哈尔滨工业大学计算机科学与技术学院
实验报告

课程名称: 机器学习

课程类型: 选修

实验题目: 多项式拟合正弦曲线

学号: 1190202110

姓名: 田雪洋

2021 年 9 月 28 日

一、实验目的

掌握最小二乘法求解（无惩罚项的损失函数），掌握增加惩罚项（2 范数）的损失函数优化，梯度下降法、共轭梯度法，理解过拟合、客服过拟合的方法（如增加惩罚项、增加样本）。

二、实验要求及实验环境

1. 实验要求

1. 生成数据，加入噪声；
2. 用高阶多项式函数拟合曲线；
3. 用解析解求解两种 loss 的最优解（无正则项和有正则项）
4. 优化方法求解最优解（梯度下降，共轭梯度）；
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. 语言不限，可以用 matlab，python。求解解析解时可以利用现成的矩阵求逆。梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不许用现成的平台，例如 pytorch，tensorflow 的自动微分工具。

2. 实验环境

Windows10; python3.8.6;Pycharm

三、设计思想（本程序中的用到的主要算法及数据结构）

对于一个多项式函数

$$y(x,w) = w_0 + w_1 x + \cdots + w_m x^m = \sum_{i=0}^{m} w_i x^i \tag{1}$$

当 m 足够大时，可以拟合任意曲线，本次实验使用多项式来拟合正弦函数 $sin(2\pi x)$。

1. 生成数据

首先，先产生若干 x 位于 [0,1] 的正弦函数 $sin(2\pi x)$ 的样本，并且给每一个目标值增加一个均值为 0，方差为 0.09 的高斯噪声。代码如下:

```
def data_produce(size,M):
    # 在 [0,2pi] 之间生成 size 个数据
    x=np.linspace(0,1,size)
    # 生成标准差为 0.09, 均值为 0 的高斯分布噪声
    gauss_noise=np.random.normal(0,0.09,size=size)
    y=np.sin(2*np.pi*x)
    # 生成测试数据集，并添加高斯噪声
    x_train=x
    y_train=y+gauss_noise
    X = []
    for i in range(M + 1):
        X.append(x_train ** i)
        X = np.array(X).T
    return x,y,x_train,y_train,X
```

2. 最小二乘法（无正则项）

采用最小二乘法,即采用如下损失函数来衡量真实值 Y 和预测值 $f(x,w)=Xw$ 之间的误差：

$$E\left(w\right)=\frac{1}{2}\cdot\left(X\cdot w-Y\right)^{2} \tag{2}$$

当上述损失函数最小时，即拟合效果最佳。因此，令

$$\frac{\partial E}{\partial \boldsymbol{w}}=\boldsymbol{X}^{T}\boldsymbol{X}\boldsymbol{w}-\boldsymbol{X}^{T}\boldsymbol{Y}=0 \tag{3}$$

得到解析解为

$$\boldsymbol{w}=\left(\boldsymbol{X}^{T}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{T}\boldsymbol{Y} \tag{4}$$

其

$$\mathbf{X}=\begin{bmatrix}1 & x_1 & \cdots & x_1^m\\ 1 & x_2 & \cdots & x_2^m\\ \vdots & \vdots & \ddots & \vdots\\ 1 & x_N & \cdots & x_N^m\end{bmatrix}$$

这样，便可以编程得到参数 w 的取值，进而求得 $f(x,w)$ 函数。这便是最小二乘法拟合的思想。代码如下：

```
def lsm(X,y_train,lamda):
    # 最小二乘法得到多项式函数的参数 w, lamda 即正则项的超参数,
    # 当 lamda=0 时，为无正则项的最小二乘法算法,
    # 当 lamda! =0 时，为带正则项的最小二乘法算法。
    w=np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)
+   lamda*np.eye(X.shape[1])),X.T),y_train)
    # 求得多项式函数的解析式为:
    pre=np.poly1d(w[::-1])
    return w,pre
def loss(X,y_train,w,lamda):
    # 计算损失函数 E(w)
    return
    1/2*np.dot((X.dot(w)-y_train).T,(X.dot(w)-y_train))
    +0.5*lamda*np.dot(w.T,w)
```

3. 最小二乘法（含正则项）

通常使用最小二乘法拟合时，可能出现过拟合的现象。因此为了避免过拟合的出现，我们在通常在误差函数上加上惩罚项来避免过拟合的出现。因

此，我们得到了

$$E(w)=\frac{1}{2}\cdot(X\cdot w-Y)^2 \tag{5}$$

对上述式子求偏导得到

$$\frac{\partial\widetilde{E}}{\partial\boldsymbol{w}}=\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}-\boldsymbol{X}^T\boldsymbol{Y}+\lambda\boldsymbol{w} \tag{6}$$

令偏导数等于 0 求得 w 为

$$\boldsymbol{w}=\left(\boldsymbol{X}^T\boldsymbol{X}+\lambda\right)^{-1}\boldsymbol{X}^T\boldsymbol{Y} \tag{7}$$

代码和无正则项的代码相同，参数 lamda 即正则项的超参数 λ, 当 lamda=0 时，为无正则项的最小二乘法算法，当 lamda！=0 时，为带正则项的最小二乘法算法。

4. 梯度下降法

梯度下降法是用来计算函数最小值的。它和下山的过程很像，采用逐步逼近的方式求取极值。梯度下降法首先在函数上随机选定一点，然后计算该点的梯度。因为梯度的方向就是函数之变化最快的方向，所以沿着梯度下降最快的方向前进，逐步逼近函数的极值点。在本次实验中，即使损失函数最小，并且记录当前的 w。

对于式子 (6)，我们设置学习速率 α 和迭代次数。w 的迭代方程为

$$\boldsymbol{w}=\boldsymbol{w}-\alpha\cdot\frac{\partial\widetilde{E}}{\partial\boldsymbol{w}} \tag{8}$$

