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： 考查课

实验题目： 多项式拟合正弦曲线

学号：

姓名：

一、实验目的

通过用不同方法拟合正弦曲线有噪声的训练点集，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)。实现从理论到实践的思维转换，深入领会机器学习种的拟合方法。

二、实验要求及实验环境

①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。

②实验环境：

Windows10; python 3.9.12; Visual Studio Code; Anaconda.

三、设计思想（本程序中的用到的主要算法及数据结构）

目标值 t=sin(x)是未知的,但是可以观测到 N 个有噪声的点（x_i,y_i）的观测值,噪声 x_i 在给定区间[-2Π，2Π]或[-Π，Π]服从随机变量的均匀分布，噪声 y_i 根据实际情况服从期望为 sin(x_i)，方差为 1/16 的高斯分布。并希望通过这些观测点的集合——即训练集来预测目标函数 t。

其中，将假设空间 H 设为：

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

y 是 x 的多项式函数，w 的线性函数。

在参数 w 的确定上，构造代价函数：

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

显然，这是一个最小二乘问题，可以通过对 E(w)求导，设导数=0，解得存在唯一解 w* .w*确定了假设空间 H，可以预测目标值 t.

但是当样本量较小或假设函数 y 阶数过高时，可能会出现过拟合现象，此时模型过于复杂，w*的范数较高，这时可以通过对代价函数加入惩罚项进行控制。

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\text{惩罚项或正则项}}$$

可以用以下 4 种方法求 w*:

①最小二乘法求解析解(无正则项):

$$
\begin{aligned}
\mathrm{E}(\mathbf{w}) &= \frac{1}{2}(\mathbf{X}\mathbf{w}-\mathbf{Y})^{\mathrm{T}}(\mathbf{X}\mathbf{w}-\mathbf{Y}) \\
&= \frac{1}{2}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}-\mathbf{Y})(\mathbf{X}\mathbf{w}-\mathbf{Y}) \\
&= \frac{1}{2}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w}-\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{Y}-\mathbf{Y}^{\mathrm{T}}\mathbf{X}\mathbf{w}+\mathbf{X}^{\mathrm{T}}\mathbf{Y}) \\
&= \frac{1}{2}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w}-2\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{Y}+\mathbf{X}^{\mathrm{T}}\mathbf{Y})
\end{aligned}
$$

令,

$$\frac{\partial \mathrm{E}}{\partial \mathbf{w}} = \mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w}-\mathbf{X}^{\mathrm{T}}\mathbf{Y} = 0$$

得,

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

```
#1.无正则项的最小二乘法拟合
def fit_GLS(trainset, m):
    X = np.array([trainset[:, 0] ** i for i in range(m + 1)]).T
    Y = trainset[:, 1]
    # 系数向量 W = (X^T * X)^(-1) * X^T * Y
    return np.dot(np.dot(np.linalg.inv(np.dot(X.T, X)), X.T), Y)
```

②最小二乘法求解析解(有正则项):

$$
\begin{aligned}
\widetilde{\mathrm{E}}(\mathbf{w}) &= \frac{1}{2}(\mathbf{X}\mathbf{w}-\mathbf{Y})^{\mathrm{T}}(\mathbf{X}\mathbf{w}-\mathbf{Y}) + \frac{\lambda}{2}||\mathbf{w}||_2 \\
&= \frac{1}{2}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}-\mathbf{Y})(\mathbf{X}\mathbf{w}-\mathbf{Y}) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \\
&= \frac{1}{2}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w}-\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{Y}-\mathbf{Y}^{\mathrm{T}}\mathbf{X}\mathbf{w}+\mathbf{X}^{\mathrm{T}}\mathbf{Y}) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \\
&= \frac{1}{2}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w}-2\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{Y}+\mathbf{X}^{\mathrm{T}}\mathbf{Y}) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}
\end{aligned}
$$

令,

$$\frac{\partial \widetilde{\mathrm{E}}}{\partial \mathbf{w}} = \mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w}-\mathbf{X}^{\mathrm{T}}\mathbf{Y}+\lambda\mathbf{w} = 0$$

得，

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

```
#2.有正则项的回归，其中 l 为正则项系数 lambda
def fit_ridge(trainset, m = 5, l = 0.5):
    X = np.array([trainset[:, 0] ** i for i in range(m + 1)]).T
    Y = trainset[:, 1]
    # 系数向量 W = (X^T * X + lambda * E)^(-1) * X^T * Y
    return np.dot(np.dot(np.linalg.inv(np.dot(X.T, X) + l * np.eye(m + 1)), X.T), Y)
```

③梯度下降法求优化解(无正则项)：

同①，对 E(w)求导可得：

$$\frac{\partial \mathrm{E}}{\partial \mathbf{w}} = \mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w} - \mathbf{X}^{\mathrm{T}}\mathbf{Y} = 0$$

随机初始化 w_0,设学习率为 lr(learning rate),对 w 梯度下降，直到向量的偏导数的 2-范数非常接近 0（小于给定的 e(误差)），迭代结束。并能输出满足误差要求时的实际迭代次数。

$$\mathbf{w} = \mathbf{w} - \alpha \cdot \frac{\partial \mathrm{E}}{\partial \mathbf{w}}$$

```
#3.梯度下降法，目标函数是凸函数可以固定学习率
def fit_GD(trainset, m = 3,  lr = 0.01, e = 1e-4):
    global iter_count
    # 随机在[0,1)内生成初始值
    w = np.random.rand(m + 1)

    N = len(trainset)
    X = np.array([trainset[:, 0] ** i for i in range(len(w))]).T
    Y = trainset[:, 1]

    while True:
        iter_count += 1
        #均方误差更容易收敛
        H_X = np.dot(X, w)
        grad = 2 * np.dot(X.T, H_X - Y) / N
        w -= lr * grad
        if np.linalg.norm(grad, ord=2) < e:
```

```
            return w
            break
```

