《模式识别与机器学习A》实验报告

实验题目： 多项式拟合正弦曲线

学 号： 2021112845

姓 名： 张智雄

**实验报告内容**

1. **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（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、增加样本量等)

1. **实验内容**
2. 生成数据，加入噪声；
3. 用高阶多项式函数拟合曲线（建议正弦函数曲线）；
4. 优化方法求解最优解（**梯度下降**）；
5. 用你得到的实验数据，解释过拟合。
6. 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。
7. 不许用现成的平台，例如Pytorch，TensorFlow的自动微分工具。
8. **实验环境**

Windows11; Anaconda+python3.11; VS Code

1. **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 算法理论支撑**

4.1.1 多项式拟合

设观测样本数为，多项式函数形式为，可形式化为一个最优化问题：

而又可写成矩阵相乘形式为

为方便表示可记为，其中为范德蒙德行列式(Vandermonde determinant)，为模型参数，则残差平方和为：

则目标函数为，将展开得到，

由于是的函数，记作，则对求导即为。

使用对进行梯度下降优化迭代，即可求得局部最优模型参数。

4.1.2 最小二乘法

上述对求导即求得模型参数，，将数据点信息代入上述公式即可求得多项式拟合模型。

4.1.3 梯度下降

梯度下降法(Gradient Descent)的基本思想是通过迭代地更新模型参数，沿着目标函数的负梯度方向，按照一定步长逐步逼近最优解。

**ALGORITHM 1** Gradient Descent (梯度下降)

1：**input** 目标优化函数，学习率，迭代次数，参数初值；

2：；

3：**while** **do**

4： ；

5： ；

6：**end while**

7：**return** //返回最优参数；

4.1.4 共轭梯度法

上述最小化问题可等价于求解等式，可转化为最小化问题

构造个相互关于共轭的向量，则空间任何向量都可以表示为，则上述优化问题可以等价于

由于相互共轭，因此当时，有，所以上式可变为

对其各项求导，即可得，因而最终最优解。

通过寻找共轭梯度进行求解能够较梯度下降优化更快的收敛，设残差，残差向量相互**正交**，迭代过程中满足。令表示每次迭代过程中的搜索方向，且满足与过去的搜索方向都**共轭**，即，联立即可解得的更新公式。

**ALGORITHM 2** Conjugate Gradient Descent(共轭梯度下降)

1：**input** 目标优化函数，迭代次数，参数初值；

2：，，残差，搜索方向，；

3：**while** **do**

4： ​；

5： ，；

6: ，；

7：**end while**

8：**return** //返回最优参数；

4.1.4 正则惩罚

还可通过增加正则化的方式对近似模型进行约束，一般包括正则项和正则项。添加正则项后，问题可以形式化为，

其中，控制正则项与误差项的均衡程度。

正则项指在损失函数中添加模型参数的范数作为正则化项，使得模型参数中的一部分变为0，从而实现特征选择和模型稀疏性。

而正则项是指在损失函数中添加模型参数的范数作为正则化项，惩罚模型参数的平方和，减小模型参数的幅度，减少模型的过拟合风险。

在此模型中参数较少，使用正则意义不大，因而在本实验中均采用正则项。

4.1.5 模型评价

除通过作图简单直观分析 ，还可数据层面评价上述各种方法的拟合效果。采用均方根误差Root-Mean-Square(RMS)进行误差评价

其中，较小的值表示拟合效果较好，误差较小；较大的值表示拟合效果较差，误差较大。

**4.2 实验设计**

4.2.1 随机数据生成

假设给定函数，从中生成𝑛个点 ，假设点， 其中是服从高斯分布的随机噪声。生成数据后，利用np.column\_stack方法生成观测样本的特征矩阵。

文本

描述已自动生成

图1 数据生成代码截图

4.2.2 最小二乘法

根据上述公式直接运算即可，加入正则项则原式子变为（当时退化）。

图形用户界面, 文本, 应用程序

描述已自动生成

图2 最小二乘法代码截图

4.2.3 梯度下降法

根据上述式子设置学习率循环反复迭代即可，加入正则项原式变为。由于在本实验中梯度下降法较难收敛，因此引入accept\_gradient作为阈值，即当梯度小于一个较小阈值时停止迭代。

