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： 专业限选

实验题目： 多项式拟合曲线

学号：7203610121
姓名： 刘天瑞

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：

硬件环境：X86-64
操作系统 OS：Windows 11
编译平台：Visual Studio 2022
实现语言：Python 3.9.7

三、设计思想（本程序中的用到的主要算法及数据结构）

3.1 生成数据的算法

主要使用课上的例子：sin(2πx)函数来产生数据样本，其中让自变量 x 均匀分布在[0 , 1] [0, 1]区间，共计 10 个样本点，对于每一个目标值（因变量）t = sin(2πx)增加一个 0 均值，方差为 0.1 的高斯噪声。

```
1     # -*- coding: gbk -*-
2
3     import numpy as np #调用numpy库用于生成矩阵
4     import matplotlib.pyplot as plt #调用matplotlib库中的pyplot用于绘图
5
6     def sin_2pi(x):#打包利用sin（2 pi x）函数
7         return np.sin(2*np.pi*x)
8
9     def get_traindata():#获得训练数据
10        x=np.linspace(0, 1, num = 10)#共计10个样本点，均匀分布在区间[0,1]之间
11        y=sin_2pi(x) + np.random.normal(0, 0.1, 10)#增加高斯噪声，方差选取0.1
12        return x,y
```

图 1

3.2 利用高阶多项式函数生成拟合曲线（不带惩罚/正则项）

利用训练集合，对于每个新的$\hat{x}$，来预测目标值$\hat{t}$。这里采用多项式函数进行学习，即利用（1）式来确定参数 w，假设阶数 m 已知。

$$y(x,w)=w_0+w_1x+...+w_mx^m=\sum_{i=0}^{m}w_ix_i \tag{1}$$

采用最小二乘法来求解析解，即建立误差函数来测量每个样本点目标值 t 与预测函数 y(x,w)之间的误差，误差（代价）函数即为式(2)。

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

将上式写成矩阵形式如式(3)。

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

上式其中 X 为多项式矩阵，w 为系数矩阵，T 为训练数据矩阵：

$$X=\begin{bmatrix}1 & x_1 & ... & x_1^m\\ 1 & x_2 & ... & x_2^m\\ ... & ... & ... & ...\\ 1 & x_N & ... & x_N^m\end{bmatrix},w=\begin{bmatrix}w_0\\ w_1\\ ...\\ w_m\end{bmatrix},T=\begin{bmatrix}t_1\\ t_2\\ ...\\ t_N\end{bmatrix}$$

再将上式通过求导便可以得到如下式（4）：

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

令

$$\frac{\partial E}{\partial w}=0$$

得出式（5），即 w^*：

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

将代价函数写成矩阵形式，求导后求出 w^*，也就是令代价函数最小的 w 的值：

$$w^* = X^{-1}T$$

```
21  def get_params(X, Y):#最小二乘法不带惩罚项，通过公式得到参数w，X为拼接得到的矩阵，Y为样本的y值的矩阵
22      w = np.linalg.inv((X.T).dot(X)).dot(X.T).dot(Y) #公式W* = (X'·X)^(-1) (X')·Y=X^(-1)·Y
23      return w
```

图 2

3.3 带惩罚/正则项的多项式函数拟合曲线

在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，可以通过在优化目标函数式(3)中的代价函数增加 w 的惩罚项，让 w*过大时给代价函数带来负面影响从而避免过拟合，因此便可以得到式（6）：

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

同样可以将式（6）写出了个矩阵形式，从而得到式（7）：

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

同理再对式（7）求导得到式（8）：

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

令

$$\frac{\partial \hat{E}}{\partial w} = 0$$

得出式（9），即为新的 w^*：

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

其中 I 为单位阵，λ 为超参数，需要人为设定值。

```
def get_params_with_penalty(X, Y, l):#最小二乘法带惩罚项，通过公式得到参数w，X为拼接得到的矩阵，Y为样本的y值的矩阵，l为人为设定的超参数
    w = np.linalg.inv((X.T).dot(X)+l*(np.identity(len(X.T)))).dot(X.T).dot(Y) #公式:w* = (X'·X+lI)^(-1) (X')·Y
    return w
```

