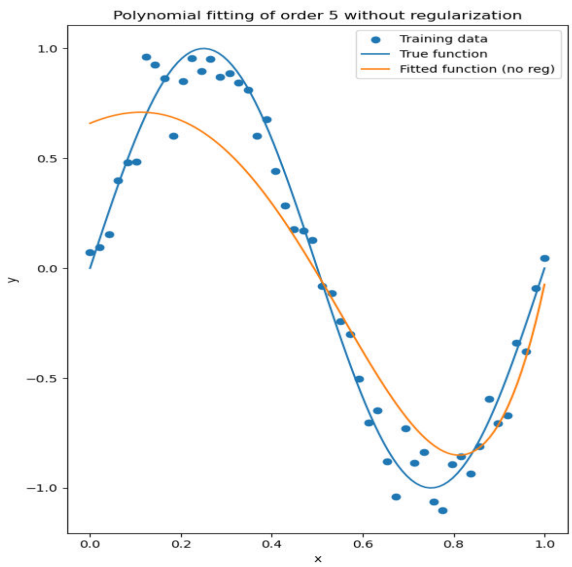
4.3.1 研究不同的多项式阶数的实验效果

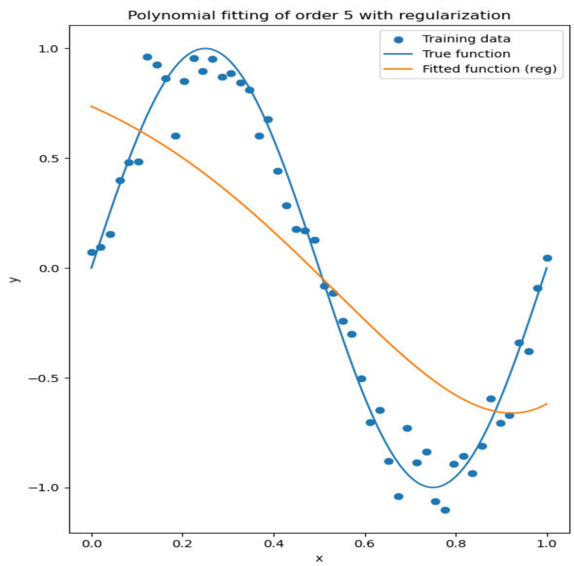


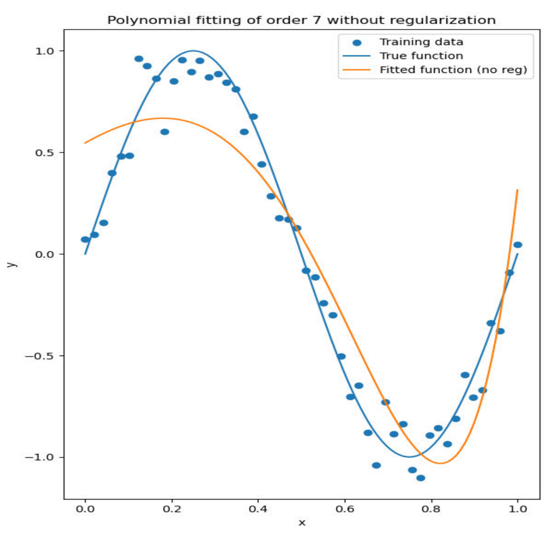


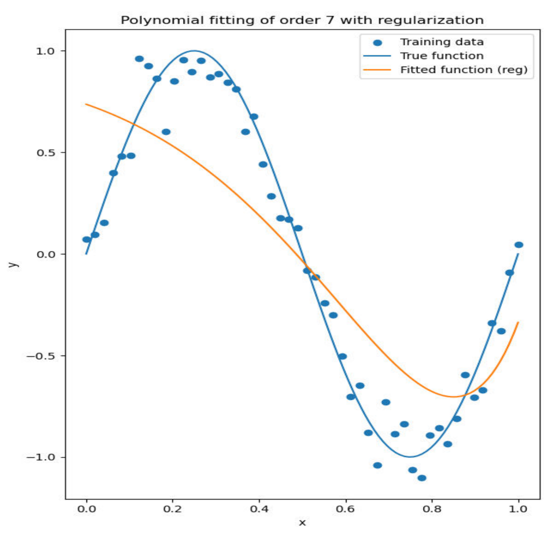


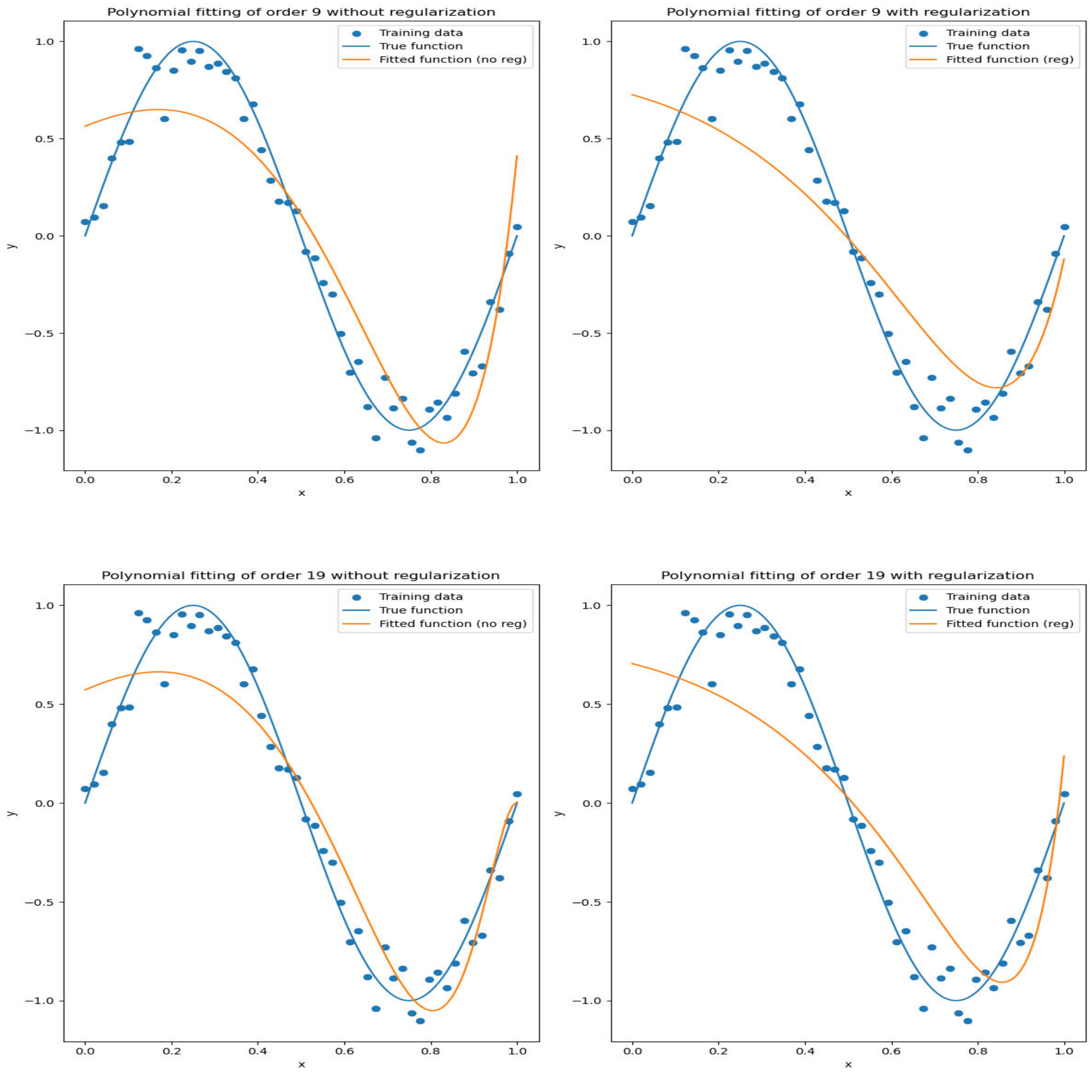
可以看出一阶，二阶的拟合效果并不好



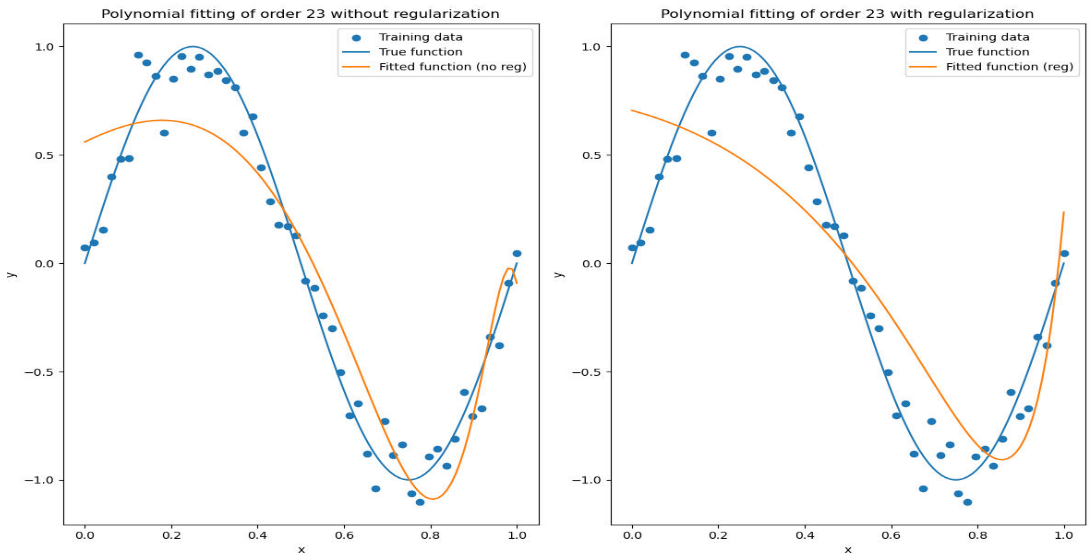




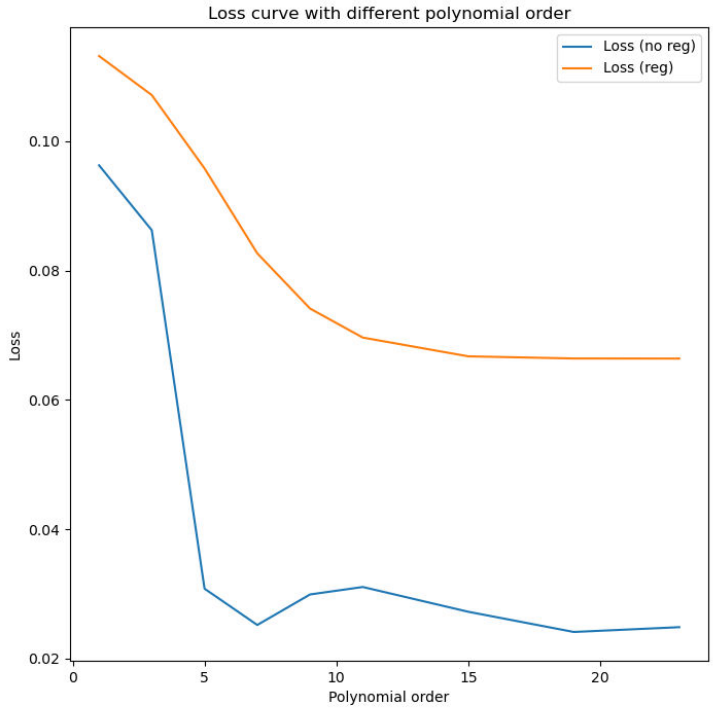