④共轭梯度法求优化解(有正则项)：

由②可知，求 w 满足 $(X^TX+\lambda E)W=Y^TX.$ ，用共轭梯度法的求解步骤：

(0) 初始化$x_{(0)}$;

(1) 初始化$d_{(0)}=r_{(0)}=b-Ax_{(0)}$;

(2) 令

$$\alpha_{(i)}=\frac{r_{(i)}^Tr_{(i)}}{d_{(i)}^TAd_{(i)}};$$

(3) 迭代$x_{(i+1)}=x_{(i)}+\alpha_{(i)}d_{(i)}$;

(4) 令$r_{(i+1)}=r_{(i)}-\alpha_{(i)}Ad_{(i)}$;

(5) 令

$$\beta_{(i+1)}=\frac{r_{(i+1)}^Tr_{(i+1)}}{r_{(i)}^Tr_{(i)}},d_{(i+1)}=r_{(i+1)}+\beta_{(i+1)}d_{(i)}.$$

(6) 当$\frac{||r_{(i)}||}{||r_{(0)}||}<\epsilon$时，停止算法；否则继续从（2）开始迭代。$\epsilon$为预先设定好的很小的值，我这里取的是$10^{-8}$.

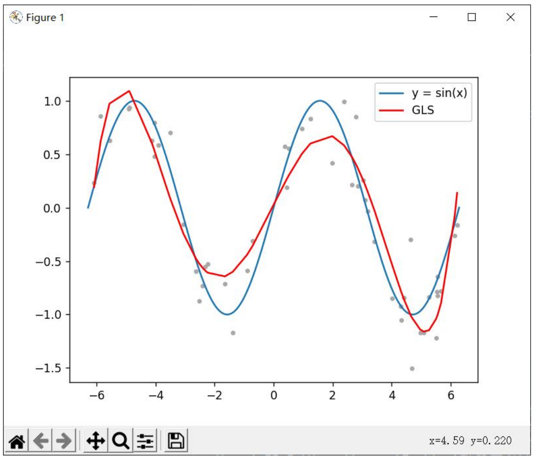
```
#4.共轭梯度法，其中 l 为正则项参数，e 为可以接受的误差 epsilon
def fit_CG(trainset, m = 5, l = 0, e = 1e-6):
    global iter_count
    X = np.array([trainset[:, 0] ** i for i in range(m + 1)]).T
    A = np.dot(X.T, X) + l * np.eye(m + 1)
    #用特征值均大于 0 断言正定
    assert np.all(np.linalg.eigvals(A) > 0), '系数矩阵非正定'
    b = np.dot(X.T, trainset[:, 1])
    w = np.random.rand(m + 1)

    # 初始化参数
    d = r = b - np.dot(A, w)
    r0 = r
    while True:
        iter_count += 1
        alpha = np.dot(r.T, r) / np.dot(np.dot(d, A), d)
        w += alpha * d
        new_r = r - alpha * np.dot(A, d)
        beta = np.dot(new_r.T, new_r) / np.dot(r.T, r)
        d = beta * d + new_r
        r = new_r
        if np.linalg.norm(r) / np.linalg.norm(r0) < e:
            break
```

