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

学号：1190201303
姓名： 王艺丹

一、 实验目的

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。

二、 实验要求及实验环境

Windows 10；Anaconda 4.8.4；python 3.7.4；jupyter notebook 6.0.1

三、 概念设计思想（主要算法及数据结构）

3.1 生成数据

算法思想：

利用函数$y=\sin{(2\pi x)}$产生样本，其中x均匀分布[0,1]在之间，对于每一个目标值x增加一个均值为mu，方差为$sigma$的高斯噪声得到$train_y$，则$(x, train_y)$即为训练数据集。

具体实现：

```
# 生成训练集数据 为sin2pix添加mu为均值，sigma为标准差的高斯噪声
def generate_data(mu, sigma):
    train_x = np.arange(0, 1, 1/sam_num) #start, end, 步长
    guass_noise = np.random.normal(mu, sigma, sam_num)
    train_y = np.sin(2 * np.pi * train_x) + guass_noise
    return train_x, train_y
```

3.2 利用高阶多项式函数拟合曲线(不带惩罚项)

算法思想：

利用给定训练数据集合，对于每个新的x，预测目标值y。采用多项式函数进行学习，即利用式(1)来确定参数，假设阶数m $(poly_deg)$已知：

$$y(x,w)=w_0+w_1x+w_2x^2+\cdots+w_mx^m=\sum_{i=1}^{m}w_ix^i \tag{1}$$

利用最小二乘法，计算实际目标值$y(x_i,w)$与估计值t_i的误差：

$$E(w)=\frac{1}{2}\sum_{i=1}^{m}\{y(x_i,w)-t_i\}^2 \tag{2}$$

将式(2)写为矩阵形式：

$$E(w)=\frac{1}{2}(Xw-T)'(Xw-T) \tag{3}$$

其中：

$$X=\begin{bmatrix}1 & x_1 & \cdots & x_1^m\\ 1 & x_2 & \cdots & x_2^m\\ \vdots & \vdots & \ddots & \vdots\\ 1 & x_N & \cdots & x_N^m\end{bmatrix}, w=\begin{bmatrix}w_0\\ w_1\\ \vdots\\ w_m\end{bmatrix}, T=\begin{bmatrix}t_0\\ t_1\\ \vdots\\ t_N\end{bmatrix}$$

对式(3)求导得：

$$\frac{\partial E}{\partial w}=X'Xw-X'T \tag{4}$$

令式(4)得零得：

$$w^*=(X'X)^{-1}X'T \tag{5}$$

注意到其中$(X'X)^{-1}X'$为X的伪逆。

具体实现：

使用 numpy 中的矩阵求伪逆、点乘等运算很容易求解，不做过多阐述。

```python
# 无惩罚性项
def fit_noloss(train_x, train_y):
    X = trans_martix(train_x)
    Y = train_y.reshape(-1, 1) # 将train_y转为列向量
    w = np.linalg.pinv(X).dot(Y) # w为各系数按幂由低至高排列构成的列向量
    w = w[::-1].reshape(1,-1).ravel() # 转为按降幂顺序排列的一维数组
    poly = np.poly1d(w) # 对应的多项式结果
    return poly
```

注意最终需要将所求得的w倒序转为一维数组，运用自带的 poly1d 函数进行多项式结果输出；其中 trans_martix 函数是将给定 x 集合转为形如 X 的矩阵，算法实现如下：

```python
# 将向量存储为samp_num*(poly_deg+1)的矩阵中
'''
w为poly_deg阶多项式中poly_deg+1个系数按幂由低到高组成的列向量
对应得到的矩阵X，设X_j为矩阵第j行向量，则X_j为（...,w_k*第j个train_x^k,...）k从0递增至poly_deg
易知矩阵X为samp_num*(poly_deg+1)阶
'''
def trans_martix(train_x):
    X = np.zeros((sam_num, poly_deg + 1)) # 初始化矩阵为0
    for i in range(sam_num):
        row = np.ones(poly_deg + 1) * train_x[i] # 令当前行全部为x_i
        nature_row = np.arange(0, poly_deg + 1) # 按幂由0~poly_deg生成的行向量
        row = row ** nature_row # 计算对应的幂，得到最终行向量
        X[i] = row # 存储到矩阵中
    return X
```

3.3 利用高阶多项式函数拟合曲线(带惩罚项)

算法思想：

不带惩罚项的多项式拟合曲线时，在参数多时w^*具有较大的绝对值，本质就是发生了过拟合。对于这种过拟合，我们可以通过在优化目标函数式中增加w的惩罚项，因此我们得到了式(6)：

$$\widetilde{E}(w)=\frac{1}{2}\sum_{i=1}^{m}\{y(x_i,w)-t_i\}^2+\frac{\lambda}{2}||w||^2 \tag{6}$$

将式(6)写为矩阵形式：

$$\widetilde{E}(w)=\frac{1}{2}[(Xw-T)'(Xw-T)+\lambda w'w] \tag{7}$$

对式(7)求导得：

$$\frac{\partial \tilde{E}}{\partial w} = X'Xw - X'T + \lambda w \tag{8}$$

令式(8)得零得：

$$w^* = (X'X + \lambda I)^{-1}X'T \tag{9}$$

其中I为单位矩阵

对于超参数λ，引入均方误差：

$$E_{RMS} = \sqrt{\frac{2E(w^*)}{N}} \tag{10}$$

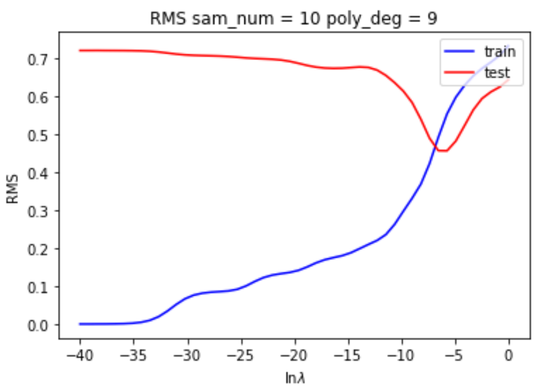
即将式(5)代入式(3)：

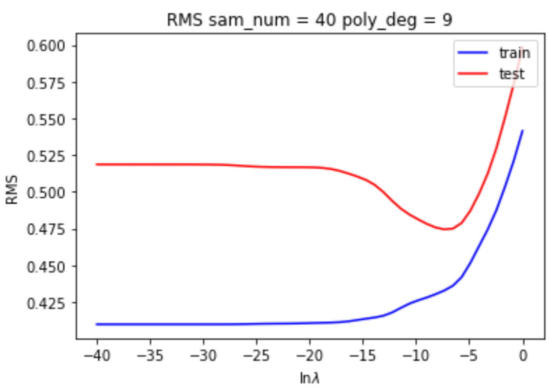
$$E(w^*) = \frac{1}{2}(Xw^* - T)'(Xw^* - T) \tag{11}$$
$$w^* = (X'X + \lambda I)^{-1}X'T$$