可以看出，随着阶数变大，拟合效果先提升后下降

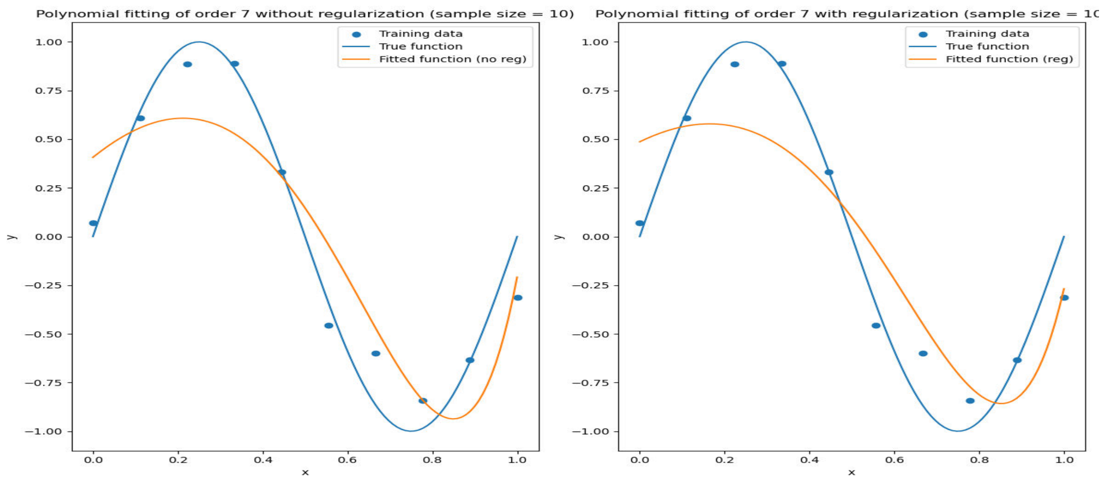


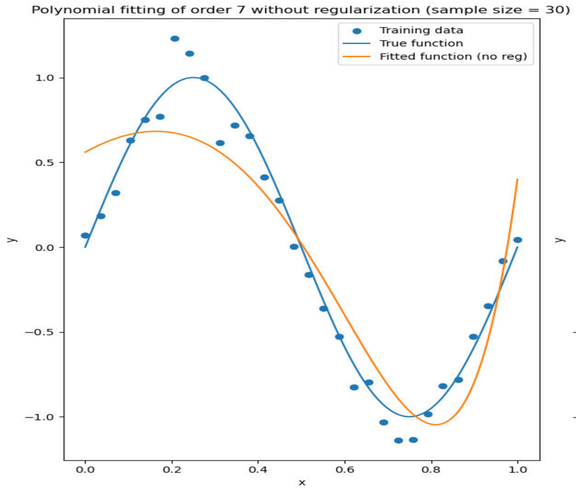
23 阶时明显过拟合

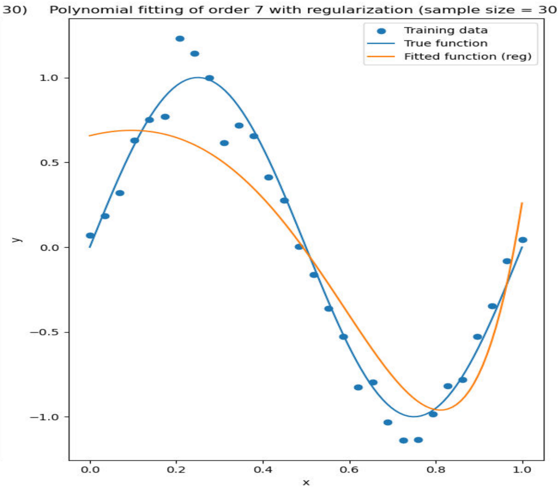


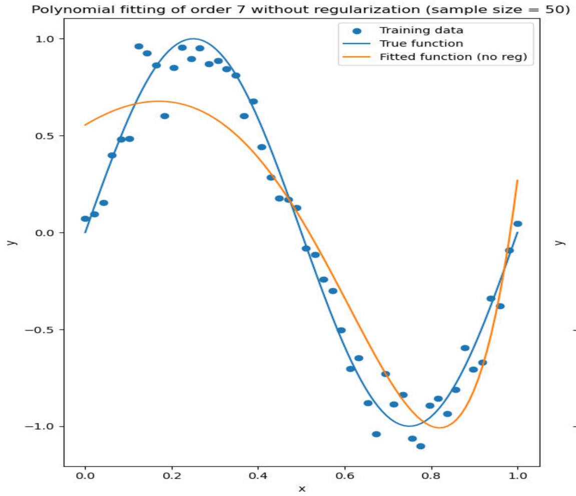
分析：根据损失函数的图像，阶数为 7 时效果较好。多项式阶数需要适中选择，过低会导致欠拟合，过高会导致过拟合。惩罚项可以有效地防止或减轻过拟合现象，提高模型的泛化能力。惩罚项的超参数 λ 需要根据数据集的大小、模型的复杂度、优化算法的特点等因素进行选择，以达到最佳的拟合效果。