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： 多项式拟合曲线

学号：1190202401
姓名：陈豪

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、克服过拟合的方法(如加惩罚项、增加样本)

二、实验要求及实验环境

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。

实验环境：

操作系统：Microsoft Windows 10 Home China 10.0.19042 N/A Build 19042
Python 版本：Python 3.9.2
编辑器：vscode

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 数据生成与划分算法：

利用 numpy 生成等间隔的数据，然后再生成正态分布的噪声，再使用 numpy 的正弦函数生成目标数据并与噪声相加。

```
def generate_data(begin,end,size,function,mu=0,sigma=0.05):
    """
    函数功能：产生数据并使用pandas的dataframe数据结构存储
    begin：样本中自变量起始下界
    end：样本中自变量终止上界
    size：样本数量的大小
    function:产生数据所使用的函数
    testratio:测试样本的比例
    mu：正态分布的均值
    sigma：正太分布的方差
    """
    x = np.linspace(begin,end,size)
    guass_noise = np.random.normal(mu,sigma,size)
    y = function(x)+guass_noise
    #np.dstack(x,y)得到的结果的shape是(1,320,2)的，所以要取第一项
    data = pd.DataFrame(np.dstack((x,y))[0],columns=['x','y'])
    return data
```

图表 1 生成数据

划分数据采用留出法：

```
def divide_data(data,testratio=0.2):
    """
    函数功能：使用留出法将数据划分为训练数据和测试数据
    data：需要是dataframe类型的
    testratio:测试数据/总数据数
    """
    test_data = data.sample(frac=testratio,axis=0)
    train_data = pd.concat([data, test_data, test_data]).drop_duplicates(keep=False)
    return train_data,test_data.sort_index()
```

图表 2 划分数据

2. 最小二乘法的解析解拟合曲线(不带惩罚项)

根据泰勒展开式和维尔斯特拉斯定理可知能够使用多项式逼近正弦函数。

已知训练集有 N 个样本点 $(x_1, y_1), (x_2, y_2), \dots (x_N, y_N)$ 其中 $x_i = (1, x_i^1, x_i^2, x_i^3, \dots, x_i^d)$，$x_i$ 是一个实数值向量，x_i^j 是指 x_i 的第 j 维的值，本次实验中认为 x_i^j 是 x_i^1 这个实数值的 j 次方

它的 d 阶多项式函数可以写成

$$y(w, x_i) = \sum_{j=0}^{d} w_j x_i^j$$

这个多项式函数对应的代价函数是：

$E(w)=\frac{1}{2}\sum_{i=1}^{N}\{y(w,x_i)-t_i\}^2, w=(1,w^1,\dots,w^d), w^j$ *是w的第j维的值。*

上式可写成如下的矩阵形式：

$$E(w)=\frac{1}{2}(Xw-T)^T(Xw-T)=\frac{1}{2}(w^TX^TXw-2w^TX^TT+T^TT)$$

其中$X=\begin{pmatrix}1 & \cdots & x_1^d\\ \vdots & \ddots & \vdots\\ 1 & \cdots & x_N^d\end{pmatrix}$ ，因此令$\frac{\partial E(w)}{\partial w}=X^TXw-X^TT=0$

可得 $w^*=(X^TX)^{-1}X^TT$

3. 最小二乘法的解析解拟合曲线（带惩罚项）

上面的求解过程中，没有加惩罚项，因此随着阶数 d 的增大，解 w 将过于复杂，导致过拟合，因此选择增加惩罚项，控制 w 的复杂度。

因此新得到的损失函数如下：

$$E(w,\lambda)=\frac{1}{2}(Xw-T)^T(Xw-T)+\frac{1}{2}\lambda||w||^2$$

对损失函数求导可以得到

$$\frac{\partial E(w)}{\partial w}=X^TXw-X^TT+\lambda w$$

令导数为 0 可得

$$w^*=(X^TX+\lambda I)^{-1}X^TT$$

4. 梯度下降拟合曲线

对于上面的损失函数：

$$E(w)=\frac{1}{2}(Xw-T)^T(Xw-T)+\frac{1}{2}\lambda||w||^2$$

除了通过求导得到极小值，还可以通过梯度下降的方法求极小值。具体的方法如下：

因为损失函数$E(w)$在任意一点 x 上可微且有定义，而根据微积分知识可知顺着梯度$\nabla E(w)$是增长最快的方向，因此梯度的反方向$-\nabla E(w)$是下降最快的方向，因此对于$\alpha>0$有下面的式子成立：

$$w_{i+1}=w_i-\alpha\nabla E(w)$$

因此对于序列$w_0,w_1,\dots$ 有$f(w_0)\geq f(w_1)\geq\cdots$

上述中α是一个超参数，代表的是学习率的值。

对于梯度下降方法，如果α过大将可能使得w离收敛越来越远，如果如果α过小将可能使得w收敛的速度过慢。因此选择合适的α很重要。

5. 共轭梯度下降拟合曲线

对于代价函数：

$$E(w)=\frac{1}{2}(w^TX^TXw-2w^TX^TT+T^TT)+\frac{1}{2}\lambda||w||^2$$

可以将其写为下面的形式

$$E(w)=\frac{1}{2}w^TAw-w^Tb+c$$

上式中：

$$A=X^TX+\lambda I$$

$$b=X^TT$$

因为 A 是内积得到的，因此是对称正定矩阵。所以上述求解极小值的问题可以等价的求解$Aw=b$，对于这一方程组可以使用共轭梯度的方式求解。
具体的求解算法如下：

输入：

给定对称矩阵 A

给定右端项 b

给定迭代初值$w_0 \in R^n$

输出：

方程的解w

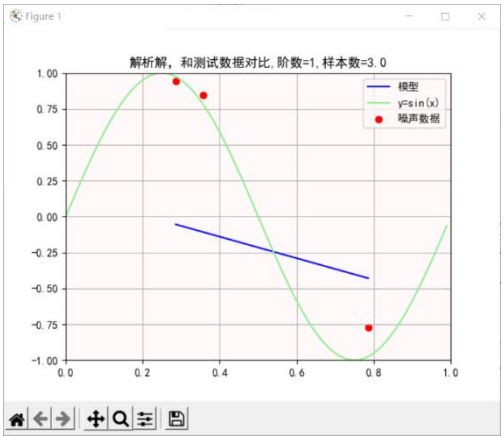
1. 计算残量$r_0=b-A$
2. 初始化$p_0=r_0$
3. Repeat
4. For k = 0,1,⋯ do
5. If $p_k=0$ then
6. Return w_k
7. End
8. Else
9. $a_k=\frac{r_k^Tr_k}{p_k^TAp_k}$
10. $w_{k+1}=w_k+a_kp_k$
11. $r_{k+1}=r_k-a_kAp_k$
12. $b_k=\frac{r_{k+1}^Tr_{k+1}}{r_k^Tr_k}$
13. $p_{k+1}=r_{k+1}+b_kp_k$
14. End
15. End
16. Until convergence

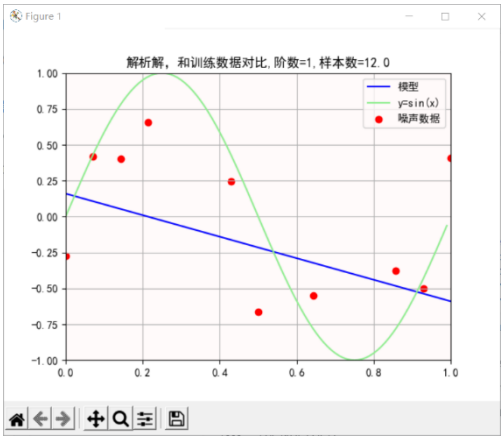
四、实验结果与分析

1. 不带惩罚项的解析解

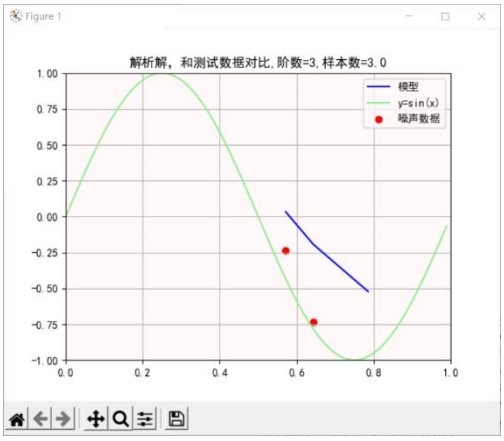
固定总样本数 15，训练样本数量为 12，测试样本数 3，噪声符合均值为 0，标准差为 0.3 的正态分布，改变阶数得到如下图

