哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2020 年春季学期

计算学部《机器学习》课程

Lab 1 实验报告

姓名	梅智敏
学号	1183710118
班号	1837101
电子邮件	1044388658@qq.com
手机号码	13385658102

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

- 生成数据，加入噪声；
- 用高阶多项式函数拟合曲线；
- 用解析解求解两种loss的最优解（无正则项和有正则项）
- 优化方法求解最优解（梯度下降，共轭梯度）；
- 用你得到的实验数据，解释过拟合。
- 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。
- 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

- Windows 10 专业教育版
- python 3.7.4
- jupyter notebook 6.0.1
- pycharm 2020.2.3

三、设计思想(本程序中的用到的主要算法及数据结构)

数学原理

最小二乘法求解析解（无惩罚项）

根据**泰勒级数**，足够高阶的多项式可以拟合任意函数$f:X\to Y$。对于我们的加入$sin(2\pi x)$来说，完全可以用多项式函数来拟合，已知训练集有N个样本点$(x_1,t_1),(x_2,t_2)\dots(x_N,t_N)$。

m阶多项式函数为

$$y(x,\boldsymbol{w})=\sum_{i=0}^{m}w_ix^i$$

使用**最小二乘法**的代价函数为

$$E(\boldsymbol{w})=\frac{1}{2}\sum_{n=1}^{N}\{y\left(x_n,\boldsymbol{w}\right)-t_n\}^2$$

其中多项式的系数矩阵$\boldsymbol{w}$如下

$$\boldsymbol{w}=\begin{pmatrix} w_0 \\ w_1 \\ \cdots \\ w_m \end{pmatrix}$$

现在令

$$\boldsymbol{T}=\begin{pmatrix} t_1 \\ t_2 \\ \cdots \\ t_N \end{pmatrix}$$

$$\boldsymbol{X}=\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ & & \cdots\cdots & & \\ 1 & x_N & x_N^2 & \cdots & x_N^m \end{pmatrix}$$

则可以将代价函数$E(\boldsymbol{w})$写成矩阵形式

$$E(\boldsymbol{w})=\frac{1}{2}(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})^T(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T}) \tag{1}$$

我们进行多项式拟合的目的就是求出$\boldsymbol{w}$,能够使得$E(\boldsymbol{w})$最小，那么我们可以对(1)式求导，从而求出解析解

(1)式化简后为

$$E(\boldsymbol{w})=\frac{1}{2}(\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}-2\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{T}+\boldsymbol{T}^T\boldsymbol{T})$$

对$\boldsymbol{w}$求导

$$\frac{\partial E}{\partial \boldsymbol{w}}=\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}-\boldsymbol{X}^T\boldsymbol{T} \tag{2}$$

令导数为0，则

$$\boldsymbol{w}^*=(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{T}=\boldsymbol{X}^{-1}\boldsymbol{T} \tag{3}$$

最小二乘法求解析解（有惩罚项）

在前面的求解过程中，没有加惩罚项，此时随着阶数m的增大，$\boldsymbol{w}^*$往往具有较大的绝对值，从而赋予多项式函数$E(\boldsymbol{w})$更强的变化能力。往往会为了更加贴合训练集中样本点，使得$\boldsymbol{w}$各维数值绝对值很大、很复杂，因为它的学习地太“过火”了，将一些不属于训练集的特征（如：噪声的影响）都学习到了，**其本质上就是发生了过拟合。于是，我们可以增加惩罚项，迫使$\boldsymbol{w}^*$的绝对值没有那么大。**

$$\widetilde{E}(\boldsymbol{w})=\frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\boldsymbol{w})-t_n\}^2+\frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

写成矩阵形式

$$\widetilde{E}(\boldsymbol{w})=\frac{1}{2}[(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})^T(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})+\lambda\boldsymbol{w}^T\boldsymbol{w}] \tag{4}$$

同前面一样，对$\boldsymbol{w}$求导

$$\frac{\partial \widetilde{E}}{\partial \boldsymbol{w}} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{T} + \lambda\boldsymbol{w} \tag{5}$$

令导数为0，求得

$$\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{T} \tag{6}$$

上式中的I表示单位阵

由于我们**引入了超参数λ来控制惩罚项的重要性**，我们就需要确定它的值。

我们可以将(4)式视为以λ和$\boldsymbol{w}$为变量的函数$\widetilde{E}(\boldsymbol{w},\lambda)$，再通过实验，手动给$\lambda$确定一个合适的范围，让$\lambda$在这个范围内取一些值，分别求出对应的解析解$\boldsymbol{w}^*$后，代回$\widetilde{E}(\boldsymbol{w},\lambda)$，选取最小值所对应的$\lambda$即可

$$\lambda = \mathrm{argmin}_{\lambda}\ \widetilde{E}(\boldsymbol{w},\lambda) \tag{7}$$

梯度下降法求优化解

梯度实际上就是多变量微分的一般化。例如，$\theta_1,\theta_2,\theta_3\ldots\theta_N$是$N$个变量，$\Theta = \begin{pmatrix}\theta_1\\ \theta_2\\ \cdots\\ \theta_N\end{pmatrix}$以矩阵形式表示这些变量，$J(\Theta)$为多元函数。

则梯度为

$$\nabla J(\Theta) = (\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \frac{\partial J}{\partial \theta_3}\cdots\frac{\partial J}{\partial \theta_N})$$

至于梯度的意义，即

- 在单变量的函数中，梯度其实就是函数的微分，代表着函数在某个给定点的切线的斜率
- 在多变量函数中，梯度是一个向量，向量有方向，**梯度的方向就指出了函数在给定点的上升最快的方向；对应的，梯度的反方向就是给定点的下降最快的方向。**

在我们的这个问题中，代价函数

$$\widetilde{E}(\boldsymbol{w}) = \frac{1}{2}[(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{T})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{T}) + \lambda\boldsymbol{w}^T\boldsymbol{w}]$$

它就是一个关于$\boldsymbol{w}$的多元函数，我们的目的就是求出$\boldsymbol{w}$使得$\widetilde{E}(\boldsymbol{w})$取最小值。那么，给定一个初始点$\boldsymbol{w}_0$，理论上**我们只需要不断地沿着当前点的梯度反方向走合适的距离，如此进行迭代，便可逐渐靠近最小值点$\boldsymbol{w}^*$**，这就是所谓的梯度下降法的核心思想。

核心的数学迭代式如下

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha\nabla\widetilde{E}(\boldsymbol{w}_i) \tag{8}$$

- 符号解释：$\boldsymbol{w}_i$为当前点，$\boldsymbol{w}_{i+1}$为下一次将要到达的点，α为学习率（步长）
- 此公式困难之处在于α的确定：过小则会造成迭代次数过多，靠近$\boldsymbol{w}^*$的速度非常慢；过大则可能导致距离$\boldsymbol{w}^*$越来越远，无法找到$\boldsymbol{w}^*$

但是我们可以通过实验来手动调参，选择一个合适的α，这样的话，就可以得到一个$\boldsymbol{w}$序列

$$\boldsymbol{w}_0,\boldsymbol{w}_1,\boldsymbol{w}_2,\boldsymbol{w}_3\ldots$$

使得

$$\widetilde{E}(\boldsymbol{w}_0) > \widetilde{E}(\boldsymbol{w}_1) > \widetilde{E}(\boldsymbol{w}_2) > \widetilde{E}(\boldsymbol{w}_3) > \ldots$$

因此，我们可以得到一个收敛到期望的最小值$\widetilde{E}(\boldsymbol{w}_i)$，其对应的$\boldsymbol{w}_i$即为所求。

共轭梯度法求优化解

我们可以将代价函数

$$\widetilde{E}(\boldsymbol{w})=\frac{1}{2}[(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})^T(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})+\lambda\boldsymbol{w}^T\boldsymbol{w}]$$

写成二次型

$$\widetilde{E}(\boldsymbol{w})=\frac{1}{2}\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w}-\boldsymbol{b}\boldsymbol{x}^T+c \tag{9}$$

不难求得

$$\frac{\partial\widetilde{E}}{\partial\boldsymbol{w}}=\boldsymbol{A}\boldsymbol{w}-\boldsymbol{b}$$

即

$$\nabla\widetilde{E}(\boldsymbol{w})=\boldsymbol{A}\boldsymbol{w}-\boldsymbol{b} \tag{10}$$

而根据前面的(5)式我们可知

$$\boldsymbol{A}=\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I} \tag{11}$$

$$\boldsymbol{b}=\boldsymbol{X}^T\boldsymbol{T} \tag{12}$$

我们的目的是求$\widetilde{E}(\boldsymbol{w})$最小值，令导数为0，问题转化为求解线性方程组

$$\begin{aligned}\boldsymbol{A}\boldsymbol{w}-\boldsymbol{b}&=0\\\boldsymbol{A}\boldsymbol{w}&=\boldsymbol{b}\end{aligned} \tag{13}$$

- tips：$\boldsymbol{A}$是对称、正定的

而所谓共轭梯度下降，就是为了求解(13)式

- 核心思想为：**在解空间的每一个维度分别取求解最优解，每一维单独去做的时候不会影响到其他维，它的最大的优势就是每个方向都走到了极致，也即是说寻找极值的过程中绝不走曾经走过的方向,那么n维空间的函数极值也就走n步即可**
- 与梯度下降方法区别：梯度下降法每次都选择梯度的反方向去迭代，这不能保障每次在每个维度上都是靠近最优解的，这就是共轭梯度优于梯度下降的原因

具体求解过程如下

首先记$\boldsymbol{w}^*$为真正的解，$\boldsymbol{w}_t$为当前点，$\boldsymbol{r}_t$为**当前优化方向**，α_t为当前步长，$\boldsymbol{A}$为前面的对阵正定矩阵

