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： 多项式拟合正弦曲线

学号：1190200610
姓名： 张景阳

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

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。

三、设计思想（本程序中的用到的主要算法及数据结构）

梯度下降法，正规方程法，梯度下降与正规方程的正则化，共轭梯度法

1. 梯度下降法：

首先定义一个代价函数cost function 为

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^i)^2$$

m是数据集中数据点的个数，也就是样本数

1/2是一个常量，这样是为了在求梯度的时候，二次方乘下来的2就和这里的1/2抵消了，自然就没有多余的常数系数，方便后续的计算，同时对结果不会有影响
y 是数据集中每个点的真实y坐标的值，也就是类标签
h 是我们的预测函数（假设函数），根据每一个输入x，根据Θ 计算得到预测的y值

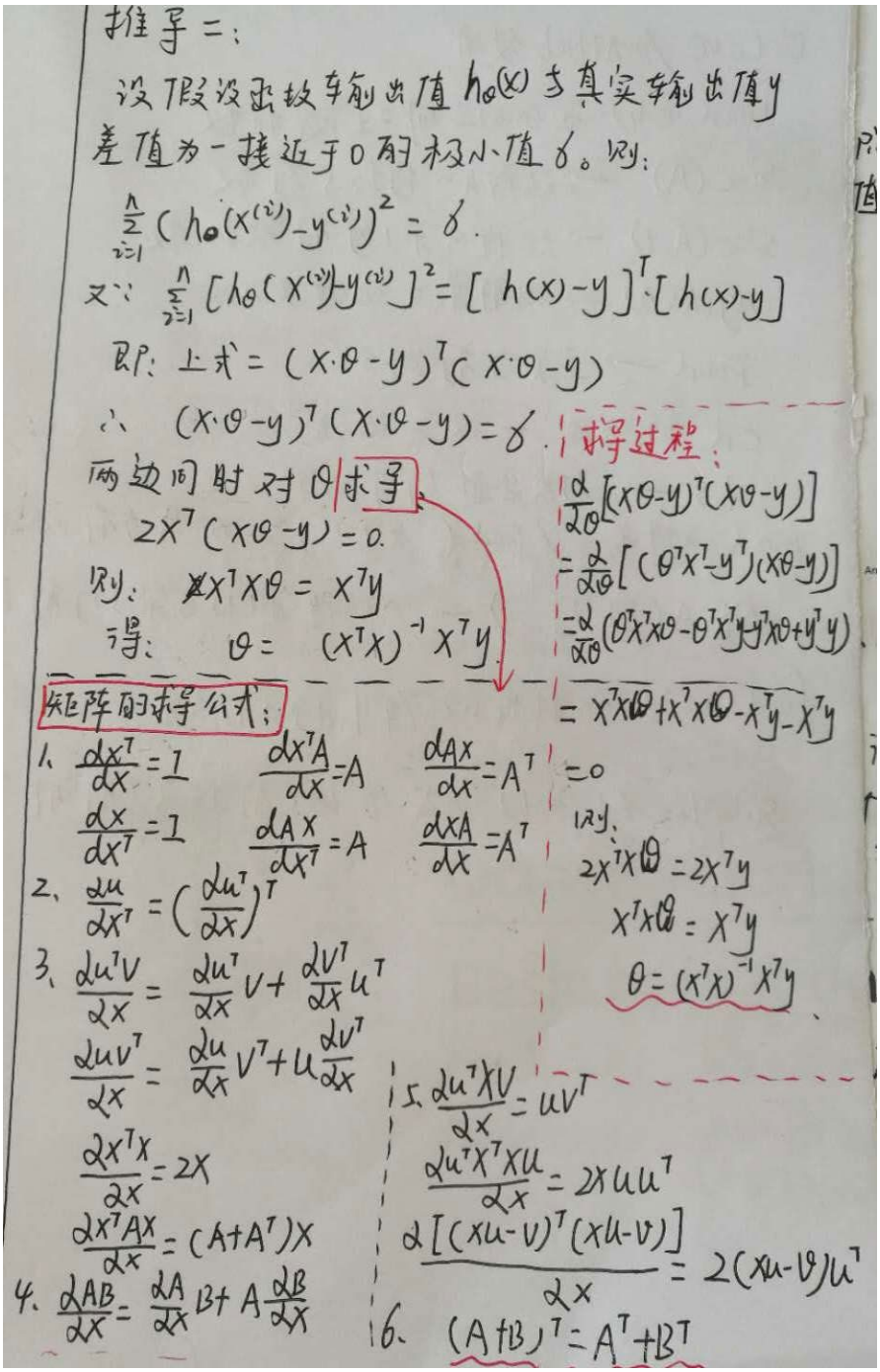
明确了代价函数和梯度，以及预测的函数形式。我们就可以开始编写代码了。但在这之前，需要说明一点，就是为了方便代码的编写，我们会将所有的公式都转换为矩阵的形式

我们的目的就是将代价函数最小化，同时在梯度下降的过程中，逐渐达到局部最优

2. 正规方程法
同梯度下降法，我们同样定义一个代价函数，并将其转换一下形式，通过求方程的解析解，将其最小化。
下面为推导过程:

$$J(\theta)=\frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

$$=\frac{1}{2}\left(X\theta-\vec{y}\right)^T\left(X\theta-\vec{y}\right)$$



可得出结果为：

$$\theta = \left(X^T X\right)^{-1} X^T \vec{y}$$

即可求出我们拟合的参数的解析解

3. 共轭梯度法

在共轭梯度法中，先令$\frac{\partial E}{\partial w} = X^T Xw - X^T Y + \lambda w$，$\frac{\partial E}{\partial w} = 0$

此时求$(X^T X + \lambda)w = X^T Y$的解析解，记$Q = X^T X + \lambda$，$b = X^T Y$

该方程的解转化为求$min_{w \in R^n} \frac{1}{2} w^T Q w - b^T w$

记负梯度$r = -\frac{\partial E}{\partial w}$，$p = r$，则共轭梯度的迭代如下：

```
while{
    a = r.T.dot(T) / (p.T.dot(Q).dot(p))
    r_prev = r
    w = w + a * p
    r = r - a * Q.dot(p)
    p = r + r.T.dot(r).dot(p) / (r_prev.T.dot(r_prev))
}
```

4. 自适应学习率算法

基于 Armijo 准则的线性回溯搜索算法：既然是计算学习率，我们就需要转换视角，将学习率 alpha 看作是未知量，因当前点为当前搜索方向??都是已知的，故有下面的关于 alpha 的函数:

$$h(\alpha) = f(x_k + \alpha * d_k)$$

梯度下降寻找 f(x)最小值，那么在??和??给定的前提下，即寻找函数的最小值，可以证明存在?使得 h(x)的导数为 0，则?就是要寻找的学习率。

1）二分线性搜索

二分线性搜索是最简单的处理方式，不断将区间[α1, α2]分成两半，选择端点异号的一侧，直到区间足够小或者找到当前最优学习率。

2）回溯线性搜索

基于 Armijo 准则计算搜素方向上的最大步长,其基本思想是沿着搜索方向移动一个较大的步长估计值，然后以迭代形式不断缩减步长，直到该步长使得函数值相对与当前函数值的减小程度大于预设的期望值(即满足 Armijo 准则)为止。

（1）二分线性搜索的目标是求得满足 $h'(?) \approx 0$ 的最优步长近似值，而回溯线性搜索放松了对步长的约束，只要步长能使函数值有足够大的变化即可。

