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：1190201018
姓名： 李昆泽

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

OS：Win 10

Python 3.8

三、设计思想

1. 数据生成

利用函数$sin(2\pi x)$产生样本，x在[0,1]上均匀分布，在此基础上加入均值为 0，方差为 0.2 的高斯噪声。加入噪声的部分通过 python 中的 numpy 库进行实现，具体代码如下图所示。

```
def generate_data(N):
    x = np.linspace(0, 1, N)
    y = np.sin(2 * np.pi * x) + np.random.normal(loc=0, scale=0.1, size=N)
    return x.reshape(N, 1), y.reshape(N, 1)
```

图 1 数据生成

至于最后的 reshape 方法是为了保证返回向量行和列的维度和预期的一致，以避免一些潜在的问题。

2. 用高阶多项式函数拟合曲线（无正则项）

采用最小二乘法对多项式函数进行拟合，即建立误差函数来测量每个样本点真实值与预测值之间的误差，误差函数如下。

$$E_w = \frac{1}{2}(\mathbf{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

其中

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

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}$$

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

另外，m表示拟合所用多项式的次数，N表示样本数。

根据最小二乘法的原理，我们所需要做的就是求上述误差函数的最小值，我们对E_w关于$\boldsymbol{w}$求偏导，结果如下。

$$\frac{\partial E_w}{\partial \boldsymbol{w}} = \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$$

令$\frac{\partial E_w}{\partial \boldsymbol{w}} = 0$，可得

$$\boldsymbol{w}^* = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})^{-\mathbf{1}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$$

由上式可知，如果要计算$\boldsymbol{w}^*$，需要$\boldsymbol{X}$和$\boldsymbol{y}$，其中$\boldsymbol{y}$在数据的生成部分就已经计算出来了，所以下面只要计算$\boldsymbol{X}$即可。

我们分别构建下面两个向量：

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\boldsymbol{order} = [1 \quad 2 \quad \cdots \quad m]$$

利用 numpy 库的 broadcasting（广播）机制，$\boldsymbol{X} = \boldsymbol{x} ** \boldsymbol{order}$，当然这个式子只在 python 里成立，是利用广播机制的一种简化计算方法。具体代码如下图所示。

```
order = np.arange(M + 1).reshape((1, -1))
X = x ** order
```

图 2 **X** 计算

这里我们暂时不给出关于$\boldsymbol{w}^*$计算的 python 代码的实现，因为有无正则项的两种情况实际上是可以进行合并的，具体代码会在下一小节中给出。

3. 用高阶多项式函数拟合曲线（有正则项）

在无正则项的高阶多项式函数拟合中，我们发现拟合出的$\boldsymbol{w}^*$中的元素普遍有较大的绝对值，我们可以加入正则项来缓解这种过拟合。加入正则项之后的误差函数如下所示。

$$\widetilde{E_w} = \frac{1}{2}(\mathbf{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

对$\widetilde{E_w}$关于$\boldsymbol{w}$求偏导，结果为

$$\frac{\partial \widetilde{E_w}}{\partial \boldsymbol{w}} = \boldsymbol{X^T X w} - \boldsymbol{X^T y} + \lambda \boldsymbol{w}$$

令$\frac{\partial \widetilde{E_w}}{\partial \boldsymbol{w}} = 0$，可得

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

在上一小节中，我们已经给出了$\boldsymbol{X}$和$\boldsymbol{y}$的计算方法，这里给出计算$\boldsymbol{w}^*$的 python 代码。

```
W = np.dot(np.dot(np.linalg.inv(np.dot(X.T, X) +
                                lamda * np.identity(M + 1)), X.T), y)
```

图 3 $\boldsymbol{w}^*$计算

这里实际上是把有无正则项的情况结合起来了，如果 lamda 等于 0，则对应无正则项的情况；如果 lamda>0，则对应加入正则项的情况。

4. 梯度下降求解最优解

这里的误差函数实际上和最小二乘法时的类似，误差函数如下。

$$E_w = \frac{1}{2}(\mathbf{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

与最小二乘法不同是，这里我们的想法是计算E_w关于$\boldsymbol{w}$的梯度。我们知道顺着梯度的方向为增长最快的方向，那么梯度的反方向即为下降最快的方向。