一般来说，λ 越大，惩罚项对模型参数的约束越强，模型越简单；λ 越小，惩罚项对模型参数的约束越弱，模型越复杂。

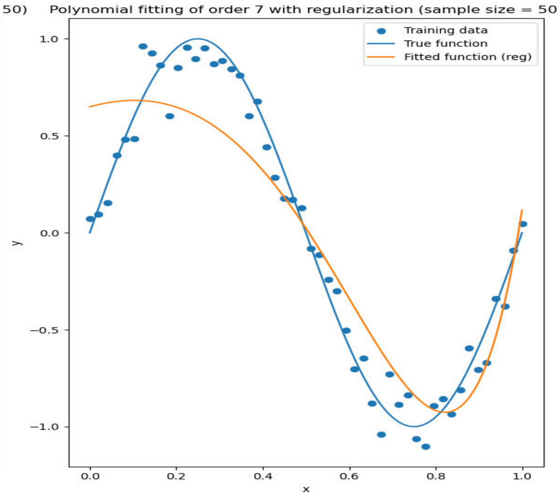
4.3.2 研究不同的数据量的实验效果

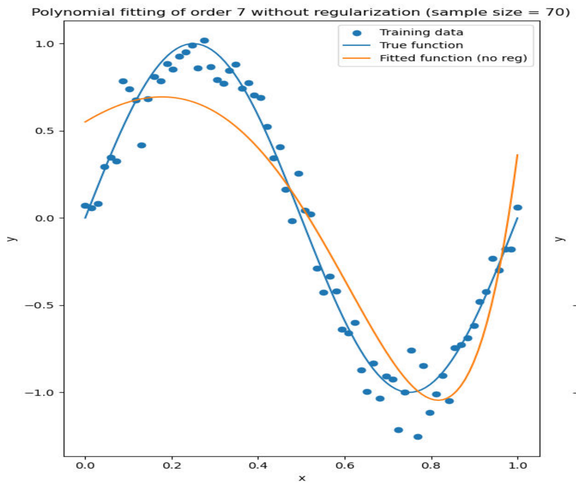


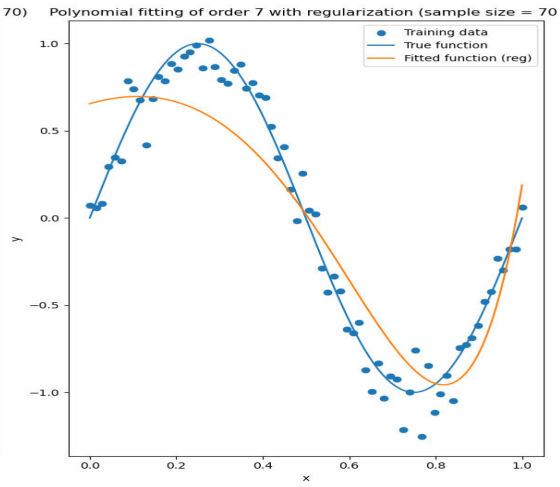

Polynomial fitting of order 7 without regularization (sample size = 30)
Training data
True function
Fitted function (no reg)
x
y