（2）二分线性搜索可以减少下降次数，但在计算最优步长上花费了不少代价；回溯线性搜索找到一个差不多的步长即可。

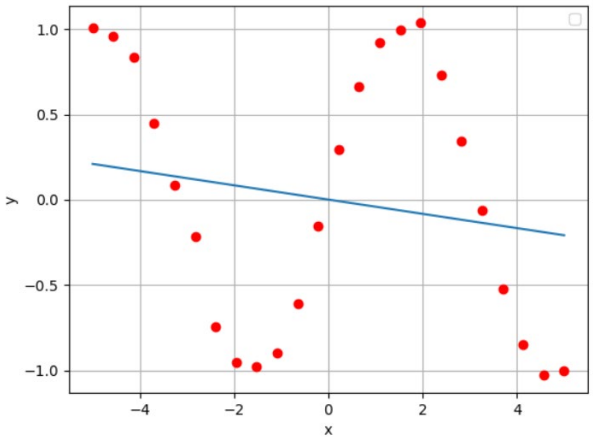
四、实验结果与分析

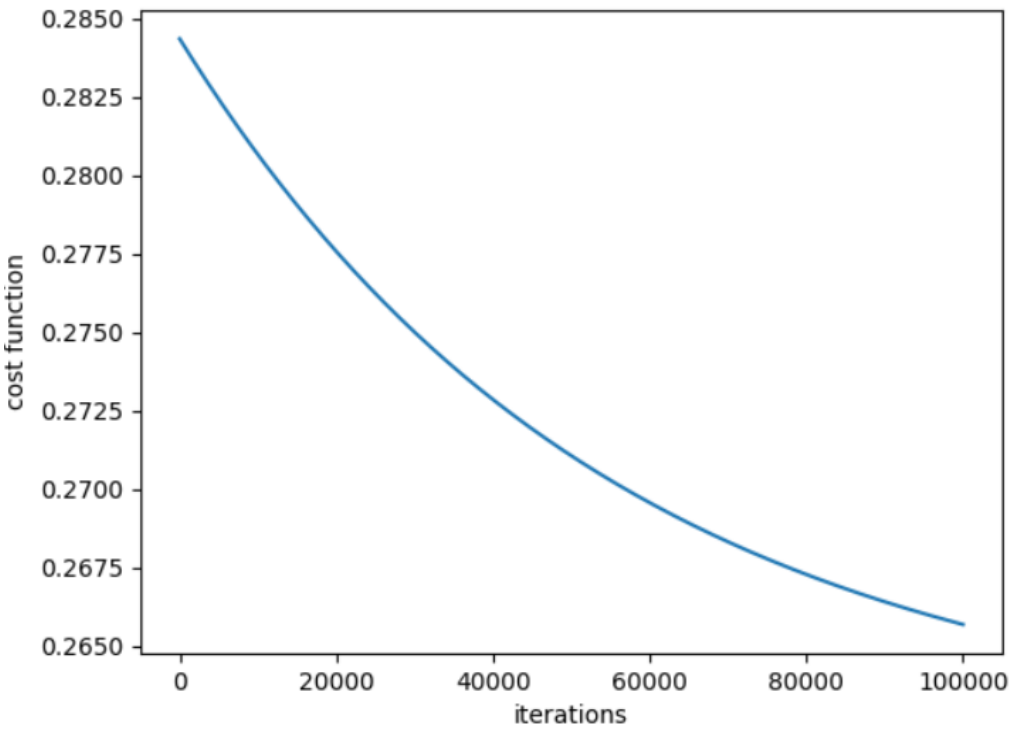
1. 不带有惩罚项的梯度下降法

(1) n = 24, max_exp = 1

拟合出的就是一条直线，拟合效果不好

下面是 100000 次的学习结果



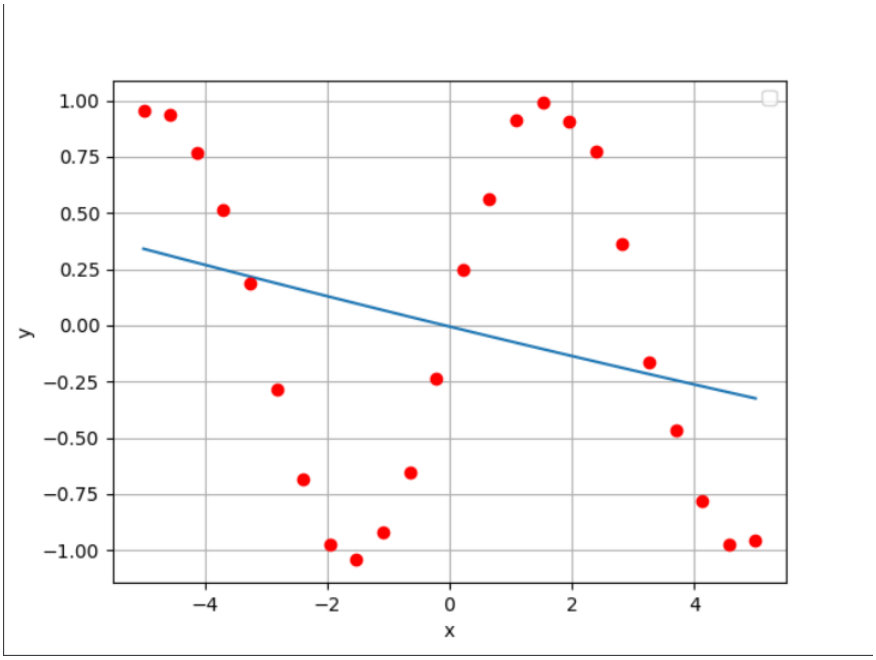


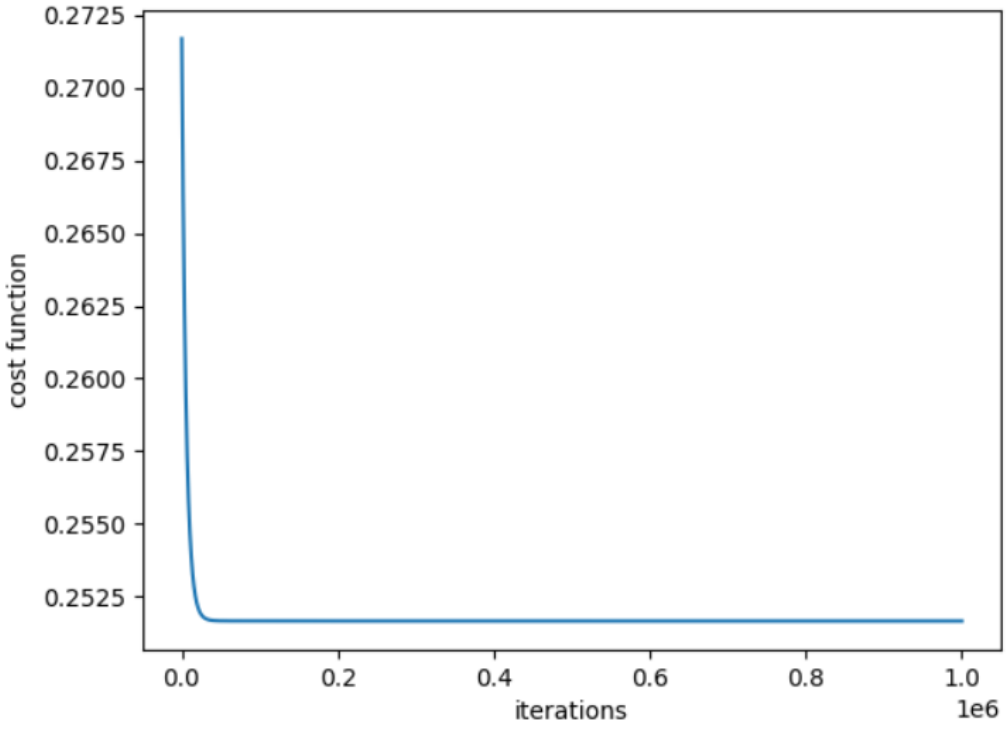
代价函数在 100000 次学习之后，大约在 0.265 左右，如果在学习下去，也是几

乎不变，因此我们需要提高我们假设函数的阶数

(2) n = 24, max_exp = 2

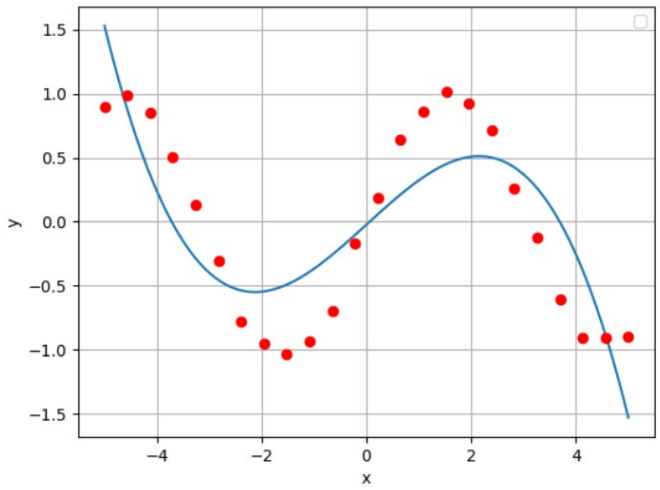
这次我们用二次函数来拟合，看卡效果如何，下面是学习 1000000 次后的结果

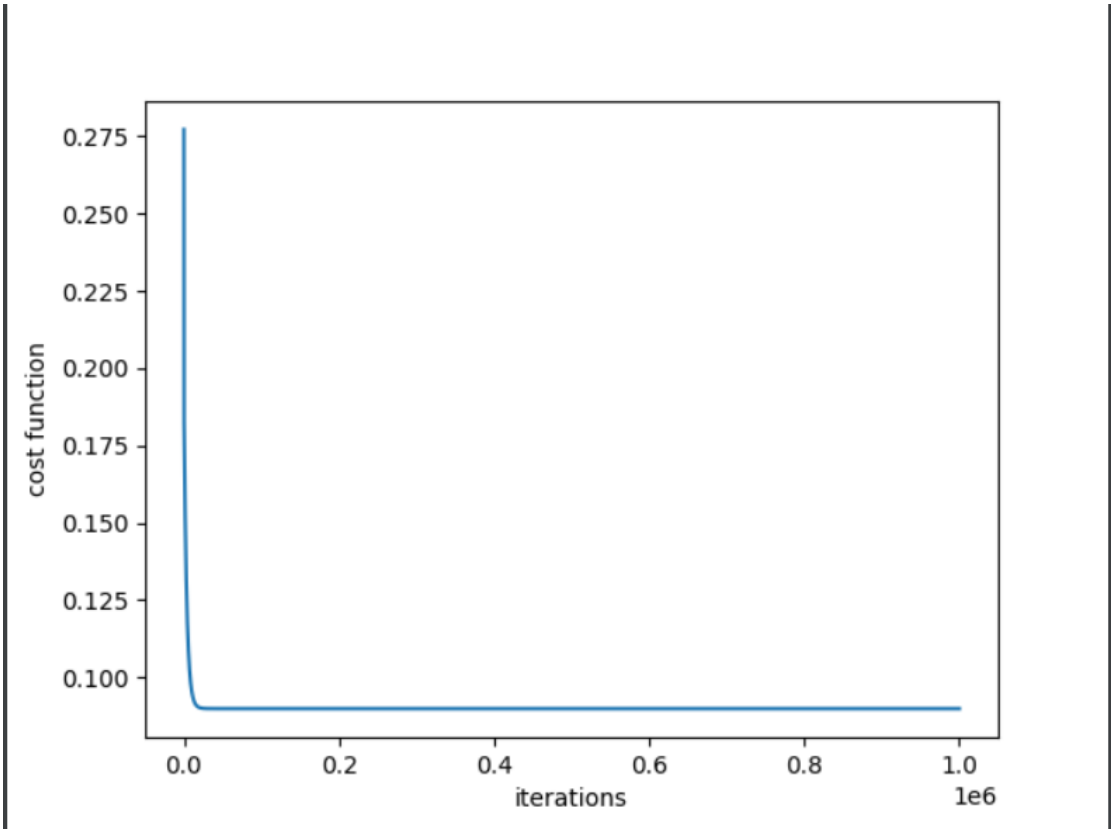




发现仍然效果不好，二次函数和线性的差不多，只是代价函数在迭代一定数量之后就是突然下滑，我们之后试一下 3 次怎么样

(3) n = 24, max_exp = 3
这次我们需要调节一下学习率 alpha，如果学习率过大的话，会导致代价函数无穷大的情况，因此我们这里的 alpha 为 1e-4，在学习 1000000 次的情况下，我们来看看效果

