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：

姓名：

目录

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

2.1 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。

2.2 实验环境

Windows10+python3.7+PyCharm

三、设计思想（本程序中的用到的主要算法及数据结构）

由泰勒展开可知，足够高阶的多项式函数理论上可以拟合任意函数。因此本实验中使用的主要思路就是使用多项式来拟合一个非线性函数（本实验中使用的是 $sin(2\pi x)$）。而多项式函数拟合需要解决的就是多项式次数和系数的问题。对于 m 阶多项式而言有 m+1 个参数这些系数（由低到高，即从低次项系数到高次项系数）组成的列向量记作 w。使用最小二乘法确定 w 的时候，设 $E(w)=1/2*(Xw-Y)^T(Xw-Y)$，其中 X 表示多项式估计中每一项的值，与 w 相乘之后表示使用 w 作为估计参数的时候的估计值。例如 X[i][j]表示第 i 个观测数据的 j 次方的值。假设观测参数有 n 个，估计次数选择 m，则 X 的维度为 n*(m+1)。Y 表示观测数据的真实标签值。本实验的目标是调整 w 的值，使得 E(w)能取得最小值，这时也就表示使用 w 作为估计参数能够带来训练集下最好的估计效果。

3.1 数据生成

生成数据过程中使用 numpy 的函数生成一定范围内的 $sin(2\pi x)$ 对应的 X 和 Y，接着为 Y 加上一个高斯分布的噪声。同时使用 X 和 Y 生成各自的训练数据集，Y 的训练数据集只需要改变维度，最终是一个列向量即可，而 X 生成的训练数据集最终应该是如下格式：

$$\begin{bmatrix} 1, x_0, x_0^2 \ldots x_0^m \\ 1, x_1, x_1^2 \ldots x_1^m \\ \cdot \\ 1, x_n, x_n^2 \ldots x_n^m \end{bmatrix}$$

因此代码如下：

```
1.  def generate(degree,size,begin=0,end=1,mu=0,sigma=0.2):
2.      x = np.linspace(begin, end, size) #原始数据集
3.      noise=np.random.normal(mu,sigma,size)
4.      y = np.sin(2 * np.pi * x)+noise #原始数据集
5.      y_train=y.reshape(size,1) # 训练数据集，size*1
6.      x_train=np.zeros((size,degree+1))#训练数据集，size*(degree+1)
```

```
    degree_list=np.arange(0,degree+1)#阶数 list
    for i in range(size):
        temp=np.ones(degree+1)*x[i]
        x_train[i]=temp**degree_list
    return x,y,x_train,y_train
```

3.2 解析解

3.2.1 无正则项

无正则项解析解过程如下：

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

令：

$$\frac{\partial E}{\partial w}=X^TXw-X^TY=0$$

解得：

$$w=(X^X)^{-1}X^TY$$

由于对于有无正则项代码实现及其相似，因此在代码实现上有无正则项使用一个函数实现。有正则项

3.2.2 有正则项

解析解过程如下：

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

令：

$$\frac{\partial E}{\partial w}=X^TXw-X^TY+\lambda w=0$$

解得：

$$w=(X^TX+\lambda)^{-1}X^TY$$

综上所述，解析解的代码实现可以用一个函数实现：

```
def generate_w(lamda,x_train,y_train):
    '''''
    :param lamda: lamda=0 表示无正则项，lamda=1 表示有正则项
    :param x_train:训练数据集
    :param y_train:训练数据集
    :return:参数 w
    '''
    # w=np.matmul(np.matmul(np.linalg.inv(np.matmul(x_train.T,x_train)+
    #                             lamda*np.eye(x_train.shape[1])),x_train.T),
y_train)
    w=np.linalg.inv(np.dot(x_train.T, x_train) + lamda * np.eye(x_train.shap
e[1])).dot(x_train.T).dot(y_train)
    return np.poly1d(w[::-1].reshape(x_train.shape[1]))
```

3.4 梯度下降法

梯度下降是迭代法的一种，使用的基本思路就是每次优化的过程中都向 loss 的下降方向（也就是梯度损失函数梯度的反方向）求得 loss 的最小值对应的参数值。之所以沿着梯度的反方向进行更新是因为从几何意义上讲，梯度向量就是函数增加最快的方向，因此对于大多数损失函数的优化过程中由于希望损失函数的值取得最小值，因此大多数时候沿着梯度向量的相反方向，梯度减少最快，更容易找到函数的最小值。在本实验中使用时有如下过程：

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

$$\frac{\partial E}{\partial w} = X^T Xw - X^T Y + \lambda w = 0$$

假设学习率为α，则对于 w 而言有如下式子：

$$w = w - \alpha \frac{\partial E}{\partial w}$$

代码实现如下：

```
def gradient_optim(lamda,x_train,y_train,alpha,epsilon,train_num):
    '''''
    :param lamda: lamda=0 表示无正则项，lamda=1 表示有正则项
    :param x_train: 训练数据集
    :param y_train: 训练数据集
    :param alpha: 学习率
    :param epsilon: 精度
    :param train_num:训练最大次数
    :return: 优化后的参数取值
    '''
    w=np.zeros((x_train.shape[1],1)) #(degree+1)*1
    new_loss=abs(loss(x_train,y_train,w,lamda))
    for i in range(train_num):
        old_loss=new_loss
        gradient=np.matmul(np.matmul(x_train.T,x_train),w)-np.matmul(x_train.T,y_train)+lamda*w
        temp_w=w
        w-=alpha*gradient
        new_loss=abs(loss(x_train,y_train,w,lamda))
        if new_loss-old_loss>0:##新一步的 loss 比之前一步 loss 还要大，表示学习率太大
          w=temp_w
          alpha/=2
        if old_loss-new_loss<epsilon:##达到精度要求，停止训练
            break
    parameters = np.poly1d(w[::-1].reshape(x_train.shape[1]))
    return parameters
```

3.5 共轭梯度法

3.5.1 原理解析

首先，共轭梯度法是一种求多元二次函数 $f(x_1, x_2, x_3, ...x_n)$ 的极值点的方法，其需要解决的问题与梯度下降法类似，但是相对于梯度下降来说收敛速度更快，但是相应的也就更不稳定，更容易出现过拟合的现象。

由于共轭梯度法考虑的是多元二次函数的极值点问题，因此可以将函数写为：$f(x_1,...,x_n) = 1/2 x^T ax - b^T x + c$。求导可知这个函数的导数为：$\frac{df(x)}{dx} = Ax - b = \nabla f(x)$。共轭梯度的思路就是每一步行进的都将当前方向都优化到了最优，也即是说寻找极值的过程中绝不走曾经走过的方向，由此可见理论上 n 维空间中的优化问题只需要 n 步就可以解决，这也是为什么它比梯度下降法快的原因。假设真实的最优解是 x^*，当前解为 x_t，那么记误差为 e_t。

