2020 年春季学期
计算学部《机器学习》课程

Lab 1 实验报告

姓名	赵仁杰
学号	1180300113
班号	1803101
电子邮件	1579974122@qq.com
手机号码	15122925619

目录

1 问题描述

1.1 实验目的

1. 利用解析解，梯度下降法以及共轭梯度法进行函数拟合。
2. 要掌握最小二乘法（有无正则项）
3. 研究各种参数对实验结果的影响

1.2 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。

1.3 实验环境

Windows 10

Python 3.6

Pycharm

2 生成数据集

2.1 生成测试集

2.1.1 生成 X 以及标准值 Y

首先，利用 random.rand 通过本函数可以返回一组服从“0~1”均匀分布的随机样本值。该随机样本值是由 data_size 个数据作为自变量，但是我们要注意将其变为列向量，之后利用 numpy 的 sin(2πx)函数计算出标准值之后再利用高斯分布计算出噪声相加生成 Y。

2.1.2 计算高次值特征

建立一个新的矩阵，其目的是要一个范德蒙行列式作为高次值特征，计算 x 对应幂值（从 0->m），生成一个[N,m+1]的矩阵作为训练集中的特征向量即可。

```
def wandermonde(m, x):
    '''

    :param m: 拟合的多项式系数
    :param x: 训练集的 x 数据
    :return:
    '''
    N = x.size
    #生成一个 m+1 长度的数组
    order = np.arange(m + 1)
    #将他往下叠加
    e = np.tile(order, (N, 1))
    e = np.transpose(e)
    #相乘，转置之后即是想要的范德蒙行列式子
    XT = np.power(x, e)
    X = np.transpose(XT)
    return X
```

2.2 生成测试集合

在[0,1]之间均匀的取 400 个点作为自变量，计算标准值之后添加与训练集相同分布的

噪声作为测试集即可。

3 解析解

利用最小二乘法计算损失函数，并求出损失函数。之后我们令损失函数对参数矩阵为 0，求出极值点的导数值，推导如下：

3.1 利用高阶多项式函数拟合曲线(不带惩罚项)

利用训练集合，对于每个新的 x，预测目标值 t。采用多项式函数进行学习，即利用式来确定参数 w，假设阶数 m 已知。

$$y(x,w) = w_0 + w_1 x + \cdots + w_m x^m = \sum_{i=0}^{m} w_i x^i \tag{1}$$

采用最小二乘法，即建立误差函数来测量每个样本点目标值 与预测函数 y(x,w)之间的误差，误差函数即式（2）

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

之后我们可以把上面的式子进行转换，如下

$$E(\mathbf{w}) = \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{T})'(\mathbf{X}\mathbf{w} - \mathbf{T})$$

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

之后进行求导即可得出，

$$\frac{\partial E}{\partial \mathbf{w}} = \mathbf{X}'\mathbf{X}\mathbf{w} - \mathbf{X}'\mathbf{T}$$

利用求导为零：

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

3.2 带惩罚项的多项式函数拟合曲线

由于在不带惩罚项的多项式拟合曲线时，在参数多时 具有较大的绝对值，本质就是发生了过拟合。
最小二乘法计算损失函数（带有正则项，实际计算无正则项方法时令$\lambda = 0$）：

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

将其写为矩阵格式可得：

$$\begin{aligned} E(w) =& \frac{1}{2}(t - Xw)^T(t - Xw) + \frac{\lambda}{2}w^T w \\ =& \frac{1}{2}(t^T - w^T X^T)(t - Xw) + \frac{\lambda}{2}w^T w \\ =& \frac{1}{2}(t^T t - w^T X^T t - t^T Xw + w^T X^T Xw) + \frac{\lambda}{2}w^T w \end{aligned}$$

将其对 w 求导可得

$$\begin{aligned} \frac{\partial E(w)}{\partial w} =& \frac{1}{2}(-X^T t - X^T t + 2X^T Xw + 2\lambda Ew) \\ =& (X^T X + \lambda E)w - X^T t \end{aligned}$$

令这个偏导数等于零向量：

$$(X^T + \lambda I)w - X^T t = \vec{0}$$

解得：

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

根据训练集的 T 和 X 计算出解析解，进行拟合。

4 梯度下降法

4.1 梯度下降法的依据

如果实值函数在某点处可微，并且有定义，那么我们知道顺着梯度 $\nabla f(\mathbf{x_i})$ 为增长最快的方向，继而- $\nabla f(\mathbf{x_i})$方向上下降最多。因此有下式成立：

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

因此，如果顺利我们可以得到一个 收敛到期望的最小值

4.2 无正则项的方法

算法如下面所示：

先利用最小二乘法计算损失函数：

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

然后在$E(x)$中对w求导：

$$\begin{aligned}\frac{\partial E(w)}{\partial w} =& \frac{1}{2}(-X^Tt - X^Tt + 2X^TXw)\\ =& X^TXw - X^Tt\end{aligned}$$