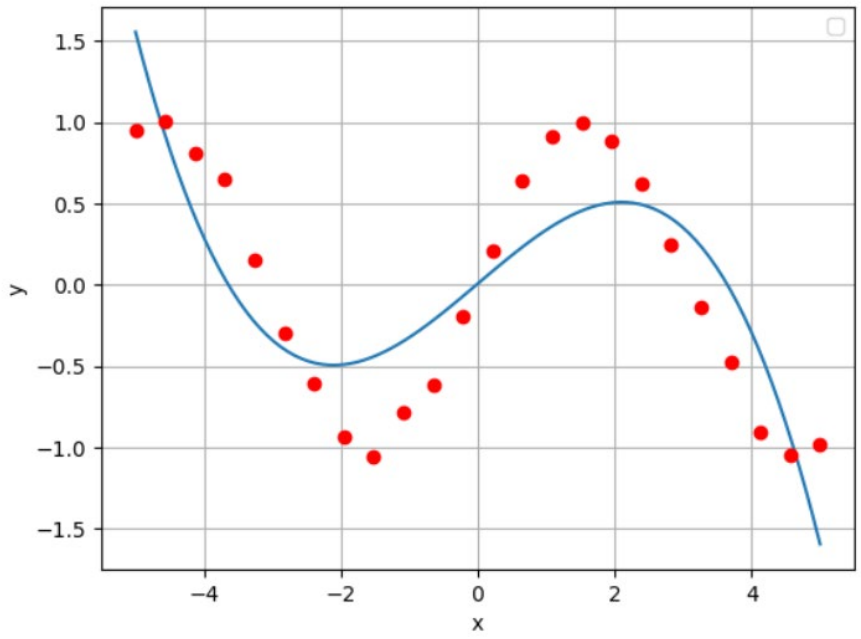


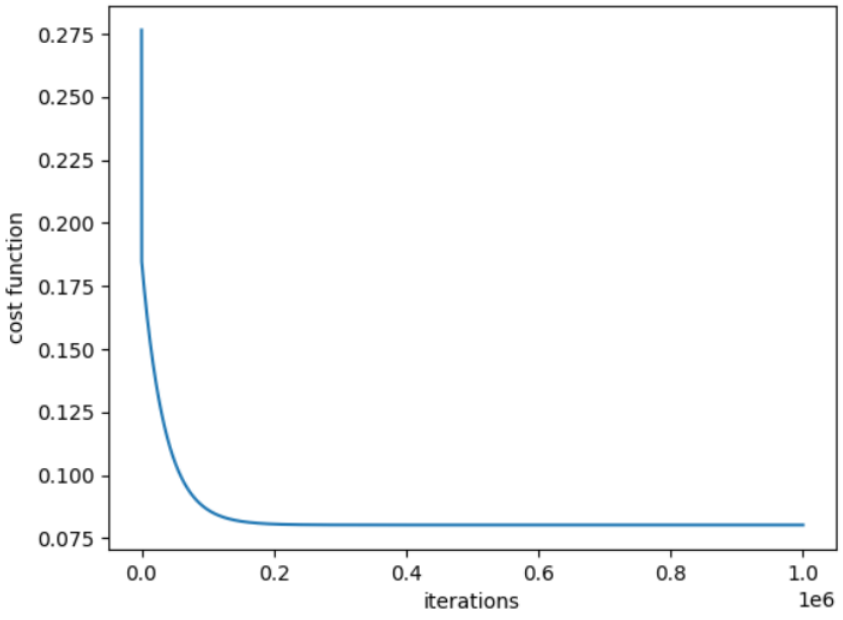


我们发现，这次的拟合效果还是可以，但是不够好，代价函数也从 0.3 降到

了 0.1，我们继续增加阶数，看看会发生什么

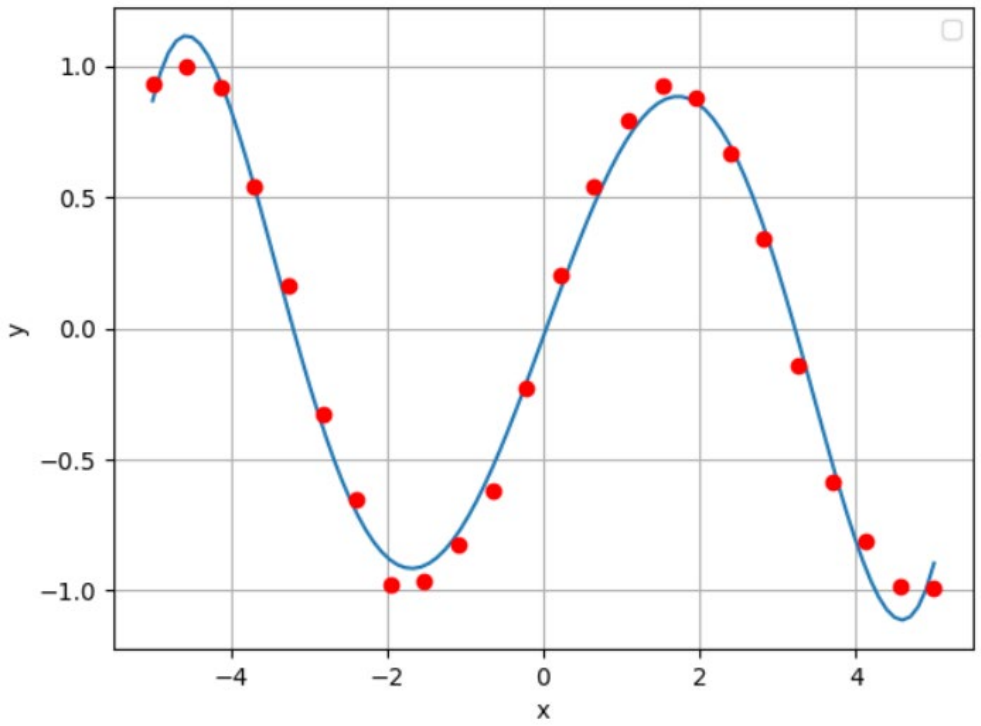
(4) n = 24, max_exp = 4
我们将 alpha 调为 1e-5，学习 1000000 次，观察实验结果

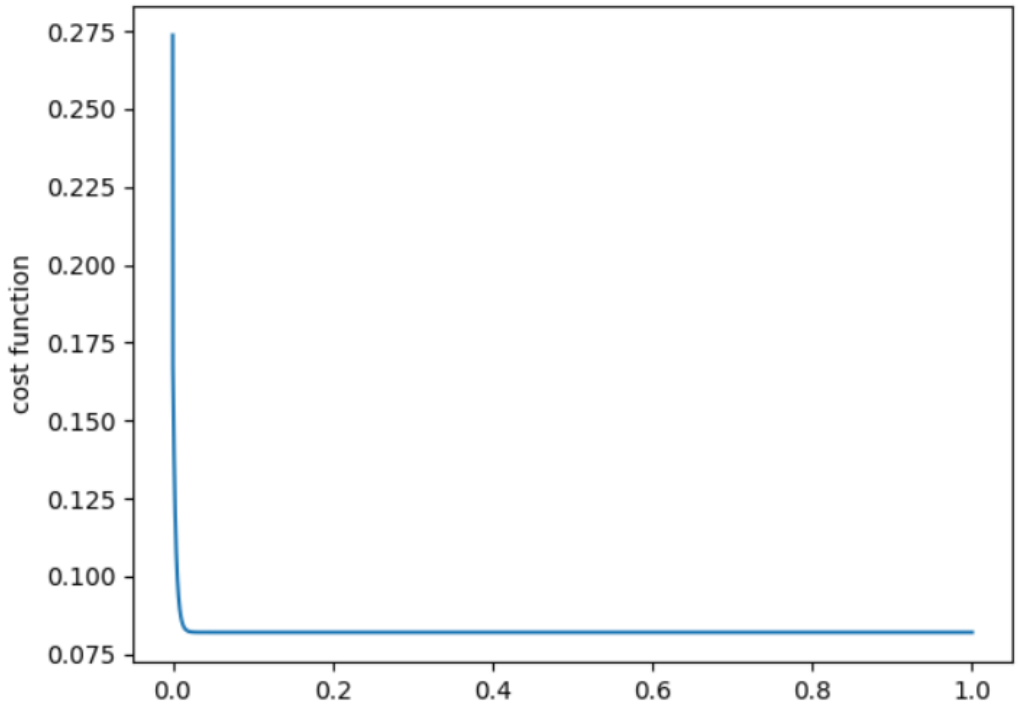




学习结果仍然差强人意，我们直接将阶数升到 6，观察现象

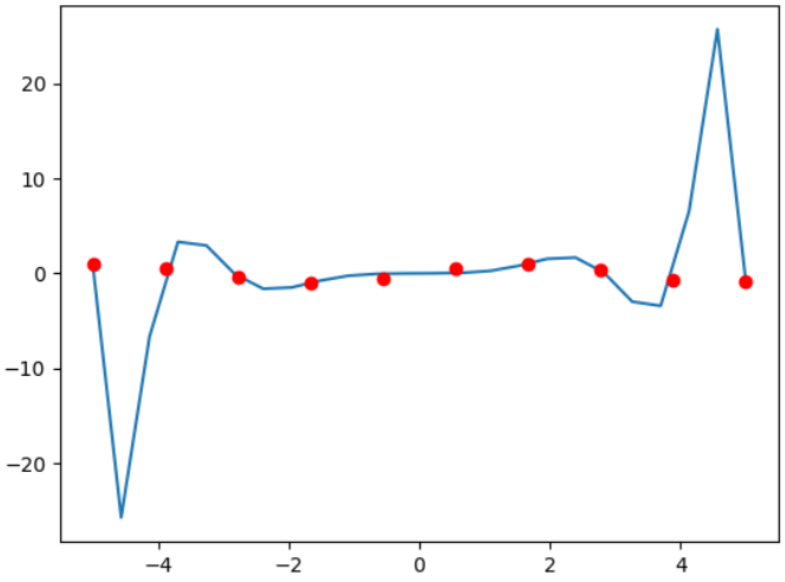
(4) n = 24, max_exp = 6

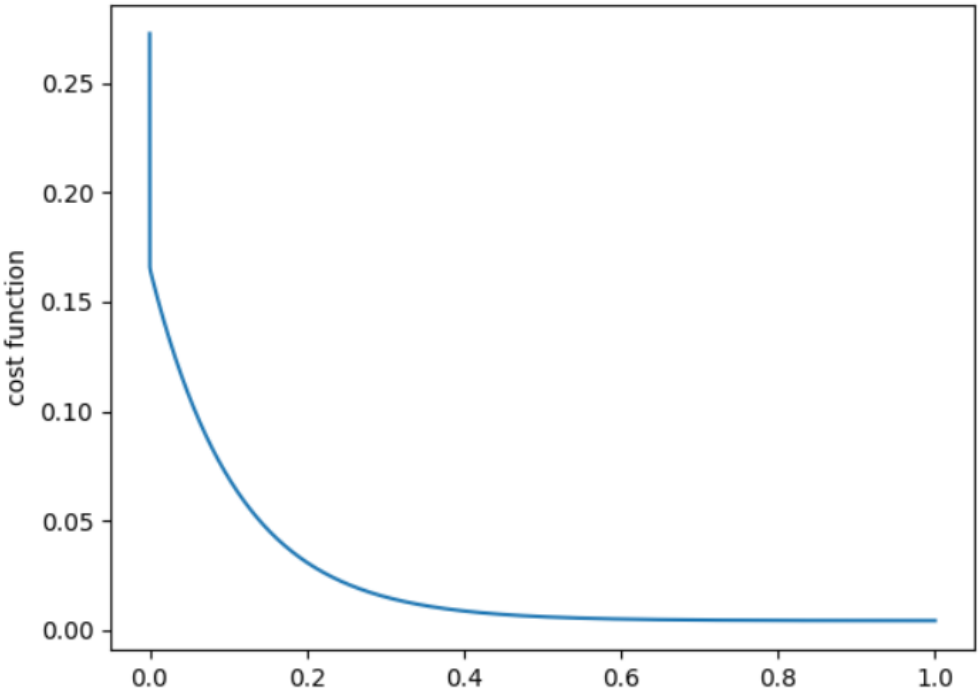

1.0
0.5
0.0
−0.5
−1.0
y
−4
−2
0
2
4
x


0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
cost function
0.0
0.2
0.4
0.6
0.8
1.0