定义误差向量

$$\boldsymbol{e}_t=\boldsymbol{w}^*-\boldsymbol{w}_t \tag{14}$$

若当前的误差跟上一步的优化方向正交，便意味着这个方向再也不需要走了，我们记每一次的优化方向为$\boldsymbol{r}_i$，即

$$\boldsymbol{r}_{t-1}^T\boldsymbol{e}_t=0$$

但是$\boldsymbol{e}_t$是未知的，因此我们转而使用共轭正交

$$\boldsymbol{r}_{t-1}^T \boldsymbol{A} \boldsymbol{e}_t = 0 \tag{15}$$

又

$$\boldsymbol{w}_{t+1} - \boldsymbol{w}_t = \alpha_t \boldsymbol{r}_t \tag{16}$$

利用(14)、(15)、和(16)式，可求得

$$\begin{aligned} \alpha_t &= \frac{\boldsymbol{r}_t^T \boldsymbol{A} \boldsymbol{e}_t}{\boldsymbol{r}_t^T \boldsymbol{A} \boldsymbol{r}_t} \\ &= \frac{\boldsymbol{r}_t^T \boldsymbol{A} \left(\boldsymbol{w}^* - \boldsymbol{w}_t \right)}{\boldsymbol{r}_t^T \boldsymbol{A} \boldsymbol{r}_t} \\ &= -\frac{\boldsymbol{r}_t^T \nabla \widetilde{E}(\boldsymbol{w})}{\boldsymbol{r}_t^T \boldsymbol{A} \boldsymbol{r}_t} \end{aligned} \tag{17}$$

可见，我们通过引入$\boldsymbol{A}$使得消除了前面的未知误差$\boldsymbol{e}_t$，使得步长α_t变得可解

下面需要确定$\boldsymbol{r}_t$，我们按照**施密特正交化**的原理去推导出共轭正交的一组向量

$$\boldsymbol{r}_t = -\nabla \widetilde{E}(\boldsymbol{w}) + \sum_{i<t} \frac{\boldsymbol{r}_i^T \boldsymbol{A} \nabla \widetilde{E}(\boldsymbol{w})}{\boldsymbol{r}_i^T \boldsymbol{A} \boldsymbol{r}_i} \boldsymbol{r}_i \tag{18}$$

- 上面的式子其实就是每一步的方向都是在起点的负梯度方向的基础上做修改, 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.

所以我们通过(18)式求解出$\boldsymbol{r}_t$， 再代入（17）式便可求出α_t，这样我们便可同前面的梯度下降法一样，利用迭代手段求解$\boldsymbol{w}$

算法实现

生成数据集并加入噪声

主要利用python中的$numpy$包提供的强大矩阵运算功能帮助我们进行计算，噪声满足均值为0，方差为$sigma$的正态分布

```
def generateData(sigma, N):
    """
    Generate DataSet with noise
    Args:
        sigma: the variance of the noise
        N: the number of the Training-set
    """
    assert sigma > 0
    assert N > 0
    assert isinstance(N, int)

    X = np.linspace(0, 1, num=N)
    noise = np.random.normal(0, scale=sigma, size=X.shape)
    T = np.sin(2 * np.pi * X) + noise
    return X, T
```

求解析解（有、无正则项）

也是利用python中的$numpy$包来实现coding，不多赘述

```
    def normal(self):
        """
        无惩罚项求解析解
        Returns:
                系数数组w, shaped (order+1, 1)
        """
        return np.linalg.pinv(self.X_matrix) @ self.T

    def regular(self, hp):
        """
        加入惩罚项求解析解
        :param hp:
                惩罚项中的超参数lambda
        :return:
                系数数组w, shaped (order+1, 1)
        """
        assert hp >= 0

        # 利用np.linalg.solve(A,B)求解w
        return np.linalg.solve(self.X_matrix.T @ self.X_matrix + hp *
np.identity(len(self.X_matrix.T)),
                               self.X_matrix.T @ self.T)
```

梯度下降法

从前面的数学原理可知，给定一个初始点$\boldsymbol{w}_0$，理论上**我们只需要不断地沿着当前点的梯度反方向走合适的距离，如此进行迭代，便可逐渐靠近最小值点$\boldsymbol{w}^*$**。至于迭代停止的条件，我选定一个精度$\delta = 1 \times 10^{-6}$，若当前点对应的梯度的绝对值小于$\delta$，则认为此时的梯度已经接近0了，可以停止迭代了，即

$$|\nabla \widetilde{E}(\boldsymbol{w}_t)| < \delta$$

伪代码如下

$$
\begin{aligned}
&k = 0 \\
&loss = \widetilde{E}(\boldsymbol{w}) = \frac{1}{2}[(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{T})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{T}) + \lambda \boldsymbol{w}^T\boldsymbol{w}] \\
&J(\boldsymbol{w}) = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{T} + \lambda\boldsymbol{w} \\
&\boldsymbol{Repeat}: \\
&\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - rate * J(\boldsymbol{w}) \\
&if(abs(J(\boldsymbol{w}_k)) < \delta): \\
&break \\
&k++ \\
&\boldsymbol{End}
\end{aligned}
$$

共轭梯度法

在前面的数学原理中，我们将问题转化为求解线性方程组$\boldsymbol{Aw}=\boldsymbol{b}$，其中$\boldsymbol{A}$是对称、正定的。

$$\boldsymbol{A}=\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I}$$

$$\boldsymbol{b}=\boldsymbol{X}^T\boldsymbol{T}$$

我们使用迭代法求解该方程组，每次迭代的优化方向$\boldsymbol{r}_t$和步长α_t均在前面给出。

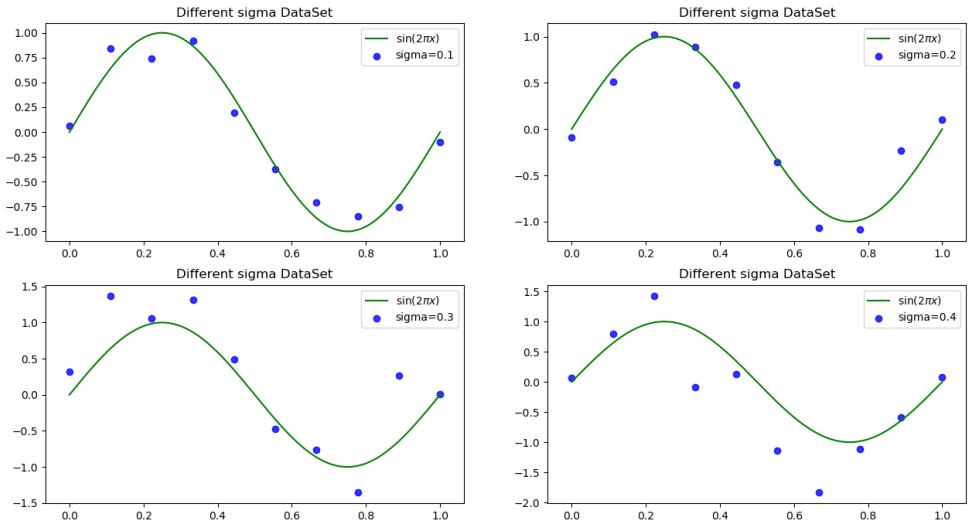
伪代码如下

$$
\begin{aligned}
&r_0 = b - Aw_0 \\
&p_0 = r_0 \\
&k = 0 \\
&\boldsymbol{Repeat}: \\
&if(abs(p_k) < \delta): \\
&break \\
&else: \\
&\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k} \\
&w_{k+1} = w_k + \alpha_k p_k \\
&r_{k+1} = r_k - \alpha_k A p_k \\
&b_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\
&p_{k+1} = r_{k+1} + b_k p_k \\
&k++ \\
&\boldsymbol{End}
\end{aligned}
$$

