哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： 多项式拟合曲线

学号：1190202107
姓名： 姚舜宇

一、 实验目的

掌握最小二乘法求解，加正则项的损失函数的推导优化，使用梯度下降法和共轭梯度法进行优化，理解过拟合以及克服过拟合的方法，如增大数据量、加正则项。

二、实验要求及实验环境

1. 生成数据、加入噪声
2. 用高阶多项式函数拟合曲线
3. 推导解析解，求得两种损失函数的最优解
4. 使用梯度下降法和共轭梯度法求解最优解
5. 使用实验得到的数据解释过拟合
6. 使用不同的数据量、超参数以及不同的多项式阶数比较实验结果

环境：Windows10，python3.9，pycharm

三、设计思想（本程序中的用到的主要算法及数据结构）

定义数据集$\{(x_i,y_i)\},i\in[1,N]$，$X=\begin{bmatrix}x_1\\x_2\\\vdots\\x_N\end{bmatrix}$，$Y=\begin{bmatrix}y_1\\y_2\\\vdots\\y_N\end{bmatrix}$。设$Y=\sin(2\pi X)+\varepsilon$，其中$\varepsilon\sim N(0,\sigma)$，为高斯噪声。定义多项式拟合函数$f$，$f(X)=X_M^TW$，其中$X_M=\begin{bmatrix}(X^0)^T\\(X^1)^T\\\vdots\\(X^M)^T\end{bmatrix}$，$W=\begin{bmatrix}w_0\\w_1\\\vdots\\w_M\end{bmatrix}$。

高阶多项式拟合（有无正则项）

根据最小二乘法，对于无正则项的损失函数，可以表示为$L_1=\|f(X)-Y\|^2$。

推导过程如下：

$$\begin{aligned}L_1&=\|f(X)-Y\|^2=(f(X)-Y)^T(f(X)-Y)\\&=f^T(X)f(X)-f^T(X)Y-Y^Tf(X)+Y^TY\\&=f^T(X)f(X)-2f^T(X)Y+Y^TY\\&=W^TX_MX_M^TW-2W^TX_MY+Y^TY\end{aligned}$$

为求得最佳参数，需要对参数求导。进行微分：

$$
\begin{aligned}
d(L_1) &= d(W^T X_M X_M^T W) - 2d(W^T X_M Y) + d(Y^T Y) \\
&= d(W^T) X_M X_M^T W + W^T X_M X_M^T dW - 2d(W^T) X_M Y \\
&= tr(W^T X_M X_M^T dW) + tr(W^T X_M X_M^T dW) - 2tr(Y^T X_M^T dW) \\
&= tr\left((2W^T X_M X_M^T - 2Y^T X_M^T) dW\right)
\end{aligned}
$$

故 $\frac{\partial L_1}{\partial W} = 2X_M X_M^T W - 2X_M Y$。令其等于 0，解得：

$$
W = (X_M X_M^T)^{-1} X_M Y
$$

对于有正则项的损失函数，设惩罚率为λ，可以表示为$L_2 = \|f(X) - Y\|^2 + \lambda \|W\|^2$。

复用无正则项的推导，损失函数转化如下：

$$
L_2 = W^T X_M X_M^T W - 2W^T X_M Y + Y^T Y + \lambda W^T W
$$

为求得最佳参数，需要对参数求导。进行微分：

$$
\begin{aligned}
d(L_2) &= tr\left((2W^T X_M X_M^T - 2Y^T X_M^T) dW\right) + \lambda d(W^T) W + \lambda W^T d(W) \\
&= tr\left((2W^T X_M X_M^T - 2Y^T X_M^T + 2\lambda W^T) dW\right)
\end{aligned}
$$

故 $\frac{\partial L_2}{\partial W} = 2X_M X_M^T W - 2X_M Y + 2\lambda W$。令其等于 0，解得：

$$
W = (X_M X_M^T + \lambda I_M)^{-1} X_M Y
$$

梯度下降法

复用有正则项的多项式拟合，梯度下降法的损失函数为：

$$
L_3 = W^T X_M X_M^T W - 2W^T X_M Y + Y^T Y + \lambda W^T W
$$

其中λ为惩罚率。

要使用梯度下降法，需要计算损失函数对于参数的梯度。

$\nabla_{L_3}(W) = \frac{\partial L_3}{\partial W} = 2X_M X_M^T W - 2X_M Y + 2\lambda W$，迭代公式为$W_{i+1} = W_i - \alpha \nabla_{L_3}(W)$。其中$\alpha$为学习率，为一个超参数。

手动选择超参数学习率，惩罚率，迭代轮数，即可使用梯度下降法进行拟合。

共轭梯度法

同上，损失函数为：

$$
L_4 = W^T X_M X_M^T W - 2W^T X_M Y + Y^T Y + \lambda W^T W
$$

构造二次型$g(W) = \frac{1}{2} W^T A W - W^T B + c$。则$\nabla_g(W) = AW - B$。和损失函数对比，可以

得到$\begin{cases} A=2X_M X_M^T+2\lambda I_M \\ B=2X_M Y \\ c=Y^T Y \end{cases}$，故$\nabla_{L_4}(W)=AW-B$。需要求损失函数的最小值，令导数为0，即梯度为0向量。可以转化为线性方程组$AW=B$。注意到这里A是对称且正定的，所以可以使用共轭梯度法。