文本

描述已自动生成

图3 梯度下降法代码截图

4.2.4 共轭梯度法

根据上述算法设置迭代更新即可，加入正则项后变为。终止采用参数收敛方法，即引入accept\_gradient作为阈值，即当残差小于一个较小阈值时停止迭代。

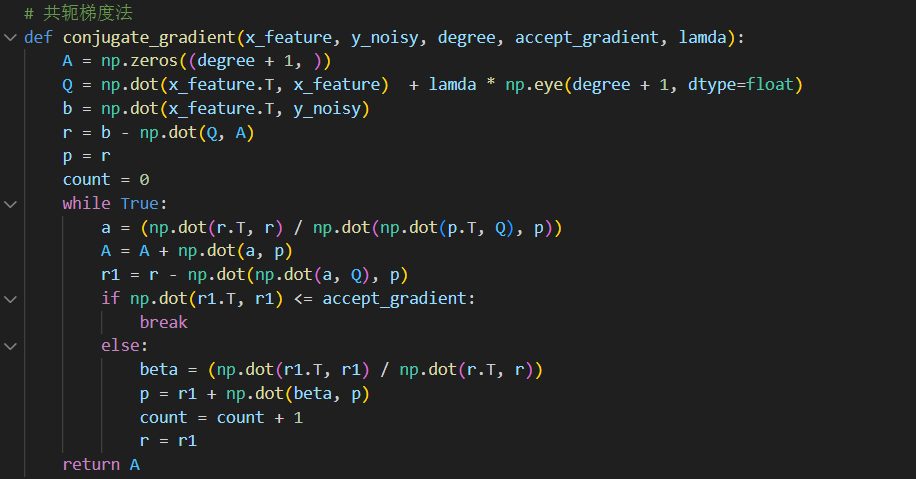


图4 共轭梯度下降代码截图

**4.3 实验结果及分析**

4.3.1 不同数据量下结果比较

设置多项式次数为5，生成噪声服从，分别使用梯度下降法和共轭梯度下降法进行结果的比较。

1. 使用梯度下降法，设置学习率为0.01，设定梯度阈值为0.01，可以看到随着数据量增加，拟合效果逐渐变好，同时也注意到梯度下降至指定阈值界限下所耗迭代次数增加。

图形用户界面, 图表

描述已自动生成

图5 梯度下降法数据量变化结果比较

1. 使用共轭梯度下降法，设置残差阈值界限为，同样可以看到随着数据量增加，拟合效果逐渐变好，但是残差下降至指定阈值界限下所耗迭代次数几乎不变，同时可以发现：**相较于梯度下降法，共轭梯度下降法所需迭代步数大大减少**。

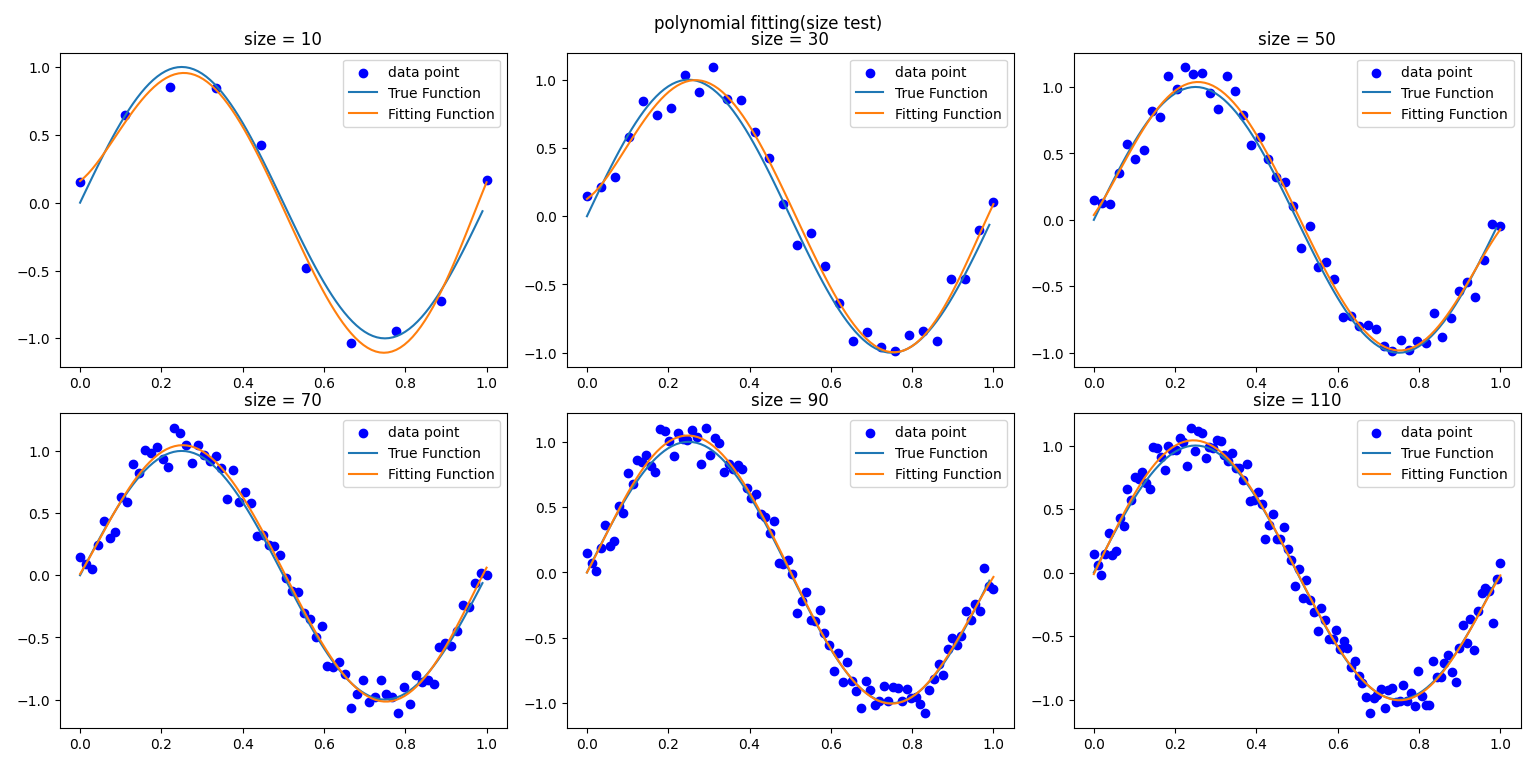


图6 共轭梯度下降法数据量变化结果比较

4.3.2 不同多项式阶数下结果比较

设置数据量为10，生成噪声服从（便于观察过拟合现象），分别使用梯度下降法和共轭梯度下降法进行结果的比较。

1. 使用梯度下降法，设定梯度阈值为0.01（高阶时一般在100000次迭代无法低于阈值），可以看到随着阶数增加，能够出现明显的曲线形状，但在100000次迭代，学习率为0.01的情况下仍较难收敛，因而此处不做分析。