由于我们每一步都需要在当前方向优化到最优，因此很容易发现我们每一次都要保证更新之后的误差向量和更新方向是正交的。假设下一步方向定义为 r_t，那么正交就可以用如下的式子表示：$r_{t-1}^T e_t = 0$，接着进行一个共轭正交的变换，变换为 $r_{t-1}^T \mathrm{A} e_t = 0$，其中矩阵 A 是一个常对称矩阵。

接下来需要确定的就是每一步的步长和方向。首先确定步长 α_t，需要保证正交成立，也就

是需要保证：$r_{t-1}^{T}\mathrm{A}e_t=0$，利用$e_t=x^*-x_t=e_t+x_t-x_{t+1}$对式子进行变换可得：

$$r_t^T A_{et+1}=r_t^T A[e_t+x_t-x_{t+1}]=r_t^T Ae_t-\alpha_t r_t^T Ar_t=0$$

可得：

$$\alpha_t=\frac{r_t^T Ae_t}{r_t^T Ar_t}=\frac{r_t^T A(x^*-x_t)}{r_t^T Ar_t}=\frac{r_t^T(b-Ax_t)}{r_t^T Ar_t}=-\frac{r_t^T\nabla f(x_t)}{r_t^T Ar_t}$$

接下来确定每一次更新的方向，事实上就是使用施密特正交化进行每一次更新，每一步的方向都是在起点的负梯度方向进行一定的正交修正，也就是说进行施密特正交化的线性无关组是每一步终点的负梯度构成的，使用如下式子：$r_t=-\nabla f(x_t)+\sum_{i<t}\frac{r_i^T A\nabla f(x_t)}{r_i^T Ar_i}r_i$。至此已经确定了每一个参数的更新方式。

3.5.2 代码实现

列出所有需要更新的值的更新方式如下：

$$\frac{\partial E}{\partial w}=X^T Xw-X^T Y+\lambda w=0$$

记 A=$X^T X+\lambda$,b=$X_T Y$，初始化为$w_0=0, r_0=b, p_0=b$，

$$a_k=\frac{r_l^T r_k}{p_k^T \mathrm{A}p_k}$$

$$w_k=w_k+a_k p_k$$

$$r_{k+1}=r_k-a_k Ap_k$$

$$b_k=\frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

$$p_{k+1}=r_{k+1}+b_k p_k$$

代码实现如下：

```
def conjugate(lamda,x_train,y_train,epsilon):
    # Aw=b 其中 A=x'x+lamda，b=x'y
    A=np.matmul(x_train.T,x_train)+lamda*np.eye(x_train.shape[1])#(degree+1)*(degree+1)
    b=np.matmul(x_train.T,y_train)#(degree+1)*1
    w=np.zeros((x_train.shape[1],1))
    r=b
    p=b
    k=0
    while True:
        k=k+1
        temp=np.matmul(r.T,r)
        a=np.matmul(r.T,r)/(np.matmul(np.matmul(p.T,A),p))
        w=w+a*p
        r=r-np.matmul(a*A,p)
        if np.matmul(r.T,r)<epsilon:
            break
        b=np.matmul(r.T,r)/temp
        p=r+b*p
    parameters = np.poly1d(w[::-1].reshape(x_train.shape[1]))
    return parameters
```