这次拟合的相当成功，我们如果将项数调到 9 次方会发生什么呢？，我们继续实验

(6) n = 24, max_exp = 9



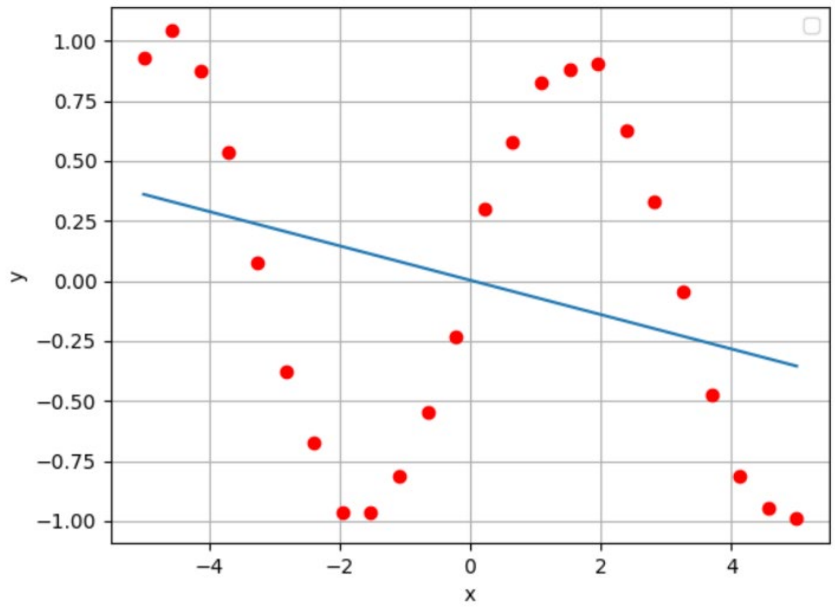


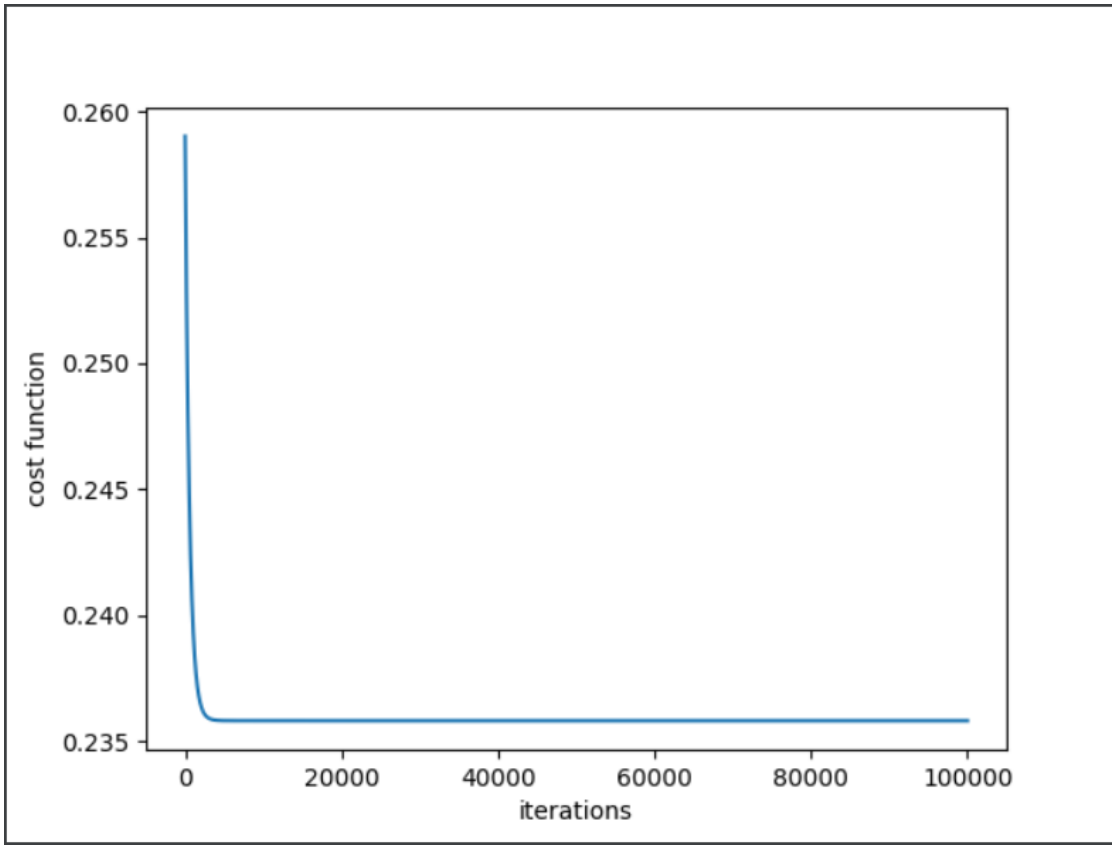
出现过拟合的情况，应该添加惩罚项来解决

2. 带有惩罚项的梯度下降法

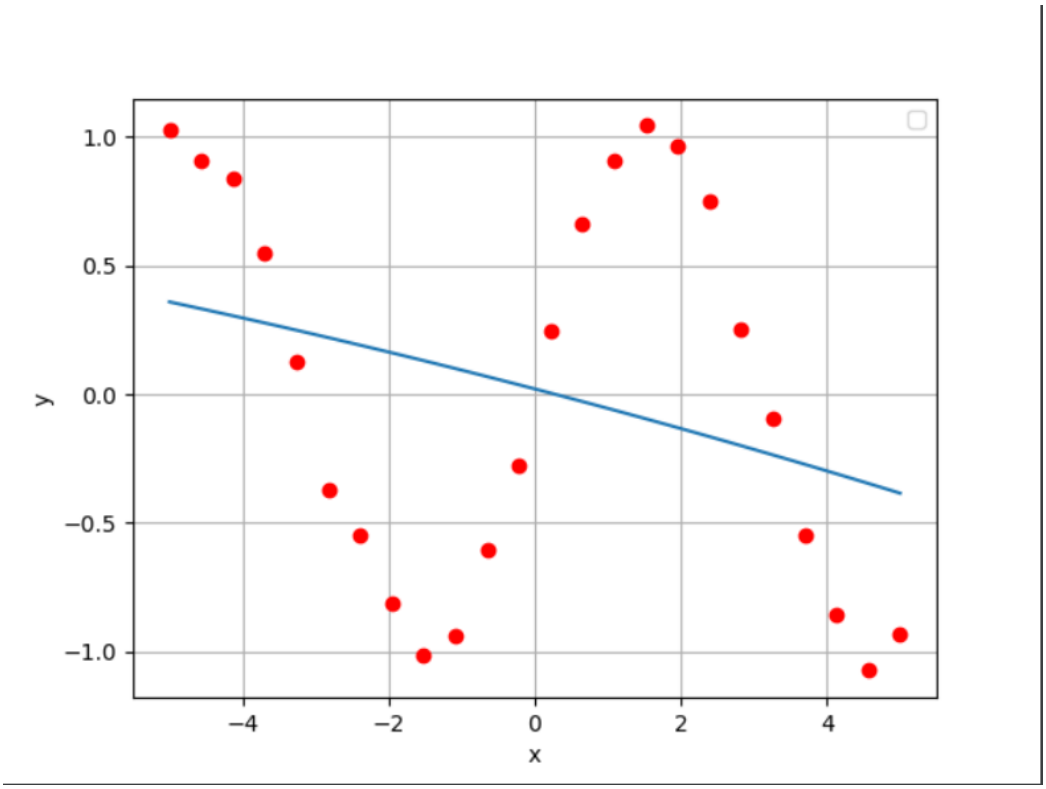
同上，也是做了 5 次实验

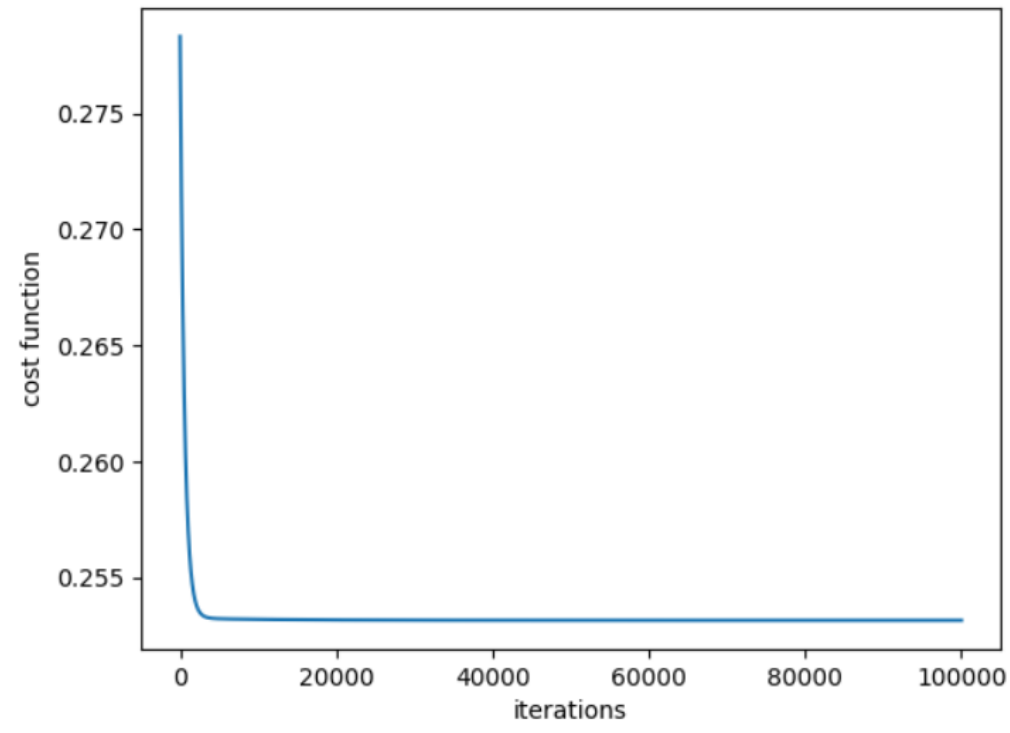
(1) n = 24, max_exp = 1 惩罚项=1e-2