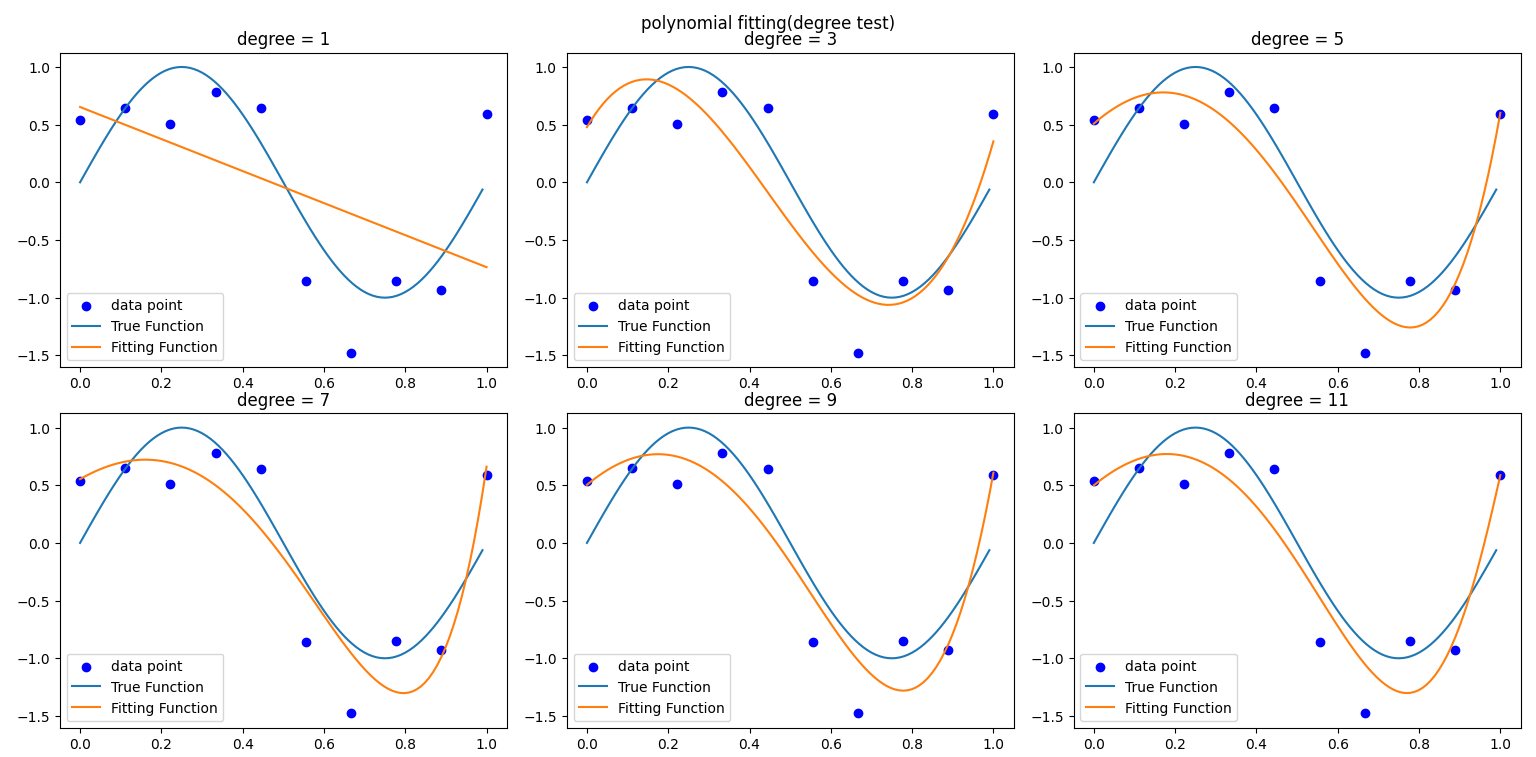


图7 梯度下降法多项式阶数变化结果比较

1. 使用共轭梯度下降法，设置残差阈值界限为，可以发现阶数较小时，模型表达能力有限，随着阶数增加初期，拟合效果逐渐变好；但是后期当阶数逐渐增大，曲线已不具有正弦形状，模型逐渐过拟合，即过度学习训练集上的数据特征（误差为0），而在验证集上失去表达能力。