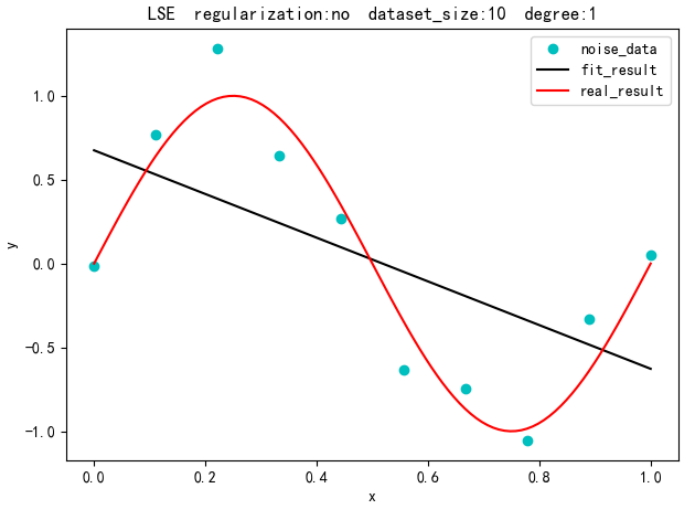
四．实验结果与分析

4.1 解析法

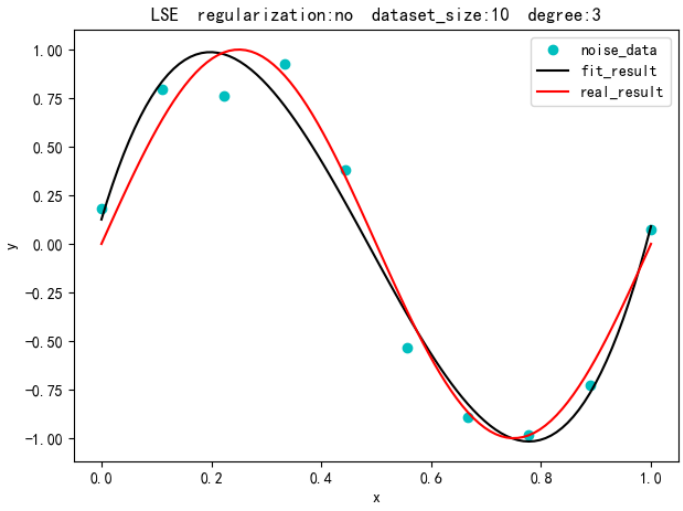
4.1.1 无正则项

训练集大小为 10，精度为 0.000001，观察在不同阶数下的拟合结果：

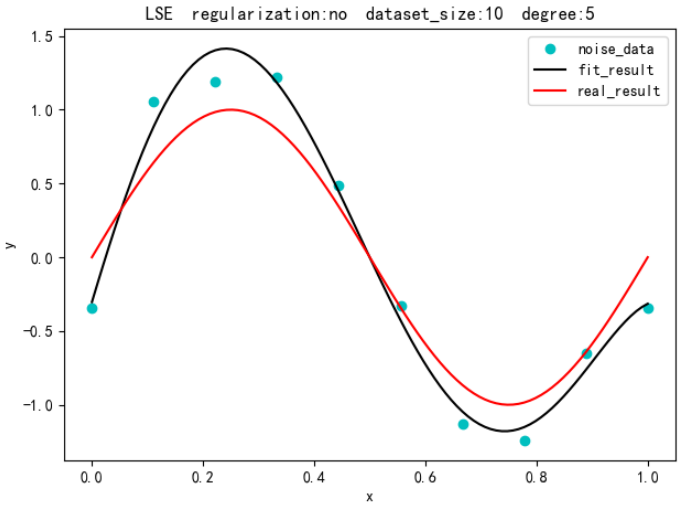
1 阶 y=-1.258 x + 0.6234



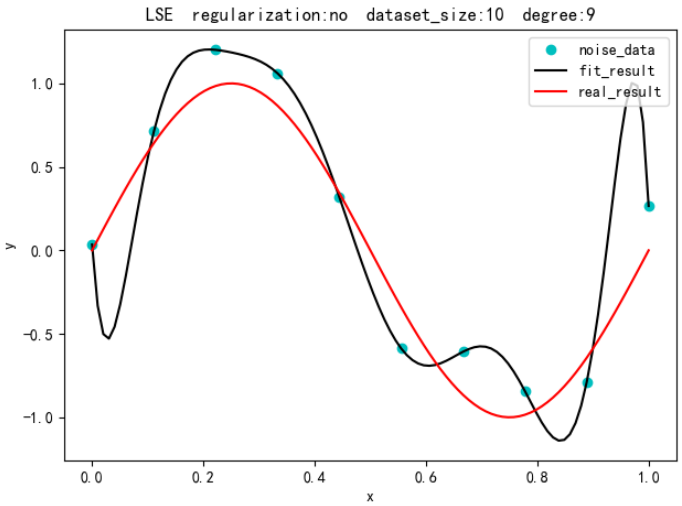
3 阶 $y = 17.71x^3 - 26.57x^2 + 8.14x + 0.4028$



5 阶 $y = -55.83x^5 + 132x^4 - 88.31x^3 + 5.925x^2 + 6.37x - 0.1689$

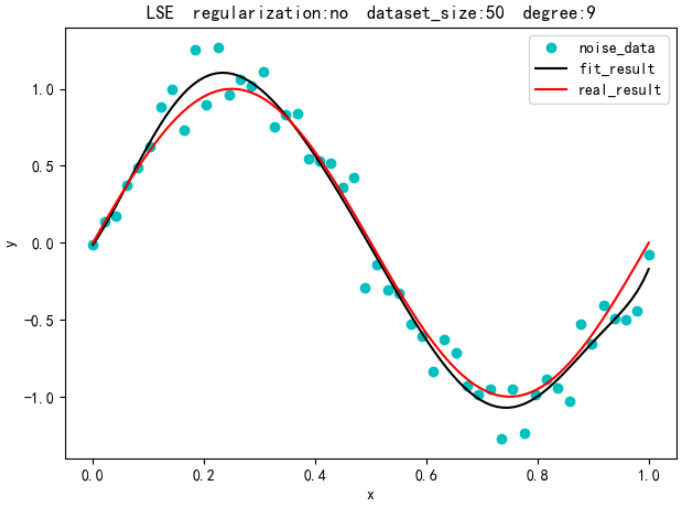


9 阶（由于拟合式子过于繁琐，不再展示，只展示拟合图片）：

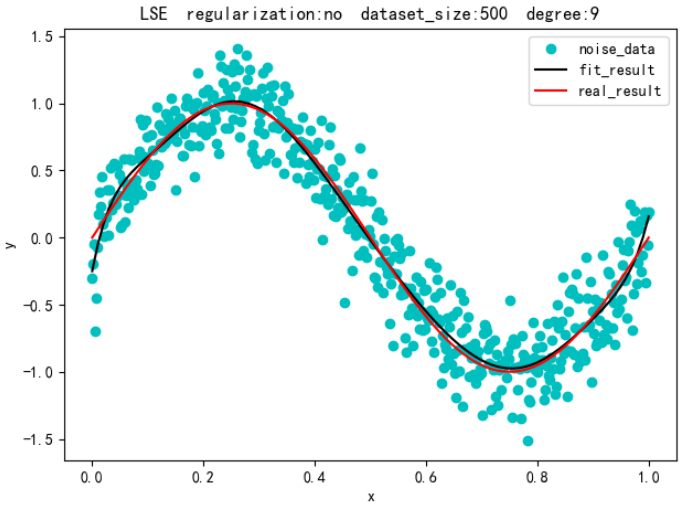


可以发现当使用 1 阶函数拟合的时候，基本没有拟合效果，属于欠拟合，不断提升阶数发现当阶数为 9 的拟合函数穿过了每一个数据点，但是很明显和目标函数相差很大，属于过拟合现象，为了解决这种过拟合现象，可以考虑增大数据集或者降低阶数，以下展示增大数据集大小的解决方法的效果：