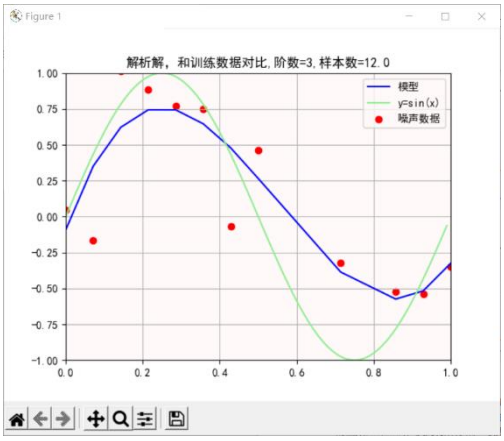
阶数为 1 时：



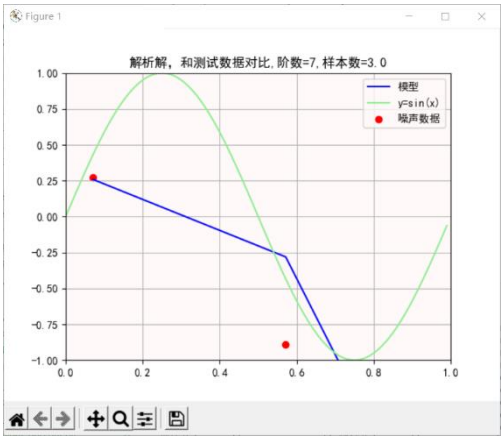


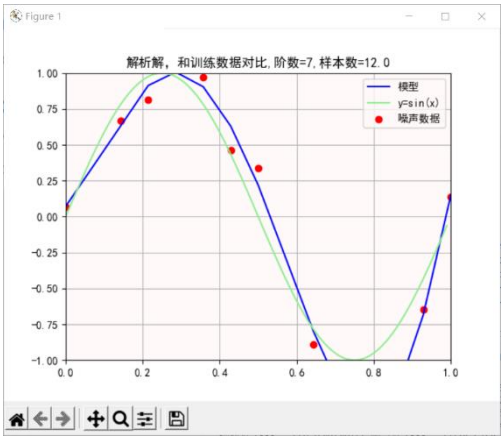
阶数为 3 时：



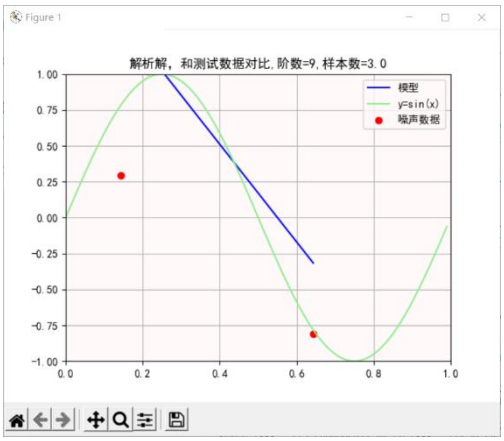


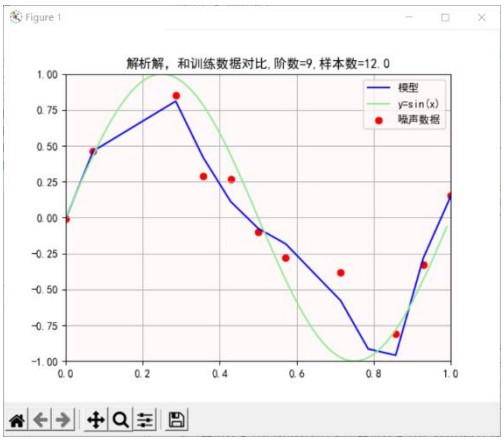
阶数为 7 时：



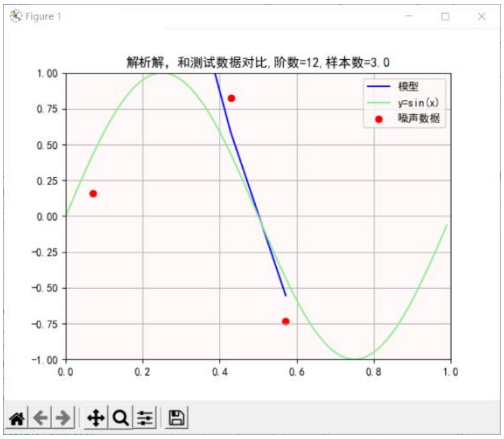


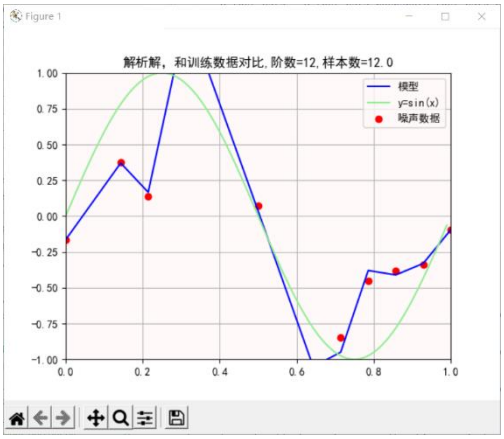
阶数为 9 时：





阶数为 12 时：





然后我运行并计算了随着阶数的提升，训练集和测试集上的损失，如下图所示：

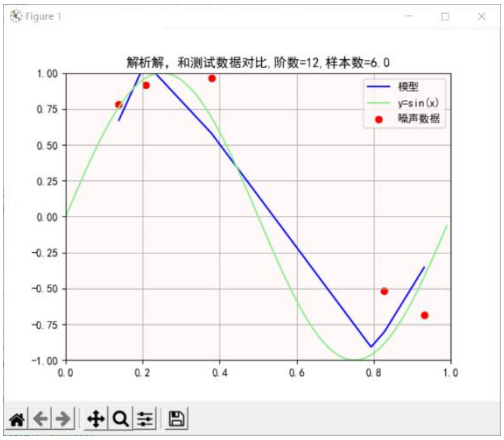
```
训练集loss = [[3.87818823]],测试集loss = [[1.64271215]],order = 0
训练集loss = [[2.22007117]],测试集loss = [[1.03590992]],order = 1
训练集loss = [[2.20360148]],测试集loss = [[1.07142932]],order = 2
训练集loss = [[0.24680459]],测试集loss = [[0.21316408]],order = 3
训练集loss = [[0.07319719]],测试集loss = [[0.22197728]],order = 4
训练集loss = [[0.0455103]],测试集loss = [[0.17256569]],order = 5
训练集loss = [[0.04454787]],测试集loss = [[0.1809172]],order = 6
训练集loss = [[0.04311656]],测试集loss = [[0.18871163]],order = 7
训练集loss = [[0.03885754]],测试集loss = [[0.19722879]],order = 8
训练集loss = [[0.03830421]],测试集loss = [[0.27216816]],order = 9
训练集loss = [[0.03896423]],测试集loss = [[0.20968919]],order = 10
训练集loss = [[0.03928376]],测试集loss = [[0.18108915]],order = 11
训练集loss = [[0.00997134]],测试集loss = [[3.47037426]],order = 12
训练集loss = [[0.00856588]],测试集loss = [[2.77306704]],order = 13
训练集loss = [[0.00736419]],测试集loss = [[2.3469944]],order = 14
训练集loss = [[0.00634232]],测试集loss = [[2.09250105]],order = 15
训练集loss = [[0.00547793]],测试集loss = [[1.95919574]],order = 16
训练集loss = [[0.00474921]],测试集loss = [[1.92142379]],order = 17
训练集loss = [[0.00413601]],测试集loss = [[1.96554173]],order = 18
训练集loss = [[0.00362021]],测试集loss = [[2.08572436]],order = 19
```

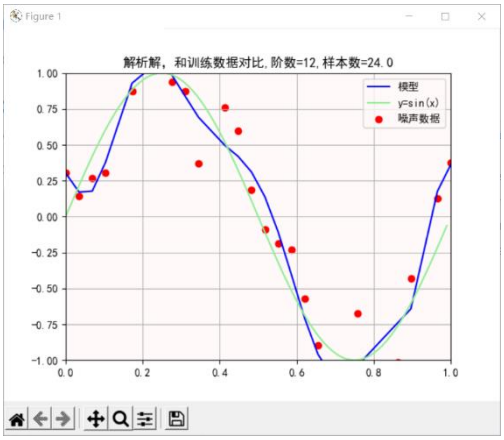
图表 3 固定样本数时的 loss

从上图可以看出，在固定样本数量的前提下，提升阶数，能够使得模型在训练集上表现的越来越好，当阶数只为一时，loss 较大，因为此时模型复杂度不够，在训练样本和测试样本上都表现不好，处在欠拟合阶段。而在 12 阶的时候，可以看到模型穿过了训练样本上的每一点。但是从 loss 上可以看出，这个时候在测试集上的 loss 反而没有下降，而是快速上升。这就是过拟合现象，也就是说模型的阶数提高，它的自由度太高，学习了过多的训练样本的特征，导致泛化能力降低，在测试集上表现差。

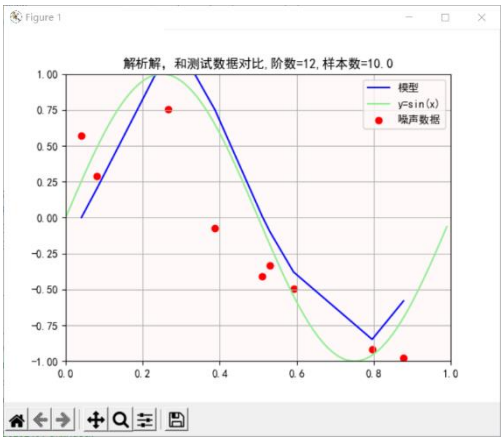
解决过拟合的方法除了加入正则项外还可以通过添加数据样本的方式来改变，因此接下来我通过增加训练样本数来观察 loss 的损失。

