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：1181000420

姓名：韦昆杰

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

- Windows10
- PyCharm
- VSCode

三、设计思想(本程序中的用到的主要算法及数据结构)

1.算法原理

我们采用多项式函数y(x,w)来模拟正弦函数曲线

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

下面是：建立误差函数，测量每个样本点目标值t与预测函数y之间的误差

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

然后可以令 $X=\begin{bmatrix}1 & x_1 & \cdots & x_1^m\\ 1 & x_2 & \cdots & x_2^m\\ \vdots & \vdots & \ddots & \vdots\\ 1 & x_N & \cdots & x_N^m\end{bmatrix}$ $W=\begin{bmatrix}w_0\\ w1\\ \vdots\\ w_m\end{bmatrix}$ $T=\begin{bmatrix}t_1\\ t_2\\ \vdots\\ t_N\end{bmatrix}$

将(2)式化成矩阵的形式如下：

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

E是w的二次函数，对其求偏导, 结果如下：

$$\frac{\partial E}{\partial w} = X'(XW - T) \tag{4}$$

然后设导数为0，存在唯一解w^*

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

下面我们在优化目标函数$E(w)$中加入对w的惩罚

$$\widetilde{E}(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 + \frac{\lambda}{2}||w^2|| \tag{6}$$

将(6)式化成矩阵的形式如下：

$$\widetilde{E}(w) = \frac{1}{2}(XW - T)'(XW - T) + \frac{\lambda}{2}W'W \tag{7}$$

$\widetilde{E}$是w的二次函数，对其求偏导, 结果如下：

$$\frac{\partial \widetilde{E}}{\partial w} = X'XW - X'T + \lambda W \tag{8}$$

然后设导数为0，存在唯一解w^*

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

其中I为单位矩阵

2.算法的实现

- 解析解法(无正则项) 由(5)式可以通过X和T求出W*，下面是具体实现的Python函数代码:

```
def fit_without_regulation(self, X, T):
    """
    计算未加入正则项的数值解W
    param X: 二维array X
    param T: array T
    return: 模型参数W
    """
    return np.linalg.pinv(X) @ T
```

- 解析解法(有正则项) 由(9)式可以通过X和T求出W*，下面是具体实现的Python函数代码:

```
def fit_with_regulation(self, X, T, Lambda=1e-7):
    """
    计算加入正则项的数值解W
    param X: 二维array X
    param T: array T
    param Lambda:权重参数
    return: 模型参数W
    """
    return np.linalg.solve(X.T @ X + Lambda * np.identity(self.m + 1), X.T @ T)
```

- 梯度下降法 梯度下降法(gradient descent)是最佳化理论里面的一个一阶找最佳解的一种方法,必须向函数上当前点对应梯度（或者是近似梯度）的反方向的规定步长距离点进行迭代搜索. 在本实验中，梯度为W*，上面的公式(4)即为无正则项的梯度：

$$\frac{\partial E}{\partial w} = X'(XW - T) \tag{4}$$

而上面的公式(8)即为有正则项的梯度：

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

因此每次更新W公式如下：

$$W = W - learning\_rate * \frac{\partial E}{\partial w}$$

下面是具体实现的Python函数代码:

```
def gradient_descent(self, X, T, rate=0.1, precision=1e-1):
    """
    通过梯度下降法计算给定learning rate和精度要求所需要的迭代次数
    param X: 二维array X
    param T: array T
    param rate: learning rate
    return: 计算出要求精度所需要的迭代次数
    """
    times = 0
    W = np.zeros((self.m + 1, 1))  # 初始化W为全为0的列向量
    while self.error(X, W, T) > precision:
        last_error = self.error(X, W, T)
        W = W - rate * X.T @ (X @ W - T)
        if self.error(X, W, T) > last_error:
            rate = rate / 2
        times += 1
    return times
```

- 共轭梯度法 下面是wikipedia给出的算法实现：

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
if $\mathbf{r}_0$ is sufficiently small, then return $\mathbf{x}_0$ as the result
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat

$$\alpha_k := \frac{\mathbf{r}_k^\mathsf{T} \mathbf{r}_k}{\mathbf{p}_k^\mathsf{T} \mathbf{A} \mathbf{p}_k}$$

$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small, then exit loop

$$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T} \mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T} \mathbf{r}_k}$$

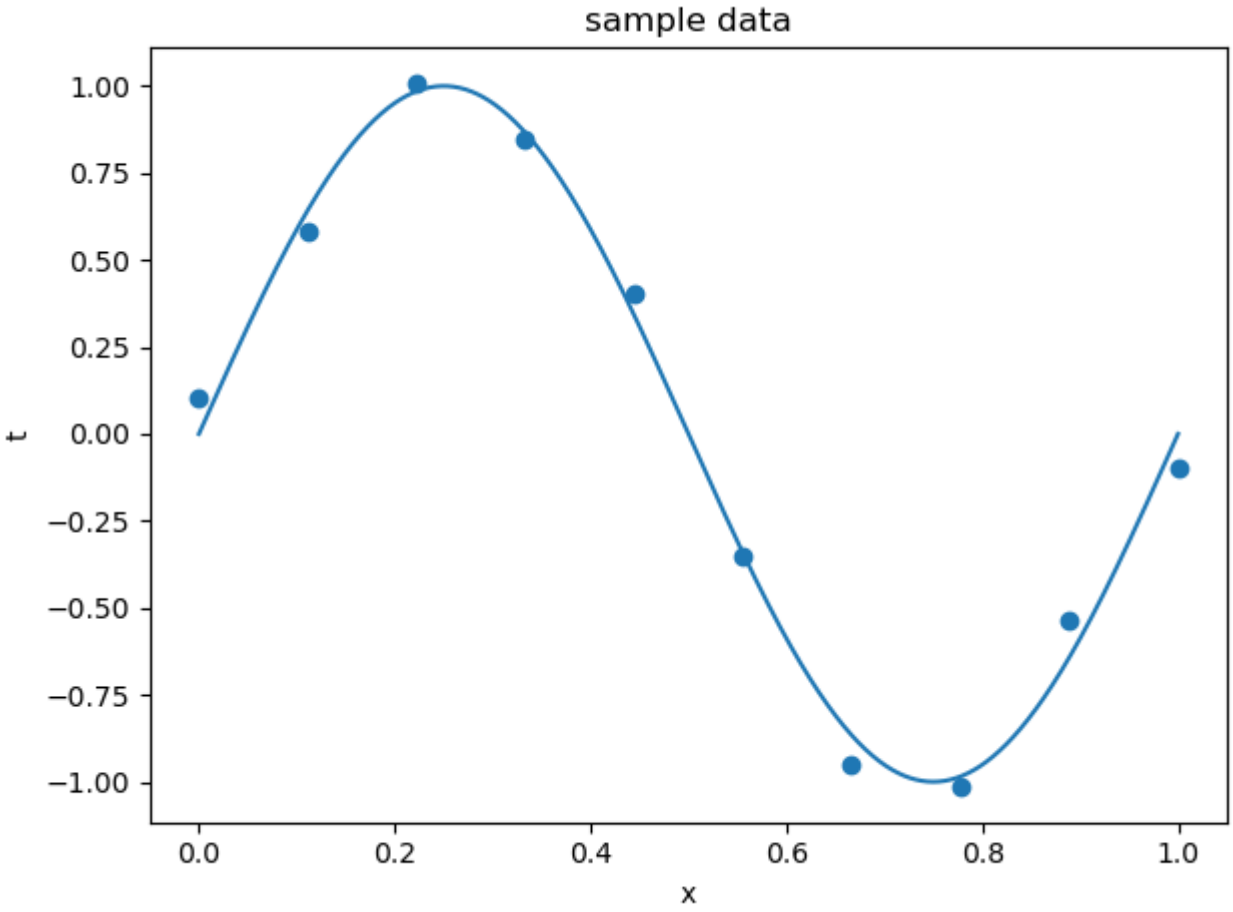
$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
$k := k + 1$
end repeat
return $\mathbf{x}_{k+1}$ as the result
下面是具体实现的Python函数代码:

```
def conjugate_gradient(self, X, T, precision=1e-5, times=0,Lambda=0):
        """
        通过共轭梯度法计算给定精度要求所需要的迭代次数，
        参考 http://en.wikipedia.org/wiki/Conjugate_gradient_method

        param X: 二维array X
        param T: array T
        param precision:精度要求，默认为0.1
        param times: 迭代次数
        param Lambda:惩罚项参数，默认为0，即为无正则项的共轭梯度
        return: 要求精度要求所需要的迭代次数和W
        """

        # 转化成Ax = b的形式求解
        A = X.T @ X + Lambda * np.identity(self.m+1)        #Lambda=0时为无正则项的共轭梯度
        x = np.zeros((self.m + 1, 1))                       #Lambda>0时为有正则项的共轭梯度
        b = X.T @ T

        r = b - np.dot(A, x)
        p = r
        rsold = r.T @ r
        # 计算给定迭代次数得到的W
        if times != 0:
            for i in range(times):
                Ap = A @ p
                alpha = rsold[0][0] / np.dot(p.T, Ap)[0][0]
                x = x + alpha * p
                r = r - alpha * Ap
                rsnew = r.T @ r
                p = r + (rsnew / rsold) * p
                rsold = rsnew
            return x
        times = 0
        while self.error(X, x, T) > precision:
            Ap = A @ p
            alpha = rsold[0][0] / np.dot(p.T, Ap)[0][0]
            x = x + alpha * p
            r = r - alpha * Ap
            rsnew = r.T @ r
            p = r + (rsnew / rsold) * p
            rsold = rsnew
            times += 1
        return times, x
```