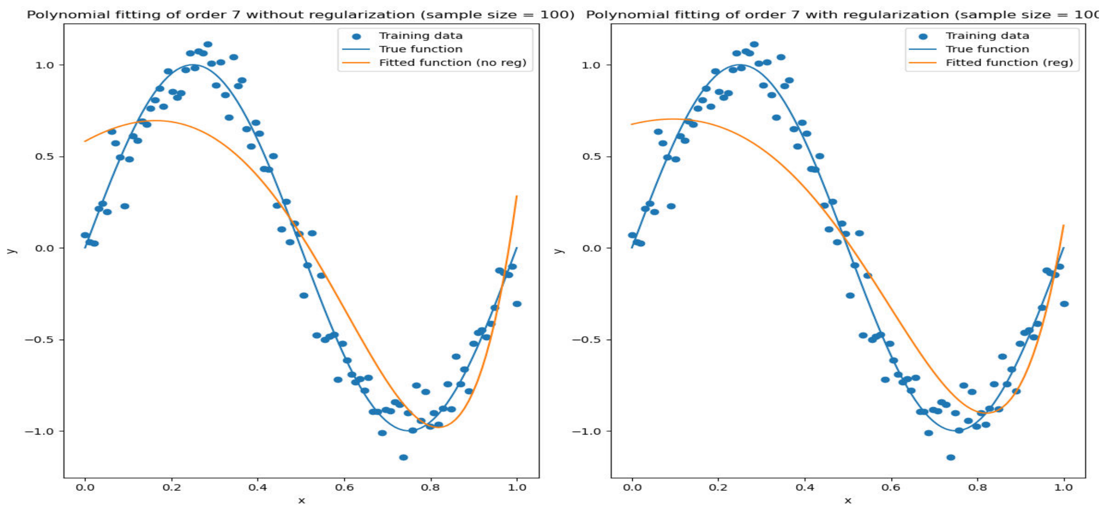

Polynomial fitting of order 7 with regularization (sample size = 30
Training data
True function
Fitted function (reg)
x
y

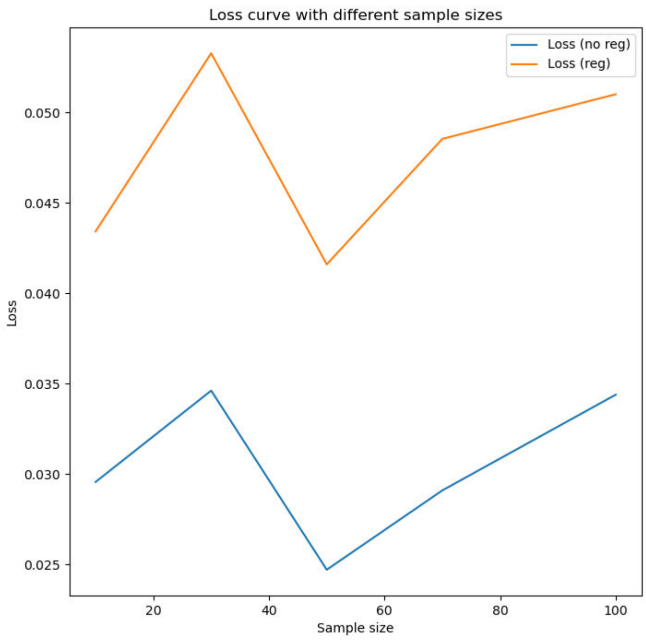

Polynomial fitting of order 7 without regularization (sample size = 50)
Training data
True function
Fitted function (no reg)
x
y


Polynomial fitting of order 7 with regularization (sample size = 50
Training data
True function
Fitted function (reg)
x
y


Polynomial fitting of order 7 without regularization (sample size = 70)
Training data
True function
Fitted function (no reg)
x
y

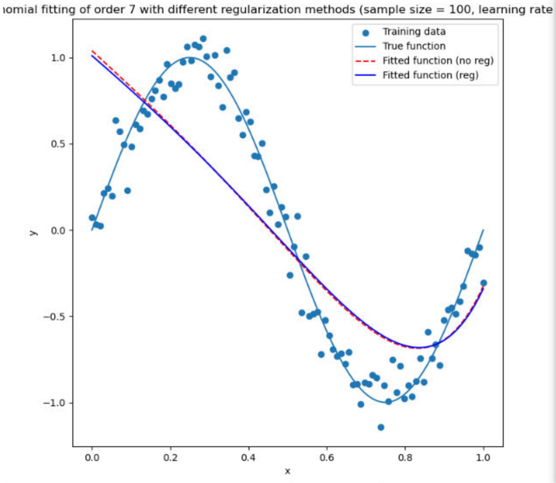

Polynomial fitting of order 7 with regularization (sample size = 70
Training data
True function
Fitted function (reg)
x
y

