《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： 2021110683

姓名： 徐柯炎

# 实验报告内容

## 实验目的

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)

## 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。
6. 不许用现成的平台，例如pytorch，tensorflow的自动微分工具。建议实验编程环境：1）安装anaconda以及必要的工具包；2）建议采用python作为主编程语言，也可以根据个人需要和兴趣采用其它编程语言；3）可以基于实验室机器，也可以使用自己的便携式计算机上完成该实验。

## 实验环境

Windows10; python3.9;PyCharm 2021.2.2

## 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）

### 多项式拟合的原理

在线性回归中，多项式拟合就是用类似泰勒级数，使用多项式来拟合正弦函数𝑠𝑖𝑛(2𝜋𝑥) 。用公式来表示就是：

其中为样本个数，为阶数，为拟合的参数。

用矩阵表示：

其中

在这里通过足够的样本来训练可以得到合适的参数也就是，从而找到一般的学习规律。

### 梯度下降法原理

那么如何用合适的方法来得到多项式拟合的参数呢？最简单的方法就是梯度下降法。接下来将简要介绍一下梯度下降法的原理。

首先定义损失函数。损失函数就是一个自变量为算法的参数，函数值为误差值的函数。梯度下降就是找让误差值最小时候这个算法对应的参数。损失函数用公式表示为：

其中为样本的真实标签（值），这里的损失函数为均方损失函数，为惩罚项，为的L2范数。

接着初始化参数，一般采用随机化的方法初始化参数，并设置学习率。

然后迭代更新参数：

其中表示当前迭代的次数，表示损失函数关于的梯度。

接着检查终止条件：

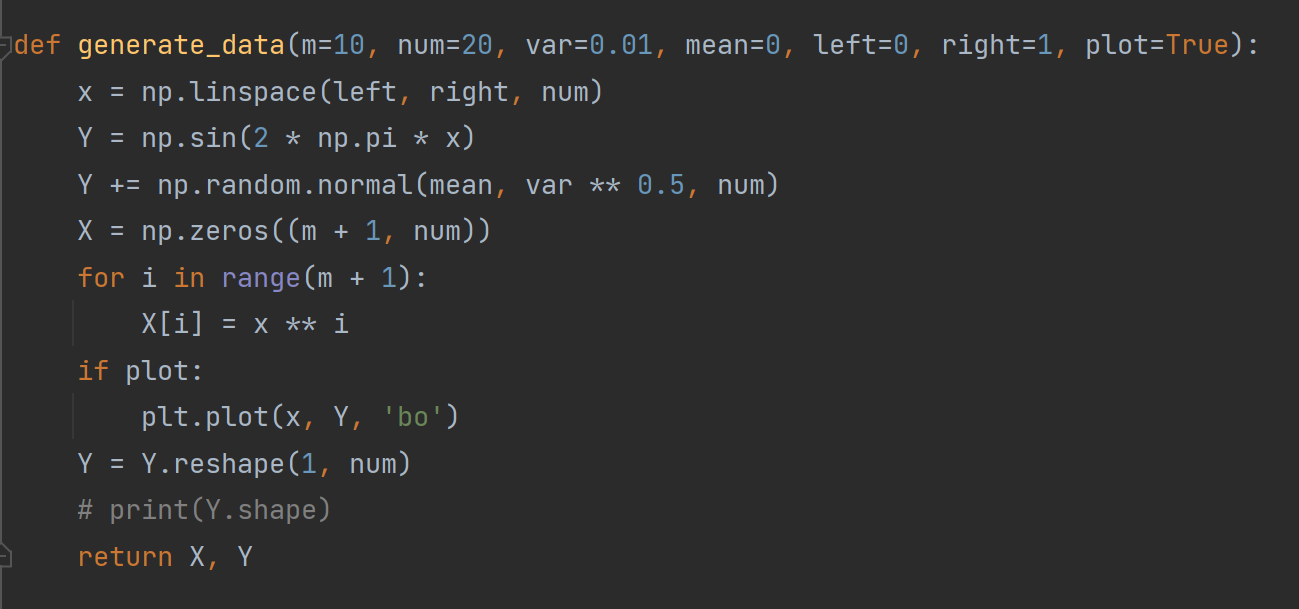
1. 可以设置迭代次数的最大值。
2. 或者可以检查参数更新的幅度是否足够小，也就是设置一个阈值。

如果终止条件不满足，则进入下一个循环迭代参数。

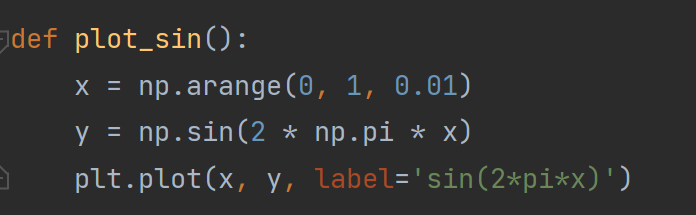
这个过程将不断地更新参数，使损失函数或目标函数逐渐收敛到最小值或局部最小值。学习率的选择和终止条件的设置都是梯度下降法的关键，它们会影响算法的性能和收敛速度。