图 3

3.4 梯度下降法求解最优解

对于 f（x）如果在 x_i 点可微且有定义，顺着梯度为增长最快方向，因此梯度的反方向即为下降最快方向。因而有式（10）对于 α > 0 成立：

$$x_{i+1} = x_i - \alpha \nabla f(x_i) \tag{10}$$

因此，如果可以顺利得到一个 f（x_n）收敛到期望的最小值，对于此次实验很大可能性可以收敛到最小值。

把代价函数按照逆梯度的方向移动，从而达到梯度下降的目的，代价函数变为矩阵的形式为如下式（11）：

$$E(w) = (Xw - T)'(Xw - T) \tag{11}$$

对其求导，得到梯度

$$\frac{\partial E}{\partial w} = 2(X'Xw - X'T) \tag{12}$$

然后将得到的 w 全部初始化为 0，人为设定一个学习率，每次将 w 的值更新为“w-梯度*学习率”，最后得到的 w 就是所要求的。

```
def gradient_descent_fit(X, Y, lr=0.01):#梯度下降法求解最优解，通过公式得到参数w，X为拼接得到的矩阵，Y为样本的y值的矩阵，lr为学习率
    w=([ 0,  0,  0,  0,0,0])#先初始化w
    for i in range(100000):#采用足够多的步数进行计算，也可以用梯度小于某个值进行计算
        Y1=np.dot(X,w)
        grad=np.dot(X.T,Y1-Y)
        w-=grad*lr
    return w
```

图 4

3.5 共轭梯度法求解最优解

共轭梯度法用于解决的主要是形如 Ax=b 的线性方程组解的问题，其中 A 必须是对称的、正定的矩阵。大概来说，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，这与梯度下降方法，每次都选择梯度的反方向去迭代，梯度下降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。

而在本问题中，我们要求解的是

$$X'Xw = T'X$$

w 是我们要解的 x，于是就有 A=$X'X$,b=$T'X$,具体操作如下所示：

1. 初始化 x_0;
2. 初始化 $d_0 = r_0 = b - Ax_0$;
3. 让 $\alpha_k = \frac{p_k^T(b-Ax_k)}{p_k^T Ap_k} = \frac{p_k^T r_k}{p_k^T Ap_k}$;
4. 迭代 $x_{k+1} = x_k + \alpha_k d_k$;
5. 让 $r_{k+1} = r_k - \alpha_k A d_k$;
6. 让 $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$,$d_{k+1} = r_{k+1} + \beta_{k+1} d_k$;
7. 当 $\frac{|r_k|}{|r_0|} < \varepsilon$ 时跳出此循环，否则继续从 3.处开始重新迭代。

基本求解的方法是先猜一个解 x_0，然后取梯度的反方向 $p_0 = b - A x$，在 n 维空间的基中 p_0 要与其与的基共轭并且为初始残差。对于共轭梯度下降，初始时取

$$w_0 = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}$$

由上式（8）（9）可得：

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

令

$$\begin{cases} A = X'X + \lambda I \\ b = X'T \end{cases} \tag{14}$$

通过共轭梯度下降法求解 Ax = b，解得 x，并记录迭代次数 k。

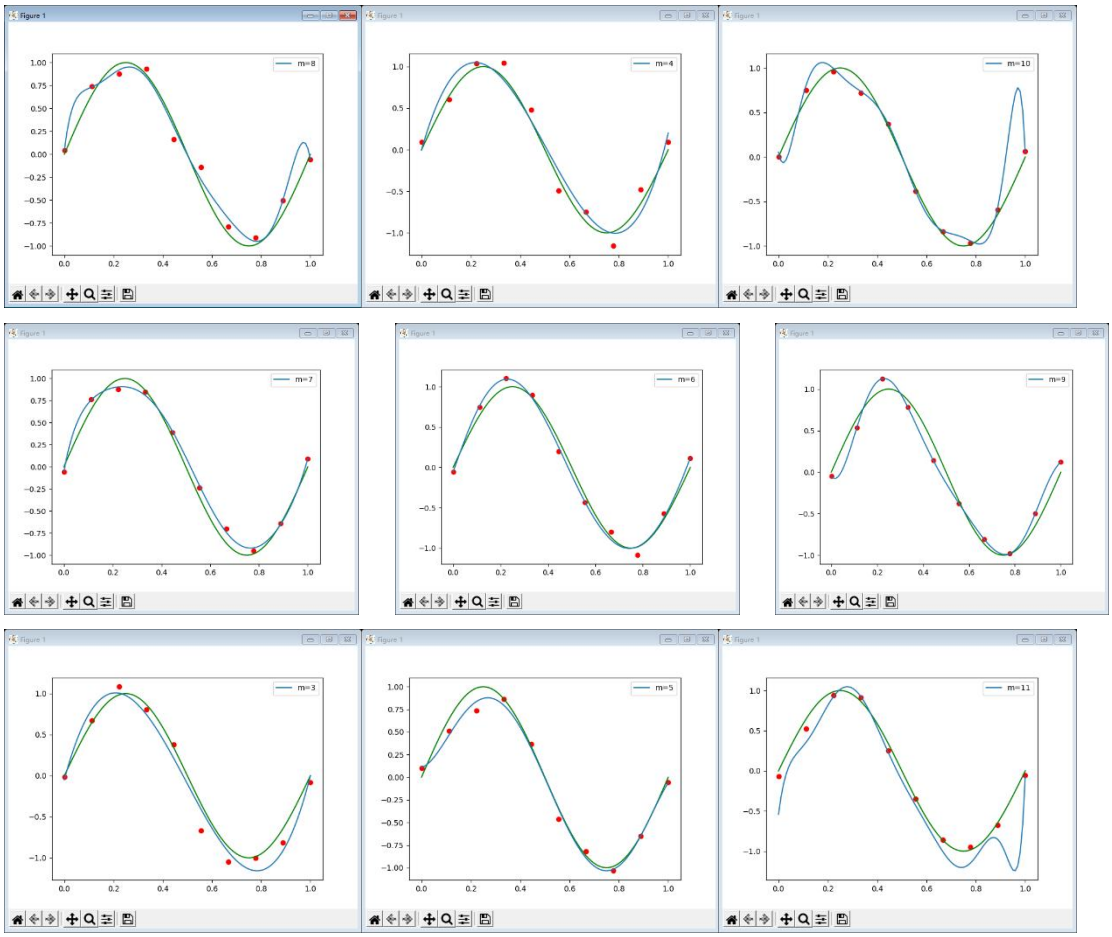
```
def conjugate_gradient_fit(X,Y,m):#共轭梯度法求解最优解，通过公式得到参数w，X为拼接得到的矩阵，Y为样本的y值的矩阵，m为阶数
    A=np.dot(X.T,X)#A=X^T·X，当成A已经正则
    b=np.dot(X.T,Y)#b=X^T·Y
    w=np.zeros(m+1)#初始化w
    ep=0.00001#设一个极小数作为跳出条件
    d=r=b-np.dot(A,w)#按照公式初始化
    r0=r
    while True:#迭代部分
        al=np.dot(r.T,r)/np.dot(np.dot(d,A),d)
        w+=al*d
        r1=r-al*np.dot(A,d)
        be=np.dot(r1.T,r1)/np.dot(r.T,r)
        d=be*d+r1
        r=r1
        if np.linalg.norm(r)/np.linalg.norm(r0)<ep:
            break
    return w
```