我们通过迭代法求解最优的$\boldsymbol{w}$。首先初始化$\boldsymbol{w}$为一个全 1 的向量，计算E_w关于$\boldsymbol{w}$的梯度，在这里数值上等于E_w关于$\boldsymbol{w}$的偏导。

$$\frac{\partial E_w}{\partial \boldsymbol{w}} = \boldsymbol{X^T X w} - \boldsymbol{X^T y}$$

我们记E_w关于$\boldsymbol{w}$的偏导为 dw，设学习率为 alpha，则每次迭代为

W = W – alpha * dw，每经过一次迭代，E_w的值就会越小，也就逐步靠近最小值，当相邻两次迭代之后的E_w的差值小于δ（δ为一个较小的数）时，停止迭代，认为此时误差函数已经基本达到最小值。此时的$\boldsymbol{w}$即为我们最终要求的参数。具体的代码实现如下。

```
def gradient_descent(M, x, y, lr=0.05, delta=1e-6):
    iter = 0
    W = np.ones(M+1).reshape(-1, 1)
    order = np.arange(M + 1).reshape((1, -1))
    X = x ** order
    last_loss = np.sum(0.5 * np.dot((y - np.dot(X, W)).T, y - np.dot(X, W)))
    loss_history.append(last_loss)
    while True:
        dw = np.dot(np.dot(X.T, X), W) - np.dot(X.T, y)
        W -= lr * dw
        loss = np.sum(0.5 * np.dot((y - np.dot(X, W)).T, y - np.dot(X, W)))
        loss_history.append(loss)

        if np.abs(loss - last_loss) < delta:
            break
        else:
            iter += 1
            last_loss = loss

    X0 = x0.reshape(-1, 1) ** order
    return np.dot(X0, W), iter
```

图 4 梯度下降代码实现

在上述代码中我们用 loss_history 来记录每一次梯度下降后的误差值，loss 和 last_loss 分别保存当前和前一次的误差值，当它们的差值小于 delta 时，退出迭代。

5. 共轭梯度法求解最优解

共轭梯度法解决形如$\boldsymbol{Ax} = \boldsymbol{b}$的线性方程组解的问题 （$\boldsymbol{A}$必须是对称的、正定的）。共轭梯度法是一个典型的共轭方向法，它的每一个搜索方向是互相共轭的，而这些搜索方向仅仅是负梯度方向与上一次迭代的搜索方向的组合，因此，存储量少，计算方便。

对于第 k 步的残差$\boldsymbol{r_k} = \boldsymbol{b} - \boldsymbol{Ax_k}$，我们根据残差去构造下一步的搜索方向$\boldsymbol{p_k}$，初始时我们令$\boldsymbol{p_0} = \boldsymbol{r_0}$。然后利用 Gram-Schmidt 方法依次构造互相共轭的搜索方向$\boldsymbol{p_k}$，具体构造的时候需要先得到第 k+1 步的残差，即$\boldsymbol{r_{k+1}} = \boldsymbol{r_k} - \alpha_k \boldsymbol{Ap_k}$，根据第 k+1 步的残差构造下一步的搜索方向$\boldsymbol{p_{k+1}} = \boldsymbol{r_{k+1}} + \beta_{k+1}\boldsymbol{p_k}$。其中