四、实验具体过程

生成数据集并加入噪声

我依次使用sigma为0.1， 0.2， 0.3， 0.4的噪声加入数据集中，结果如下图

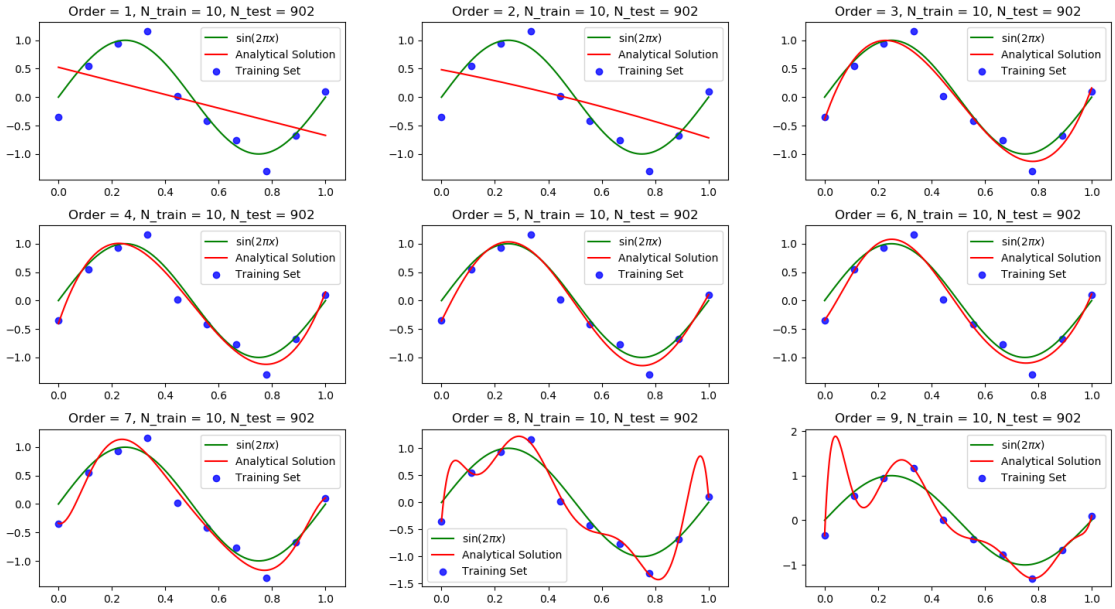


可见随着sigma增大，各个样本点偏离原函数$sin(2\pi x)$程度越大。**在后续实验中，若无特殊说明，sigma取0.3**

解析解（无正则项）

保持训练样本数目不变，变化多项式函数的阶数$order$

我保持$N_{train}=10$令$order$取1~9，分别绘制拟合图线如下

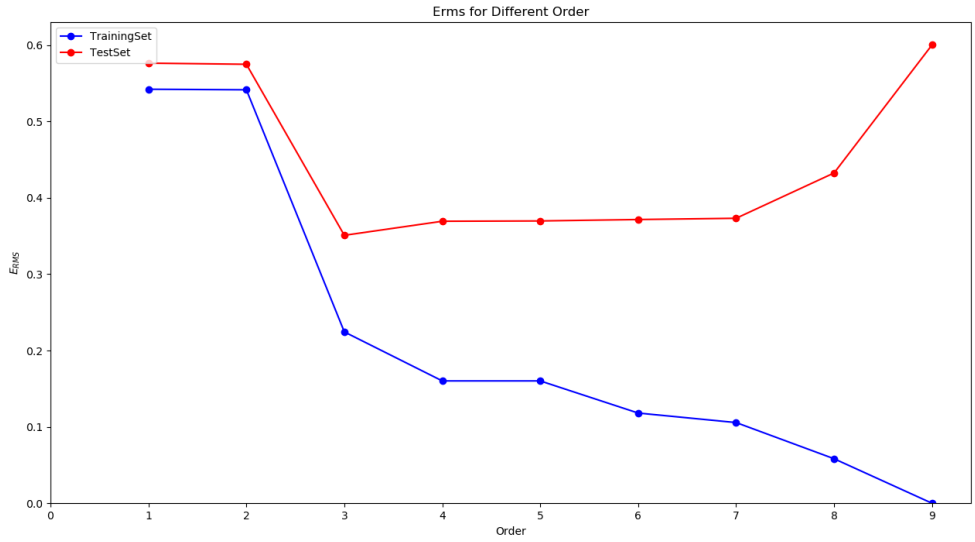


可以发现

- 在$order$很小（取1和2）时，拟合效果很差。这是由于阶数过低时，模型的复杂度低，变化能力很弱，并不能很好地在测试集合上拟合真实曲线，泛化性能不足。
- 在$order$为3~6时，拟合效果很不错。此时模型的复杂度有所提高，变化能力增强，能够很好地在测试集上拟合真实曲线，证明此时地泛化性能很好。
- $order$继续增大时，拟合出的曲线虽然可以"很好地"甚至“完美地”贴合训练集上的10个样本点，但是距离测试集上的真实曲线差距越来越大，这**其实就是发生了过拟合现象。**特别的，在$order$为9时，系数向量$\boldsymbol{w}$有唯一解，因此它的拟合曲线可以完美贴合10个样本点，**这时候模型的变化能力过强，将一些外部干扰的影响（噪声）也学习到了。最后的系数向量$\boldsymbol{w}$中各维数字绝对值很大，拟合曲线有明显的“摇摆”现象，偏离真实函数，泛化性能不足，这就是所谓的过拟合现象。**

下面展示不同$order$时的E_{RMS}，用来表征预测值与真实值之间的误差，即泛化性能的好坏

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



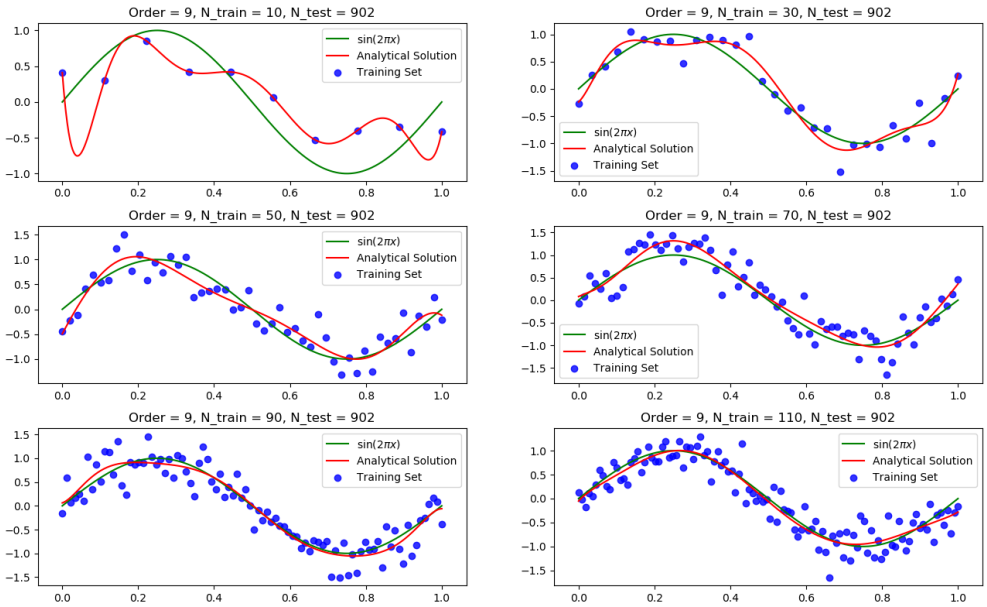
此图也很好地验证了前面对不同$order$， 模型泛化性能的变化分析

保持多项式阶数$order$不变，变化训练样本数目

在前面的实验中，10个样本点时，$order = 9$时模型泛化性能最差，发生了过拟合现象。所以我保持阶数$order$为9，变化训练样本数目

```
N_trainRange = range(10, 130, 20)
```

分别作出拟合曲线



可见，随着N_{train}的增大，作出的拟合曲线与真实的$sin(2\pi x)$越发贴合，**说明过拟合现象可以通过增加训练集样本点的数目来克服。**

解析解（有正则项）

在数学原理部分的叙述中，我们**引入了超参数λ来控制正则项的重要性**，并知道如何确定λ

$$\lambda = \mathrm{argmin}_{\lambda}\, \widetilde{E}(\boldsymbol{w}, \lambda)$$

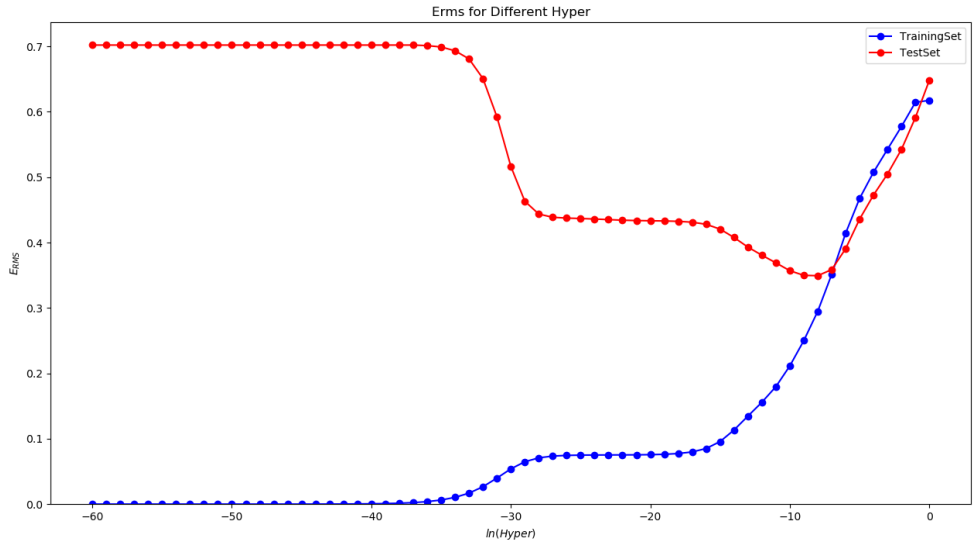
又

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

故

$$\lambda=\mathrm{argmin}_{\lambda}\ E_{RMS}$$

根据第一章课件中的提示，我可以人为地给λ划定一个大致范围，在这个范围内给λ取值，并求出对应的E_{RMS}，作图如下



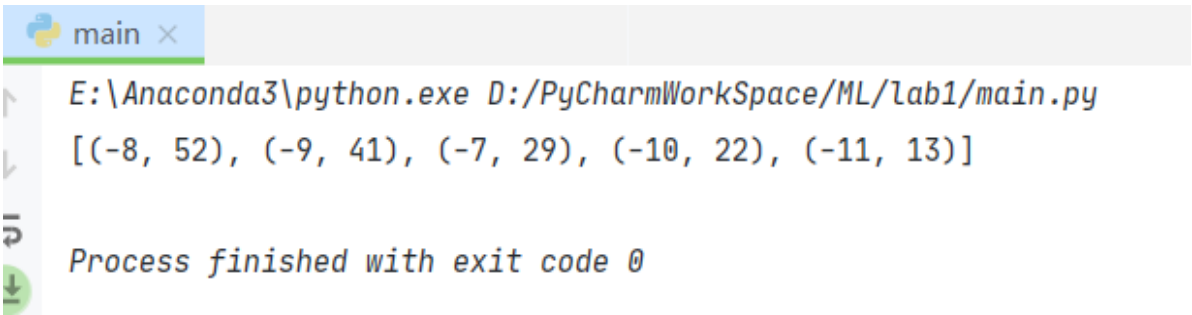
可以发现

- 当$ln(\lambda)$在（-60，-35）区间左右时，测试集上的E_{RMS}几乎不变。因为此时λ过小，正则项的重要性太低，模型退化为原模型。
- 当$ln(\lambda)$在（-35，-8）区间左右时，测试集上的E_{RMS}稳步下降。此时的λ较为合适，模型的泛化能力较强。
- 当$ln(\lambda)$在（-8，0）区间左右时，测试集上的E_{RMS}又开始上升。因为此时的λ过大，正则项重要性很高，迫使模型中的系数向量$\boldsymbol{w}$各维绝对值很小，模型的泛化性能又会下降。

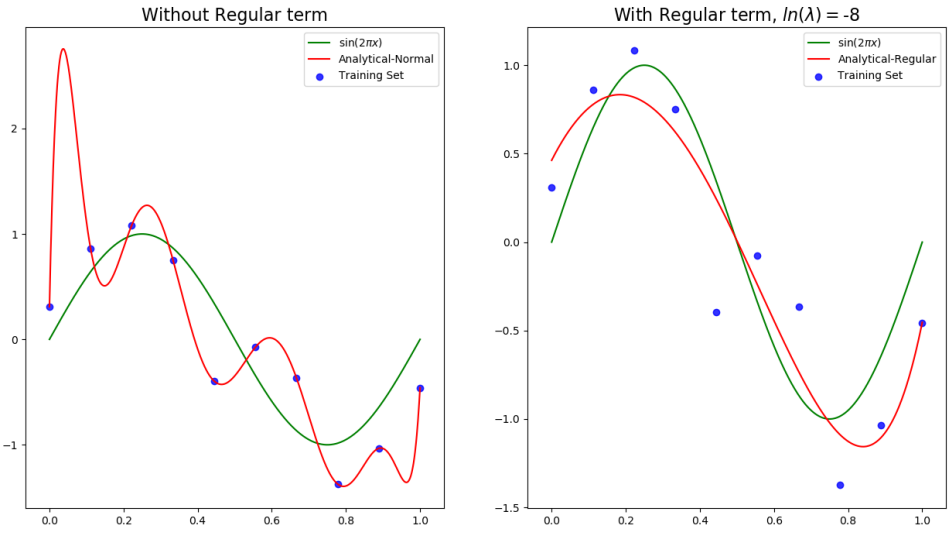
于是，按照这个思路，我利用循环做了203次实验，每次都记录最佳的λ，最后统计输出被选为“best”次数前五名的λ