图 5

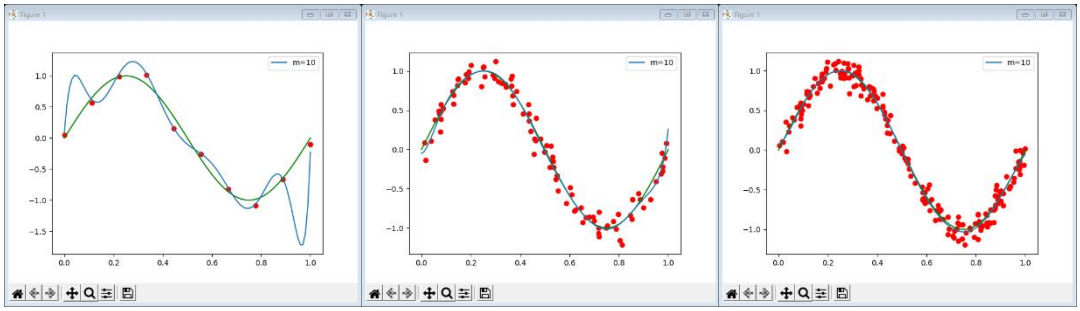
四、 实验结果与分析

4.1 利用多项式函数生产拟合曲线

无惩罚项时生成的拟合曲线如下所示（m 为阶数）：



再对阶数 m = 10 时进行训练数据量的调整：



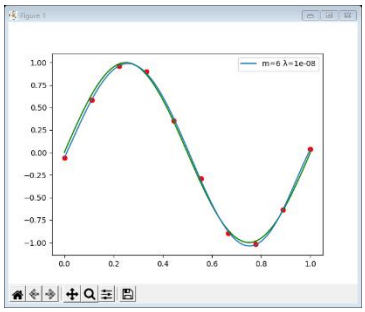
我们可以看到在固定训练样本的大小之后，在多项式阶数 m 为 3 时的拟合效果已经很好了。继续提高多项式的阶数，尤其在阶数为 6 的时候曲线“完美的”经过了所有的节点，这种剧烈的震荡并没有很好的拟合真实的背后的函数 $\sin(2\pi x)$，反而将所有噪声均很好的拟合，即表现出来一种过拟合的情况。