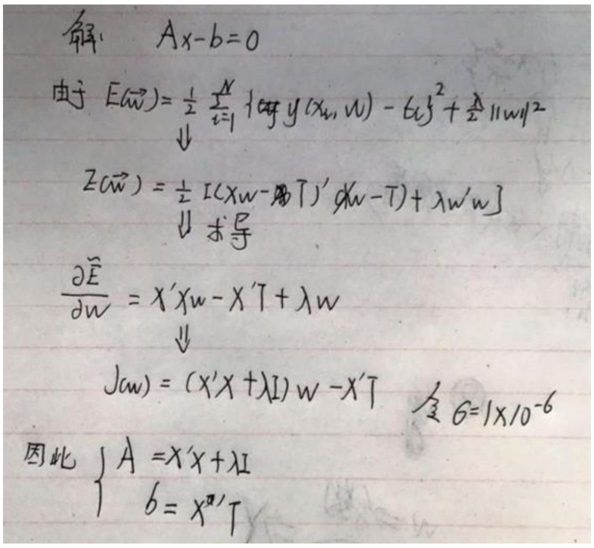
之后计算梯度下降：

$$w_{new} = w - learningRate * \frac{\partial E(w)}{\partial w}$$

不断重复上述计算过程，直至损失函数值$E(w)$不再下降或者轮数达到要求时停止。

4.3 添加正则项

因为数学公式比较多，所以直接手写啦。
推导过程如下：



解： $Ax - b = 0$

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

$\Downarrow$

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

$\Downarrow$ 求导

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

$\Downarrow$

$J(w) = (X'X + \lambda I)w - X'T$ 　令 $\delta = 1\times10^{-6}$

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

如此所示，之后计算梯度下降：

$$w_{new} = w - learningRate * \frac{\partial E(w)}{\partial w}$$

不断重复上述计算过程，直至损失函数值$E(w)$不再下降或者轮数达到要求时停止。

5 共轭梯度法

5.1 共轭梯度法简介

简介：共轭梯度法解决的主要是形如 的线性方程组解的问题，其中 必须是对称的、正定的。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，每次都选泽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。

5.2 共轭梯度的实现

对于线性方程组$Ax = b$，可以按照如下方法求解：

$$\begin{aligned}
&\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0 \\
&\mathbf{p}_0 := \mathbf{r}_0 \\
&k := 0 \\
&\text{repeat} \\
&\qquad \alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k} \\
&\qquad \mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k \\
&\qquad \mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k \\
&\qquad \text{if } r_{k+1} \text{ is sufficiently small, then exit loop} \\
&\qquad \beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k} \\
&\qquad \mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k \\
&\qquad k := k+1 \\
&\text{end repeat}
\end{aligned}$$