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

$$\beta_{k+1} = \frac{\boldsymbol{r_{k+1}^T r_{k+1}}}{\boldsymbol{r_k^T r_k}}$$

具体代码如下图所示。

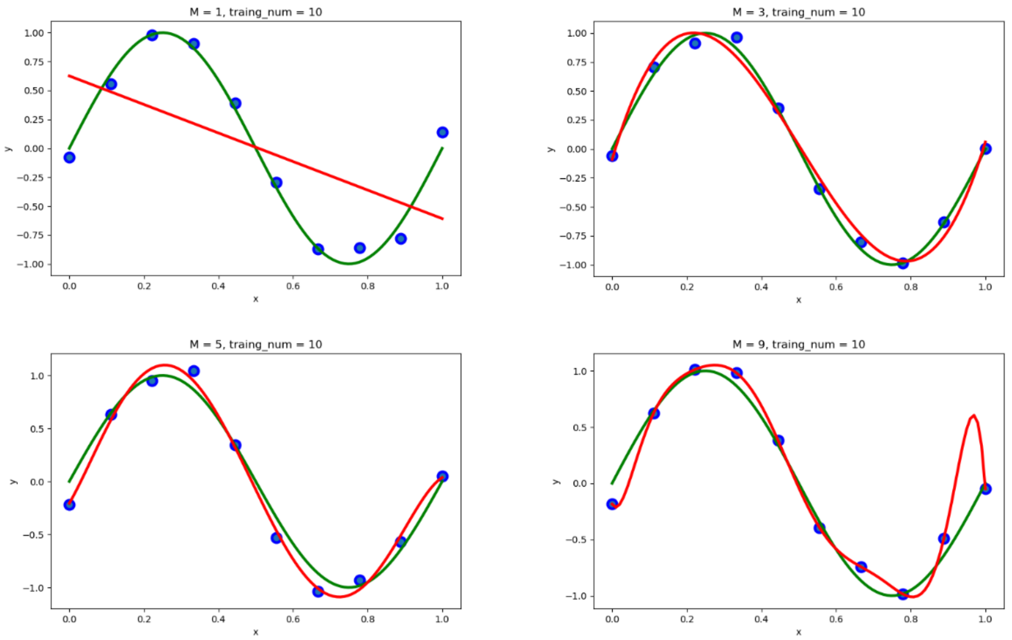
```python
def conjugate_gradient(M, x, y):
    iter = 0
    W = np.ones(M + 1).reshape(-1, 1)
    order = np.arange(M + 1).reshape((1, -1))
    X = x ** order

    A = np.dot(X.T, X)
    b = np.dot(X.T, y)
    r = b - np.dot(A, W)
    p = r
    while True:
        r1 = r
        alpha = np.dot(r.T, r) / np.dot(p.T, np.dot(A, p))
        W = W + alpha * p
        r = b - np.dot(A, W)
        q = np.linalg.norm(np.dot(A, W) - b) / np.linalg.norm(b)
        if q < 10 ** -6:
            break
        else:
            iter ++
            beta = np.linalg.norm(r) ** 2 / np.linalg.norm(r1) ** 2
            p = r + beta * p

    X0 = x0.reshape(-1, 1) ** order
    return np.dot(X0, W), iter
```