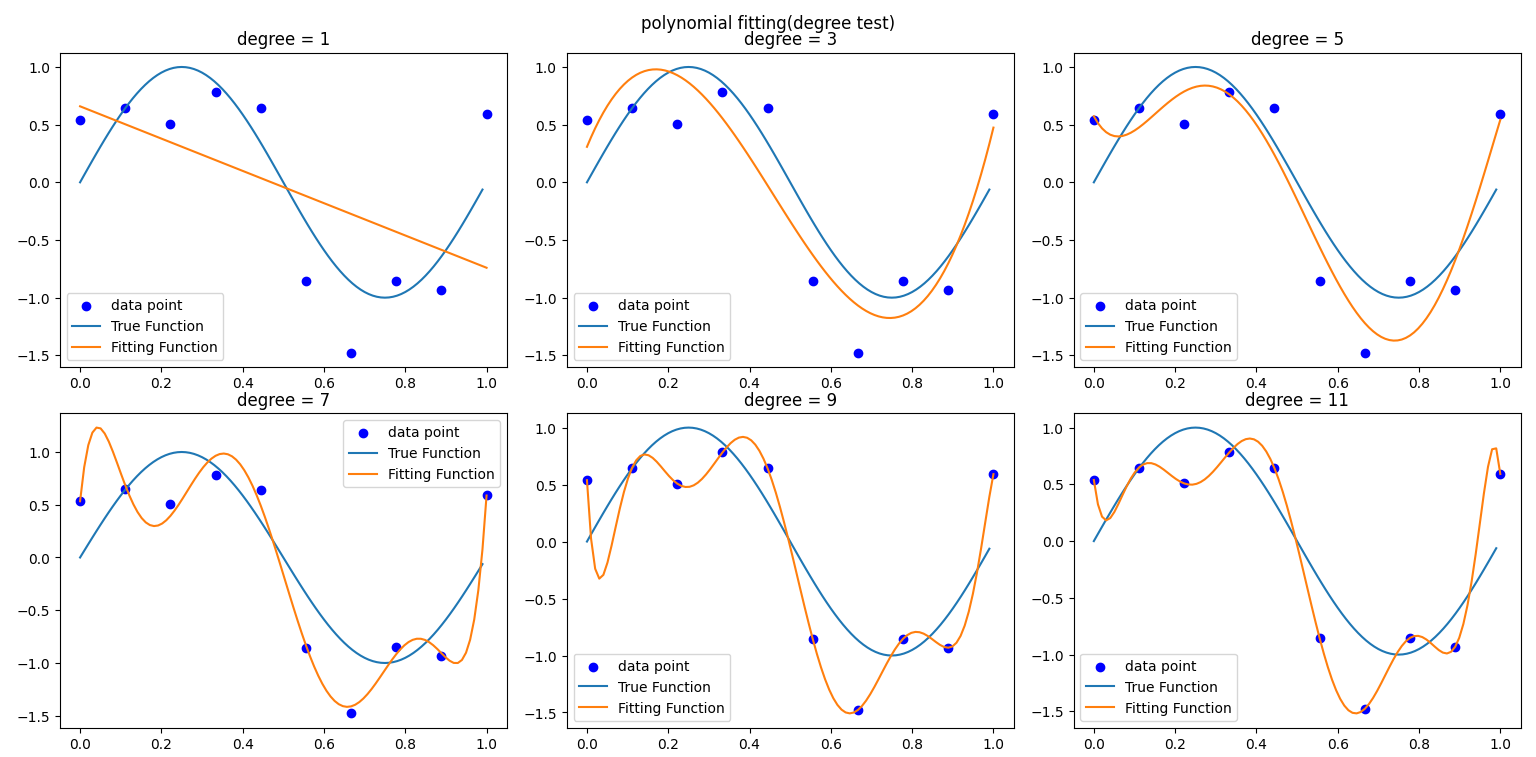


图8 共轭梯度下降法多项式阶数变化结果比较

观察RMS评价函数变化曲线也能够发现，随阶数增加，训练过程中RMS逐渐减少甚至等于0；而在测试集上当阶数大于5时，RMS陡然增加。

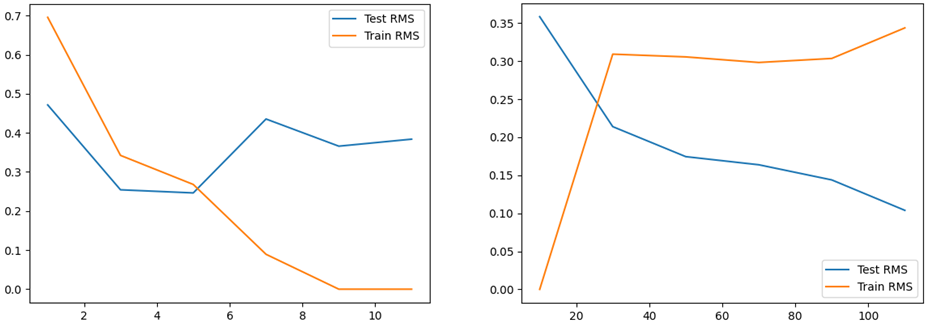


图9 共轭梯度下降法RMS变化

（左图为RMS随多项式阶数变化，右图为在阶数为10的情况下RMS随数据量变化）

4.3.3 过拟合及其处理

过拟合(Overfitting)是指机器学习模型过度适应训练数据，以至于在新数据上的表现变差。过拟合的主要原因可以归结为**模型过于复杂**，拥有过多参数或特征，以至于可以完全适应训练数据中的每个细节和噪声。这种过度适应导致模型在训练数据上表现优秀，但在未知数据上泛化能力较弱。