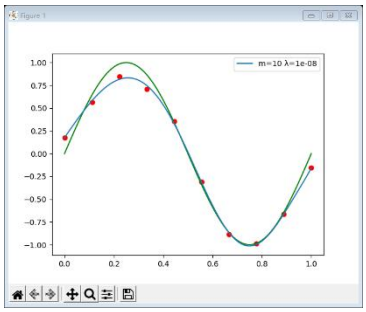
其出现过拟合的本质原因是在阶数过大的情况下，模型的复杂度和拟合的能力都增强，因此可以通过过大或者过小的系数来实现震荡以拟合所有的数据点，以至于甚至拟合了所有的噪声。在这里由于我们的数据样本大小只有 10，所以在阶数为 6 的时候，其对应的系

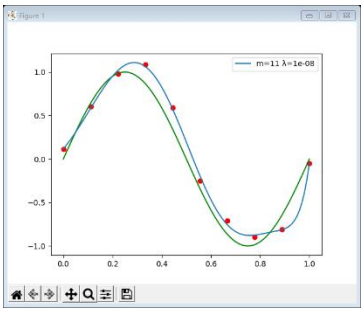
数向量 w 恰好有唯一解，因此可以穿过所有的样本点。

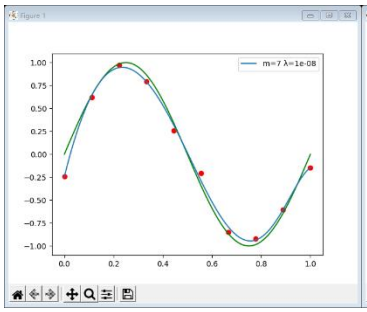
对于过拟合我们可以通过增加样本的数据或者通过增加惩罚项的方式来解决。增加数据集样本数量，使其超过参数向量的大小，就会在一定程度上解决过拟合问题。

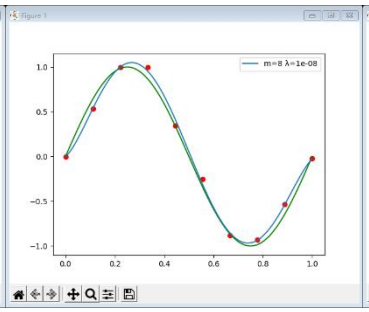
有惩罚项时生成的拟合曲线如下所示：（因为 m 较小时与无惩罚项情况下相差无几，故 m 从 6 开始取，超参数 λ 选择 10^{-8}）