图 5 共轭梯度下降算法实现

四、实验结果与分析

1. 最小二乘法（无正则项）

固定样本大小为 10，分别使用不同的多项式阶数进行测试。

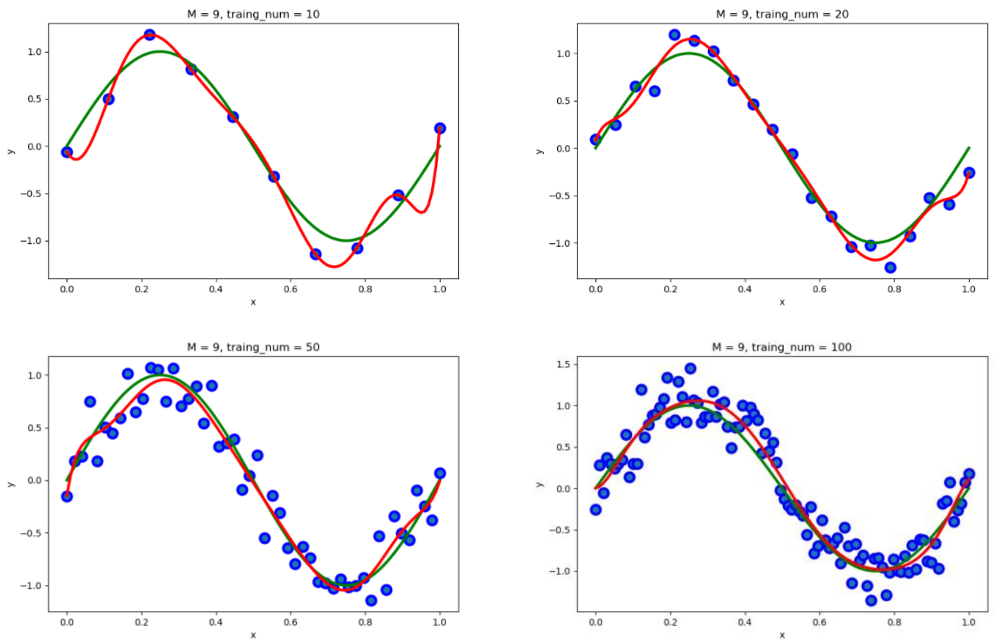


可以发现，在样本数量较少时，当 M=3 时拟合情况较好。当多项式次数较低时，无法拟合出所需要的曲线形状，属于“欠拟合”；而在多项式次数较高（例如 M=9）时，曲线虽然很好的拟合了所有的点，但实际上不能很好的拟合$sin(2\pi x)$这个函数，表现出一种“过拟

合”的情况。

在阶数过大的情况下，模型的复杂度和拟合的能力都增强，因此可以通过选择绝对值较大的系数来实现一种“震荡”，并以此来拟合所有的数据点。

下面，我们固定多项式次数为 9，改变样本数量，观察通过样本数量的改变能否缓解过拟合的情况。下面分别是样本数量为 10，20，50，100 时拟合曲线图。



可以发现，在固定多项式次数的情况下，随着样本数量的变化，“过拟合”的现象有所缓解。在样本数量为 10 和 20 时，过拟合现象还是比较明显的，但当样本数量达到 50 和 100 之后，过拟合现象得到明显好转。

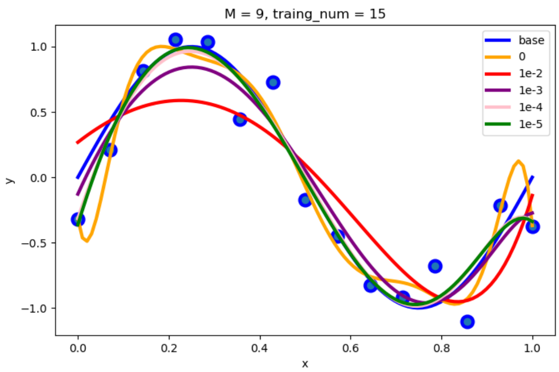
2. 最小二乘法（有正则项）

（1）参数λ的选择

加入正则项的目的是为了缓解过拟合，正则项的系数λ的大小实际上反映了缓解过拟合的程度。

当$\lambda = 0$时，没有任何缓解过拟合的效果，当λ过大时，可能会出现“欠拟合”的情况，下面我们通过实验寻找较优的λ值。

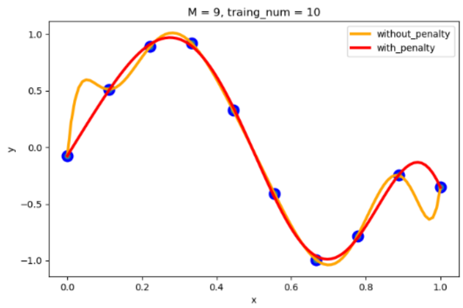
固定样本数和多项式次数，将不同的λ值代入，比较拟合效果。

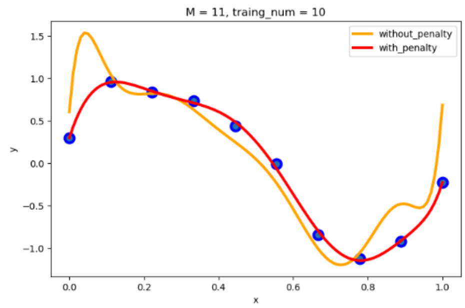


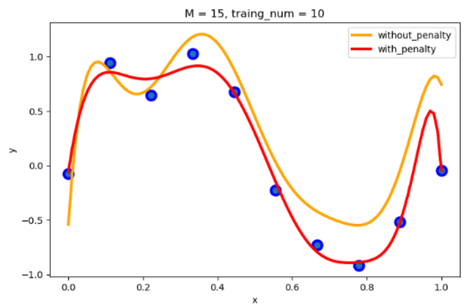
通过上图我们发现，当$\lambda \geq 10^{-3}$时，会出现欠拟合的情况，当$\lambda = 10^{-4}$或者10^{-5}时与原函数较为接近，拟合效果较好，这里取$\lambda = 10^{-4}$。