数据集大小：50



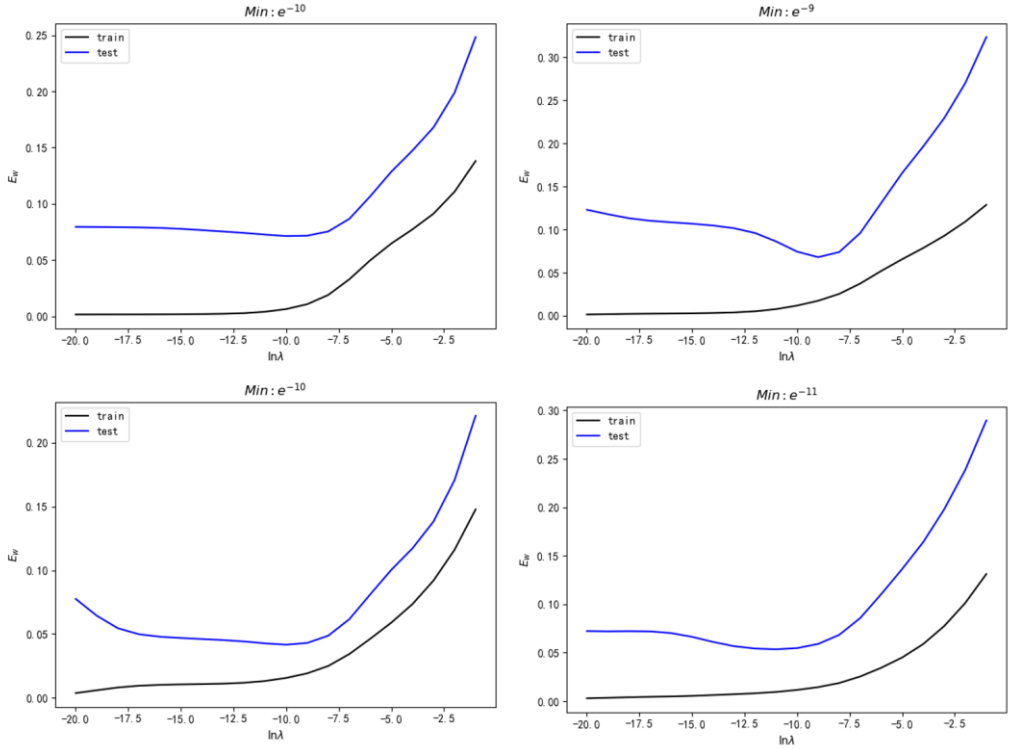
数据集大小：500



可以发现当出现过拟合的时候，如果不断增大数据集的大小可以消除过拟合的现象。观察上述拟合可以发现当数据集大小为 500 的时候基本上已经和预计的原函数拟合。

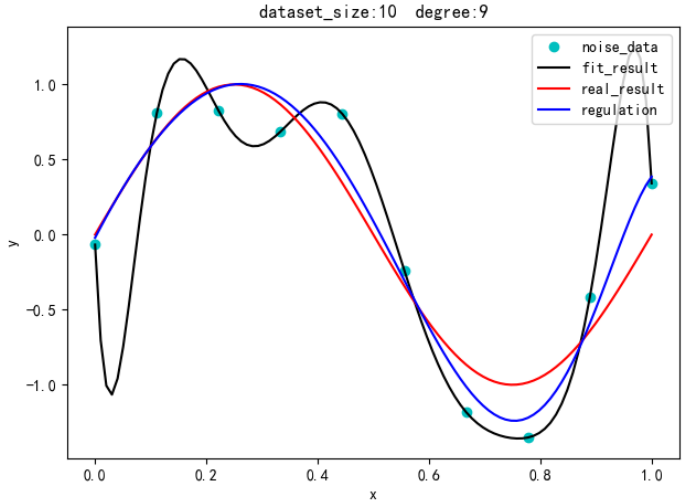
4.1.2 有正则项

带正则项的估计需要确定一个相对较优的λ，可以使用损失函数来对于使用不同正则项参数的时候的效果进行估计：



由于正则项的目的是抑制过拟合，因此我们更关心在测试集上的损失函数值，因此进行多次实验之后发现最优的λ大约在（e^{-12}，e^{-8}）

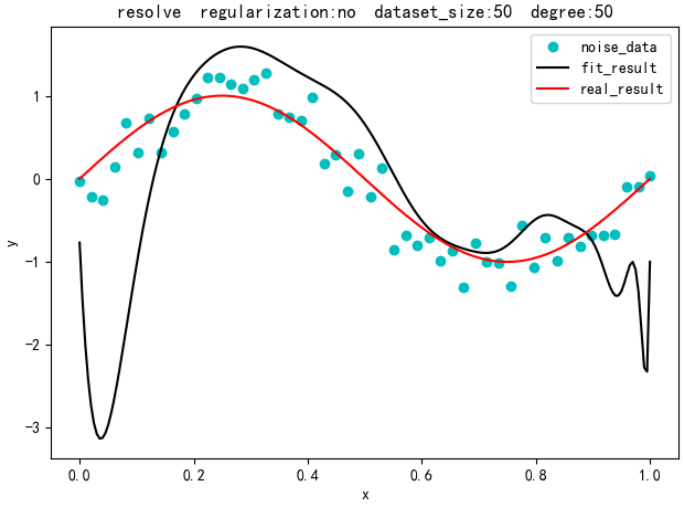
选择$\lambda = e^{-9}$，取训练集大小为 10，阶数为 9，将带正则项与不带正则项的图像进行对比：



可以发现加入正则项有效降低了过拟合出现的情况。

4.1.3 解析解处理高阶拟合出错问题讨论

经过测试可以发现，在使用解析解作为求解方式的时候，如果使用最高次数为 50 的多项式来拟合数据，无论数据集的大小，总是明显出现拟合错误，详细拟合结果见下图（数据集大小 50，阶数 50）：

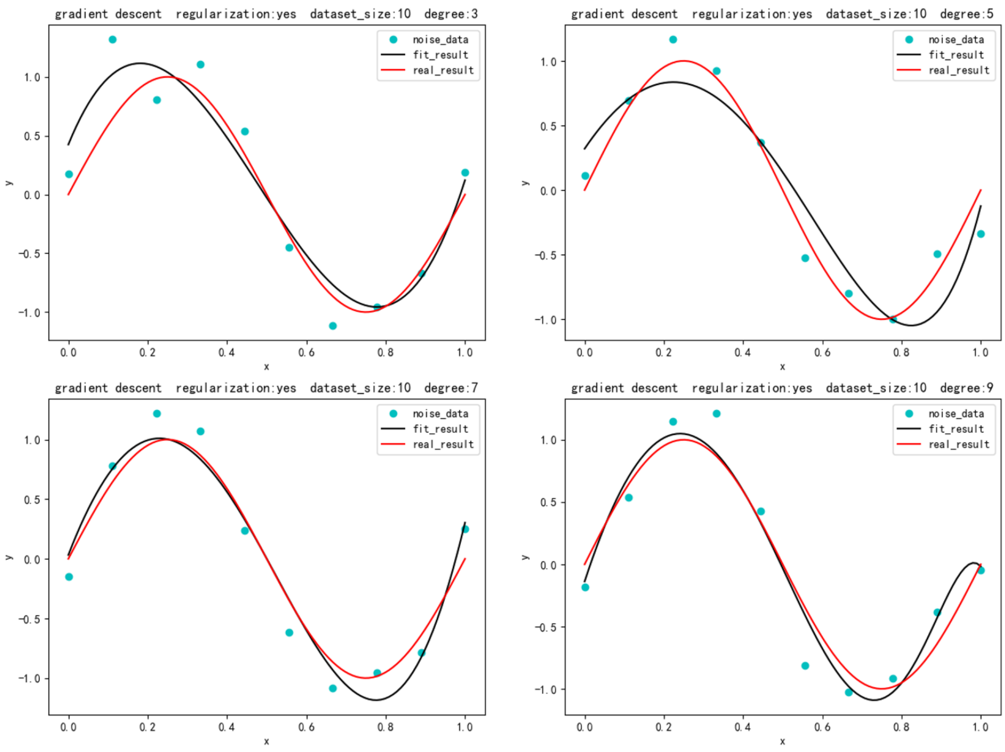


明显上图出现了错误的求解，通过查询一些资料与源码，可知一般情况下 np.linalg.inv 方法默认情况下在不检测正交性的情况下进行 LU 分解之后计算矩阵的逆，由于在这个过程中需要进行较多的浮点运算，因此猜测是因为在这一过程中当阶数太高的时候发生了一定的浮点计算误差，导致了降低了性能，增加了数值误差。而又因为当阶数很高的时候对于每一个系数的依赖都比较高，比较小的误差反应出的最终拟合误差都是巨大的，因此带来了上述误差。

