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：必修

实验题目：多项式拟合正弦函数

学号：1190202105

姓名：傅浩东

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：Windows 11; Visual Studio Code; python 3.9.7

三. 设计实验（主要算法和数据结构）

1. 数据生成

利用$sin(2\pi x)$函数产生样本，均匀分布在范围$[0.1]$内，采用函数`numpy.linspace`生成均匀分布的一维向量，样本个数为N，设为样本的X轴上的值，接下来通过sin函数获得Y轴上的值，通过函数`numpy.random.normal`加上一个均值为0，方差为0.2的高斯噪音。

```
def geneData(number, loc, scale):
    Xn = np.linspace(0, 1, num=number)
    noise = np.random.normal(loc=loc, scale=scale, size=len(Xn))
    T = np.sin(2 * np.pi * Xn) + noise
    return Xn, T
```

以上添加了高斯噪音的是我们的训练样本，而选择没加噪音的作为测试样本。初步选择训练样本数量为10，而测试样本数量为100。

2. 解析解

首先建立如下多项式函数，其中$y(x,w)$是x的多项式函数，是w的线性函数。

$$y(x,w)=w_0+w_1x+w_2x^2+\cdots+w_mx^m=\sum_{i=0}^{m}w_ix^i$$

目标就是假设给定m，确定参数w，建立误差函数，测试每个样本点目标值t与预测函数y之间的误差，显然是最小二乘问题。假设：

$$X=\begin{bmatrix}1 & x_1 & x_1^2 & \cdots & x_1^m\\ 1 & x_2 & x_2^2 & \cdots & x_2^m\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & x_N & x_N^2 & \cdots & x_N^m\end{bmatrix}$$

$$w=(w_0,w_1,\cdots,w_m)^T, T=(t_1,t_2,\cdots,t_N)^T$$

(1) 不带正则项(惩罚项)的多项式拟合

建立误差函数$E(x)$，测量每个样本点目标值t与预测函数y之间的误差：

$$\begin{aligned}E(w)&=\frac{1}{2}\sum_{i=0}^{N}\{y(x_i,w)-t_i\}^2\\&=\frac{1}{2}(Xw-T)^T(Xw-T)\\&=\frac{1}{2}(w^TX^TXw+T^TT-T^TXw-w^TX^TT)\end{aligned}$$

认为当 w可以使得$E(w)$达到最小时，预测模型$Y(w,x)$就在给定的样本上尽可能地逼近了目标函数，由此将w视为预测模型$Y(x,w)$的参数列向量。求导，设倒数为0，求唯一解w^*：

$$\begin{aligned}\frac{\partial E(w)}{\partial w}&=X^TXw-X^TT=0\\w^*&=(X^TX)^{-1}X^TT\end{aligned}$$

(2) 带正则项(惩罚项)的多项式拟合

对于上述误差函数，参数多时往往$w*$具有较大的绝对值，即随着曲线训练程度加深，拟合的曲线会尽可能贴近或穿过训练样本点，但是对于其他点就未必考虑到。因此，为了防止w变化过于剧烈，将w二范数纳入考量，即作为惩罚项：

$$\begin{aligned}E(x)&=\frac{1}{2}\sum_{i=0}^{N}\{y(x_i,w)-t_i\}^2+\frac{\lambda}{2}||w||^2\\&=\frac{1}{2}(Xw-T)^T(Xw-T)+\frac{\lambda}{2}w^Tw\end{aligned}$$

对上式求导可以得到：

$$\begin{aligned}\frac{\partial E(w)}{\partial w}&=X^TXw-X^TT+\lambda w=0\\w^*&=(X^TX+\lambda I)^{-1}X^TT\end{aligned}$$

根据上述可知，是否有正则项参与的区别在于λI这一项，但是也可以不带正则项的解析解中λ为0，因此可以用函数`numpy.linalg.solve`来解得唯一解$w*$，在这里就不再赘述。

3. 梯度下降法

梯度下降方法基于以下的观察：如果实值函数$F(x)$在点a处可微且有定义，那么函数$F(x)$在a点沿着梯度相反方向$-\nabla F(a)$下降最多。因而如果下式成立，

$$b=a-\gamma\nabla F(a)$$

对于$\gamma>0$为一个足够小的值时，有$F(a)\geq F(b)$，从函数F的局部极小值的初始估计x_0开始，有序列$x_0,x_1,x_2,\cdots$，使得$x_{n+1}=x_n-\gamma_n\nabla F(x_n),n\geq 0$。可以得到：

$$F(x_0)\geq F(x_1)\geq F(x_2)\geq\cdots$$

如果可能的话，序列将使函数收敛到局部最小值，其中γ为迭代步长，在本题的算法中初始化为0.01。

算法设计与过程

1. 确定当前位置的损失函数的梯度，对于w向量,其梯度表达式如下，w初始化为0:

$$\frac{\partial}{\partial w}E(w)$$

2. 用步长乘以损失函数的梯度，得到当前位置下降的距离，即$\gamma\frac{\partial}{\partial w}E(w)$。
3. 确定w向量里面的每个值，梯度下降的距离如果小于δ则算法终止（δ初始化为10^{-6}），当前w向量即为最终结果，否则进入步骤4。
4. 更新w向量，其更新表达式如下，迭代次数加一，更新完毕后回到步骤1。

$$w = w - \gamma\frac{\partial}{\partial w}E(w)$$

4. 共轭梯度

共轭梯度法是求解系数矩阵为对称正定矩阵的线性方程组的数值解的方法，共轭梯度法是一个迭代方法，它适用于系数矩阵为稀疏矩阵的线性方程组。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，这保证了每个维度上都是靠近最优的。

算法设计与过程

1. 初始设$p_0 = r_0$，根据残差$r_k = b - Aw_k$去构造下一个循环搜索分享p_k，具体构造时先得到$k+1$步的残差，即$r_{k+1} = r_k - \alpha_k A p_k$，其中$\alpha_k$为:

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

2. 更新$w_{k+1} = w_k + \alpha_k p_k$，如果$r_k^T r_k < \delta$则退出循环并返回迭代次数和得到的$w$。
3. 更新$p_{k+1} = r_k + \beta_k p_k$，其中$\beta_k$有:

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

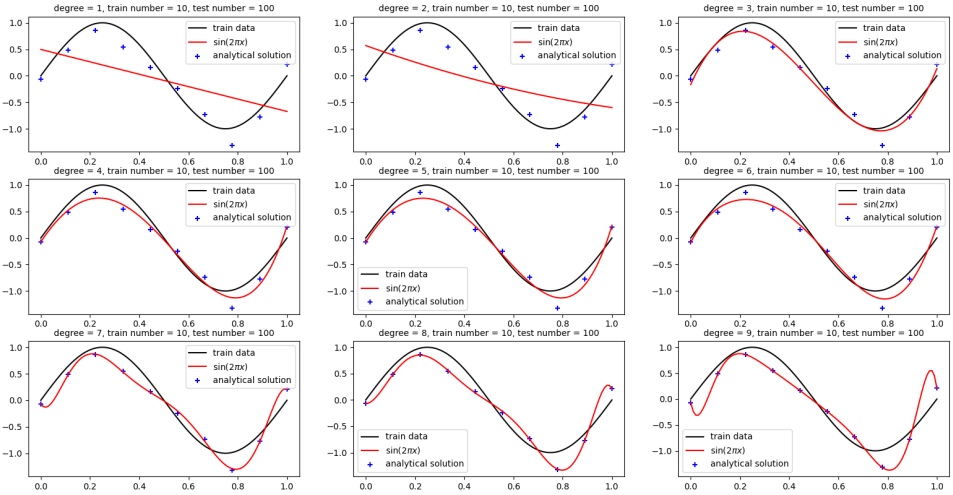
4. 迭代次数加一。

四. 实验结果与分析

1. 解析解

（1）不带正则项的多项式拟合

固定训练样本数量$N = 10$，分别测试阶数从1~9的拟合结果，如下图所示。可以看到在多项式阶数为3时的拟合效果已经很好，继续提高多项式的阶数在4~7的范围内表现也比较优异，但是继续增加阶数，拟合曲线将开始呈现穿过全部训练样本的趋势，直到阶数为9时，拟合曲线穿过全部的样本点。