### 实验过程

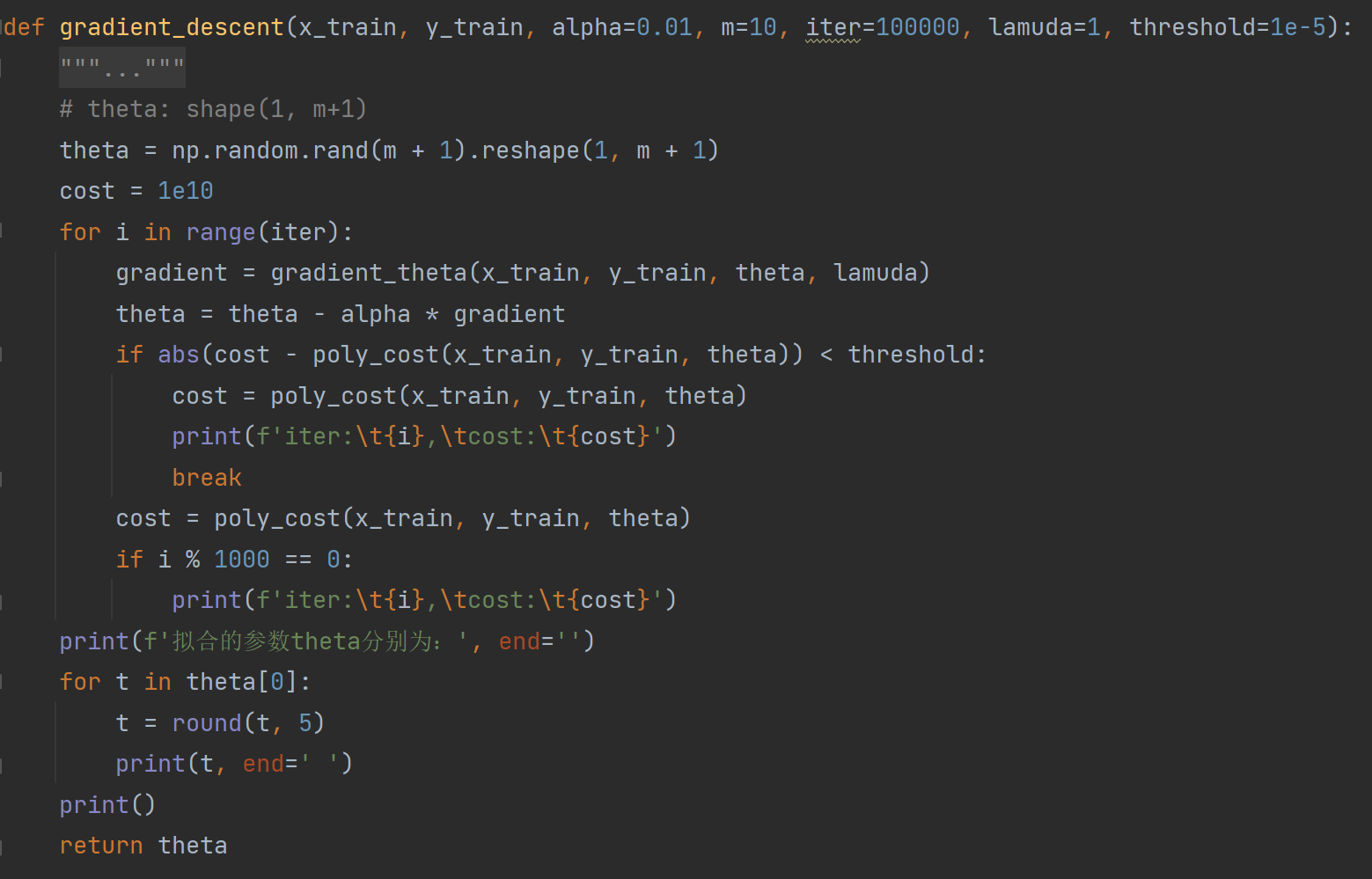
首先生成数据，加入噪声。



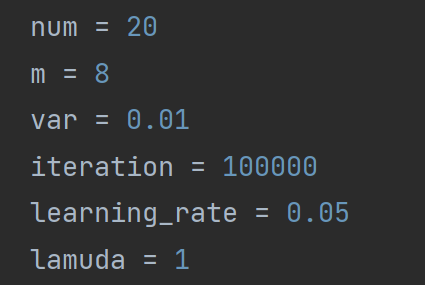
用高阶多项式函数拟合曲线（建议正弦函数曲线），正弦曲线如下。



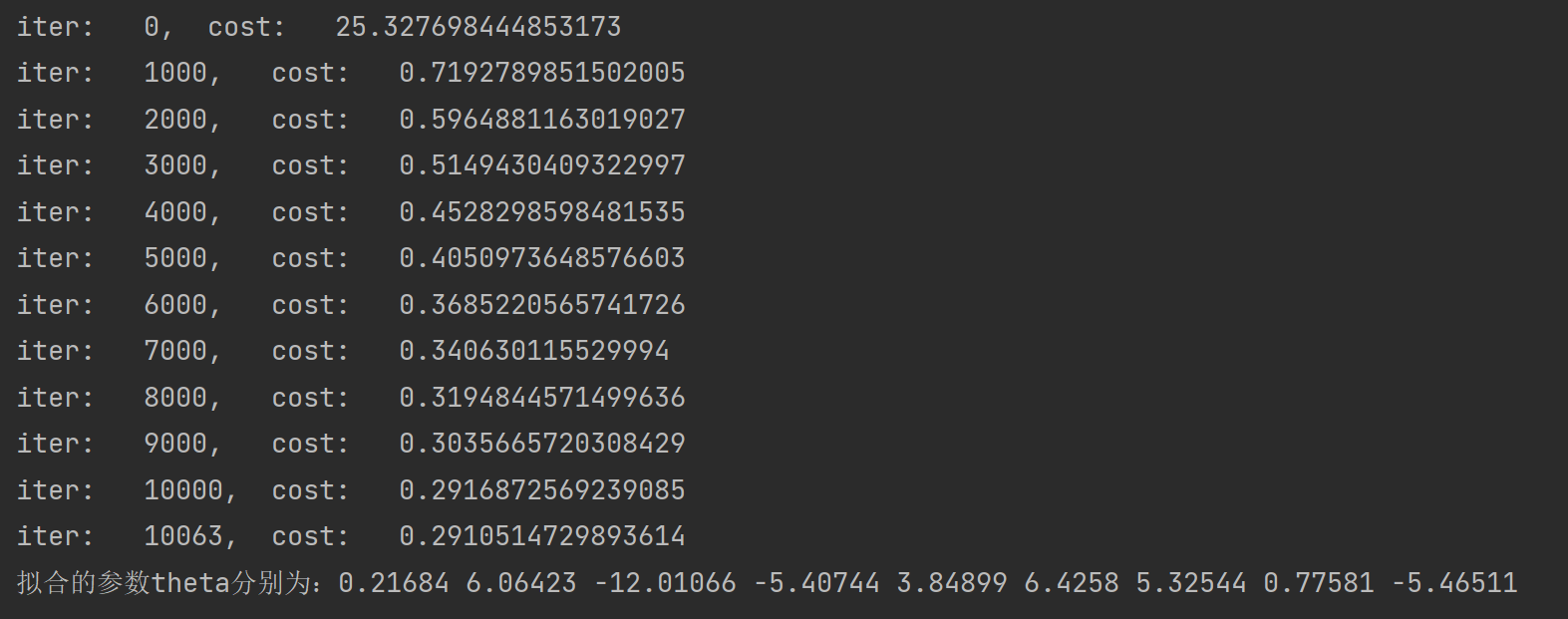