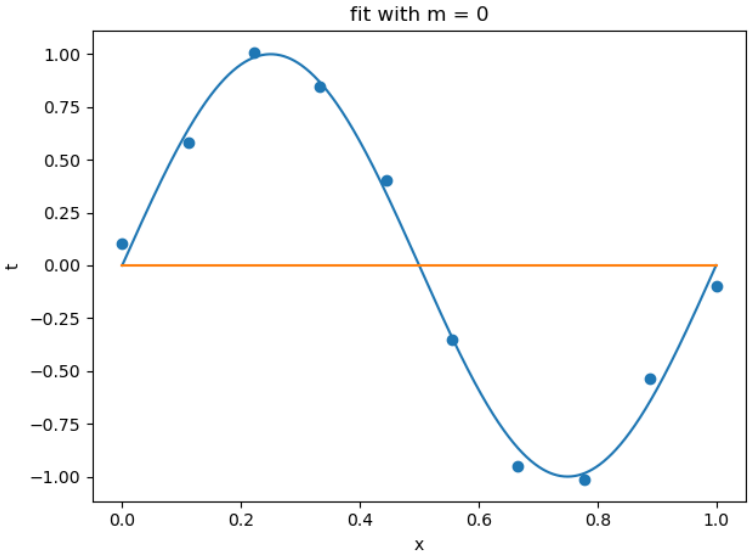
四、实验结果分析

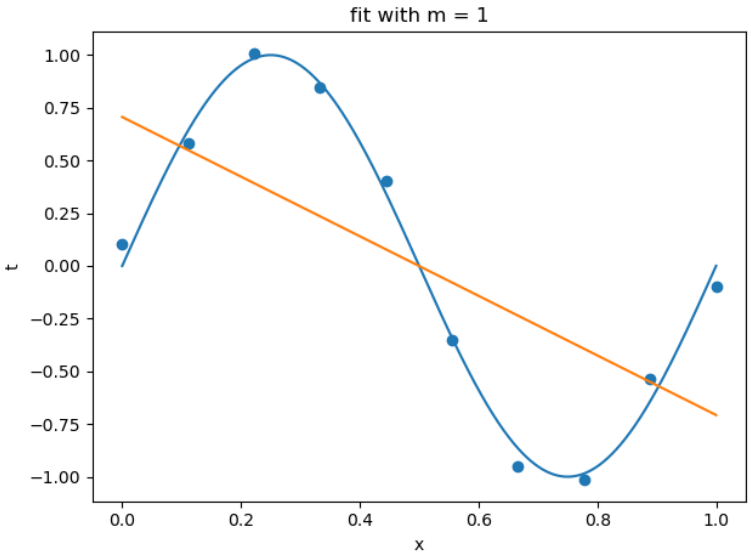
1.不带惩罚项的解析解

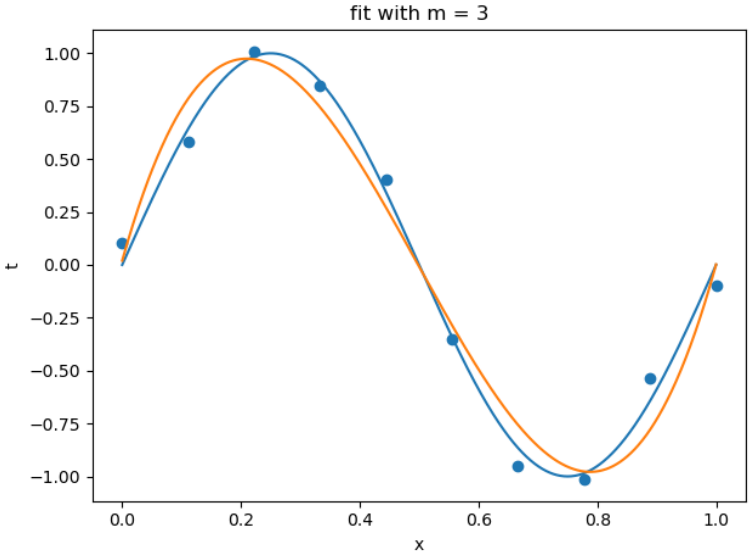
先利用$sin(2\pi x)$产生样本,样本数据量$size = 10$，x均匀分布在$[0, 1]$,然后对每个目标值t加一个0均值的高斯噪声，这样就产生了样本数据:

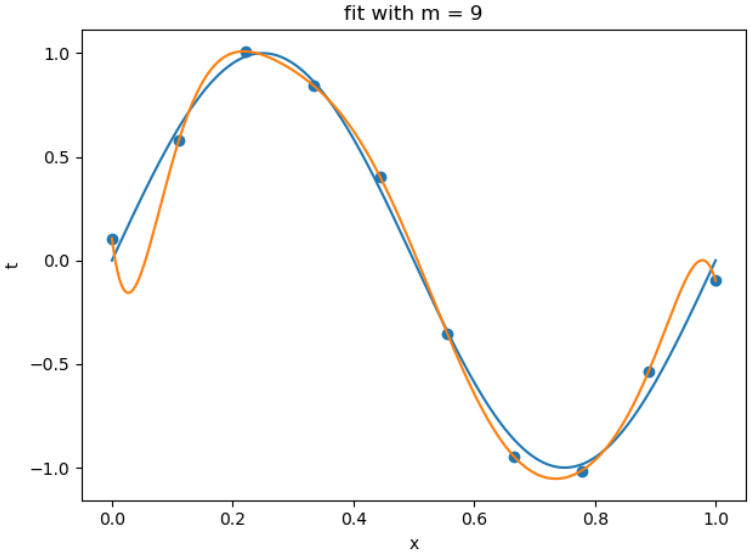


在用于拟合的多项式函数阶数取不同值时，观察最佳拟合曲线情况，如下图是M分别取0-9的曲线拟合图(未加惩罚项)：

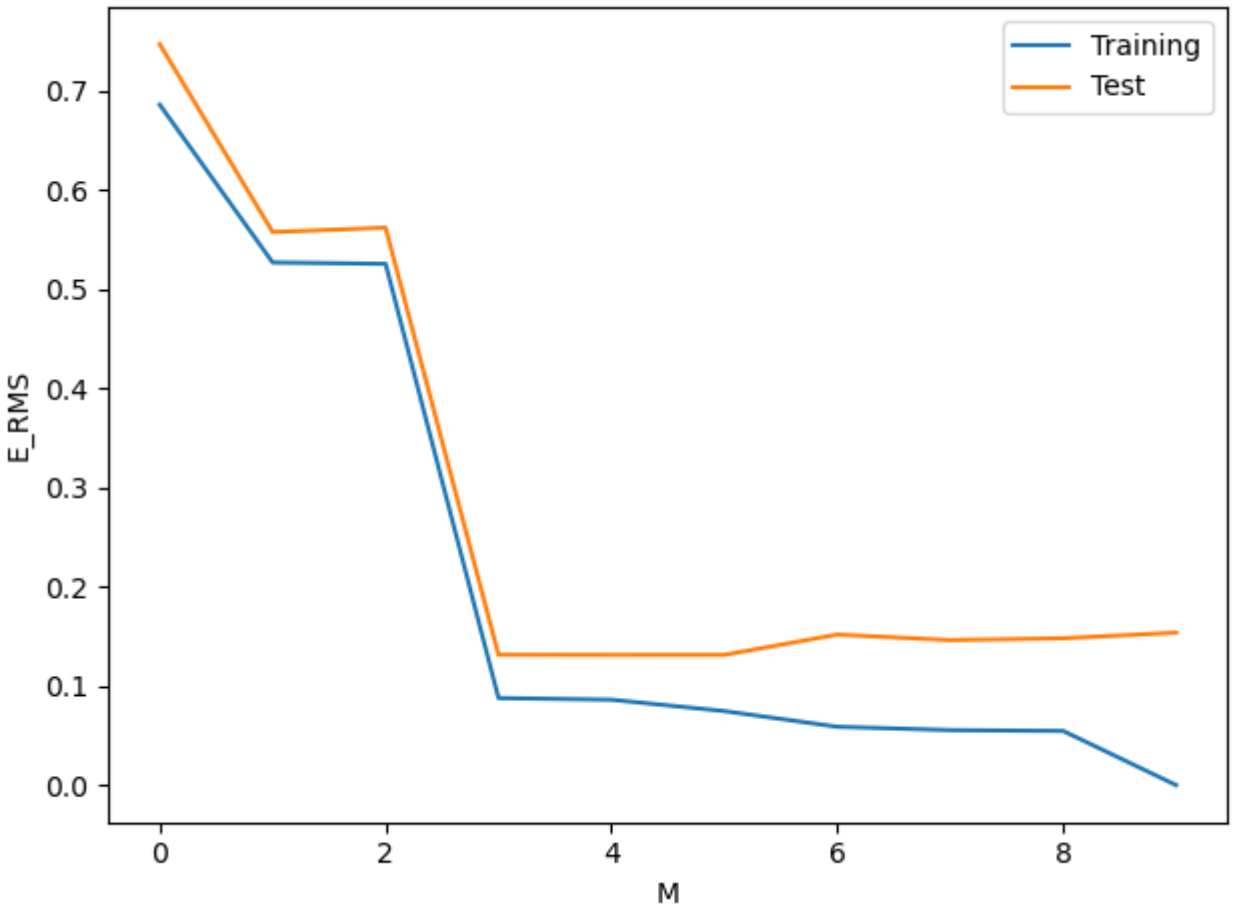








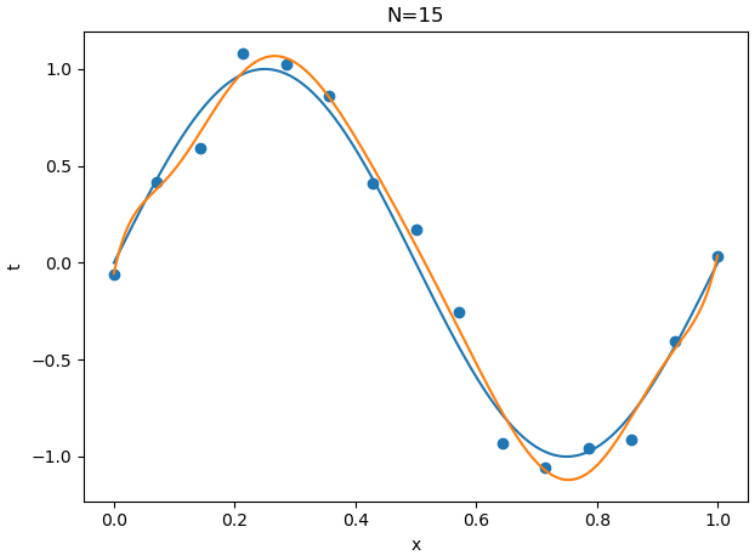
同时我们比较训练数据和测试数据的方均根值E_{RMS}随M的变化情况，如下图：

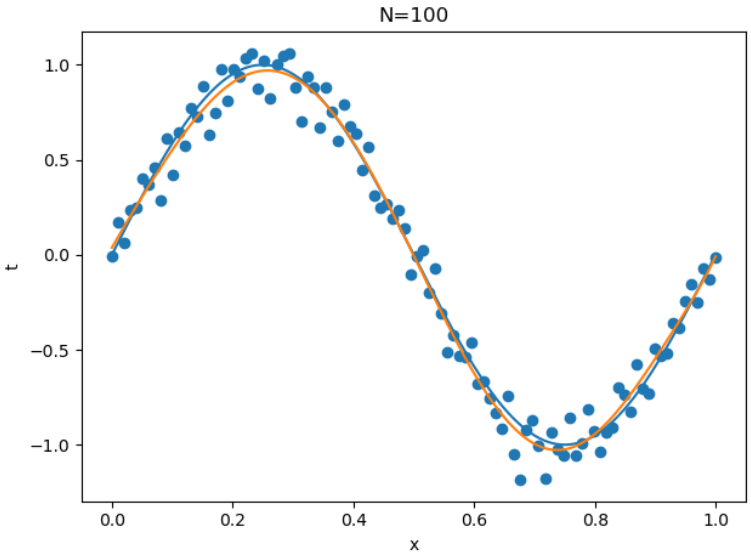


我们可以发现

- 当M较小时，随着M的增大，训练数据和测试数据的误差都在降低，但当M增大到某一数值后，训练数据的方均根值E_{RMS}仍在降低，而测试数据的方均根值E_{RMS}却在变大，说明出现了过拟合