过拟合主要有两种方法去解决，在生成噪声服从不变的前提下：

1. 增大样本数量，可以看到随着样本数增加，过拟合程度明显减弱。

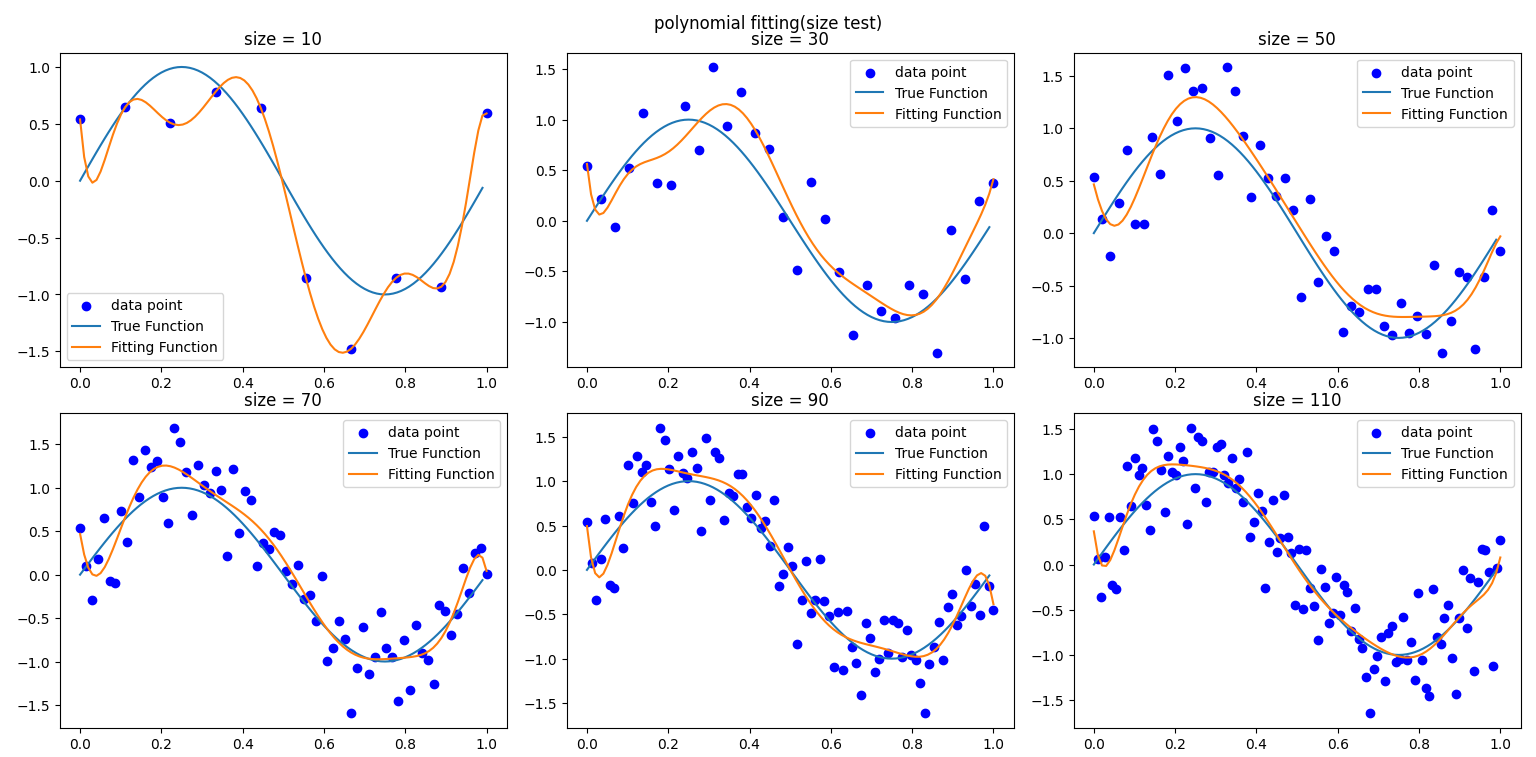


图 10 共轭梯度下降法数据量变化结果比较（）

1. 添加正则项，通过调整的大小可以不同程度的降低模型复杂度，从而减弱过拟合程度。可以看到，比重大时降低模型复杂度，表达能力也较差；比重适当时模型复杂度与问题匹配；比重小时退化成原模型，几乎无改变。