优化方法求解最优解（梯度下降），梯度下降函数如下。

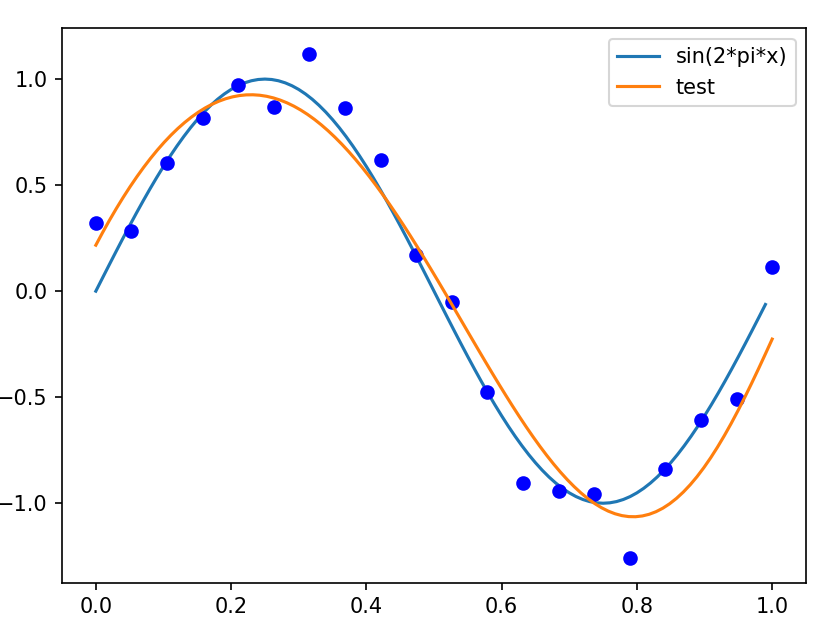


实验设置的参数如下：



实验结果如下：

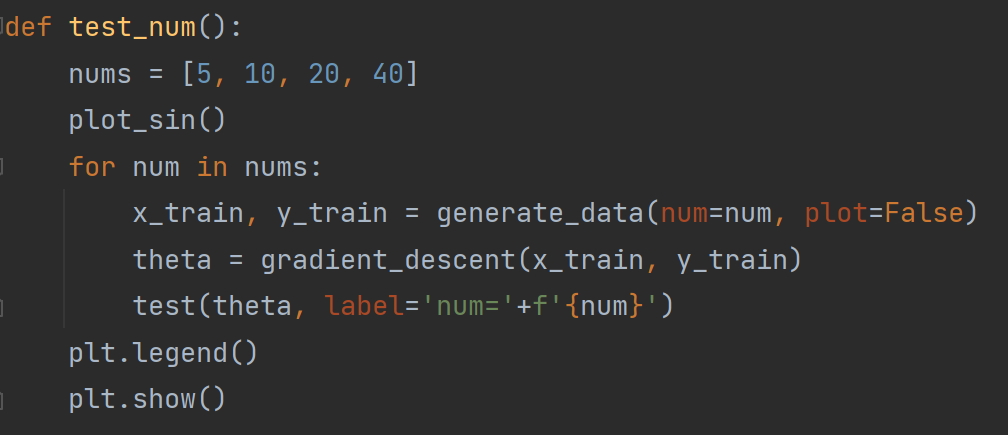




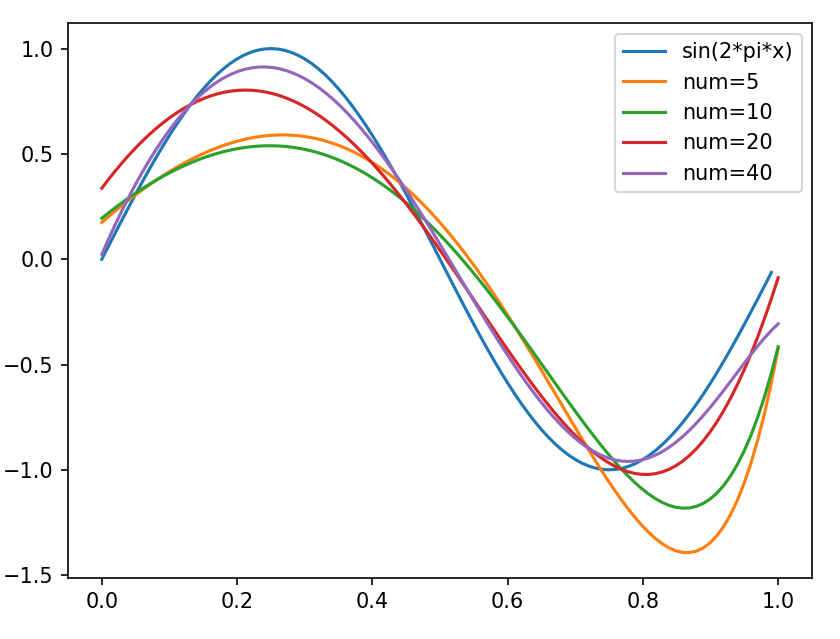