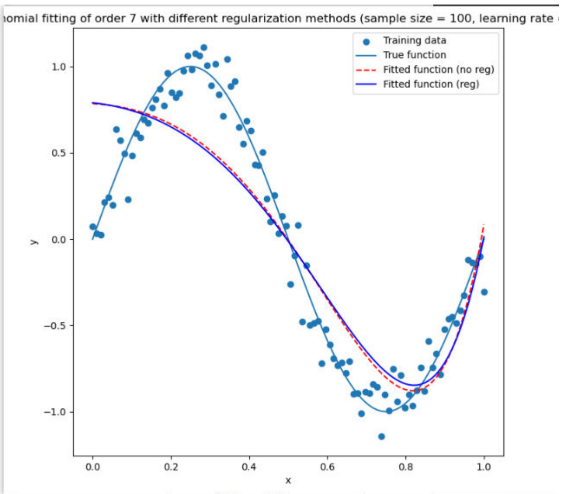


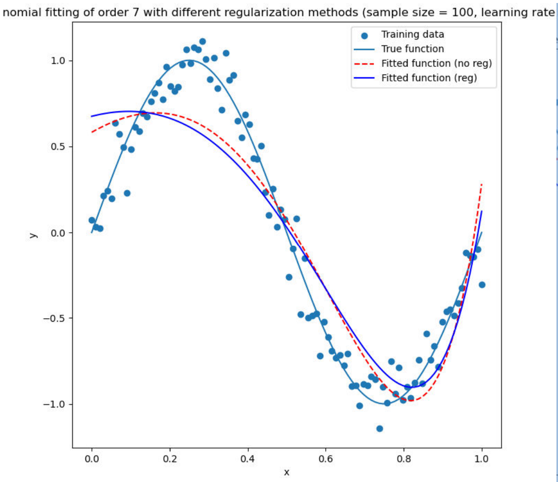


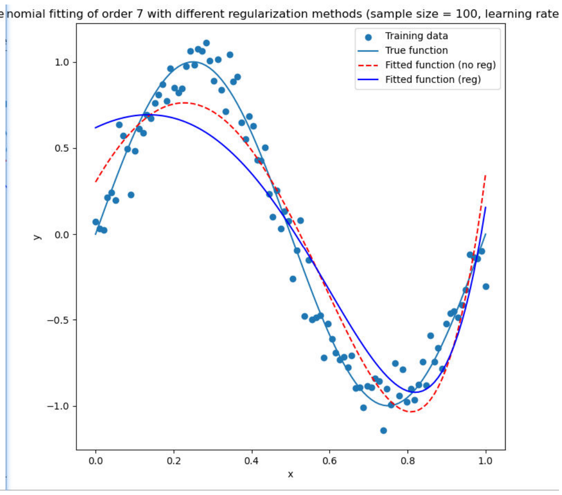
分析：根据损失函数的图像，数据量为 50，拟合效果最好，数据量过大效果不一定好。

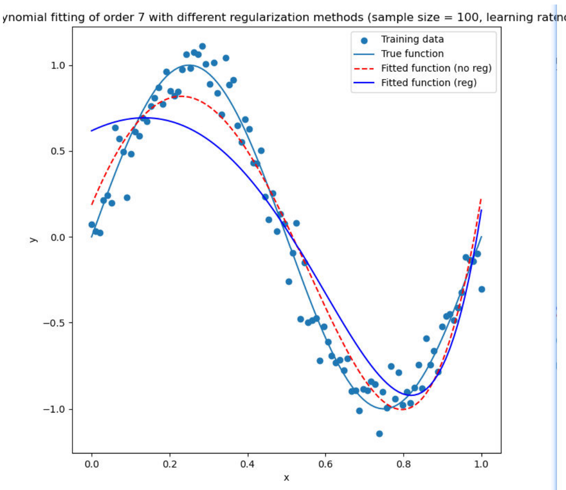
4.3.3 研究不同的学习率的实验效果

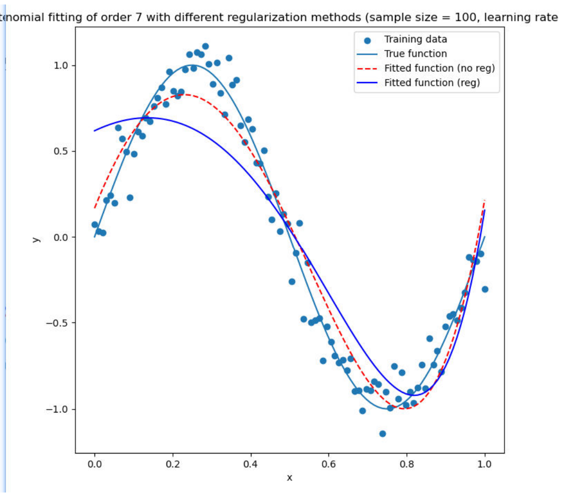

omial fitting of order 7 with different regularization methods (sample size = 100, learning rate
Training data
True function
Fitted function (no reg)
Fitted function (reg)
x
y

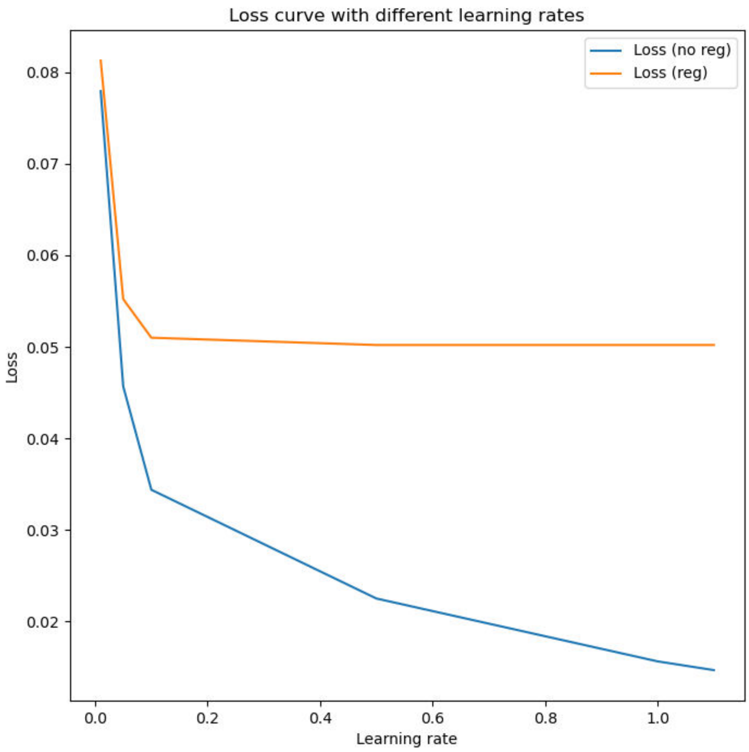

omial fitting of order 7 with different regularization methods (sample size = 100, learning rate
Training data
True function
Fitted function (no reg)
Fitted function (reg)
x
y


nomial fitting of order 7 with different regularization methods (sample size = 100, learning rate
Training data
True function
Fitted function (no reg)
Fitted function (reg)
x
y


nomial fitting of order 7 with different regularization methods (sample size = 100, learning rate
Training data
True function
Fitted function (no reg)
Fitted function (reg)
x
y