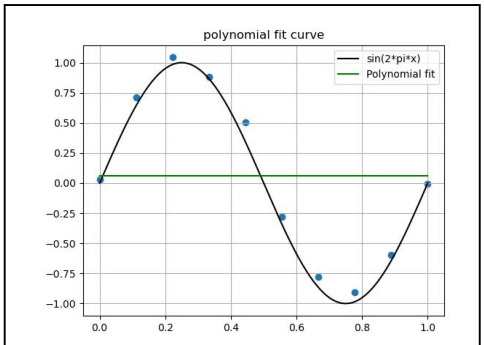
算法流程如下。首先计算初始残差$r_0=AW_0-B, p_0=-r_0$。接下来在第k轮循环中，首先计算步长$a_k=\frac{r_k^T r_k}{p_k^T A p_k}$，然后通过向共轭方向移动来更新解：$W_{k+1}=W_k+a_k p_k$。下面更新残差$r_{k+1}=r_k+a_k A p_k$，并计算系数和新的搜索方向$\beta_k=\frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$，$p_{k+1}=-r_{k+1}+\beta_k p_k$。

最终得到的W即为最终解。

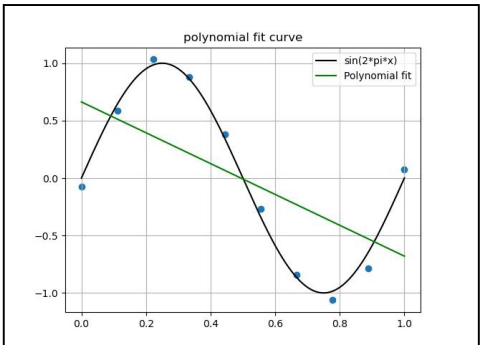
四、实验结果与分析

1. 使用解析解进行拟合

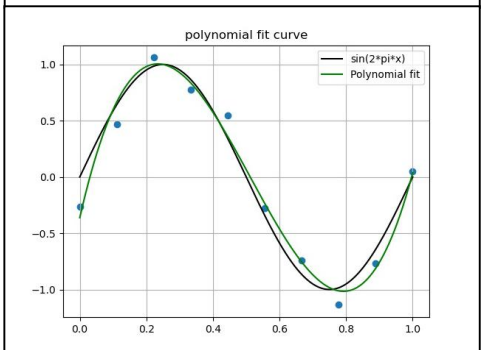
首先生成10个样本点，并加入方差为0.3的高斯噪声。当没有正则项时，对于10个样本点，加入标准差为0.1的高斯噪声，分别使用0、1、3、9阶多项式进行对样本点的拟合。可视化结果如下：黑色曲线为$\sin(2\pi x)$，绿色曲线为计算出的多项式。



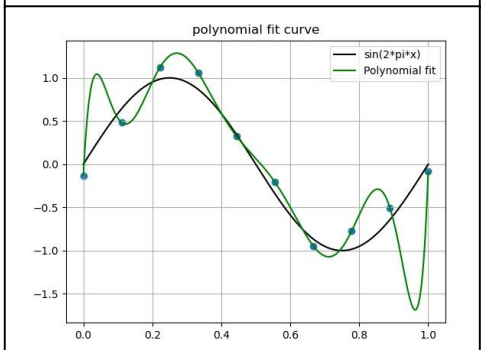
阶数：0



阶数：1



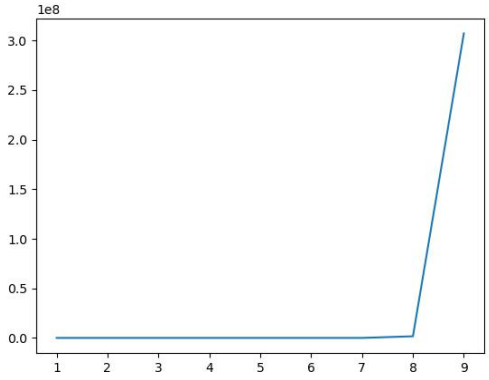
阶数：3



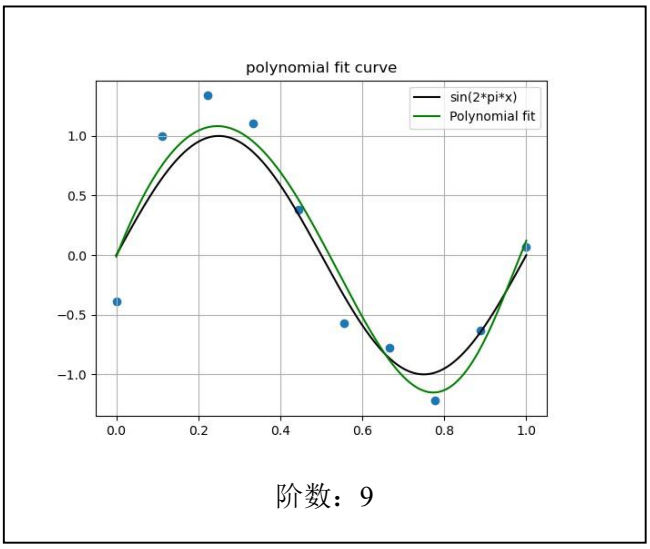
阶数：9

可以看到，对 10 个样本点而言，阶数过低时欠拟合，无法准确表示模型；阶数过高时过拟合，9 次多项式能够完美经过 10 个样本点，因为 10 次多项式共有 10 个未知数，对于 10 个样本点恰好形成一个线性方程组且系数矩阵满秩，从图中可以看出，9 次多项式进行拟合的结果泛化能力过差。