### 实验测试

1. 样本个数

测试代码如下：



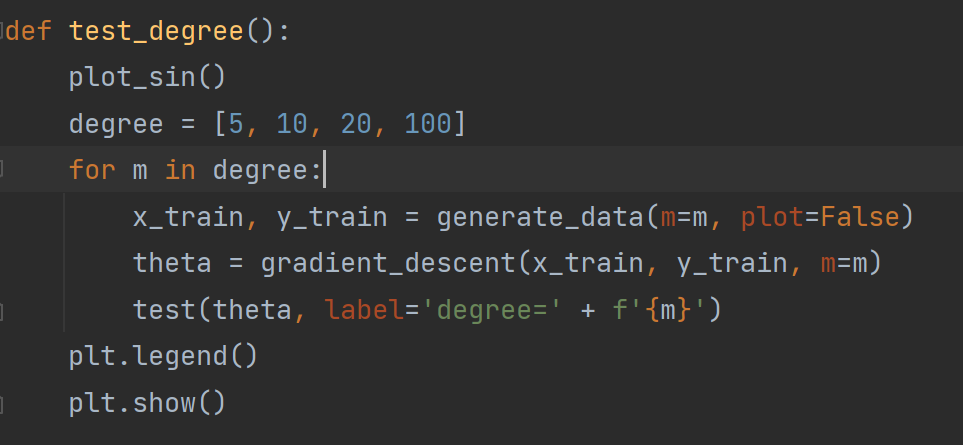
测试结果如下：



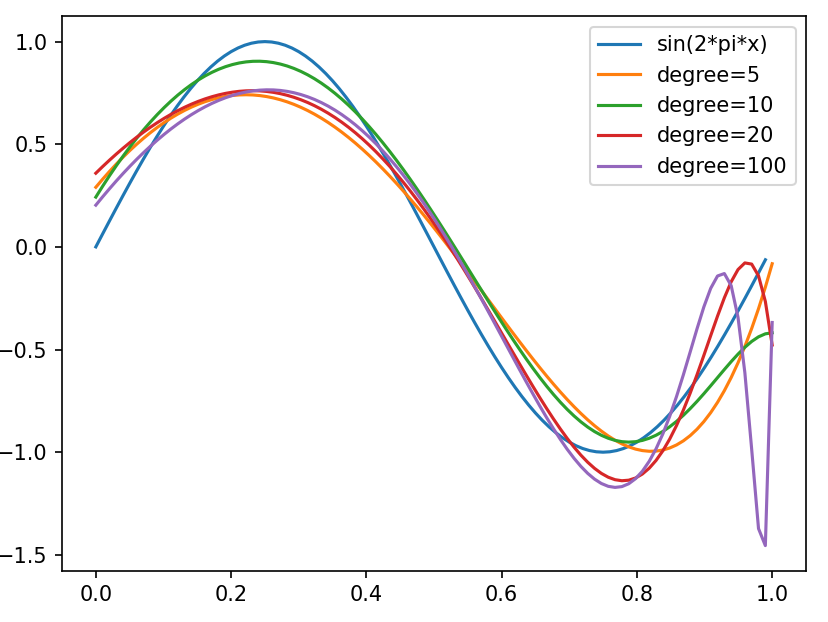
可以从上图看出，测试样本越少，越容易过拟合，也就是预测值和样本标签值几乎完全一致，但与实际情况不符。