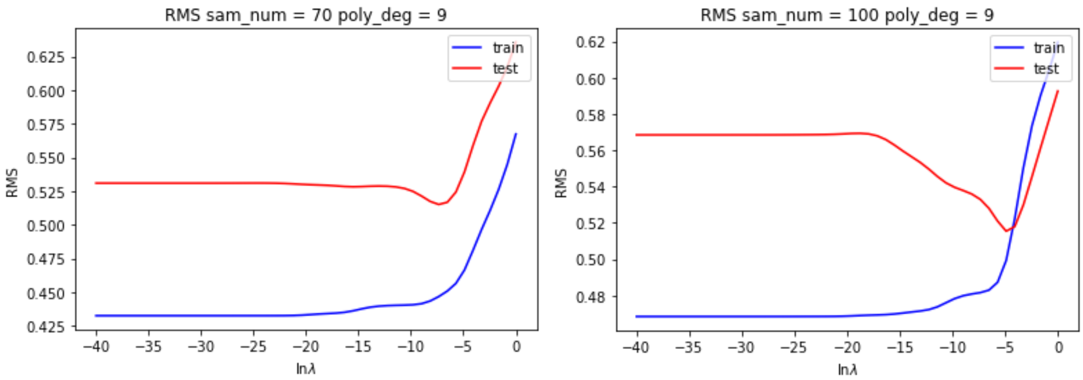
分别生成sam_sum相等的训练样本与测试样本，$ln\lambda$在$[-40,0]$之间等间隔的选取 50 个点，计算对应的w^*_train与E_{RMS}_train，再应用测试样本与w^*_train，计算E_{RMS}_test，可视化对应曲线，具体实现如下：

```python
# 求解最佳惩罚项系数lamda
def cal_lamda():
    train_x, train_y = generate_data(0, 0.5)
    test_x, test_y = generate_data(0, 0.5)
    X_test = trans_martix(test_x)
    Y_test = test_y.reshape(-1, 1)
    X = trans_martix(train_x)
    Y = train_y.reshape(-1, 1)
    RMS_train = np.zeros(50)
    RMS_test = np.zeros(50)
    ln_lamda = np.linspace(-40, 0, 50)
    for i in range(0, 50):
        lamda = np.exp(ln_lamda[i])
        trainRMS, w = cal_RMS(X, Y, lamda)
        RMS_train[i] = trainRMS
        loss2 = loss(X_test, Y_test, w)
        testRMS = math.sqrt(2*loss2/sam_num)
        RMS_test[i] = testRMS
    train_plot = plt.plot(ln_lamda, RMS_train, 'b', label='train')
    test_plot = plt.plot(ln_lamda, RMS_test, 'r', label='test')
    plt.xlabel('$\ln\lambda$')
    plt.ylabel('RMS')
    plt.title('RMS ' + 'sam_num = %d' %sam_num + ' ' + 'poly_deg = %d' %poly_deg )
    plt.legend(loc=1)
    plt.show()
    return
```

改变数据集大小，固定$poly_deg = 9$，结果如下：







易知最优超参数λ在$[e^{-10}, e^{-5}]$之间，最终选取$\lambda = e^{-7}$

具体实现：

```python
# 有惩罚项 lamda/2 * ||w||^2
def fit_withloss(train_x, train_y, lamda):
    X = trans_martix(train_x)
    Y = train_y.reshape(-1, 1)
    #w = np.linalg.inv(np.dot(X.T, X)+lamda).dot(X.T).dot(Y)
    w = np.linalg.inv(np.dot(X.T, X)+lamda * np.eye(X.shape[1])).dot(X.T).dot(Y)
    w = w[::-1].reshape(1, -1).ravel()
    poly = np.poly1d(w)
    return poly
```

3.4 梯度下降法求解最优解

算法思想：

对于$f(x)$如果在点x_i处可微且有定义，易知沿梯度$\nabla f(x_i)$为增长最快的方向，因此逆梯度方向即为下降最快的方向。则下式成立：

$$x_{i+1} = x_i - \alpha \nabla f(x_i) \tag{12}$$

那么对于序列$x_0, x_1, \ldots$有$f(x_0) \geq f(x_1) \geq \cdots$，则可以得到一个顺应$f(x_N)$的最小值，给定一定精度范围即可求解精度范围内的最优解。

补充：

$abs(loss_{k+1} - loss_k) < eps$这一条件并不可以确保收敛，应再加一个判断条件让步长也小于给定精度。

具体实现：

应用带惩罚项 loss 的优化函数进行求解，由式(8)可得：

$$J(w) = (X'X + \lambda I)w - X'T \tag{13}$$

设置$rate$为学习率(即α)， δ 为精度，伪代码如下：

$$\begin{aligned}
&\mathbf{w_0} = \mathbf{0} \\
&\widetilde{E}(\mathbf{w}) = \frac{1}{2N}[(\mathbf{Xw} - \mathbf{T})'(\mathbf{Xw} - \mathbf{T}) + \lambda \mathbf{w}'\mathbf{w}] \\
&J(\mathbf{w}) = (\mathbf{X}'\mathbf{X} + \lambda \mathbf{I})\mathbf{w} - \mathbf{X}'\mathbf{T} \\
&\mathbf{loss_0} = \widetilde{\mathbf{E}}(\mathbf{w_0}) \\
&\mathbf{repeat:} \\
&\qquad \mathbf{w_{k+1}} = \mathbf{w_k} - rate * J(\mathbf{w_k}) \\
&\qquad \mathbf{loss_{k+1}} = \widetilde{\mathbf{E}}(\mathbf{w_{k+1}}) \\
&\qquad \mathbf{if\ \ abs(loss_{k+1} - loss_k) < \delta\ then\ break\ loop} \\
&\qquad k = k + 1 \\
&\mathbf{end\ repeat}
\end{aligned}$$

补充：

```python
if abs(old_loss - new_loss) <= eps and np.linalg.norm(w) <= eps: # 达到精度要求 退出循环
    break
```

具体代码实现不做赘述，见附录，计算损失函数与梯度调用矩阵运算即可实现。

3.5 共轭梯度法求解最优解

虽然梯度下降法的每一步都是朝着局部最优的方向前进的，但是在不同的迭代轮数中会选择非常近似的方向，说明这个方向的误差并没通过一次更新方向和步长更新完，在这个方向上还存在误差，因此参数更新的轨迹是锯齿状。共轭梯度法的思想是，选择一个优化方向后，本次选择的步长能够将这个方向的误差更新完，在以后的优化更新过程中不再需要再在这个方向进行更新. 由于每次将一个方向优化到了极小，后面的优化过程将不再影响之前优化方向上的极小值，所以理论上共轭梯度法将误差分至 m 个互不相关的维度，每一步完全消去一个维度上的误差，以至于可以在 m 步结束，大大提高效率。

定理4.3

设H为n阶对称正定矩阵,$s^0,s^1,s^2,\cdots,s^{n-1}\in\mathbf{R}^n$是一组$H-$共轭方向,对问题(4-2), 若从任一初始点$x^0$出发,依次沿方向$s^0,s^1,s^2,\cdots,s^{n-1}$进行精确一维搜索.则至多经过$n$次迭代, 即可求得$f(x)$的最小点.

共轭梯度法解决的主要是形如$Ax=b$的线性方程组求解问题，其中A必须是对称的、正定的。**概述即为，共轭梯度下降就是在解空间的每一个维度分别取求解最优解的，每一维单独去做的时候不会影响到其他维。**

对于多项式拟合，即求解$(X'X+\lambda)w=X'T$的解析解，记$A=X'X+\lambda$, $b=X'YT$，则问题转为求解$Ax=b$解析解的形式，可以对其应用共轭梯度法进行求解。

对于第k步的残差$r_k=b-Ax_k$，根据该残差构造下一步搜索方向p_k，初始令$p_0=r_0=-\nabla f(x_0)$，然后构造互相共轭的搜索方向，$r_{k+1}=r_k-\alpha_k A p_k$，其中$\alpha_k$如式(12)；进一步根据$r_{k+1}$构造下一个搜索方向$p_{k+1}=r_{k+1}+\beta_{k+1}p_k$，其中$\beta_{k+1}=\frac{r_{k+1}'r_{k+1}}{r_k'r_k}$，得到$x_{k+1}=x_k+\alpha_k p_k$，其中：

$$\alpha_k=\frac{p_k'(b-Ax_k)}{p_k'Ap_k}=\frac{p_k'r_k}{p_k'Ap_k} \tag{14}$$

具体实现：

$$\text{初始令}w_0=\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix},\ A=X'X+\lambda,\ \ b=X'YT，\text{令}p_0=r_0=-\nabla f(x_0)$$