（2）比较拟合效果

我们首先固定样本个数，改变多项式次数，对比是否添加正则项的拟合效果。







我们可以发现，添加正则项可以有效缓解“过拟合”。但是，当样本容量较小时，如果选用的多项式次数过高，虽然添加正则项可以缓解过拟合，但是依然无法避免过拟合。

下面我们来分析一下为什么添加正则项可以缓解过拟合。将正则项加入损失函数的方法后，若参数中关于$\boldsymbol{w}$的正则项部分过大，那么损失函数的值就会因此变大。因此，为了取得损失函数的极小值，那么就需要取一个$\boldsymbol{w}$范数较小的$\boldsymbol{w}$值，即$\boldsymbol{w}$中元素的绝对值较小，而这正好可以有效抑制过拟合。

与此同时，我们也发现，如果单纯依靠添加正则项是不一定能消除过拟合的，当选用的多项式次数较高时，我们还需要适当的增加样

本数。

3. 梯度下降法

固定学习率为 0.01，停止精度为 1e-6，选用不同次数的多项式和不同的样本数进行对比实验，统计所需的迭代次数，具体如下表所示。

表 1 梯度下降法

多项式阶数	样本数	迭代次数
3	10	125868
3	20	85230
3	50	57386
5	10	31148
5	20	19646
5	50	177264
9	10	78853
9	20	66662
9	50	38532

总的来看，在固定多项式阶数时，随着样本数量的增多，梯度下降的迭代次数减少。从迭代次数的绝对数值上来看，梯度下降的迭代次数普遍在 10000 以上，迭代次数还是相当多的。

4. 共轭梯度法

固定停止精度为 1e-6，选用不同次数的多项式和不同的样本数进行对比实验，统计所需的迭代次数，具体如下表所示。

表 2 共轭梯度法

多项式阶数	样本数	迭代次数
3	10	4
3	20	4
3	50	4
5	10	8
5	20	8
5	50	8
9	10	18
9	20	17

9	50	19

从上表可以看出，随着多项式次数的增加，共轭梯度法的迭代次数也随之增加。当固定多项式次数时，改变样本数对迭代次数的影响不大。除此之外，相比于梯度下降法，共轭梯度法的迭代次数最大不超过 20，远远小于梯度下降法的迭代次数，求解的速度也更快。

5. 四种拟合方法的对比

我们固定样本数和多项式阶数，分别采用四种方法对数据进行拟合，并给出最后的$\boldsymbol{w}^*$。

表 3 多项式次数为 3，样本数为 10

w	最小二乘法（无正则项）	最小二乘法（有正则项）	梯度下降法	共轭梯度法
w_0	0.19982977	0.24015884	0.33441568	0.19982977
w_1	7.72376688	7.12090602	5.7308859	7.72376688
w_2	-25.8624544	-24.33924798	-20.84516583	-25.8624544
w_3	17.94348251	16.95404396	14.69100364	17.94348251