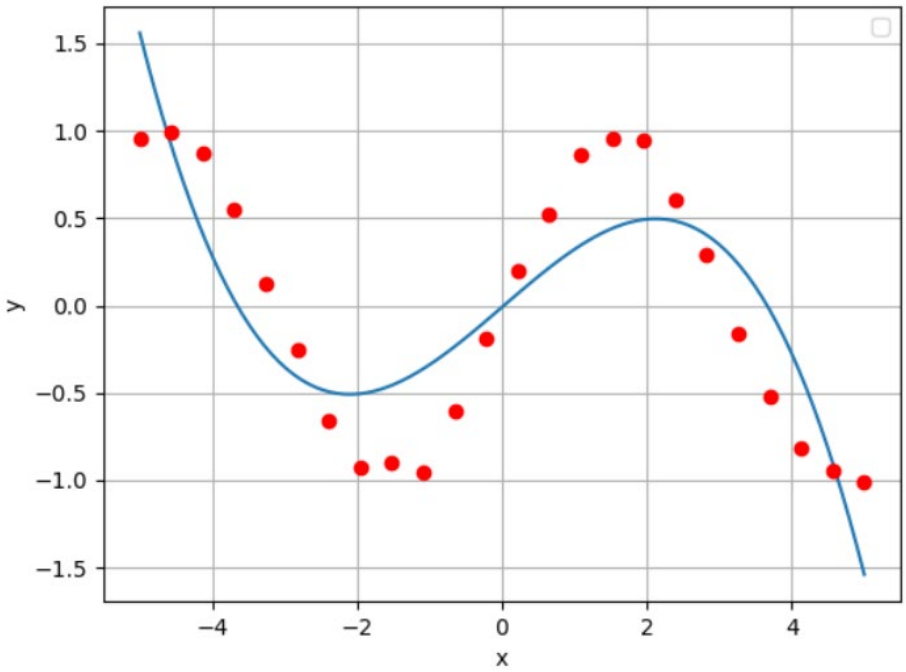


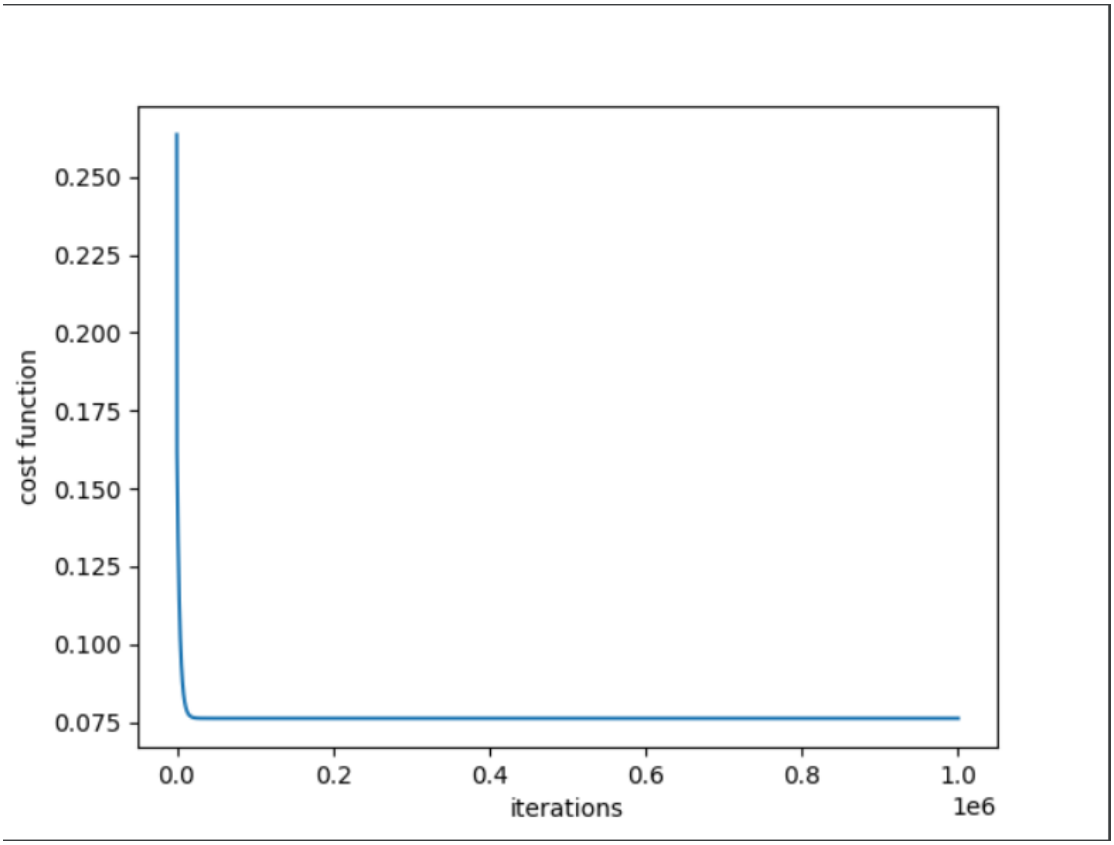
(2) n = 24, max_exp =2 惩罚项=1e-2



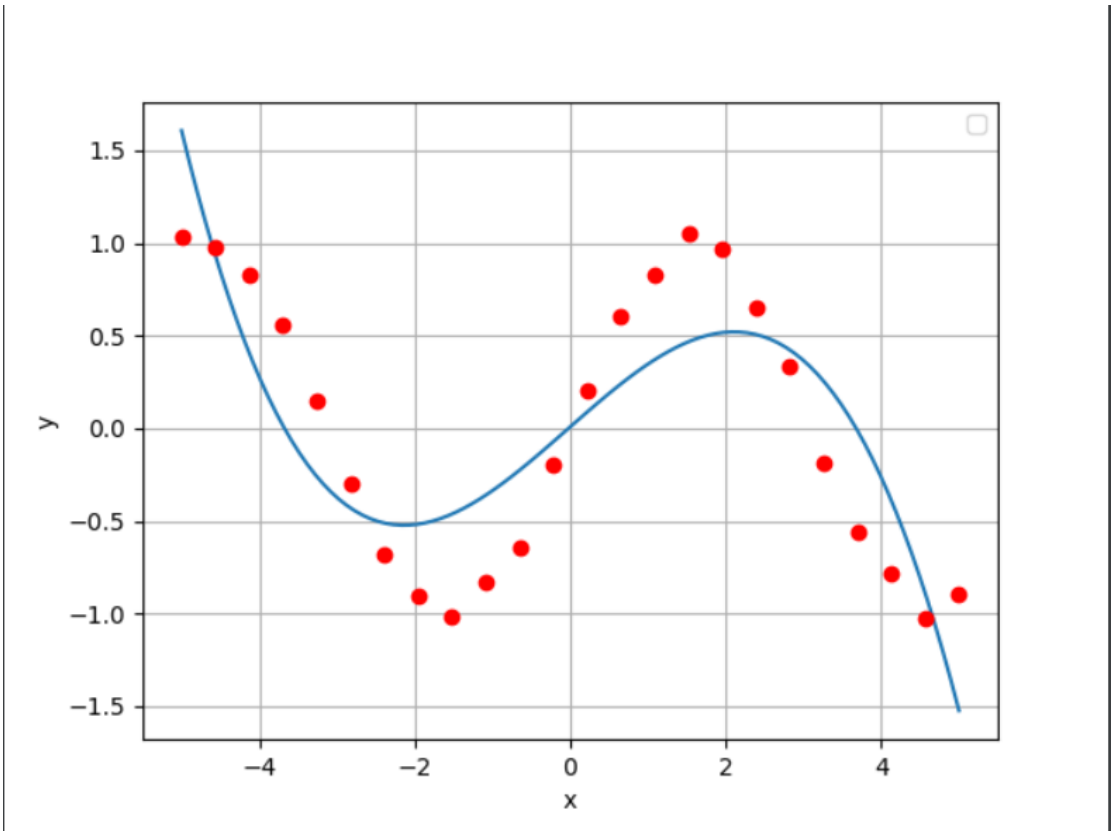


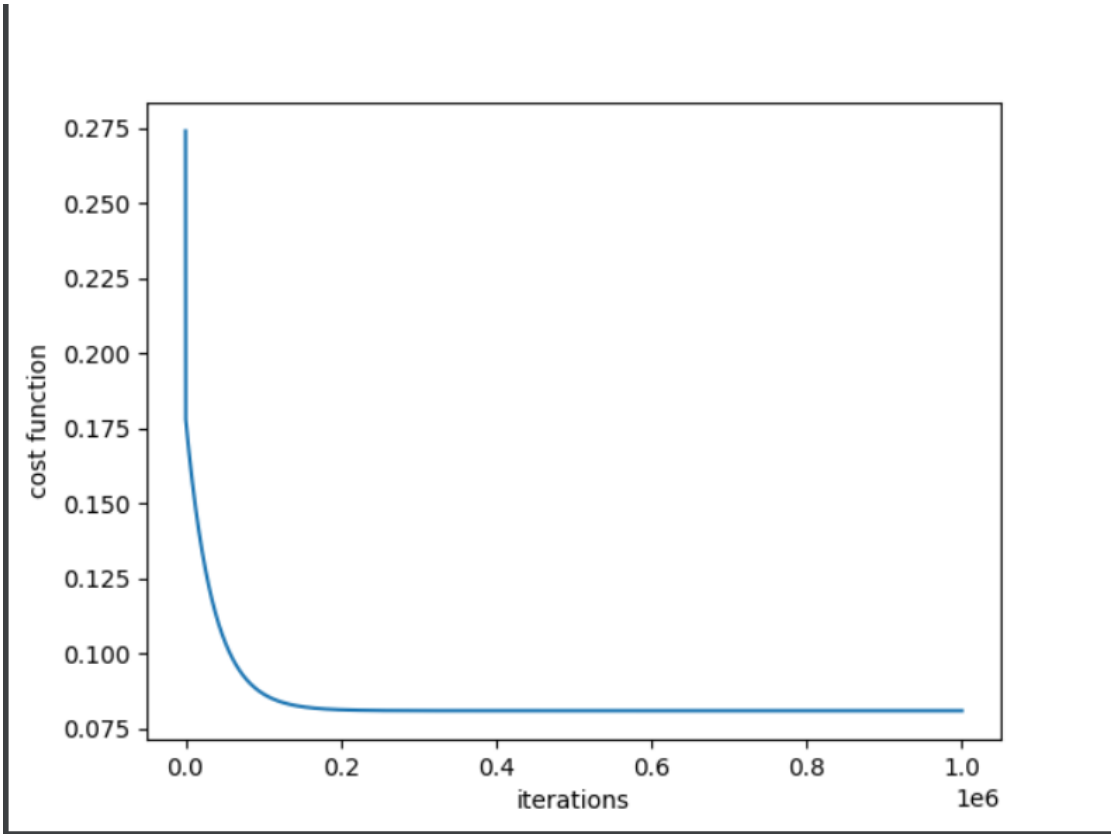
(3) n = 24， max_exp = 3，惩罚项=1e-1



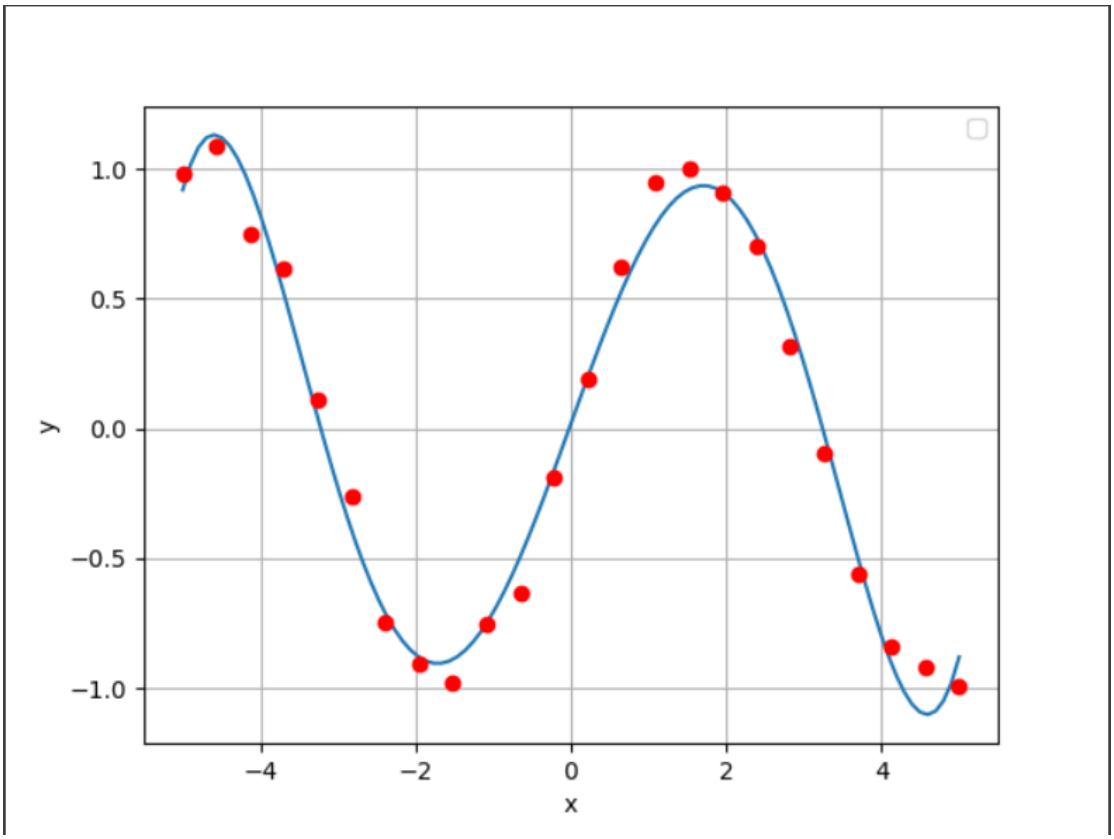