$ln(\lambda)$	**-8**	**-9**	**-7**	**-10**	**-11**
被选为“best”的次数	52	41	29	22	13

截图为证



于是乎，我们得到了最佳的超参数$\lambda=e^{-8}$，下面仍然对于$order=9, N_{train}=10$展示一下有无正则项的拟合效果



可以明显地发现加入正则项后，模型的泛化性能好了很多。**这说明，增加正则项是克服过拟合现象的另一有效手段。**

在后续实验中，若无特殊说明，λ都取e^{-8}。

梯度下降法

我们给定一个初始点$\boldsymbol{w}_0$，理论上**只需要不断地沿着当前点的梯度反方向走合适的距离，如此进行迭代，便可逐渐靠近最小值点$\boldsymbol{w}^*$**，这就是所谓的梯度下降法的核心思想。

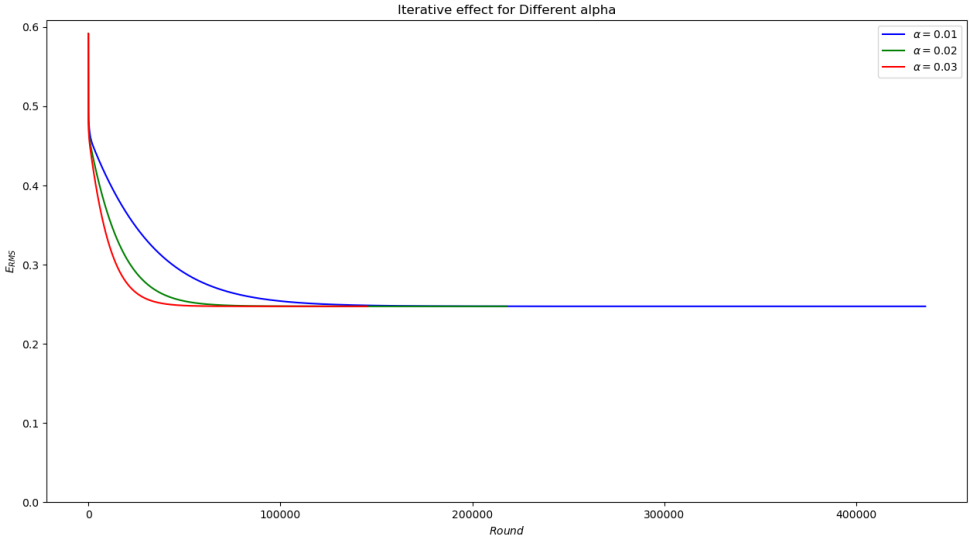
$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla \widetilde{E}(\boldsymbol{w}_i)$$