下面我们来分析训练样本数量对拟合的作用，我们可以改变样本数量，来观察曲线的拟合情况，下面是N=15,M=9和N=100，M=9的比较：

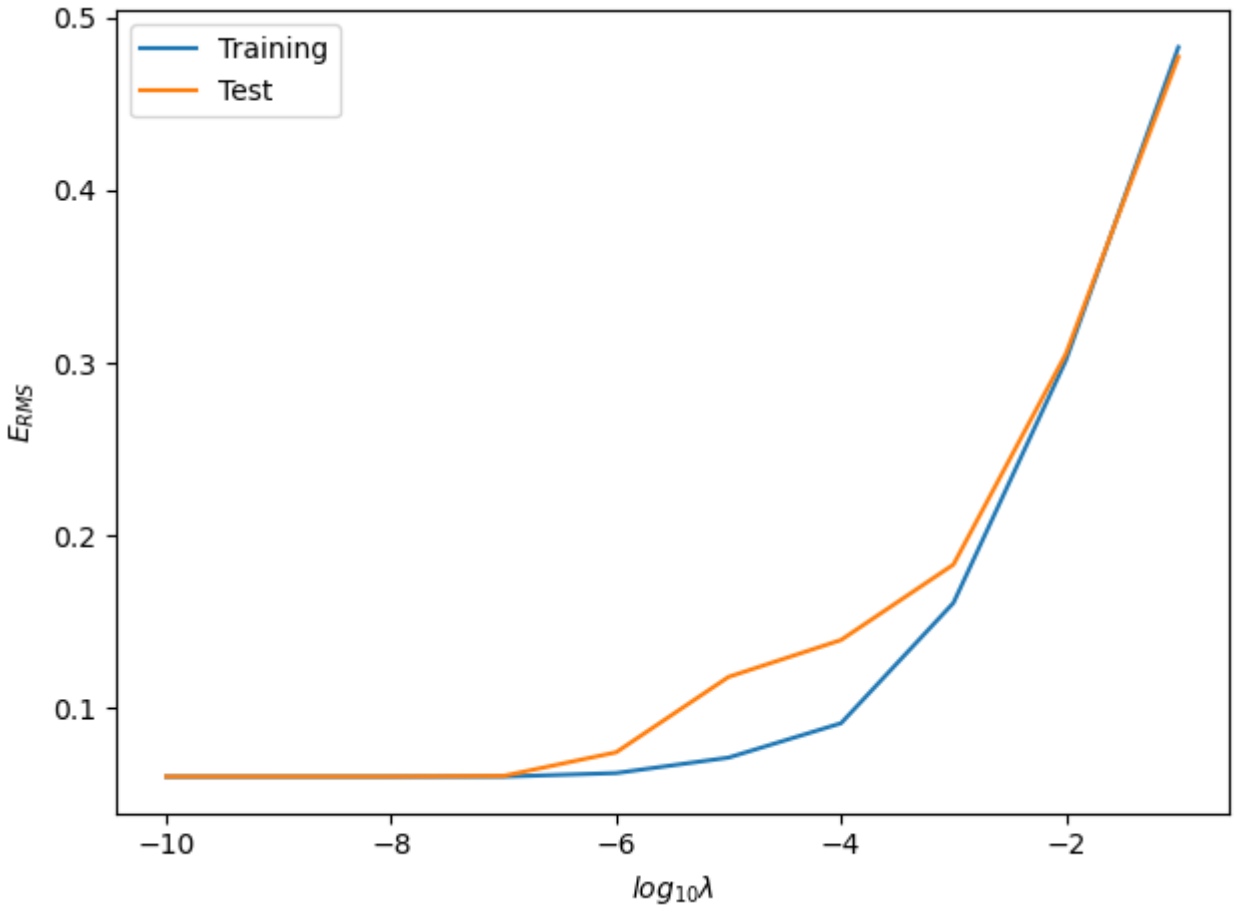




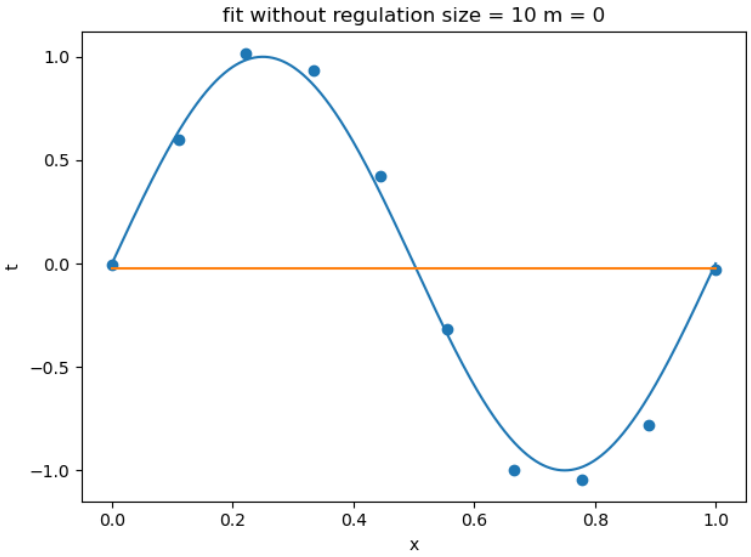
- 我们发现增大样本数量可减少过学习程度

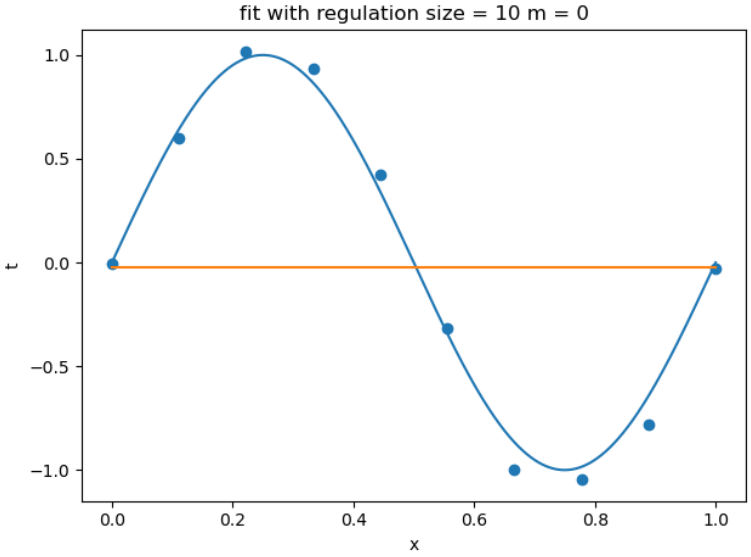
2.带惩罚项的解析解

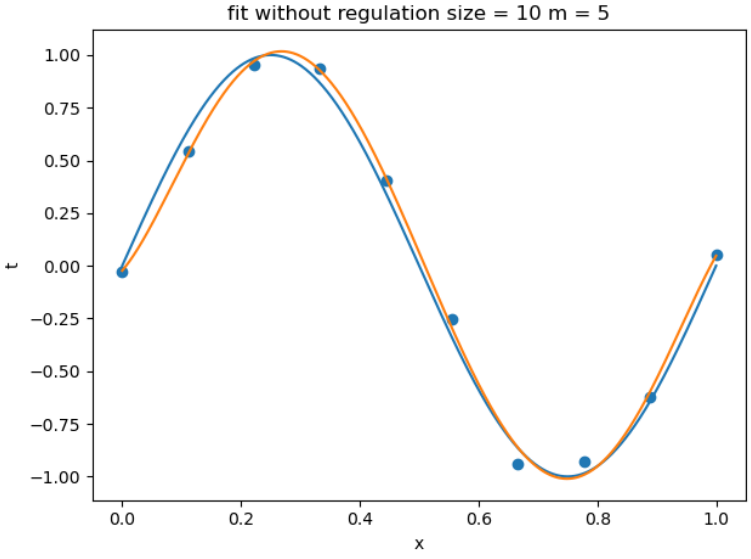
可以在优化目标函数$E(w)$中加入对w的惩罚作为正则项，对于权重参数λ的选择，我们分为训练数据和测试数据画出E_{RMS}随λ的变化情况(阶数=5，数据量=10)，结果发现$\lambda = 10^{-7}$时效果最好,如下图:

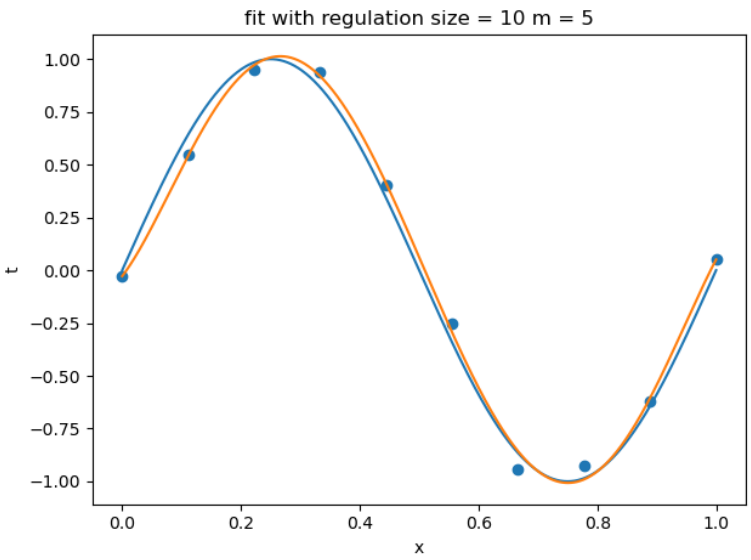


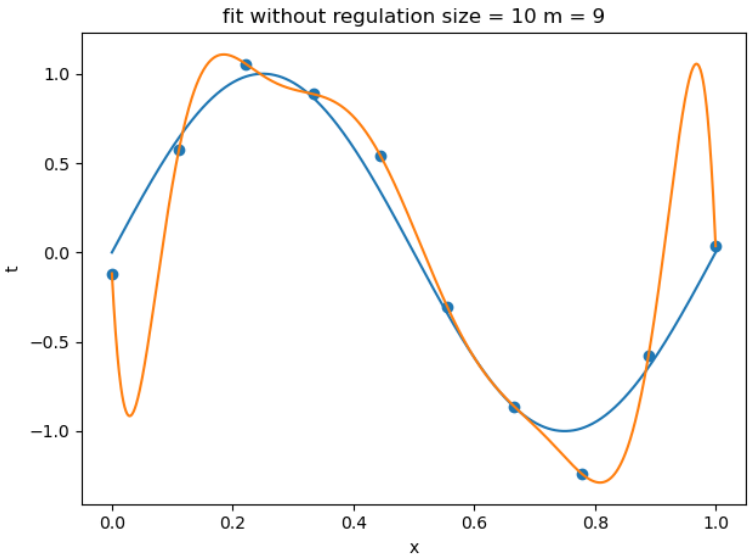
因此我们设置权重参数λ为10^{-7}，在用于拟合的多项式函数阶数取不同值时，观察最佳拟合曲线情况，如下是size=10,m分别等于0，5，9时的曲线拟合图(有惩罚项与无惩罚项对比):

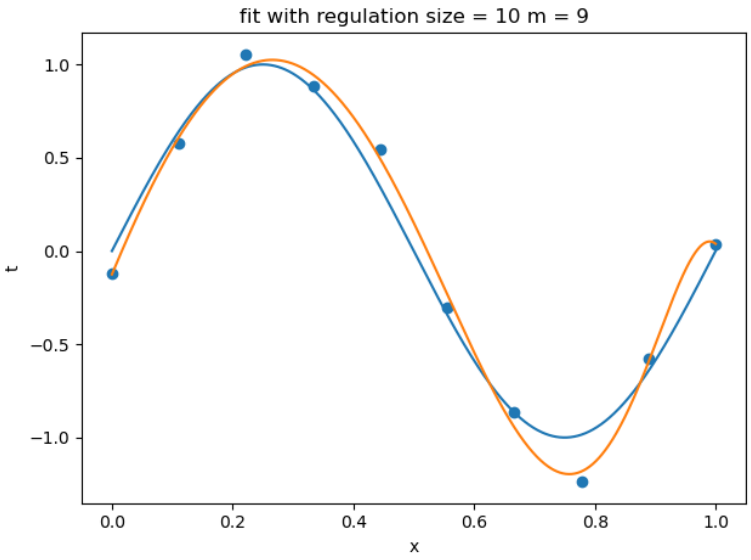

fit without regulation size = 10 m = 0
t
x
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0


fit with regulation size = 10 m = 0
t
x
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0


fit without regulation size = 10 m = 5
t
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