在函数拟合问题中，系数矩阵即为我们需要求解的x，我们可以按照如下公式构造A和B：

$$A = X^T X + hyper * E_{len(X^T)}$$
$$B = X^T Y$$

之后，按照 5.1 中算法进行迭代计算，直至损失函数足够小或迭代次数足够多后得到模

型。

6 实验结果

6.1 解析解结果分析

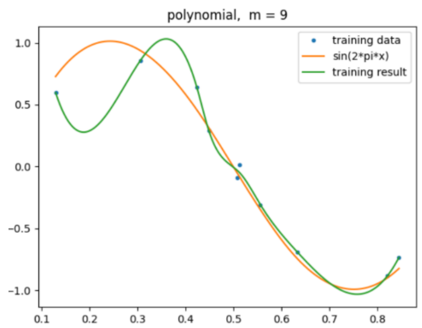
6.1.1 五正则项的拟合结果

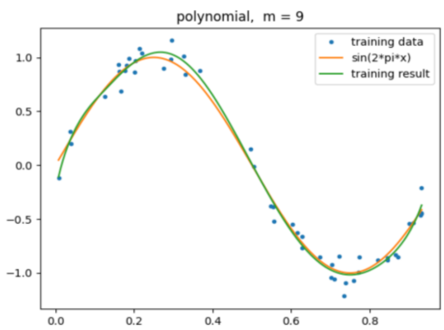
（1） 改变数据量对实验结果的影响

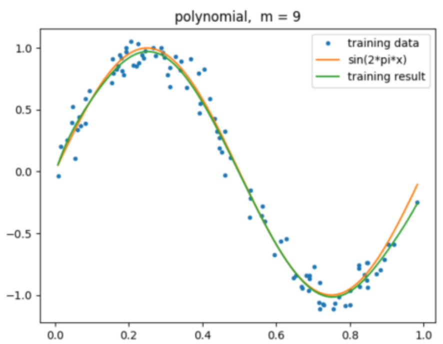
多项式的阶数为 9，不加入正则项，分别取不同的训练集规模

其中-绿色为拟合函数的曲线，蓝色点为训练集样本点，橙色为 sin（2Pix）

第一个图是训练集数据量为 10，下一个是训练集数据量为 50，最后一个为训练集数据量 100







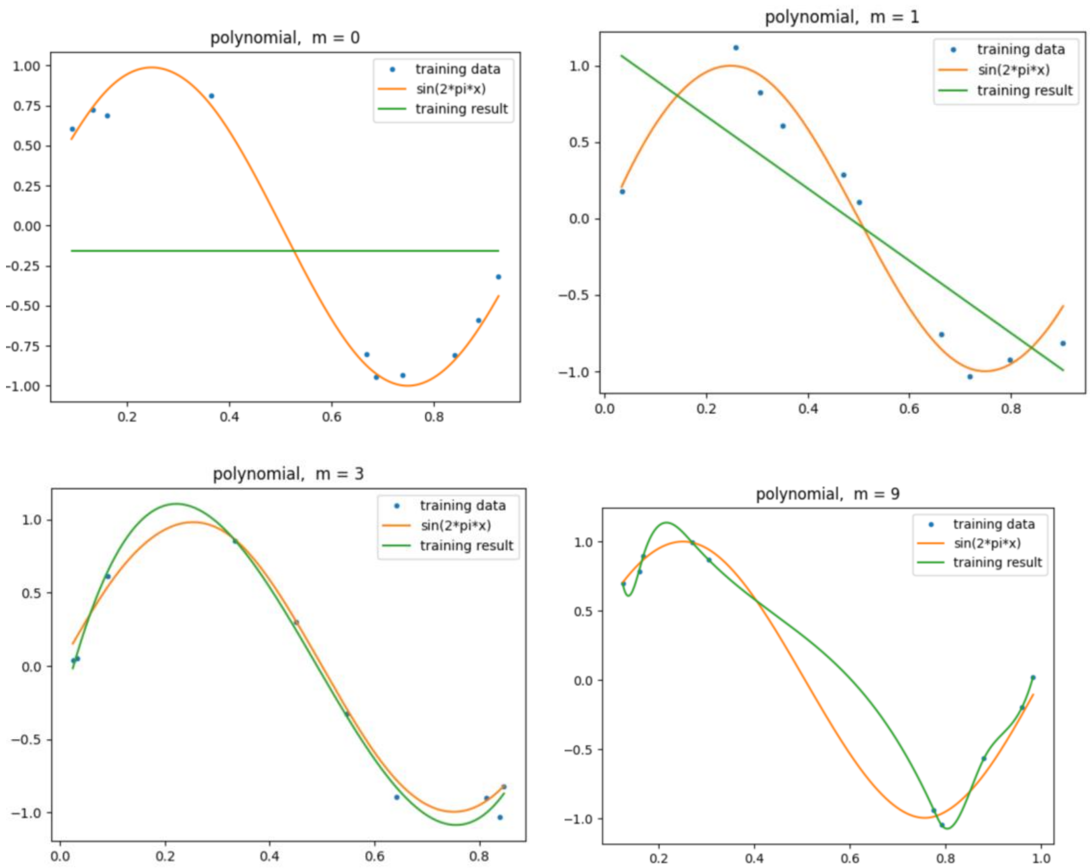
对于阶数固定为 0 的多项式，我们对上面三个图像进行观察可以发现，如果训练集数据量越大的画，拟合效果越好，过拟合现象越来越不明显。

第一个是数据量为 10 的 u 想你练级，图像完全的穿过了所有的点，过拟合现象十分严重。这样的原因是，在阶数过大的时候，模型的复杂度和拟合宁理都增强，因此可以通过一些特定的系数使训练集的所有点都在图像上面，损失函数最小。尤其是针对第一个图，用九次函数取拟合是个点的时候，完全可以求得一组合适的系数使得损失函数为 0.同时进行对比，当训练集数据量越来越多的时候，拟合效果越好。

（2） 改变多项式的阶数 m 对实验结果的影响：“

这个时候，保证训练集数据的数量不变，一直为 10，无正则项，分别取不同的阶数

观察结果（其中-绿色为拟合函数的曲线，蓝色点为训练集样本点，橙色为 sin（2Pix）
）

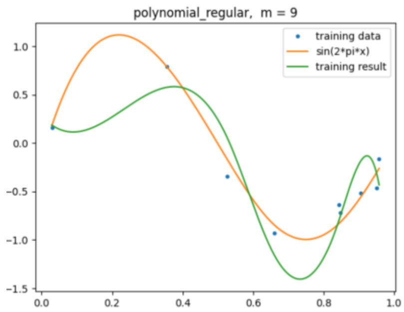


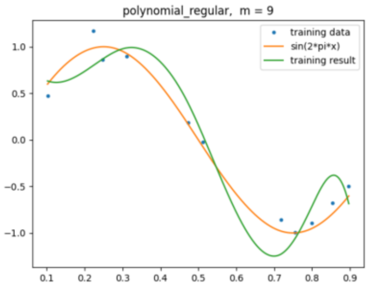
多项式的阶数直接决定了拟合函数的灵活性以及复杂性，首先，如果拟合函数的阶数过小，拟合能力过弱，则会出现比较严重的欠拟合现象---对应图一和图二。

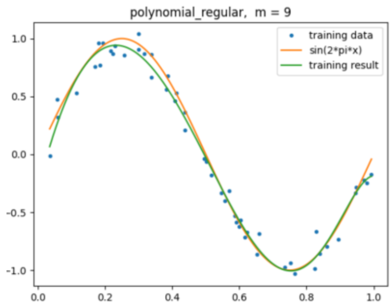
如果拟合函数的阶数过大，拟合能力就会过强，会导致对于训练集中的数据过分拟合，出现过拟合现象----对应图四。

6.1.2 有正则项的拟合结果：

（1）　改变数据量对实验结果的影响
多项式阶数为 9，我们将正则化参数设置为 0.00001，分别取不同的训练集规模（其中-绿色为拟合函数的曲线，蓝色点为训练集样本点，橙色为 sin（2Pix）
）
第一数据量为 10，第二个为 11
第三个为 50





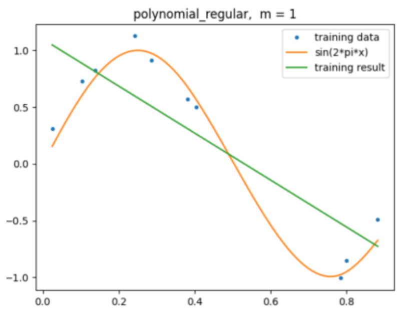


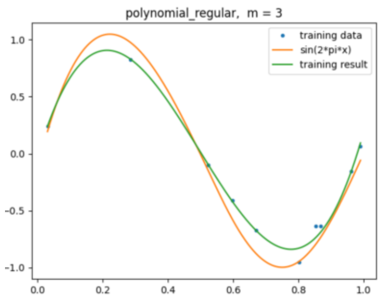
有正则项的结果基本与无正则项的结果类似，但过拟合现象明显减少，因为正则项作为一种惩罚，对过于“复杂”的模型会给予更高的惩罚，防止出现过于复杂的模型形成过拟合的现象；同时，如果正则化系数过大，则会产生严重的欠拟合现象（见下图）。