1. 多项式阶数

测试代码如下：



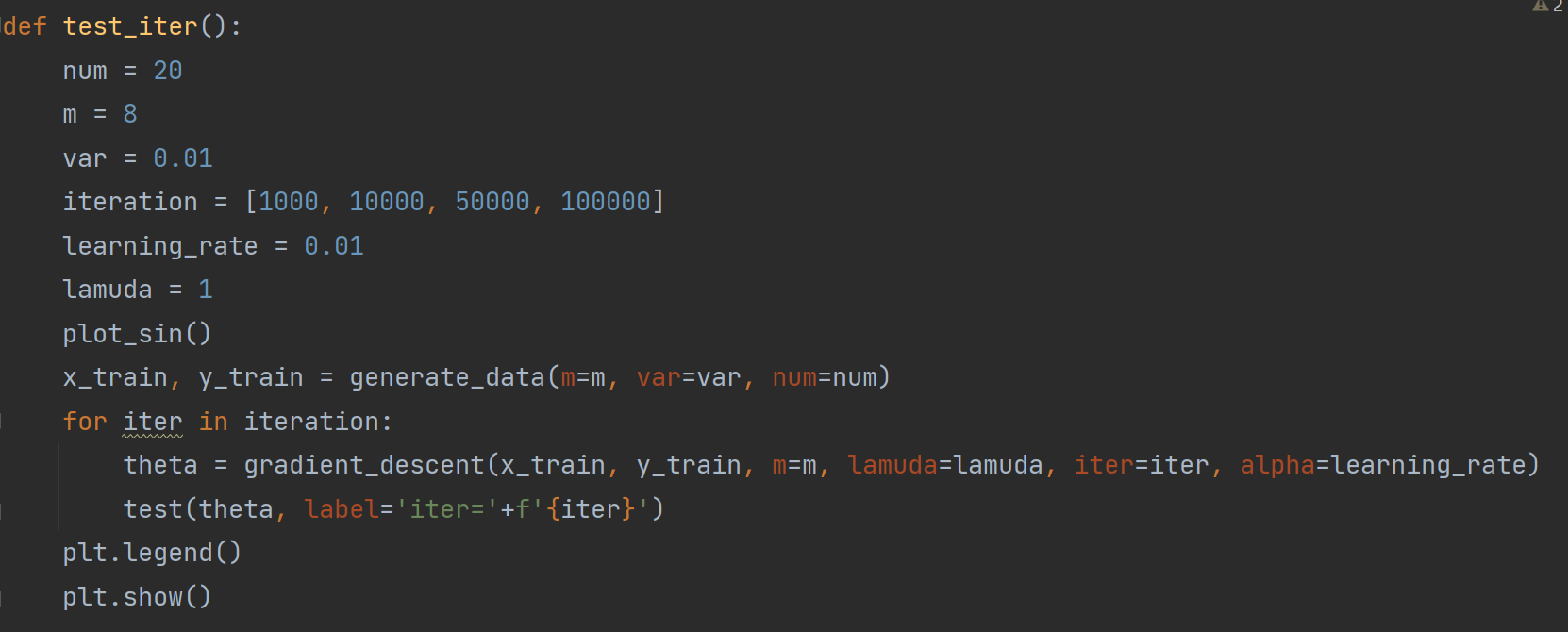
测试结果如下：



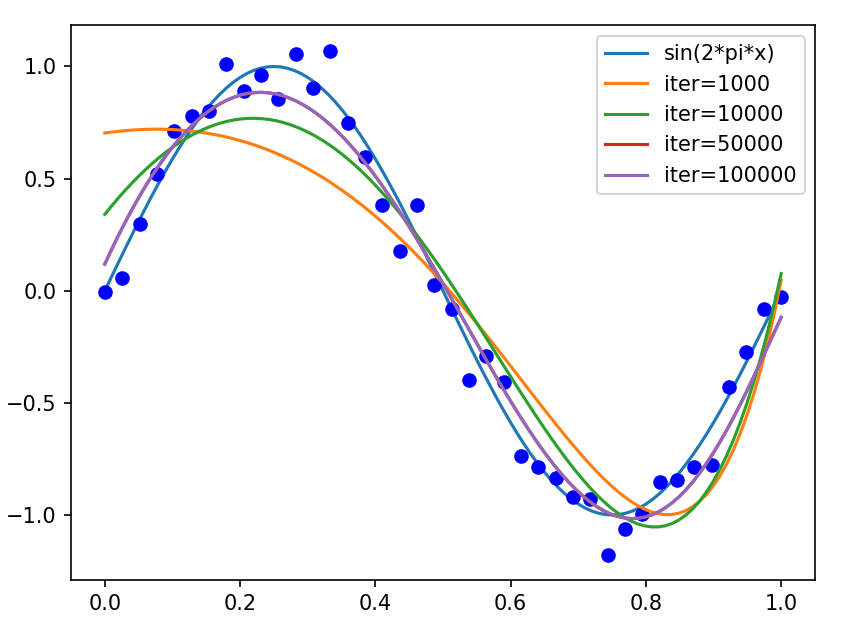
可以从上图看出，多项式阶数越大，曲线尾部越容易出现震荡的现象，也就是越容易过拟合，与正弦函数差距较大。