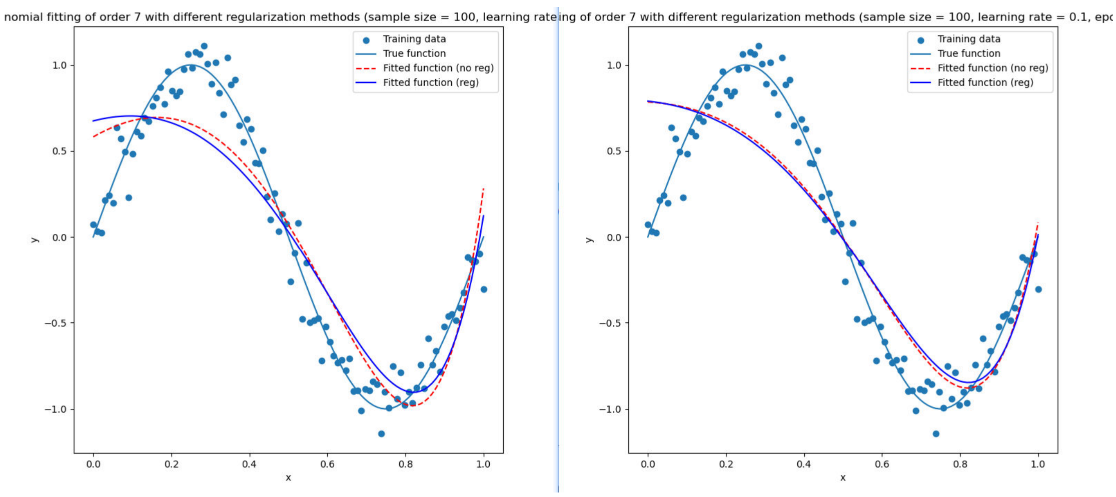

ynomial fitting of order 7 with different regularization methods (sample size = 100, learning rat
Training data
True function
Fitted function (no reg)
Fitted function (reg)
x
y

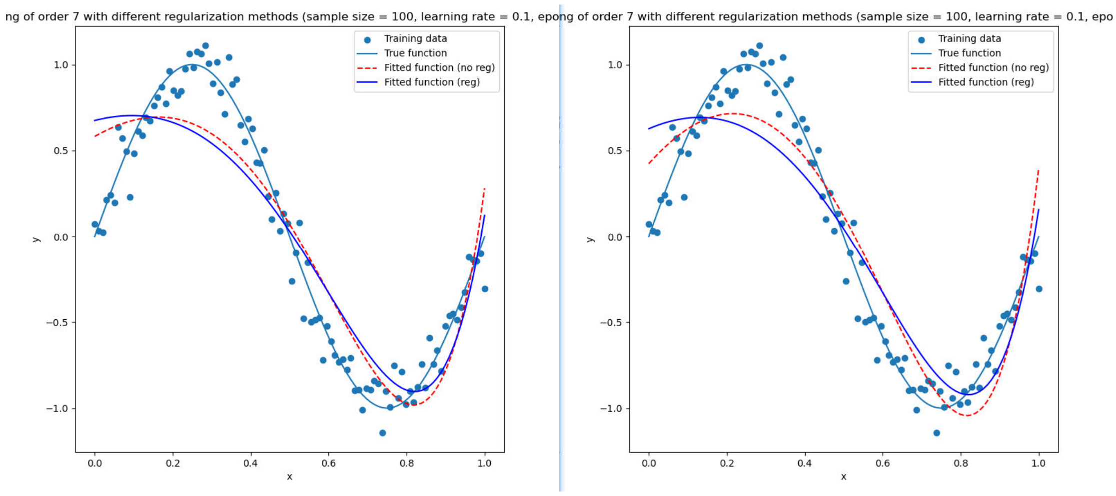

nomial fitting of order 7 with different regularization methods (sample size = 100, learning rate
Training data
True function
Fitted function (no reg)
Fitted function (reg)
x
y

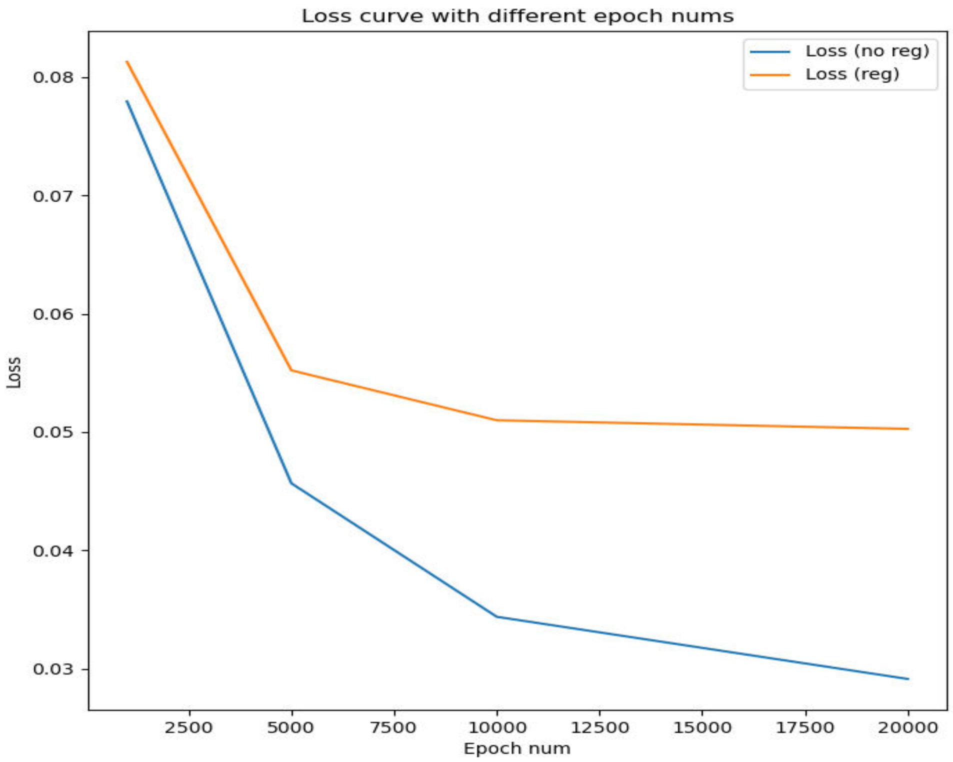


分析：根据损失函数的图像，学习率需要适当选择，过大会导致震荡或发散，过小会导致收敛缓慢，一般在 0.01 到 0.1 之间比较合适。

4.3.4 研究不同的迭代次数的实验效果







分析：根据损失函数的图像，迭代次数需要足够多，以保证模型能够收敛到最优解，一般在 10000 到 100000 之间比较合适。

5．实验总体结论

梯度下降法每次沿着负梯度方向更新模型参数，需要设置一个合适的学习率 α，以控制更新的步长。需要设置一个终止条件ϵ，以判断是否达到最优解。

数据量越大，拟合效果越好，因为数据越能代表真实分布，模型越能泛化到新的数据。

学习率需要适当选择，过大会导致震荡或发散，过小会导致收敛缓慢，一般在 0.01 到 0.1 之间比较合适。

迭代次数需要足够多，以保证模型能够收敛到最优解，一般在 10000 到 100000 之间比较合适。

不带惩罚项的多项式拟合，当多项式阶数较低时，会出现欠拟合现象，即模型过于简单，不能很好地逼近真实数据分布；当多项式阶数较高时，会出现过拟合现象，即模型过于复杂，对训练数据的噪声和细节也进行了拟合，导致泛化能力下降，无法适应新的数据。一般来说，三阶或四阶的多项式可以达到较好的拟合效果。