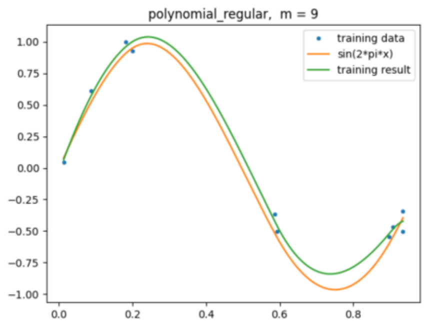
（2） 改变阶数 m 对实验结果的影响

依旧将数据量设置为 10，系数为 0.00001，分别取不同的训练及模型和 m。

多项式阶数为 9，我们将正则化参数设置为 0.00001，分别取不同的训练集规模（其中-绿色为拟合函数的曲线，蓝色点为训练集样本点，橙色为 sin（2Pix））







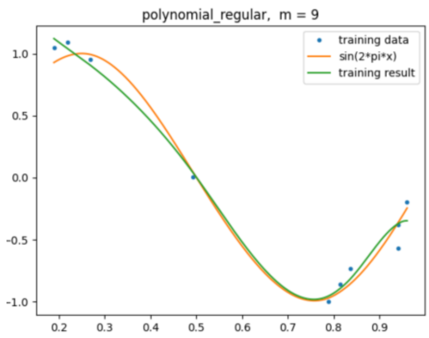
有正则项的结果基本与无正则项的结果类似，但过拟合现象明显减少，同样是因为

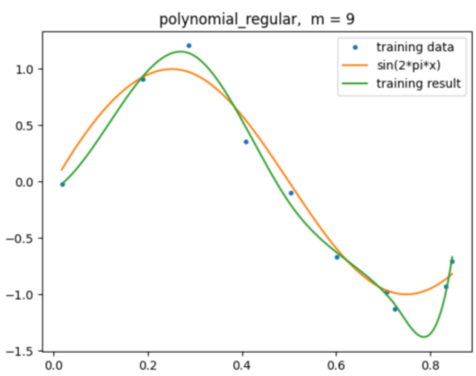
正则项的添加。

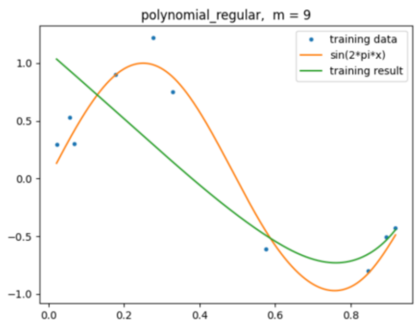
（3） 正则项系数 lambda 对实验结果的影响

设置训练集数据量为 10，多项式阶数为 9，分别取不同的正则项系数（其中-绿色为拟合函数的曲线，蓝色点为训练集样本点，橙色为 sin（2Pix））

```
第一个系数为 0.00001
第二个系数为 0.00000001
第三个系数为 0.1
```







在这里，我们可以看到，添加了正则化系数会对过拟合现象产生不同程度的抑制，正则化系数的大小直接影响抑制程度的大小。当正则化系数过小时（lambda = 1e-11）时，过拟合现象仍然存在；当正则化系数过大时，拟合效果会大大下降，产生强烈的欠拟合现象。这是因为，同一个复杂度的模型，正则化系数越大，对这个模型的惩罚就越大，这使得原本合适的模型反而会产生较大的损失函数值，根据优化方向将继续降低函数的复杂度至低于需要的复杂度。

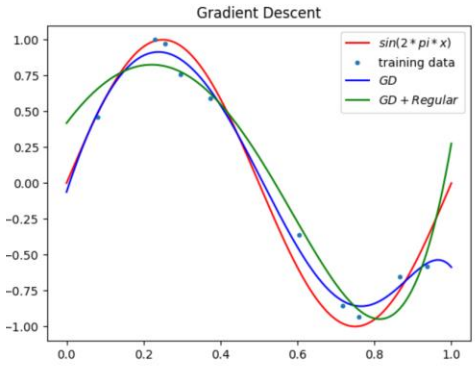
6.2 梯度下降法结果：

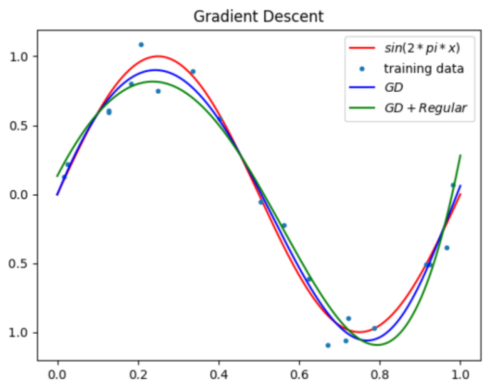
这里为了对比的方便，我直接将有正则项和没有正则项一起对比，得出结论。

（1）改变数据量对实验结果的影响：

多项式阶数为 9，正则化参数为 0.00001，分别取不同的训练集规模（红色线为正弦曲线，绿色线为拟合函数+正则的曲线，浅蓝色点为训练集样本点，深蓝色为拟合函数的曲线）

第一个数据量为 10，第二个数据量为 20



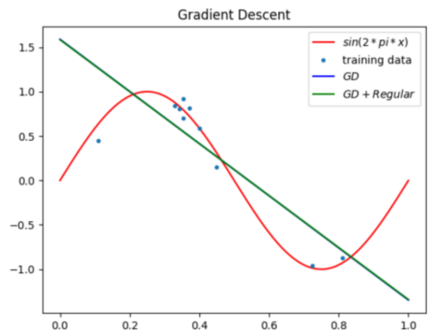


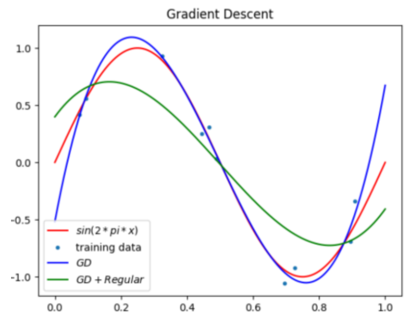
后续的因为数据量太大，在这里并没有展示。我们有上面可以分析得知，不同的数据量会影响拟合的效果。数据量越大，拟合效果越好。因为随机噪声的存在，数据集越大，噪声的影响就越小，添加了正则项，可以看到，在同等条件下，添加了正则项之后明显提高了拟合效果。