1. 迭代次数

测试代码如下：



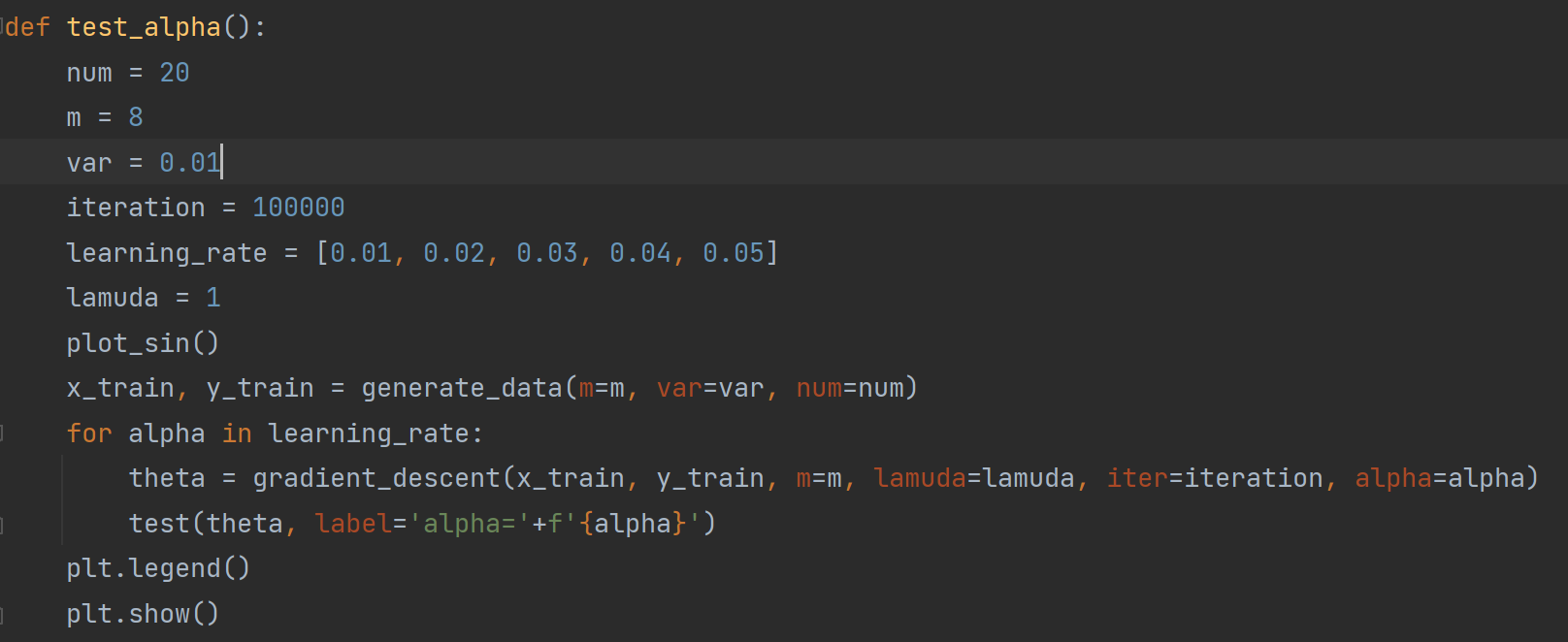
测试结果如下：



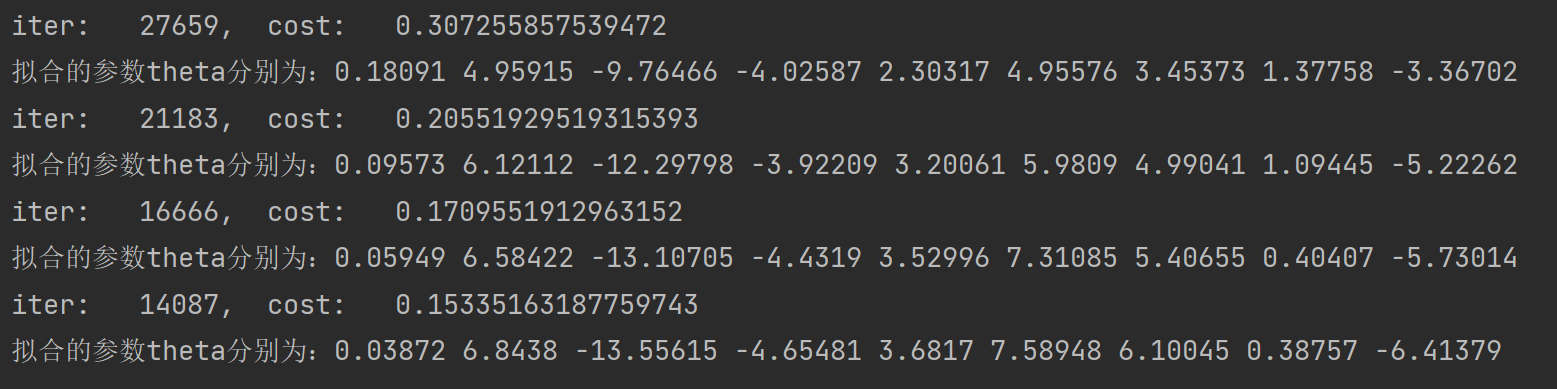
可以从上图看出，迭代次数越多，拟合的效果越好，次数越少，越容易出现欠拟合的现象。