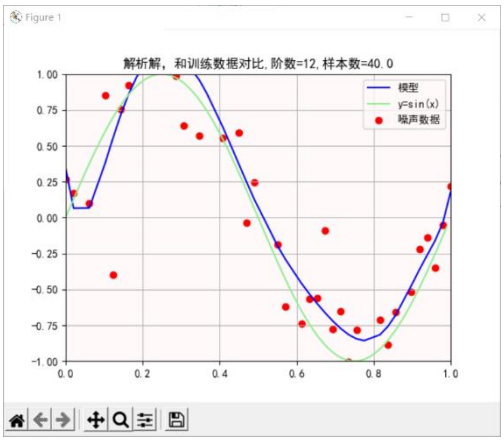
训练样本数=24 时



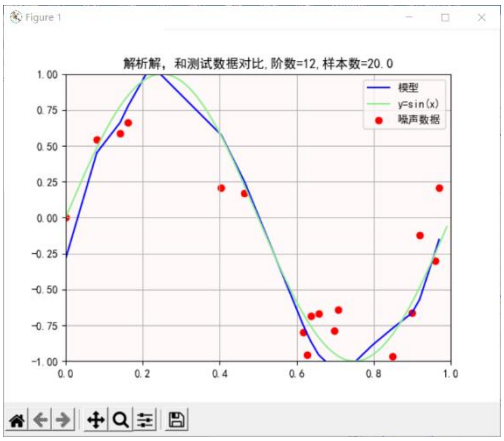


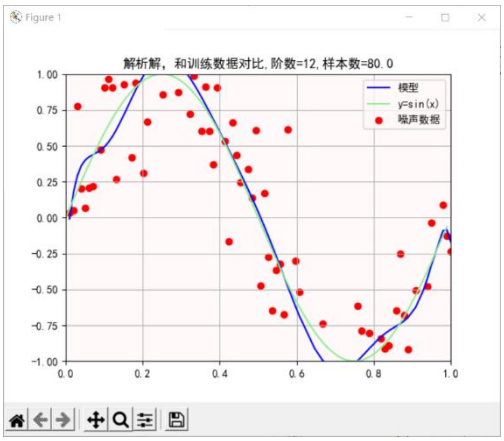
训练样本数=40 时





训练样本数=80 时





通过增加样本数我们可以看出过拟合现象得到了显著的缓解，同时随着样本数增加上述的 loss 值变化也相对不大，说明拟合的很好。

2. 带惩罚项的解析解

除了通过增加样本数量来避免过拟合，还可以通过为损失函数添加正则项的方式避免过拟合。

$$\lambda = argmin_{\lambda} E(w, \lambda)$$

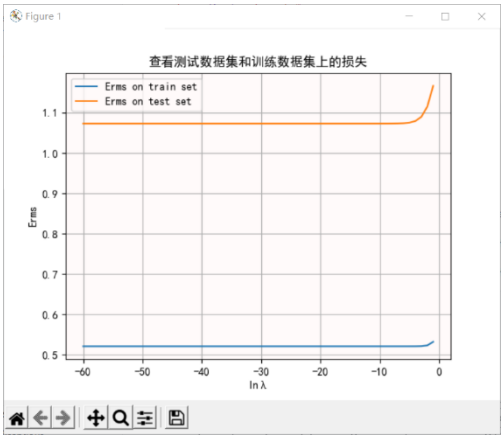
而为了达到更好的效果，因此这里选择计算均方误差来查找效果最好的惩罚项系数

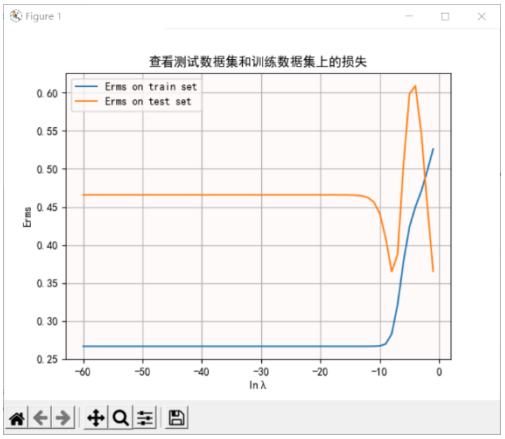
$$E_{RMS} = \sqrt{\frac{2E(w^*, \lambda)}{N}}$$

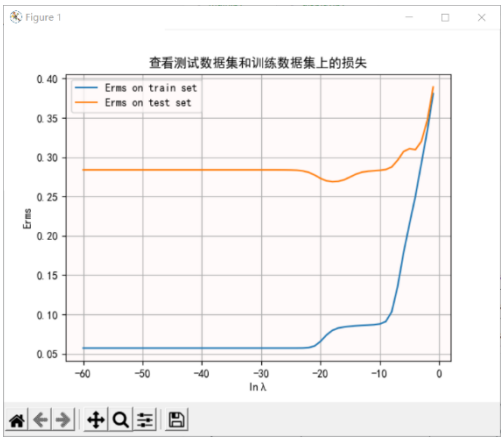
因此可以得到

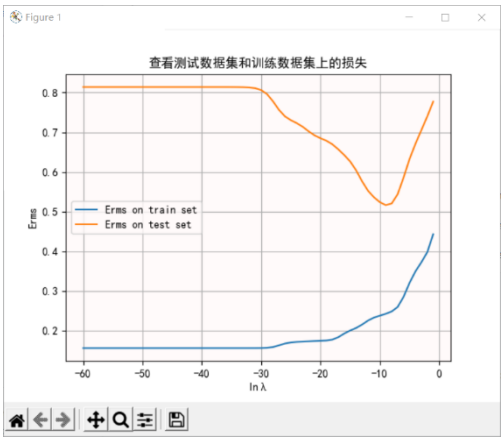
$$\lambda = argmin_{\lambda} E_{RMS}$$

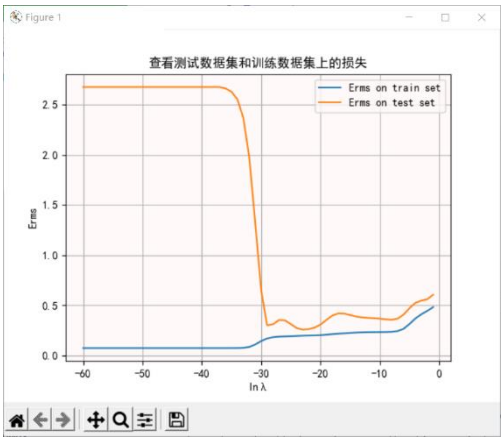
检验在训练样本数为 12，阶数=1，3，7，9，12 时训练集和测试集上的图像：





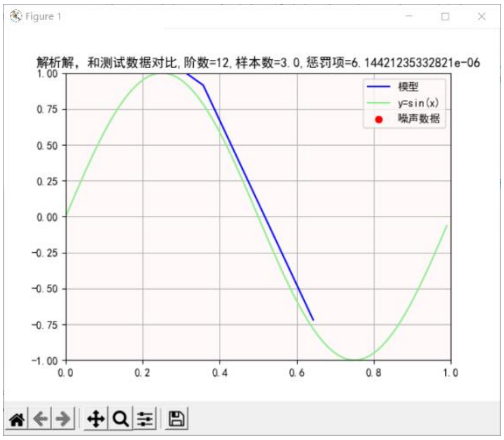


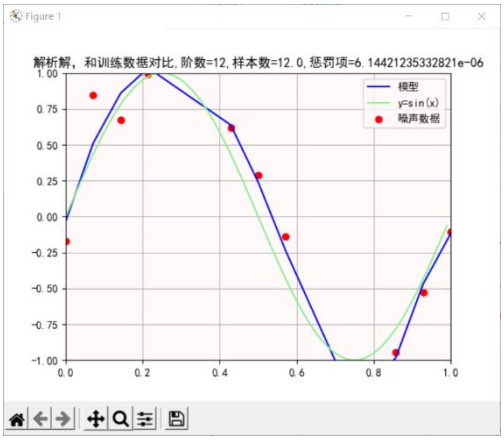




从上图可以看出在λ比较小的时候测试集上的 Erms 基本不动，而当$\lambda\epsilon(e^{-20},e^{-10})$的时候下降明显，而当$\lambda$继续增大后 Erms 增大。因此选择$\lambda$在$(e^{-20},e^{-10})$的范围的时候惩罚项效果比较好。因此在接下来实验中选取$\lambda = e^{-12}$作为惩罚系数。

下面演示在$\lambda = e^{-12}$时的解析解图像。

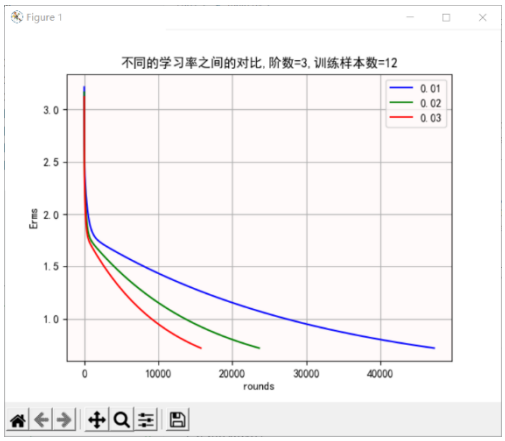


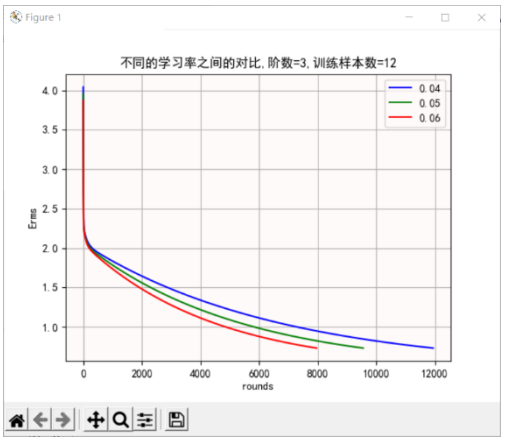


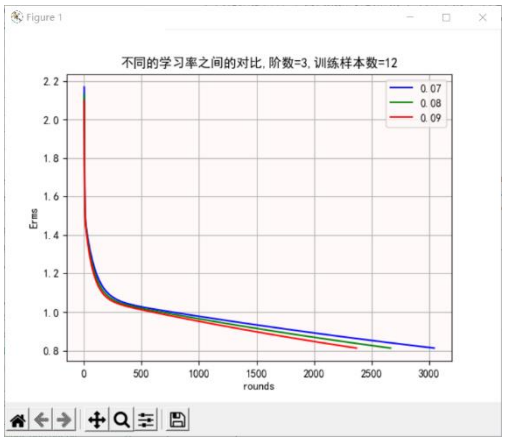
可以看出这时避免过拟合的效果比较明显，得到的模型与 y=sin(x)的效果比较吻合，得到了较好的模型。