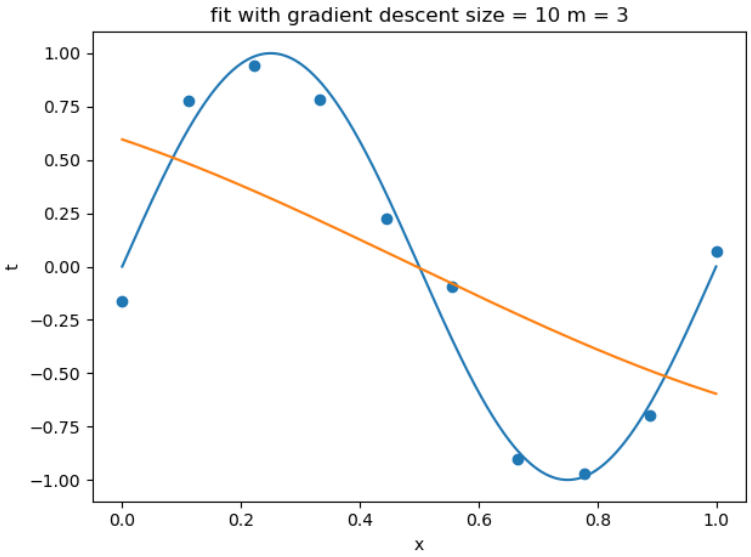


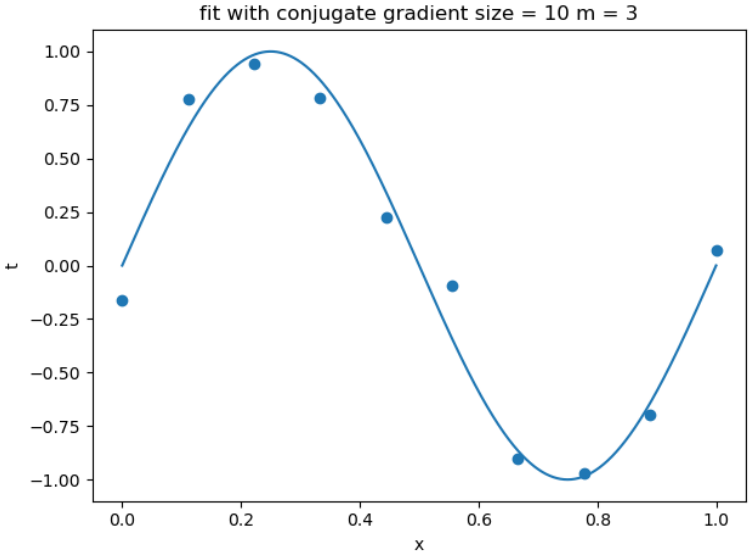


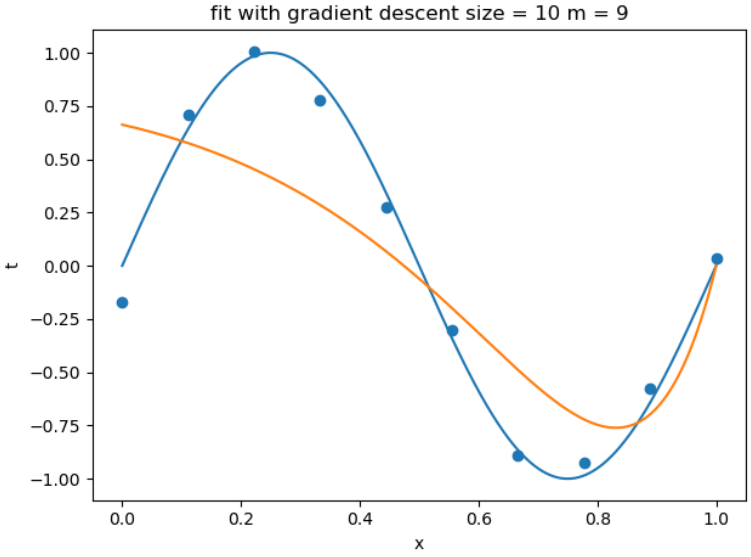


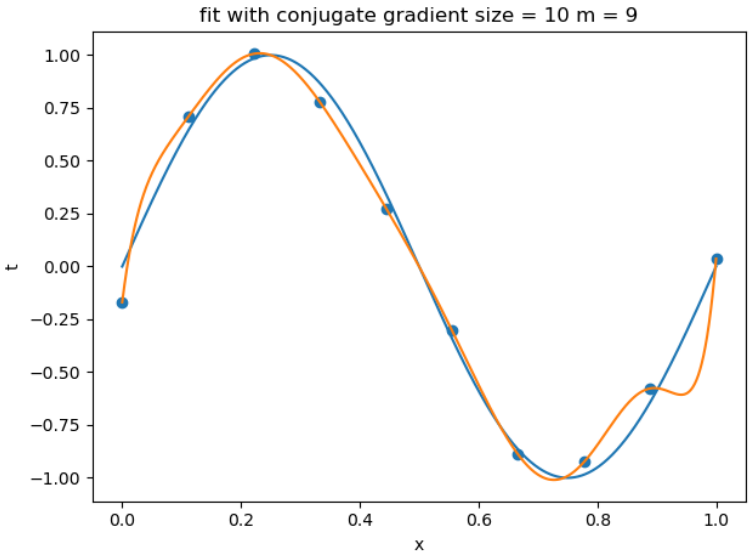
3.梯度下降法和共轭梯度法拟合情况比较

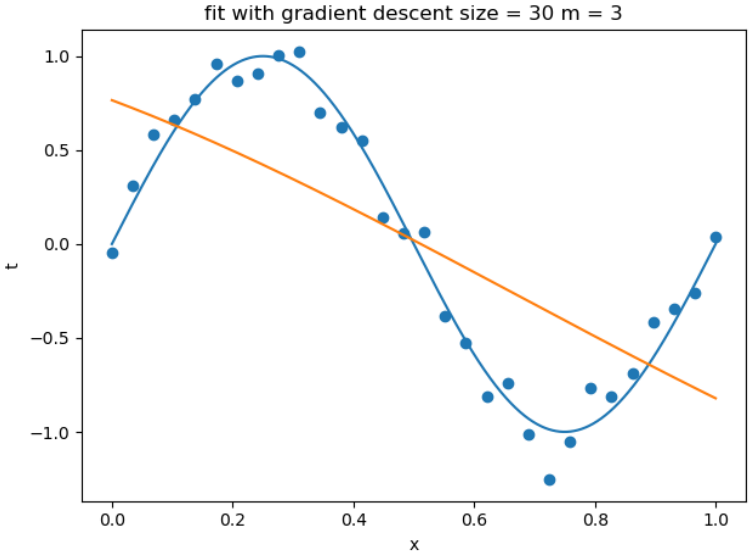
上面我们比较了在解析解下有无惩罚项拟合的比较，下面我们比较梯度下降法和共轭梯度法对有无惩罚项的拟合情况(迭代次数为100次)：

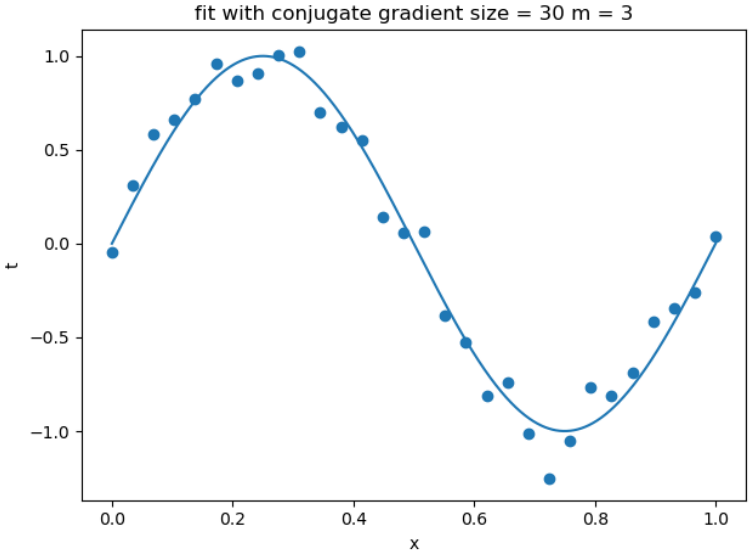

fit with gradient descent size = 10 m = 3
t
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

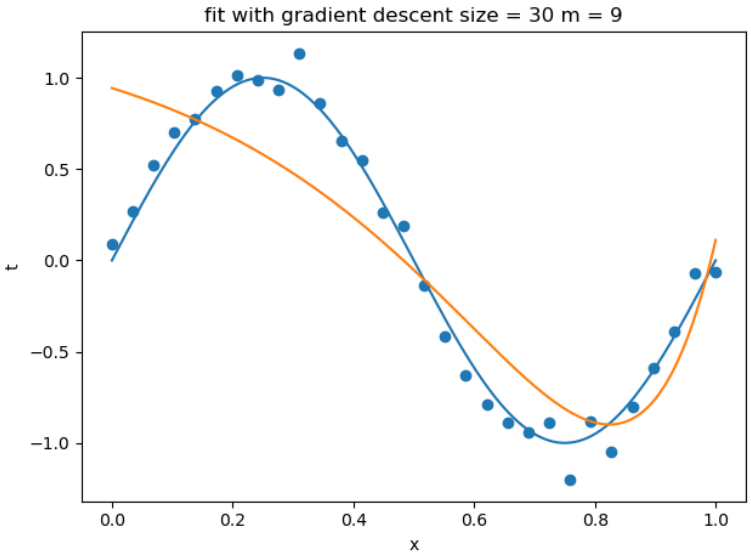

fit with conjugate gradient size = 10 m = 3
t
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