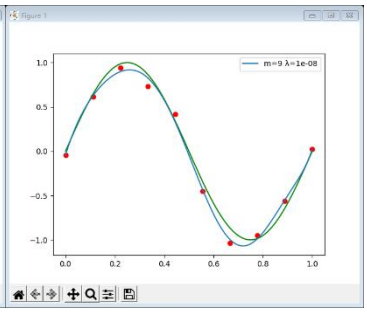




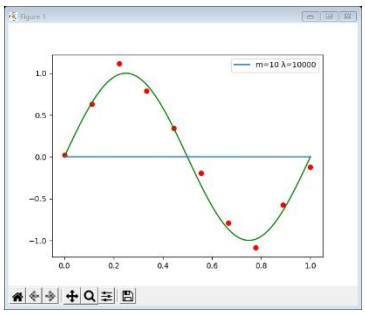


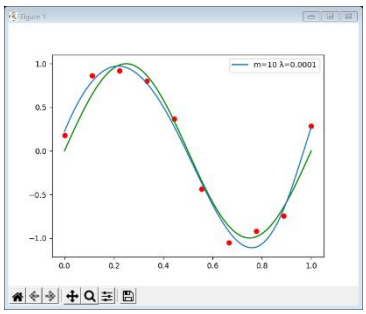


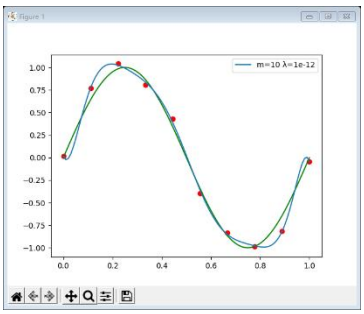


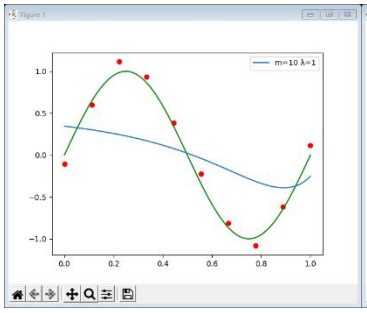


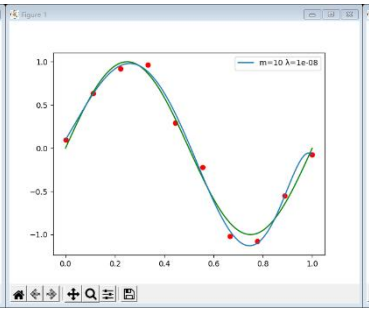
现在对超参数进行更改：（取 m=10）

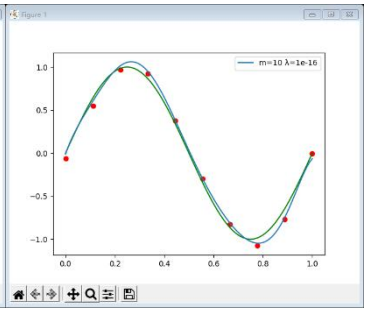




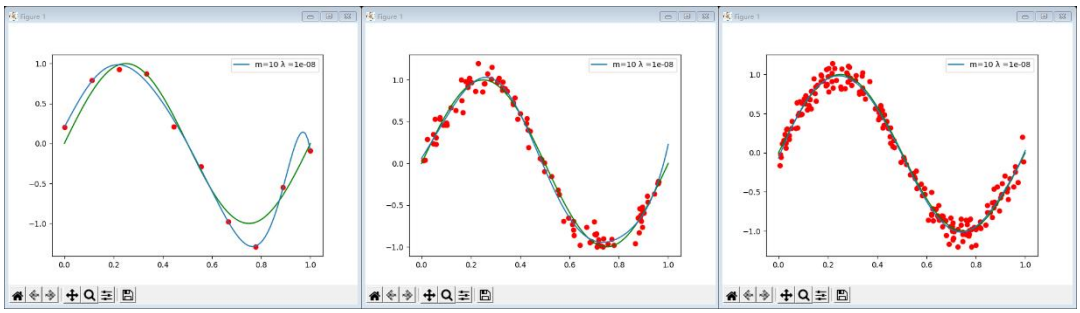








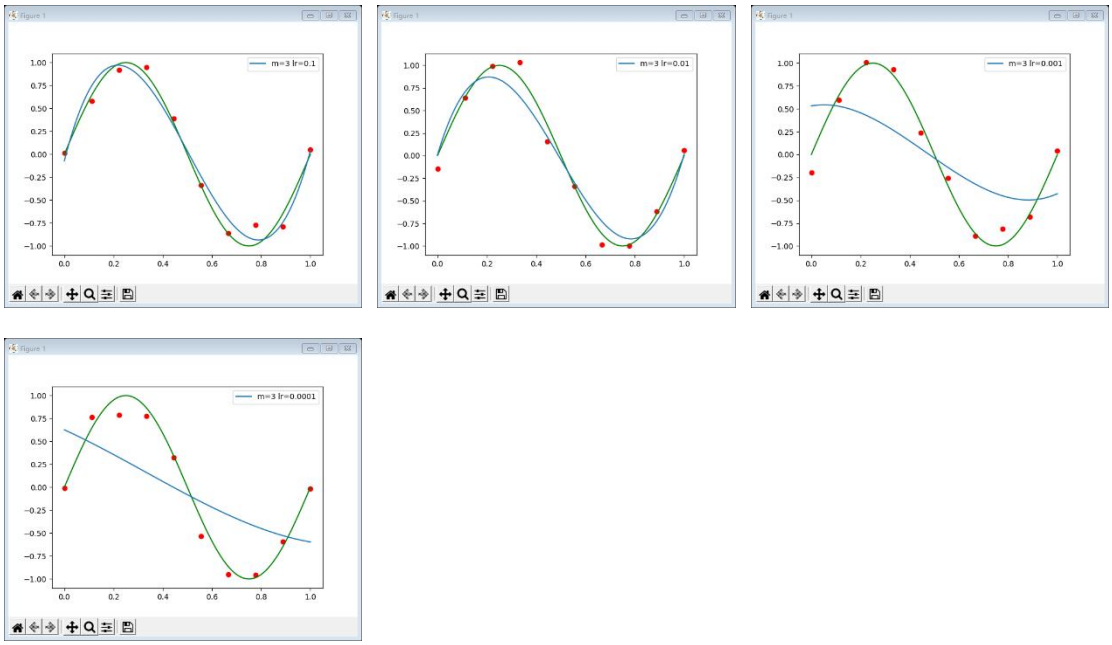
再对训练数据的量进行调整，阶数 m 选择 10，超参数 λ 选择 10^{-8}：