带惩罚项的多项式拟合，可以有效地防止或减轻过拟合现象，提高模型的泛化能力。惩罚项的超参数 λ 需要根据数据集的大小、模型的复杂度、优化算法的特点等因素进行选择，以达到最佳的拟合效果。一般来说，λ 越大，惩罚项对模型参数的约束越强，模型越简单；λ 越小，惩罚项对模型参数的约束越弱，模型越复杂。

6. 完整实验代码

```
# 导入必要的库
import numpy as np
import matplotlib.pyplot as plt

# 定义正弦函数
def sin_func(x):
    return np.sin(2 * np.pi * x)

# 定义多项式函数
def poly_func(w, x):
    n = len(w)
    y = 0
    for i in range(n):
        y += w[i] * (x ** i)
    return y

# 定义损失函数（均方误差）
def loss_func(y_true, y_pred):
    return 0.5 * np.mean((y_true - y_pred) ** 2)

# 定义带惩罚项的损失函数（L2 范数）
def loss_func_reg(y_true, y_pred, w, lam):
    return loss_func(y_true, y_pred) + 0.5 * lam * np.sum(w ** 2)
```

```python
# 定义梯度下降法
def gradient_descent(x, y, w, lr, epochs, lam=0):
    n = len(w)
    m = len(x)
    loss_list = [] # 存储损失函数值
    for i in range(epochs):
        # 计算预测值
        y_pred = poly_func(w, x)
        # 计算损失值
        loss = loss_func_reg(y, y_pred, w, lam)
        loss_list.append(loss)
        # 计算梯度
        grad = np.zeros(n)
        for j in range(n):
            grad[j] = np.mean((y_pred - y) * (x ** j)) + lam * w[j]
        # 更新权重
        w = w - lr * grad
    return w, loss_list

# 生成数据，加入噪声
np.random.seed(2023) # 设置随机种子
sample_size = 50 # 样本数量
x = np.linspace(0, 1, sample_size) # 在[0,1]区间内均匀采样
y = sin_func(x) + np.random.normal(0, 0.1, sample_size) # 加入正态分布噪声

# 用不同阶数多项式函数拟合曲线（建议正弦函数曲线）
poly_orders = [1, 3, 5, 7,9,11,15,19,23] # 多项式阶数列表
w_init = np.random.normal(0, 1, max(poly_orders) + 1) # 初始化权重
lr = 0.1 # 学习率
epochs = 10000 # 迭代次数

# 不加惩罚项的情况
w_no_reg_list = [] # 存储不同阶数的权重
loss_no_reg_list = [] # 存储不同阶数的损失值
for order in poly_orders:
    w_no_reg, loss_no_reg = gradient_descent(x, y, w_init[:order+1], lr,
epochs)
    w_no_reg_list.append(w_no_reg)
    loss_no_reg_list.append(loss_no_reg)

# 加惩罚项的情况
lam = 0.01 # 惩罚系数
w_reg_list = [] # 存储不同阶数的权重
loss_reg_list = [] # 存储不同阶数的损失值
```

```
for order in poly_orders:
    w_reg, loss_reg = gradient_descent(x, y, w_init[:order+1], lr, epochs,
lam)
    w_reg_list.append(w_reg)
    loss_reg_list.append(loss_reg)

# 绘制拟合曲线和损失曲线
x_test = np.linspace(0, 1, 100) # 测试数据
y_test = sin_func(x_test) # 真实值

for i in range(len(poly_orders)):
    order = poly_orders[i]

    plt.figure(figsize=(12, 8))
    plt.subplot(1, 2, 1)
    plt.scatter(x, y, label='Training data')
    plt.plot(x_test, y_test, label='True function')
    plt.plot(x_test, poly_func(w_no_reg_list[i], x_test), label='Fitted
function (no reg)')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title('Polynomial fitting of order {} without
regularization'.format(order))

    plt.subplot(1, 2, 2)
    plt.scatter(x, y, label='Training data')
    plt.plot(x_test, y_test, label='True function')
    plt.plot(x_test, poly_func(w_reg_list[i], x_test), label='Fitted
function (reg)')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title('Polynomial fitting of order {} with
regularization'.format(order))

    plt.tight_layout()
    plt.show()

plt.figure(figsize=(8, 8))
plt.plot(poly_orders, [loss_no_reg[-1] for loss_no_reg in
loss_no_reg_list], label='Loss (no reg)')
plt.plot(poly_orders, [loss_reg[-1] for loss_reg in loss_reg_list],
label='Loss (reg)')
```

```
plt.xlabel('Polynomial order')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss curve with different polynomial order')
plt.show()

# 用不同阶数多项式函数拟合曲线（建议正弦函数曲线）
poly_order = 7 # 多项式阶数
w_init = np.random.normal(0, 1, poly_order + 1) # 初始化权重
lr = 0.1 # 学习率
epochs = 10000 # 迭代次数

# 不加惩罚项的情况
w_no_reg_dict = {} # 存储不同数据量的权重
loss_no_reg_dict = {} # 存储不同数据量的损失值

# 加惩罚项的情况
lam = 0.001 # 惩罚系数
w_reg_dict = {} # 存储不同数据量的权重
loss_reg_dict = {} # 存储不同数据量的损失值

# 不同数据量列表
sample_sizes = [10,  30,  50, 70, 100]

# 对每个数据量进行拟合和绘图
for sample_size in sample_sizes:
    # 生成数据，加入噪声
    np.random.seed(2023) # 设置随机种子
    x = np.linspace(0, 1, sample_size) # 在[0,1]区间内均匀采样
    y = sin_func(x) + np.random.normal(0, 0.1, sample_size) # 加入正态分布
噪声
    # 不加惩罚项的情况
    w_no_reg, loss_no_reg = gradient_descent(x, y, w_init[:poly_order+1],
lr, epochs)
    w_no_reg_dict[sample_size] = w_no_reg
    loss_no_reg_dict[sample_size] = loss_no_reg
    # 加惩罚项的情况
    w_reg, loss_reg = gradient_descent(x, y, w_init[:poly_order+1], lr,
epochs, lam)
    w_reg_dict[sample_size] = w_reg
    loss_reg_dict[sample_size] = loss_reg
    # 绘制拟合曲线
    x_test = np.linspace(0, 1, 100) # 测试数据
```

```
    y_test = sin_func(x_test) # 真实值
    plt.figure(figsize=(12, 8))