由于是采用设置迭代次数的方式，所以当新的损失函数大于原来的损失函数时，说明已经过来极值点了，此时，将学习速率减半，重新迭代，直到完成迭代次数。代码如下：

```
def Gradient_descent(X,y_train,deta,lamda,times):
    # 梯度下降法
    # 设置初始的 w 的值
    w=np.zeros(X.shape[1])
    w0=w
    # 计算初始的损失函数的取值
    loss_0=loss(X,y_train,w,lamda)
```

```
    # 根据设置的迭代次数，进行迭代
    for i in range(1,times+1):
        #E（w）对 w 求偏导

        gradient=((X.T).dot(X)).dot(w)-np.dot(X.T,y_train)+lamda*w
        #w 的迭代方程，deta 为学习速率
        w=w-deta*gradient
        # 计算新的损失函数
        loss_new=loss(X,y_train,w,lamda)
        # 当损失函数变大时，学习速率减半
        if(loss_new<loss_0):
            loss_0=loss_new
            w0=w
        else:
            deta=deta*0.5
    # 得到多项式函数
    pre = np.poly1d(w0[::-1])
    return w0,pre,loss_0
```

5. 共轭梯度法

共轭梯度法是用来求解线性方程 $Ax = b$。但共轭梯度法并不能用来解任意的线性方程。它要求系数矩阵是对称且正定的矩阵。根据式 (6)，显然该结果是符合要求的。因此首先令

$$A = X^T \cdot X + \lambda \cdot I, B = X^T \cdot Y \tag{9}$$

然后令 $r_0 = Ax_0 - b, p_0 = -r_0$。接着，得到如下算法：

$$\begin{aligned} \alpha_k &= \frac{r_k^T r_k}{p_k^T A p_k} \\ w_{k+1} &= w_k + \alpha_k p_k \\ r_{k+1} &= r_k + \alpha_k A p_k \\ \beta_{k+1} &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\ p_{k+1} &= -r_{k+1} + \beta_{k+1} p_k \end{aligned} \tag{10}$$

该算法一直执行，直到 $r_k^T r_k$ 不满足设定的精度。代码如下：

```
def Conjugate_gradient(X,y_train,lamda,epsilon):
    # 共轭梯度法
    # 计算 A,b
    A=X.T.dot(X)+lamda*np.eye(X.shape[1])
    b=X.T.dot(y_train)
    # 初始化 w, r,p, 将 rkT*rk 保留
    w = np.zeros(X.shape[1])
    r=A.dot(w)-b
    p=-r
    t=r.T.dot(r)
    # 循环, 直到 t 不满足精度为止
    while t>epsilon:
        # 算法迭代
        alpha=t/p.T.dot(A).dot(p)
        w=w+alpha*p
        r=r+alpha*A.dot(p)
        beta=r.T.dot(r)/t
        p=-r+beta*p
        t=r.T.dot(r)
    # 得到多项式函数的解析式
    pre = np.poly1d(w[::-1])
```