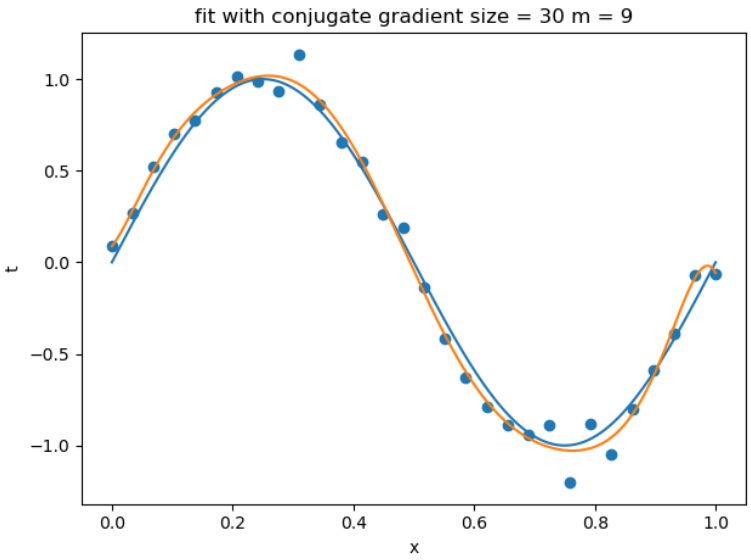

fit with gradient descent size = 10 m = 9
t
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0


fit with conjugate gradient size = 10 m = 9
t
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0


fit with gradient descent size = 30 m = 3
t
x
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0


fit with conjugate gradient size = 30 m = 3
t
x
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0





- 从上面的对比可以发现，相同的迭代次数，共轭梯度比梯度下降的拟合情况好得多

下面我们对使用梯度下降法和共轭梯度法达到相同精度所需次数的简单比较，发现：

```
使用梯度下降法，精度要求在1*10^-1所需次数:38305
使用共轭梯度法，精度要求在1*10^-1所需次数:5
使用共轭梯度法，精度要求在1*10^-5所需次数:161
```

- 共轭梯度法相较梯度下降法，对于相同训练数据达到相同精度所需次数小得多，收敛速度更快

五、结论

- 阶数M较小时，模型表达能力有限（不灵活），方均根误差大
- 阶数M=3-8时，拟合较好，方均根误差较小
- 阶数M过大时，虽然误差进一步降低，但发生了过拟合，对新的测试数据误差反而变大
- 增大样本数量可减少过学习程度
- 在优化目标函数加入正则项作为惩罚可以避免w^*具有过大的绝对值，从而减少过拟合
- 梯度下降的收敛速度慢，所需迭代次数高，而共轭梯度收敛速度快，所需迭代次数低，对多项式曲线拟合的效果更好

六、参考文献

- [1] gradient descent wikipedia
- [2] conjugate gradient method wikipedia
- [3] 深入浅出--梯度下降法及其实现
- [4] 机器/深度学习-基础数学(二):梯度下降法(gradient descent)

七、附录:源代码(带注释)

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

warnings.filterwarnings("ignore")


class lab1:
    def __init__(self, size, m):
        self.size = size  # 样本数据量
        self.m = m  # 拟合多项式函数的阶数

    def generate_data(self):
        """
        产生size个数据
        return:
        x:x轴点的array
        y:y轴点的array, y=sin(2*pi*x)
        """
        x = np.linspace(start=0, stop=1, num=self.size)
        mu, sigma = 0, 0.1  # mean and standard deviation
        y = np.sin(2 * np.pi * x) + np.random.normal(mu, sigma, size=self.size)
        return x, y

    def trans(self, x, y):
        """
        将x,y两个array根据多项式函数的阶m转化成相应的array X, T
        param x: x轴点的array
        param y: y轴点的array, y=sin(2*pi*x)
        return:
        X, T:对应的两个array
        """
        X = np.zeros((self.size, self.m + 1))
        for i in range(self.m + 1):
            for j in range(self.size):
                X[j, i] = np.power(x[j], i)
        T = y.reshape(self.size, 1)
        return X, T

    def error(self, X, W, T):
        """
        根据X,W,T三个矩阵根据公式计算误差
        param X: 二维array X
        param W: array W
        param T: array T
        return: 误差函数的计算结果
        """
        error = 1 / 2 * (X @ W - T).T @ (X @ W - T)  # error.shape = (1,1)
        return error[0][0]

    def E_RMS(self, error):
        """
        计算误差的方均根值
        return: 误差的的方均根值E_RMS
        """
        return np.sqrt(2 / self.size * error)

    def fit_without_regulation(self, X, T):
```

```
    """
    计算未加入正则项的数值解W
    param X: 二维array X
    param T: array T
    return: 模型参数W
    """
    return np.linalg.pinv(X) @ T

def fit_with_regulation(self, X, T, Lambda=1e-7):
    """
    计算加入正则项的数值解W
    param X: 二维array X
    param T: array T
    param Lambda:权重参数
    return: 模型参数W
    """
    return np.linalg.solve(X.T @ X + Lambda * np.identity(self.m + 1), X.T @ T)

def gradient_descent(self, X, T, rate=0.1, precision=1e-1, times=0,Lambda=0):
    """
    通过梯度下降法计算给定learning rate和精度要求所需要的迭代次数
    param X: 二维array X
    param T: array T
    param rate: learning rate
    param precision:精度要求，默认为0.1
    param times: 迭代次数
    param Lambda:惩罚项参数，默认为0，即为无正则项的梯度下降
    return: 计算出要求精度所需要的迭代次数和W
    """
    W = np.zeros((self.m + 1, 1))  # 初始化W为全为0的列向量
    # 计算给定迭代次数得到的W
    if times != 0:
        for i in range(times):
            last_error = self.error(X, W, T)
            W = W - rate * (X.T @ (X @ W - T) + Lambda * W)      #Lambda=0时为无正则项的梯度下降
            if self.error(X, W, T) > last_error:                #Lambda>0时为有正则项的梯度下降
                rate = rate / 2
        return W
    times = 0
    while self.error(X, W, T) > precision:
        last_error = self.error(X, W, T)
        W = W - rate * (X.T @ (X @ W - T) + Lambda * W)
        if self.error(X, W, T) > last_error:
            rate = rate / 2
        times += 1
    return times, W

def conjugate_gradient(self, X, T, precision=1e-5, times=0,Lambda=0):
    """
    通过共轭梯度法计算给定精度要求所需要的迭代次数，
    参考 http://en.wikipedia.org/wiki/Conjugate_gradient_method

    param X: 二维array X
    param T: array T
    param precision:精度要求，默认为0.1
    param times: 迭代次数
    param Lambda:惩罚项参数，默认为0，即为无正则项的共轭梯度
    return: 要求精度要求所需要的迭代次数和W
    """
```

```
        # 转化成Ax = b的形式求解
        A = X.T @ X + Lambda * np.identity(self.m+1)        #Lambda=0时为无正则项的共轭梯度
        x = np.zeros((self.m + 1, 1))                       #Lambda>0时为有正则项的共轭梯度
        b = X.T @ T

        r = b - np.dot(A, x)
        p = r
        rsold = r.T @ r
        # 计算给定迭代次数得到的W
        if times != 0:
            for i in range(times):
                Ap = A @ p
                alpha = rsold[0][0] / np.dot(p.T, Ap)[0][0]
                x = x + alpha * p
                r = r - alpha * Ap
                rsnew = r.T @ r
                p = r + (rsnew / rsold) * p
                rsold = rsnew
            return x
        times = 0
        while self.error(X, x, T) > precision:
            Ap = A @ p
            alpha = rsold[0][0] / np.dot(p.T, Ap)[0][0]
            x = x + alpha * p
            r = r - alpha * Ap
            rsnew = r.T @ r
            p = r + (rsnew / rsold) * p
            rsold = rsnew
            times += 1
        return times, x

    def poly(self, X, W):
        """
        通过自变量X和参数W生成相应的多项式函数，方便绘图比较
        param W:自变量值
        return:对应的多项函数值
        """
        result = 0
        for i in range(self.m + 1):
            result += W[i][0] * np.power(X, i)
        return result