因此寻找一个适当的有助于有效降低模型复杂度，得到一个与问题匹配的模型。

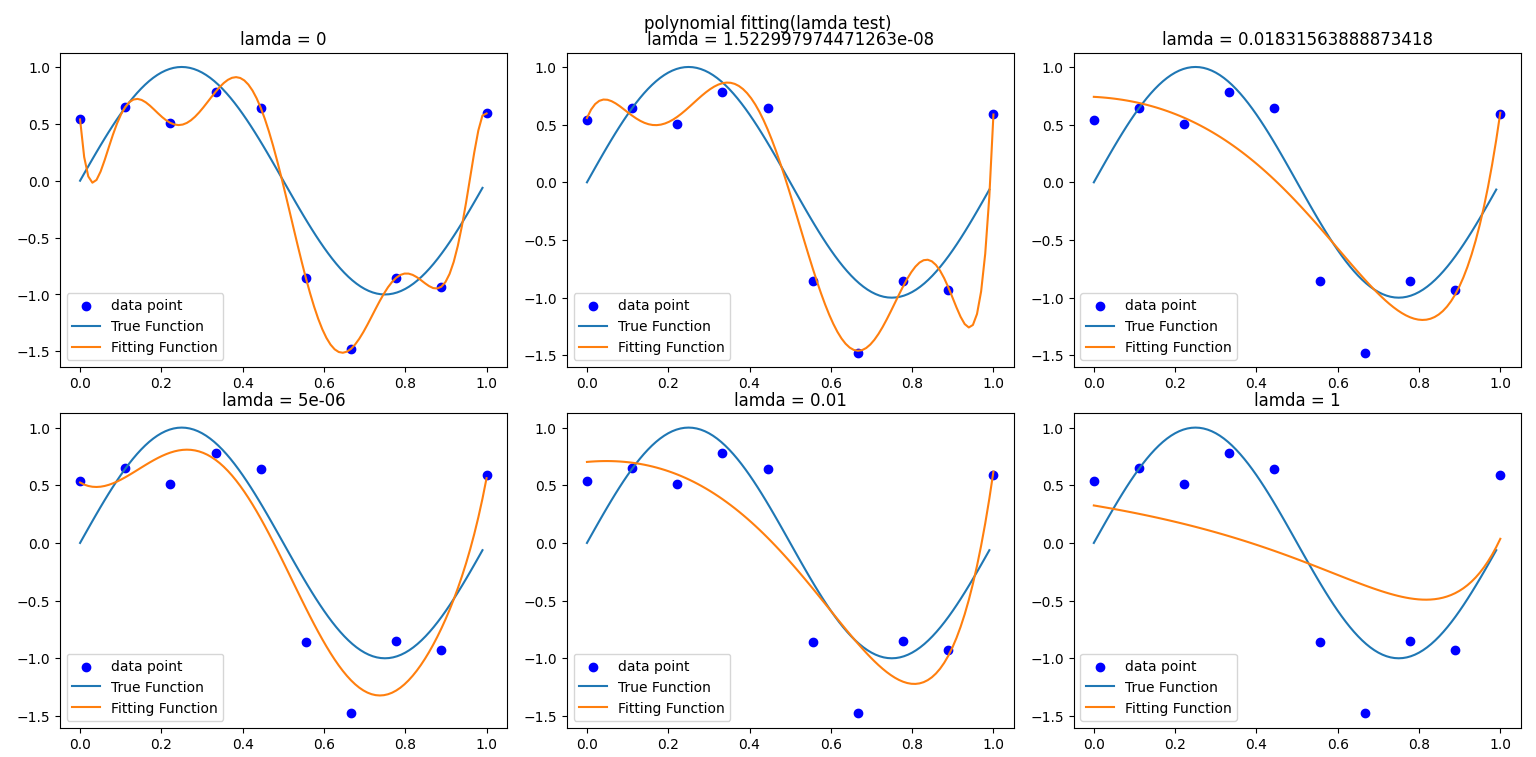


图11 共轭梯度下降法正则项强度变化结果比较（）

1. **实验结论**

最小二乘法直接使用数学公式求解，但是矩阵求逆存在不稳定性和误差，在高于10阶时调用并不能完全拟合样本点。

梯度下降法是基于一阶导数的优化方法，沿负梯度方向更新参数来寻找最优解。但在此多项式拟合问题中较难拟合。

共轭梯度法引入了共轭向量的概念，把目标函数分成许多方向，然后不同方向分别求出极值在综合起来。其仅利用了函数的一阶导数信息，但是克服了梯度下降收敛慢的缺点，同时又避免了牛顿法求二阶导计算量大的问题，能够有效解决二次优化问题。相较于梯度下降法，迭代次数大幅减少、复杂度更低。

模型阶数的选择影响模型的学习能力，较大的阶数可以增强模型的学习能力，但可能导致过拟合。过拟合情况取决于模型选用的阶数以及求解参数的方法，不同方法可能对过拟合的阶数产生不同影响。