我选定精度$\delta = 1 \times 10^{-6}$，$\boldsymbol{w}_0$为全零向量。

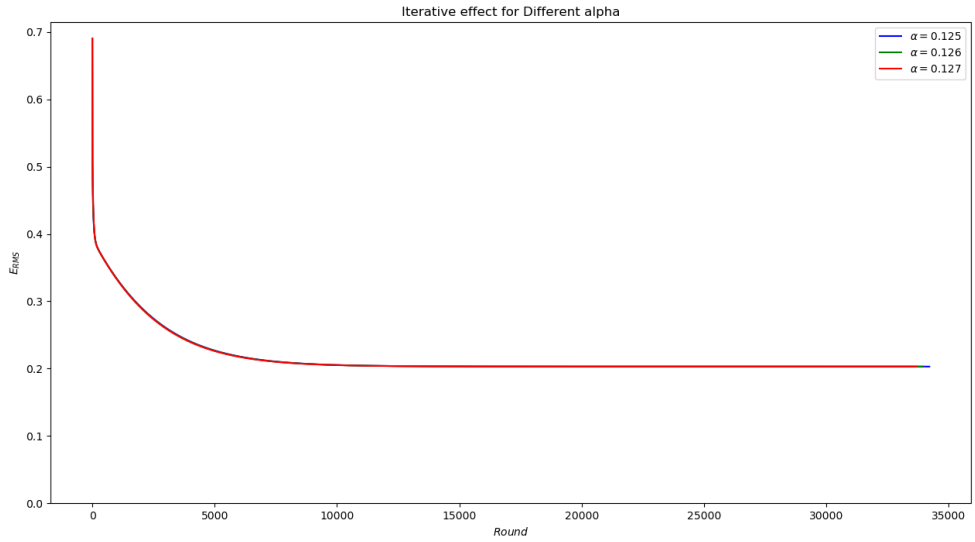
我们需要进行手动调参来确定学习率α

- 若α太大，可能会导致$\boldsymbol{w}$序列不收敛
- 若α太小，可能会导致$\boldsymbol{w}$序列收敛速度太慢

因此，我们选取α的原则就是：在保证收敛的情况下尽可能大

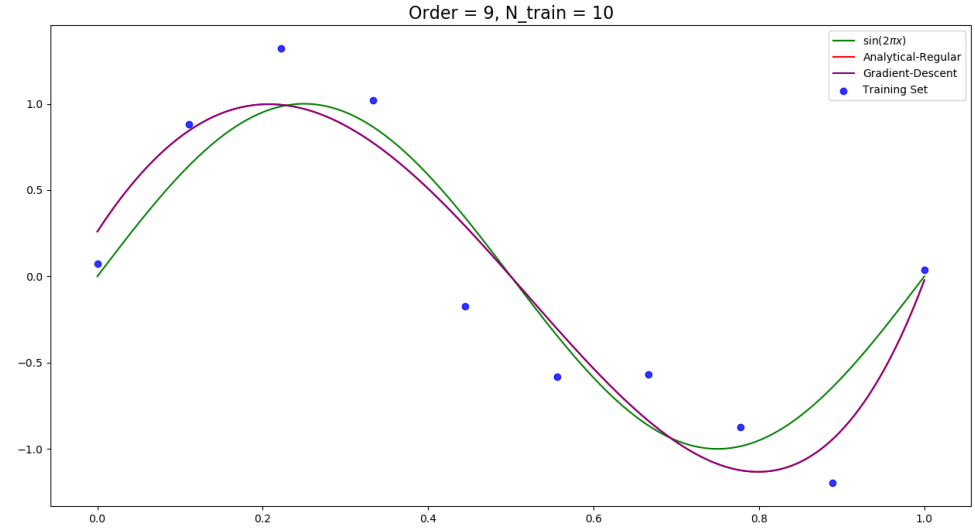


以上图为例，选取$\alpha = 0.03$，再增大α继续实验

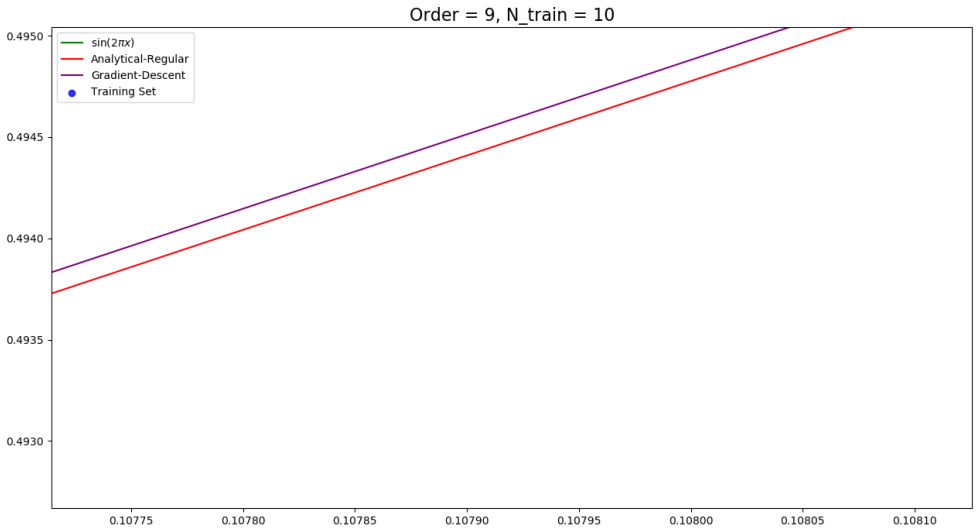


最终确定$\alpha = 0.127$, 此时三个参数的图线已经**极度接近以至于难以肉眼分辨。**

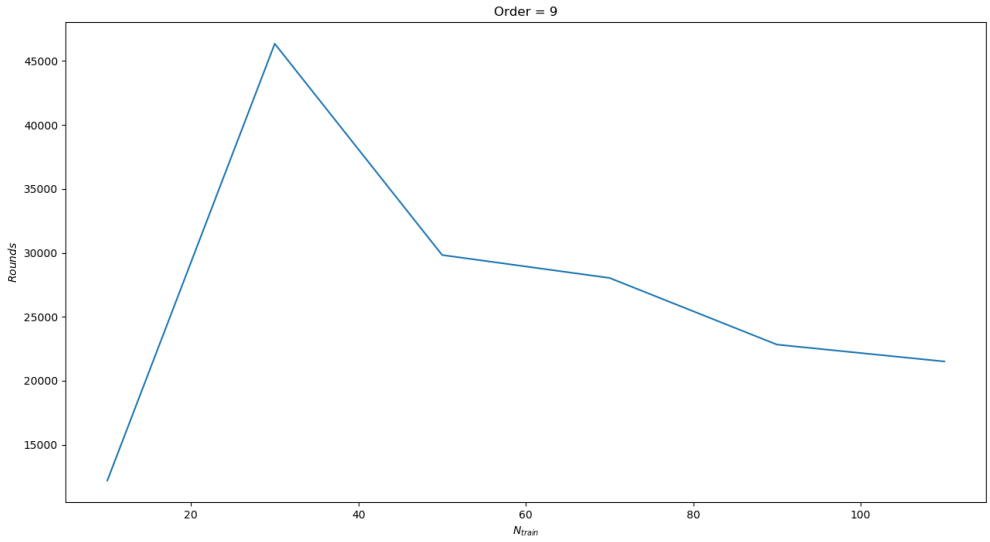
下面展示与解析解（有正则项)的对比



可见二者的结果极其接近，放大后才可看出区别，这说明我们的梯度下降法找到了一个很接近最优解的$\boldsymbol{w}$

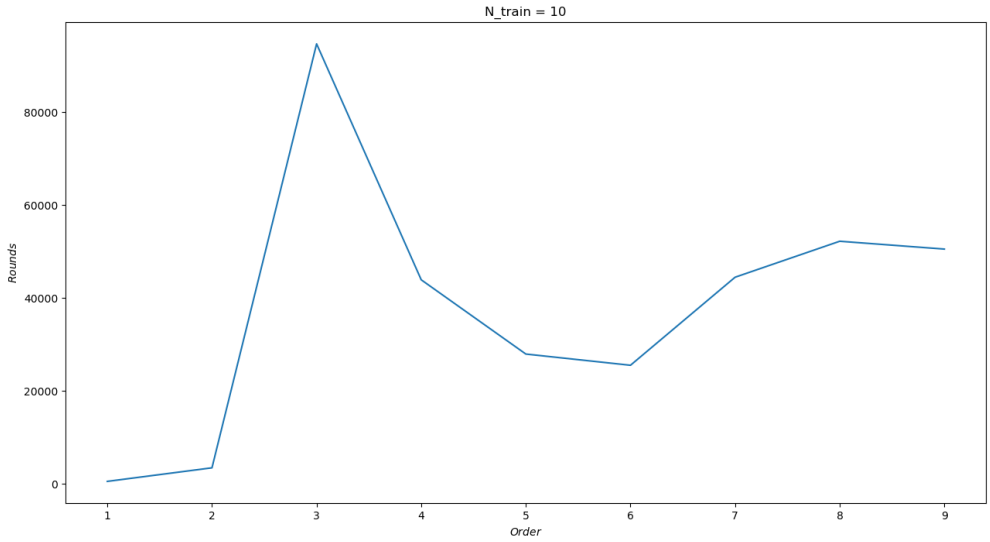


保持$order$为9，增大N_{train}，查看迭代次数的变化



可以发现：N_{train}对迭代轮数的影响很大，会造成较大范围的波动

保持N_{train}为10，改变$order$，查看迭代次数的变化

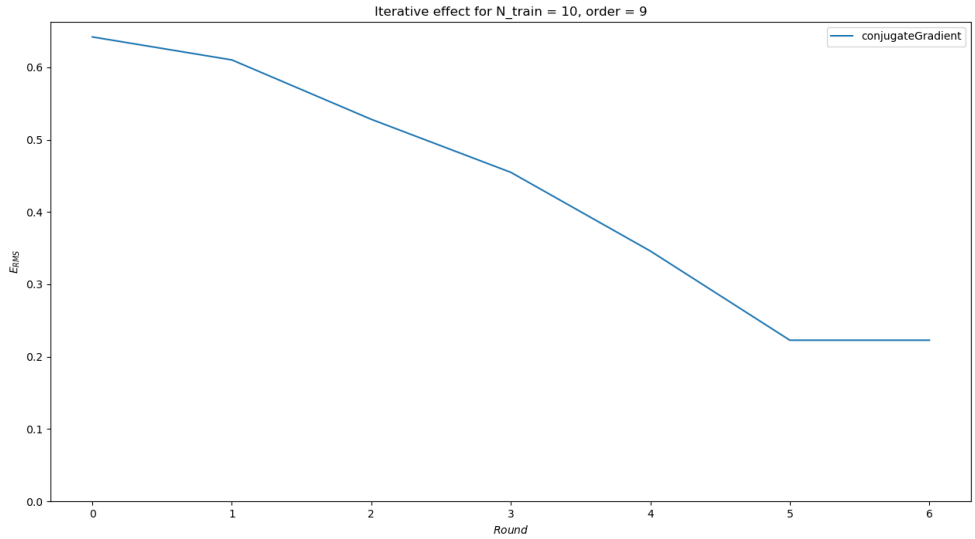


可以发现：$order$对迭代轮数的影响也很大，会造成较大范围的波动

共轭梯度法

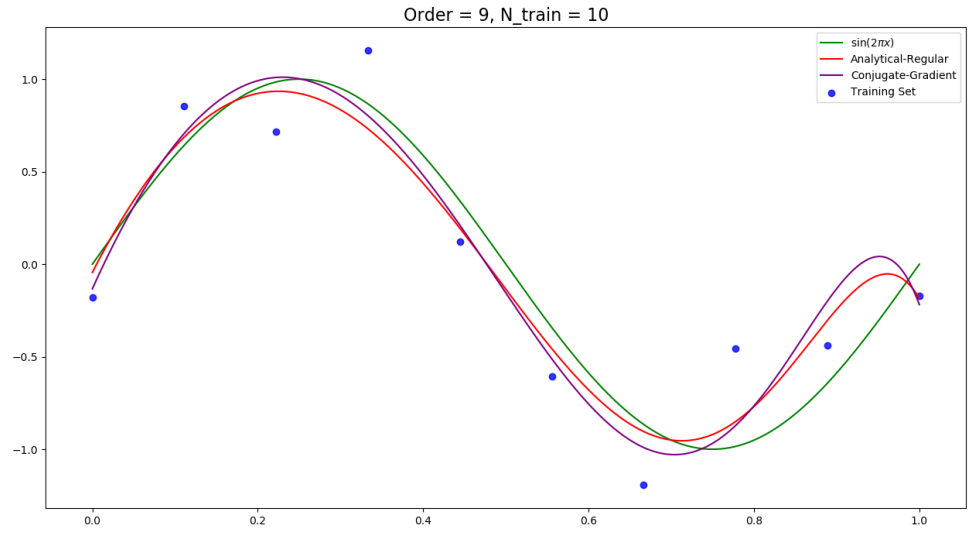
我们同样给定一个初始点$\boldsymbol{w}_0$为全零向量，**在解空间的每一个维度分别取求解最优解，每一维单独去做的时候不会影响到其他维，它的最大的优势就是每个方向都走到了极致，也即是说寻找极值的过程中绝不走曾经走过的方向,那么n维空间的函数极值也就走n步即可**

以$order = 9, N_{train} = 10$为例，展示迭代过程：



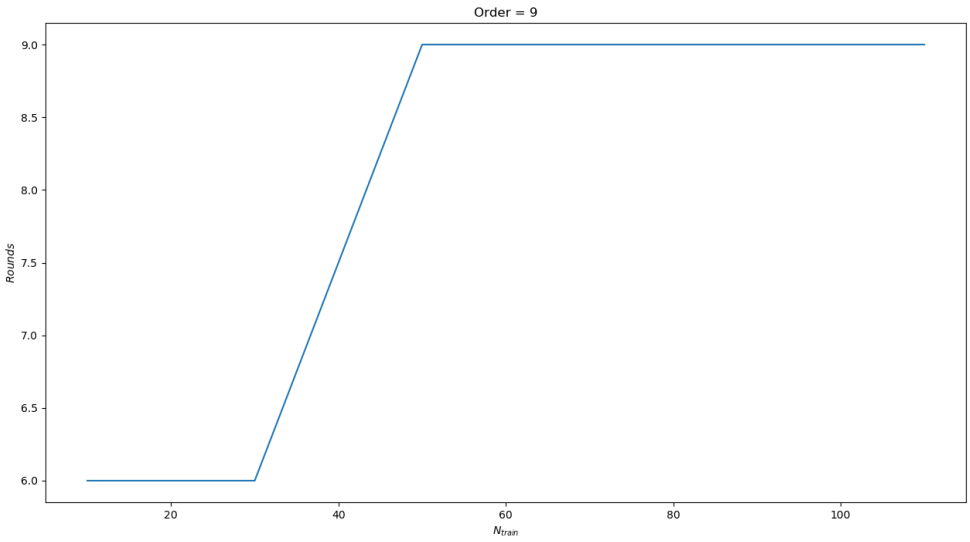
可见经过6轮迭代便达到了精度要求，即<=解空间的维度n

将拟合效果与解析解对比：



可见拟合效果也非常不错，且此方法迭代次数远远小于梯度下降法

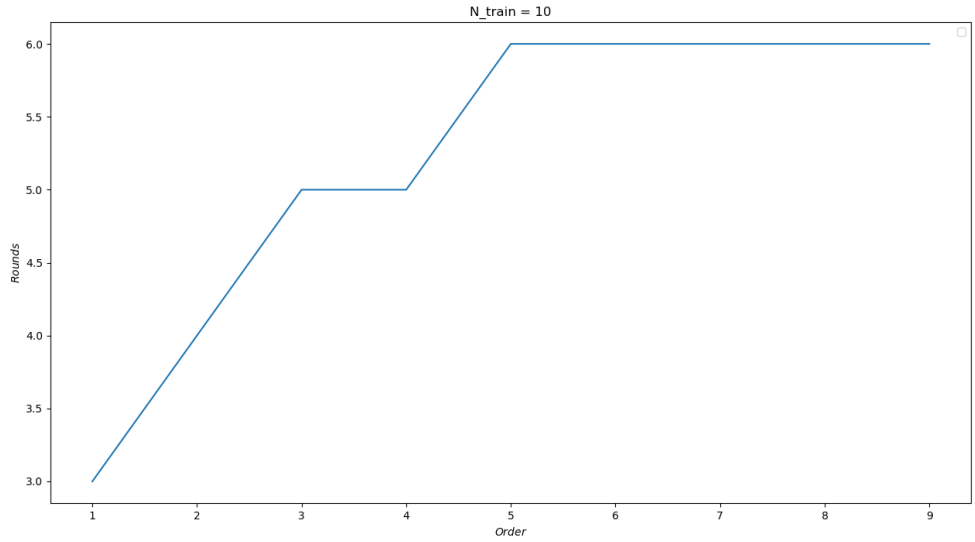
保持$order$为9，增大N_{train}，查看迭代次数的变化



可以发现：

- 训练样本数目增大，迭代轮数会逐渐增大
- 迭代轮数仍然<=解空间维数n

保持N_{train}为10，改变$order$，查看迭代次数的变化



可以发现：

- order增大，迭代轮数也会逐渐增加
- 迭代轮数仍然<=解空间维数n

五、结论

- 在使用无正则项的解析解来拟合时，若阶数$order$过大，会发生了过拟合现象

- 增加正则项是克服过拟合现象的一个有效手段
- 增加样本数量也是克服过拟合现象的一个有效手段
- 梯度下降法求解的迭代次数往往很大（大于10000）
- 共轭梯度法求解的迭代次数一定<=解空间维数n
- 当固定$order$，增大训练集样本数目，梯度下降法的迭代次数变化很大，而共轭梯度法的变化很小
- 当固定训练集样本数目，改变$order$，梯度下降法的迭代次数变化很大，而共轭梯度法的变化很小