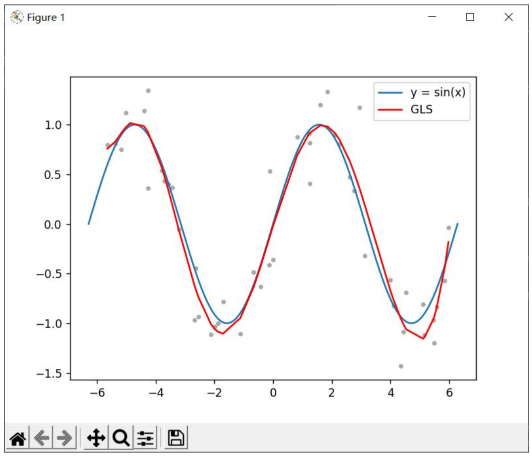
```
    return w
```

四、实验结果与分析

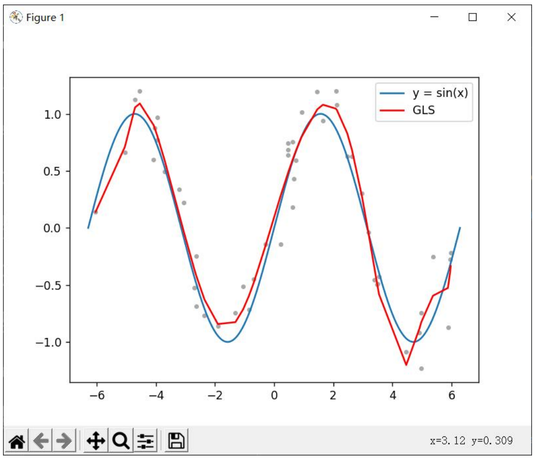
①最小二乘法求解析解(无正则项)：



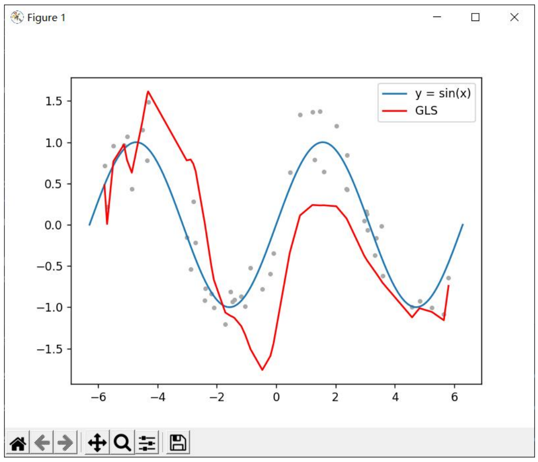
N = 50, m = 5



N = 50, m = 8

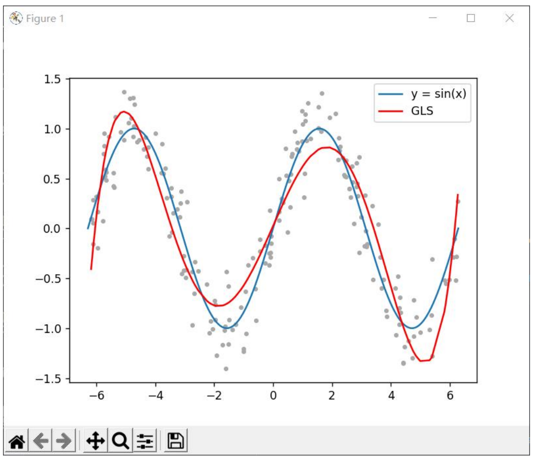


N = 50, m = 12

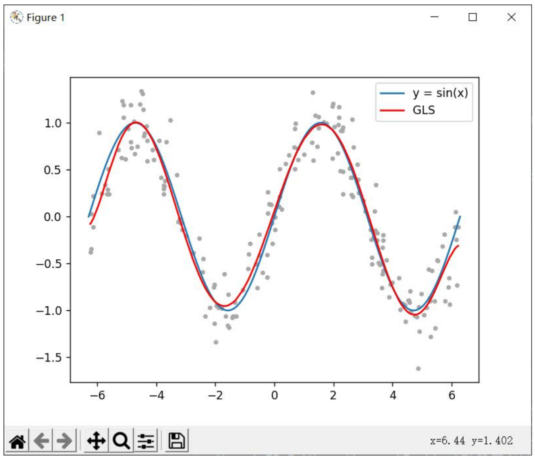


N = 50, m= 25

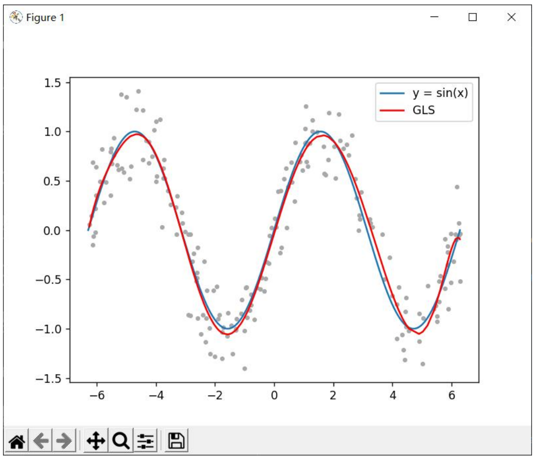
当训练集样本量 N=50 时，阶数随着 m=5、8、12 到 25 的过程中，从 m=5 的欠拟合，到 m=8 的拟合良好，到 m=12-25 过拟合程度逐渐加重。



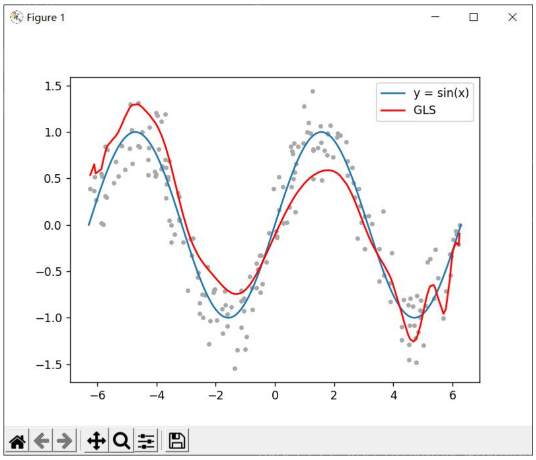
N=200, m = 5



N=200, m = 8



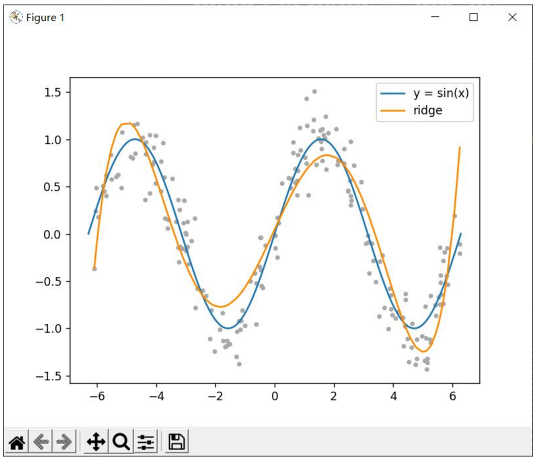
N=200, m = 12



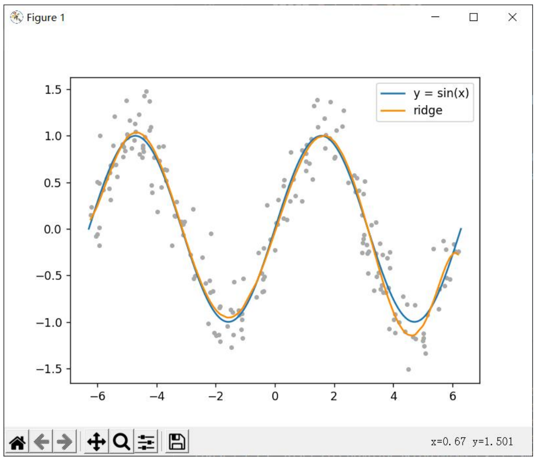
N=200, m = 25

当训练集样本量 N=200 时，阶数 m=5、8 的欠拟合程度降低，到 m=12 的拟合良好，到 m=25 过拟合程度严重。相比样本量 N=50,样本较少的时候，过拟合出现地阶数更大，总体拟合效果更好。