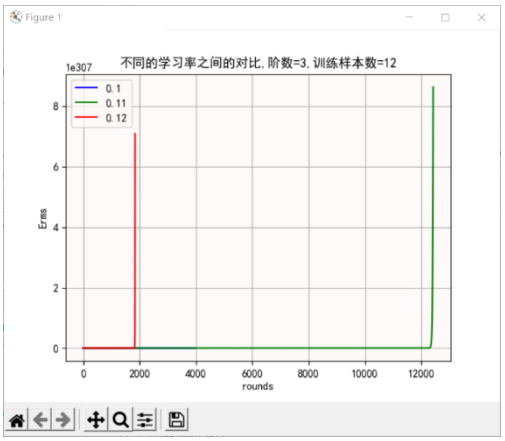
3. 梯度下降法的结果

梯度下降方法是求优化解的一种迭代方法，它的收敛速度和正确性取决于学习率α的取值，因此对于训练样本为 12，阶数为 3 的情况下对学习率在[0.01,1.12]时训练的情况如下所示



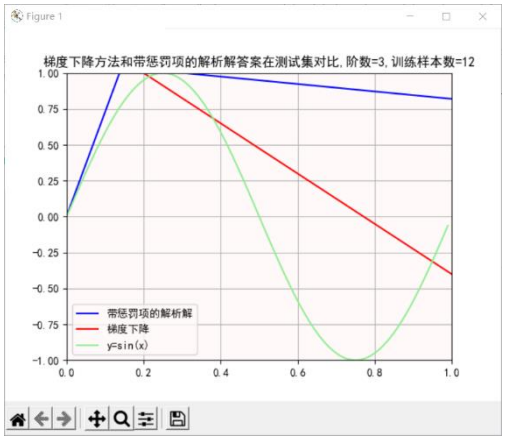


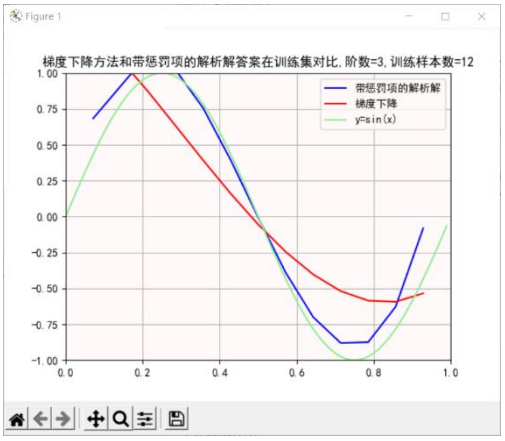




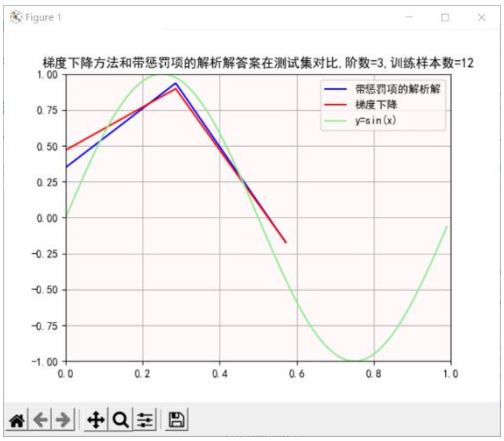
从上图中可以看出当学习率过低时，收敛速度很慢，需要多达 4w 轮才能够得到收敛结果，而当学习率适当增大后，比如 0.07，0.08，0.09 的时候仅仅只需要两三千轮即可收敛得到结果，而当学习率过高的时候将无法收敛，典型的就比如图中学习率大于 0.1 时候的场景。

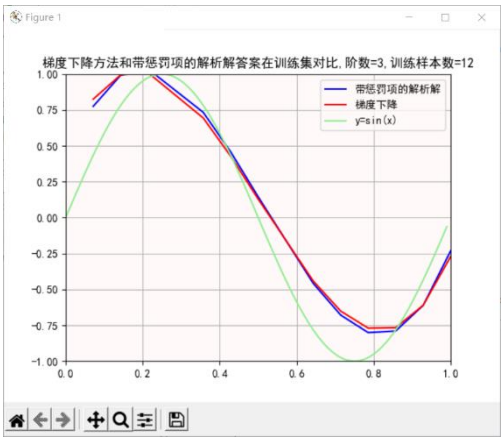
接下来展示梯度下降方法和带惩罚项的解析解的在阶数=3，训练样本数=12 时的对比





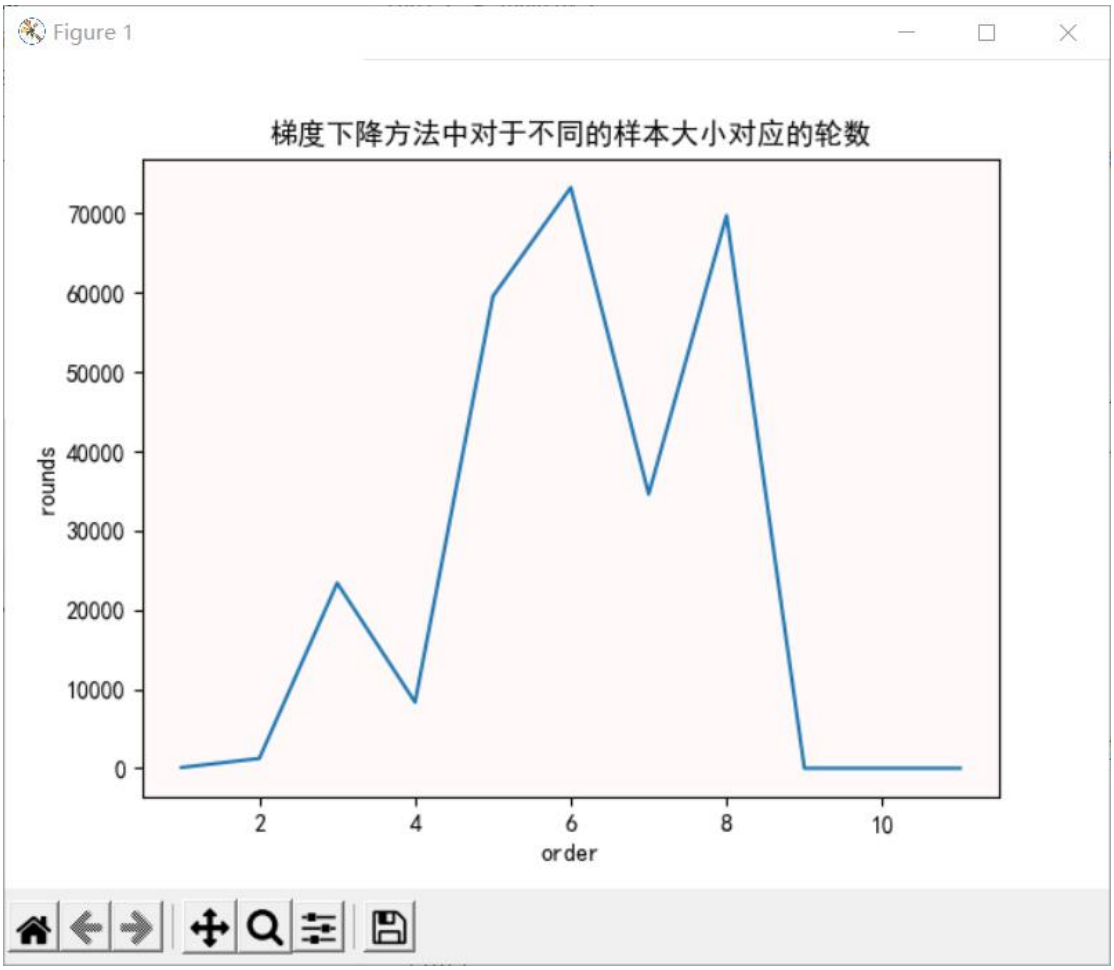
两者存在一些差别，这可能是梯度下降的收敛判别值 delta=1e-3 的选择过大导致的。因此下面对 delta 重新取值为 1e-5。再次训练得到的结果如下。