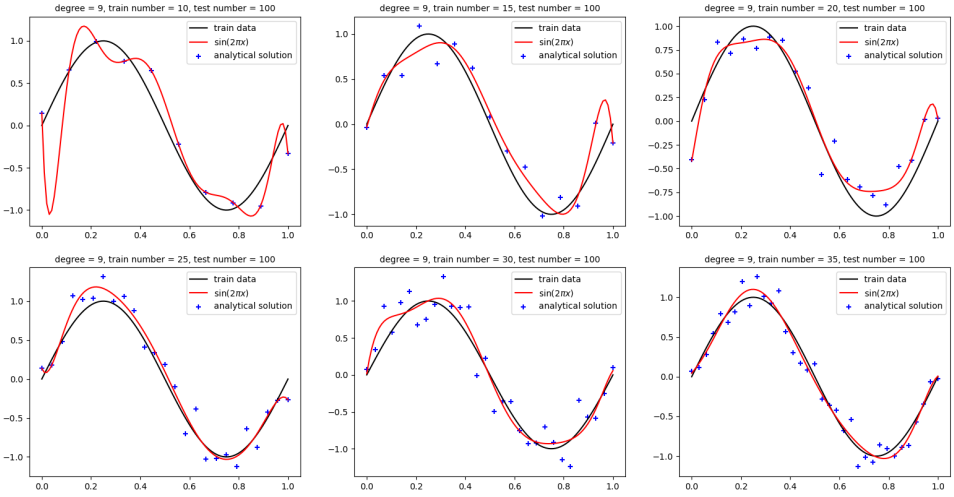


不同阶数时最佳拟合曲线参数w^*的值有（保留两位小数）：

	N=1	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9
w_0^*	0.50	0.57	-0.17	-0.07	-0.08	-8.53e-2	-7.01e-2	-7.11e-2	-7.09e-2
w_1^*	-1.17	-1.67	10.45	7.00	7.47	9.27e+0	-7.08e+0	1.25e-1	-2.20e+1
w_2^*		0.50	-31.44	-13.97	-17.99	-4.02e+1	2.30e+2	8.43e+1	5.95e+2
w_3^*			21.29	-6.73	4.63	1.00e+2	-1.50e+3	-4.04e+2	-5.00e+3
w_4^*				14.01	0.99	-1.83e+2	4.34e+3	2.06e+2	2.18e+4
w_5^*					5.21	1.69e+2	-6.43e+3	2.18e+3	-5.67e+4
w_6^*						-5.47e+1	4.73e+3	-5.31e+3	9.11e+4
w_7^*							-1.37e+3	4.78e+3	-8.87e+4
w_8^*								-1.54e+3	4.79e+4
w_9^*									-1.10e+4

当模型学习到训练样本中的细节和噪声（这里是将$sin(2\pi x)$加入的高斯噪音都拟合得很好），以至于它对模型的性能产生负面影响时，就会发生过度拟合。这意味着训练数据中的噪声或随机波动被模型作为概念提取和学习。问题是这些概念不适用于其他数据，并对模型的泛化能力产生负面影响。在多项式模型中，Overfitting的本质原因在于，当阶数变大时，多项式模型的复杂度和拟合的能力都增强，因此多余的学习能力将增加的高斯噪音都包含其中，会产生极大或极小的系数，如上表所示。

对于在学习目标函数时具有更大灵活性的非参数和非线性模型，过拟合的可能性较大。因此非参数机器学习算法还包含用于限制和约束模型学习多少细节的参数或技术，在这里表现为下一部分将要描述的正则项作为惩罚项来避免过拟合；当然还可以增加训练样本的数量来解决过拟合，如下图所示。将训练样本数量从10增加到30，拟合曲线的过拟合现象趋于逐步缓解。



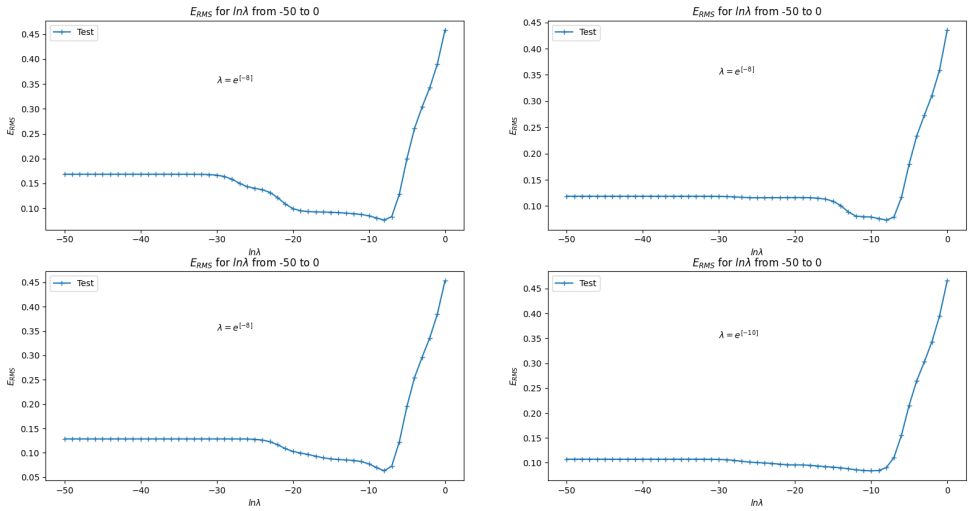
（2）带正则项(惩罚项)的多项式拟合

对于带正则项(惩罚项)的解析解，首先要确定w二范数系数超参数λ的最优值，在这里要考虑到在生成训练样本数据时，将高斯噪音的均值设置为0、方差设置为0.2。从(e^{-50}, e^{0})范围内，设计10000次最高阶数九阶，训练样本数量为20的实验，统计每次测试样本均根方差$E(RMS)$最小值时，$ln\lambda$的取值，即从-50到0，如下表所示在这有限次数的统计内，$ln\lambda$取值在范围$(-9, -7)$内取得最优解（即$E(RMS)$最小）的概率最大，尤其是当$ln\lambda$取-8时多项式拟合最好，最优的超参数为$\lambda = e^{-8}$。