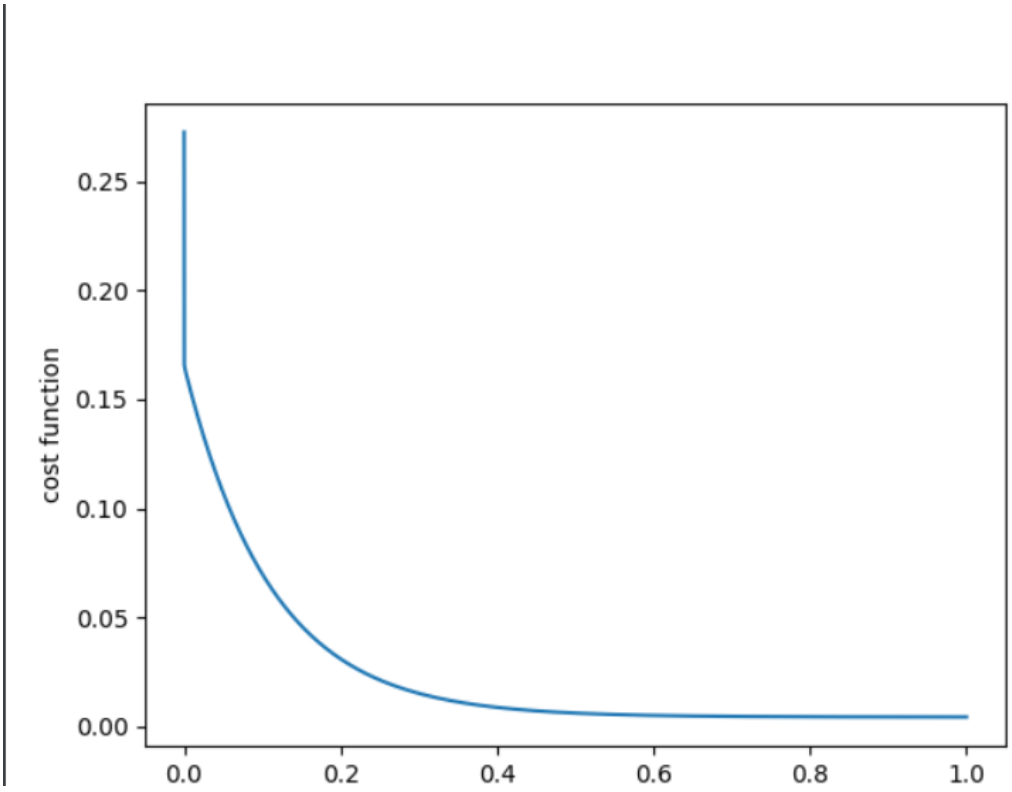
(4) n = 24，max_exp = 4，惩罚项 = 1e-2





(5) n = 24, max_exp = 6, 惩罚项为 1e-2



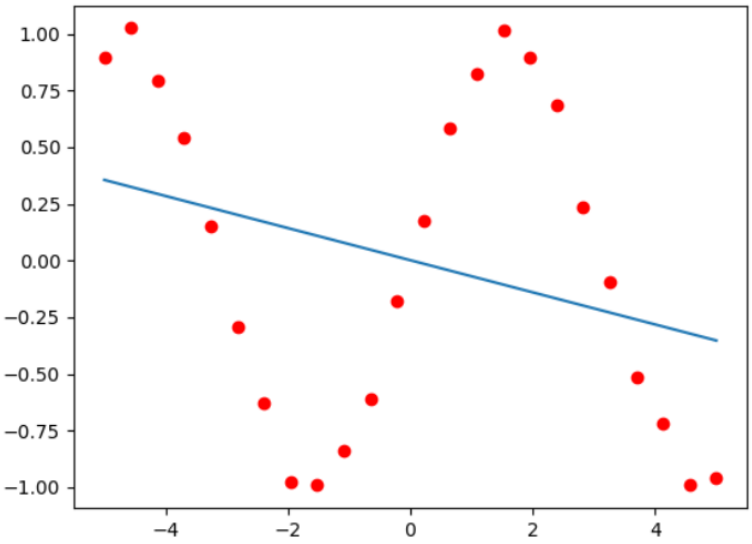


(6) n = 24, max_exp = 9, 惩罚项 = 1e-4
会解决过拟合的问题

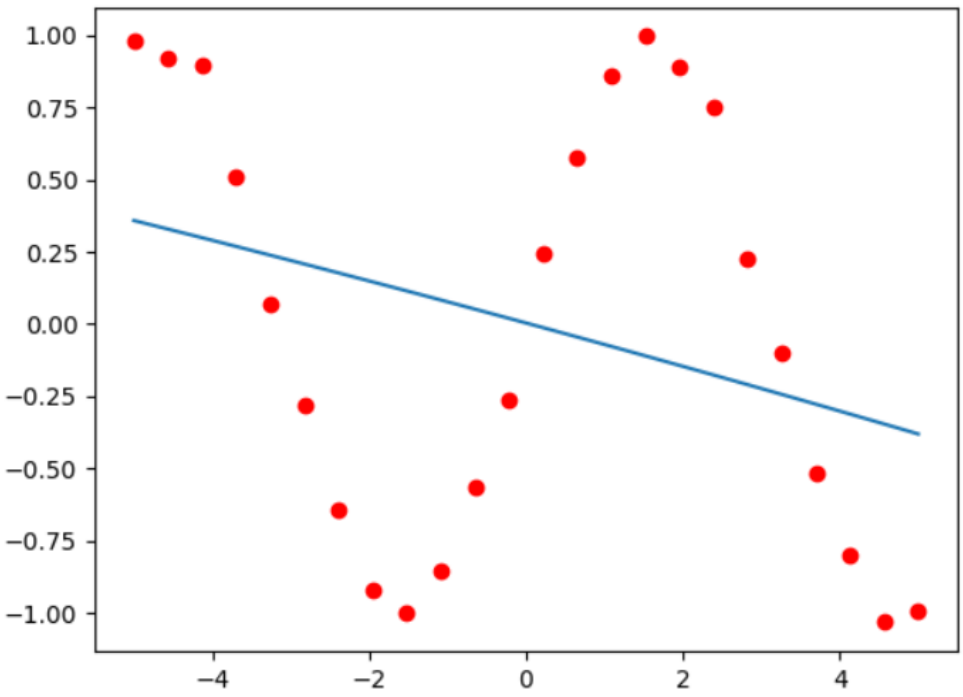
3. 不带惩罚项的正规方程

同上

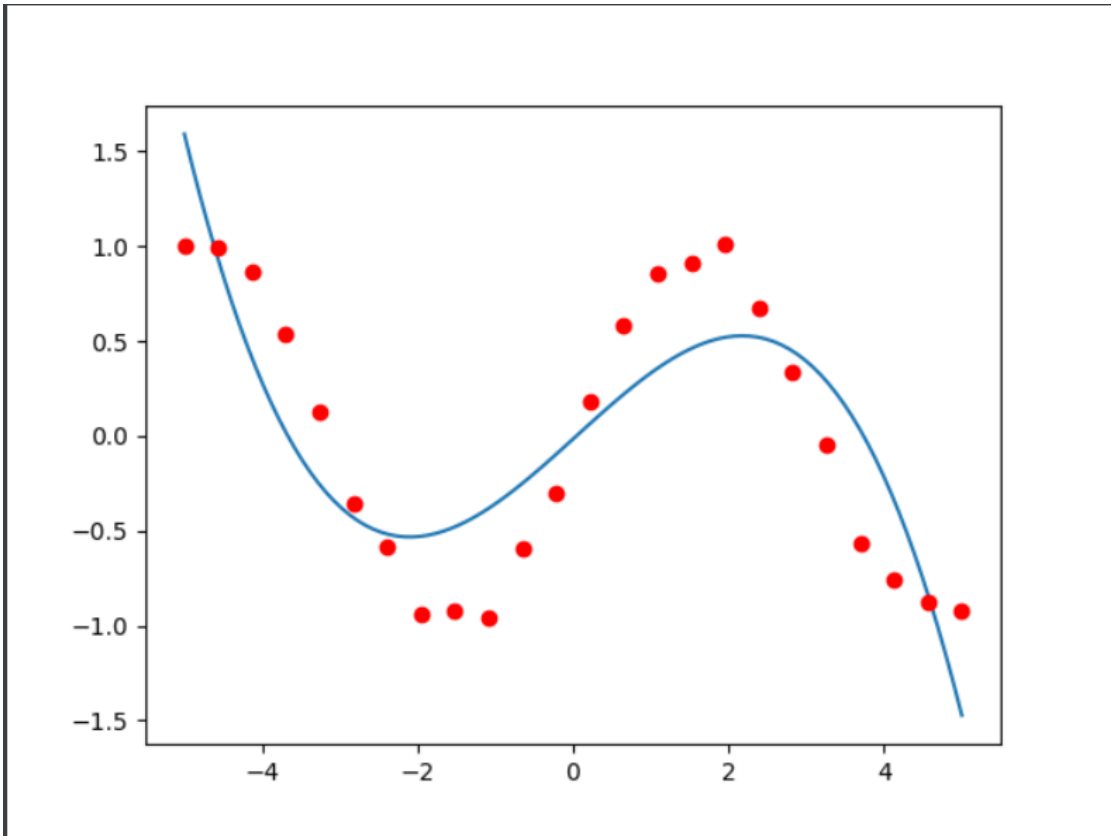
(1) max_exp = 1