可以看出这一次两者基本上完美的对齐，仅有部分差别。因此可以看出 delta 的取值也是很重要的一点。

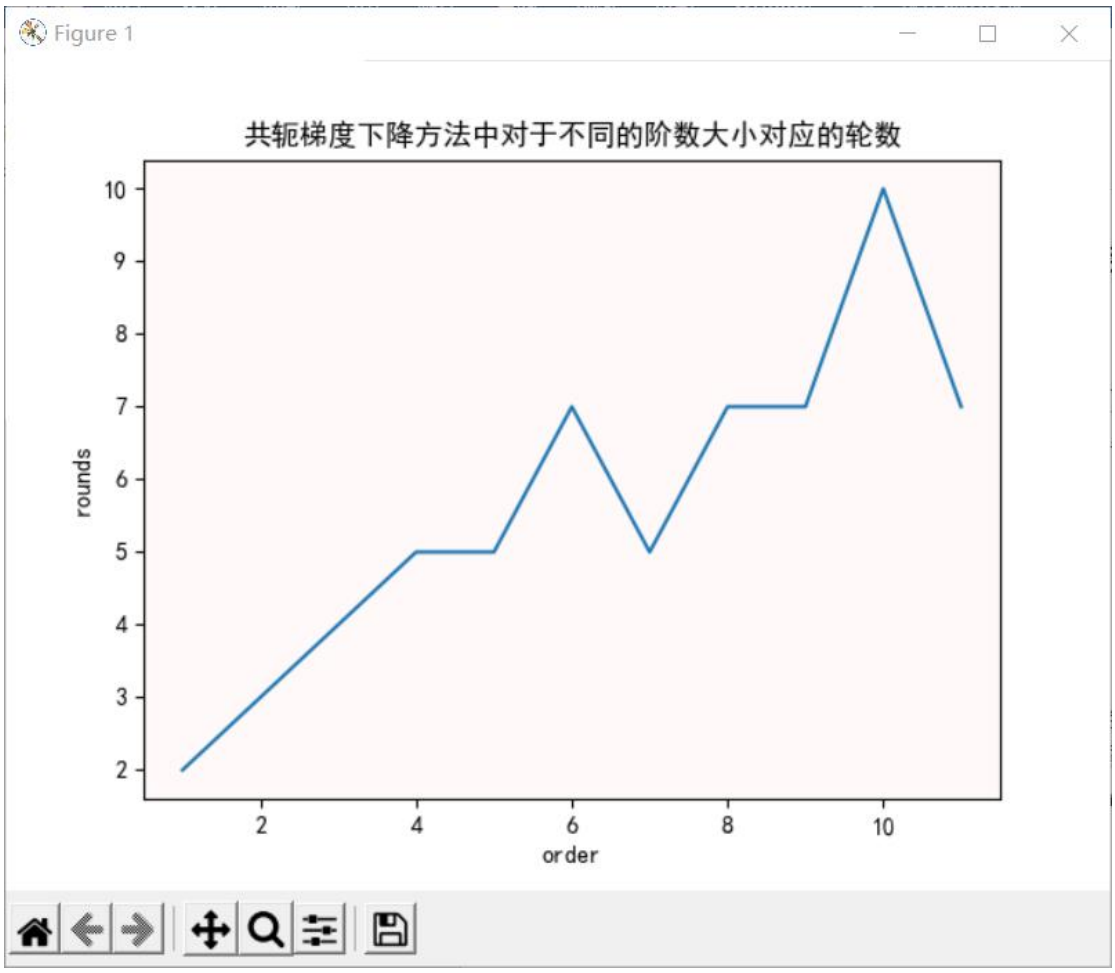
下面固定样本数量，对于不同的 order 进行训练查看得到的轮数。



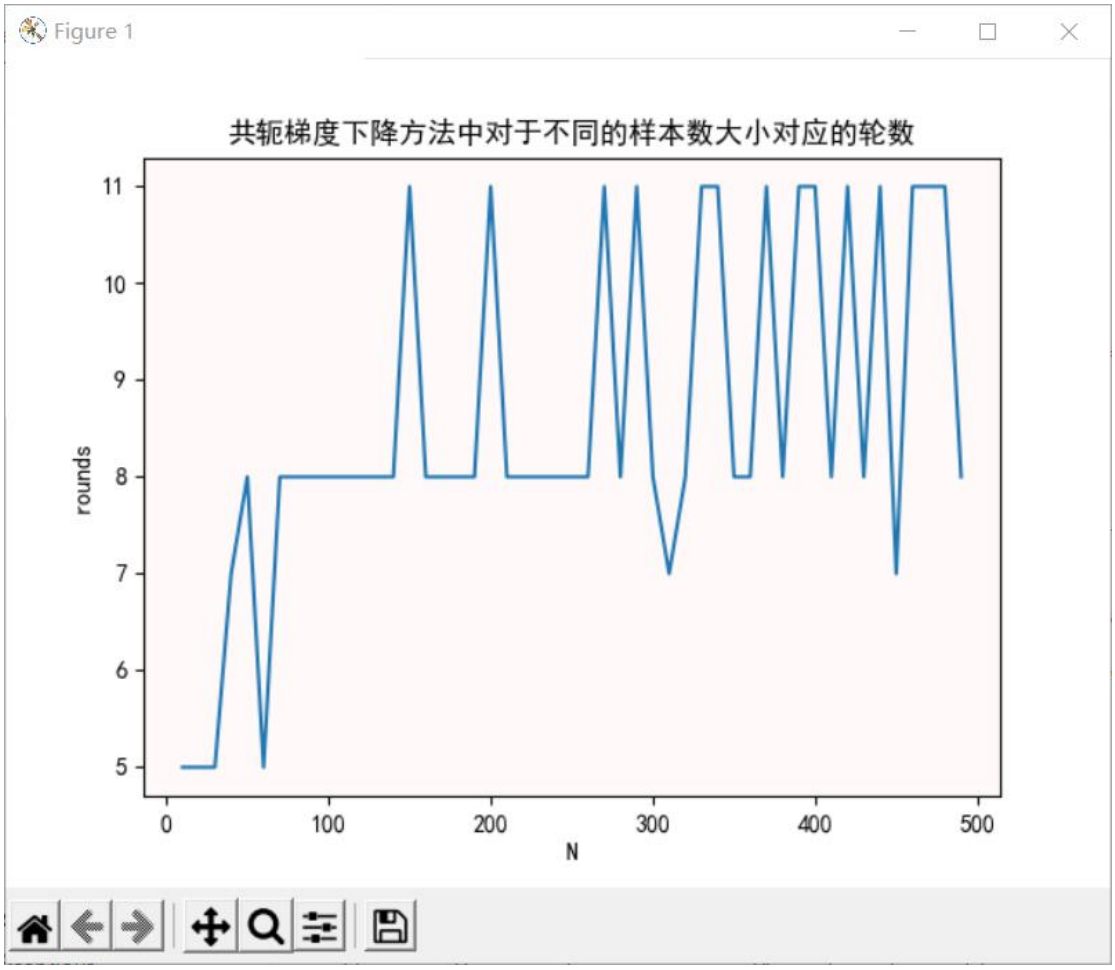
可以看出随着阶数的提升，大致上运行的轮数波动比较大，总体上呈现上升趋势。（注:高阶的时候变为 0 是因为我设定了如果算出来的 w 达到了 nan，时轮数作为 0）同时可以发现随着阶数的升高，很容易就出现不收敛的情况，此时学习率需要降低。

4. 共轭梯度下降法的结果

共轭梯度法相比梯度下降法在速度上面有了很大的优势，下面查看一下在训练样本=12 时，不同阶数下，共轭梯度算法的轮数。



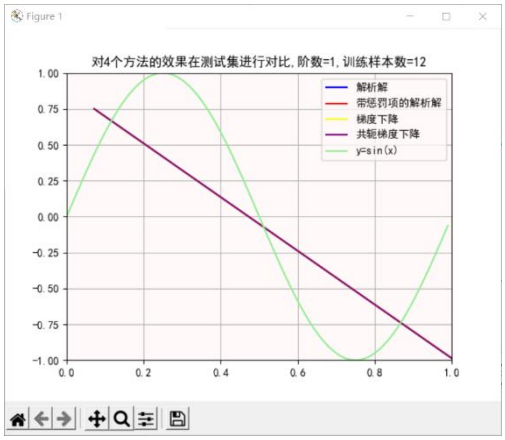
可以看出阶数上升，所需要的轮数也上升了，但是共轭梯度下降的轮数实际上很小，只需要几轮就可以得到答案了。在来看看对于不同大小的样本数量下共轭梯度的表现。(这里选定的 order=7)