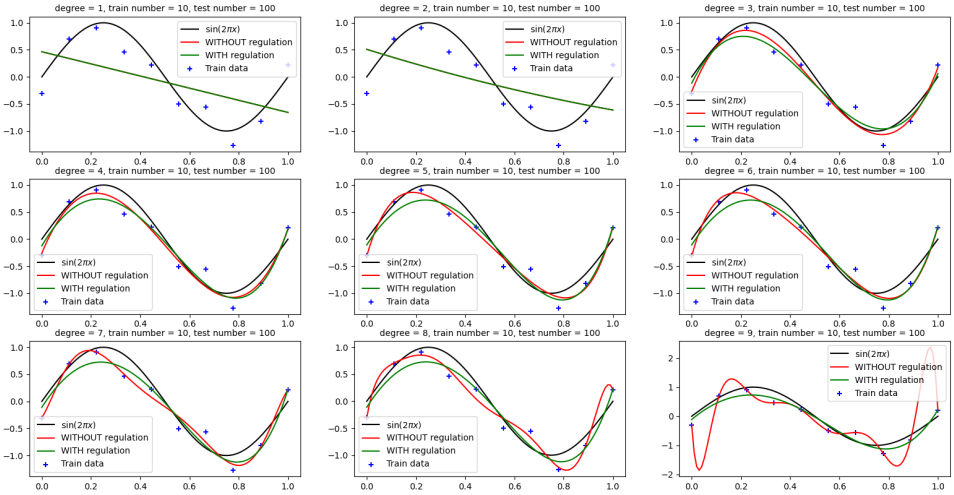
$$E_{RMS} = \sqrt{\frac{2E(w^*)}{N}}$$

$ln\lambda$	**0**	**-1**	**⋯**	**-5**	**-6**	**-7**	**-8**	**-9**	**-10**	**-11**	**-12**	**-13**	**⋯**
次数	0	0	⋯	0	7	1117	2418	1797	731	468	803	771	⋯

随机选取上述实验中的四次，显然范围(e^{-50}, e^{-30})内，均根方差$E(RMS)$保持相对稳定的值，接下来在缓慢下降，一直到(e^{-10}, e^{-5})范围取得最小值，即错误率最小，接着$E(RMS)$又会急剧升高。



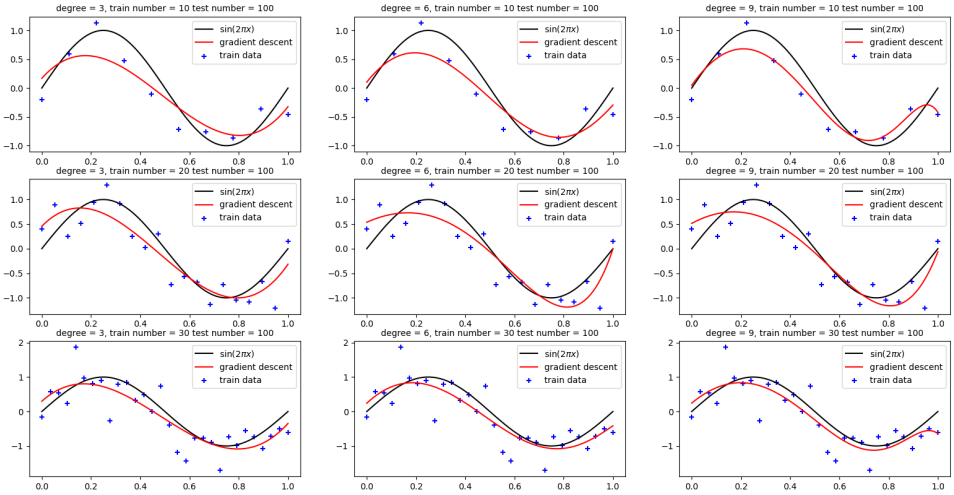
基于此，选择超参数$\lambda = e^{-8}$来拟合多项式，并与不带惩罚项的解析解对比，如下所示，拟合曲线的过拟合现象在一定程度上得到了解决，并且过拟合程度越大发挥的效果越好：



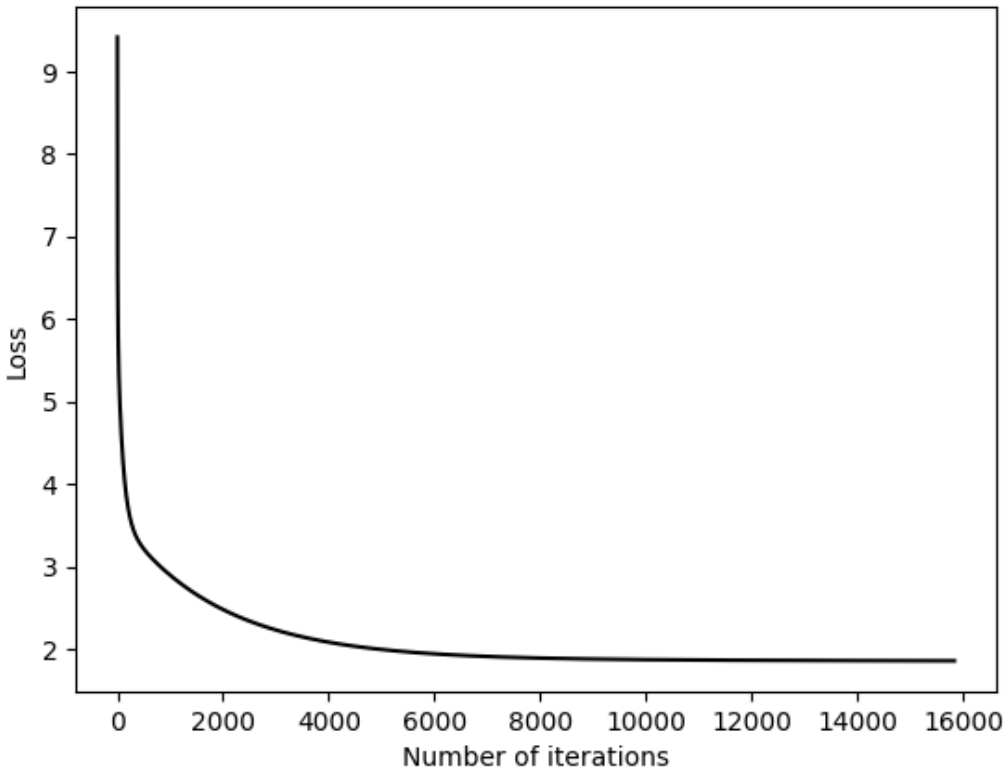
2. 梯度下降法求最优解

利用梯度下降法进行多次实验，对比不同**训练样本数、多项式最高阶数**的训练结果，结果得到的迭代次数、w以及图例如下所示（在这里，将学习率设置为0.01，当小于10^{-6}时认为趋于零，以及惩罚项系数的两倍$\lambda = e^{-8}$）。可以看到固定阶数时随着训练样本数的增加，迭代次数逐步减少；同一样本数随着最高阶数的增加，迭代次数先增后减；结果与之前不带惩罚项的解析解相比，多项式系数都较小，或者说在一个正常范围内，几乎没有过拟合现象。如图所示，随着最高阶数的增加，其实曲线最后呈现的形态等都是很相似的，但是过高的阶数的确对其有一点的影响；而随着训练样本的增加，曲线越会贴近$sin(2\pi x)$。