1. 学习率

测试代码如下：



测试结果如下：



可以从上图看出， 学习率越高，迭代次数越少，越容易收敛；但是在测试中，如果学习率过大的话，导致梯度较大，导致梯度爆炸，这会导致我们的loss在训练时变为nan，也称之为数据溢出。

## 实验总体结论

### 梯度下降法的优点与缺点：

1. 优点：

广泛应用： 梯度下降法是一种通用的优化方法，可用于解决各种机器学习和数据分析问题，包括线性回归、逻辑回归、神经网络等。

速度快： 梯度下降法通常在迭代的早期就可以快速收敛到局部最优解附近，特别是在大规模数据集上。

易于实现： 梯度下降法的算法相对简单，容易实现，并且适用于各种机器学习框架和编程语言。

1. 缺点：

选择学习率困难： 学习率是梯度下降法的关键超参数，选择不当的学习率可能导致算法不收敛或收敛速度过慢。需要进行调试和调优。

可能陷入局部最优解： 对于非凸损失函数，梯度下降法可能会陷入局部最优解，而无法找到全局最优解。这取决于初始参数和学习率的选择。

可能受局部梯度幅度影响： 如果损失函数在某些方向上的梯度较小，而在其他方向上的梯度较大，梯度下降法可能会受到梯度幅度不平衡的影响，导致收敛困难。完整实验代码

### 调试参数的一些结论

1. 样本个数

测试样本越少，越容易过拟合，也就是预测值和样本标签值几乎完全一致，但与实际情况不符。

1. 多项式阶数

多项式阶数越大，曲线尾部越容易出现震荡的现象，也就是越容易过拟合，与正弦函数差距较大。

1. 迭代次数

迭代次数越多，拟合的效果越好，次数越少，越容易出现欠拟合的现象。

1. 学习率选择

学习率越高，迭代次数越少，越容易收敛；但是在测试中，如果学习率过大的话，导致梯度较大，导致梯度爆炸，这会导致我们的loss在训练时变为nan，也称之为数据溢出。

## 完整实验代码

1. # --\*-- coding:utf-8 --\*--
2. """
3. 作者：徐柯炎
4. 日期：2023年10月05日
5. """
6. **import** numpy as np
7. **import** matplotlib.pyplot as plt

10. # 画出sin函数
11. **def** plot\_sin():
12. x = np.arange(0, 1, 0.01)
13. y = np.sin(2 \* np.pi \* x)
14. plt.plot(x, y, label='sin(2\*pi\*x)')

17. # 生成数据（高斯噪声）
18. **def** generate\_data(m=8, num=20, var=0.01, mean=0, left=0, right=1, plot=True):
19. x = np.linspace(left, right, num)
20. Y = np.sin(2 \* np.pi \* x)
21. Y += np.random.normal(mean, var \*\* 0.5, num)
22. X = np.zeros((m + 1, num))
23. **for** i **in** range(m + 1):
24. X[i] = x \*\* i
25. **if** plot:
26. plt.plot(x, Y, 'bo')
27. Y = Y.reshape(1, num)
28. # print(Y.shape)
29. **return** X, Y

32. # 求解梯度
33. **def** gradient\_theta(x\_train, y\_train, theta, lamuda):
34. gradient = np.dot(np.dot(theta, x\_train) - y\_train, x\_train.T) + np.sum(lamuda \* theta)
35. **return** gradient

38. # 损失函数
39. **def** poly\_cost(x\_train, y\_train, theta):
40. **return** 0.5 \* np.sum((np.dot(theta, x\_train) - y\_train) \*\* 2)

43. **def** least\_square(x\_train, y\_train):
44. theta = np.linalg.inv(np.dot(x\_train, x\_train.T)).dot(x\_train).dot(y\_train.T).reshape(1, -1)
45. # print(np.linalg.inv(np.dot(x\_train, x\_train.T)))
46. # print(theta)
47. **return** theta