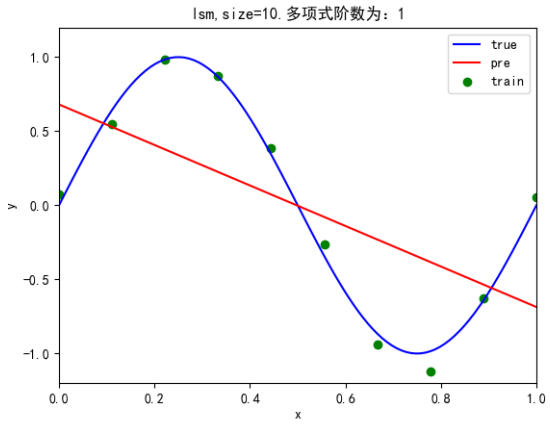
```
    return w,pre
```

四、实验结果与分析

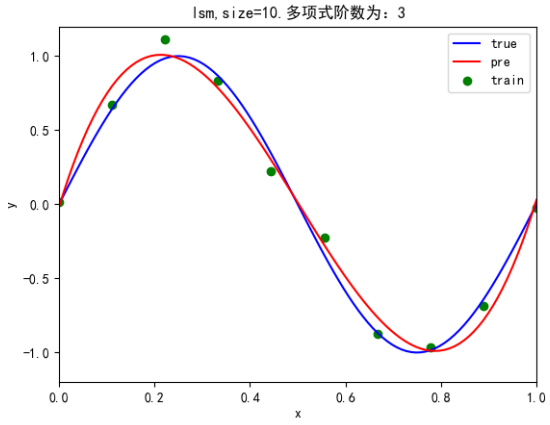
1. 无正则项的解析解

当训练集的数据量为 10 时，不同阶数的拟合效果：

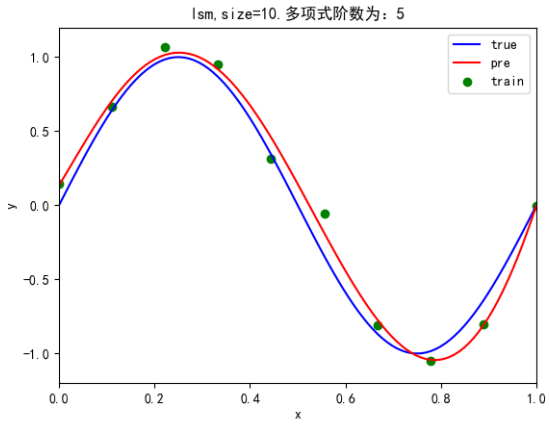
- 拟合函数解析为一次函数



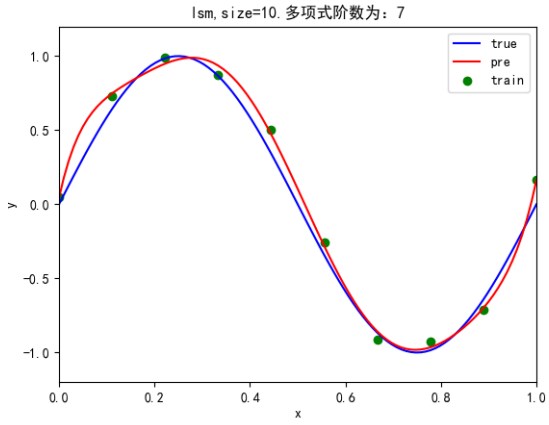
- 拟合函数解析为三次函数



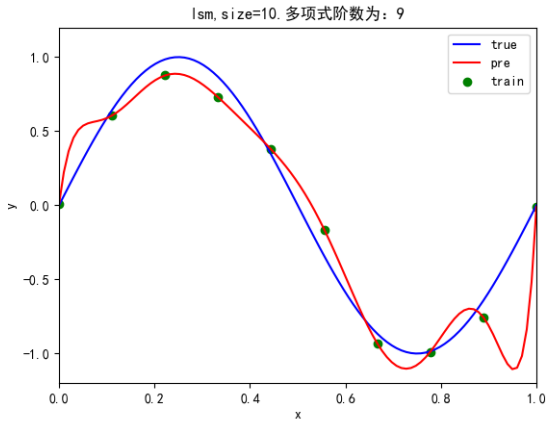
- 拟合函数解析为五次函数



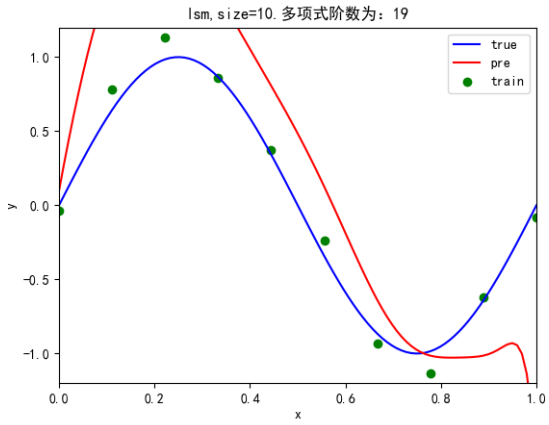
- 拟合函数解析为七次函数



- 拟合函数解析为九次函数

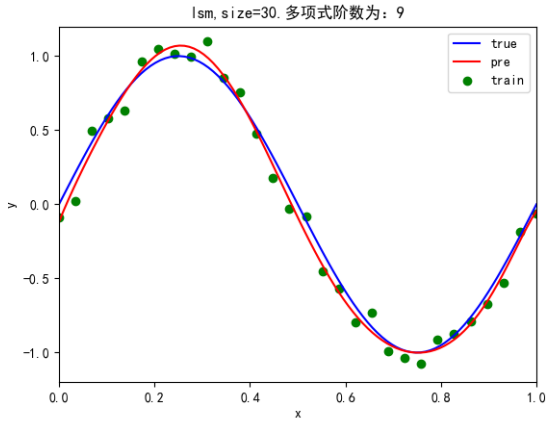


- 拟合函数解析为十九次函数

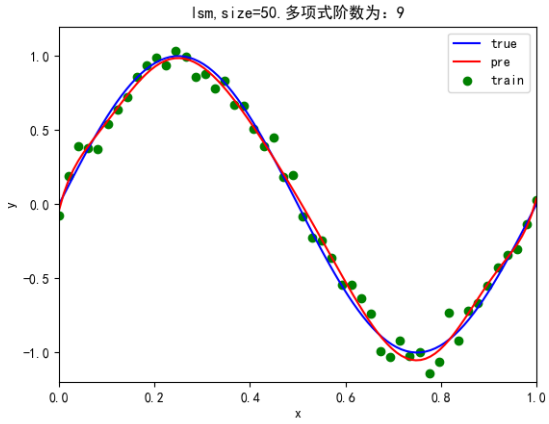


可以看到，拟合函数为一次函数时，预测函数和原函数差别较大，基本无法拟合，此时明显为欠拟合的状态。当拟合函数的次数为 3 次，5 次时拟合效果已经很好了，预测曲线和原曲线已经基本上重合，此时的方根均值分别为：0.0545 和 0.0177，拟合效果极佳。但是，当拟合函数的次数增加到 7 次以后，7 次函数和 9 次函数已经全部穿过了测试点，但是可以看到，曲线已经比原曲线相差甚远，已经开始扭曲了，当次数继续增加到 19 次时，预测曲线已经彻底扭曲，和原曲线相差越来越大，过拟合的现象越来越严重。当我们逐渐增加训练集的样本数量时，对于九次函数结果如下：

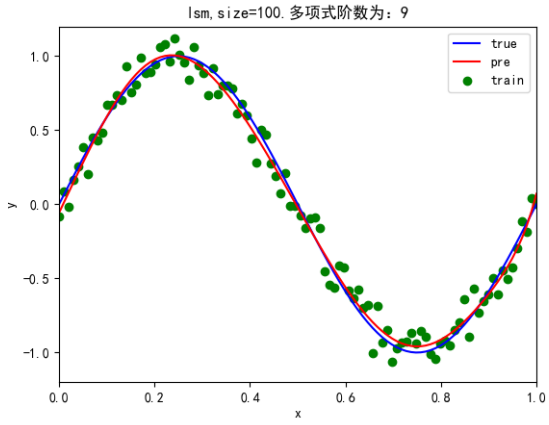
- 训练集样本容量为 30



- 训练集样本容量为 50



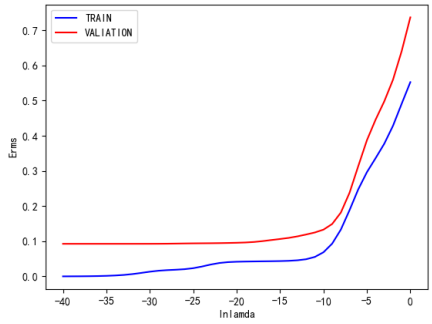
- 训练集样本容量为 100



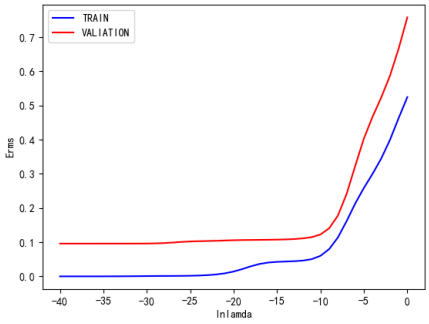
可以看到，随着样本容量的增加，曲线的拟合效果越来越好，这说明：增加训练集样本容量可以消除过拟合现象。