样本数	degree	迭代次数	w（保留两位小数）
10	3	90045	[0.83 4.16 -19.35 14.47]
10	6	112573	[0.77 4.30 -15.15 1.11 8.76 5.21 -4.81]
10	9	76421	[0.72 4.84 -14.25 -2.99 5.39 7.38 5.38 1.71 -2.26 -5.86]
20	3	85651	[0.18 9.39 -30.20 20.47]
20	6	111351	[0.18 7.97 -19.78 6 -0.03 9.28 6.89 -4.53]
20	9	62063	[0.14 8.00 -16.50 -5.60 3.51 7.97 8.07 4.62 -1.39 -9.09]
30	3	67336	[0.24 7.88 -25.72 17.61]
30	6	70739	[0.32 5.23 -12.01 -2.83 5.07 5.25 -0.77]
30	9	44252	[0.28 5.46 -10.90 -5.31 2.10 5.70 5.69 3.24 -0.70 -5.48]



接下来，实际上还对比了初始化w_0不同，得到的结果有什么异同，我选取了w_0全为0或者全为1，最后两条曲线之间的距离很小，几乎重合，但是放大之后还是存在客观上的一定差异，但是都很微小，因此这里就不再赘述。以下给出一组随机数据`loss`随迭代次数的变化，可以看到随着迭代次数增加，`loss`会趋于稳定：

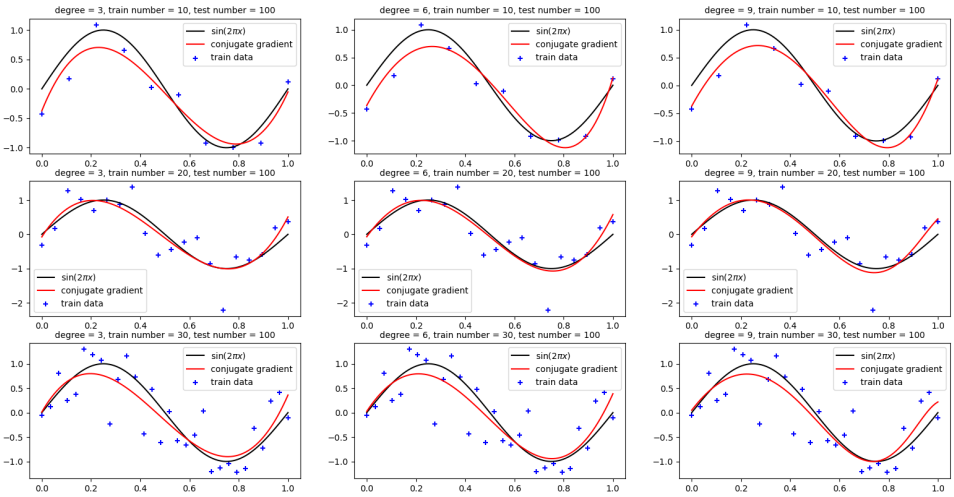


在这里，还应该同时考虑一阶导数的变化带来的影响，在这里可以认为当一阶导数收敛的时候才收敛，由此引进向量（矩阵）范数衡量两个向量（矩阵）之间的距离，同样，我们还可以利用向量（矩阵）范数定义向量（矩阵）序列收敛。

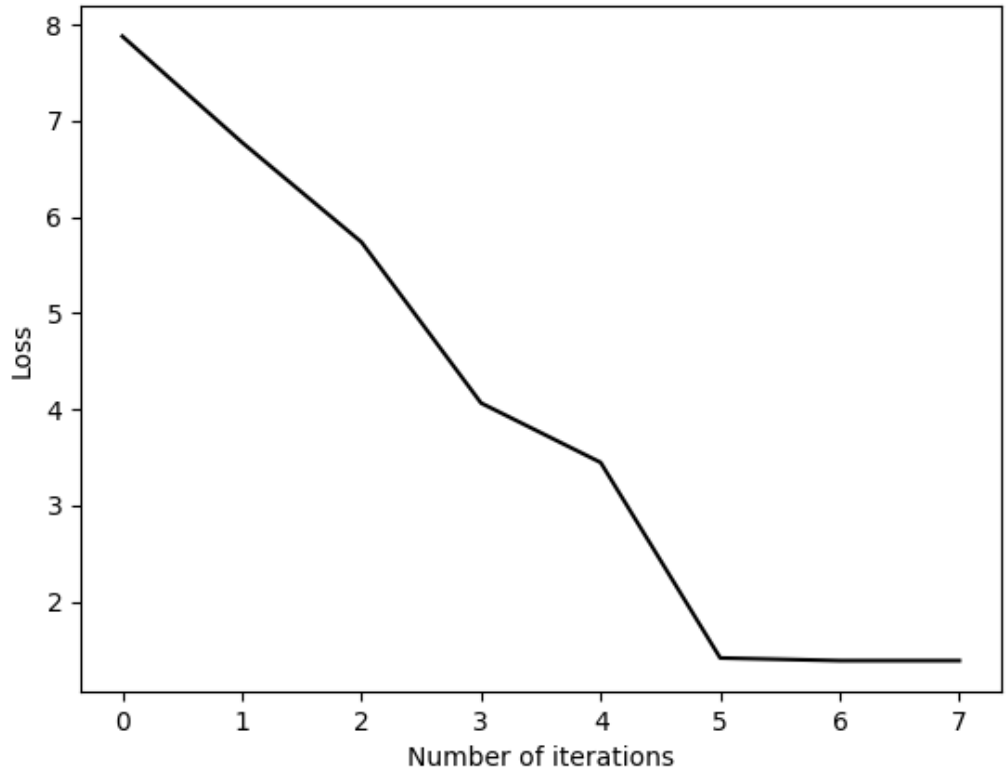
设 $\mathbb{C}^n$ 上的向量序列 $\{x_k\}$ 以及向量 x_0 满足：对给定的向量范数 $\|\bullet\|_v$，$\lim_{k\to\infty}\|x_k-x_0\|_v=0$，则称序列 $\{x_k\}$ 依范数 $\|\bullet\|_v$ 收敛于 x_0，并记为 $\lim_{k\to\infty}x_k=x_0$。在这里我取一阶范数，即`loss`一阶导向量与上一次循环一阶导向量的差中的每个元素的绝对值都小于给定的$\delta=10^{-6}$。

3. 共轭梯度法求最优解

样本数	degree	迭代次数	w（保留两位小数）
10	3	4	[-0.37 10.35 -28.96 18.93]
10	6	5	[-0.36 8.12 -15.35 -2.32 5.39 4.96 -0.30]
10	9	7	[-0.37 8.08 -14.37 -3.65 3.56 4.95 3.49 1.28 -0.71 -2.14]
20	3	4	[-0.08 11.31 -34.77 24.04]
20	6	5	[-0.07 9.98 -24.46 1.76 12.31 7.57 -6.51]
20	9	7	[-0.076 9.41 -20.57 -2.02 6.28 7.19 5.15 2.07 -1.49 -5.47]
30	3	4	[1.26e-02 8.73e+00 -2.79e+01 1.95e+01]
30	6	5	[0.03 7.57 -19.31 0.70 10.70 6.92 -6.22]
30	9	7	0.05 6.57 -14.04 -3.30 2.86 5.54 6.23 4.63 -0.03 -8.28]