六、附录:源代码(带注释)

main.py

```
import drawImage as dI
import statistics as sta
import numpy as np

sigma = 0.3  # noise的方差
N_train = 10  # 训练样本的数量
N_test = 100 * (N_train - 1) + 2  # 测试样本的数量(保证只有0,1点与训练集重复)
orderRange = range(1, 10)  # 多项式的阶
sigmaRange = np.array([0.1, 0.2, 0.3, 0.4])
layout = (3, 3)
# 展示：不同siama的图
# dI.DifferentSigmaImage(sigmaRange,N_train,N_test,layout)
# 展示：固定N_train,不同order时的拟合曲线
dI.DifferentOrderImage(sigma, N_train, N_test, orderRange, layout)
# orderRange = range(10)
# 展示: 固定N_train,不同order时的Erms图线
# dI.DifferentOrderErms(sigma, N_train, N_test, orderRange)
```

```
# 寻找最佳order
# x = sta.BestOrder(sigma, N_train, N_test, orderRange, times=100)
# print(x)
# 展示：固定order=9，不同N_train时的拟合曲线
N_trainRange = range(10, 130, 20)
# dI.DifferentNTrainImage(sigma, N_trainRange, N_test, layout)
hpRange = range(-60, 1, 1)
# 展示：固定order = 9， 不同hp时的Erms
# dI.DifferentHpErms(sigma, N_train, N_test, hpRange)
# 进行times次实验，寻找最佳hp--经过实验得到最佳hp为e^(-8)
# x = sta.BestHp(sigma, N_train, N_test, hpRange, times=203)
# print(x)
# 展示：order = 9时，有无正则项的图线区别
# dI.NormalVsRegular(sigma, N_train, N_test, hp=-8)
# dI.DifferentOrderErms(sigma, N_train, N_test, orderRange, hp=-8)
# alphaRange = np.array([0.125, 0.126, 0.127])
# dI.DifferentAlphaErms(sigma, N_train, 3, hp = -8, alphaRange = alphaRange)
# alpha = 0.03
# dI.GdVsRegular(sigma, N_train, N_test, 9, -8, alpha)
# dI.ErmsForCG(sigma, N_train = 10, order=9, hp=-8)
# dI.DifferentNRound(sigma, N_trainRange, 9, -8, alpha)
# dI.DifferentOrderRound(sigma, N_train=10, orderRange=orderRange, hp = -8,
alpha=alpha)
# dI.CGVsRegular(sigma,N_train,N_test, 9, -8)
# dI.DifferentNRound_CG(sigma,N_trainRange, 9, -8)
# dI.DifferentOrderRound_CG(sigma,N_train, orderRange,-8)
```

analytical.py

```
import numpy as np


class AnalyticSolution(object):

    def __init__(self, X_matrix, T):
        """
        generate an Object of Analytic_Solution
        Args:
            X_matrix: a matrix shaped (N, (order+1)) for X
            T: a ndarray of true values
        """
        self.X_matrix = X_matrix
        self.T = T

    def normal(self):
        """
        无惩罚项求解析解
        Returns:
                系数数组w, shaped (order+1, 1)
        """
        return np.linalg.pinv(self.X_matrix) @ self.T

    def regular(self, hp):
        """
        加入惩罚项求解析解
```

```
        :param hp:
                惩罚项中的超参数lambda
        :return:
                系数数组w，shaped (order+1, 1)
        """
        assert hp >= 0

        # 利用np.linalg.solve(A,B)求解w
        return np.linalg.solve(self.X_matrix.T @ self.X_matrix + hp *
np.identity(len(self.X_matrix.T)),
                               self.X_matrix.T @ self.T)
```

gradientDescent.py

```
import numpy as np
import basicOperation as Bo


class GradientDescent(object):
    def __init__(self, X, T, order, hp, alpha, delta=1e-5):

        self.X = X
        self.T = T
        self.order = order
        self.hp = hp
        self.alpha = alpha
        self.delta = delta

    def __loss(self, w):
        """
        求出当前w所对应的loss，使用E_RMS来量化
        :param w: 当前的w
        :return: 当前的loss
        """
        Y = Bo.predictY(self.X, w, self.order)
        return Bo.E_rms(Y, self.T, w, self.hp)

    def __gradient(self, w):
        """
        求代价函数的梯度
        :param w: 当前的w
        :return: 当前的梯度
        """
        X_matrix = Bo.xMatrix(self.X, self.order)
        return np.transpose(X_matrix) @ X_matrix @ w + np.exp(self.hp) * w -
X_matrix.T @ self.T

    def solve(self, w_0):
        """
        梯度下降法求解w
        :param w_0: w的初始值，通常取全零向量
        :return: 优化解w, 迭代轮数列表， lossList组成的三元组
        """

        # w表示当前的优化解
```

```
        w = w_0
        gradient = self.__gradient(w)
        # 记录迭代轮数
        k = 0

        roundList = [k]
        lossList = [self.__loss(w_0)]

        while not np.all(np.absolute(gradient) <= self.delta):
            k += 1
            w = w - self.alpha * gradient
            gradient = self.__gradient(w)
            roundList.append(k)
            lossList.append(self.__loss(w))

        return w, np.array(roundList), np.array(lossList)
```

conjudateGradient.py

```
import numpy as np
import basicOperation as Bo


class ConjugateGradient(object):
    def __init__(self, X, T, order, hp, delta=1e-5):
        self.X = X
        self.T = T
        self.order = order
        self.hp = hp
        self.delta = delta

    def A(self):
        # 求出代价函数E(w)写成二次型后的A矩阵
        X_matrix = Bo.xMatrix(self.X, self.order)
        return X_matrix.T @ X_matrix + np.exp(self.hp)

    def b(self):
        # 求出代价函数E(w)写成二次型后的b矩阵
        X_matrix = Bo.xMatrix(self.X, self.order)
        return X_matrix.T @ self.T

    def __loss(self, w):
        # 求当前loss，以Erms来衡量
        Y = Bo.predictY(self.X, w, self.order)
        return Bo.E_rms(Y, self.T, w, self.hp)

    def solve(self, A, b, w):
        """
        共轭梯度法求解方程组Aw = b
        :param A: 代价函数E(w)写成二次型后的A矩阵
        :param b: 代价函数E(w)写成二次型后的b矩阵
        :param w: 初始向量w，通常用全零向量
        :return: 该方程组的解w, 迭代轮数列表， lossList组成的三元组
        """
        k = 0   # 迭代轮数计数器
```

```
        roundList = [k]
        lossList = [self.__loss(w)]

        grad = np.dot(A, w) - b
        p = -grad
        while True:
            k += 1
            roundList.append(k)

            if abs(np.sum(p)) < self.delta:
                # 记录下最后一次的loss
                lossList.append(self.__loss(w))
                break
            gradSquare = np.dot(grad, grad)
            Ap = np.dot(A, p)
            alpha = gradSquare / np.dot(p, Ap)
            w += alpha * p
            newGrad = grad + alpha * Ap
            beta = np.dot(newGrad, newGrad) / gradSquare
            p = -newGrad + beta * p
            grad = newGrad

            # 记录当前的loss并加入列表
            lossList.append(self.__loss(w))

        return w, np.array(roundList), np.array(lossList)
```

basicOperation.py

```
import numpy as np


def generateData(sigma, N):
    """
    Generate DataSet with noise
    Args:
        sigma: the variance of the noise
        N: the number of the Training-set
    """
    assert sigma > 0
    assert N > 0
    assert isinstance(N, int)

    X = np.linspace(0, 1, num=N)
    noise = np.random.normal(0, scale=sigma, size=X.shape)
    T = np.sin(2 * np.pi * X) + noise
    return X, T


def xMatrix(X, order=2):
    """
    Transform an ndarray's shape from (N, ) to (N, (order + 1)) .
    Args:
        X: an ndarray.
        order:int, order for polynomial.
```

```
    Returns:
        e.g.
        in:  [a b c]
        out: [[1 a a^2...a^order]
              [1 b b^2...b^order]
              [1 c c^2...c^order]]
    """
    assert X.ndim == 1
    assert isinstance(order, int)

    # 一维数组变二维
    cols = np.ones(len(X))[:, np.newaxis]
    for i in range(0, order):
        # 使用hstack水平连接时，要求2个数组的dim相等，故需要把cols[:, i]*X转换为二维数组
        cols = np.hstack((cols, (cols[:, i] * X)[:, np.newaxis]))
    return cols


def predictY(X, w, order):
    """
    根据数组X和系数数组w给出预测的Y数组
    Args:
        X: an ndarray shaped (N, )，代表各点的横坐标
        w: an ndarray shaped (order+1, )，代表多项式的各个系数
        order: The order of polynomials
    Returns:
        Y: an ndarray shaped (N,)，代表各点的纵坐标
    """
    X_matrix = xMatrix(X, order=order)

    return X_matrix @ w


def E_rms(Y, T, w, hp=0):
    """
    求出“根均方差”E_rms
    :param Y: 预测值矩阵
    :param T: 真实值矩阵
    :param w: 系数数组
    :param hp: 超参数
    :return: Y和T之间的根均方差的值
    """
    assert Y.ndim == 1
    assert Y.shape == T.shape

    if hp != 0:
        return np.sqrt(np.mean(np.square(Y - T)) + np.exp(hp) * (np.transpose(w)
@ w) / len(T))
    else:
        return np.sqrt(np.mean(np.square(Y - T)))
```