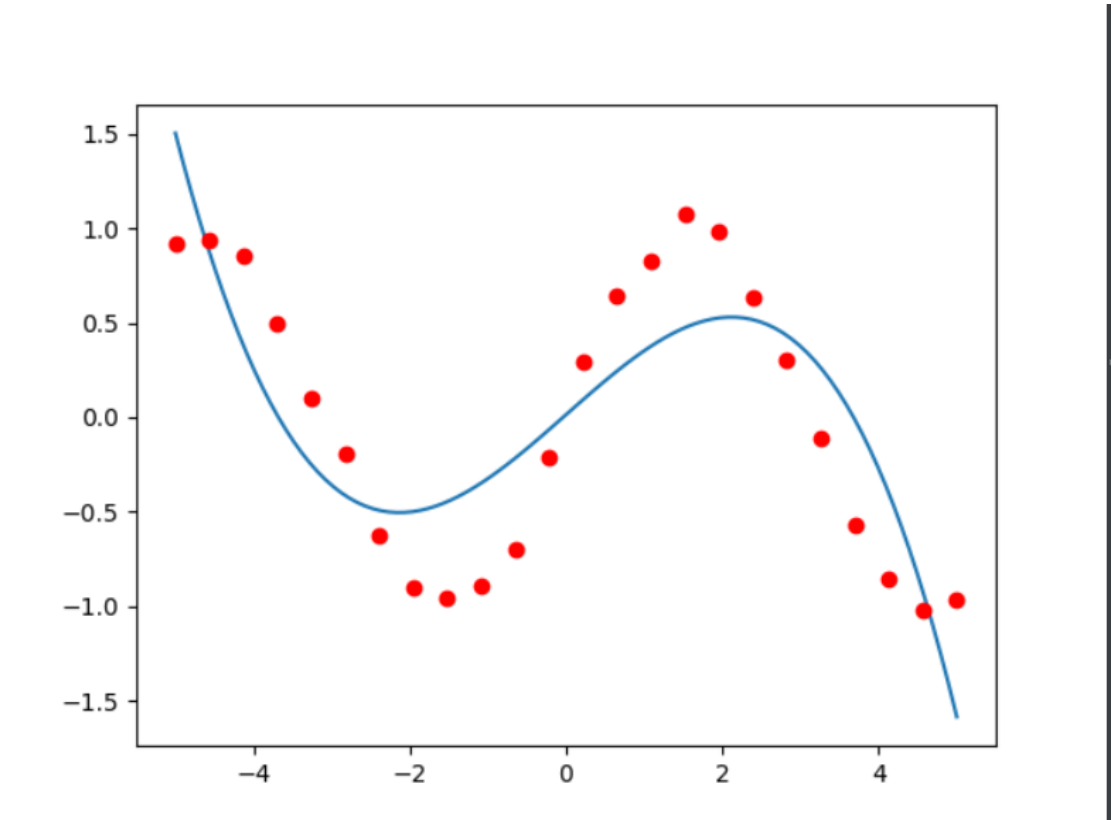
(2) max_exp = 2



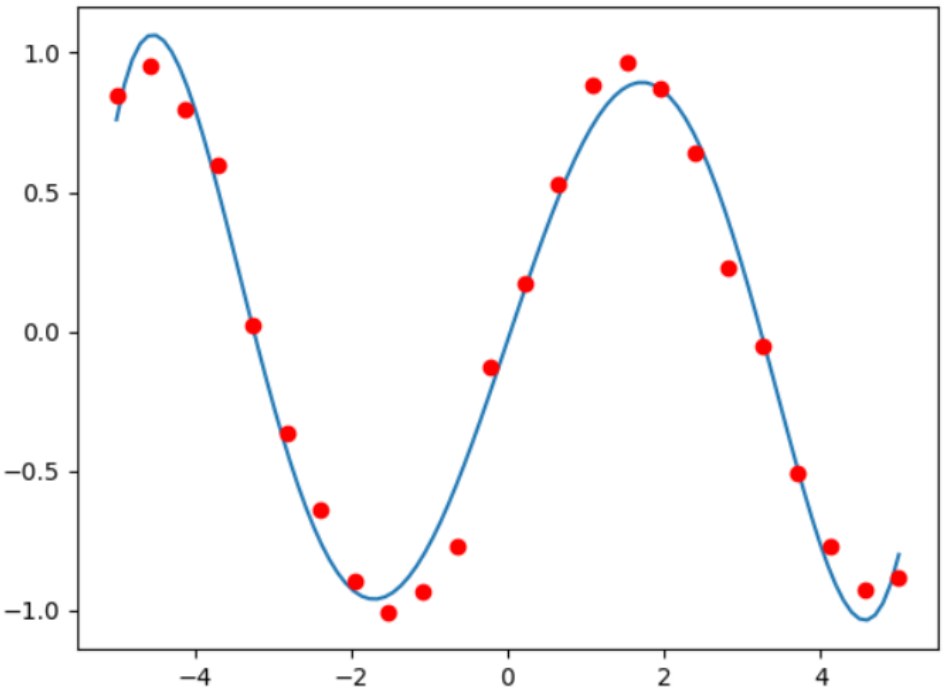
(3) max_exp = 3



(4) max_exp = 4

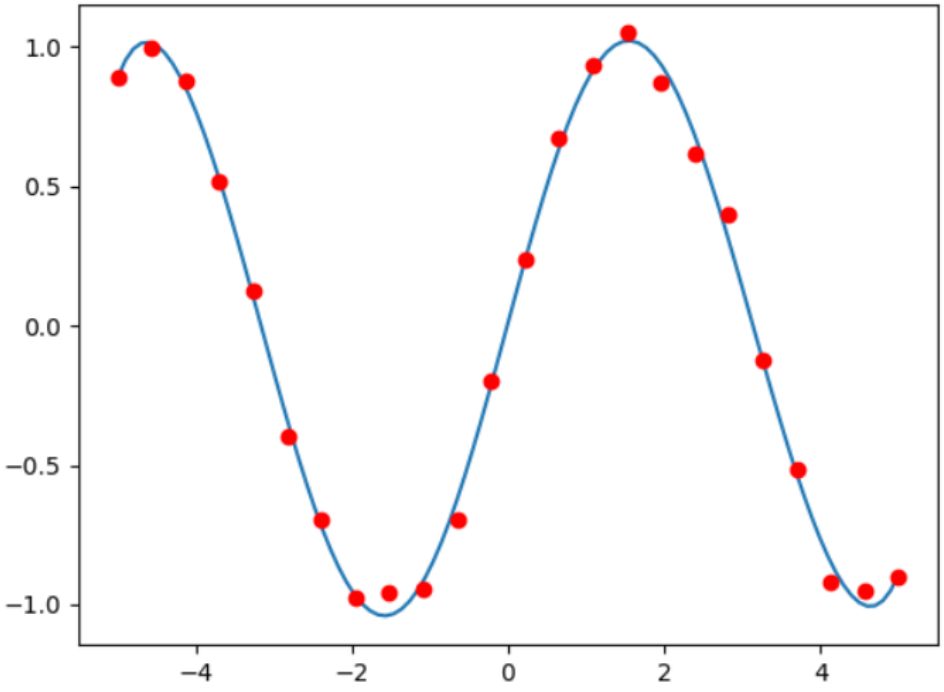


(5) max_exp = 6

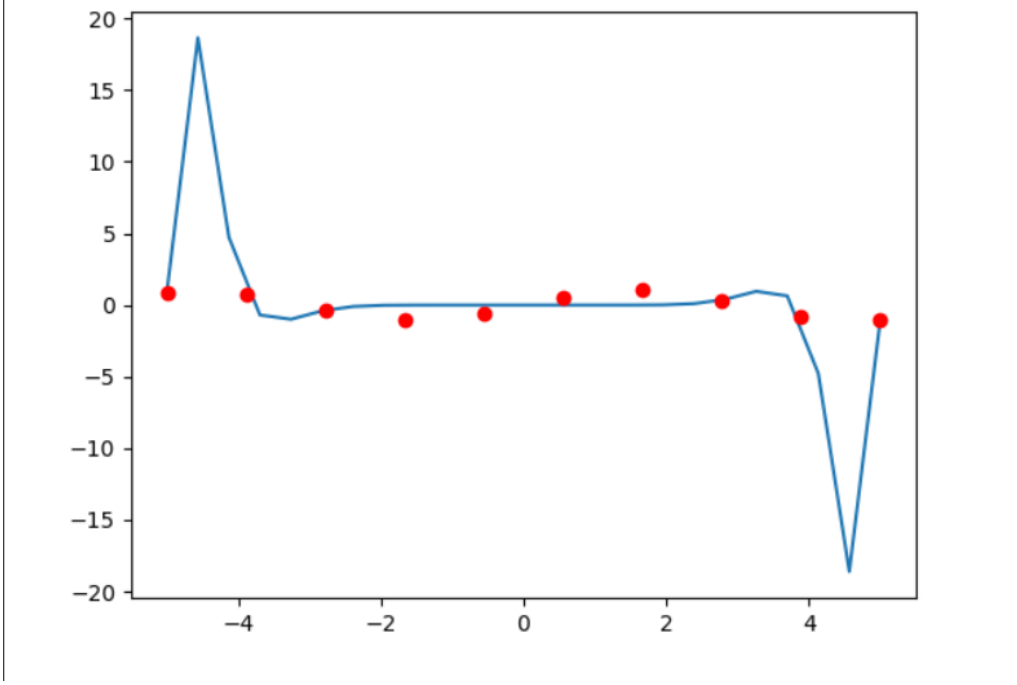


---

(6) max_exp = 9



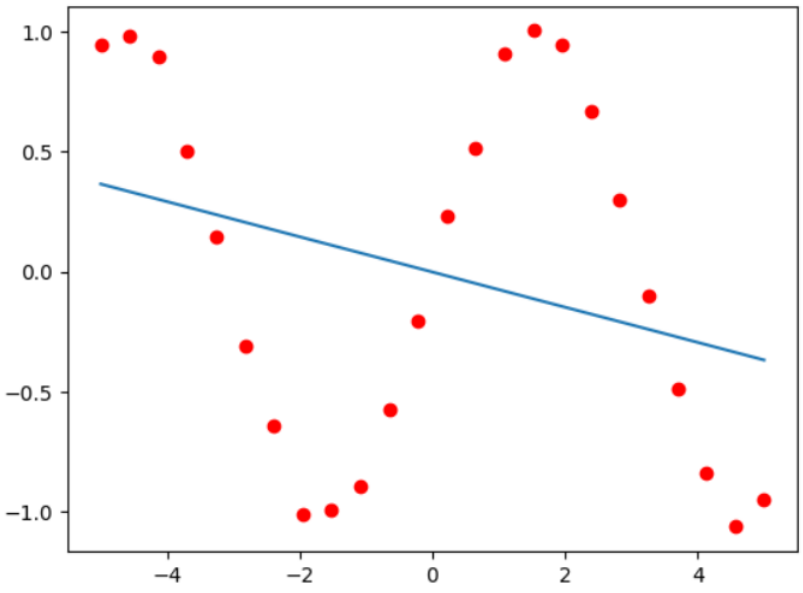
(7)max_exp = 15