4.2 梯度下降法

4.2.1 阶数影响

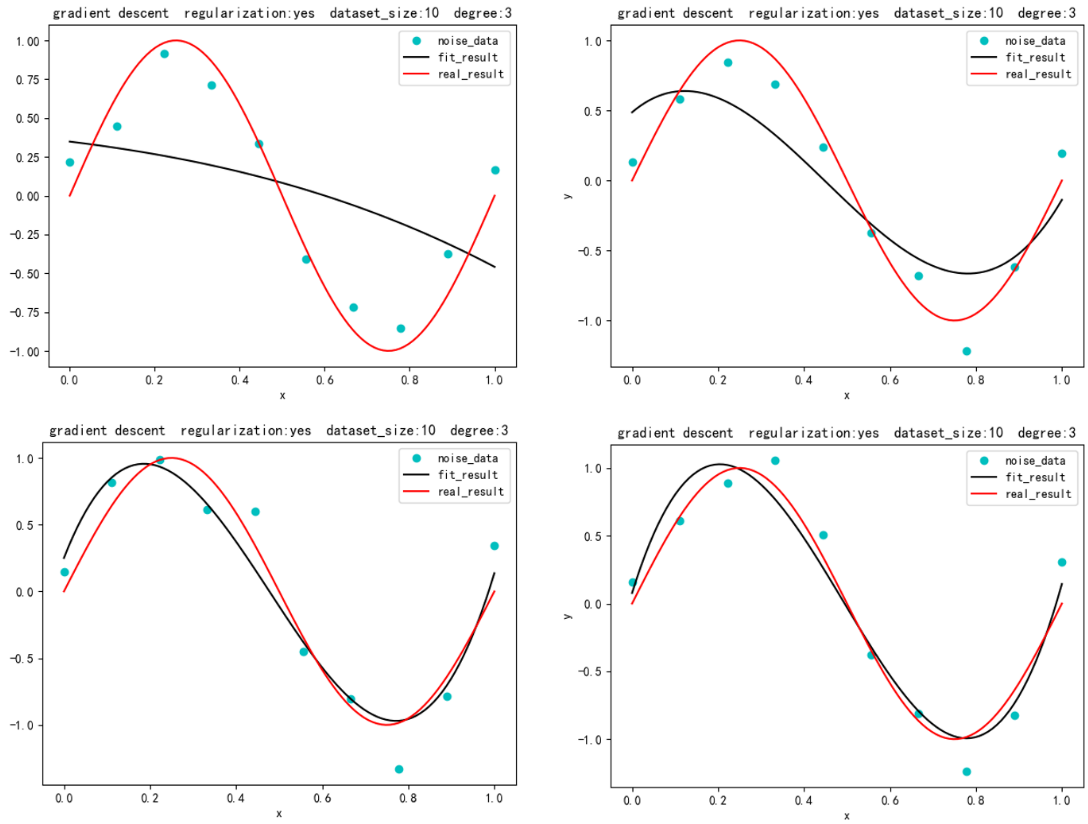
考虑阶数对于拟合效果的影响，固定数据集大小为 10，精度为10^{-6}，正则项为$\lambda = e^{-9}$



通过观察发现在一定程度上增大拟合的阶数可以提高拟合效果，且由于加入了正则项，尽管次数上升，在梯度下降方法中过拟合现象出现的不是很明显。

4.2.2 精度影响

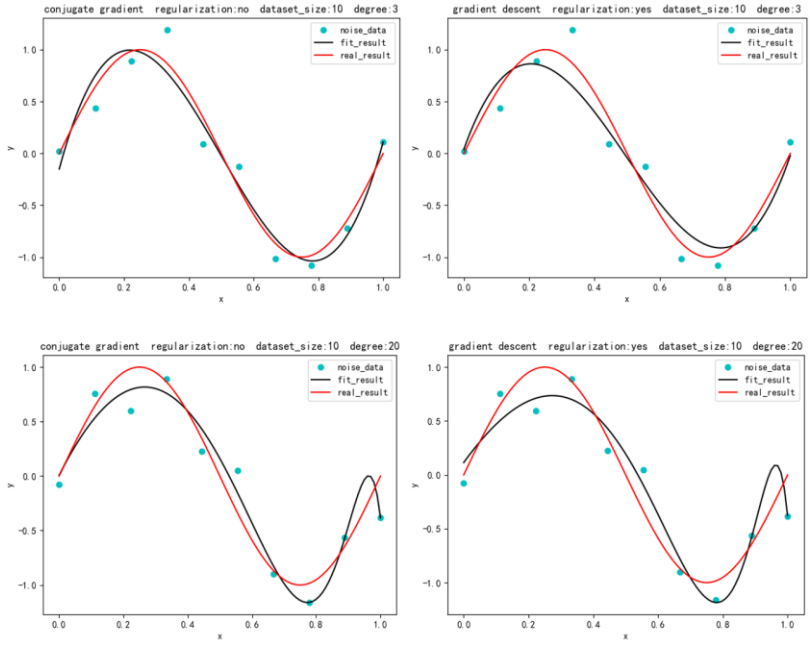
考虑精度影响的时候固定阶数 3，数据集大小 10，正则项为 $\lambda = e^{-9}$。下图中从左上到右下依次精度为10^{-3}，10^{-5}，10^{-6}，10^{-7}：



可以发现精度的增加可以使得估计效果明显上升，同时也使得迭代次数增大。因此如果训练数据集较大，或是每一次迭代计算代价较大，精度需要与最终的估计效果进行权衡。

4.2.3 正则项影响

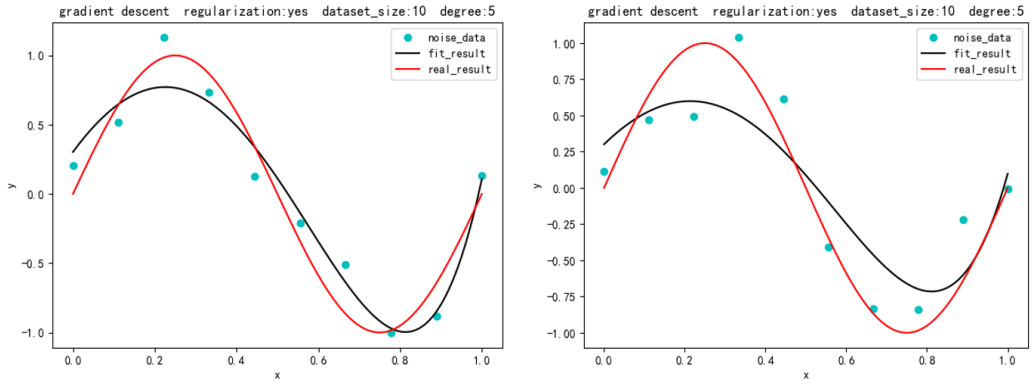
考虑正则项影响，固定数据集大小为 10，精度为10^{-6}，下方图中四张图，上方两张图阶数为 3，下方阶数为 20，左侧两张图无正则项，右侧两张图有正则项。



可以发现一个问题：对于梯度下降来说似乎正则项并不能很好起到抑制过拟合现象，通过查看训练时的 loss，可以发现 loss 还未收敛，也就是说上述图中都还未达到在训练集上最理想的情况，因此正则项起不到明显作用，这也印证了梯度下降法收敛较慢的理论推导。由于实验中经过测试梯度下降难以收敛，因此无法展示梯度下降法收敛情况。

4.2.4 学习率影响

考虑学习率的影响，固定阶数为 5，数据集大小为 10，精度为10^{-6}，正则项为$\lambda = e^{-9}$。下图中左图学习率为 0.01，右图为 0.001