算法主要思想如下：

$$
\begin{aligned}
&\mathbf{p_0} = \mathbf{r_0} \\
&k = 0 \\
&\textbf{repeat} \\
&\qquad \alpha_\mathbf{k} = \frac{\mathbf{r_k^T r_k}}{\mathbf{p_k^T A p_k}} \\
&\qquad \mathbf{w_{k+1}} = \mathbf{w_k} + \alpha_\mathbf{k} \mathbf{p_k} \\
&\qquad \textbf{if } \|\mathbf{r_{k+1}}\| < \delta \textbf{ then break loop} \\
&\qquad \beta_\mathbf{k} = \frac{\mathbf{r_{k+1}^T r_{k+1}}}{\mathbf{r_k^T r_k}} \\
&\qquad \mathbf{p_{k+1}} = \mathbf{r_{k+1}} + \beta_\mathbf{k} \mathbf{p_k} \\
&\qquad k = k + 1 \\
&\textbf{end repeat} \\
&\textbf{The result is } \mathbf{w_{k+1}}
\end{aligned}
$$

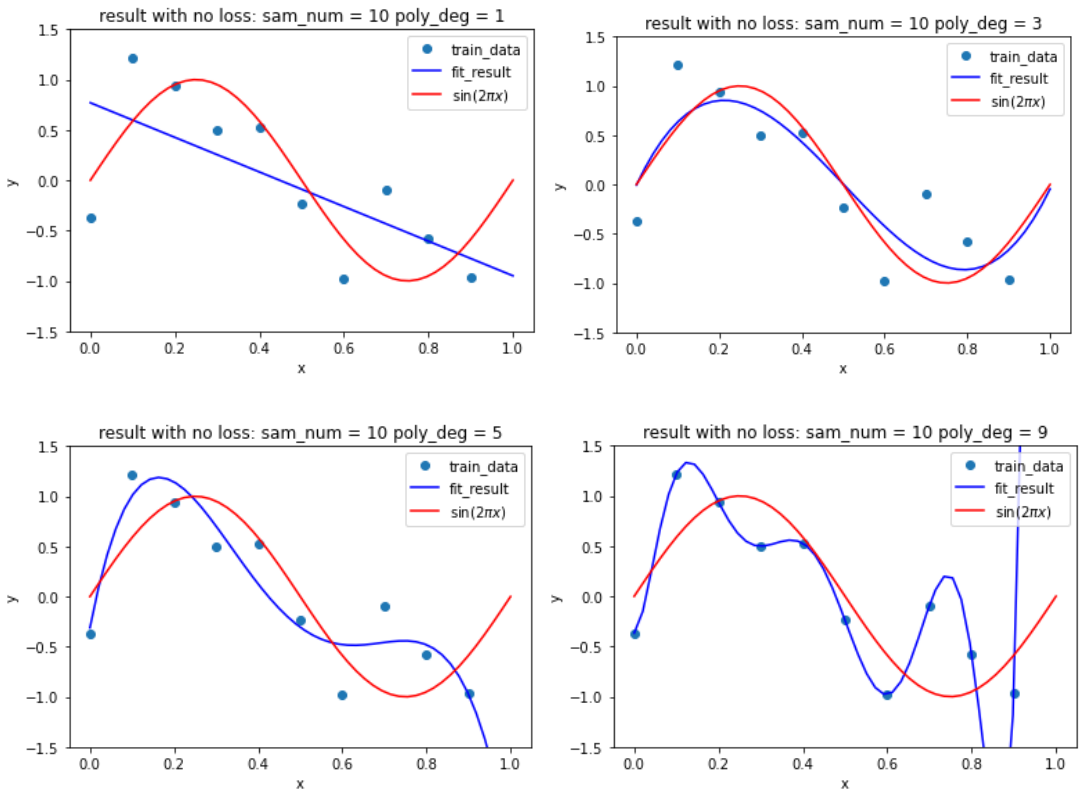
具体代码实现见附录。

四、 实验结果与分析

默认高斯噪声均值$mu = 0$，方差$sigma = 0.5$，其中$poly_deg$对应多项式阶数，sam_sum对应样本容量

4.1 不带惩罚项的解析解

$sam\_sum = 10$，$poly_deg$由 1 增长至 9：（数量过多，只选取部分展示）



可以直观地从图像中看到对于同一训练样本，在多项式阶数为 3 时的拟合效果已较为理想；随着多项式的阶数的提高，在阶数为 9，所得拟合曲线“完

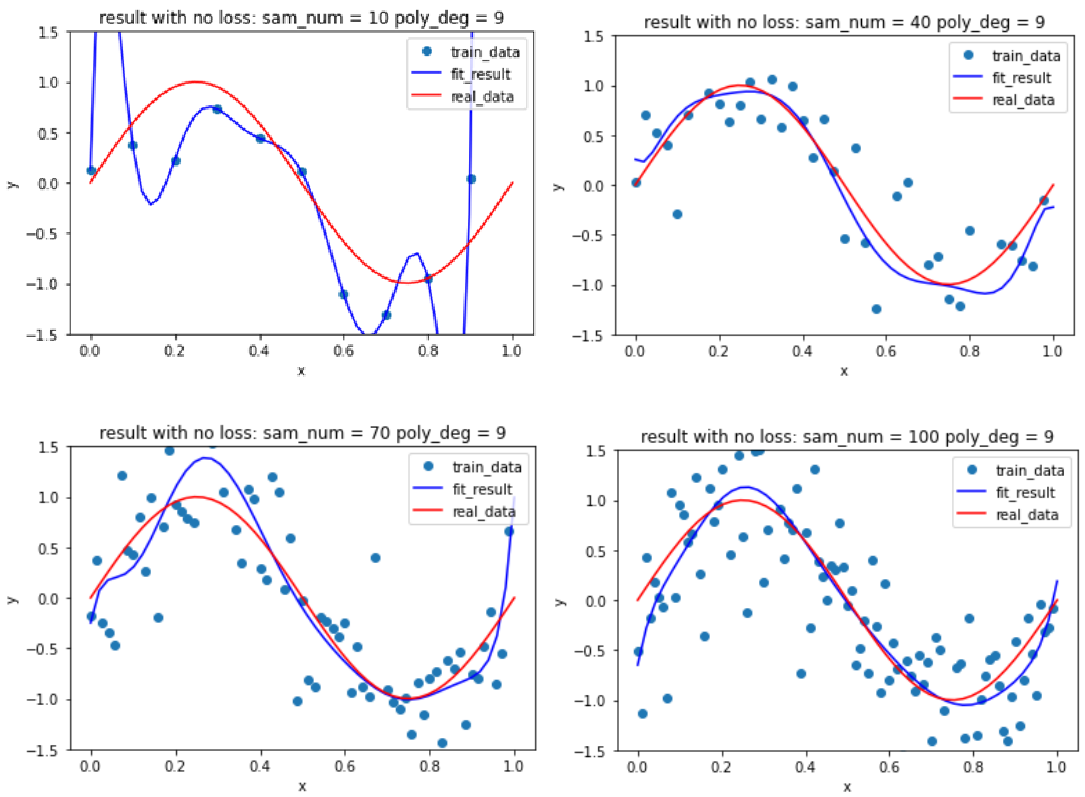
美的”经过了全部样本点，误差为 0，但曲线震荡剧烈，并没有很好的拟合实际的函数$y = \sin(2\pi x)$ ，即表现为过拟合。