2. 含正则项的解析解

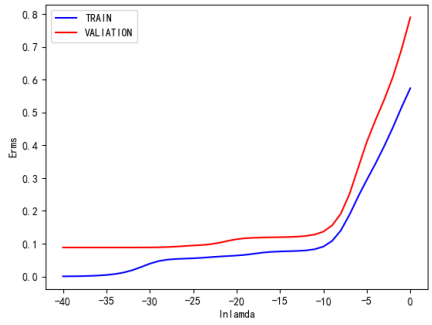
不仅增加样本容量可以消除过拟合现象，增加正则项同样可以消除过拟合现象，下面我们研究不同取值的参数对拟合效果的影响。我们将使用均方根来评价不同参数取值的拟合效果。其中训练集的样本容量为 10，验证集样本容量为 20，多项式的阶数为 9，多次运行的拟合效果如下：



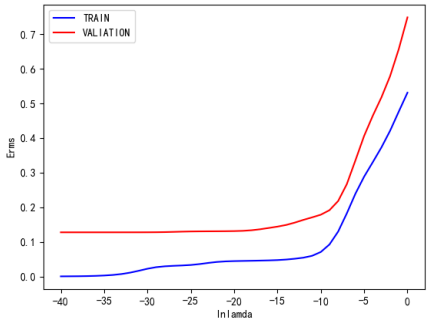
(a) pic1.



(b) pic2.

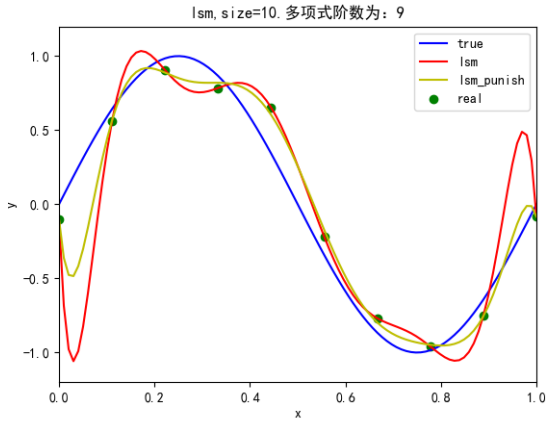


(c) pic3.



(d) pic4.

可以看到，当参数 lamda 在 $\left(e^{-40}, e^{-10}\right)$ 之间时均方差几乎保持不变，且训练集的接近 0，在测试集上保持稳定。故取 $\lambda = e^{-30}$ 时，把在训练集样本容量为 10，阶数为 9 的条件下的带正则项和不带正则项的最小二乘法拟合的图像进行比较。

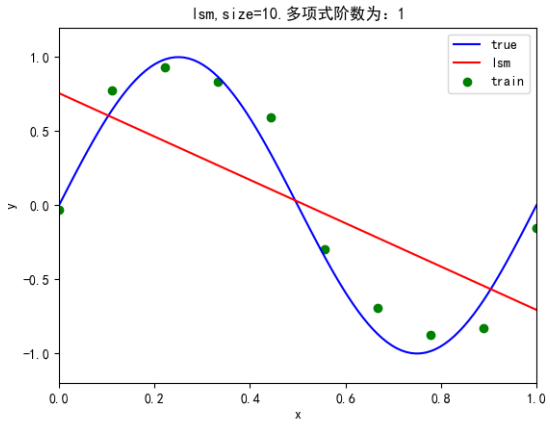


可以看到，加入正则项后的确有助于减轻过拟合现象，提高拟合效果。

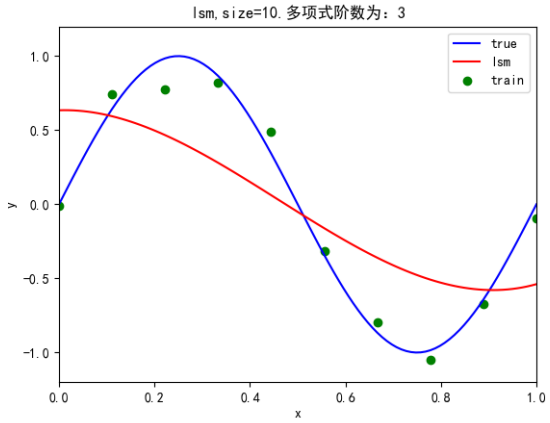
3. 梯度下降法求得优化解

取 $\lambda = e^{-12}$ 时，使用梯度下降法，初始学习速率为 0.01，迭代 100000 次，训练集的样本容量为 10，得到的不同阶数下的拟合图像如下