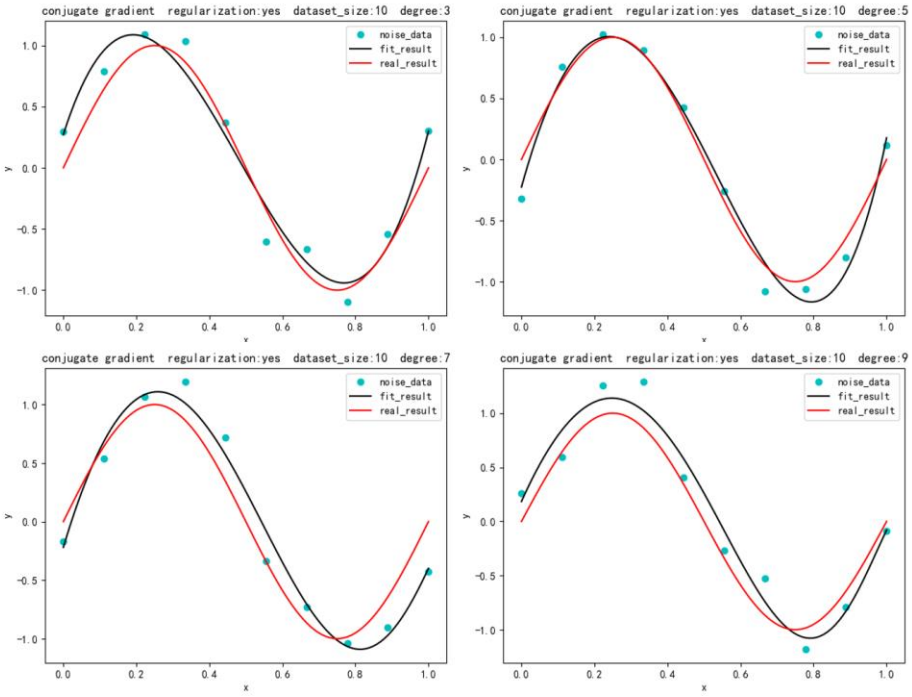


观察由于都未到达最优点，因此学习率较大的好处就是能以较快的速度收敛。但是学习率如果选择太大很容易在最优点附近发生震荡，从而导致无法收敛到最优点。

4.3 共轭梯度法

4.3.1 阶数影响

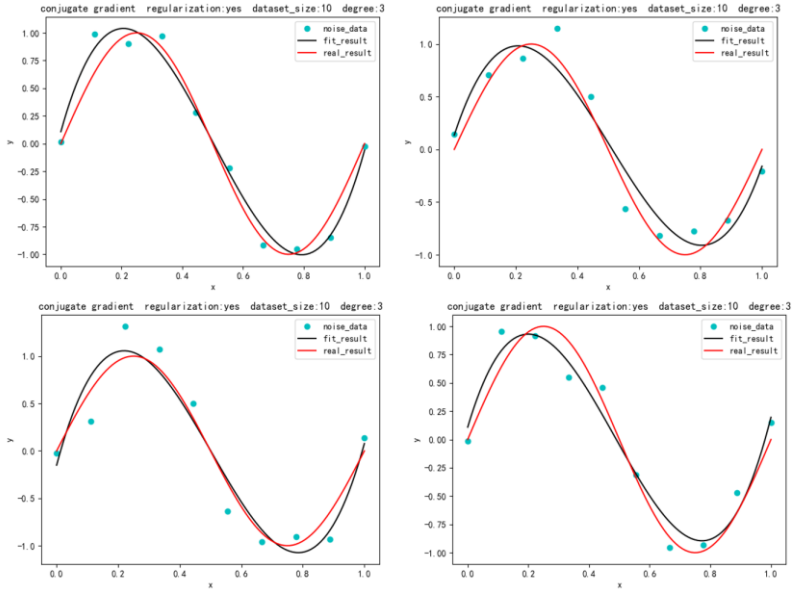
考虑阶数对于拟合效果的影响，固定数据集大小为 10，精度为10^{-6}，正则项为$\lambda = e^{-9}$



通过观察发现在一定程度上增大拟合的阶数可以提高拟合效果，但是较为明显的是尽管加入了正则项约束，但是在阶数增大的时候共轭梯度法还是在一定程度上表现出了过拟合的趋势，因此尽管有了正则项，拟合的阶数也不可以无限增长。

4.3.2 精度影响

考虑精度影响的时候固定阶数 3，数据集大小 10，正则项为$\lambda = e^{-9}$。下图中从左上到右下依次精度为10^{-3}，10^{-5}，10^{-6}，10^{-7}：