可以绘制出过拟合对损失函数的影响。可以看出，当阶数到 9 阶时，损失函数会急剧增加。



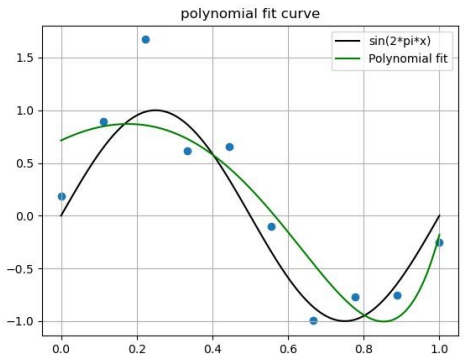
为克服过拟合，可以引入正则项。正则项衡量了模型的复杂程度，损失函数优先选择较为简单的模型。尝试不同的惩罚率，认为惩罚率取 0.001 时拟合程度较好。针对 9 阶多项式，实验结果如下。



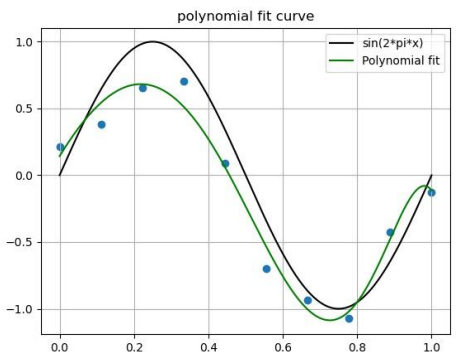
阶数：9

从图中可以看出，加入正则项后，模型有了较好的泛化能力。

如果使用不同的惩罚率，将会得到不同的拟合效果。下面分别使用 0.01 和 0.0001 的惩罚率进行实验。结果如下。



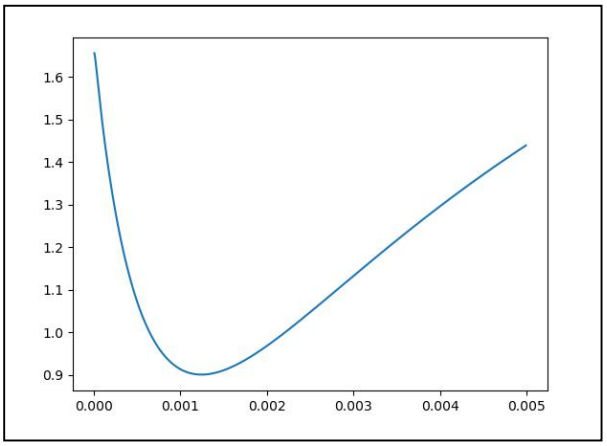
惩罚率：0.01；阶数：9



惩罚率：0.0001；阶数：9

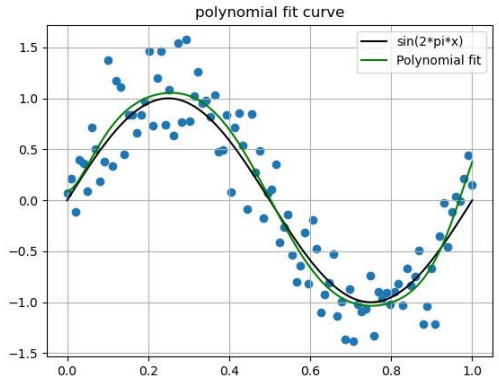
观察结果可以得到，惩罚率较低或较高时，都无法很好拟合样本。理论上分析而言，正则项衡量了模型的复杂程度，如果惩罚率较大，则计算出的模型复杂性较低；惩罚率较小，模型复杂度较高。

为更加直观地看到不同惩罚率对结果的影响，我绘制出惩罚率-损失的图像如下。横坐标为惩罚率，从 0 均匀变化到 0.005。纵坐标为损失值。



可以看出，随着惩罚率的增大，损失值先减小后增大，约在 0.0013 处取得最佳值。此结果和上述分析较为符合，如果惩罚率较大，则计算出的模型复杂性较低；惩罚率较小，模型复杂度较高，损失值都较大。

克服过拟合的另一种方法是增大数据量。使用 100 个样本点，并且不加正则项，仍然使用 9 阶多项式进行拟合，实验结果如下。

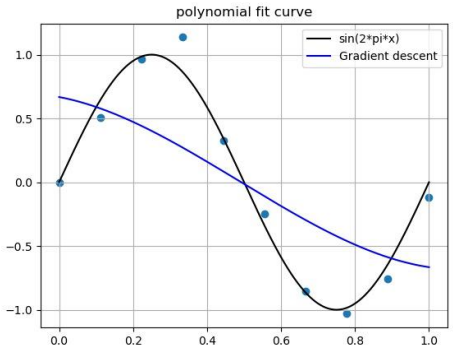


阶数：9；样本量：100

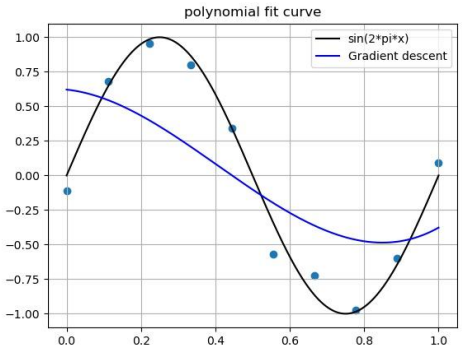
可以看到，多项式拟合的曲线和$\sin(2\pi x)$非常接近，具有良好的拟合效果。