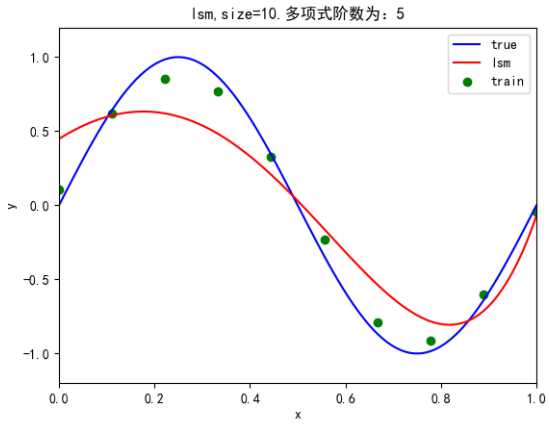
- 拟合函数解析为一次函数



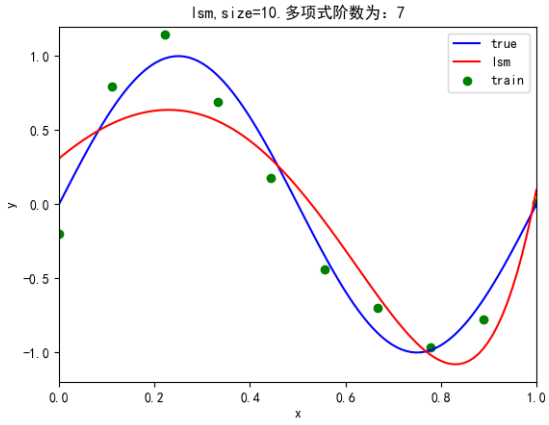
- 拟合函数解析为三次函数



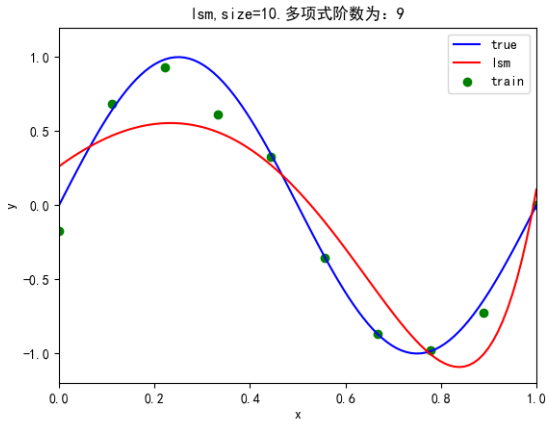
- 拟合函数解析为五次函数



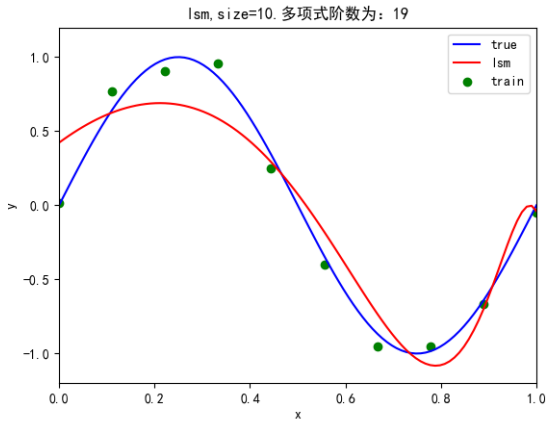
- 拟合函数解析为七次函数



- 拟合函数解析为九次函数

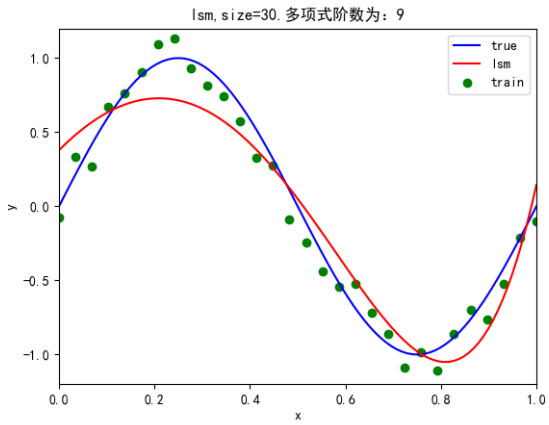


- 拟合函数解析为十九次函数

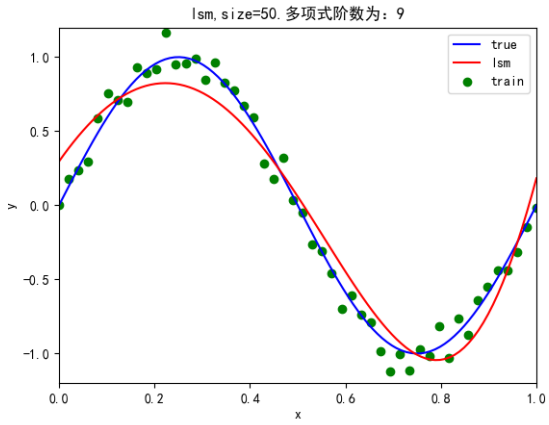


当固定拟合函数的阶数为 9，训练集的样本容量增大，得到的结果如下

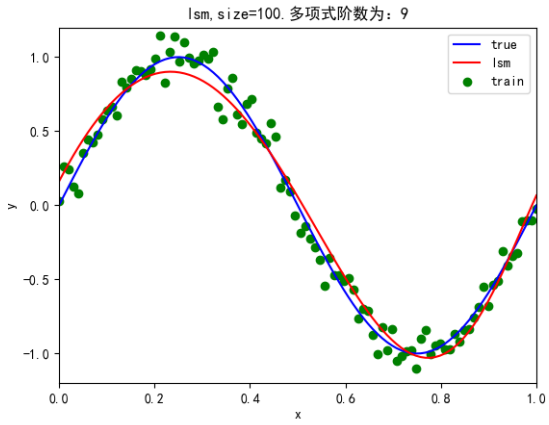
- 训练集样本容量为 30



- 训练集样本容量为 50



- 训练集样本容量为 100

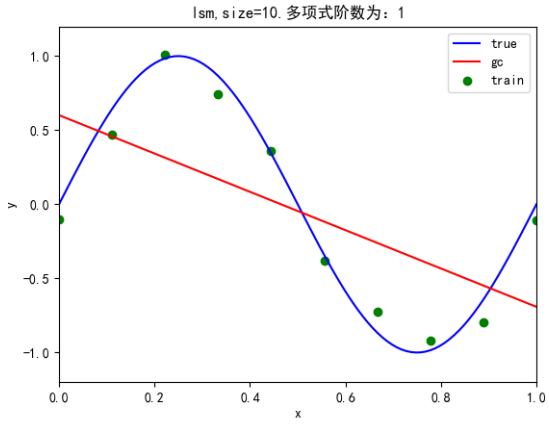


可以看到，在样本容量较少时，梯度下降法的效果不是很理想，远不如最小二乘法，但随着样本容量的增大，梯度下降法的拟合效果逐渐变优。

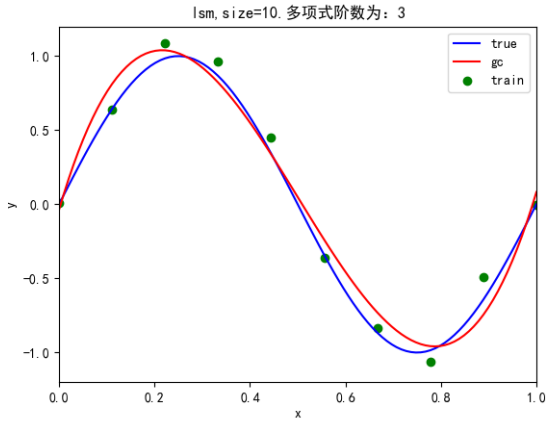
4. 共轭梯度法求得优化解