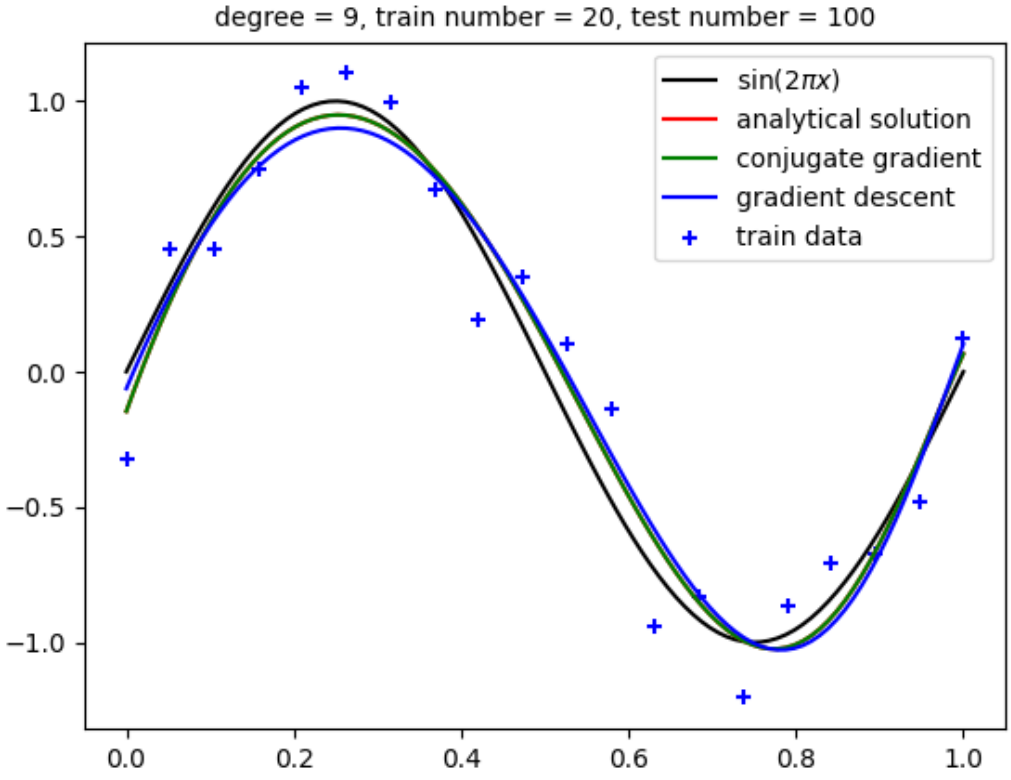


客观上迭代次数相对于梯度下降法减少了很多，降低至了个位数，因此收敛会快很多。相对于样本数，多项式最高阶数对迭代次数影响更为巨大，阶数越大迭代次数越多，几乎决定了迭代次数；并且而对于共轭梯度下降，需要的迭代次数均不超过解空间的维度（即最高阶数加一）。随机选取一组数据`loss`随迭代次数的变化，可以看到随着迭代次数增加，`loss`会急剧下降：

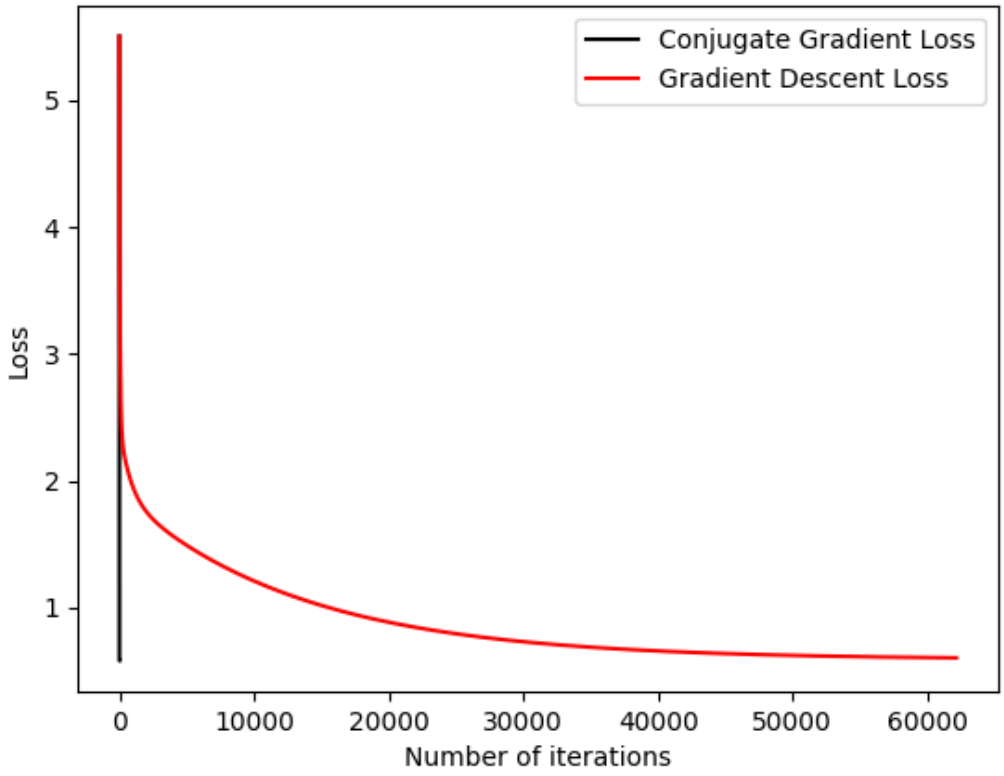


4. 对比

不同方法得到的拟合结果：



对比梯度下降和共轭梯度两类方法loss：



五. 结论

1. 对于过拟合问题，可以增加训练样本数量和增加惩罚项（在这里是二项范数）来解决，因为模型复杂度与模型参数个数和模型参数的绝对值（范数）有关，在这里过少的训练样本和过高的多项式阶数都会导致过拟合。
2. 相对于梯度下降法，共轭梯度迭代次数少，收敛快并且拟合效果更好。

六. 参考文献

[1] https://blog.csdn.net/songyunli1111/article/details/79322103

[2] Gradient descent - Wikipedia

[3] Conjugate gradient method - Wikipedia

[4] https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

七. 附录：源代码（带注释）

数据处理：generate_data.py

```
import numpy as np

def geneData(number, loc, scale):
    """ 生成训练样本数据
    Args:
        number: 在[0,1]范围内选取的数据个数
        loc: 高斯分布均值
        scale: 高斯分布的标准偏差
```

```
    Returns:
        Xn: 在[0,1]范围内均匀分布的一维数组
        T: sin(2πx)，高斯噪音标准差为scale，均值为loc
    """
    Xn = np.linspace(0, 1, num=number)
    noise = np.random.normal(loc=loc, scale=scale, size=len(Xn))
    T = np.sin(2 * np.pi * Xn) + noise
    return Xn, T


def generateX(row, degree):
    """ 将一维数组 row 转化为len(row) * (degree + 1)矩阵X
    Args:
        row: 一维数组，样本横坐标
        degree: 多项式最高度数
    Returns:
        len(row) * (degree + 1) 的矩阵，每一行表示 row 中每个元素的0到degree次幂
    """
    X = np.empty((len(row), degree + 1), dtype=np.double)
    pow = np.arange(0, degree+1)
    for i in range(len(row)):
        row_i = np.ones(degree + 1) * row[i]
        row_i = np.power(row_i, pow)
        X[i] = row_i
    return X
```