增加训练样本可以帮助缓解过拟合问题，提高模型的泛化能力。引入惩罚项（如正则化）也可以控制模型复杂度，减少过拟合风险。但是正则化系数的选取需要经过足够的实验。

1. **完整实验代码**

polynomial\_fitting.py

|  |
| --- |
| 1. import numpy as np 2. import matplotlib.pyplot as plt 3. import math 4. *# 计算RMS误差函数* 5. def root\_mean\_square(given\_function, fitted\_function): 6. return np.sqrt(np.mean((given\_function - fitted\_function) \*\* 2)) 7. *# 绘制结果并计算RMS误差* 8. def draw(x, y\_noisy, degree, w): 9. x\_t = np.arange(0, 1, 0.01) 10. y\_t = np.sin(2 \* np.pi \* x\_t) 11. plt.scatter(x, y\_noisy, c='b', label='data point') 12. plt.title('polynomial fitting') 13. plt.plot(x\_t, y\_t, label='True Function') 14. test\_sample = 100 15. test\_x = np.linspace(0, 1, test\_sample) 16. test\_x\_feature = np.column\_stack([test\_x \*\* i for i in range(degree + 1)]) 17. test\_y = np.dot(test\_x\_feature, w) 18. ture\_y = np.sin(2 \* np.pi \* test\_x) 19. rms = root\_mean\_square(test\_y, ture\_y) 20. plt.plot(test\_x, test\_y, label='Fitting Function') 21. plt.legend() 22. plt.show() 23. return rms 24. *# 绘制多个图像并计算RMS误差* 25. def draw\_10(fig, axs, x, y\_noisy, degree, w, size, lamda, string, lr, n): 26. i = int((n - 1) / 3) 27. j = int((n - 1) - i \* 3) 28. x\_t = np.arange(0, 1, 0.01) 29. y\_t = np.sin(2 \* np.pi \* x\_t) 30. axs[i,j].scatter(x, y\_noisy, c='b', label='data point') 31. axs[i,j].plot(x\_t, y\_t, label='True Function') 32. test\_sample = 100 33. test\_x = np.linspace(0, 1, test\_sample) 34. test\_x\_feature = np.column\_stack([test\_x \*\* i for i in range(degree + 1)]) 35. test\_y = np.dot(test\_x\_feature, w) 36. ture\_y = np.sin(2 \* np.pi \* test\_x) 37. rms = root\_mean\_square(test\_y, ture\_y) 38. axs[i,j].plot(test\_x, test\_y, label='Fitting Function') 39. if string == 'degree': 40. axs[i,j].set\_title(f'{string} = ' + str(degree)) 41. elif string == 'size': 42. axs[i,j].set\_title(f'{string} = ' + str(size)) 43. elif string == 'lamda': 44. axs[i,j].set\_title(f'{string} = ' + str(lamda)) 45. elif string == 'lr': 46. axs[i,j].set\_title(f'{string} = ' + str(lr)) 47. axs[i,j].legend() 48. return rms 49. *# 生成数据* 50. def data\_generate(mean, var, size, degree, seed=None): 51. if seed is not None: 52. np.random.seed(seed) 54. x = np.linspace(0, 1, size) 55. y\_true = np.sin(2 \* np.pi \* x) 56. y\_noisy = np.sin(2 \* np.pi \* x) + np.random.normal(mean, var, len(x)) 57. x\_feature = np.column\_stack([x \*\* i for i in range(degree + 1)]) 58. return x, x\_feature, y\_true, y\_noisy 59. *# 最小二乘法* 60. def least\_square(X, Y, lamda, degree): 61. A = np.linalg.inv(np.dot(X.T, X) + lamda \* np.eye(degree+1)).dot(X.T).dot(Y) 62. return A 63. *# 共轭梯度法* 64. def conjugate\_gradient(x\_feature, y\_noisy, degree, alpha, accept\_gradient, lamda): 65. A = np.zeros((degree + 1, )) 66. Q = np.dot(x\_feature.T, x\_feature)  + lamda \* np.eye(degree + 1, dtype=float) 67. b = np.dot(x\_feature.T, y\_noisy) 68. r = b - np.dot(Q, A) 69. p = r 70. count = 0 71. while True: 72. a = (np.dot(r.T, r) / np.dot(np.dot(p.T, Q), p)) 73. A = A + np.dot(a, p) 74. r1 = r - np.dot(np.dot(a, Q), p) 75. if np.dot(r1.T, r1) <= accept\_gradient: 76. break 77. else: 78. beta = (np.dot(r1.T, r1) / np.dot(r.T, r)) 79. p = r1 + np.dot(beta, p) 80. count = count + 1 81. r = r1 82. print("Iterations:", count) 83. return A 84. *#梯度下降法* 85. def gradient\_descent(X, Y, degree, alpha, accept\_gradient, lamda, limit=100000): 86. iteration = 0 87. A = np.zeros(degree + 1) 88. while iteration < limit: 89. gradient\_w = np.dot(X.T, X).dot(A) - np.dot(X.T, Y) + lamda \* A 90. norm\_gradient\_w = np.linalg.norm(gradient\_w) 91. if norm\_gradient\_w < accept\_gradient: 92. break 93. iteration += 1 94. A -= alpha \* gradient\_w 95. print("Iterations:", iteration) 96. return A 97. *# 测试不同的多项式阶数，比较实验效果* 98. def test\_degree(): 99. fig, axs = plt.subplots(2, 3, figsize=(15, 10)) 100. rms\_test = [] 101. rms\_train = [] 102. n = 1 103. list = [1, 3, 5, 7, 9, 11] 104. for degree in list: 105. lamda = 0 106. mean = 0 107. size = 10 108. var = 0.36 109. alpha = 0.01 110. accept\_gradient = 1e-20 111. seed = 177 112. x, x\_feature, y\_true, y\_noisy = data\_generate(mean, var, size, degree, seed) 113. A = conjugate\_gradient(x\_feature, y\_noisy, degree, alpha, accept\_gradient, lamda) 114. y\_pred = np.dot(x\_feature, A) 115. rms\_train.append(root\_mean\_square(y\_noisy, y\_pred)) 116. x = x = draw\_10(fig, axs, x, y\_noisy, degree, A, size, lamda, 'degree', alpha, n) 117. n += 1 118. rms\_test.append(x) 