取 $\lambda = e^{-10}$ 时，使用共轭梯度法，精度为 e^{-6}，训练集的样本容量为 10，得到的不同阶数下的拟合图像如下

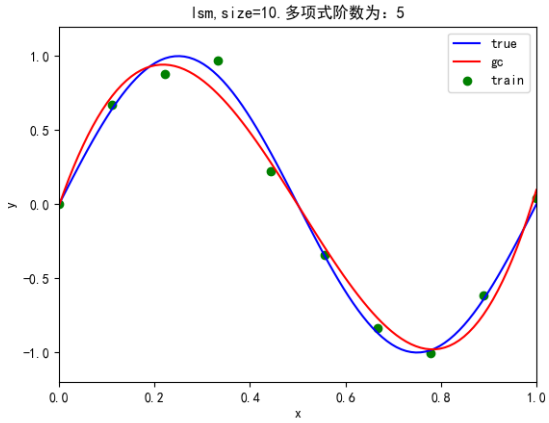
- 拟合函数解析为一次函数



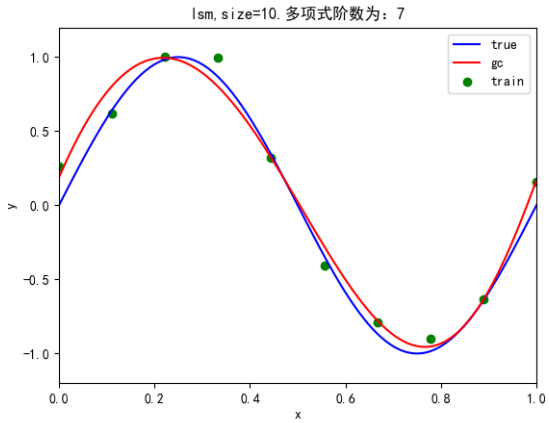
- 拟合函数解析为三次函数



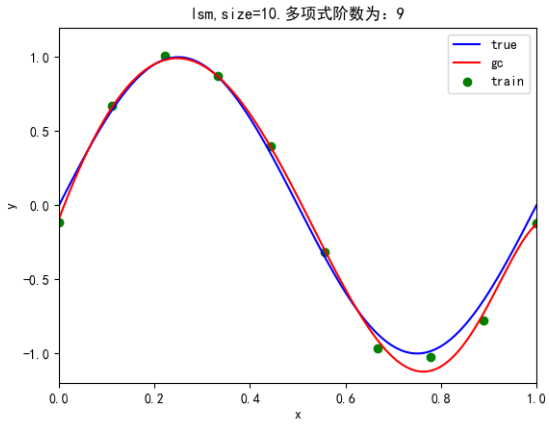
- 拟合函数解析为五次函数



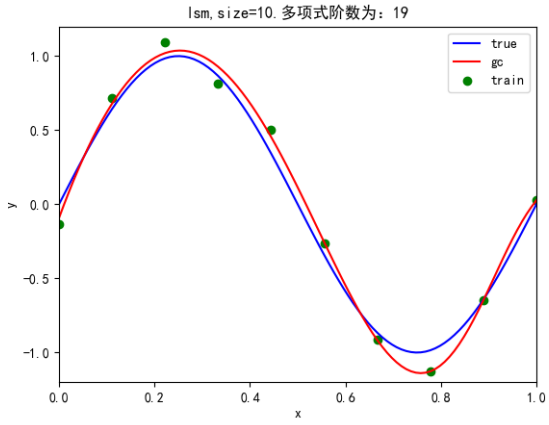
- 拟合函数解析为七次函数



- 拟合函数解析为九次函数

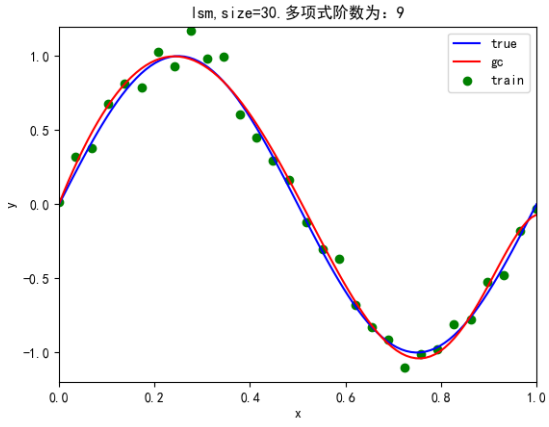


- 拟合函数解析为十九次函数

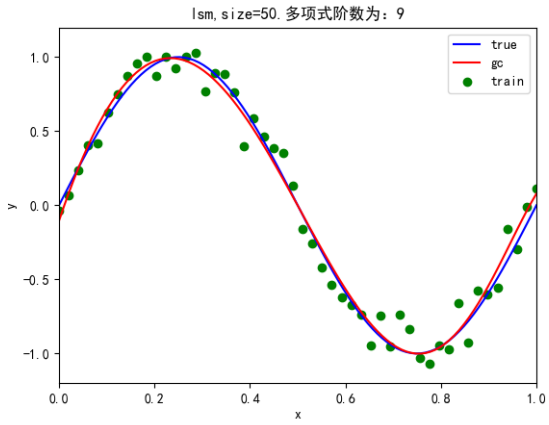


当我们逐渐增加训练集的样本数量时，对于九次函数结果如下：

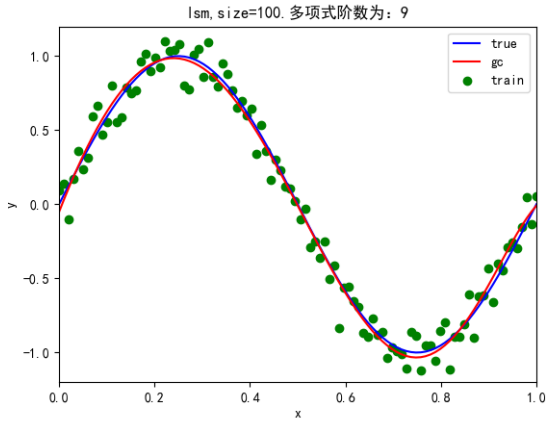
- 训练集样本容量为 30



- 训练集样本容量为 50



- 训练集样本容量为 100



可以看到，随着样本容量的增加，曲线的拟合效果越来越好。

5. 四种方法拟合对比

通过对上面的图像分析可以得到。在上面的四种方法中，最小二乘法带正则项的拟合效果最好，其次是不带正则项的最小二乘法和共轭梯度法。梯度下降法在样本容量较小时拟合效果较差，但随着样本容量的提升，梯度下降法的拟合效果也在逐渐变好。