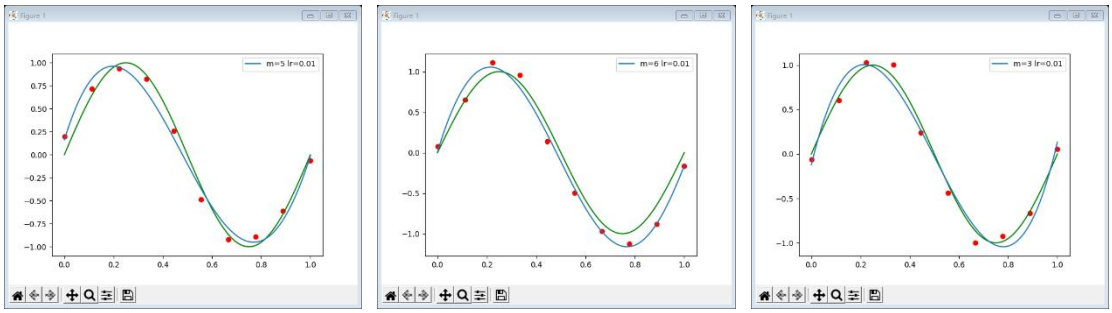
对于不同训练集，超参数 λ 的选择不同。因此每生成新训练集，则通过带惩罚项的解析解计算出当前的 best_lambda，用于带惩罚项的解析解、梯度下降和共轭梯度下降。

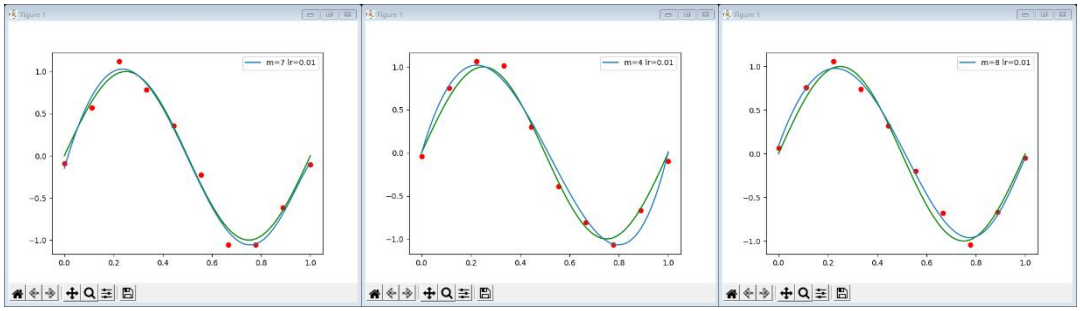
4.2 梯度下降法求解最优解

当阶数 m 为 3 时，更改学习率（learning_rate）lr，梯度缩减次数选择为 100，000 次（停止的精度要求为 10^{-5}），生成拟合曲线如下所示：