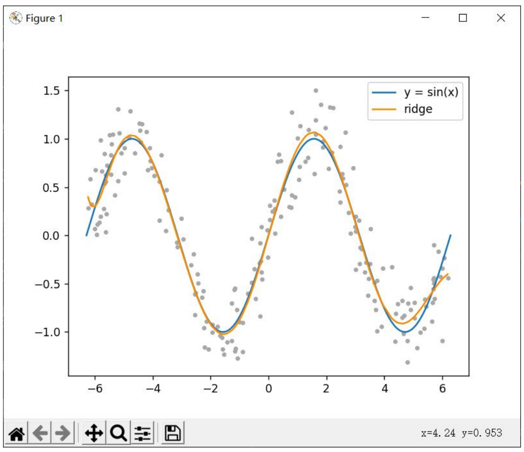
②最小二乘法求解析解(有正则项)：



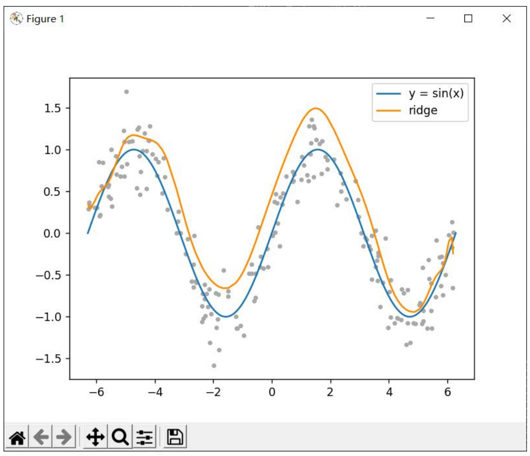
N=200, m=8, l=0.002



N=200, m=12, l=0.002

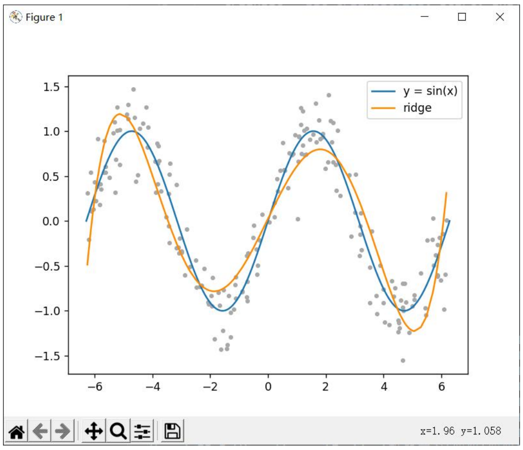


N=200, m=12, l=0.002

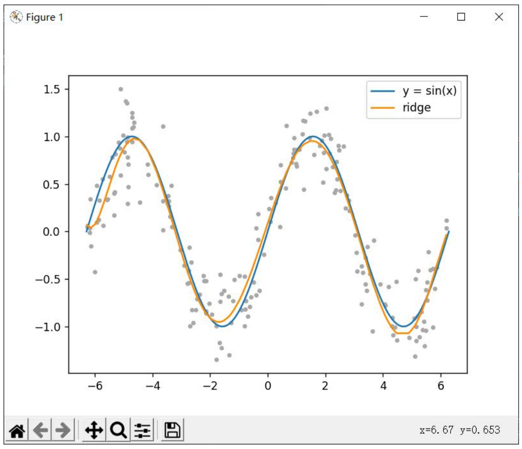


N=200, m=25, l=0.002

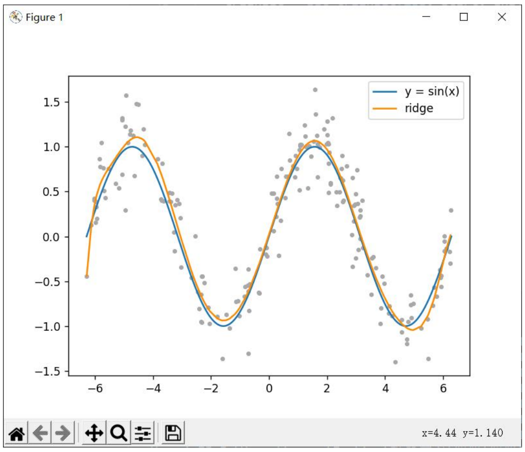
当在代价函数种加入正则项时，样本量同样为 N=200，正则项系数 l=0.002，可以看出在一定程度上改善了阶数 m 过大时的过拟合现象，使得拟合曲线整体更加平滑了。而低阶拟合曲线变化相对较小。