解析解 analytical_solution.py

```
"""
    解析解求解两种loss的最优解（无正则项和有正则项）
"""
import numpy as np
import matplotlib.pyplot as plt
from generate_data import *

class AnalyticalSolution(object):
    def __init__(self, X, T, lamda=0):
        self.X = X
        self.T = T
        self. lamda = lamda

    def fit(self):
        """ 带正则项的解析解
        Args:
            X: len(row) * (degree + 1) 的矩阵，每一行是每个元素的0到degree次幂
            T: 目标值的列向量
            lamda: 正则项系数的二倍，若不带正则项
        Returns:
            w* = (X'X+lanmda I)^(-1)X'T      带正则项
            w* = (X'X)^(-1)X'T               不带正则项
        """
        # return np.linalg.inv(self.lamda * np.identity(len(self.X.T))
        #                      + self.X.T @ self.X) @ self.X.T @ self.T
        # wrong answer
        return np.linalg.solve(self.lamda * np.identity(len(self.X.T)) +
                               self.X.T @ self.X, self.X.T @ self.T)
```

```
    def E_rms(self, y_true, y_pred):
        """ 计算E_RMS
        """
        return np.sqrt(np.mean(np.square(y_true - y_pred)))


if __name__ == '__main__':
    """
        增加训练样本数量来解决过拟合问题
    """
    degree = 9
    for index in range(1, 7):
        number_train = 10 + 5 * (index - 1)  # 训练样本的数量
        number_test = 100  # 测试样本的数量
        # 训练样本数据生成
        xn_train, T_train = geneData(number_train, 0.0, 0.2)
        # 测试样本数据生成
        xn_test = np.linspace(0, 1, number_test)
        T_test = np.sin(2 * np.pi * xn_test)
        # 生成训练、测试样本相关X
        X_train = generateX(xn_train, degree)
        X_test = generateX(xn_test, degree)
        plt.subplot(2, 3, index)
        # 训练数据点图
        plt.scatter(xn_train, T_train, marker="+", color="b")
        # 测试数据图
        plt.plot(xn_test, T_test, color="k")
        # 无惩罚项（正则项）的解析解
        fit_without_regular_term = AnalyticalSolution(X_train, T_train)
        w = fit_without_regular_term.fit()
        # 拟合结果图
        plt.plot(xn_test, np.dot(X_test, w), "r")
        plt.legend(labels=["train data","$\sin(2\pi x)$","analytical solution"],
loc='best')
        plt.title("degree = " + str(degree) + ", train number = "+
                  str(number_train)+ ", test number = 100", fontsize=
"medium")
    plt.show()


    """
        无惩罚项，N=10，阶数从1到9的拟合结果
    """
    number_train = 10  # 训练样本的数量
    number_test = 100  # 测试样本的数量

    # 训练样本数据生成
    xn_train, T_train = geneData(number_train, 0.0, 0.2)
    # 测试样本数据生成
    xn_test = np.linspace(0, 1, number_test)
    T_test = np.sin(2 * np.pi * xn_test)

    rms = []
    # 从最高次数1到9进行拟合
    for degree in range(1, 10):
        # 生成训练、测试样本相关X
        X_train = generateX(xn_train, degree)
```

```
        X_test = generateX(xn_test, degree)
        plt.subplot(3, 3, degree)
        # 训练数据点图
        plt.scatter(xn_train, T_train, marker="+", color="b")
        # 测试数据图
        plt.plot(xn_test, T_test, color="k")
        # 无惩罚项（正则项）的解析解
        fit_without_regular_term = AnalyticalSolution(X_train, T_train)
        w = fit_without_regular_term.fit()
        Y = np.dot(X_test, w)
        rms.append(fit_without_regular_term.E_rms(Y, T_test))
        # 拟合结果图
        plt.plot(xn_test, np.dot(X_test, w), "r")
        plt.legend(labels=["train data","$\sin(2\pi x)$","analytical solution"],
loc='best')
        plt.title("degree = " + str(degree) + ", train number = 10, test number
= 100", fontsize= "medium")
        print(w)

    print(rms)
    plt.show()


    """
        对于带正则项（惩罚项）的解析解，统计10000次实验得到最优lambda
        此处所加的高斯噪音均值为0，方差为0.2
    """
    number_train = 20  # 训练样本的数量
    number_test = 100  # 测试样本的数量
    degree = 9
    # 收集lamda可能大小的量
    buckets = np.zeros(51, dtype=np.int32)

    # 解多次，得到可能的最优lamda
    for i in range(10000):
        # 训练样本数据生成
        xn_train, T_train = geneData(number_train, 0.0, 0.2)
        # 测试样本数据生成
        xn_test = np.linspace(0, 1, number_test)
        T_test = np.sin(2 * np.pi * xn_test)
        # 生成训练、测试样本相关X
        X_train = generateX(xn_train, degree)
        X_test = generateX(xn_test, degree)
        # 测试lamda为e^(-50)到e^0的效果
        lamda_to_test = range(-50, 1)
        RMS_lamda = []
        for ex in lamda_to_test:
            fit_with_regular_term = AnalyticalSolution(X_train, T_train,
np.exp(ex))
            w = fit_with_regular_term.fit()
            Y = np.dot(X_test, w)
            RMS_lamda.append(fit_with_regular_term.E_rms(Y, T_test))
        temp = np.array(lamda_to_test) # 将range转化为array类型，以求取最值索引
        best_ln_lamda = temp[np.where(RMS_lamda == np.min(RMS_lamda))]
        buckets[abs(best_ln_lamda)] = buckets[abs(best_ln_lamda)] + 1
    print(buckets)
```

```
    """
        多次随机训练，寻找最优超参数
    """
    for index in range(1, 5):
        number_train = 20  # 训练样本的数量
        number_test = 100  # 测试样本的数量
        degree = 9
        # 训练样本数据生成
        xn_train, T_train = geneData(number_train, 0.0, 0.2)
        # 测试样本数据生成
        xn_test = np.linspace(0, 1, number_test)
        T_test = np.sin(2 * np.pi * xn_test)
        # 生成训练、测试样本相关X
        X_train = generateX(xn_train, degree)
        X_test = generateX(xn_test, degree)

        plt.subplot(2, 2, index)
        # 测试lamda为e^(-50)到e^0的效果
        lamda_to_test = range(-50, 1)
        RMS_lamda = []
        for ex in lamda_to_test:
            fit_with_regular_term = AnalyticalSolution(X_train, T_train,
np.exp(ex))
            w = fit_with_regular_term.fit()
            Y = np.dot(X_test, w)
            RMS_lamda.append(fit_with_regular_term.E_rms(Y, T_test))
        temp = np.array(lamda_to_test) # 将range转化为array类型，以求取最值索引
        best_ln_lamda = temp[np.where(RMS_lamda == np.min(RMS_lamda))]
        annotate = "$\lambda = e^{" + str(best_ln_lamda) + "}$"
        plt.ylabel("$E_{RMS}$")
        plt.xlabel("$ln \lambda$")
        plt.annotate(annotate, xy=(-30, 0.35))
        plt.plot(lamda_to_test, RMS_lamda, '+-', label="Test")
        plt.title("$E_{RMS}$ for $ln \lambda$ from -50 to 0")
        plt.legend()
    plt.show()


    """
        对比有无惩罚项的拟合结果
    """
    number_train = 10  # 训练样本的数量
    number_test = 100  # 测试样本的数量

    # 训练样本数据生成
    xn_train, T_train = geneData(number_train, 0.0, 0.2)
    # 测试样本数据生成
    xn_test = np.linspace(0, 1, number_test)
    T_test = np.sin(2 * np.pi * xn_test)

    # 从最高次数1到9进行拟合
    for degree in range(1, 10):
        # 生成训练、测试样本相关X
        X_train = generateX(xn_train, degree)
        X_test = generateX(xn_test, degree)
        plt.subplot(3, 3, degree)
        # 训练数据点图
```

```
        plt.scatter(xn_train, T_train, marker="+", color="b", label="Train
data")
        # 测试数据图
        plt.plot(xn_test, T_test, color="k", label="$\sin(2\pi x)$")
        # 无惩罚项（正则项）的解析解
        fit_without_regular_term = AnalyticalSolution(X_train, T_train)
        fit_with_regular_term = AnalyticalSolution(X_train, T_train, np.exp(-8))
        w = fit_without_regular_term.fit()
        w_lamda = fit_with_regular_term.fit()
        # Y = np.dot(X_test, w)
        # rms.append(fit_without_regular_term.E_rms(Y, T_test))
        # 拟合结果图
        plt.plot(xn_test, np.dot(X_test, w), "r", label="WITHOUT regulation")
        plt.plot(xn_test, np.dot(X_test, w_lamda), "g", label="WITH regulation")
        plt.legend(loc='best')
        plt.title("degree = " + str(degree) + ", train number = 10, test number
= 100", fontsize= "medium")
        print(w)
    plt.show()
```