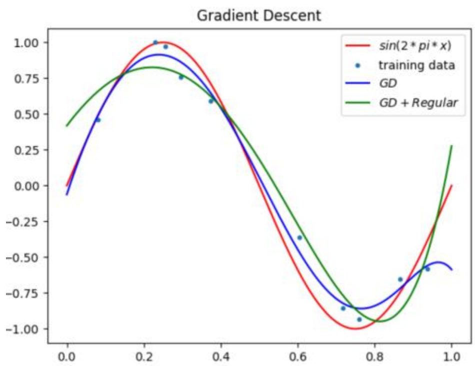
(2)改变多项式阶数 M 对实验结果的影响

训练集数据量为 10，正则化参数为 0.00001，分别取不同的阶数（红色线为正弦曲线，绿色线为拟合函数+正则的曲线，浅蓝色点为训练集样本点，深蓝色为拟合函数的曲线）

第一个阶数为 1，第二个阶数为 3，第三个阶数为 9





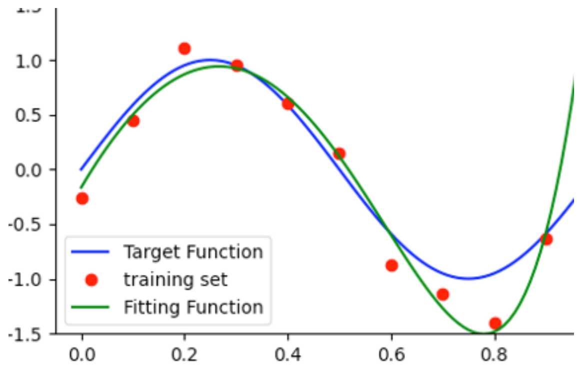


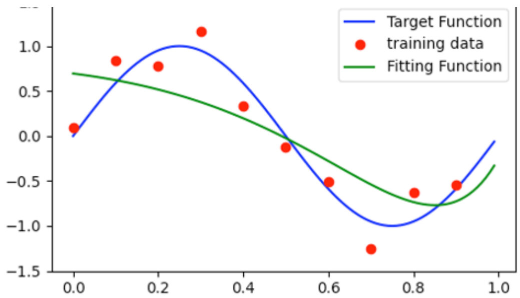
和解析解类似，阶数越高，函数的拟合效果越强；阶数越低，拟合效果越差。

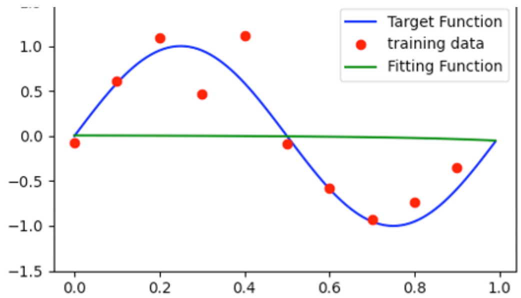
（3）正则系数对它的影响

训练集数据量为 10，多项式阶数为 9，分别取不同的训练集规模，（红色线为正弦曲线，绿色线为拟合函数+正则的曲线，浅蓝色点为训练集样本点，深蓝色为拟合函数的曲线）由于这里讨论系数，因此将不带正则项的移除，可以更加直观。

第一个为 0.00000001，第二个是 0.0001，第三个是 0.1







这里有关正则系数的改变同上面的解析解类似。

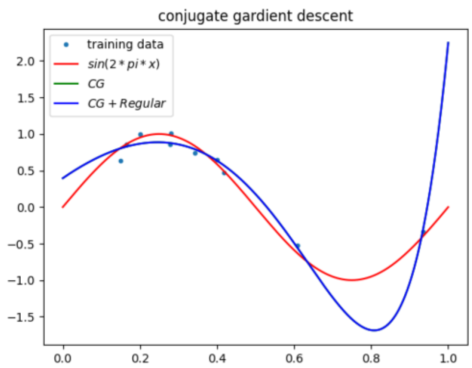
6.3 共轭梯度法

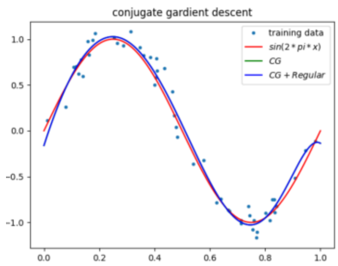
这里为了对比的方便，我直接将有正则项和没有正则项一起对比，得出结论。

（1）改变数据量对实验结果的影响：

多项式阶数为 9，正则化参数为 1e-11，分别取不同的训练集规模（红色线为正弦曲线，绿色线为拟合函数+正则的曲线，浅蓝色点为训练集样本点，深蓝色为拟合函数的曲线）

第一个数据量为 10，第二个数据量为 50





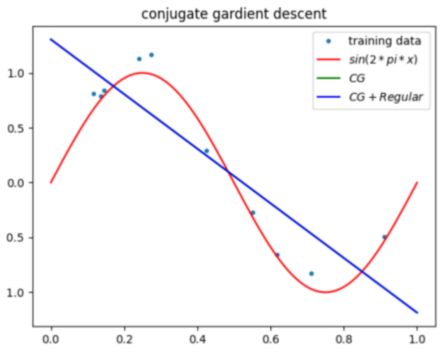
共轭梯度法的拟合效果略微优于梯度下降法，而且训练速度快，可以在 10 步以内计算出最优的系数结果。