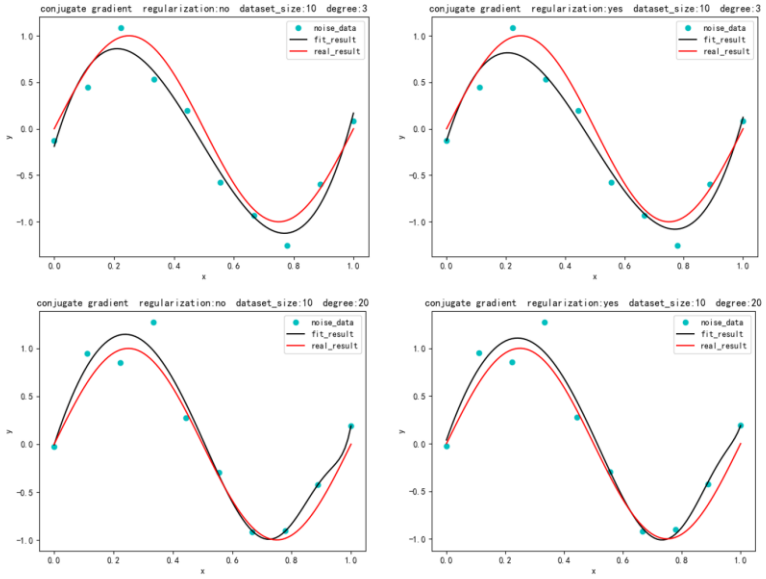


可以发现精度的增加可以使得估计效果明显上升，同时也使得迭代次数增大。因此如果训练数据集较大，或是每一次迭代计算代价较大，精度需要与最终的估计效果进行权衡。

4.3.3 正则项影响

考虑正则项影响，固定数据集大小为 10，精度为10^{-6}，下方图中四张图，上方两张图

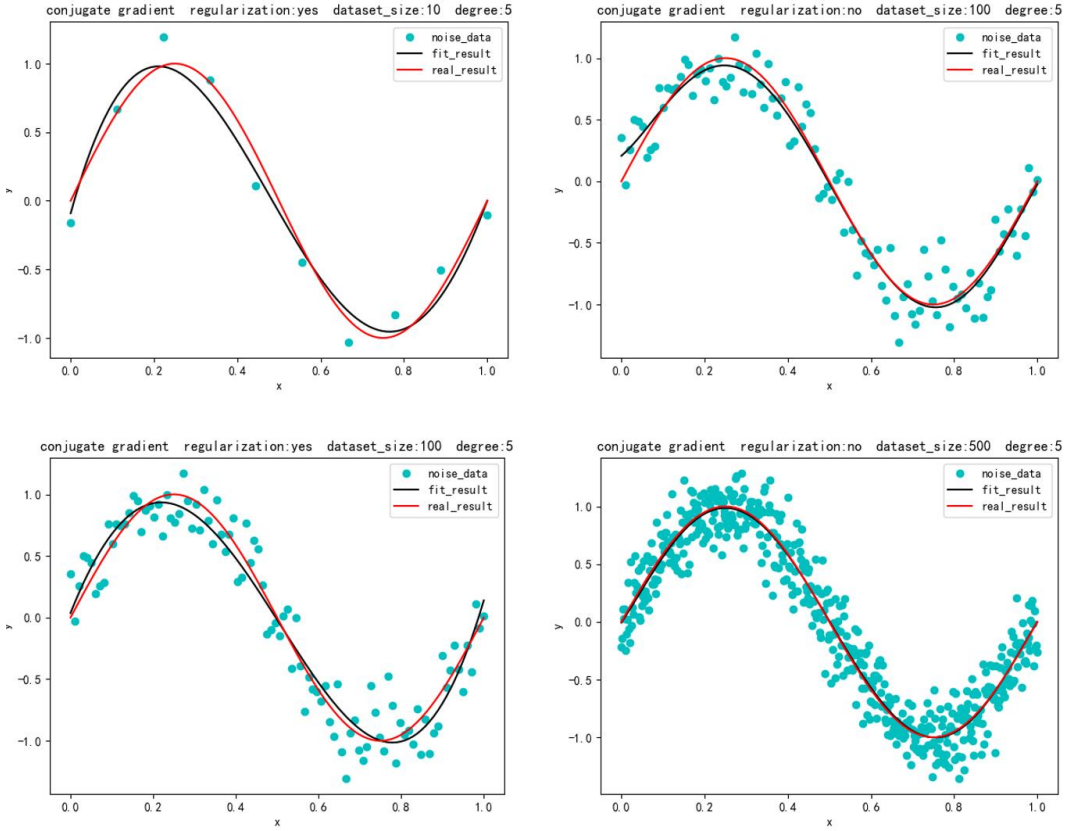
阶数为 3，下方阶数为 20，左侧两张图无正则项，右侧两张图有正则项。



观察可以发现当阶数较低的时候无正则项的拟合效果比有正则项略好，而当阶数较大的时候无正则项的过拟合比有正则项更为严重。这是符合直观认识的，当阶数较低的时候基本不出现过拟合，那么正则项在一定程度上将会约束拟合的效果；而当出现明显的过拟合现象的时候过拟合起到抑制过拟合的效果。事实上可以说正则项作为惩罚项主要的功能就是抑制拟合的效果从而达到抑制过拟合的效果。

4.3.4 数据集大小影响

考虑数据集大小的影响，固定阶数为 5，精度为10^{-6}，正则项为$\lambda = e^{-9}$。



观察发现在其他参数一致的情况下数据集越大拟合效果越好，这与直觉基本符合。

五．结论

基于本次实验使用多项式拟合正弦函数的任务中可以发现：

多项式次数的提高能够提高模型对于训练集数据的拟合效果，但是由于能力太强也更可能出现过拟合现象，这种过拟合也就导致了模型拟合出的多项式通过了训练数据集中的每一个点，但是无法拟合出正弦函数曲线。通过增大数据集规模可以有效缓解过拟合。

正则项作为惩罚项可以有效缓解过拟合现象，这是因为训练过程中如果某几项参数值比较大，正则项也增大，从而整个 loss 增大，进行下一步优化，而某几项参数很大的情况很可能出现过拟合（直观上来说在图像上反映出来的就是很容易出现梯度绝对值的点），因此这就是为什么正则项可以有效缓解过拟合。

对于精度来说，一般来说如果不限制迭代次数，精度越高拟合效果越好。

针对梯度下降法，学习率表示了每一步更新时候的步长，使用的学习率较大的时候可能会更快到达最优点附近，但是也意味着可能跳过最优点，在最优点附近震荡；而选用的学习率如果太小可能造成迭代很多轮之后还是无法到达最优点附近。

比较三种方法，解析法肯定是最精确的，但是如果针对较为复杂的任务解析法显然比较难以完成任务；而比较梯度下降和共轭梯度，共轭梯度收敛速度更快，但是稳定性也会更差，比较容易出现过拟合现象。因此这两种方法在实际使用过程中需要视情况而定。

六．参考文献

[1]https://baike.baidu.com/item/%E6%96%BD%E5%AF%86%E7%89%B9%E6%AD%A3%E4%BA%A4%E5%8C%96/756386?fr=aladdin。

[2] https://zhuanlan.zhihu.com/p/64227658

[3] numpy.linalg.inv 如何计算正交矩阵的逆？ - 问答 - Python 中文网 (cnpython.com)