statistic.py

```
from collections import Counter
import numpy as np
```

```
import analytical
import basicOperation as Bo


def BestOrder(sigma, N_train, N_test, orderRange, times=100):
    """
    找到最佳阶数Order，返回（order，counter）的tuple
    :param sigma: 噪声方差
    :param N_train: 训练集数据数目
    :param N_test: 测试集数据数目
    :param orderRange: order范围列表
    :param times: 进行的实验次数
    :return: times次实验中被选为最佳的次数最多的阶数order，以及被选为最佳的次数counter，所
组成的tuple
    """
    bestOrderList = []
    # 做times次实验
    for i in range(times):
        X_train, T_train = Bo.generateData(sigma, N_train)
        X_test, T_test = Bo.generateData(sigma, N_test)
        ErmsTrainList = []
        ErmsTestList = []
        for order in orderRange:
            # 先通过TrainSet得到系数数组w，进而预测Y
            anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train,
order), T_train)
            w = anaSolution.normal()
            Y_train = Bo.predictY(X_train, w, order)
            # 得到ErmsTrain并添加进List中
            ErmsTrain = Bo.E_rms(Y=Y_train, T=T_train, w=w)
            ErmsTrainList.append(ErmsTrain)

            Y_test = Bo.predictY(X_test, w, order)
            # 得到ErmsTest并添加进List中
            ErmsTest = Bo.E_rms(Y=Y_test, T=T_test, w=w)
            ErmsTestList.append(ErmsTest)

        bestIndex = np.where(ErmsTestList == np.min(ErmsTestList))[0][0]
        bestOrder = orderRange[bestIndex]
        bestOrderList.append(bestOrder)

    collectionOrders = Counter(bestOrderList)
    most_counterNum = collectionOrders.most_common(1)
    return most_counterNum[0]


def BestHp(sigma, N_train, N_test, hpRange, times=100):
    """
    找到最佳hp，返回（hp，counter）的tuple
    :param sigma: 噪声方差
    :param N_train: 训练集数据数目
    :param N_test: 测试集数据数目
    :param hpRange: hp范围列表
    :param times: 进行的实验次数
    :return: times次实验中被选为最佳的次数最多的超参数hp，以及被选为最佳的次数counter，所组
成的tuple
    """
    bestHpList = []
```

```
    # 做times次实验
    for i in range(times):
        X_train, T_train = Bo.generateData(sigma, N_train)
        X_test, T_test = Bo.generateData(sigma, N_test)
        ErmsTrainList = []
        ErmsTestList = []
        for hp in hpRange:
            # 先通过TrainSet得到系数数组w，进而预测Y
            anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train,
order=9), T_train)
            w = anaSolution.regular(np.exp(hp))
            Y_train = Bo.predictY(X_train, w, order=9)
            # 得到ErmsTrain并添加进List中
            ErmsTrain = Bo.E_rms(Y=Y_train, T=T_train, w=w, hp=hp)
            ErmsTrainList.append(ErmsTrain)

            Y_test = Bo.predictY(X_test, w, order=9)
            # 得到ErmsTest并添加进List中
            ErmsTest = Bo.E_rms(Y=Y_test, T=T_test, w=w, hp=hp)
            ErmsTestList.append(ErmsTest)

        bestIndex = np.where(ErmsTestList == np.min(ErmsTestList))[0][0]
        bestHp = hpRange[bestIndex]
        bestHpList.append(bestHp)

    collectionHps = Counter(bestHpList)
    most_counterNum = collectionHps.most_common(5)
    return most_counterNum
```

drawlamge.py

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

import analytical
import basicOperation as Bo
import gradientDescent as gd
import conjugateGradient as cg


def DifferentSigmaImage(sigmaRange,N_train, N_test, layout):
    fig, axes = plt.subplots(*layout)

    for index in range(len(sigmaRange)):
        sigma = sigmaRange[index]
        X_test = np.linspace(0, 1, num=N_test)
        X_train, T = Bo.generateData(sigma, N_train)
        axTarget = axes[index // layout[1]][index % layout[1]]
        sns.regplot(X_train, T, fit_reg=False, color='b',
label="sigma="+str(sigma), ax=axTarget)
        sns.lineplot(X_test, np.sin(2 * np.pi * X_test), color="g",
label="$\\sin(2\\pi x)$",
                     ax=axTarget)
        # 图的相关设置
```

```
        title = 'Different sigma DataSet'
        props = {'title': title}
        axTarget.set(**props)

    plt.show()


def DifferentOrderImage(sigma, N_train, N_test, orderRange, layout):
    """
    draw不同order下的拟合曲线图像
    :param sigma: 噪声方差
    :param N_train: 训练集数据数目
    :param N_test: 测试集数据数目
    :param orderRange: order范围列表
    :param layout: 图像布局
    :return: 所需图像
    """
    X_train, T = Bo.generateData(sigma, N_train)
    X_test = np.linspace(0, 1, num=N_test)

    fig, axes = plt.subplots(*layout)
    for index in range(len(orderRange)):
        # 求出当前的order
        order = orderRange[index]
        # 确定目标坐标系
        axTarget = axes[index // layout[1]][index % layout[1]]
        # 使用regplot（）函数制作散点图，您必须提供至少2个list-like：X轴和Y轴上的点的位置。
        # 默认情况下绘制线性回归拟合直线，可以使用fit_reg = False将其删除
        sns.regplot(X_train, T, fit_reg=False, color='b', label="Training Set",
ax=axTarget)
        # 画真正的sin(2*np.pi*x)图线
        sns.lineplot(X_test, np.sin(2 * np.pi * X_test), color="g",
label="$\\sin(2\\pi x)$",
                     ax=axTarget)
        # 画多项式拟合曲线
        anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train, order), T)
        sns.lineplot(X_test, Bo.predictY(X_test, anaSolution.normal(), order),
color="r", label="Analytical Solution",
                     ax=axTarget)

        # 图的相关设置
        title = "Order = " + str(order) + ", N_train = " + str(N_train) + ",
N_test = " + str(N_test)
        props = {'title': title}
        axTarget.set(**props)

    plt.show()


def DifferentOrderErms(sigma, N_train, N_test, orderRange, hp = 0):
    """
    画出不同order下的Erms图像
    :param sigma: 噪声方差
    :param N_train: 训练集数据数目
    :param N_test: 测试集数据数目
    :param orderRange: order范围列表
    :param hp: 超参数
    :return: 所需的图像
```

```
    """
    X_train, T_train = Bo.generateData(sigma, N_train)
    X_test, T_test = Bo.generateData(sigma, N_test)
    ErmsTrainList = []
    ErmsTestList = []
    for order in orderRange:
        # 先通过TrainSet得到系数数组w，进而预测Y
        anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train, order),
T_train)
        w = anaSolution.normal()
        # 适配有无正则项的情况
        if(hp != 0):
            w = anaSolution.regular(np.exp(hp))

        Y_train = Bo.predictY(X_train, w, order)
        # 得到ErmsTrain并添加进List中
        ErmsTrain = Bo.E_rms(Y=Y_train, T=T_train, w=w, hp=hp)
        ErmsTrainList.append(ErmsTrain)

        Y_test = Bo.predictY(X_test, w, order)
        # 得到ErmsTest并添加进List中
        ErmsTest = Bo.E_rms(Y=Y_test, T=T_test, w=w, hp=hp)
        ErmsTestList.append(ErmsTest)

    fig, axes = plt.subplots()
    axes.plot(orderRange, ErmsTrainList, 'b-o', label="TrainingSet")
    axes.plot(orderRange, ErmsTestList, 'r-o', label="TestSet")
    title = "Erms for Different Order"

    props = {'title': title,
             'xlabel': 'Order',
             'ylabel': '$E_{RMS}$',
             }
    axes.set(**props)

    bestIndex = np.where(ErmsTestList == np.min(ErmsTestList))[0][0]
    bestOrder = orderRange[bestIndex]
    print("bestOrder:", bestOrder, np.min(ErmsTestList))
    axes.legend()
    axes.set_ylim(bottom=0)
    axes.set_xticks(np.arange(0, 10))

    plt.show()


def DifferentNTrainImage(sigma, N_trainRange, N_test, layout, order=9):
    fig, axes = plt.subplots(*layout)

    for index in range(len(N_trainRange)):
        N_train = N_trainRange[index]
        X_train, T = Bo.generateData(sigma, N_train)
        X_test = np.linspace(0, 1, num=N_test)

        # 确定目标坐标系
        axTarget = axes[index // layout[1]][index % layout[1]]
        # 使用regplot（）函数制作散点图，您必须提供至少2个list-like：X轴和Y轴上的点的位置。
        # 默认情况下绘制线性回归拟合直线，可以使用fit_reg = False将其删除
```

```
        sns.regplot(X_train, T, fit_reg=False, color='b', label="Training Set",
ax=axTarget)
        # 画真正的sin(2*np.pi*x)图线
        sns.lineplot(X_test, np.sin(2 * np.pi * X_test), color="g",
label="$\\sin(2\\pi x)$",
                     ax=axTarget)
        # 画多项式拟合曲线
        anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train, order), T)
        sns.lineplot(X_test, Bo.predictY(X_test, anaSolution.normal(), order),
color="r", label="Analytical Solution",
                     ax=axTarget)

        # 图的相关设置
        title = "Order = " + str(order) + ", N_train = " + str(N_train) + ",
N_test = " + str(N_test)
        props = {'title': title}
        axTarget.set(**props)

    plt.show()


def DifferentHpErms(sigma, N_train, N_test, hpRange):
    """
    画出不同超参数hp下的Erms图像
    :param sigma: 噪声方差
    :param N_train: 训练集数据数目
    :param N_test: 测试集数据数目
    :param hpRange: hp范围列表
    :return: 所需的图像
    """
    X_train, T_train = Bo.generateData(sigma, N_train)
    X_test, T_test = Bo.generateData(sigma, N_test)
    ErmsTrainList = []
    ErmsTestList = []
    for hp in hpRange:
        # 先通过TrainSet得到系数数组w，进而预测Y
        anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train, 9),
T_train)
        # 此处使用加入惩罚项的w,实际使用的超参数为e^hp
        w = anaSolution.regular(np.exp(hp))
        Y_train = Bo.predictY(X_train, w, 9)
        # 得到ErmsTrain并添加进List中
        ErmsTrain = Bo.E_rms(Y=Y_train, T=T_train, w=w, hp=hp)
        ErmsTrainList.append(ErmsTrain)