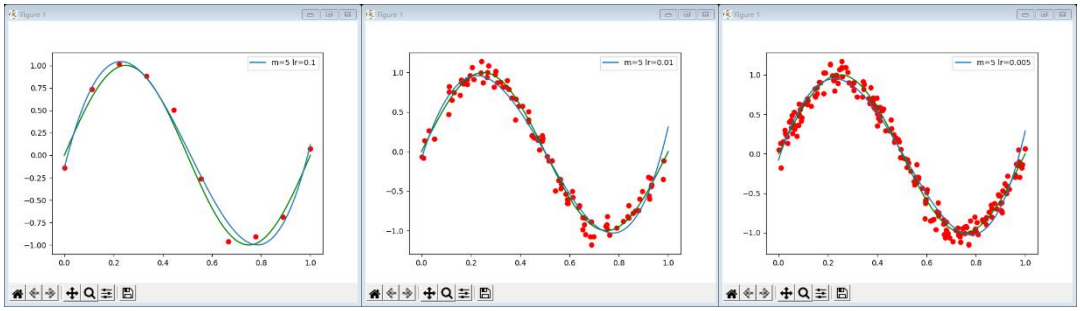


学习率 lr 为 0.01，次数为 1，000，000 次，不断更改阶数 m：



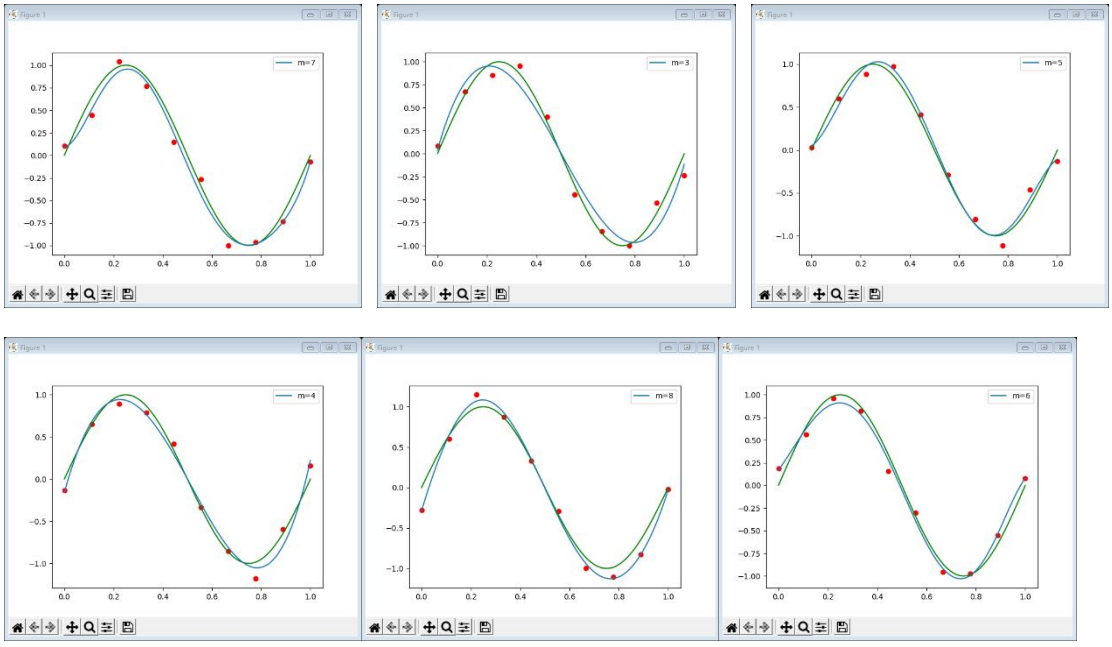


再增加数据量，阶数 m 为 5，学习次数重新改为 100，000 次，学习率 lr 因数据量变化需要也做相应改变：

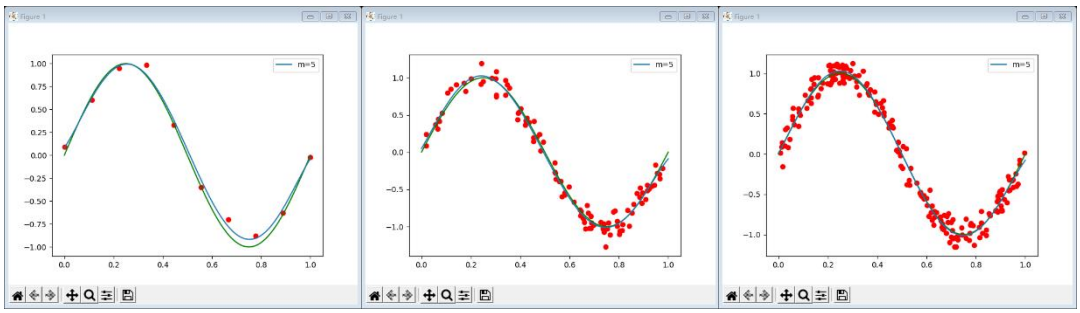


4.3 共轭梯度法求解最优解

不断改变阶数 m 生成拟合曲线如下所示：



阶数 m 确定为 5，再改变数据量：



五、结论

1. 增加训练样本的数据可以有效地解决过拟合的问题。使用最小二乘法求解最优解编译方便，相对于其他方法更方便理解，但在阶数过高时容易出现过拟合现象，原因是算法精度过高，虽然能够细致地学习更精细的数据，但是却事与愿违地将噪声的数据当成是精确的模型数据加入到结果模型中，从而为得到更高的准确性生成一个错误模型。能够细致地学习更精细的数据，但是却事与愿违地将噪声的数据当成是精确的模型数据，自动加入到了结果模型中，从而为了得到更高的准确性生成了一个错误的模型。

2. 对于训练样本限制较多的问题，通过增加惩罚项仍然可以有效解决过拟合问题。加入正则项并且使用正确的超参数 λ 后（可以通过设置其为变量来比较参数好坏，本实验没有去寻找最佳的超参数范围）对过拟合便有较大程度的抑制，在数据量较多时基本可以做到“完全”拟合出原始函数。

3. 梯度下降法需要人为定义学习率 lr，学习率越低则需要迭代的步骤数目便越多，而所得的参数也会更准确，本实验直接人为定义了迭代步骤数，提高了效率，在运行时对步骤数目进行更改以获得较好的迭代效果。其实还可以对梯度变化量进行限制：即小于某个极小数时再输出，但可能会因为学习率较低而进入死循环。最后所得模型的误差相较于其他方法会较大。

4. 共轭梯度法编译总体较为麻烦，但拟合曲线效果极佳，并且程序运行极快，在数据足够多时能够和加入正则/惩罚项的最小二乘法计算得到的模型具有相同极佳的拟合效果。

5. 对于梯度下降法和共轭梯度法而言，两种方法拟合结果相似，但共轭梯度下降收敛速度却远优于梯度下降、梯度下降稳定程度略优于共轭梯度下降。且二者相对于解析解而言，共轭梯度法的拟合效果解析解的效果更好。相比之下，共轭梯度下降法便是能够有效解决梯度下降法迭代次数多和复杂度高的有效方法。

六、参考文献

[1] CSDN 实际应用 2：sin(2πx)函数绘制（基于 matplotlib 的 python 数据可视化）https://blog.csdn.net/weixin_44940488/article/details/118066533

[2] Pattern Recognition and Machine Learning https://www.springer.com/us/book/9780387310732

[3] Shewchuk J R. An introduction to the conjugate gradient method without the agonizing pain[J]. 1994.