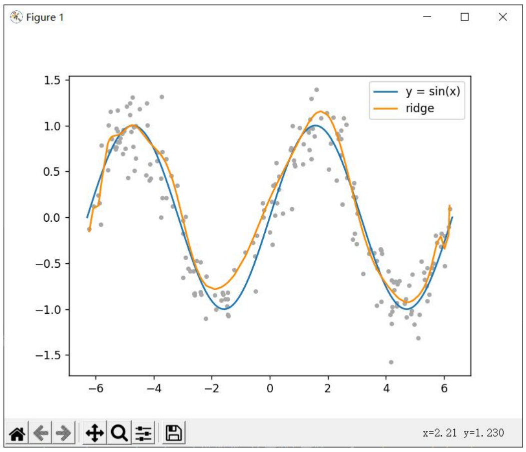
N=200, m=5, l=0.2



N=200, m=8, l=0.2

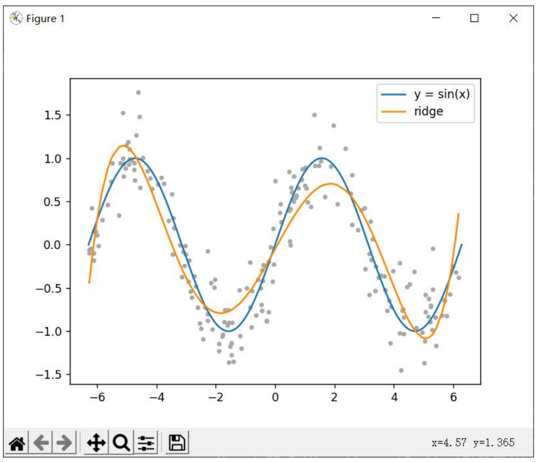


N=200, m=12, l=0.2

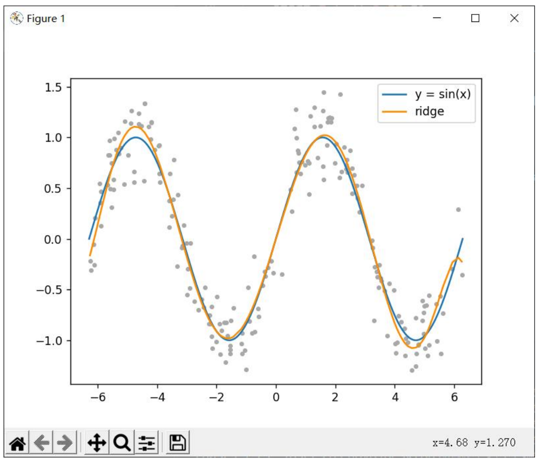


N=200, m=25, l=0.2

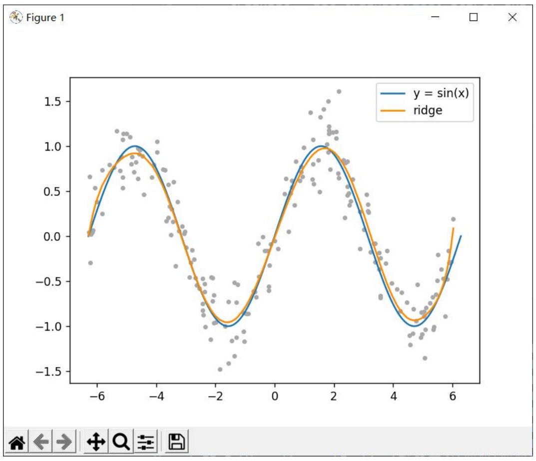
当正则项系数 l=0.2 时，可以看出相比 l = 0.002，高阶 m 过大时的过拟合现象得到了更大的改善，即便阶数很大也没有发生很严重的过拟合。低阶拟合变化不明显。



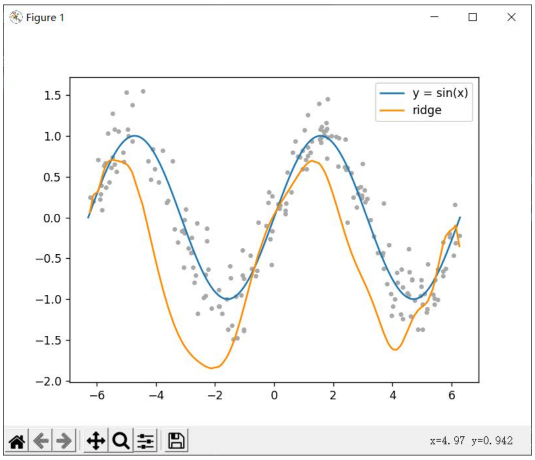
N=200, m=5, l=2



N=200, m=8, l=2



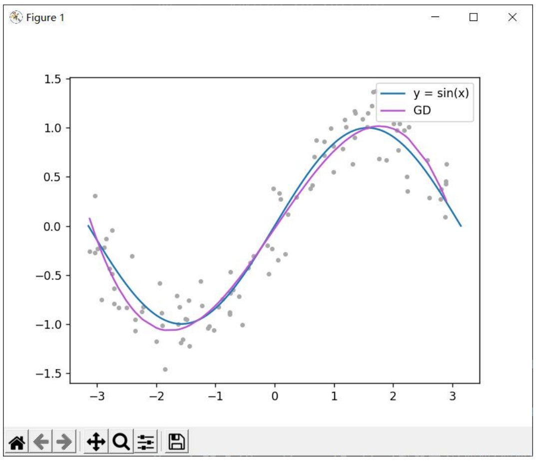
N=200, m=12, l=2



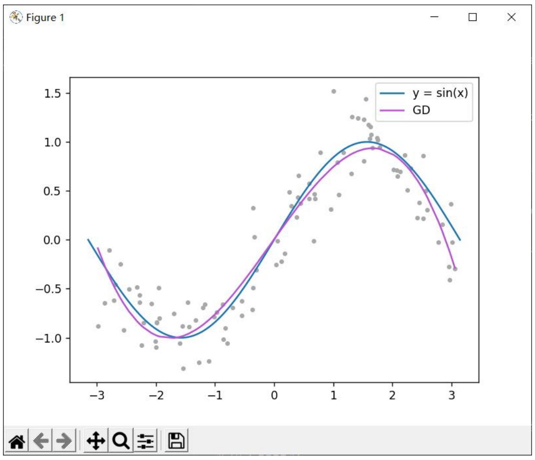
N=200, m=25, l=2

当正则项系数 l=2 时，可以看出相比 l = 0.002 和 l = 0.2，高阶 m 过大时拟合效果不好，因为正则项的比重过大，导致拟合函数更注重系数变小，相对降低了拟合训练集点的要求。低阶拟合变化不明显。