2. 使用梯度下降法进行优化

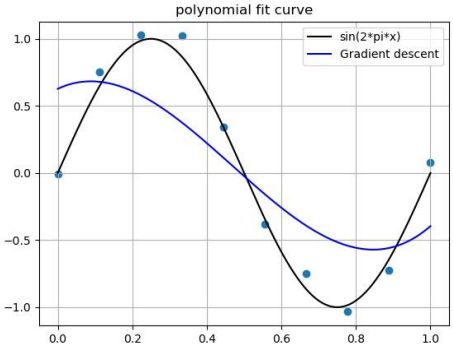
不妨固定训练集样本数为 10，学习率为 0.005，惩罚率为 0.005，对不同阶数的多项式和不同的迭代轮数进行实验。数据有：3 阶、5000 轮；3 阶、10000 轮；3 阶、50000 轮；9 阶、5000 轮、9 阶、10000 轮；9 阶、50000 轮。



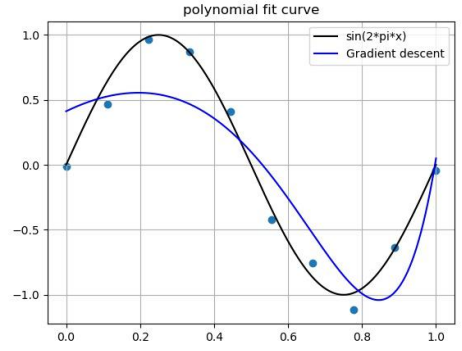
阶数：3；轮数：5000



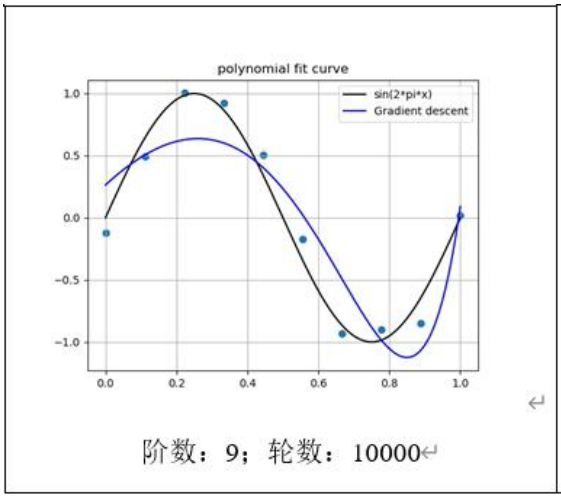
阶数：3；轮数：10000



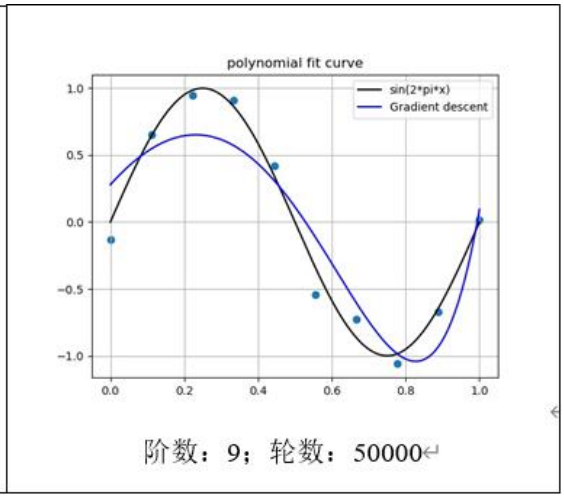
阶数：3；轮数：50000



阶数：9；轮数：5000



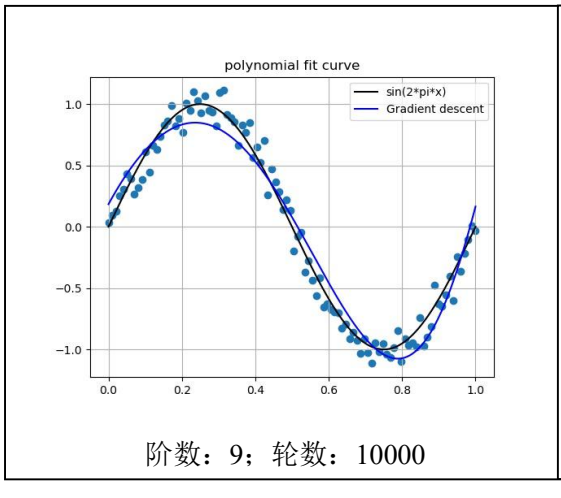
阶数：9；轮数：10000



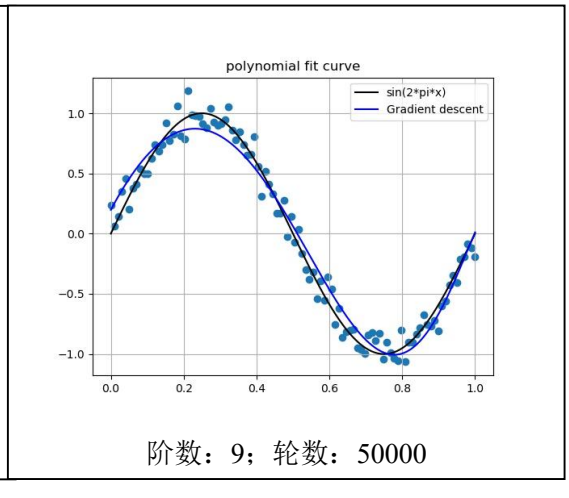
阶数：9；轮数：50000

对于梯度下降法而言，需要超参数学习率。学习率衡量了每次迭代时移动的步长。如果学习率过大，结果可能不收敛。学习率较小，可能需要更多的迭代轮数。一个好的学习率，既需要尽可能地收敛到最优解，又需要较少的迭代轮数。从上图可以看出，随着迭代次数的增加，多项式曲线越来越接近于数据点。和解析解相比，梯度下降法需要更长的计算时间，并且结果在一定程度上也不如解析解法。