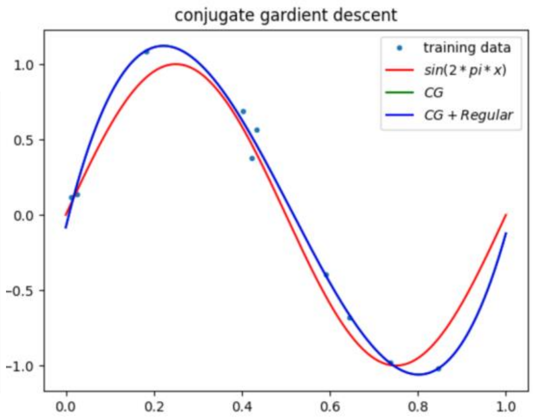
（2）阶数 m 对这个问题的影响

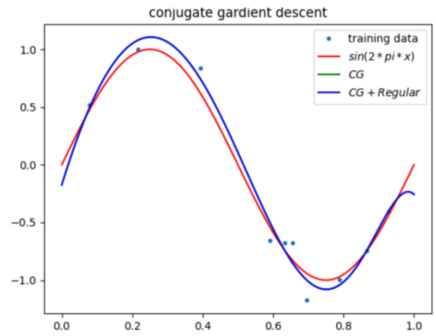
训练集数据量为 10，正则化参数为 1e-11，分别取不同的阶数

（红色线为正弦曲线，绿色线为拟合函数+正则的曲线，浅蓝色点为训练集样本点，深蓝色为拟合函数的曲线）

第一个阶数是 1，第二个是 3，第三个是 9





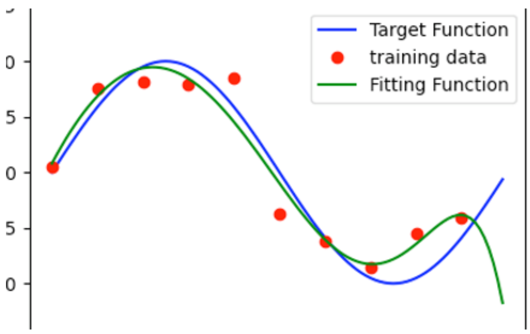


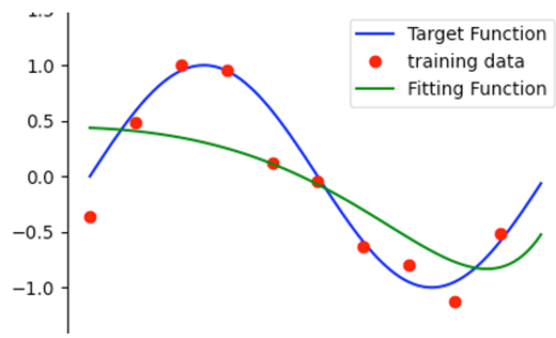
在共轭梯度法下，改变多项式的阶数的实验结果与另两种方法几乎相同，在 9 阶十个点的情况下仍然出现了过拟合，但是训练速度快，而且精度高。

（3）正则系数对它的影响

训练集数据量为 10，多项式阶数为 9，分别取不同的训练集规模，（红色线为正弦曲线，绿色线为拟合函数+正则的曲线，浅蓝色点为训练集样本点，深蓝色为拟合函数的曲线）由于这里讨论系数，将不带正则项的移除，可以更加直观。

第一个是 0.0001，第二个是 0.1





正则化系数在这个方法中对拟合效果的影响与其他方法中基本相同，过低的正则化系数对拟合效果的影响不明显；过高的正则化参数又会大大削弱函数的拟合效果，产生严重的欠拟合现象。

7 实验结论

1：共轭梯度法收敛的效果极快，且精度高；

梯度下降法的效果与速度均不如共轭梯度法。

2、正则项可以有效的降低函数的拟合效果，并以此来解决过拟合的问题；但是，过小的正则化系数对函数拟合的效果影响不大；过大的正则化系数会过分削弱函数的拟合效果。
3： 对于同一个拟合函数，其他参数都相同的情况下，训练集的规模越大，拟合函数的拟合效果越好，训练集的最小大小为拟合函数的阶数加一，否则会出现过拟合现象（无正则化条件下）
4：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。
5：直观的来书，增加数据的适量可以明显的降低过拟合的效果。