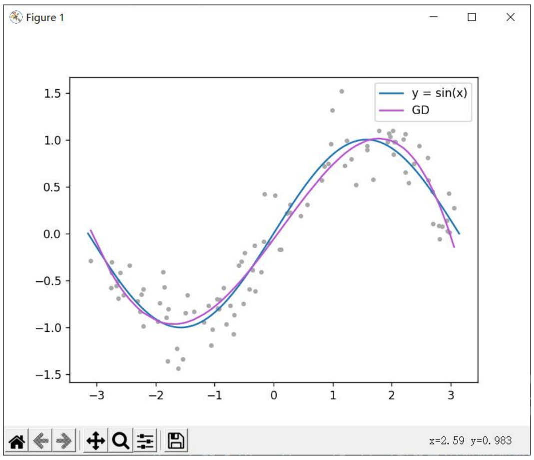
③梯度下降法求优化解(无正则项)：



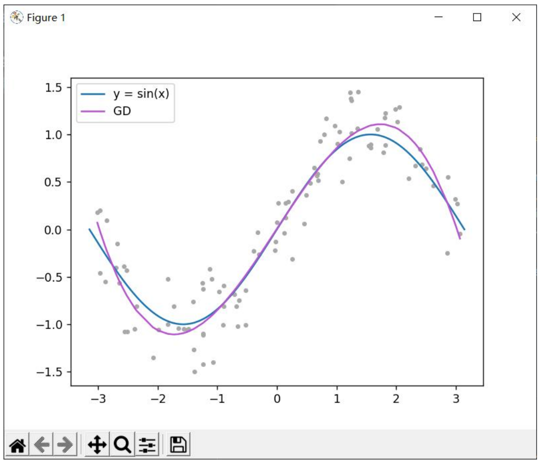
N=100, m=3, lr = 0.001, e = 0.0001，迭代 9358 次



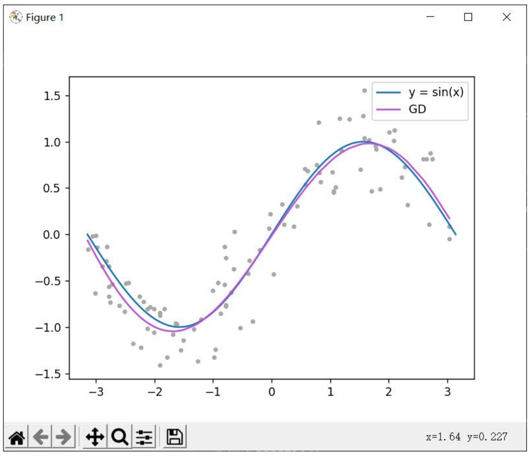
N=100, m=3, lr = 0.001, e = 0.0001，迭代 1000079 次

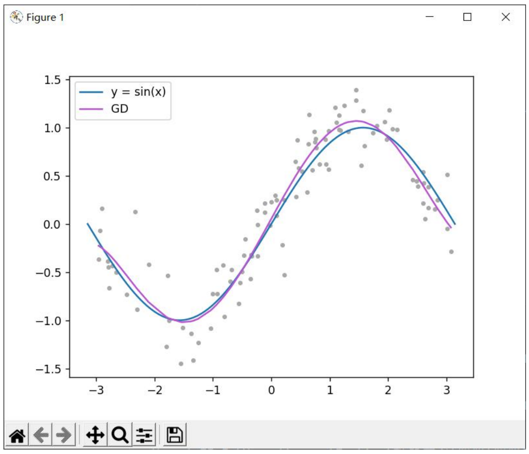


N=100, m=4, lr = 0.0001, e = 0.0001，迭代 173670 次



N=100, m=4, lr = 0.0001, e = 0.00001，迭代 1759113 次

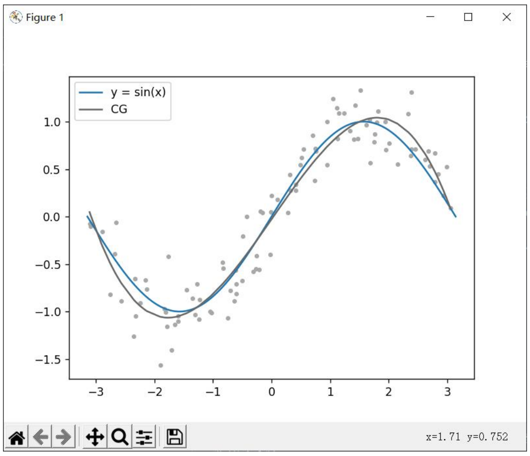


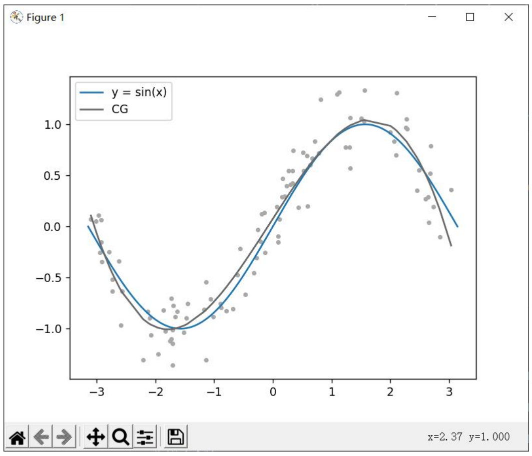


N=100, m=5, lr=0.0001, e = 0.0001,迭代 216189 次　　N=100, m=6, lr = 0.00001, e = 0.0001, 迭代 3841693 次