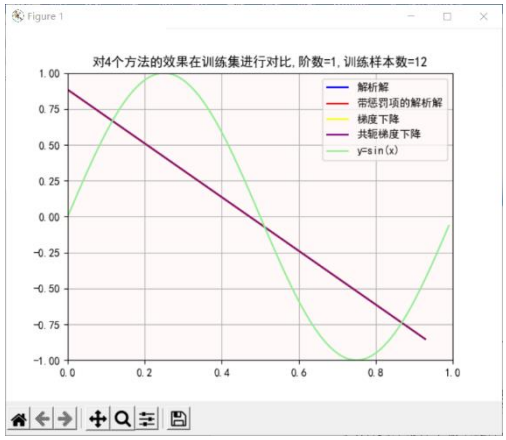


可以看到即使样本数量提升，运行的轮数依然比较小。所以共轭梯度的速度确实很快。而对于共轭梯度的效果，将在下面这一小节中展现。

5. 多个方法对比

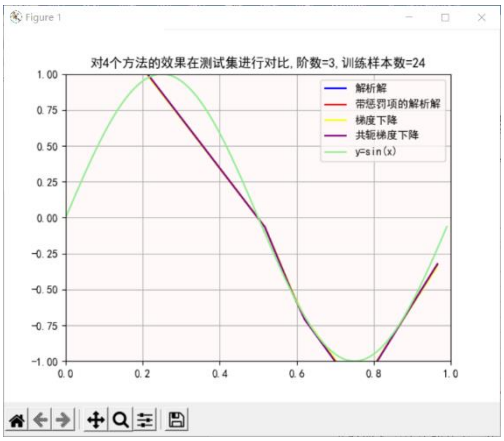
在训练样本=12，阶数=1 时

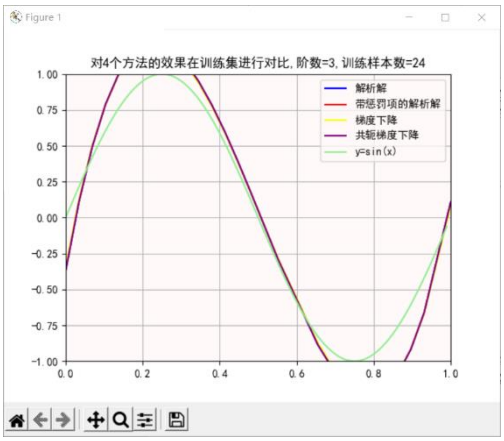




此时四个方法的解基本一致，只有当放大看的时候才能看出细微的差别。

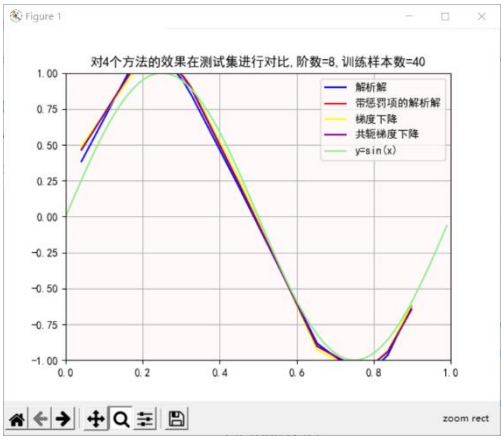
在训练样本=24，阶数=3 时

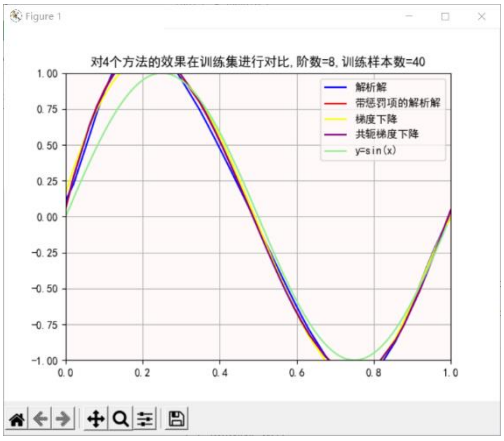




此时四个方法的解基本一致，只有当放大看的时候才能看出细微的差别。而且放大后容易发现，带惩罚项的解析解和共轭梯度法的解是最相近的。

当训练样本=40，阶数=8 时





此时四个方法的解依然基本一致，只有当放大看的时候才能看出细微的差别。而且放大后容易发现，带惩罚项的解析解和共轭梯度法的解是依然最相近的。

五、结论

使用解析解求解时，如果阶数过大，将会导致过拟合问题。

对于过拟合的问题，可以通过增加样本数量或者增加正则项的方式解决。

对于梯度下降，它的学习率是比较难选择的，对于不同的训练样本数量，模型阶数，它的学习率都要求不同。

对于共轭梯度和带惩罚项的解析解，这两种方法得到的结果基本一致，而梯度下降与他们之所以存在差别的原因，很多时候是因为梯度下降的收敛条件 delta 的判断，只有当 delta 设置的足够小的时候，梯度下降才能够收敛到和他们一致，但梯度下降的速度比较满，所以如果要收敛到足够小需要的时间也会增加很多。