出现过拟合的本质原因是，在阶数过大的情况下，模型的复杂度和拟合的能力都增强，可以通过大或者过小的系数来实现震荡以拟合所有的数据点，以至于几乎可以拟合了所有样本点。

在这里由于我们的数据样本大小只有 10，所以在阶数为 9 的时候，其对应的系数向量恰好有唯一解，因此可以穿过所有的样本点。

对于过拟合情况，可以通过**增加样本集容量**或**增加惩罚项**的方式来解决。

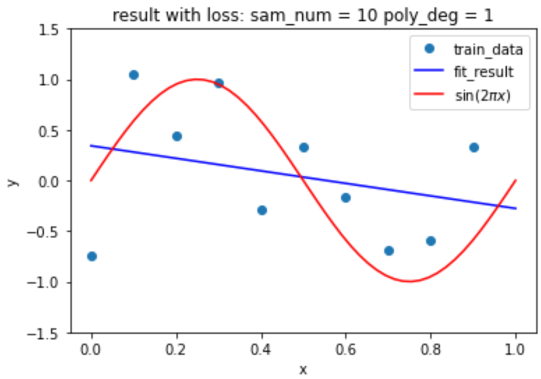
固定阶数$poly_deg = 9$，增加样本容量：

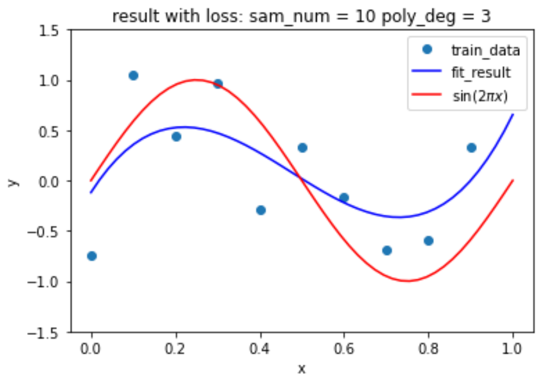


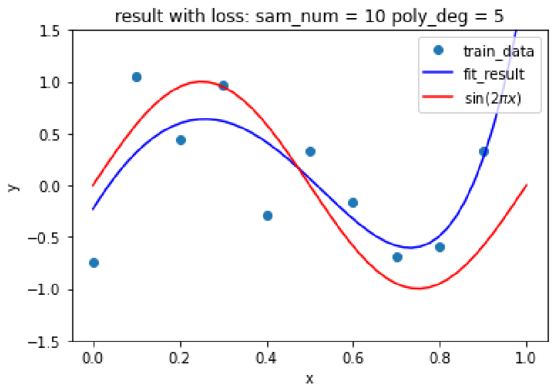
可以看到，随着样本容量增多，过拟合现象有所解决。

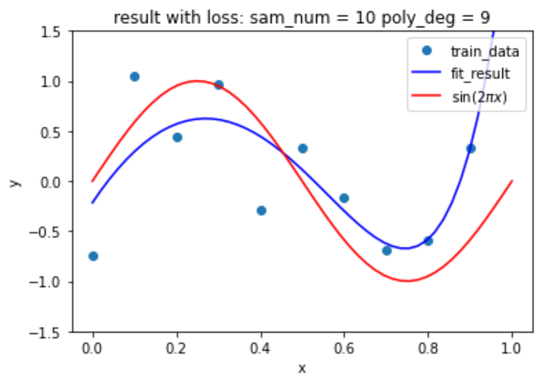
4. 2 带惩罚项的解析解

$sam\_sum = 10$，$poly_deg$逐渐增大，$\lambda = e^{-7}$：







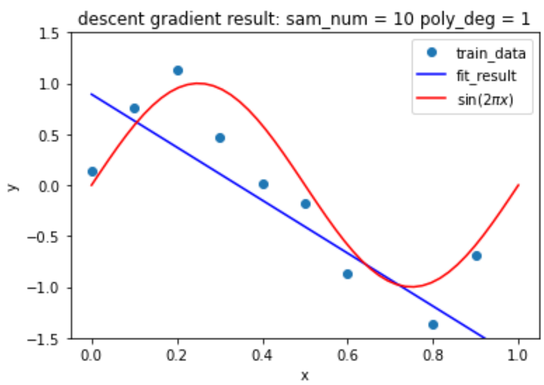


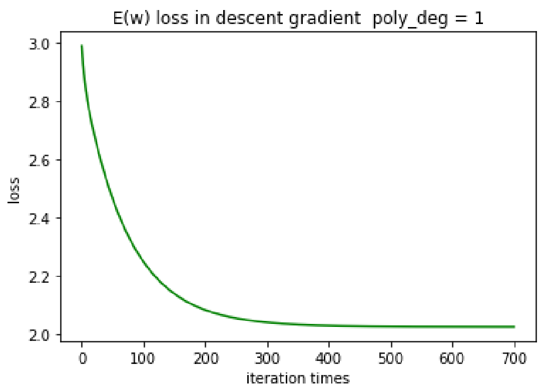
可以看到，对应阶数为 9 时，过拟合情况得到了一定解决，验证了增加惩罚项对于解决过拟合问题的作用。

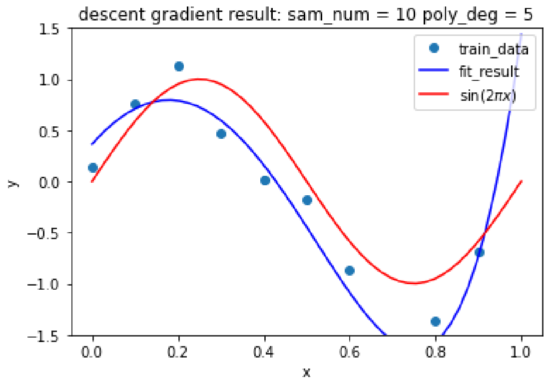
4.3 梯度下降法

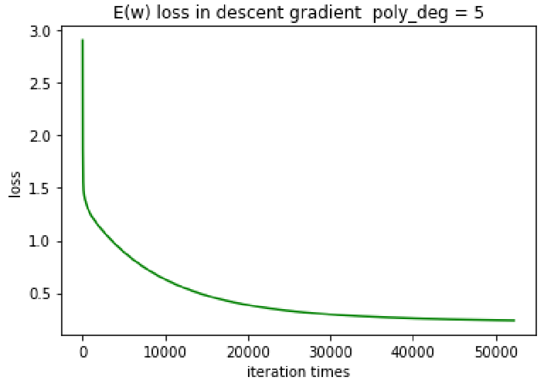
选取学习率$\alpha = 0.01$，精度$eps = 1 \times 10^{-6}$：