梯度下降法 gradient_descent.py

```
"""
    优化方法求解最优解（梯度下降法）
"""
import numpy as np
import matplotlib.pyplot as plt
from generate_data import *

class GradientDescent(object):
    def __init__(self, X, T, w_0, lamda=np.exp(-8), rate=0.01, delta=1e-6):
        """ 用于多项式函数使用梯度下降拟合初始化
        Args:
            X: len(row) * (degree + 1) 的矩阵，每一行是每个元素的0到degree次幂
            T: 目标值的列向量
            w_0: 初始解，通常以全零向量
            lamda: 正则项系数部分，之前求得最合适的范围在1e-7或者1e-8之间，默认为1e-8
            rate: 学习率，初始为0.5
            delta: 精度要求，初始为1e-6，当小于这个值时认为趋于0
        """
        self.X = X
        self.T = T
        self.w_0 = w_0
        self.lamda = lamda
        self.rate = rate
        self.delta = delta

    def loss(self, w):
        temp = self.X @ w - self.T
        return 0.5 * np.mean(temp.T @ temp + self.lamda * w.T @ w)

    def __derivative(self, w):
        """ 一阶函数导数
        """
        return self.X.T @ self.X @ w + self.lamda * w - self.X.T @ self.T
```

```
    def fit(self):
        """ 多项式函数使用梯度下降拟合
        Returns:
            w: 梯度下降优化得到的最优解
        """
        losses = []
        loss_0 = self.loss(self.w_0)
        losses.append(loss_0)
        k = 0
        w = self.w_0
        der_0 = self.__derivative(w)
        while True:
            # der_0 = der
            der = self.__derivative(w)
            wk = w - self.rate * der
            loss = self.loss(wk)
            losses.append(loss)
            # 增加条件，当一阶导数收敛（在这里取序列收敛性）时，才认为函数收敛
            max, min = (der - der_0).max(), (der - der_0).min()
            if np.abs(loss - loss_0) < self.delta and np.abs(max) < self.delta
and np.abs(min) < self.delta:
            # if np.abs(loss - loss_0) < self.delta and (der - der_0) <
self.delta:
                # print(der, der_0)
                break
            else:
                k = k + 1
                loss_0 = loss
                w = wk
                der_0 = der
        return k, w, losses


if __name__ == "__main__":

    # num = 0
    # for index in range(1, 4):
    #     number_train = 10 * index  # 训练样本的数量
    #     number_test = 100  # 测试样本的数量

    #     # 训练样本数据生成
    #     xn_train, T_train = geneData(number_train, 0.0, 0.4)
    #     # 测试样本数据生成
    #     xn_test = np.linspace(0, 1, number_test)
    #     T_test = np.sin(2 * np.pi * xn_test)

    #     degrees = [3, 6, 9]
    #     for degree in degrees:

    #         num += 1
    #         # 生成训练、测试样本相关X
    #         X_train = generateX(xn_train, degree)
    #         X_test = generateX(xn_test, degree)
    #         gradient_descent = GradientDescent(X_train, T_train,
np.zeros(degree + 1))
    #         k, w, losses = gradient_descent.fit()
    #         print("training number: "+ str(number_train) + " degree: "
+str(degree) +
```

```
    #                 " 迭代次数: "+ str(k) + "\n w: " + str(w))

    #         plt.subplot(3, 3, num)
    #         # 训练数据点图
    #         plt.scatter(xn_train, T_train, marker="+", color="b", label="train
data")
    #         # 测试数据图
    #         plt.plot(xn_test, T_test, color="k", label="$\sin(2\pi x)$")
    #         # 拟合结果图
    #         plt.plot(xn_test, np.dot(X_test, w), color="r", label="gradient
descent")
    #         plt.legend(loc='best')
    #         plt.title("degree = " + str(degree) + ", train number = " +
str(number_train) +
    #                     " test number = 100", fontsize= "medium")
    # plt.show()


    """
        对比不同w_0给梯度下降结果带来的差异
    """
    number_train = 20  # 训练样本的数量
    number_test = 100  # 测试样本的数量

    # 训练样本数据生成
    xn_train, T_train = geneData(number_train, 0.0, 0.4)
    # 测试样本数据生成
    xn_test = np.linspace(0, 1, number_test)
    T_test = np.sin(2 * np.pi * xn_test)

    degree = 6
    # 生成训练、测试样本相关X
    X_train = generateX(xn_train, degree)
    X_test = generateX(xn_test, degree)
    conjugate_gradient_0 = GradientDescent(X_train, T_train, np.zeros(degree +
1))
    # conjugate_gradient_1 = GradientDescent(X_train, T_train, np.ones(degree +
1))
    k_0, w_0, losses_0 = conjugate_gradient_0.fit()
    # k_1, w_1, losses_1 = conjugate_gradient_1.fit()
    print(k_0, w_0)

    # 训练数据点图
    plt.scatter(xn_train, T_train, marker="+", color="b", label="train data")
    # 测试数据图
    plt.plot(xn_test, T_test, color="k", label="$\sin(2\pi x)$")
    # 拟合结果图
    plt.plot(xn_test, np.dot(X_test, w_0), color="r", label="$w_0 = 0$")
    # plt.plot(xn_test, np.dot(X_test, w_1), color="g", label="$w_0 = 1$")
    plt.legend(loc='best')
    plt.title("degree = " + str(degree) + ", train number = 10, test number =
100", fontsize= "medium")
    plt.show()

    plt.plot(losses_0, color="k", label="Gradient Descent Loss")
    plt.xlabel("Number of iterations")
    plt.ylabel("Loss")
    plt.legend(loc='best')
```

```
    plt.show()
```