50. # 梯度下降
51. **def** gradient\_descent(x\_train, y\_train, alpha=0.01, m=8, iter=100000, lamuda=1, threshold=1e-5):
52. """
53. :param threshold
54. :param m: degree
55. :param x\_train: shape(m+1, num)
56. :param y\_train: shape(1, num)
57. :param alpha: learning rate
58. :param iter: iteration
59. :param lamuda: penalty term
60. :return: theta
61. """
62. # 参数初始化
63. # theta: shape(1, m+1)
64. theta = np.random.rand(m + 1).reshape(1, m + 1)
65. cost = 1e10
66. **for** i **in** range(iter):
67. # 求解当前梯度
68. gradient = gradient\_theta(x\_train, y\_train, theta, lamuda)
69. # 更新参数
70. theta = theta - alpha \* gradient
71. # 结束迭代的条件
72. **if** abs(cost - poly\_cost(x\_train, y\_train, theta)) < threshold:
73. cost = poly\_cost(x\_train, y\_train, theta)
74. **print**(f'iter:\t{i},\tcost:\t{cost}')
75. **break**
76. # 计算损失
77. cost = poly\_cost(x\_train, y\_train, theta)
78. # if i % 1000 == 0:
79. # print(f'iter:\t{i},\tcost:\t{cost}')
80. **print**(f'拟合的参数theta分别为：', end='')
81. **for** t **in** theta[0]:
82. t = round(t, 5)
83. **print**(t, end=' ')
84. **print**()
85. **return** theta

88. # 测试拟合效果
89. **def** test(theta, num=100, left=0, right=1, label="test"):
90. de = theta.shape[1]
91. # print((de, num))
92. x = np.linspace(left, right, num).reshape(1, -1)
93. X = np.zeros((de, num))
94. **for** i **in** range(de):
95. X[i] = x \*\* i
96. y = np.dot(theta, X)
97. # print(x.shape, y.shape)
98. plt.plot(x[0], y[0], label=label)

101. # 测试学习率变化
102. **def** test\_alpha():
103. num = 20
104. m = 8
105. var = 0.01
106. iteration = 100000
107. learning\_rate = [0.01, 0.02, 0.03, 0.04]
108. lamuda = 1
109. plot\_sin()
110. x\_train, y\_train = generate\_data(m=m, var=var, num=num)
111. **for** alpha **in** learning\_rate:
112. theta = gradient\_descent(x\_train, y\_train, m=m, lamuda=lamuda, iter=iteration, alpha=alpha)
113. test(theta, label='alpha='+f'{alpha}')
114. plt.legend()
115. plt.show()

118. # 测试不同迭代次数
119. **def** test\_iter():
120. num = 40
121. m = 8
122. var = 0.01
123. iteration = [1000, 10000, 50000, 100000]
124. learning\_rate = 0.01
125. lamuda = 1
126. plot\_sin()
127. x\_train, y\_train = generate\_data(m=m, var=var, num=num)
128. **for** iter **in** iteration:
129. theta = gradient\_descent(x\_train, y\_train, m=m, lamuda=lamuda, iter=iter, alpha=learning\_rate)
130. test(theta, label='iter='+f'{iter}')
131. plt.legend()
132. plt.show()

135. # 测试训练样本个数
136. **def** test\_num():
137. nums = [5, 10, 20, 40]
138. plot\_sin()
139. **for** num **in** nums:
140. x\_train, y\_train = generate\_data(num=num, plot=False)
141. theta = gradient\_descent(x\_train, y\_train)
142. test(theta, label='num='+f'{num}')
143. plt.legend()
144. plt.show()

147. # 测试多项式阶数
148. **def** test\_degree():
149. plot\_sin()
150. degree = [5, 10, 20, 100]
151. **for** m **in** degree:
152. x\_train, y\_train = generate\_data(m=m, plot=False)
153. theta = gradient\_descent(x\_train, y\_train, m=m)
154. test(theta, label='degree=' + f'{m}')
155. plt.legend()
156. plt.show()

159. # 测试惩罚项
160. **def** test\_lamuda():
161. plot\_sin()
162. m = 20
163. lamudas = [0.1, 0.5, 1, 2]
164. iter = 100000
165. threshold = 1e-8
166. num = 20
167. x\_train, y\_train = generate\_data(num=num, m=20)
168. **for** lamuda **in** lamudas:
169. theta = gradient\_descent(x\_train, y\_train, lamuda=lamuda, threshold=threshold, m=20, iter=iter)
170. test(theta, label='lamuda=' + f'{lamuda}')
171. plt.legend()
172. plt.show()

175. # 测试主函数
176. **def** test\_main():
177. num = 20
178. m = 8
179. var = 0.01
180. iteration = 100000
181. learning\_rate = 0.05
182. lamuda = 1
183. plot\_sin()
184. x\_train, y\_train = generate\_data(m=m, var=var, num=num)
185. # theta = least\_square(x\_train, y\_train)
186. theta = gradient\_descent(x\_train, y\_train, m=m, lamuda=lamuda, iter=iteration, alpha=learning\_rate)
187. test(theta)
188. plt.legend()
189. plt.show()

192. **if** \_\_name\_\_ == '\_\_main\_\_':
193. test\_alpha()

## 参考文献

[详解梯度下降法（干货篇）](https://zhuanlan.zhihu.com/p/112416130?ivk_sa=1024320u)