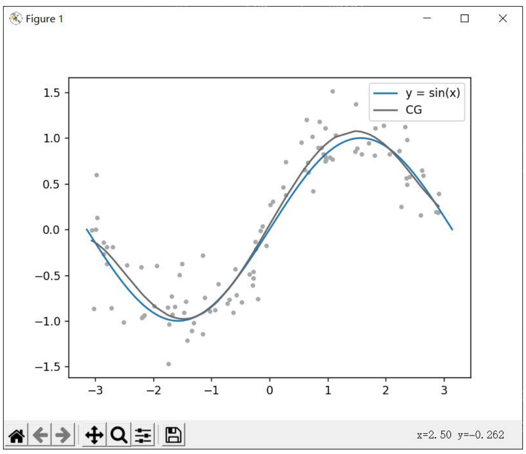
在梯度下降法实验中，设置学习率 lr 以 10 倍逐次递减，图中展示了 lr 使得梯度下降恰能收敛时的值，从中可以看出，阶数 m 越大，收敛所需要的 lr 越小。m=6 及以上时迭代次数已经过多，运算时间较长，故不进行阶数 m 更大的梯度下降。此外，为了尽快达到收敛，数据集的数据范围也不能太大(这里设置为(-Π，Π))。在 m<=6 时，各阶拟合效果都不错。但是随着学习率 lr 的减小和误差要求 e 的减小，迭代次数升高很多，运算速度比最小二乘法要慢得多。

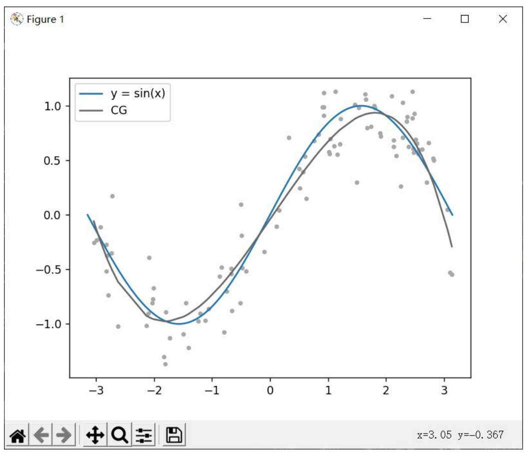
④共轭梯度法求优化解(有正则项)：



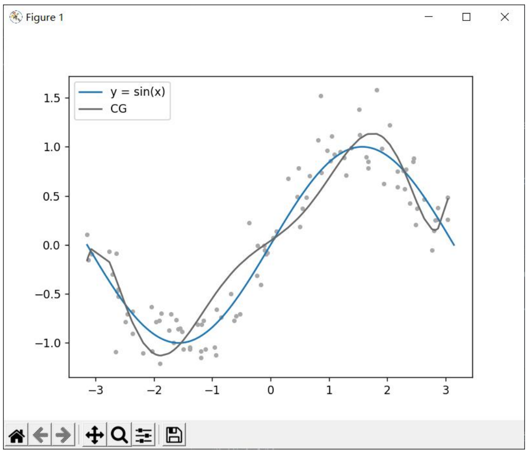


m = 3, l = 0.02, 迭代了 4 次　　m = 3, l = 2, 迭代了 4 次

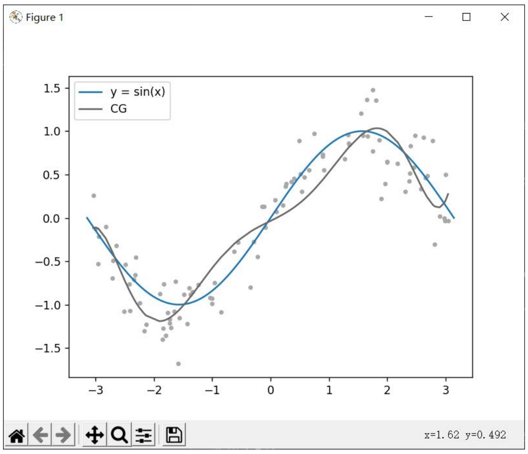




m = 5, l = 0.02, 迭代了 7 次　　m = 5, l = 2, 迭代了 8 次



$m = 7, l = 0.02$，迭代了 8 次



$m = 7, l = 2$，迭代了 8 次

共轭梯度法的迭代在理论上只要 m 步就能找到真解，实际计算中，考虑到舍入误差，一般迭代 3m 到 5m 步。可以看出共轭梯度法要比梯度下降法在各个阶数 m 的情况下，都要快得多。

在 $2<m<7$ 时，拟合效果不错，正则项系数 l 的影响也不大。当 $m>=7$ 时，出现了过拟合现象，正则项系数增大能稍稍减小过拟合现象。

五、结论

在对正弦函数的多项式拟合中，多项式的次数越高，其模型能力越强。当不加正则项时，高次多项式配合较少的数据集的拟合效果会出现过拟合。所以增大数据集可以有效地解决过拟合问题。