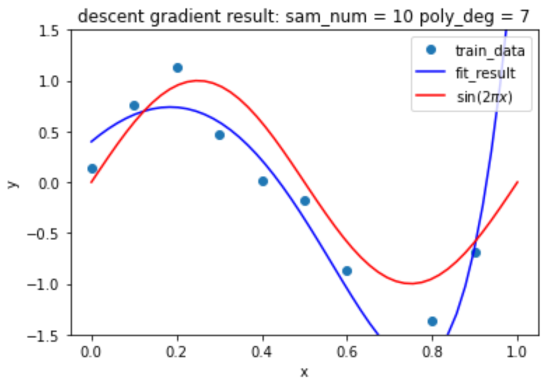
1) $sam\_sum = 10$，$poly_deg$由 1 增长至 9：

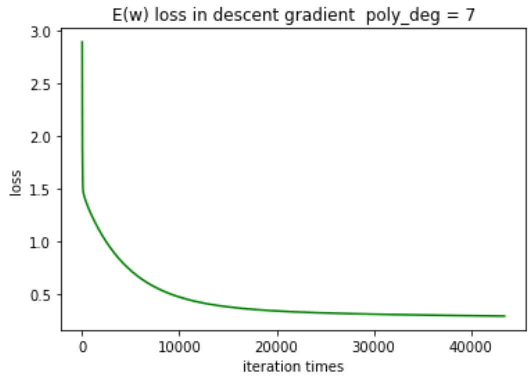


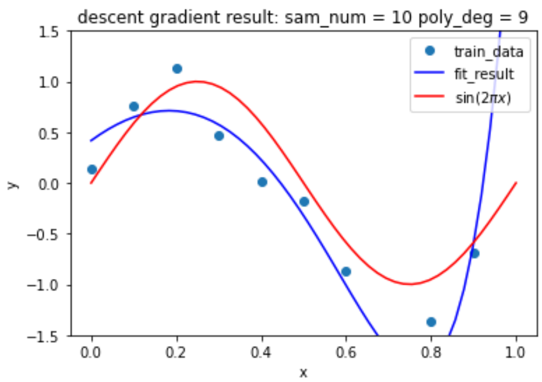


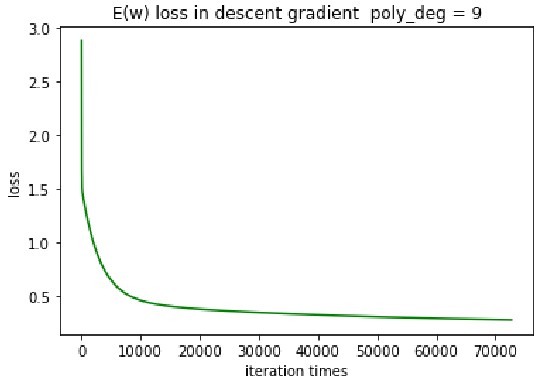




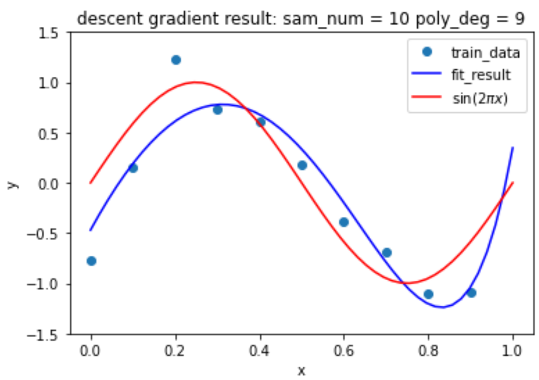


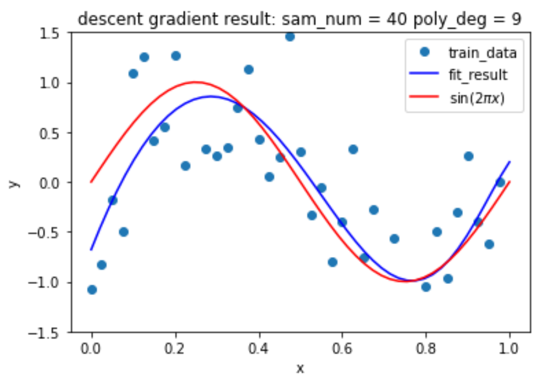


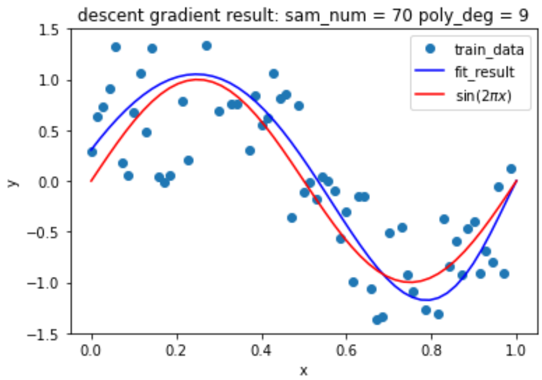


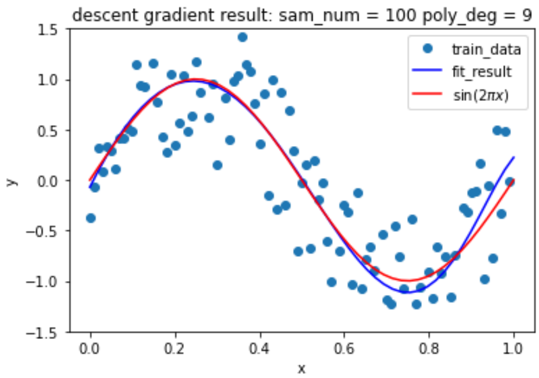


2) $poly\_deg = 9$，增大样本容量：







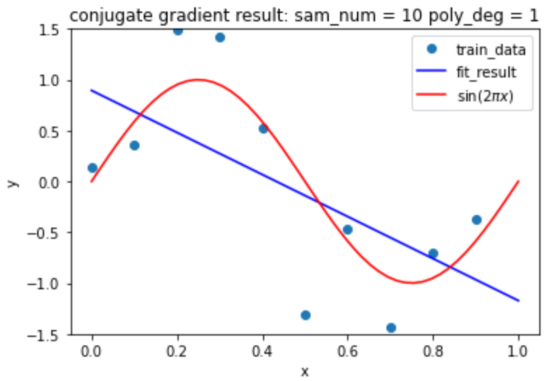


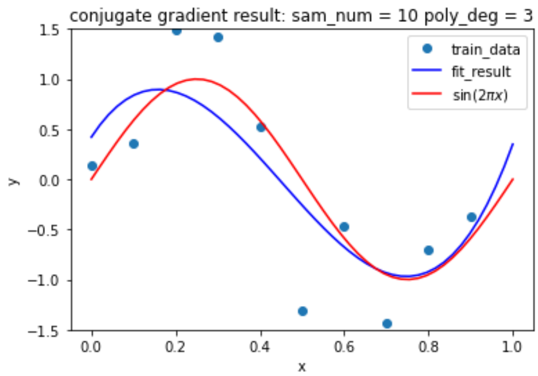
可以看到随着样本容量提升，拟合效果显著提升，当样本集为 100 时，曲线几乎与实际函数$y = \sin(2\pi x)$重合

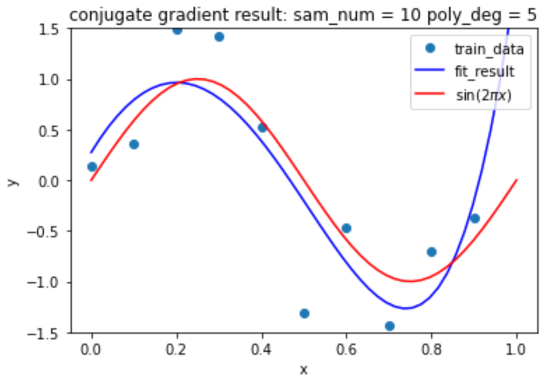
4. 4 共轭梯度法

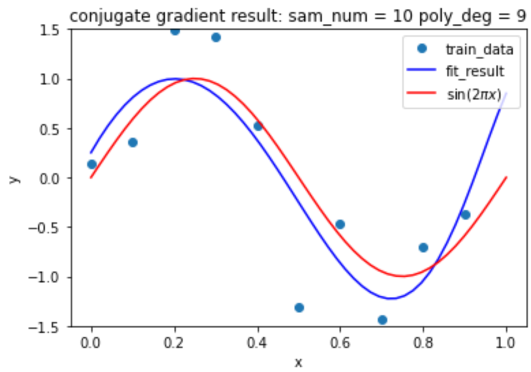
选取惩罚项系数$\lambda = e^{-7}$，精度$eps = 1 \times 10^{-6}$：