五、结论

在本次实验的多项式函数拟合中，我们可以看到多项式的次数越高，拟合效果越好，但是可能会出现过拟合的现象，但是我们可以通过增加样本

容量和正则项的方法来减轻过拟合现象。

这过拟合现象出现的原因是由于样本数量少但拟合能力过强，导致拟合结果过分依赖已知的，少量的数据，其泛化能力减弱造成的，这种过强的拟合效果无法拟合出正弦曲线，其曲线会因尽量穿过已知点而被过分扭曲。所以增大样本容量可以减轻过拟合现象的出现，甚至消除。另一方面，在拟合函数中加入参数的正则项后，过拟合现象得到减轻。这是因为当参数数量变多时，模型的复杂度降低，从而使之与目标曲线相适应。因此，增加正则项是减轻过拟合现象的手段之一。

梯度下降法相比共轭梯度收敛速度很慢，迭代次数很大，而且性能较差。但共轭梯度法适用条件更差，它必须要求线性方程的系数矩阵正定且对称，因此它的适用范围更小。但是若是满足条件，共轭梯度法的性能要明显高于梯度下降法。

六、参考文献

七、附录：源代码（带注释）

exp1.py

```
import math

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


def data_produce(size,M):
    # 在 [0,2pi] 之间生成 size 个数据
    x=np.linspace(0,1,size)
```

```
    # 生成标准差为 0.09，均值为 0 的高斯分布噪声
    gauss_noise=np.random.normal(0,0.09,size=size)
    y=np.sin(2*np.pi*x)
    # 生成测试数据集，并添加高斯噪声
    x_train=x
    y_train=y+gauss_noise
    X = []
    for i in range(M + 1):
        X.append(x_train ** i)
        X = np.array(X).T
    return x,y,x_train,y_train,X
def lsm(X,y_train,lamda):
    # 最小二乘法得到多项式函数的参数 w，lamda 即正则项的超参数，
    # 当 lamda=0 时，为无正则项的最小二乘法算法，
    # 当 lamda! =0 时，为带正则项的最小二乘法算法。

    w=np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)+lamda*np.eye(X.shape[
    # 求得多项式函数的解析式为：
    pre=np.poly1d(w[::-1])
    return w,pre
    def loss(X,y_train,w,lamda):
    # 计算损失函数 E(w)
    return
    1/2*np.dot((X.dot(w)-y_train).T,(X.dot(w)-y_train))+lamda/2*np.do
def ERMS(loss,size):
    # 计算方根均值 Erms
    return np.sqrt(2 * loss / size)
def Gradient_descent(X,y_train,deta,lamda,times):# 梯度下降法
    # 设置初始的 w 的值
    w=np.zeros(X.shape[1])
```

```
    w0=w
    # 计算初始的损失函数的取值
    loss_0=loss(X,y_train,w,lamda)
    # 根据设置的迭代次数，进行迭代
    for i in range(1,times+1):
        #E（w）对 w 求偏导

        gradient=((X.T).dot(X)).dot(w)-np.dot(X.T,y_train)+lamda*w
        #w 的迭代方程，deta 为学习速率
        w=w-deta*gradient
        # 计算新的损失函数
        loss_new=loss(X,y_train,w,lamda)
        # 当损失函数变大时，学习速率减半
        if(loss_new<loss_0):
            loss_0=loss_new
             w0=w
        else:
             deta=deta*0.5
    # 得到多项式函数
    pre = np.poly1d(w0[::-1])
    return w0,pre,loss_0
def Conjugate_gradient(X,y_train,lamda,epsilon):
    # 共轭梯度法
    # 计算 A,b
    A=X.T.dot(X)+lamda*np.eye(X.shape[1])
    b=X.T.dot(y_train)
    # 初始化 w，r,p，将 rkT*rk 保留
    w = np.zeros(X.shape[1])
    r=A.dot(w)-b
    p=-r
```

```
69      t=r.T.dot(r)
70      # 循环，直到 t 不满足精度为止
71      while t>epsilon:
72      # 算法迭代
73          alpha=t/p.T.dot(A).dot(p)
74          w=w+alpha*p
75          r=r+alpha*A.dot(p)
76          beta=r.T.dot(r)/t
77          p=-r+beta*p
78          t=r.T.dot(r)
79          # 得到多项式函数的解析式
80      pre = np.poly1d(w[::-1])
81      return w,pre
82  def plot_lsm(size,lamda,a):
83      # 生成数据集
84      x,y,x_train,y_train,X=data_produce(size,a)
85      # 最小二乘法预测
86      w,pre=lsm(X,y_train,lamda)
87      w_loss,pre_loss=lsm(X,y_train,math.exp(-30))
88      # 打印方差均根
89      print(ERMS(loss(X,y_train,w,lamda),size))
90      print(pre)
91      plt.xlabel("x")
92      plt.ylabel("y")
93      plt.xlim(0,1)
94      plt.ylim(-1.2,1.2)
95      plt.rcParams['font.sans-serif'] = ['SimHei']
96      plt.rcParams['axes.unicode_minus'] = False
97      plt.title("lsm,size="+str(size)+". 多项式阶数为: "+str(a))
98      x = np.linspace(0, 1, 100)
```

```
99      y = np.sin(x * 2 * np.pi)
100     plt.plot(x,y,color='b',label='true')
101     plt.scatter(x_train,y_train,color='g',label='real')
102     plt.plot(x,pre(x),color='r',label='lsm')
103     plt.plot(x, pre_loss(x),color='y' ,label='lsm_punish')
104     plt.legend()
105     plt.savefig(str(size)+".png")
106 def plot_lsm_lamda(size,a=9):
107     x,y,x_train1,y1_train,x_train=data_produce(size,a)
108     x, y, x_test1, y1_test, x_test = data_produce(size*2,a)
109     # 划分数据集为测试集，训练集，验证集
110     #x_train, x_test, y1_train, y1_test = train_test_split(X,
        ↪  y_train, random_state=1, test_size=10)
111     lamda=np.linspace(-40,0,40).astype(int)
112     ERMtest=[]
113     ERMtrain=[]
114     # 对于测试集和预测集分别计算均方根误差
115     for i in lamda:
116         t=math.exp(i)
117         w_test,pre=lsm(x_test,y1_test,t)
118
            ↪  ERMtest.append(ERMS(loss(x_test,y1_test,w_test,t),size))
119         w_train, pre = lsm(x_train, y1_train, t)
120         ERMtrain.append(ERMS(loss(x_train, y1_train, w_train,
            ↪  t), size))
121     plt.xlabel("lnlamda")
122     plt.ylabel("Erms")
123     plt.rcParams['font.sans-serif'] = ['SimHei']
124     plt.rcParams['axes.unicode_minus'] = False
125     minERM=1.0
```

```
126     minlamda=0
127     # 找出均方根误差最小值
128     for i in range(0,len(ERMtest)):
129         if minERM>=ERMtest[i]:
130             minERM=ERMtest[i]
131             minlamda=lamda[i]
132     print(minERM)
133     plt.plot(lamda,ERMtrain,color='b',label='TRAIN')
134     plt.plot(lamda,ERMtest,color='r',label='VALIATION')
135     plt.legend()
136     plt.savefig("erm4.png")
137 def plot_Gu(size,lamda,a):
138     # 生成数据集
139     x,y,x_train,y_train,X=data_produce(size,a)
140     # 梯度下降法
141     times=10000
142     w,pre,loss_0=Gradient_descent(X,y_train,0.01,lamda,times)
143     # 打印方差均根
144     print(ERMS(loss_0,size))
145     print(pre)
146     plt.xlabel("x")
147     plt.ylabel("y")
148     plt.xlim(0,1)
149     plt.ylim(-1.2,1.2)
150     plt.rcParams['font.sans-serif'] = ['SimHei']
151     plt.rcParams['axes.unicode_minus'] = False
152     plt.title("lsm,size="+str(size)+". 多项式阶数为: "+str(a))
153     x = np.linspace(0, 1, 100)
154     y = np.sin(x * 2 * np.pi)
155     plt.plot(x,y,color='b',label='true')
```

```
156     plt.scatter(x_train,y_train,color='g',label='train')
157     plt.plot(x,pre(x),color='r',label='lsm')
158     plt.legend()
159     #plt.show()
160     plt.savefig("g"+str(size)+".png")
161 def plot_Gc(size,lamda,a):
162     # 生成数据集
163     x,y,x_train,y_train,X=data_produce(size,a)
164     # 共轭梯度法
165     w,pre=Conjugate_gradient(X,y_train,lamda,1e-6)
166     # 打印方差均根
167     print(ERMS(loss(X,y_train,w,lamda),size))
168     print(pre)
169     plt.xlabel("x")
170     plt.ylabel("y")
171     plt.xlim(0,1)
172     plt.ylim(-1.2,1.2)
173     plt.rcParams['font.sans-serif'] = ['SimHei']
174     plt.rcParams['axes.unicode_minus'] = False
175     plt.title("lsm,size="+str(size)+". 多项式阶数为: "+str(a))
176     x = np.linspace(0, 1, 100)
177     y = np.sin(x * 2 * np.pi)
178     plt.plot(x,y,color='b',label='true')
179     plt.scatter(x_train,y_train,color='g',label='train')
180     plt.plot(x,pre(x),color='r',label='gc')
181     plt.legend()
182     #plt.show()
183     plt.savefig("gc"+str(size)+".png")
```