        Y_test = Bo.predictY(X_test, w, 9)
        # 得到ErmsTest并添加进List中
        ErmsTest = Bo.E_rms(Y=Y_test, T=T_test, w=w, hp=hp)
        ErmsTestList.append(ErmsTest)

    fig, axes = plt.subplots()
    axes.plot(hpRange, ErmsTrainList, 'b-o', label="TrainingSet")
    axes.plot(hpRange, ErmsTestList, 'r-o', label="TestSet")
    title = "Erms for Different Hyper"

    props = {'title': title,
             'xlabel': '$ln(Hyper)$',
             'ylabel': '$E_{RMS}$',
```

```
            }
    axes.set(**props)

    bestIndex = np.where(ErmsTestList == np.min(ErmsTestList))[0][0]
    bestHp = hpRange[bestIndex]
    print("bestHp:", bestHp, np.min(ErmsTestList))
    axes.legend()
    axes.set_ylim(bottom=0)
    plt.show()


def NormalVsRegular(sigma, N_train, N_test, hp):
    # 2个图像都使用相同的训练集和测试集
    X_train, T_train = Bo.generateData(sigma, N_train)
    X_test, T_test = Bo.generateData(sigma, N_test)
    layout = (1, 2)
    fig, axes = plt.subplots(*layout)

    # 下面画不加惩罚项的图
    sns.regplot(X_train, T_train, fit_reg=False, color='b', label="Training
Set", ax=axes[0])
    # 画真正的sin(2*np.pi*x)图线
    sns.lineplot(X_test, np.sin(2 * np.pi * X_test), color="g",
label="$\\sin(2\\pi x)$",
                 ax=axes[0])
    # 画多项式拟合曲线
    anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train, order = 9),
T_train)
    sns.lineplot(X_test, Bo.predictY(X_test, anaSolution.normal(), order = 9),
color="r", label="Analytical-Normal",
                 ax=axes[0])
    axes[0].set_title('Without Regular term', fontsize=16)

    # 下面画加上惩罚项的图
    sns.regplot(X_train, T_train, fit_reg=False, color='b', label="Training
Set", ax=axes[1])
    # 画真正的sin(2*np.pi*x)图线
    sns.lineplot(X_test, np.sin(2 * np.pi * X_test), color="g",
label="$\\sin(2\\pi x)$",
                 ax=axes[1])
    # 画多项式拟合曲线
    anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train, order=9),
T_train)
    w = anaSolution.regular(np.exp(hp))
    sns.lineplot(X_test, Bo.predictY(X_test, w, order=9), color="r",
label="Analytical-Regular",
                 ax=axes[1])
    axes[1].set_title('With Regular term, ' + '$ln(\\lambda)=$'+str(hp),
fontsize=16)

    plt.show()


def DifferentAlphaErms(sigma, N_train, order, hp, alphaRange):
    X_train, T_train = Bo.generateData(sigma, N_train)
    colorList = ['b', 'g', 'r']

    fig, axes = plt.subplots()
```

```
    for alpha in alphaRange:
        # 选图线颜色
        index = np.where(alphaRange == alpha)[0][0]
        color = colorList[index]

        # 先通过TrainSet得到系数数组w，进而预测Y
        gdSolution = gd.GradientDescent(X_train, T_train, order, hp, alpha)
        w_0 = np.zeros(order+1)
        w, roundList, lossList = gdSolution.solve(w_0)

        axes.plot(roundList, lossList, color = color, label="$\\alpha =
$"+str(alpha))

    title = "Iterative effect for Different alpha"
    props = {'title': title,
             'xlabel': '$Round$',
             'ylabel': '$E_{RMS}$',
             }
    axes.set(**props)
    axes.legend()
    axes.set_ylim(bottom=0)
    plt.show()


def GdVsRegular(sigma, N_train, N_test, order, hp, alpha):
    # 2个图像都使用相同的训练集和测试集
    X_train, T_train = Bo.generateData(sigma, N_train)
    X_test, T_test = Bo.generateData(sigma, N_test)
    fig, axes = plt.subplots()

    sns.regplot(X_train, T_train, fit_reg=False, color='b', label="Training
Set", ax=axes)
    # 画真正的sin(2*np.pi*x)图线
    sns.lineplot(X_test, np.sin(2 * np.pi * X_test), color="g",
label="$\\sin(2\\pi x)$",
                 ax=axes)
    # 画多项式拟合曲线
    anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train, order=order),
T_train)
    w = anaSolution.regular(np.exp(hp))
    sns.lineplot(X_test, Bo.predictY(X_test, w, order=order), color="r",
label="Analytical-Regular",
                 ax=axes)

    gdSolution = gd.GradientDescent(X_train, T_train,order,hp,alpha)
    w_0 = np.zeros(order + 1)
    w2, roundList, lossList = gdSolution.solve(w_0)
    sns.lineplot(X_test, Bo.predictY(X_test, w2, order=order), color="purple",
label="Gradient-Descent",
                 ax=axes)
    axes.set_title('Order = '+str(order) + ', N_train = '+str(N_train),
fontsize=16)

    plt.show()


def DifferentNRound(sigma, N_trainRange, order, hp, alpha):
    fig, axes = plt.subplots()
```

```
    roundNumber = []
    for N_train in N_trainRange:
        X_train, T_train = Bo.generateData(sigma, N_train)
        gdSolution = gd.GradientDescent(X_train, T_train, order, hp, alpha)
        w_0 = np.zeros(order+1)
        w, roundList, lossList = gdSolution.solve(w_0)
        roundNumber.append(roundList[-1])

    axes.plot(N_trainRange, np.array(roundNumber))
    title = "Order = "+str(order)
    props = {'title': title,
             'xlabel': '$N_{train}$',
             'ylabel': '$Rounds$',
             }
    axes.set(**props)
    axes.legend()
    plt.show()


def DifferentOrderRound(sigma, N_train, orderRange, hp, alpha):
    fig, axes = plt.subplots()
    roundNumber = []
    X_train, T_train = Bo.generateData(sigma, N_train)

    for order in orderRange:
        gdSolution = gd.GradientDescent(X_train, T_train, order, hp, alpha)
        w_0 = np.zeros(order+1)
        w, roundList, lossList = gdSolution.solve(w_0)
        roundNumber.append(roundList[-1])

    axes.plot(orderRange, np.array(roundNumber))
    title = "N_train = "+str(N_train)
    props = {'title': title,
             'xlabel': '$Order$',
             'ylabel': '$Rounds$',
             }
    axes.set(**props)
    axes.legend()
    plt.show()


def ErmsForCG(sigma, N_train, order, hp):
    X_train, T_train = Bo.generateData(sigma, N_train)

    fig, axes = plt.subplots()

    cgSolution = cg.ConjugateGradient(X_train, T_train, order, hp)
    w_0 = np.zeros(order + 1)
    w, roundList, lossList = cgSolution.solve(cgSolution.A(), cgSolution.b(),
w_0)

    axes.plot(roundList, lossList, label="conjugateGradient")

    title = "Iterative effect for N_train = " + str(N_train) + ', order = ' +
str(order)
    props = {'title': title,
             'xlabel': '$Round$',
             'ylabel': '$E_{RMS}$',
```

```
            }
    axes.set(**props)
    axes.legend()
    axes.set_ylim(bottom=0)
    plt.show()


def CGVsRegular(sigma, N_train, N_test, order, hp):
    X_train, T_train = Bo.generateData(sigma, N_train)
    X_test, T_test = Bo.generateData(sigma, N_test)
    fig, axes = plt.subplots()

    sns.regplot(X_train, T_train, fit_reg=False, color='b', label="Training
Set", ax=axes)
    # 画真正的sin(2*np.pi*x)图线
    sns.lineplot(X_test, np.sin(2 * np.pi * X_test), color="g",
label="$\\sin(2\\pi x)$",
                 ax=axes)
    # 画多项式拟合曲线
    anaSolution = analytical.AnalyticSolution(Bo.xMatrix(X_train, order=order),
T_train)
    w = anaSolution.regular(np.exp(hp))
    sns.lineplot(X_test, Bo.predictY(X_test, w, order=order), color="r",
label="Analytical-Regular",
                 ax=axes)

    cgSolution = cg.ConjugateGradient(X_train, T_train,order,hp)
    w_0 = np.zeros(order + 1)
    w2, roundList, lossList = cgSolution.solve(cgSolution.A(), cgSolution.b(),
w_0)

    sns.lineplot(X_test, Bo.predictY(X_test, w2, order=order), color="purple",
label="Conjugate-Gradient",
                 ax=axes)

    axes.set_title('Order = ' + str(order) + ', N_train = ' + str(N_train),
fontsize=16)

    plt.show()


def DifferentNRound_CG(sigma, N_trainRange, order, hp):
    fig, axes = plt.subplots()
    roundNumber = []
    for N_train in N_trainRange:
        X_train, T_train = Bo.generateData(sigma, N_train)
        cgSolution = cg.ConjugateGradient(X_train, T_train,order,hp)
        w_0 = np.zeros(order+1)
        w, roundList, lossList = cgSolution.solve(cgSolution.A(),
cgSolution.b(), w_0)
        roundNumber.append(roundList[-1])

    axes.plot(N_trainRange, np.array(roundNumber))
    title = "Order = "+str(order)
    props = {'title': title,
             'xlabel': '$N_{train}$',
             'ylabel': '$Rounds$',
             }
```

```
    axes.set(**props)
    plt.show()


def DifferentOrderRound_CG(sigma, N_train, orderRange, hp):
    fig, axes = plt.subplots()
    roundNumber = []
    X_train, T_train = Bo.generateData(sigma, N_train)

    for order in orderRange:
        cgSolution = cg.ConjugateGradient(X_train, T_train,order,hp)
        w_0 = np.zeros(order + 1)
        w, roundList, lossList = cgSolution.solve(cgSolution.A(),
cgSolution.b(), w_0)
        roundNumber.append(roundList[-1])

    axes.plot(orderRange, np.array(roundNumber))
    title = "N_train = " + str(N_train)
    props = {'title': title,
             'xlabel': '$Order$',
             'ylabel': '$Rounds$',
             }
    axes.set(**props)
    axes.legend()
    plt.show()
```