1) $sam\_sum = 10$，$poly_deg$由 1 增长至 9：



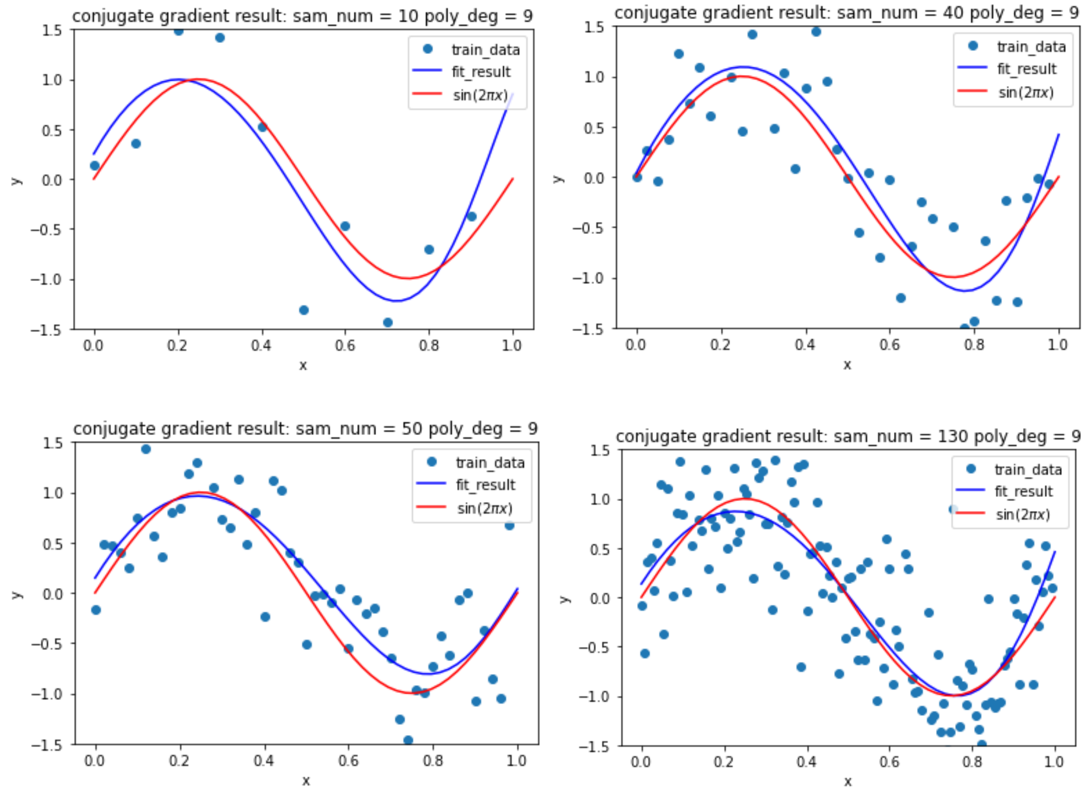




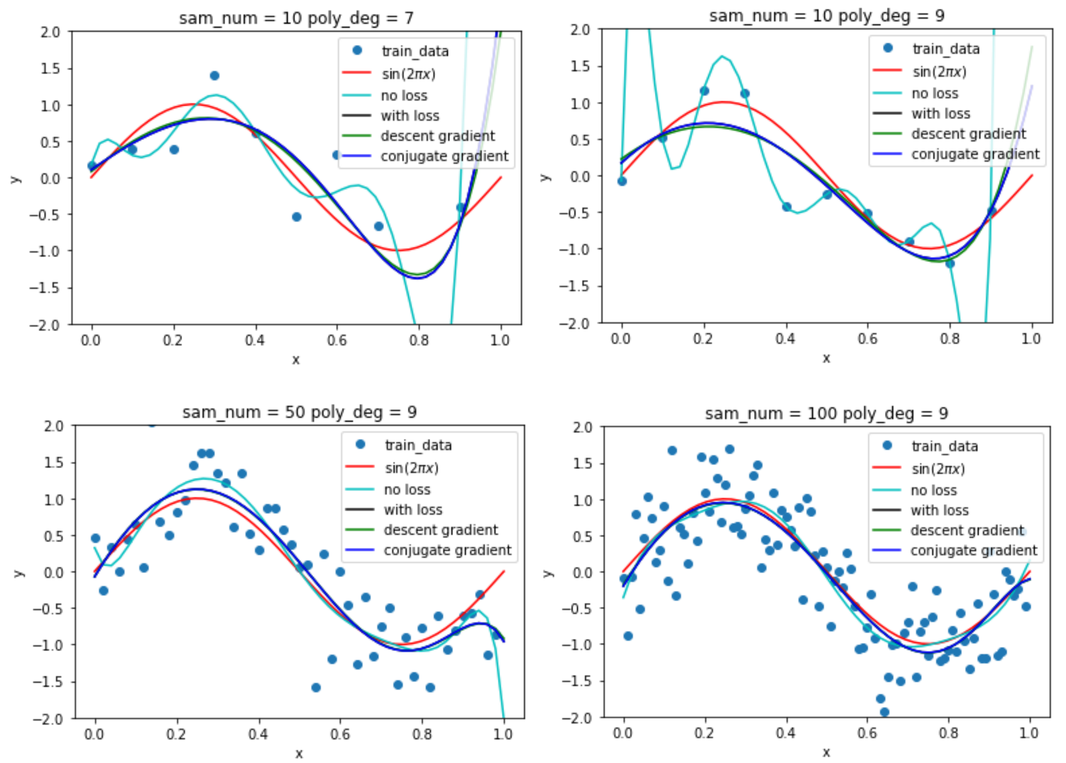


可以看到，共轭梯度法的优化拟合速率很快，且拟合效果在同等阶数下明显优于其他方法。

2) $poly\_deg = 9$，增大样本容量：



4.5 各方法拟合效果对比



易知，在样本容量较小且阶数较高的情况下，无惩罚项的多项式拟合效果容易出现过拟合现象，拟合效果欠佳；梯度下降法与共轭梯度法拟合效果更好；随着样本容量提高，各方法拟合效果均达到理想状态且效果相差不大。

4.6 实验结果分析

在使用多项式对正弦函数$y = \sin(2\pi x)$进行拟合时，根据实验结果易知：多项式的阶数越高，其拟合能力越强。在训练样本数量=10 且不加入惩罚项的情况下，阶数较高的多项式出现了过拟合的现象；在训练样本数量=100 且不加入惩罚项时，阶数 9 相对于 100 来说很低，所以并未出现过拟合的现象。但在训练样本数量=10 且加入惩罚项的情况下，阶数较高的多项式也能较好的解决过拟合现象。这是由于参数增多时，往往具有较大的绝对值，加入正则项可以有效地降低参数的绝对值，从而使模型复杂度与问题匹配。但是当样本足够多时，惩罚项相对于数据就影响不大了。当惩罚项过小时，过拟合现象并没有明显的改善，惩罚项过大时，会出现欠拟合现象。增加数据量可以 避免因为数据量较少而产生的随机误差，避免随机误差被当做数据特征被学习进模型中。