下面尝试使用更多的数据点进行拟合。使用 100 个样本点，对于 9 次多项式，分别迭代 10000 次和 50000 次，观察结果。



阶数：9；轮数：10000

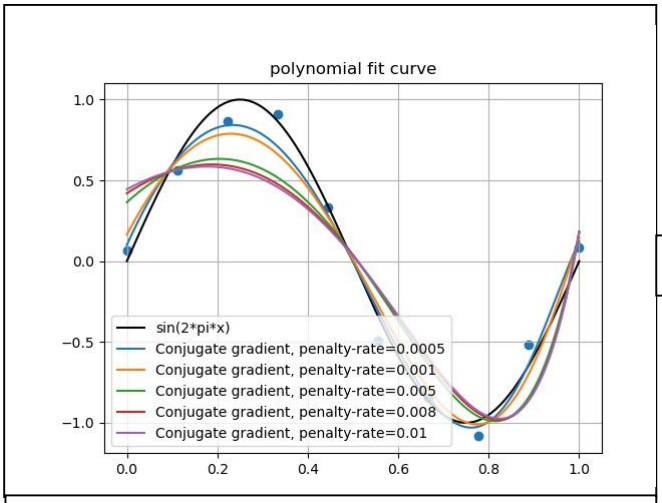


阶数：9；轮数：50000

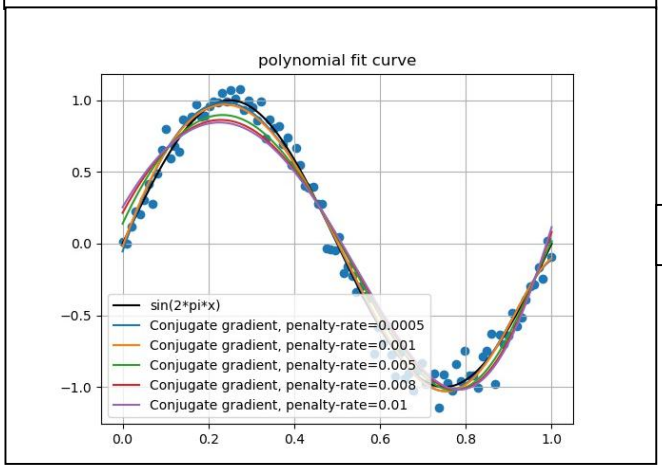
可以发现，数据点较多时，梯度下降法也能够表现出较好的拟合效果。

3. 使用共轭梯度法进行优化

首先使用 10 个样本点和 9 次多项式进行拟合。惩罚率定为 0.001。记向量维度为 n，理论上共轭梯度法只需要 n 次迭代即可求得最优解，考虑到精度原因，这里进行迭代 30 次。使用不同的惩罚率和样本量，并绘制出曲线。