main1.py

```python
import math

import exp1
import numpy as np
import matplotlib.pyplot as plt
# 不带正则项的最小二乘法的解析解
# 样本容量为 10
# exp1.plot_lsm(size=10,lamda=0,a=1)# 一次函数
#exp1.plot_lsm(size=10,lamda=0,a=3)# 三次函数
#exp1.plot_lsm(size=10,lamda=0,a=5)# 五次函数
#exp1.plot_lsm(size=10,lamda=0,a=7)# 七次函数
#exp1.plot_lsm(size=10,lamda=0,a=9)# 九次函数
#exp1.plot_lsm(size=10,lamda=0,a=19)# 十九次函数
# 样本容量为 30
#exp1.plot_lsm(size=30,lamda=0,a=9)# 九次函数
# 样本容量为 50
#exp1.plot_lsm(size=50,lamda=0,a=9)# 九次函数
# 样本容量为 100
#exp1.plot_lsm(size=100,lamda=0,a=9)# 九次函数
# 带正则项的最小二乘法的解析解
#exp1.plot_lsm_lamda(10)
# 梯度下降法不同阶数
#exp1.plot_Gu(size=10,lamda=math.exp(-12),a=1)# 一次函数
#exp1.plot_Gu(size=10,lamda=math.exp(-12),a=3)# 三次函数
#exp1.plot_Gu(size=10,lamda=math.exp(-12),a=5)# 五次函数
#exp1.plot_Gu(size=10,lamda=math.exp(-12),a=7)# 七次函数
#exp1.plot_Gu(size=10,lamda=math.exp(-12),a=9)# 九次函数
#exp1.plot_Gu(size=10,lamda=math.exp(-12),a=19)# 十九次函数
```

```
# 样本容量为 30
#exp1.plot_Gu(size=30,lamda=math.exp(-12),a=9)# 九次函数
# 样本容量为 50
#exp1.plot_Gu(size=50,lamda=math.exp(-12),a=9)# 九次函数
# 样本容量为 100
#exp1.plot_Gu(size=100,lamda=math.exp(-12),a=9)# 九次函数
# 共轭梯度法
#exp1.plot_Gc(size=10,lamda=math.exp(-10),a=1)# 一次函数
#exp1.plot_Gc(size=10,lamda=math.exp(-10),a=3)# 三次函数
#exp1.plot_Gc(size=10,lamda=math.exp(-10),a=5)# 五次函数
#exp1.plot_Gc(size=10,lamda=math.exp(-10),a=7)# 七次函数
#exp1.plot_Gc(size=10,lamda=math.exp(-10),a=9)# 九次函数
#exp1.plot_Gc(size=10,lamda=math.exp(-10),a=19)# 十九次函数
# 样本容量为 30
#exp1.plot_Gc(size=30,lamda=math.exp(-10),a=9)# 九次函数
# 样本容量为 50
#exp1.plot_Gc(size=50,lamda=math.exp(-10),a=9)# 九次函数
# 样本容量为 100
#exp1.plot_Gc(size=100,lamda=math.exp(-10),a=9)# 九次函数
```