共轭梯度 conjugate_gradient.py

```
"""
    优化方法求解最优解（共轭梯度法）
"""
import numpy as np
import matplotlib.pyplot as plt
from generate_data import *

class ConjugateGradient(object):
    def __init__(self, X, T, w_0, lamda=np.exp(-8), delta=1e-6):
        """ 共轭梯度初始化
        Args:
            X: len(row) * (degree + 1) 的矩阵，每一行是每个元素的0到degree次幂
            T: 目标值的列向量
            w_0: 一般假设初始化为0
            lamda: 正则项系数部分，之前求得最合适的范围在1e-7或者1e-8之间，初始化为e^(-8)
            delta: 精度要求，初始为1e-6，当小于这个值时认为趋于0
        """
        self.X = X
        self.T = T
        self.w_0 = w_0
        self.lamda = lamda
        self.delta = delta
        # A = X'X + lamda I
        self.A = X.T @ X + np.identity(len(X.T)) * lamda
        # b = X'X
        self.b = X.T @ T

    def loss(self, w):
        temp = self.X @ w - self.T
        return 0.5 * np.mean(temp.T @ temp + self.lamda * w.T @ w)

    def fit(self):
        w = self.w_0
        r_0 = self.b - self.A @ self.w_0
        p = r_0
        k = 0
        losses = []
        losses.append(self.loss(w))
        while True:
            alpha = (r_0.T @ r_0) / (p.T @ self.A @ p)
            w = w + alpha * p
            r = r_0 - alpha * self.A @ p
            if r_0.T @ r_0 < self.delta:
                break
            beta = (r.T @ r) / (r_0.T @ r_0)
            p = r + beta * p
            r_0 = r
            k += 1
            losses.append(self.loss(w))
        return k, w, losses


if __name__ == "__main__":
```

```
    number_train = 20  # 训练样本的数量
    number_test = 100  # 测试样本的数量

    # 训练样本数据生成
    xn_train, T_train = geneData(number_train, 0.0, 0.4)
    # 测试样本数据生成
    xn_test = np.linspace(0, 1, number_test)
    T_test = np.sin(2 * np.pi * xn_test)

    degree = 9
    # 生成训练、测试样本相关X
    X_train = generateX(xn_train, degree)
    X_test = generateX(xn_test, degree)
    conjugate_gradient = ConjugateGradient(X_train, T_train, np.zeros(degree +
1))
    k, w, losses = conjugate_gradient.fit()
    print(k,w)

    # 训练数据点图
    plt.scatter(xn_train, T_train, marker="+", color="b", label="train data")
    # 测试数据图
    plt.plot(xn_test, T_test, color="k", label="$\sin(2\pi x)$")
    # 拟合结果图
    plt.plot(xn_test, np.dot(X_test, w), color="r", label="conjugate gradient")
    plt.legend(loc='best')
    plt.title("degree = " + str(degree) + ", train number = 10, test number =
100", fontsize= "medium")
    plt.show()

    plt.plot(losses, color="k", label="Conjugate Gradient Loss")
    plt.xlabel("Number of iterations")
    plt.ylabel("Loss")
    plt.legend(loc='best')
    plt.show()

    # num = 0
    # for index in range(1, 4):
    #     number_train = 10 * index  # 训练样本的数量
    #     number_test = 100  # 测试样本的数量

    #     # 训练样本数据生成
    #     xn_train, T_train = geneData(number_train, 0.0, 0.4)
    #     # 测试样本数据生成
    #     xn_test = np.linspace(0, 1, number_test)
    #     T_test = np.sin(2 * np.pi * xn_test)

    #     degrees = [3, 6, 9]
    #     for degree in degrees:
    #         num += 1
    #         # 生成训练、测试样本相关X
    #         X_train = generateX(xn_train, degree)
    #         X_test = generateX(xn_test, degree)
    #         conjugate_gradient = ConjugateGradient(X_train, T_train,
np.zeros(degree + 1))
    #         k, w, losses = conjugate_gradient.fit()
    #         print("training number: "+ str(number_train) + " degree: "
+str(degree) +
```

```
    #                 " 迭代次数: "+ str(k) + "\n w: " + str(w))

    #         plt.subplot(3, 3, num)
    #         # 训练数据点图
    #         plt.scatter(xn_train, T_train, marker="+", color="b", label="train
data")
    #         # 测试数据图
    #         plt.plot(xn_test, T_test, color="k", label="$\sin(2\pi x)$")
    #         # 拟合结果图
    #         plt.plot(xn_test, np.dot(X_test, w), color="r", label="conjugate
gradient")
    #         plt.legend(loc='best')
    #         plt.title("degree = " + str(degree) + ", train number = " +
str(number_train) +
    #                     ", test number = 100", fontsize= "medium")
    # plt.show()
```

对比 comparison.py

```
import numpy as np
import matplotlib.pyplot as plt
from analytical_solution import *
from conjugate_gradient import *
from gradient_descent import *
from generate_data import *


if __name__ == '__main__':
    degree = 9
    number_train = 20 # 训练样本的数量
    number_test = 100  # 测试样本的数量
    # 训练样本数据生成
    xn_train, T_train = geneData(number_train, 0.0, 0.2)
    # 测试样本数据生成
    xn_test = np.linspace(0, 1, number_test)
    T_test = np.sin(2 * np.pi * xn_test)
    # 生成训练、测试样本相关X
    X_train = generateX(xn_train, degree)
    X_test = generateX(xn_test, degree)
    # 训练数据点图
    plt.scatter(xn_train, T_train, marker="+", color="b", label="train data")
    # 测试数据图
    plt.plot(xn_test, T_test, color="k", label="$\sin(2\pi x)$")
    # 无惩罚项（正则项）的解析解
    fit_with_regular_term = AnalyticalSolution(X_train, T_train, np.exp(-8))
    w_AnalyticalSolution = fit_with_regular_term.fit()
    conjugate_gradient = ConjugateGradient(X_train, T_train, np.zeros(degree +
1))
    k, w_ConjugateGradient, losses = conjugate_gradient.fit()
    gradient_descent = GradientDescent(X_train, T_train, np.zeros(degree + 1))
    k_1, w_GradientDescent, losses_1 = gradient_descent.fit()
    # 拟合结果图
    plt.plot(xn_test, np.dot(X_test, w_AnalyticalSolution), color="r",
label="analytical solution")
    plt.plot(xn_test, np.dot(X_test, w_ConjugateGradient), color="g",
label="conjugate gradient")
```

```
    plt.plot(xn_test, np.dot(X_test, w_GradientDescent), color="b",
label="gradient descent")
    plt.legend(loc='best')
    plt.title("degree = " + str(degree) + ", train number = "+
                str(number_train)+ ", test number = "+ str(number_test),
fontsize= "medium")
    plt.show()

    plt.plot(losses, color="k", label="Conjugate Gradient Loss")
    plt.plot(losses_1, color="r", label="Gradient Descent Loss")
    plt.xlabel("Number of iterations")
    plt.ylabel("Loss")
    plt.legend(loc='best')
    plt.show()
```