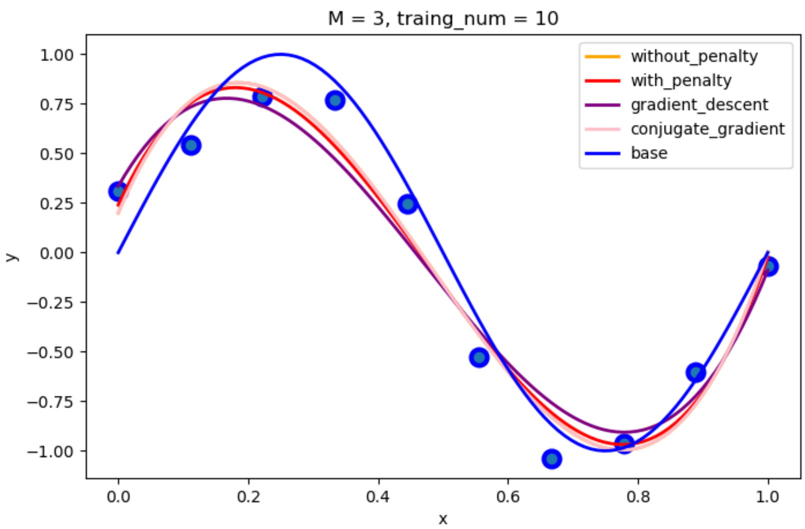
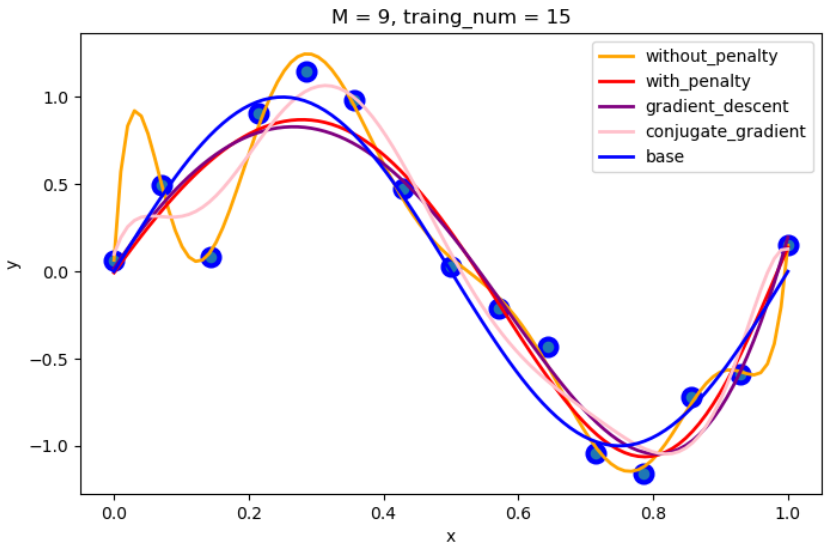


表 4 多项式次数为 9，样本数为 15

w	最小二乘法（无正则项）	最小二乘法（有正则项）	梯度下降法	共轭梯度法
w_0	6.50488008e-02	-6.97143189e-03	0.01759942	9.94244679e-02
w_1	6.51289792e+01	5.40192503e+00	5.81186938	1.13842167e+01

w_2	-1.62455037e+03	-4.50711089e+00	-9.03178866	-2.11876374e+02
w_3	1.55356601e+04	-1.32766248e+01	-5.54132768	1.70448238e+03
w_4	-7.43829033e+04	-1.38306327e+00	0.45789601	-5.97234179e+03
w_5	2.01527406e+05	7.64265988e+00	3.93767527	9.46859020e+03
w_6	-3.23989812e+05	9.06199382e+00	4.53417226	-4.61581692e+03
w_7	3.06489316e+05	5.27315596e+00	3.03039094	-4.95291407e+03
w_8	-1.57610141e+05	-8.12348443e-01	0.23468795	6.90822229e+03
w_9	3.39899808e+04	-7.25643033e+00	-3.2548999	-2.33970062e+03



从上面的两个对比实验可以看出，一般情况下，最小二乘法（无正则项）和共轭梯度法更容易出现过拟合的情况，w里元素的绝对值相对较大。而最小二乘法（有正则项）和梯度下降法不太容易出现过拟合，而且拟合效果较好。

五、结论

（1）在解析解中加入正则项可以有效缓解过拟合；
（2）增加样本数量可以有效缓解过拟合；
（3）对于梯度下降法和共轭梯度法而言，梯度下降法拟合速度较慢，迭代次数较多，但是不容易出现过拟合；共轭梯度法迭代次数较少，但可能会出现过拟合。

六、参考文献

[1] Pattern Recognition and Machine Learning
[2] Sum John and Leung ChiSing. Regularization Effect of Random Node Fault/Noise on Gradient Descent Learning Algorithm.[J]. IEEE transactions on neural networks and learning systems, 2021, PP
[3] 高前明.一种充分下降的共轭梯度法及其收敛性[J].淮阴师范学院学报(自然科学版),2021,20(03):212-216+234.