[4] CSDN python 可视化：fig, ax = plt.subplots()画多表图的 3 中常见样例&自定义图表格式
https://blog.csdn.net/htuhxf/article/details/82986440
[5] CSDN np.linspace 用法介绍
https://blog.csdn.net/Asher117/article/details/87855493

七、附录：源代码（带注释）

```
# -*- coding: gbk -*-

import numpy as np #调用 numpy 用于生成矩阵
import matplotlib.pyplot as plt #调用 matplotlib 库中的 pyplot 用于绘图

def sin_2pi(x):#打包利用 sin（2 pi x）函数
    return np.sin(2*np.pi*x)

def get_traindata():#获得训练数据
    x = np.linspace(0, 1, num = 10)#共计 10 个样本点，均匀分布在区间[0,1]之间，要求数据较少时使用这一项，下面为要求数据较多的情况时使用
    x = np.random.rand(200)
    y = sin_2pi(x) + np.random.normal(0, 0.1, 200)#增加高斯噪声，方差选取 0.1
    return x,y

def matrix(x, m = 3):#根据自变量 x 生成 X 矩阵，第 i 列为 x 的 i-1 次方，m 表示阶数，默认初始值为 3
    X = pow(x, 0)#第一列全为 1
    for i in range(1 ,m + 1):
        X = np.column_stack((X, pow(x, i)))#将每一列拼接起来就扩展变成矩阵
    return X

def get_params(X, Y):#最小二乘法不带惩罚项，通过公式得到参数 w，X 为拼接得到的矩阵，Y 为样本因变量 y 值的矩阵
    w = np.linalg.inv((X.T).dot(X)).dot(X.T).dot(Y) #公式 W* = (X’ •X)ˆ(-1) (X’) •Y=Xˆ(-1) •Y
    return w

def get_params_with_penalty(X, Y, l):#最小二乘法带惩罚项，通过公式得到参数 w，X 为拼接得到的矩阵，Y 为样本因变量 y 值的矩阵，l 为人为设定的超参数
    w = np.linalg.inv((X.T).dot(X) + l * (np.identity(len(X.T)))).dot(X.T).dot(Y) #公式:w* = (X’ • X+lI)ˆ(-1) (X’) • Y
    return w

def gradient_descent_fit(X, Y, lr = 0.01):#梯度下降法求解最优解，通过公式得到参数 w，X 为
```

```
拼接得到的矩阵，Y 为样本因变量 y 值的矩阵，lr 为学习率
    w = ([ 0, 0, 0, 0, 0, 0])#初始化 w
    for i in range(100000):#采用足够多的步数进行计算，也可以用梯度小于某个极小值进行计算
        Y1 = np.dot(X, w)
        grad = np.dot(X.T, Y1-Y)
        w -= grad * lr
    return w

def conjugate_gradient_fit(X, Y, m):#共轭梯度法求解最优解，通过公式得到参数 w，X 为拼接得到的矩阵，Y 为样本因变量 y 值的矩阵，m 为阶数
    A = np.dot(X.T, X)#A=X^T·X，当成 A 已经正则
    b = np.dot(X.T, Y)#b=X^T·Y
    w = np.zeros(m + 1)#初始化 w
    epslon = 0.00001#设一个极小数作为跳出条件
    d = r = b - np.dot(A, w)#按照上述公式进行初始化
    r0=r
    while True:#迭代部分
        al = np.dot(r.T, r) / np.dot(np.dot(d, A), d)
        w += al * d
        r1 = r - al*np.dot(A, d)
        be = np.dot(r1.T, r1) / np.dot(r.T, r)
        d = be * d + r1
        r = r1
        if np.linalg.norm(r) / np.linalg.norm(r0) < epslon:
            break
    return w

x = np.linspace(0, 1, num = 100)
y=sin_2pi(x)
m = 5
plt.plot(x, y, color = 'green')#绘制原函数的图像，曲线用绿色绘制
x1, y1 = get_traindata()
plt.scatter(x1, y1, color = 'red')#绘制样本数据的散点图，散点用红色绘制
X = matrix(x1, m)
Y = y1
w = get_params(X, Y)#用样本值得到参数 w
X1 = matrix(x, m)#绘制曲线需要更多的数据，使用原先定义 100 个 x 的值得到拟合曲线
Y1 = X1.dot(w)
plt.plot(x, Y1, label = "m = " + str(m))#绘制拟合曲线图像
plt.legend()
plt.show()
```