对于共轭梯度算法，采用不同的样本数量和阶数大小对它的运行时间来说影响不大，它往往只需要几轮时间即可求得不错的解，是一个高效的算法。

六、参考文献

数值分析 2020 春苏州大学
统计学习方法（李航）
机器学习（西瓜书）

七、附录：源代码（带注释）

main.py

```
from conjugat_gradient import ConjugatGradient
from gradient_descent import *
from get_data import *
from analytical_solution import *
from display import *

if __name__ == '__main__':

    begin = 0
    end = 1
    size = 50
    order = 8
    testratio = 0.2
    penalty_lambda = np.exp(-12)
    alpha = 0.02
    xlimit=(begin,end)
    ylimit=(-1,1)

    x_train_data,y_train_data,x_test_data,y_test_data = get_data(begin,end,size,lambda x :np.sin(2*np.pi*x),testratio)
    x_matrix = get_x_matrix(x_train_data,order)
    x_test_matrix = get_x_matrix(x_test_data,order)

    # 查找比较好 lambda
    #show_Erms(x_train_data,y_train_data,x_test_data,y_test_data,order,title='查看测试数据集和训练数据集上的损失',penalty_lambda=range(-60,0))

    # #numpy 自带的多项式拟合
    # w = np.polyfit(x_train_data.reshape(x_train_data.shape[0]),y_train_data.reshape(y_train_data.shape[0]),order)
    # w = w[::-1]
    # show_compared_to_data(w,x_test_data,y_test_data,'numpy 自带的多项式拟合',lambda x:np.sin(2*np.pi*x),xlimit,ylimit,order)
    # show_compared_to_data(w,x_train_data,y_train_data,'numpy 自带的多项式拟合',lambda x:np.sin(2*np.pi*x),xlimit,ylimit,order)

    #开始训练不带惩罚项的解析解
    train = AnalyticalSolution(x_matrix,y_train_data)
```

```
34.     w0 = train.normal()
35.     # #训练结果
36.     # show_compared_to_data(w0,x_test_data,y_test_data,'解析解，和测试数据对比,阶数={0},样本数={1}'.format(order,size*testratio),lambda x :np.sin(2*np.pi*x),xlimit,ylimit,order)
37.     # print('训练集 loss = {0},测试集 loss = {1},order = {2}'.format(train.normal_loss(),train.normal_loss_test(x_test_matrix,y_test_data),order))
38.     # show_compared_to_data(w0,x_train_data,y_train_data,'解析解，和训练数据对比,阶数={0},样本数={1}'.format(order,size-size*testratio),lambda x :np.sin(2*np.pi*x),xlimit,ylimit,order)
39.
40.
41.     #开始训练带惩罚项的解析解
42.     train = AnalyticalSolution(x_matrix,y_train_data)
43.     w1 = train.with_penalty(penalty_lambda)
44.     # #训练结果
45.     # show_compared_to_data(w1,x_test_data,y_test_data,'解析解，和测试数据对比,阶数={0},样本数={1},惩罚项={2}'.format(order,size*testratio,penalty_lambda),lambda x :np.sin(2*np.pi*x),xlimit,ylimit,order)
46.     # print(train.with_penalty_loss())
47.     # show_compared_to_data(w1,x_train_data,y_train_data,'解析解，和训练数据对比,阶数={0},样本数={1},惩罚项={2}'.format(order,size-size*testratio,penalty_lambda),lambda x :np.sin(2*np.pi*x),xlimit,ylimit,order)
48.
49.     #开始梯度下降训练
50.     model = GradientDescent(x_matrix,y_train_data,penalty_lambda,alpha)
51.     w2 = model.train()
52.     # print('梯度下降运行了{0}轮'.format(len(w2[0])))
53.     # show_compared_to_data(w2[1],x_test_data,y_test_data,'梯度下降，和测试数据对比,阶数={0},样本数={1},惩罚项={2},学习率={3}'.format(order,size*testratio,penalty_lambda,alpha),lambda x :np.sin(2*np.pi*x),xlimit,ylimit,order)
54.     # show_compared_to_data(w2[1],x_train_data,y_train_data,'梯度下降，和训练数据对比,阶数={0},样本数={1},惩罚项={2},学习率={3}'.format(order,size-size*testratio,penalty_lambda,alpha),lambda x :np.sin(2*np.pi*x),xlimit,ylimit,order)
55.
56.     #开始共轭梯度下降训练
```

```
57.     model = ConjugatGradient(x_matrix,y_train_data,penalty_lambda
    ,1e-6)
58.     w_0 = np.zeros(x_matrix.shape[1]).reshape(x_matrix.shape[1],1
    )
59.     w3 = model.train(w_0)
60.     # print('共轭梯度下降运行了{0}轮'.format(w3[0]))
61.     # #训练结果
62.     # show_compared_to_data(w3[1],x_test_data,y_test_data,'共轭梯
    度下降，和测试数据对比,阶数={0},样本数={1},惩罚项
    ={2}'.format(order,size*testratio,penalty_lambda),lambda x :np.si
    n(2*np.pi*x),xlimit,ylimit,order)
63.     # show_compared_to_data(w3[1],x_train_data,y_train_data,'共轭
    梯度下降，和训练数据对比,阶数={0},样本数={1},惩罚项
    ={2}'.format(order,size-
    size*testratio,penalty_lambda),lambda x :np.sin(2*np.pi*x),xlimit
    ,ylimit,order)
64.
65.
66.     # #尝试多阶数不带惩罚项的解析解
67.     # for order in range(20):
68.     #     #开始训练不带惩罚项的解析解
69.     #     x_matrix = get_x_matrix(x_train_data,order)
70.     #     x_test_matrix = get_x_matrix(x_test_data,order)
71.     #     train = AnalyticalSolution(x_matrix,y_train_data)
72.     #     w = train.normal()
73.     #     #训练结果
74.     #     # show_compared_to_data(w,x_test_data,y_test_data,'解析
    解，和测试数据对比,阶数={0},样本数
    ={1}'.format(order,size*testratio),lambda x :np.sin(2*np.pi*x),xl
    imit,ylimit,order)
75.     #     print('训练集 loss = {0},测试集
    loss = {1},order = {2}'.format(train.normal_loss(),train.normal_l
    oss_test(x_test_matrix,y_test_data),order))
76.     #     # show_compared_to_data(w,x_train_data,y_train_data,'解
    析解，和训练数据对比,阶数={0},样本数={1}'.format(order,size-
    size*testratio),lambda x :np.sin(2*np.pi*x),xlimit,ylimit,order)
77.
78.     # #尝试不同的学习率的梯度下降
79.     # show_alpha(x_train_data,y_train_data,alpha=[0.1,0.11,0.12],
    order=order,title='不同的学习率之间的对比,阶数={0},训练样本数
    ={1}'.format(order,x_train_data.size),penalty_lambda=penalty_lamb
    da,labels=['0.1','0.11','0.12'],colors=['blue','green','red'])
80.
81.     # #梯度下降方法和带惩罚项的解析解答案对比
```

```
    # show_comparasion(w=[w1,w2[1]],x=x_train_data,y=y_train_data,title='梯度下降方法和带惩罚项的解析解答案在训练集对比,阶数={0},训练样本数={1}'.format(order,x_train_data.size),colors=['blue','red'],labels=['带惩罚项的解析解','梯度下降'],function = lambda x :np.sin(2*np.pi*x),xlimit=(0,1),ylimit=(-1,1),order=order)
    # show_comparasion(w=[w1,w2[1]],x=x_test_data,y=y_test_data,title='梯度下降方法和带惩罚项的解析解答案在测试集对比,阶数={0},训练样本数={1}'.format(order,x_train_data.size),colors=['blue','red'],labels=['带惩罚项的解析解','梯度下降'],function = lambda x :np.sin(2*np.pi*x),xlimit=(0,1),ylimit=(-1,1),order=order)

    # #梯度下降方法中对于不同的阶数大小对应的轮数
    # N = range(1,12)
    # rounds = []
    # for n in N:
    #     x_train_data,y_train_data,x_test_data,y_test_data = get_data(begin,end,size,lambda x :np.sin(2*np.pi*x),testratio)
    #     x_matrix = get_x_matrix(x_train_data,n)
    #     model = GradientDescent(x_matrix,y_train_data,penalty_lambda,alpha)
    #     w2 = model.train()
    #     if np.isnan(w2[1]).any() :
    #         rounds.append(0)
    #     else:
    #         rounds.append(len(w2[0]))
    # plt.plot(N,rounds)
    # plt.xlabel('order')
    # plt.ylabel('rounds')
    # plt.title('梯度下降方法中对于不同的阶数大小对应的轮数')
    # plt.show()

    # #共轭梯度下降方法中对于不同的阶数大小对应的轮数
    # N = range(1,12)
    # rounds = []
    # for n in N:
    #     x_train_data,y_train_data,x_test_data,y_test_data = get_data(begin,end,size,lambda x :np.sin(2*np.pi*x),testratio)
    #     x_matrix = get_x_matrix(x_train_data,n)
    #     model = ConjugatGradient(x_matrix,y_train_data,penalty_lambda,1e-6)
```

```
    #     w_0 = np.zeros(x_matrix.shape[1]).reshape(x_matrix.shape[1],1)
    #     w3 = model.train(w_0)
    #     if np.isnan(w3[1]).any() :
    #         rounds.append(0)
    #     else:
    #         rounds.append(w3[0])
    # plt.plot(N,rounds)
    # plt.xlabel('order')
    # plt.ylabel('rounds')
    # plt.title('共轭梯度下降方法中对于不同的阶数大小对应的轮数')
    # plt.show()

    # #共轭梯度下降方法中对于不同的样本数大小对应的轮数
    # N = range(10,500,10)
    # rounds = []
    # for n in N:
    #     x_train_data,y_train_data,x_test_data,y_test_data = get_data(begin,end,n,lambda x :np.sin(2*np.pi*x),testratio)
    #     x_matrix = get_x_matrix(x_train_data,order)
    #     model = ConjugatGradient(x_matrix,y_train_data,penalty_lambda,1e-6)
    #     w_0 = np.zeros(x_matrix.shape[1]).reshape(x_matrix.shape[1],1)
    #     w3 = model.train(w_0)
    #     if np.isnan(w3[1]).any() :
    #         rounds.append(0)
    #     else:
    #         rounds.append(w3[0])
    # plt.plot(N,rounds)
    # plt.xlabel('N')
    # plt.ylabel('rounds')
    # plt.title('共轭梯度下降方法中对于不同的样本数大小对应的轮数')
    # plt.show()

    #对4个方法的效果进行对比
    show_comparasion([w0,w1,w2[1],w3[1]],x_test_data,y_test_data,title='对4个方法的效果在测试集进行对比,阶数={0},训练样本数={1}'.format(order,x_train_data.size),function=lambda x:np.sin(2*np.pi*x),colors=['blue','red','yellow','purple'],labels=['解析解','带惩罚项的解析解','梯度下降','共轭梯度下降'],order=order)
    show_comparasion([w0,w1,w2[1],w3[1]],x_train_data,y_train_data,title='对4个方法的效果在训练集进行对比,阶数={0},训练样本数={1}'.format(order,x_train_data.size),function=lambda x:np.sin(2*
```

```
    np.pi*x),colors=['blue','red','yellow','purple'],labels=['解析解
    ','带惩罚项的解析解','梯度下降','共轭梯度下降'],order=order)
144.
145.
```

analytical_solution.py

```
1.  import numpy as np
2.
3.  class AnalyticalSolution(object):
4.      """
5.      使用最小二乘法求解解析解
6.      """
7.
8.      def __init__(self, x_matrix, y_tag):
9.          """
10.         x_matrix 的维度是 N*m，N 意味着样本数量，m 意味着多项式的阶数,X
    是可逆的
11.         y_tag 的维度是 N*1,N 意味着 N 个样本相对应的目标值
12.         """
13.         self.x_matrix = x_matrix
14.         self.y_tag = y_tag
15.
16.
17.     def normal(self):
18.         """
19.         这个函数返回不带惩罚项的解析解 w，w 的维度是 m*1
20.         """
21.         k1 = np.dot(self.x_matrix.T,self.x_matrix)
22.         k2 = np.linalg.pinv(k1)
23.         k3 = self.x_matrix.T
24.         k4 = self.y_tag
25.         self.w = np.dot(np.dot(k2,k3),k4)
26.         return self.w.reshape(self.w.shape[0],1)
27.
28.
29.     def with_penalty(self,penalty_lambda):
30.         """
31.         这个函数返回带惩罚项的解析解 w，w 的维度是 m*1
32.         penalty_lambda：惩罚系数 λ
33.         """
34.         self.penalty_lambda = penalty_lambda
35.         penalty = self.penalty_lambda*np.eye(self.x_matrix.shape[
    1])
```

```
        self.w = np.linalg.pinv((self.x_matrix.T @ self.x_matrix)+penalty) @ self.x_matrix.T @ (self.y_tag)
        return self.w.reshape(self.w.shape[0],1)

    def normal_loss(self):
        """
        这个函数返回在训练数据集上的损失函数值
        """
        return (self.x_matrix @ self.w - self.y_tag).T @ (self.x_matrix @ self.w - self.y_tag) * 0.5

    def normal_loss_test(self,x_test_matrix,y_test_tag):
        """
        这个函数返回在测试数据集上的损失函数值
        """
        return (x_test_matrix @ self.w - y_test_tag).T @ (x_test_matrix @ self.w - y_test_tag) * 0.5


    def with_penalty_loss(self):
        """
        这个函数返回带惩罚项的损失函数值
        """
        f = self.penalty_lambda * self.w.T@self.w * 0.5
        return (self.x_matrix @ self.w - self.y_tag).T @ (self.x_matrix @ self.w - self.y_tag) * 0.5 + f

    def with_penalty_loss_test(self,x_test_matrix,y_test_tag):
        """
        这个函数返回带惩罚项的在测试数据集上的损失函数值
        """
        f = self.penalty_lambda * self.w.T@self.w * 0.5
        return (x_test_matrix @ self.w - y_test_tag).T @ (x_test_matrix @ self.w - y_test_tag) * 0.5 + f

    def E_rms(self,y_true,y_pred):
        """
        这个函数计算根均方误差
        """
        return np.sqrt(np.mean(np.square(y_true - y_pred)))
```