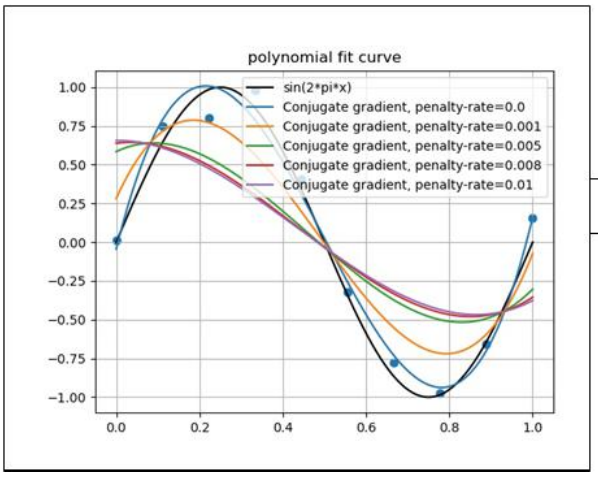
样本量：10



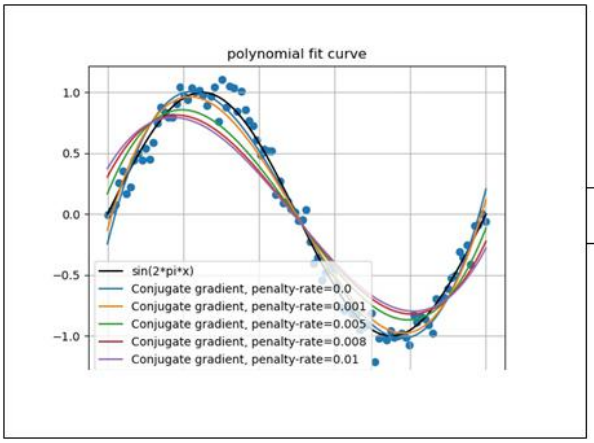
样本量：100

根据绘图结果，可以看到共轭梯度法不容易发生显著的过拟合现象，随着惩罚率的增大，模型的复杂度降低，泛化能力减弱。

如果使用 3 阶多项式进行拟合，实验结果如下。



样本量：10↵



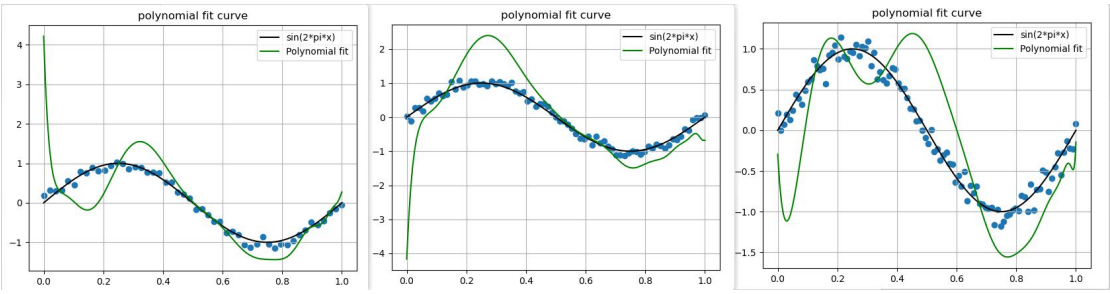
样本量：100↵

结论相似，共轭梯度法不需要正则项，并且对样本量、多项式阶数不敏感，相比于梯度下降法收敛速度更快，效果更好。

4. 尝试使用更高阶的多项式拟合数据

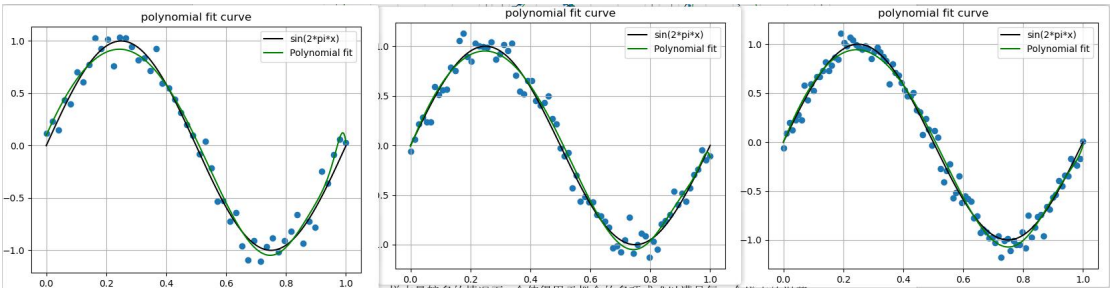
令 M=50，对于从 50 到 100 个样本量，使用不同方法进行实验。

使用解析解的方式，首先不加正则项，下图分别是 50 个样本，75 个样本，100 个样本。



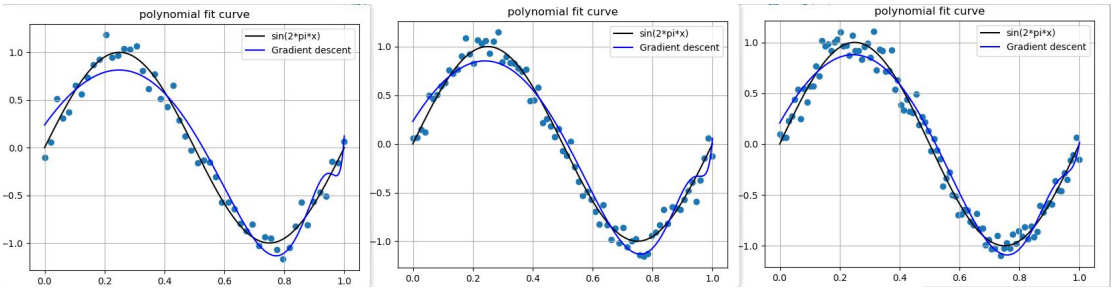
可以看出，绘制出的曲线完全无法拟合样本。初步猜想的原因是阶数太高，由于精度有限，所以会产生较大误差。

使用 0.001 惩罚率的正则项，50 个样本，75 个样本，100 个样本的结果如下。



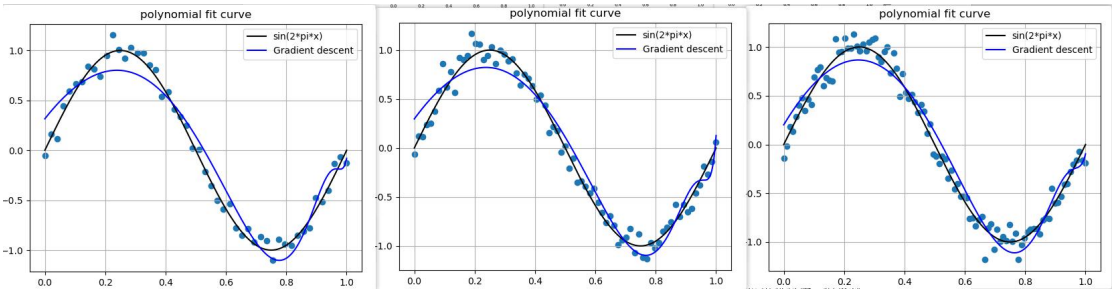
加入正则项之后，效果要好得多。因为正则项将多项式高次项的系数降低到非常小的值，精度几乎不会有影响。

使用梯度下降法进行优化，迭代次数 20000 轮，学习率 0.002，首先不加惩罚项，三种样本值结果如下。



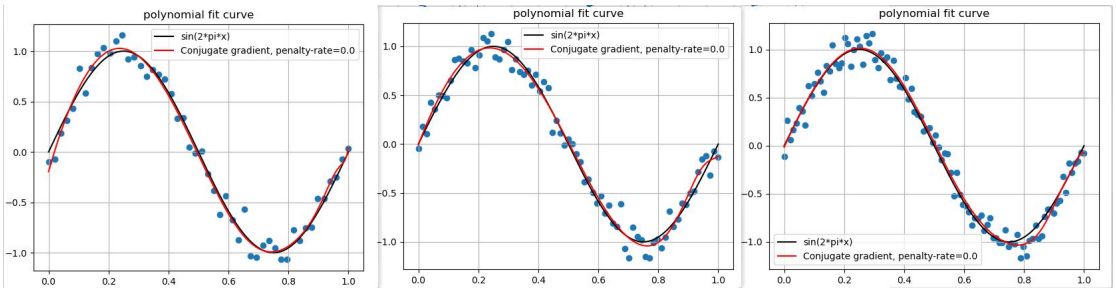
能够看出当样本数量较少时有过拟合的现象，但随着样本数量的增加，过拟合情况有所缓解。