由此可知：**可通过增加训练样本数量及加设惩罚项来解决过拟合问题**；且**按理论预测，当多项式阶数与训练集数据规模几乎一致时会出现过拟合现象。**

根据实验结果易知，随着次数的增加，梯度下降法可能下降过快导致无法收敛，必要时需要要减小学习率。因此在梯度下降法运行的过程中需要动态的调节学习率，如果后一次迭代的 $loss$ 比前一次大，则将学习率减半。当两次迭代的$loss$绝对值之差小于给定精度时，可认为在精度范围内，已找到极值点，停止迭代。

```python
# 共轭梯度法
'''
求解(X^T*X+lamda)w=X^T*Y的解析解，记A=X^T*X+lamda, b=X^T*Y，则问题转为Aw=b的形式，应用共轭梯度法求解
'''
def conjugate_gradient(train_x, train_y, lamda, eps):
    X = trans_martix(train_x)
    Y = train_y.reshape(-1,1)
    A = np.dot(X.T,X)+ lamda * np.eye(X.shape[1])
    b = np.dot(X.T,Y)
    w = np.zeros((X.shape[1],1)) # 初始w为(poly_deg+1)*1的全零列向量
    g = np.dot(X.T, X).dot(w) - np.dot(X.T, Y) + lamda * w # 梯度
    r = -g
    p = r #  初始化s_0=r_0=-g
    for i in range(X.shape[1]): # 理论至多poly_deg+1次搜索即可确定
        temp = np.dot(r.T,r)/(np.dot(p.T,A).dot(p))
        r_old = r # 记录上一个r
        w = w + temp*p
        r = r - (temp*A).dot(p)
        b = np.dot(r.T,r)/np.dot(r_old.T,r_old)
        p = r + b*p #更新搜索方向
        if np.dot(r.T,r) < eps:
            break
    w = w[::-1].reshape(1,-1).ravel()
    poly = np.poly1d(w)
    print('iteration times = ', i) # 打印迭代次数
    return poly
```

在共轭梯度求解过程中添加一行代码打印最终迭代次数，结果如下：

$poly\_deg = 9$，$sam\_num = 10$ 时： `iteration times =  4`

$poly\_deg = 9$，$sam\_num = 100$ 时： `iteration times =  6`

易知，**共轭梯度法经过有限次的迭代即找到极小值**，相对于梯度下降法，共轭梯度法优化速度更快，由相应方法对应拟合效果图中，也可以直观地发现**共轭梯度法相较于梯度下降法拟合效果更好**。

五、 结论

- ✧ 增加训练样本规模/加设惩罚项可以有效的解决过拟合问题
- ✧ 对于训练样本限制较多的问题，增加惩罚项仍然可以有效解决过拟合问题
- ✧ 共轭梯度法相对于梯度下降法优化速度更快，且拟合效果更好
- ✧ 在梯度下降法中，如果学习率过小，迭代次数就会增加，迭代时间变长；学习率过大，可能无法收敛甚至得到错误结果
- ✧ 梯度下降过程中，需要判断下降的快慢动态减小学习率，防止其过大或过小

六、 参考文献

[1] 李航, 《统计学习方法》（第三版）

[2] 周志华, 《机器学习》

[3] Pattern Recognition and Machine Learning.