get_data.py

```
import numpy as np
```

```
2. import pandas as pd
3.
4. def generate_data(begin,end,size,function,mu=0,sigma=0.3):
5.     """
6.     函数功能：产生数据并使用 pandas 的 dataframe 数据结构存储
7.     begin：样本中自变量起始下界
8.     end：样本中自变量终止上界
9.     size：样本数量的大小
10.    function:产生数据所使用的函数
11.    testratio:测试样本的比例
12.    mu：正态分布的均值
13.    sigma：正太分布的方差
14.    """
15.    x = np.linspace(begin,end,size)
16.    guass_noise = np.random.normal(mu,sigma,size)
17.    y = function(x)+guass_noise
18.    #np.dstack(x,y)得到的结果的 shape 是(1,320,2)的，所以要取第一项
19.    data = pd.DataFrame(np.dstack((x,y))[0],columns=['x','y'])
20.    return data
21.
22.def divide_data(data,testratio=0.2):
23.    """
24.    函数功能：使用留出法将数据划分为训练数据和测试数据
25.    data：需要是 dataframe 类型的
26.    testratio:测试数据/总数据数
27.    """
28.    test_data = data.sample(frac=testratio,axis=0)
29.    train_data = pd.concat([data, test_data, test_data]).drop_dup
   licates(keep=False)
30.    return train_data,test_data.sort_index()
31.
32.def get_data(begin,end,size,function,testratio):
33.    """
34.    函数功能：生成数据并划分，同时是得到数据格式一致
35.    """
36.    train_data,test_data = divide_data(generate_data(begin,end,si
   ze,function),testratio)
37.    x_train_data = np.array(train_data['x'].tolist())
38.    x_train_data = x_train_data.reshape(x_train_data.shape[0],1)
39.    y_train_data = np.array(train_data['y'].tolist())
40.    y_train_data = y_train_data.reshape(y_train_data.shape[0],1)
41.    x_test_data = np.array(test_data['x'].tolist())
42.    x_test_data = x_test_data.reshape(x_test_data.shape[0],1)
43.    y_test_data = np.array(test_data['y'].tolist())
```

```
    y_test_data = y_test_data.reshape(y_test_data.shape[0],1)
    return x_train_data,y_train_data,x_test_data,y_test_data

def get_x_matrix(x_vector,order=3):
    """
    函数功能：获得类似范德蒙德行列式的结构
    x_vector：自变量
    order：选择使用多少维度的模型，也就是多项式的最高指数
    """
    x_matrix = []
    for i in range(len(x_vector)):
        x_series = [1]
        for j in range(order):
            x_series.append(x_series[-1]*x_vector[i])
        x_matrix.append(x_series)
    return np.array(x_matrix,dtype='float')
```

gradient_descent.py

```
import numpy as np

class GradientDescent(object):

    def __init__(self,x_matrix,y_tag,penalty_lambda,alpha,delta=1e-5):
        """
        x_matrix 的维度是 N*m，N 意味着样本数量，m 意味着多项式的阶数
        y_tag 的维度是 N*1,N 意味着 N 个样本相对应的目标值
        penalty:惩罚系数
        alpha：学习率
        delta：当梯度小于等于 delta 时停止算法
        """
        self.x_matrix = x_matrix
        self.y_tag = y_tag.reshape(y_tag.shape[0],1)
        self.penalty_lambda = penalty_lambda
        self.alpha = alpha
        self.delta = delta

    def __gradient(self,w):
        """
        计算当前的梯度
        w:当前的解 w
        """
        k1 = self.x_matrix.T @ self.x_matrix @ w
```

```
        k2 = self.x_matrix.T @ self.y_tag
        k3 = self.penalty_lambda * w
        return k1 - k2 + k3

    def __loss(self,w):
        """
        计算当前的损失值
        """
        f = self.penalty_lambda * w.T@w * 0.5
        return (self.x_matrix @ w - self.y_tag).T @ (self.x_matrix @ w - self.y_tag) * 0.5 + f

    def train(self):
        """
        获取模型结果
        返回轮数，结果，每一轮的损失值
        """
        w = np.zeros((self.x_matrix.shape[1], 1))
        k = 1
        current_gradient = self.__gradient(w)
        rounds = []
        loss = []
        while not np.absolute(current_gradient.T @ current_gradient) <= self.delta:
            w = w - self.alpha * current_gradient
            print(current_gradient)
            current_gradient = self.__gradient(w)
            rounds.append(k)
            loss.append(self.__loss(w))
            k+=1
            if np.isnan(w).any() :
                break
        return np.array(rounds),w,np.array(loss)
```

conjugat_gradient.py

```
import numpy as np

class ConjugatGradient(object):

    def __init__(self,x_matrix,y_tag,penalty,delta):
        self.x_matrix = x_matrix
        self.y_tag = y_tag
        self.penalty = penalty
```

```
        self.delta = delta

    def __switch(self,w_0):
        A = self.x_matrix.T @ self.x_matrix + self.penalty * np.eye(self.x_matrix.shape[1])
        x = w_0
        b = self.x_matrix.T @ self.y_tag
        return A,x,b

    def train(self,w_0):
        A,x,b = self.__switch(w_0)
        r = b - np.dot(A,x)
        p = r
        round = 0
        while True:
            if np.dot(p.T,p) < self.delta:
                return round,x
            else:
                pre_w = x
                a = np.dot(r.T,r)/np.dot(np.dot(p.T,A),p)
                x = pre_w + a*p
                pre_r = r
                r = pre_r - a*np.dot(A,p)
                b = np.dot(r.T,r)/np.dot(pre_r.T,pre_r)
                p = r + b*p
                round += 1

```

display.py

```
import matplotlib.pyplot as plt
import numpy as np
from gradient_descent import GradientDescent
from analytical_solution import AnalyticalSolution
from get_data import get_x_matrix

plt.rcParams['axes.facecolor']='snow'
plt.rcParams['font.sans-serif'] = ['SimHei']  #显示中文
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

def show_compared_to_data(w,x,y,title,function,xlimit=(0,1),ylimit=(-1,1),order=3):
    """
    函数功能：用于模型和给的数据之间的对比
```

```
    w:多项式参数，也就是训练好后的模型
    x:测试样本的自变量
    y:目标值
    xlimit 和 ylimit：自变量和因变量的显示范围
    order:模型阶数
    """
    plt.title(title)
    plt.scatter(x,y,c='red',label='噪声数据')
    plt.plot(x,np.dot(get_x_matrix(x,order),w),c='blue',label='模型')
    x=np.arange(xlimit[0],xlimit[1],0.01)
    y=function(x)
    plt.plot(x,y,c='lightgreen',label='y=sin(x)')
    plt.xlim(xlimit)
    plt.ylim(ylimit)
    plt.legend()
    plt.grid()
    plt.show()

def show_comparasion(w,x,y,title,colors,labels,function,xlimit=(0,1),ylimit=(-1,1),order=3):
    """
    函数功能：用于多个模型之间的对比
    w:多项式参数，也就是训练好后的模型，可以是多个
    x:测试样本的自变量
    y:目标值
    colors：每个模型应该采用的颜色标记
    labels：每个模型的文字标记
    xlimit 和 ylimit：自变量和因变量的显示范围
    order:模型阶数
    """
    assert len(w) == len(labels)
    assert len(labels) == len(colors)
    plt.title(title)
    for i in range(len(w)):
        plt.plot(x,np.dot(get_x_matrix(x,order),w[i]),c=colors[i],label=labels[i])
    x=np.arange(xlimit[0],xlimit[1],0.01)
    y=function(x)
    plt.plot(x,y,c='lightgreen',label='y=sin(x)')
    plt.xlim(xlimit)
    plt.ylim(ylimit)
    plt.legend()
    plt.grid()
```

```
    plt.show()

def show_Erms(x_train_data,y_train_data,x_test_data,y_test_data,order,title,penalty_lambda):
    """
    函数功能：分别打印在数据集和训练集上带惩罚项解析解的 Erms
    """
    plt.title(title)
    x_train_matrix = get_x_matrix(x_train_data,order)
    x_test_matrix = get_x_matrix(x_test_data,order)
    train_E_rms = []
    test_E_rms = []
    for l in penalty_lambda:
        e = np.exp(l)
        train = AnalyticalSolution(x_train_matrix,y_train_data)
        w = train.with_penalty(e)
        train_E_rms.append(train.E_rms(y_train_data,np.dot(x_train_matrix,w)))
        test_E_rms.append(train.E_rms(y_test_data,np.dot(x_test_matrix,w)))
    plt.xlabel('lnλ')
    plt.ylabel('Erms')
    plt.plot(penalty_lambda,train_E_rms,label='Erms on train set')
    plt.plot(penalty_lambda,test_E_rms,label='Erms on test set')
    plt.legend()
    plt.grid()
    plt.show()

def show_alpha(x_train_data,y_train_data,alpha,order,title,colors,labels,penalty_lambda):
    """
    函数功能：对梯度下降方法，对于不同的学习率展示比较曲线
    """
    plt.title(title)
    x_train_matrix = get_x_matrix(x_train_data,order)

    for i in range(len(alpha)):
        model = GradientDescent(x_train_matrix,y_train_data,penalty_lambda,alpha=alpha[i])
        w = model.train()
        print('梯度下降运行了{0}轮'.format(len(w[0])))
        plt.plot(w[0].reshape(w[0].shape[0],1),w[2].reshape(w[2].shape[0],1),color=colors[i],label=labels[i])
```

```
92.     plt.xlabel('rounds')
93.     plt.ylabel('Erms')
94.     plt.legend()
95.     plt.grid()
96.     plt.show()
97.
```