实验代码：

```
# -*- coding: utf-8 -*-
# creat_random.py
# author 1180300113
# time 2020/10/
import numpy as np
import random
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt


# 生成光滑曲线，在图表展示出来
def draw(x, y, label):
    '''

    :param x: 代表数据的横坐标
    :param y: 加入噪声之后的估计值
    :param label: 标签
    :return:
    '''
    #生成连续的多个数据
    x_trans = np.linspace(x.min(), x.max(), 500)
    #拟合出来相应的函数
    func = interp1d(x, y, kind='cubic')
    #通过函数计算
    y_trans = func(x_trans)
    plt.plot(x_trans, y_trans, label=label)

# 展示曲线，进行对比
def show(m, x, y, z, p, method):
    '''

    :param m: 表示多项式阶数
    :param x: 训练集数据的横坐标
    :param y: 加入噪声的训练集值
    :param z: 未加入噪声的训练集
```

```
    :param p: 经过多项式拟合系数计算出来的值
    :param method: 提示相对应的方法
    :return:
    '''
    plt.title(method + ",  " + "m = %d"%m)
    plt.plot(x, y, linestyle="", marker='.', label="training data")
    draw(x, z, "sin(2*pi*x)")
    draw(x, p, "training result")
    plt.legend(loc="upper right")
    plt.show()

# 生成范德蒙行列式
def wandermonde(m, x):
    '''


    :param m: 拟合的多项式系数
    :param x: 训练集的 x 数据
    :return:
    '''
    N = x.size
    #生成一个 m+1 长度的数组
    order = np.arange(m + 1)
    #将他往下叠加
    e = np.tile(order, (N, 1))
    e = np.transpose(e)
    #相乘，转置之后即是想要的范德蒙行列式子
    XT = np.power(x, e)
    X = np.transpose(XT)
    return X

# 计算估计值
def calu(X, w):
    return np.matmul(X, w).T

# 不带惩罚项下解析解
def polynomial(m, x, y):
    '''


    :param m: 拟合多项式的阶数
    :param x: 训练集的 x 分布
    :param y: 估计值
    :return:
    '''
    yt = y.T
```

```
    #计算出范德蒙行列式
    X = wandermonde(m, x)
    res1 = np.matmul(X.T, X)
    res2 = np.matmul(X.T, yt)
    w = np.linalg.solve(res1, res2)
    #得到拟合之后的值
    return calu(X, w)

# 带惩罚项的解析解
def polynomial_regular(m, x, y, lamda):
    '''

    :param m: 拟合多项式的阶数
    :param x: 数据 x 分布
    :param y: 加入噪声
    :param lamda: 系数
    :return:
    '''
    yt = y.T
    X = wandermonde(m, x)
    a = np.matmul(X.T, X) - lamda*np.identity(m+1)
    b = np.matmul(X.T, yt)
    w = np.linalg.solve(a, b)
    return calu(X, w)

# 生成含有噪声的数据
def creat_data(data):
    x = np.random.rand(data)
    x.sort(axis = 0)
    y = np.sin(2*np.pi*x)
    z = np.sin(2*np.pi*x)
    for i in range(x.size):
      m = 0.0
      s = 0.1
      y[i] += random.gauss(m, s)
    return x,y,z
#生成数据
x,y,z = creat_data(10)

# 数据验证
#p = polynomial(0, x, y)
#show(0, x, y, z, p, "polynomial")
#
#p = polynomial(1, x, y)
```

```
#show(1, x, y, z, p, "polynomial")
#
#p = polynomial(3, x, y)
#show(3, x, y, z, p, "polynomial")
#
#p = polynomial(5, x, y)
#show(5, x, y, z, p, "polynomial")
#
# p = polynomial(7, x, y)
# show(7, x, y, z, p, "polynomial")
#
#p = polynomial(9, x, y)
#show(9, x, y, z, p, "polynomial")
#
#p = polynomial_regular(9, x, y, 0.00001)
#show(9, x, y, z, p, "polynomial_regular")
#
#p = polynomial_regular(9, x, y, 0.00000001)
#show(9, x, y, z, p, "polynomial_regular")
#
p = polynomial_regular(9, x, y, 0.1)
show(9, x, y, z, p, "polynomial_regular")
#
#p = polynomial_regular(9, x, y, 1.0)
#show(9, x, y, z, p, "polynomial_regular")
```

梯度下降

```
import numpy as np
import matplotlib.pyplot as plot
import random
import numpy
import math

# 生成含有噪声的数据
def creat_data(data):
    x = np.random.rand(data)
    x.sort(axis = 0)
    y = np.sin(2*np.pi*x)
    z = np.sin(2*np.pi*x)
    for i in range(x.size):
      m = 0.0
      s = 0.1
      y[i] += random.gauss(m, s)
    return x,y,z
```

```
#添加正则项的梯度下降算法
def GradientDescentRegular(x, y, w_init):
    # 设置正则系数
    lamda = 0.001
    # 将数据转化为矩阵
    w = np.mat(w_init.copy()).T
    Y = np.mat(y).T
    # 设置布长为 0.05
    step = 0.05
    # 范德蒙德矩阵的计算
    X = np.mat(np.zeros((x.size, w_init.size)))
    for i in range(x.size):
        for j in range(w_init.size):
            X[i, j] = pow(x[i], j)
    # 初始设置损失函数值
    loss_function = 1.0
    # 设置迭代次数为 0
    iteratornNum = 0
    # 迭代终止条件有两个
    #一是 loss_function > 0.001 二是 iteratornNum < 200000
    while (loss_function > 0.001 and iteratornNum < 200000):
        iteratornNum = iteratornNum + 1
        # 更新 w，并设置惩罚系数为 0.001
        w -= step * (np.dot(X.T, (np.dot(X, w) - Y)) + lamda * w)
        loss_function = 0.0
        for i in range(x.size):
            loss_function += pow(polynomial(x[i], np.array(w.T)[0]) -
y[i], 2)
        # 计算损失函数值
        loss_function /= (2 * x.size)
    W = np.zeros(w_init.size)
    for i in range(w_init.size):
        W[i] = w[i, 0]
    return W


def GradientDescent(x, y, w_init):
    # 正宗梯度下降算法
    w = np.mat(w_init.copy()).T
    Y = np.mat(y).T
    step = 0.05
    # 范德蒙德矩阵的计算
    X = np.mat(np.zeros((x.size, w_init.size)))
```

```
    for i in range(x.size):
        for j in range(w_init.size):
            X[i, j] = pow(x[i], j)
    # 初始设置损失函数值
    loss_function = 1.0
    # 迭代次数设置为 0
    iteratornNum = 0
    # 迭代终止条件为 loss_function > 0.001 and iteratornNum < 200000
    while (loss_function > 0.001 and iteratornNum < 200000):
        iteratornNum = iteratornNum + 1
        w -= step * np.dot(X.T, (np.dot(X, w) - Y))  # 更新 w，并设置惩罚
系数为 0.001
        loss_function = 0.0
        for i in range(x.size):
            loss_function += pow(polynomial(x[i], np.array(w.T)[0]) -
y[i], 2)
        loss_function /= (2 * x.size)  # 计算损失函数值
    W = np.zeros(w_init.size)
    for i in range(w_init.size):
        W[i] = w[i, 0]
    print("loss_function:", loss_function)
    print("The iteratornNum times is:", iteratornNum)
    return W

def polynomial(x, w):
    answer = 0.0
    for i in range(w.size):
        answer += w[i] * pow(x, i)
    return answer

size = 10
power = 4
#生成数据
x,y,z=creat_data(size)

#用所生成标准化数据
x_points=np.linspace(0,1,num=1000)
#构造随机的正分布的数据
w_init=np.random.randn(power)

#用红色打印，标准化 sin(2*pi*x) 数据
plot.title("Gradient Descent")
plot.plot(x_points,np.sin(2*np.pi*x_points),label='$sin(2*pi*x)$',col
or="red")
```

```
#打印训练集数据
plot.plot(x, y, linestyle="", marker='.', label="training data")


w_gradient_descent=GradientDescent(x,y,w_init)
plot.plot(x_points,polynomial(x_points,w_gradient_descent),label=
'$GD$',color="blue")

w_gradient_descent_regu=GradientDescentRegular(x,y,w_init)
plot.plot(x_points,polynomial(x_points,w_gradient_descent_regu),label
= '$GD+Regular$',color="green")

plot.legend()
plot.show()
```