在目标函数中加入参数的惩罚项后，过拟合现象得到明显改善。这是由于参数增多并且平滑程度降低时，往往具有较大的范数，比如加入与系数相乘的 2-范数的正则项，可以有效地降低参数的绝对值和增加曲线的平滑程度，从而使模型复杂度与问题匹配。所以对于训练样本限制较多的问题，增加惩罚项是解决过拟合问题的有效手段。

在使用梯度下降时，由于我们的目标函数是二次的凸函数，只有一个最值点，所以梯度下降的初值可以随机生成。如果梯度下降学习率设置的比较大，那么下降结果将在目标函数最值附近逐渐向上跳

动，从而无法收敛。而能够收敛的学习率会随着模型阶数的升高而降低，迭代也会变慢。

梯度下降相比共轭梯度收敛速度很慢，迭代次数很大，而共轭梯度的稳定性较差，更容易出现过拟合现象，但对于数据量较大复杂度较高的情况，共轭梯度一般更优。

六、参考文献

[1] Shewchuk J R . An introduction to the conjugate gradient method without the agonizing pain[J]. Carnegie Mellon University, 1994.

[2] Snyman J A , Wilke D N . Practical Mathematical Optimization: Basic Optimization Theory and Gradient-Based Algorithms, Springer Optimization and Its Applications 133[M]. 2018.

[3] Guenin B , Knemann J , Tunel L . A Gentle Introduction to Optimization. 2018.

七、附录：源代码（带注释）

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

#准备：数据集及精确解
def get_trainset(N, start, end):
    # 噪点横坐标~U(strat, end)
    x = sorted(np.random.rand(N) * (end - start) + start)
    # 噪声~N(0,1/16)
    y = np.sin(x) + np.random.randn(N) / 4
    return np.array([x,y]).T

def draw_exact_solution(start, end):
    x = np.linspace(start, end, num = round(100*(end-start)))
    y = np.sin(x)
    plt.plot(x, y, label = 'y = sin(x)')
```

```
#1.无正则项的最小二乘法拟合
def fit_GLS(trainset, m):
    X = np.array([trainset[:, 0] ** i for i in range(m + 1)]).T
    Y = trainset[:, 1]
    # 系数向量 W = (X^T * X)^(-1) * X^T * Y
```

```
    return np.dot(np.dot(np.linalg.inv(np.dot(X.T, X)), X.T), Y)
```

```
#2.有正则项的回归，其中 l 为正则项系数 lambda
def fit_ridge(trainset, m = 5, l = 0.5):
    X = np.array([trainset[:, 0] ** i for i in range(m + 1)]).T
    Y = trainset[:, 1]
    # 系数向量 W = (X^T * X + lambda * E)^(-1) * X^T * Y
    return np.dot(np.dot(np.linalg.inv(np.dot(X.T, X) + l * np.eye(m + 1)), X.T), Y)
```

```
#3.梯度下降法，目标函数是凸函数可以固定学习率
def fit_GD(trainset, m = 3,  lr = 0.01, e = 1e-4):
    global iter_count
    # 随机在[0,1)内生成初始值
    w = np.random.rand(m + 1)

    N = len(trainset)
    X = np.array([trainset[:, 0] ** i for i in range(len(w))]).T
    Y = trainset[:, 1]
```

```
    while True:
        iter_count += 1
        #均方误差更容易收敛
        H_X = np.dot(X, w)
        grad = 2 * np.dot(X.T, H_X - Y) / N
        w -= lr * grad
        if np.linalg.norm(grad, ord=2) < e:
            return w
            break

#4.共轭梯度法，其中 l 为正则项参数
def fit_CG(trainset, m = 5, l = 0, e = 1e-6):
    global iter_count
    X = np.array([trainset[:, 0] ** i for i in range(m + 1)]).T
    A = np.dot(X.T, X) + l * np.eye(m + 1)
    #用特征值均大于 0 断言正定
    assert np.all(np.linalg.eigvals(A) > 0), '系数矩阵非正定'
    b = np.dot(X.T, trainset[:, 1])
    w = np.random.rand(m + 1)
```

```
    # 初始化参数
    d = r = b - np.dot(A, w)
    r0 = r
    while True:
        iter_count += 1
```

```
        alpha = np.dot(r.T, r) / np.dot(np.dot(d, A), d)
        w += alpha * d
        new_r = r - alpha * np.dot(A, d)
        beta = np.dot(new_r.T, new_r) / np.dot(r.T, r)
        d = beta * d + new_r
        r = new_r
        if np.linalg.norm(r) / np.linalg.norm(r0) < e:
            break
    return w
```

```
def draw(trainset, w, color, label):
    X = np.array([trainset[:, 0] ** i for i in range(len(w))]).T
    Y = np.dot(X, w)
    plt.plot(trainset[:, 0], Y, c = color, label = label)
```

```
if __name__ == '__main__':
    iter_count = 0
    #拟合在[-2Π, 2Π]上进行
    start = (-1) * np.pi
    end  = 1 * np.pi
    trainset = get_trainset(100, start, end)
    # 绘制训练集散点图
    for [x, y] in trainset:
        plt.scatter(x, y, color = 'darkgray', marker = '.')

    draw_exact_solution(start, end)
    w = fit_CG(trainset, m = 7, l = 0.02, e = 1e-6)
    #这个语句只用来判断梯度下降法或共轭梯度法的迭代速度
    print("迭代了{}次".format(iter_count))

    draw(trainset, w, color = 'dimgrey', label = 'CG')
    plt.legend()
    plt.show()
```