七、附录：源代码（带注释）

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

loss_history = []

# 数据点个数
N = 10

# 多项式阶数
M = 3

# 标准曲线
x0 = np.linspace(0, 1, 100)
y0 = np.sin(2 * np.pi * x0)


# 采样函数
def generate_data(N):
    x = np.linspace(0, 1, N)
    y = np.sin(2 * np.pi * x) + np.random.normal(loc=0, scale=0.2, size=N)   # 增加高斯噪声
    return x.reshape(N, 1), y.reshape(N, 1)
```

```
# 最小二乘法
def regress(M, x, y, lamda=0):
    # 计算 X
    order = np.arange(M + 1).reshape((1, -1))
    X = x ** order

    # 计算 W
    W = np.dot(np.dot(np.linalg.inv(np.dot(X.T, X) +
                                    lamda * np.identity(M + 1)), X.T), y)
    # loss = np.sum(0.5 * np.dot((y - np.dot(X, W)).T, y - np.dot(X, W)) + 0.5 * lamda * np.dot(W.T,
W))
    X0 = x0.reshape(-1, 1) ** order
    return np.dot(X0, W), W


# 梯度下降
def gradient_descent(M, x, y, lr=0.01, delta=1e-6):
    # 初始化参数
    iter = 0
    W = np.ones(M+1).reshape(-1, 1)
    order = np.arange(M + 1).reshape((1, -1))
    X = x ** order

    # 计算初始损失
    last_loss = np.sum(0.5 * np.dot((y - np.dot(X, W)).T, y - np.dot(X, W)))
    loss_history.append(last_loss)
    while True:
        # 计算梯度
        dW = np.dot(np.dot(X.T, X), W) - np.dot(X.T, y)

        # 梯度下降
        W -= lr * dW

        # 计算新的损失
        loss = np.sum(0.5 * np.dot((y - np.dot(X, W)).T, y - np.dot(X, W)))
        loss_history.append(loss)
```

```
        if np.abs(loss - last_loss) < delta:
            break
        else:
            # 更新迭代次数，更新损失
            iter += 1
            last_loss = loss

    X0 = x0.reshape(-1, 1) ** order
    return np.dot(X0, W), W, iter

# 共轭梯度下降法
def conjugate_gradient(M, x, y):
    # 初始化参数
    iter = 0
    W = np.ones(M + 1).reshape(-1, 1)
    order = np.arange(M + 1).reshape((1, -1))
    X = x ** order

    A = np.dot(X.T, X)
    b = np.dot(X.T, y)
    r = b - np.dot(A, W)
    p = r
    while True:
        # 迭代
        r1 = r
        alpha = np.dot(r.T, r) / np.dot(p.T, np.dot(A, p))
        W = W + alpha * p
        r = b - np.dot(A, W)

        # 计算误差
        q = np.linalg.norm(np.dot(A, W) - b) / np.linalg.norm(b)
        if q < 10 ** -6:
            break
        else:
            # 更新
            iter += 1
            beta = np.linalg.norm(r) ** 2 / np.linalg.norm(r1) ** 2
```

```python
                p = r + beta * p

    X0 = x0.reshape(-1, 1) ** order
    return np.dot(X0, W), W, iter

# 获取带噪声的数据
x, y = generate_data(N)

# 四种方式获取 y 和 W
y_1, W1 = regress(M, x, y)
y_2, W2 = regress(M, x, y, lamda=1e-4)
y_3, W3, iter = gradient_descent(M, x, y)
y_4, W4, iter = conjugate_gradient(M, x, y)

print(W1)
print(W2)
print(W3)
print(W4)

# 作图
plt.figure(1, figsize=(8, 5))
plt.plot(x0, y_1, 'orange', linewidth=2, label='without_penalty')
plt.plot(x0, y_2, 'r', linewidth=2, label='with_penalty')
plt.plot(x0, y_3, 'purple', linewidth=2, label='gradient_descent')
plt.plot(x0, y_4, 'pink', linewidth=2, label='conjugate_gradient')
plt.plot(x0, y0, 'b', linewidth=2, label='base')
plt.scatter(x, y, marker='o', edgecolors='b', s=100, linewidth=3)
plt.title(f'M = {M}, traing_num = {N}')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc="best", fontsize=10)
plt.show()
```