    plt.subplot(1, 2, 1)
    plt.scatter(x, y, label='Training data')
    plt.plot(x_test, y_test, label='True function')
    plt.plot(x_test, poly_func(w_no_reg, x_test), label='Fitted function
(no reg)')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title('Polynomial fitting of order {} without regularization
(sample size = {})'.format(poly_order, sample_size))
    plt.subplot(1, 2, 2)
    plt.scatter(x, y, label='Training data')
    plt.plot(x_test, y_test, label='True function')
    plt.plot(x_test, poly_func(w_reg, x_test), label='Fitted function
(reg)')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title('Polynomial fitting of order {} with regularization (sample
size = {})'.format(poly_order, sample_size))
    plt.tight_layout()
    plt.show()
# 绘制损失值随数据量的变化
plt.figure(figsize=(8, 8))
plt.plot(sample_sizes, [loss_no_reg_dict[s][-1] for s in sample_sizes],
label='Loss (no reg)')
plt.plot(sample_sizes, [loss_reg_dict[s][-1] for s in sample_sizes],
label='Loss (reg)')
plt.xlabel('Sample size')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss curve with different sample sizes')
plt.show()


# 不同学习率列表（新增）
learning_rates = [0.01, 0.05 ,0.1,0.5,1,1.1]

# 对每个学习率进行拟合和绘图（修改）
for lr in learning_rates: # 新增

    # 不加惩罚项的情况
```

```
    w_no_reg, loss_no_reg = gradient_descent(x, y, w_init[:poly_order+1], lr, epochs)
    w_no_reg_dict[lr] = w_no_reg # 修改
    loss_no_reg_dict[lr] = loss_no_reg # 修改

    # 加惩罚项的情况
    w_reg, loss_reg = gradient_descent(x, y, w_init[:poly_order+1], lr, epochs, lam)
    w_reg_dict[lr] = w_reg # 修改
    loss_reg_dict[lr] = loss_reg # 修改

    # 绘制拟合曲线
    x_test = np.linspace(0, 1, 100) # 测试数据
    y_test = sin_func(x_test) # 真实值

    plt.figure(figsize=(8, 8))
    plt.scatter(x, y, label='Training data')
    plt.plot(x_test, y_test, label='True function')
    plt.plot(x_test, poly_func(w_no_reg, x_test), label='Fitted function (no reg)', color='red', linestyle='--')
    plt.plot(x_test, poly_func(w_reg, x_test), label='Fitted function (reg)', color='blue', linestyle='-')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title('Polynomial fitting of order {} with different regularization methods (sample size = {}, learning rate = {})'.format(poly_order, sample_size, lr)) # 修改
    plt.show()

# 绘制损失值随学习率的变化（修改）
plt.figure(figsize=(8, 8))
plt.plot(learning_rates, [loss_no_reg_dict[lr][-1] for lr in learning_rates], label='Loss (no reg)') # 修改
plt.plot(learning_rates, [loss_reg_dict[lr][-1] for lr in learning_rates], label='Loss (reg)') # 修改
plt.xlabel('Learning rate') # 修改
plt.ylabel('Loss')
plt.legend()
plt.title('Loss curve with different learning rates') # 修改
plt.show()

# 不同迭代次数列表（新增）
epoch_nums = [1000, 5000, 10000,20000]
```

```
# 固定学习率为0.05（新增）
lr = 0.1

# 对每个迭代次数进行拟合和绘图（修改）
for epochs in epoch_nums: # 新增

    # 不加惩罚项的情况
    w_no_reg, loss_no_reg = gradient_descent(x, y, w_init[:poly_order+1],
lr, epochs)
    w_no_reg_dict[epochs] = w_no_reg # 修改
    loss_no_reg_dict[epochs] = loss_no_reg # 修改

    # 加惩罚项的情况
    w_reg, loss_reg = gradient_descent(x, y, w_init[:poly_order+1], lr,
epochs, lam)
    w_reg_dict[epochs] = w_reg # 修改
    loss_reg_dict[epochs] = loss_reg # 修改

    # 绘制拟合曲线
    x_test = np.linspace(0, 1, 100) # 测试数据
    y_test = sin_func(x_test) # 真实值

    plt.figure(figsize=(8, 8))
    plt.scatter(x, y, label='Training data')
    plt.plot(x_test, y_test, label='True function')
    plt.plot(x_test, poly_func(w_no_reg, x_test), label='Fitted function
(no reg)', color='red', linestyle='--')
    plt.plot(x_test, poly_func(w_reg, x_test), label='Fitted function
(reg)', color='blue', linestyle='-')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title('Polynomial fitting of order {} with different regularization
methods (sample size = {}, learning rate = {}, epoch num =
{})'.format(poly_order, sample_size, lr, epochs)) # 修改
    plt.show()

# 绘制损失值随迭代次数的变化（修改）
plt.figure(figsize=(8, 8))
plt.plot(epoch_nums, [loss_no_reg_dict[e][-1] for e in epoch_nums],
label='Loss (no reg)') # 修改
plt.plot(epoch_nums, [loss_reg_dict[e][-1] for e in epoch_nums],
label='Loss (reg)') # 修改
```

```python
plt.xlabel('Epoch num') # 修改
plt.ylabel('Loss')
plt.legend()
plt.title('Loss curve with different epoch nums') # 修改
plt.show()
```

7. 参考文献

[1]刘远超 著. 深度学习基础, 北京: 高等教育出版社, 2023.9
[2]周志华 著. 机器学习, 北京: 清华大学出版社, 2016.1
[3]李航 著. 统计学习方法, 北京: 清华大学出版社, 2019.5