# start test
size = 10  # 样本数据量
m = 5  # 拟合多项式函数的阶数
model = lab1(size, m)

# generate train data
x, y = model.generate_data()
# generate test data
x_test, y_test = model.generate_data()
# generate data of size 100
x_100, y_100 = lab1(size=100, m=5).generate_data()
# generate data of size 15
x_15, y_15 = lab1(size=15, m=5).generate_data()
```

```
# plot generate data and y = sin(2*pi*x) for comparison
plt.scatter(x, y)
x1 = np.linspace(0, 1, 1000)
plt.plot(x1, np.sin(2 * np.pi * x1))
plt.title('sample data')
plt.xlabel('x')
plt.ylabel('t')
plt.show()

# trans to generate X and T
X, T = model.trans(x, y)
# trans to generate X_Test and T_Test
X_Test, T_Test = model.trans(x_test, y_test)
# Analytical Solution
W_without_regulation = model.fit_without_regulation(X, T)
W_with_regulation = model.fit_with_regulation(X, T, 1e-7)

# 阶数m取不同值时，最佳拟合曲线情况
Train_E_RMS = []
Test_E_RMS = []
for m in range(size):
    train = lab1(size, m)
    train_X, train_T = train.trans(x, y)
    test_X, test_T = train.trans(x_test, y_test)
    fit_without_regulation = model.fit_without_regulation(train_X, train_T)
    if m == 0 or m == 1 or m == 3 or m == 9:
        plt.scatter(x, y)
        plt.plot(x1, np.sin(2 * np.pi * x1))
        plt.plot(x1, train.poly(x1, fit_without_regulation))
        plt.title(f'fit with m = {m}')
        plt.xlabel('x')
        plt.ylabel('t')
        plt.show()
    train_error = model.error(train_X, fit_without_regulation, train_T)
    test_error = model.error(test_X, fit_without_regulation, test_T)
    Train_E_RMS.append(model.E_RMS(train_error))
    Test_E_RMS.append(model.E_RMS(test_error))

# plot E_RMS with M in training data and test data
M = np.linspace(0, size - 1, size)
plt.plot(M, Train_E_RMS, label='Training')
plt.plot(M, Test_E_RMS, label='Test')
plt.xlabel('M')
plt.ylabel('$E_{RMS}$')
plt.legend()
plt.show()

# compare fit with different size(15 and 100)
train = lab1(size=100, m=9)
plt.scatter(x_100, y_100)
plt.plot(x1, np.sin(2 * np.pi * x1))
train_X, train_T = train.trans(x_100, y_100)
fit_without_regulation = model.fit_without_regulation(train_X, train_T)
plt.plot(x1, train.poly(x1, fit_without_regulation))
plt.title('N=100')
plt.xlabel('x')
plt.ylabel('t')
plt.show()
```

```
train = lab1(size=15, m=9)
plt.scatter(x_15, y_15)
plt.plot(x1, np.sin(2 * np.pi * x1))
train_X, train_T = train.trans(x_15, y_15)
fit_without_regulation = model.fit_without_regulation(train_X, train_T)
plt.plot(x1, train.poly(x1, fit_without_regulation))
plt.title('N=15')
plt.xlabel('x')
plt.ylabel('t')
plt.show()

# 分为训练数据和测试数据来对m=5,size=10时plotE_RMS随lambda的变化情况，结果发现lambda=10^-7时效果最好
Train_E_RMS = []
Test_E_RMS = []
M = []
for log10_lambda in range(-10, 0):
    M.append(log10_lambda)
    Lambda = 10 ** log10_lambda
    train_error = model.error(X, model.fit_with_regulation(X, T, Lambda), T)
    test_error = model.error(X_Test, model.fit_with_regulation(X_Test, T_Test, Lambda), T_Test)
    Train_E_RMS.append(model.E_RMS(train_error))
    Test_E_RMS.append(model.E_RMS(test_error))
plt.plot(M, Train_E_RMS, label='Training')
plt.plot(M, Test_E_RMS, label='Test')
plt.xlabel('$log_{10}\lambda$')
plt.ylabel('$E_{RMS}$')
plt.legend()
plt.show()

# size=10,m分别等于0，5，9时的曲线拟合图(有惩罚项与无惩罚项对比)
for size in [10]:
    for m in [0, 5, 9]:
        train = lab1(size, m)
        x, y = train.generate_data()
        train_X, train_T = train.trans(x, y)
        fit_without_regulation = train.fit_without_regulation(train_X, train_T)
        fit_with_regulation = train.fit_with_regulation(train_X, train_T)
        plt.scatter(x, y)
        plt.plot(x1, np.sin(2 * np.pi * x1))
        plt.plot(x1, train.poly(x1, fit_without_regulation))
        plt.title(f'fit without regulation size = {size} m = {m}')
        plt.xlabel('x')
        plt.ylabel('t')
        plt.show()
        plt.scatter(x, y)
        plt.plot(x1, np.sin(2 * np.pi * x1))
        plt.plot(x1, train.poly(x1, fit_with_regulation))
        plt.title(f'fit with regulation size = {size} m = {m}')
        plt.xlabel('x')
        plt.ylabel('t')
        plt.show()


# 使用梯度下降法和共轭梯度法达到相同精度所需次数的比较
times, W = model.gradient_descent(X, T, precision=1e-1)
print(f'使用梯度下降法，精度要求在1*10^-1所需次数:{times}')
times, W = model.gradient_descent(X, T, precision=1e-1,Lambda=1e-7)
print(f'使用共轭梯度法，精度要求在1*10^-1所需次数:{times}')
```

```
times, W = model.conjugate_gradient(X, T, precision=1e-5)
print(f'使用共轭梯度法，精度要求在1*10^-5所需次数:{times}')

# 相同迭代次数下，比较梯度下降法和共轭梯度法对曲线的拟合情况(设定迭代次数为100)
for size in [10, 30]:
    for m in [3, 9]:
        train = lab1(size, m)
        x, y = train.generate_data()
        train_X, train_T = train.trans(x, y)
        fit_gradient_descent = train.gradient_descent(train_X, train_T, times=100)
        fit_conjugate_gradient = train.conjugate_gradient(train_X, train_T, times=100)
        plt.scatter(x, y)
        plt.plot(x1, np.sin(2 * np.pi * x1))
        plt.plot(x1, train.poly(x1, fit_gradient_descent))
        plt.title(f'fit with gradient descent size = {size} m = {m}')
        plt.xlabel('x')
        plt.ylabel('t')
        plt.show()
        plt.scatter(x, y)
        plt.plot(x1, np.sin(2 * np.pi * x1))
        plt.plot(x1, train.poly(x1, fit_conjugate_gradient))
        plt.title(f'fit with conjugate gradient size = {size} m = {m}')
        plt.xlabel('x')
        plt.ylabel('t')
        plt.show()
```