119. print(A) 120. plt.suptitle('polynomial fitting(degree test)') 121. plt.tight\_layout() 122. plt.show() 123. plt.plot(list, rms\_test, label='Test RMS') 124. plt.plot(list, rms\_train, label='Train RMS') 125. plt.legend() 126. plt.show() 127. *# 测试不同的数据量，比较实验效果* 128. def test\_size(): 129. fig, axs = plt.subplots(2, 3, figsize=(15, 10)) 130. rms\_test = [] 131. rms\_train = [] 132. list = [10, 30, 50, 70, 90, 110] 133. n = 1 134. for size in list: 135. degree = 10 136. lamda = 0 137. mean = 0 138. var = 0.36 139. alpha = 0.01 140. accept\_gradient = 1e-20 141. seed = 177 142. x, x\_feature, y\_true, y\_noisy = data\_generate(mean, var, size, degree, seed) 143. A = conjugate\_gradient(x\_feature, y\_noisy, degree, alpha, accept\_gradient, lamda) 144. y\_pred = np.dot(x\_feature, A) 145. rms\_train.append(root\_mean\_square(y\_noisy, y\_pred)) 146. x = draw\_10(fig, axs, x, y\_noisy, degree, A, size, lamda, 'size', alpha, n) 147. n += 1 148. print(n) 149. rms\_test.append(x) 150. print(A) 151. plt.suptitle('polynomial fitting(size test)') 152. plt.tight\_layout() 153. plt.show() 154. plt.plot(list, rms\_test, label='Test RMS') 155. plt.plot(list, rms\_train, label='Train RMS') 156. plt.legend() 157. plt.show() 158. *# 测试不同学习率，比较实验效果* 159. def test\_lr(): 160. fig, axs = plt.subplots(2, 3, figsize=(15, 10)) 161. rms\_test = [] 162. rms\_train = [] 163. i = 1 164. list = [0.1, 0.01, 0.001, 0.0001, 1e-5, 1e-6] 165. for alpha in list: 166. degree = 5 167. lamda = 0 168. mean = 0 169. size = 10 170. var = 0.1 171. accept\_gradient = 0.01 172. seed = 177 173. x, x\_feature, y\_true, y\_noisy = data\_generate(mean, var, size, degree, seed) 174. A = gradient\_descent(x\_feature, y\_noisy, degree, alpha, accept\_gradient, lamda) 175. y\_pred = np.dot(x\_feature, A) 176. rms\_train.append(root\_mean\_square(y\_noisy, y\_pred)) 177. x = draw\_10(fig, axs, x, y\_noisy, degree, A, size, lamda, 'lr', alpha, i) 178. i += 1 179. rms\_test.append(x) 180. print(A) 181. plt.suptitle('polynomial fitting(learning rate test)') 182. plt.tight\_layout() 183. plt.show() 184. *# 测试不同lamda，比较实验效果* 185. def test\_lamda(): 186. fig, axs = plt.subplots(2, 3, figsize=(15, 10)) 187. rms\_test = [] 188. rms\_train = [] 189. i = 1 190. for lamda in [0, math.exp(-18), math.exp(-4), 5e-6, 0.01, 1]: 191. degree = 10 192. alpha = 0.01 193. mean = 0 194. size = 10 195. var = 0.36 196. accept\_gradient = 1e-20 197. seed = 177 198. x, x\_feature, y\_true, y\_noisy = data\_generate(mean, var, size, degree, seed) 199. A = conjugate\_gradient(x\_feature, y\_noisy, degree, alpha, accept\_gradient, lamda) 200. y\_pred = np.dot(x\_feature, A) 201. rms\_train.append(root\_mean\_square(y\_noisy, y\_pred)) 202. x = draw\_10(fig, axs, x, y\_noisy, degree, A, size, lamda, 'lamda', alpha, i) 203. i += 1 204. rms\_test.append(x) 205. print(A) 206. plt.suptitle('polynomial fitting(lamda test)') 207. plt.tight\_layout() 208. plt.show() 209. *#固定情况* 210. def main(): 211. degree = 5 *#阶数* 212. lamda = 0 *#惩罚项的lamda* 213. mean = 0 214. size = 10 *#训练样本个数* 215. var = 0.1 *#高斯噪声的方差* 216. alpha = 0.1 *#梯度下降法的步长* 217. accept\_gradient = 0.0001 *#梯度下降法阈值* 218. seed = 177 219. x, x\_feature, y\_true, y\_noisy = data\_generate(mean, var, size, degree, seed) *#生成数据* 220. A = gradient\_descent(x\_feature, y\_noisy, degree, alpha, accept\_gradient, lamda) *#梯度下降法* 221. draw(x, y\_noisy, degree, A) 222. print(A) 223. *# 过拟合情况* 224. def overfitting\_test(): 225. degree = 10 *#阶数* 226. lamda =  0.00*#惩罚项的lamda* 227. mean = 0 228. size = 10 *#训练样本个数* 229. var = 0.36 *#高斯噪声的方差* 230. alpha = 0.1 *#梯度下降法的步长* 231. accept\_gradient = 1e-20 *#梯度下降法阈值* 232. seed = 1314 233. x, x\_feature, y\_true, y\_noisy = data\_generate(mean, var, size, degree, seed) *#生成数据* 234. A = conjugate\_gradient(x\_feature, y\_noisy, degree, alpha, accept\_gradient, lamda) *#梯度下降法* 235. draw(x, y\_noisy, degree, A) 236. print(A) 237. if \_\_name\_\_ == '\_\_main\_\_': 238. overfitting\_test() 239. main() 240. test\_lamda() 241. test\_degree() 242. test\_size() 243. test\_lr() |

1. **参考文献**

[1] 李航. 统计学习方法[M]. 清华大学出版社, 2012.

[2] 周志华. 机器学习[M]. 清华大学出版社, 2016.

[3] 深入理解L1、L2正则化https://zhuanlan.zhihu.com/p/29360425

[4] 最小二乘公式推导https://blog.csdn.net/qq\_45717425/article/details/120665970

[5] 数据拟合：直线拟合-多项式拟合https://blog.csdn.net/qq\_34777600/article/ details/79501932

[6] 共轭梯度法(Conjugate gradient)详解https://blog.csdn.net/bitcarmanlee/article/ details/121522734