共轭梯度

```
import numpy as np
import matplotlib.pyplot as plot
import scipy as sp
import random

def d1Size(X):
    sum = 0.0
    for i in range(X.size):
        sum += pow(X[i,0],2)
    return sum


 #不添加正则项的共轭梯度法
def ConjungateGradient(x, y, w_init):
    # 实现数据的矩阵化
    w = np.mat(w_init.copy()).T
    Y = np.mat(y.copy()).T
    X = np.mat(np.zeros((x.size, w_init.size)))
    for i in range(x.size):
        for j in range(w_init.size):
            X[i, j] = pow(x[i], j)
     # 正定矩阵
    A = np.dot(X.T,X)
    g = (np.dot(np.dot(X.T,X),w)-np.dot(X.T,Y))
    if(d1Size(g)!=0):
        # 计算梯度大小
        d = -g
```

```
        while(d1Size(g)>0.00001):
            step = (np.dot(-g.T,d)/np.dot(np.dot(d.T,A),d))[0,0]
            w += step*d
            g = (np.dot(np.dot(X.T, X), w) - np.dot(X.T, Y))
            # 计算步长
            speed =
(np.dot(np.dot(d.T,A),g)/np.dot(np.dot(d.T,A),d))[0,0]
            d = -g+speed*d
    new_w = np.zeros(w_init.size)
    for i in range(w_init.size):
        new_w[i] = w[i,0]
    return new_w

    # 添加正则项的共轭梯度法
def ConjungateGradientRegular(x,y,w_init):
    lamda=1e-10
    # 实现数据的矩阵化
    w = np.mat(w_init.copy()).T
    Y = np.mat(y.copy()).T
    X = np.mat(np.zeros((x.size, w_init.size)))
    for i in range(x.size):
        for j in range(w_init.size):
            X[i, j] = pow(x[i], j)
    E = np.identity(w_init.size)
    # 添加惩罚项的正定矩阵
    A = np.dot(X.T,X) + E*lamda
    g = (np.dot(np.dot(X.T, X), w) - np.dot(X.T, Y))
    if(d1Size(g)!=0):
        d = -g
        # 计算梯度大小
        while(d1Size(g)>0.00001):
            step = (np.dot(-g.T,d)/np.dot(np.dot(d.T,A),d))[0,0]
            w += step*d
            g = (np.dot(np.dot(X.T, X), w) - np.dot(X.T, Y))
            speed =
(np.dot(np.dot(d.T,A),g)/np.dot(np.dot(d.T,A),d))[0,0]#计算步长
            d = -g+speed*d
    new_w = np.zeros(w_init.size)
    for i in range(w_init.size):
        new_w[i] = w[i,0]
    return new_w

# 生成含有噪声的数据
def creat_data(data):
```

```
    x = np.random.rand(data)
    x.sort(axis = 0)
    y = np.sin(2*np.pi*x)
    z = np.sin(2*np.pi*x)
    for i in range(x.size):
      m = 0.0
      s = 0.1
      y[i] += random.gauss(m, s)
    return x,y,z


def polynomial(x, w):
    answer = 0.0
    for i in range(w.size):
        answer += w[i] * pow(x, i)
    return answer

size = 10
power = 10
#利用函数生成数据
x,y,z=creat_data(size)

#用所生成标准化数据
x_points=np.linspace(0,1,num=1000)

#构造随机的正分布的数据
w_init=np.random.randn(power)


plot.title("conjugate gardient descent")
#打印训练集数据
plot.plot(x, y, linestyle="", marker='.', label="training data")

plot.plot(x_points,np.sin(2*np.pi*x_points),label='$sin(2*pi*x)$',col
or="red")

w_FR1=ConjungateGradient(x,y,w_init)
print("w_FR1:",w_FR1)
plot.plot(x_points,polynomial(x_points,w_FR1),label=
'$CG$',color="green")

w_FR2=ConjungateGradientRegular(x,y,w_init)
print("w_FR2:",w_FR2)
plot.plot(x_points,polynomial(x_points,w_FR2),label='$CG+Regular$',co
```

```
lor="blue")

plot.legend()
plot.show()
```