[4] Gradient descent wiki

[5] Conjugate gradient method wiki

七、 附录：源代码（带注释）

```
import numpy as np # 矩阵操作库函数
import math
import matplotlib.pyplot as plt # 绘图可视化
global sam_num
global poly_deg
# 生成训练集数据 为sin2pix 添加mu 为均值，sigma 为标准差的高斯噪声
def generate_data(mu, sigma):
    train_x = np.arange(0, 1, 1/sam_num) #start,end,步长
    guass_noise = np.random.normal(mu, sigma, sam_num)
    train_y = np.sin(2 * np.pi * train_x) + guass_noise
    return train_x, train_y
# 将向量存储为samp_num*(poly_deg+1)的矩阵中
'''
w 为 poly_deg 阶多项式中 poly_deg+1 个系数按幂由低到高组成的列向量
对应得到的矩阵 X，设 X_j 为矩阵第 j 行向量，则 X_j 为（...,w_k*第 j 个 train_x^k,...）k 从 0 递增至 poly_deg
易知矩阵 X 为 samp_num*(poly_deg+1)阶
'''
def trans_martix(train_x):
    X = np.zeros((sam_num, poly_deg + 1)) # 初始化矩阵为0
    for i in range(sam_num):
        row = np.ones(poly_deg + 1) * train_x[i] # 令当前行全部为 x_i
```

```python
        nature_row = np.arange(0, poly_deg + 1) # 按幂由0~poly_deg 生成的行向量
        row = row ** nature_row # 计算对应的幂，得到最终行向量
        X[i] = row # 存储到矩阵中
    return X
# 无惩罚性项
def fit_noloss(train_x, train_y):
    X = trans_martix(train_x)
    Y = train_y.reshape(-1, 1) # 将train_y 转为列向量
    w = np.linalg.pinv(X).dot(Y) # w 为各系数按幂由低至高排列构成的列向量
    w = w[::-1].reshape(1,-1).ravel() # 转为按降幂顺序排列的一维数组
    poly = np.poly1d(w) # 对应的多项式结果
    return poly
# 计算损失函数 E(w) = 1/2 * (Xw - Y)^{T} . (Xw - Y)
def loss(X, Y, w):
#     X = trans_martix(train_x)
#     Y = train_y.reshape(-1,1)
    M = X.dot(w)-Y
    loss = np.dot(M.T,M)/2
    return loss
# 计算均方误差
def cal_RMS(X, Y, lamda):
    w = np.linalg.inv(np.dot(X.T,X)+lamda * np.eye(X.shape[1])).dot(X.T).dot(Y)
    loss1 = loss(X,Y,w)
    RMS = math.sqrt(2*loss1/sam_num)
    return RMS,w
# 求解最佳惩罚项系数 lamda
def cal_lamda():
    train_x,train_y = generate_data(0, 0.5)
    test_x,test_y = generate_data(0, 0.5)
    X_test = trans_martix(test_x)
    Y_test = test_y.reshape(-1,1)
    X = trans_martix(train_x)
    Y = train_y.reshape(-1, 1)
    RMS_train = np.zeros(50)
    RMS_test = np.zeros(50)
    ln_lamda = np.linspace(-40,0,50)
    for i in range(0,50):
        lamda = np.exp(ln_lamda[i])
        trainRMS,w = cal_RMS(X,Y,lamda)
        RMS_train[i] = trainRMS
        loss2 = loss(X_test,Y_test,w)
```

```python
        testRMS = math.sqrt(2*loss2/sam_num)
        RMS_test[i] = testRMS
    train_plot = plt.plot(ln_lamda, RMS_train, 'b', label='train')
    test_plot = plt.plot(ln_lamda, RMS_test, 'r', label='test')
    plt.xlabel('$\ln\lambda$')
    plt.ylabel('RMS')
    plt.title('RMS ' + 'sam_num = %d' %sam_num + ' ' + 'poly_deg = %d' %poly_deg )
    plt.legend(loc=1)
    plt.show()
    return
# 改变数据集大小，控制阶数不变，求解最优 lamda
# sam_num = 10
# poly_deg = 9
# cal_lamda()
# 有惩罚项 lamda/2 * ||w||^2
def fit_withloss(train_x, train_y, lamda):
    X = trans_martix(train_x)
    Y = train_y.reshape(-1,1)
    #w = np.linalg.inv(np.dot(X.T,X)+lamda).dot(X.T).dot(Y)
    w = np.linalg.inv(np.dot(X.T,X)+lamda * np.eye(X.shape[1])).dot(X.T).dot(Y)
    w = w[::-1].reshape(1,-1).ravel()
    poly = np.poly1d(w)
    return poly
# 计算梯度
def gradient(X, Y, w, lamda):
    gra = np.dot(X.T, X).dot(w) - np.dot(X.T, Y) + lamda * w
    return gra
# 梯度下降法 lamda 为系数，alpha 为学习率，精度为 eps， 至多循环 n 次
def descent_gradient(train_x, train_y, lamda, alpha, eps, n):
    X = trans_martix(train_x)
    Y = train_y.reshape(-1,1)
    w = np.zeros((X.shape[1],1)) # 初始 w 为(poly_deg+1)*1 的全零列向量
    new_loss = abs(loss(X, Y, w))
    n_list = []
    loss_list = []
    for i in range(n):
        old_loss = new_loss # 记录原 loss
        old_w = w # 记录原 w
        w -= alpha * gradient(X,Y,w,lamda) # 更新 w
        new_loss = loss(X,Y,w) # 更新 loss
```

```python
        n_list.append(i) # 记录迭代次数
        loss_list.append(float(new_loss)) # 记录 loss
        if old_loss < new_loss: # 下降过快
            alpha /= 2 # 将学习率减半
            w = old_w # 更正 w
        if abs(old_loss - new_loss) <= eps and np.linalg.norm(w) <= eps: # 达到精度要求 退出循环
            break
    w = w[::-1].reshape(1,-1).ravel()
    poly = np.poly1d(w)
    return poly, loss_list, n_list
# 共轭梯度法
'''
求解(X^T*X+lamda)w=X^T*Y 的解析解，记 A=X^T*X+lamda, b=X^T*Y，则问题转为 Aw=b 的形式，应用共轭梯度法求解
'''
def conjugate_gradient(train_x, train_y, lamda, eps):
    X = trans_martix(train_x)
    Y = train_y.reshape(-1,1)
    A = np.dot(X.T,X)+ lamda * np.eye(X.shape[1])
    b = np.dot(X.T,Y)
    w = np.zeros((X.shape[1],1)) # 初始 w 为(poly_deg+1)*1 的全零列向量
    g = np.dot(X.T, X).dot(w) - np.dot(X.T, Y) + lamda * w # 梯度
    r = -g
    p = r #  初始化 s_0=r_0=-g
    for i in range(X.shape[1]): # 理论至多 poly_deg+1 次搜索即可确定
        temp = np.dot(r.T,r)/(np.dot(p.T,A).dot(p))
        r_old = r # 记录上一个 r
        w = w + temp*p
        r = r - (temp*A).dot(p)
        b = np.dot(r.T,r)/np.dot(r_old.T,r_old)
        p = r + b*p #更新搜索方向
        if np.dot(r.T,r) < eps:
            break
    w = w[::-1].reshape(1,-1).ravel()
    poly = np.poly1d(w)
#     print('iteration times = ', i) # 打印迭代次数
    return poly
# 拟合效果可视化
def plt_show(train_x, train_y, poly_fit, title):
    plot1 = plt.plot(train_x, train_y, 'o',label='train_data')
    real_x = np.linspace(0, 1, 50)
```

```
    real_y = np.sin(real_x * 2 * np.pi)
    fit_y = poly_fit(real_x)
    plot2 = plt.plot(real_x, fit_y, 'b', label='fit_result')
    plot3 = plt.plot(real_x, real_y, 'r', label='$\sin(2\pi x)$')
    plt.ylim(-1.5,1.5)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(loc=1)
    plt.title(title + 'sam_num = %d' %sam_num + ' ' + 'poly_deg = %d' %poly_deg)
    plt.show()
'''各方法拟合效果对比'''
'''样本集为 10，阶数增大'''
def compare_show(plt_noloss, plt_withloss, plt_degra, plt_cog):
    train_data = plt.plot(train_x, train_y, 'o',label = 'train_data')
    real_x = np.linspace(0, 1, 50)
    real_y = np.sin(real_x * 2 * np.pi)
    plt_true = plt.plot(real_x, real_y, 'r', label = '$\sin(2\pi x)$')

    y_noloss = plt_noloss(real_x)
    plt1 = plt.plot(real_x, y_noloss, 'c', label = 'no loss')

    y_withloss = plt_withloss(real_x)
    plt2 = plt.plot(real_x, y_withloss, 'k', label = 'with loss')

    y_degra = plt_degra(real_x)
    plt3 = plt.plot(real_x, y_degra, 'g', label = 'descent gradient')

    y_cog = plt_cog(real_x)
    plt4 = plt.plot(real_x, y_cog, 'b', label = 'conjugate gradient')

    plt.ylim(-2,2)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(loc=1)
    plt.title('sam_num = %d' %sam_num + ' ' + 'poly_deg = %d' %poly_deg)
    plt.show()
```

```
# 一般情况下默认样本集为10，阶数为9
sam_num = 10
poly_deg = 9

mu = 0
sigma = 0.5
lamda = np.exp(-7)
train_x, train_y = generate_data(mu, sigma) # 获取数据

'''无惩罚项拟合结果'''
'''样本集为 10，阶数逐渐增大'''
# for i in range(0,9,2):
#     poly_deg = i + 1
#     plt_noloss = fit_noloss(train_x, train_y)
#     plt_show(train_x, train_y, plt_noloss, 'result with no loss: ')
#     print(plt_noloss)
'''poly_deg=9，增多样本容量，无惩罚项过拟合分析'''
# for i in range(3):
#     sam_num +=30
#     train_x, train_y = generate_data(mu, sigma)
#     poly_deg = 9
#     plt_noloss = fit_noloss(train_x, train_y)
#     plt_show(train_x, train_y, plt_noloss, 'result with no loss: ')
#     print(plt_noloss)

'''有惩罚项，惩罚项系数 lamda = e^-7'''
# for i in range(0,9,2):
#     poly_deg = i + 1
#     plt_withloss = fit_withloss(train_x, train_y, lamda)
#     plt_show(train_x, train_y, plt_withloss, 'result with loss: ')
#     print(plt_withloss)

alpha = 0.01
eps = 1e-6
n = 100000
'''梯度下降法：学习率 alpha=0.01，精度 eps=1e-6，惩罚项系数 lamda = e^-7，迭代次数至多为 100000'''
'''样本集为 10，阶数逐渐增大'''
# for i in range(0,9,2):
#     poly_deg = i + 1
```

```
#     plt_degra, loss_list, n_list = descent_gradient(train_x, train_y, lamda, alpha, eps, n)
#     plt_show(train_x, train_y, plt_degra, 'descent gradient result: ')
#     print(plt_degra)
#     plt.plot(n_list, loss_list, 'g')
#     plt.xlabel('iteration times')
#     plt.ylabel('loss')
#     plt.title('E(w) loss in descent gradient ' + ' poly_deg = %d' %poly_deg)
#     plt.show()
'''固定阶数为 9，样本集增多大'''
# for i in range(3):
#     sam_num +=30
#     train_x, train_y = generate_data(mu, sigma)
#     poly_deg = 9
#     plt_degra = descent_gradient(train_x, train_y, lamda, alpha, eps, n)[0]
#     plt_show(train_x, train_y, plt_degra, 'descent gradient result: ')
#     print(plt_degra)

'''共轭梯度法：精度 eps=1e-6，惩罚项系数 lamda = e^-7'''
'''样本集为 10，阶数增大'''
# for i in range(0,9,2):
#     poly_deg = i + 1
#     plt_cog = conjugate_gradient(train_x, train_y, lamda, eps)
#     plt_show(train_x, train_y, plt_cog, 'conjugate gradient result: ')
#     print(plt_cog)
'''固定阶数为 9，增大样本集'''
# for i in range(3):
#     sam_num += 30
#     train_x, train_y = generate_data(mu, sigma)
#     poly_deg = 9
#     plt_cog = conjugate_gradient(train_x, train_y, lamda, eps)
#     plt_show(train_x, train_y, plt_cog, 'conjugate gradient result: ')
#     print(plt_cog)

'''拟合效果对比'''
plt1 = fit_noloss(train_x, train_y)
plt2 = fit_withloss(train_x, train_y,lamda)
```

```
plt3 = descent_gradient(train_x, train_y, lamda, alpha, eps, n)[0]
plt4 = conjugate_gradient(train_x, train_y, lamda, eps)
compare_show(plt1,plt2,plt3,plt4)
```