使用 0.001 惩罚率的正则项，50 个样本，75 个样本，100 个样本的结果如下。



相比于没有正则项，泛化能力更强，并且随着样本数量的增加，过拟合情况有一定的缓解。

使用共轭梯度法进行优化。使用 50 次迭代，不加正则项，结果如下。



可以看出没有发生过拟合的现象。经过实验，当加入惩罚率为 0.001 的正则项时，拟合效果仍然较优。所以通过对比可以得到，对于高次多项式拟合，使用共轭梯度法得到的结果最好，过拟合程度最低。

五、结论

对于使用多项式拟合由函数加噪声产生的样本点，一般来说多项式次数越高，拟合能力越强。当样本点数量较少时，如果次数较高，容易产生过拟合的现象。要克服过拟合，一般有两种方法。一是加正则项，正则项是对模型复杂度的一个衡量，加入到损失函数中，我们的目标就变成了误差最小和模型尽可能简单的一个折中。惩罚率表示这两个标准的权重。二是增加样本量，样本量较多的情况下，会使得用于拟合的多项式难以满足每一个样本的误差都极小，增强了模型的泛化能力。

梯度下降法是一种迭代算法，在每一轮迭代中参数向其负梯度的方向进行更新。这里有超参数惩罚率和学习率。惩罚率同上，是正则项的权重。学习率表示了每一轮迭代参数更新的步长大小。如果步长过大，优化过程难以收敛。若步长略大，收敛速度虽然较快，但参数可能会在最优解附件来回跳跃。若步长较小，每一步的更新效果较小，需要更多的迭代轮数，但是如果迭代轮数足够，可以得到相比于大学习率更优的解。

共轭梯度法也是一种迭代算法，这种算法将解空间的向量正交化，每次迭代只需要在每

一个维度上更新到最优情况，所以理论上其迭代次数等于解空间的维度，求解速度远快于梯度下降法。并且对于高次多项式拟合，使用共轭梯度法得到的结果最好，过拟合程度最低。

六、参考文献

共轭梯度法 https://zhuanlan.zhihu.com/p/100050013

七、附录：源代码（带注释）

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


def Generate_data(min_x=0, max_x=10, num=10, scale=0.1, draw=True):
    """
    generate data  sin(2*pi*x)+Gaussian noise
    :param scale: the standard deviation of the Gaussian noise
    :param min_x: the minimum of x
    :param max_x: the maximum of x
    :param num: the number of data points
    :param draw: if you want to draw the points
    :return: X,Y both are num*1 matrix
    """
    _X = np.linspace(min_x, max_x, 1000)
    if draw is True:
        plt.plot(_X, np.sin(2 * np.pi * _X), c='k', label='sin(2*pi*x)')
    X = np.linspace(min_x, max_x, num)
    Y = np.sin(2 * np.pi * X)
    sigma = np.random.normal(loc=0, scale=scale, size=num)  # 生成高斯噪声
    Y += sigma
    if draw is True:
        plt.scatter(X, Y)
    X = np.mat(X).T
    Y = np.mat(Y).T
    return X, Y


def cal_Vandermonde_Matrix(X, m):
    """
    cal vandermonde matrix
    :param X: the x of data
    :param m: the max power of polynomial
    :return: vandermonde matrix
    """
```

```
    num = X.shape[0]
    Xm = np.mat(np.ones((m + 1, num)))
    t = X.T
    for i in range(1, m + 1):
        Xm[i] = t
        t = np.multiply(t, X.T)
    return Xm


def Polynomial_fit(X, Y, m=1, penalty_rate=float(0), draw=True):
    """
    use polynomial to fit the curve
    :param X: the x of data
    :param Y: the y of data
    :param m: the max power of polynomial
    :param penalty_rate: penalty rate for regularization loss function
    :param draw: if you want to draw the curve
    :return: a list of the parameters of the polynomial
    """
    Xm = cal_Vandermonde_Matrix(X, m)
    W = (Xm @ Xm.T + penalty_rate * np.mat(np.identity(m + 1))).I @ Xm @ Y
# 计算解析解
    loss = W.T @ Xm @ Xm.T @ W - 2 * W.T @ Xm @ Y + Y.T @ Y + penalty_rate *
 W.T @ W  # 计算损失函数的值
    print('loss: ' + str(np.array(loss.tolist()).flatten()[0]))
    w = np.array(W.tolist()).flatten()
    func = np.poly1d(w[::-1])
    x = np.linspace(min_x, max_x, 1000)
    if draw is True:
        plt.plot(x, func(x), c='g', label='Polynomial fit')
    return w


def Gradient_descent(X, Y, m=1, epoch=1000, lr=0.01, penalty_rate=float(0),
draw=True):
    """
    use gradient descent to optimization  the curve
    :param X: the x of data
    :param Y: the y of data
    :param m: the max power of polynomial
    :param epoch: the number of iterations
    :param lr: the learning rate
    :param penalty_rate: penalty rate for regularization loss function
    :param draw: if you want to draw the curve
```