七、附录：源代码（带注释）

```
import numpy as np
import matplotlib.pyplot as plt
##保证绘图中输出中文正常
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
## 生成数据并加入噪声,sin(2pix),同时生成训练数据集
def generate(degree,size,begin=0,end=1,mu=0,sigma=0.2):
    '''''
    :param degree:拟合的阶数
    :param size:数据集大小
    :param begin:数据集开始点
    :param end:结束点
    :param mu:噪音的均值
    :param sigma:噪音方差
    :return:无噪音数据集，加上噪音数据集
    '''
    x = np.linspace(begin, end, size) #原始数据集
    noise=np.random.normal(mu,sigma,size)#噪音
    y = np.sin(2 * np.pi * x)+noise #原始数据集
    degree_list = np.arange(0, degree + 1)  # 阶数 list
    x_train = np.zeros((size, degree + 1))  # 训练数据集，size*(degree+1)
    for i in range(size):
        temp=np.ones(degree+1)*x[i]
        x_train[i]=temp**degree_list
    y_train = y.reshape(size, 1)  # 训练数据集，size*1
    return x,y,x_train,y_train
##计算 loss
def loss(x_train,y_train,w,lamda):
    '''''
    :param x_train: 训练数据集
```

```
    :param y_train: 训练数据集
    :param w: 估计参数
    :param lamda: 正则项系数
    :return:loss
    '''
    temp=np.matmul(x_train,w)-y_train
    return (np.matmul(temp.T,temp)+lamda*np.matmul(w.T,w))/2
##解析法计算参数
def resolve(lamda, x_train, y_train):
    '''''
    :param lamda: 表示正则项系数，==0 的时候表示无正则项
    :param x_train:训练数据集
    :param y_train:训练数据集
    :return:参数 w,解析解的结果
    '''
    w=np.matmul(np.matmul(np.linalg.inv(np.matmul(x_train.T,x_train)+
                                lamda*np.eye(x_train.shape[1])),x_train.T),y_
train)
    return w,np.poly1d(w[::-1].reshape(x_train.shape[1]))
##梯度下降求最优解
def gradient_optim(lamda,x_train,y_train,alpha,epsilon,train_num):
    '''''
    :param lamda: 表示正则项系数，==0 的时候表示无正则项
    :param x_train: 训练数据集
    :param y_train: 训练数据集
    :param alpha: 学习率
    :param epsilon: 精度
    :param train_num:训练最大次数
    :return: 优化后的参数取值
    '''
    w=np.zeros((x_train.shape[1],1)) #(degree+1)*1
    new_loss=loss(x_train,y_train,w,lamda)
    temp=0
    for i in range(train_num):
        old_loss=new_loss
        gradient=np.matmul(np.matmul(x_train.T,x_train),w)-np.matmul(x_train
.T,y_train)+lamda*w
        temp_w=w
        w-=alpha*gradient
        new_loss=loss(x_train,y_train,w,lamda)
        temp=i
        if new_loss-old_loss>0:##新一步的 loss 比之前一步 loss 大，学习率太大
          w=temp_w
          alpha/=2
        if new_loss-old_loss<epsilon:##达到精度要求，停止训练
            break
    print('梯度下降代数：'+str(temp))
    return np.poly1d(w[::-1].reshape(x_train.shape[1]))
##共轭梯度法优化
def conjugate(lamda,x_train,y_train,epsilon):
    '''''
    :param lamda: 表示正则项系数，==0 的时候表示无正则项
    :param x_train: 训练集
    :param y_train:训练集
    :param epsilon:训练精度
    :return:优化结果
    '''
    # Aw=b 其中 A=x'x+lamda，b=x'y
    A=np.matmul(x_train.T,x_train)+lamda*np.eye(x_train.shape[1])#(degree+1)
*(degree+1)
    b=np.matmul(x_train.T,y_train)#(degree+1)*1
    w=np.zeros((x_train.shape[1],1))
```

```
    r=b
    p=b
    k=0
    while True:
        k=k+1
        temp=np.matmul(r.T,r)
        a=np.matmul(r.T,r)/(np.matmul(np.matmul(p.T,A),p))
        w=w+a*p
        r=r-np.matmul(a*A,p)
        if np.matmul(r.T,r)<epsilon:
             break
         b=np.matmul(r.T,r)/temp
         p=r+b*p
    return np.poly1d(w[::-1].reshape(x_train.shape[1]))
def figure_show(x, y,fit, title):
    '''''
    :param x:数据集
    :param y: 数据集
    :param fit: 估计多项式
    :param title: 图名
    :return: 绘图
    '''
    plt.plot(x, y,'co', label='noise_data')
    real_x = np.linspace(0, 1, 200)
    real_y = np.sin(real_x * 2 * np.pi)
    fit_y = fit(real_x)
    plt.plot(real_x, fit_y, 'black', label='fit_result')
    plt.plot(real_x, real_y, 'r', label='real_result')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(loc=1)
    plt.title(title)
    plt.show()
def loss_show(x_train,y_train,x_test,y_test,train_size,test_size):
    '''''
    :param x_train: 训练集
    :param y_train: 训练集
    :param x_test: 测试集
    :param y_test: 测试集
    :param train_size: 训练集大小
    :param test_size: 测试集大小
    :return: 绘图
    '''
    ln_lamda=list(np.linspace(-20,-1,20))
    loss_train = []#size=20
    loss_test = []#size=20
    for i in range(20):
        lamda=np.exp(ln_lamda[i])
        w,fit=resolve(lamda, x_train, y_train)
        loss_train.append(float(loss(x_train,y_train,w,lamda)/train_size))

        loss_test.append(float(loss(x_test,y_test,w,lamda)/test_size))
    plt.plot(ln_lamda, loss_train, 'black', label='train')
    plt.plot(ln_lamda, loss_test, 'b', label='test')
    min_test_loss=min(loss_test)
    min_loss_index=loss_test.index(min_test_loss)
    lamda=min_loss_index-20
    plt.xlabel('$\ln \lambda$')
    plt.ylabel('$E_w$')
    plt.legend(loc=2)
    plt.title('$Min: e^{' + str(lamda) + '}$')
    plt.show()

def draw(x,y,fit,method,lamda,size,degree):
```

```
    '''''
    :param x:数据集
    :param y:数据集
    :param fit:估计式子
    :param method:估计方法名称，str
    :param lamda:正则项系数
    :param size:数据集大小
    :param degree:估计多项式次数
    :return:
    '''
    if lamda!=0:
        lamda='yes'
    else:
        lamda = 'no'
    title=method+'  regularization:'+lamda+'  dataset_size:'+str(size)+'  degree:'+str(degree)

    figure_show(x,y,fit,title)
def compare_draw(x,y,fit1,fit2,fit3,lamda,size,degree):
    '''''
    :param x:数据集
    :param y:数据集
    :param fit1:第一种估计（在本例中为解析法）
    :param fit2:第二种估计
    :param fit3:第三种估计
    :param lamda:正则项系数
    :param size:数据集规模
    :param degree:估计的次数
    :return:画图
    '''
    plt.plot(x, y, 'co', label='noise_data')
    real_x = np.linspace(0, 1, 100)
    real_y = np.sin(2 * np.pi*real_x )
    fit_y_1 = fit1(real_x)
    fit_y_2 = fit2(real_x)
    fit_y_3 = fit3(real_x)
    plt.plot(real_x, fit_y_1, 'b', label='RESOLVE_fit_result')
    plt.plot(real_x, fit_y_2, 'r', label='GD_fit_result',linestyle='--')
    plt.plot(real_x, fit_y_3, 'y', label='CG_fit_result',linestyle='--')
    plt.plot(real_x, real_y, 'g', label='real_result',linestyle='--')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(loc=1)
    plt.title('regularization:'+str(lamda)+'  dataset_size:'+str(size)+'  degree:'+str(degree))
    plt.show()
if __name__=="__main__":
    degree=9
    size=10
    alpha=0.001
    epsilon=1e-10
    train_num=10000000
    lamda= 0

    x,y,x_test,y_test=generate(degree,20,begin=0,end=1)
    x, y, x_train, y_train = generate(degree, size)
    w,fit1=resolve(lamda,x_train,y_train)
    fit2=gradient_optim(lamda,x_train,y_train,alpha,epsilon,train_num)
    fit3=conjugate(lamda,x_train,y_train,epsilon)
    loss_show(x_train,y_train,x_test,y_test,size,size)
    draw(x,y,fit1,'resolve',lamda,size,degree)
    draw(x, y, fit2, 'gradient descent', lamda, size, degree)
    draw(x, y, fit3, 'conjugate gradient', lamda, size, degree)
```