```
    :return: a list of the parameters of the polynomial
    """
    Xm = cal_Vandermonde_Matrix(X, m)
    W = np.mat(np.random.random(size=(m + 1, 1)))  # 随机初始化初始解
    for i in range(epoch):  # 迭代 epoch 轮进行训练
        W = W - lr * (2 * Xm @ Xm.T @ W - 2 * Xm @ Y + 2 * penalty_rate * W)  # 向梯度方向移动
        loss = W.T @ Xm @ Xm.T @ W - 2 * W.T @ Xm @ Y + Y.T @ Y + penalty_rate * W.T @ W  # 计算损失函数
        print('epoch ' + str(i + 1) + '  loss: ' + str(np.array(loss.tolist()).flatten()[0]))
    w = np.array(W.tolist()).flatten()
    func = np.poly1d(w[::-1])
    x = np.linspace(min_x, max_x, 1000)
    if draw is True:
        plt.plot(x, func(x), c='b', label='Gradient descent')
    return w


def Conjugate_gradient(X, Y, m=1, epoch=10, penalty_rate=float(0), draw=True):
    """
    use conjugate descent to optimization  the curve
    :param X: the x of data
    :param Y: the y of data
    :param m: the max power of polynomial
    :param epoch: the number of iterations
     :param penalty_rate: penalty rate for regularization loss function
     :param draw: if you want to draw the curve
     :return: a list of the parameters of the polynomial
     """
     Xm = cal_Vandermonde_Matrix(X, m)
     W = np.mat(np.random.random(size=(m + 1, 1)))  # 随机初始化初始解
     A = 2 * Xm @ Xm.T + 2 * penalty_rate * np.mat(np.identity(m + 1))
     B = 2 * Xm @ Y
     grad = A @ W - B
     p = -grad
     for i in range(epoch):
         a = (grad.T @ grad).tolist()[0][0] / (p.T @ A @ p).tolist()[0][0]  # 计算最佳步长
         W = W + a * p
         grad1 = grad + a * A @ p
         beta = (grad1.T @ grad1).tolist()[0][0] / (grad.T @ grad).tolist()[0][0]
```

```
115.         grad = grad1
116.         p = -grad + beta * p  # 计算最佳方向
117.         loss = W.T @ Xm @ Xm.T @ W - 2 * W.T @ Xm @ Y + Y.T @ Y + penalty_r
    ate * W.T @ W  # 计算损失函数
118.         print('epoch ' + str(i + 1) + '  loss: ' + str(np.array(loss.tolist
    ()).flatten()[0]))
119.     w = np.array(W.tolist()).flatten()
120.     func = np.poly1d(w[::-1])
121.     x = np.linspace(min_x, max_x, 1000)
122.     if draw is True:
123.         plt.plot(x, func(x), c='r', label='Conjugate gradient, penalty-rate
    ='+str(penalty_rate))
124.     return w
125.
126.
127. def trainModel(method):
128.     """
129.     use train set to train a model
130.     :param method:
131.     :return:
132.     """
133.     if method == 'polynomial_fit':
134.         Polynomial_fit(X, Y, m=m, penalty_rate=0.001)  # 使用多项式拟合
135.     elif method == 'gradient_descent':
136.         Gradient_descent(X, Y, m=m, epoch=10000, lr=0.005, penalty_rate=0.0
    05)  # 使用梯度下降法优化
137.     elif method == 'Conjugate_gradient':
138.         Conjugate_gradient(X, Y, m=m, epoch=30, penalty_rate=0.001)  # 使用
    随机梯度下降法优化
139.     else:
140.         print('wrong')
141.         exit(0)
142.
143.
144. def testModel(method):
145.     """
146.     use test set to test the model
147.     :param method: different optimization method
148.     :return: none
149.     """
150.     X_test, Y_test = Generate_data(min_x=min_x, max_x=max_x, num=20, scale=
    0.1, draw=False)  # 测试集
151.     prlist = []
152.     losslist = []
```

```
    for i in range(1, 500):
        pr = 0.00001 * i
        prlist.append(pr)
        w = 0
        if method == 'polynomial_fit':
            w = Polynomial_fit(X, Y, m=m, penalty_rate=pr, draw=False)
        elif method == 'gradient_descent':
            w = Gradient_descent(X, Y, m=m, epoch=20000, lr=0.005, penalty_rate=pr, draw=False)
        elif method == 'Conjugate_gradient':
            w = Conjugate_gradient(X, Y, m=m, epoch=30, penalty_rate=pr, draw=False)
        else:
            print('wrong')
            exit(0)
        func = np.poly1d(w[::-1])
        Y_pre = np.mat(func(np.array(X_test)))
        l = Y_test - Y_pre
        loss = l.T @ l
        loss = loss.tolist()[0][0]
        print(loss)
        losslist.append(loss)
    plt.plot(prlist, losslist)
    plt.show()


min_x = 0  # x 的最小值
max_x = 1  # x 的最大值
num_data = 100  # 样本数目
m = 3  # 拟合多项式的阶数
X, Y = Generate_data(min_x=min_x, max_x=max_x, num=num_data, scale=0.1)  # 生成样本点

# 训练模型
trainModel('polynomial_fit')
trainModel('gradient_descent')
trainModel('Conjugate_gradient')

plt.title('polynomial fit curve')
plt.legend()
plt.grid()
plt.savefig('result.jpg')
plt.show()

```

```
# 测试模型
# testModel('polynomial_fit')
```