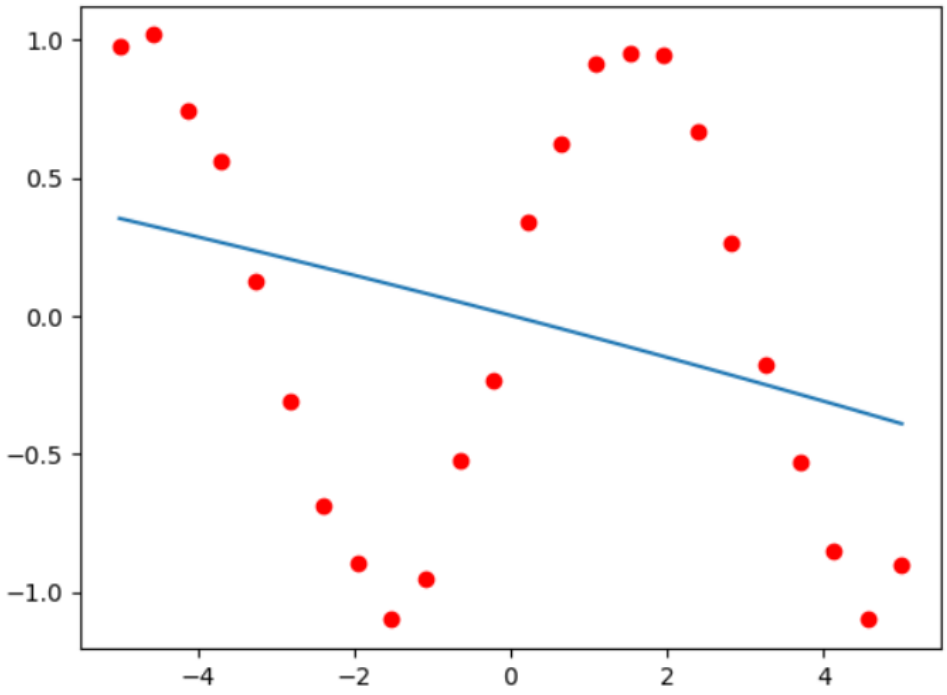
也会出现过拟合的情况

4. 带惩罚项的正规方程

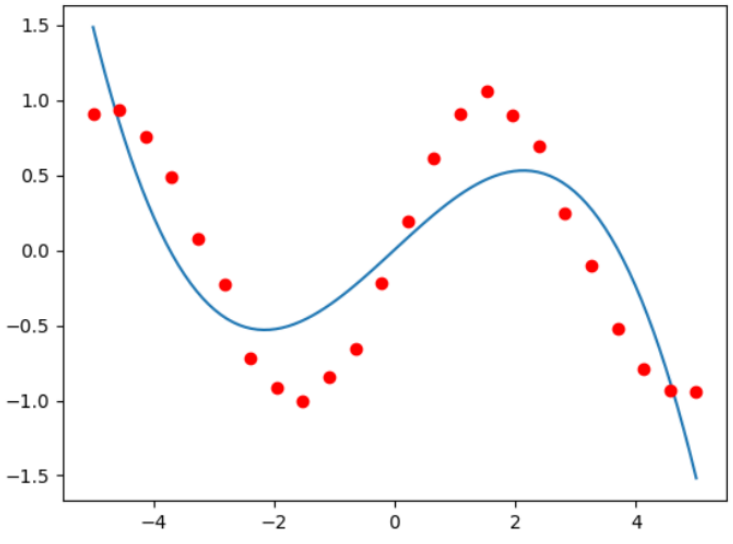
(1) max_exp = 1



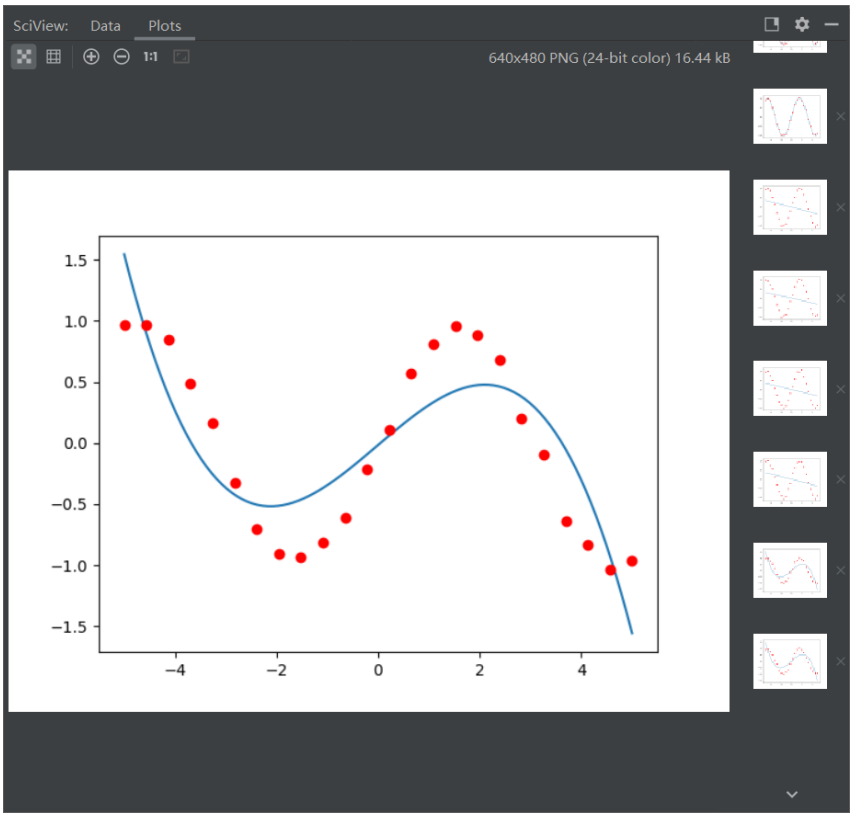
(2) max_exp = 2



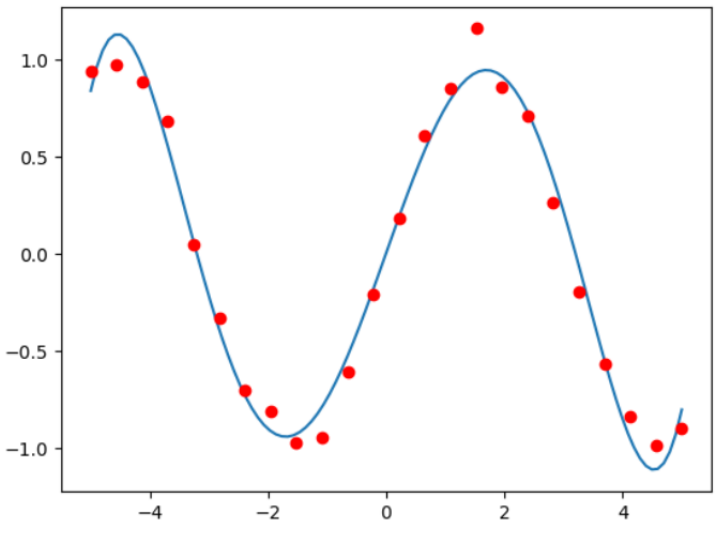
(3) max_exp = 3



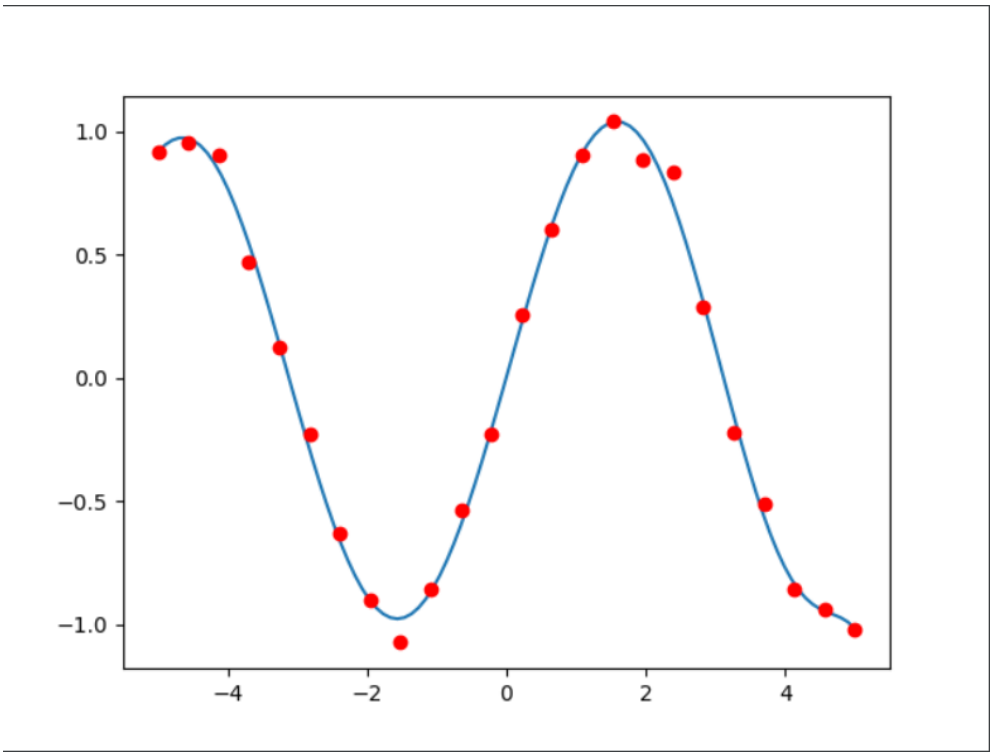
(4) max_exp = 4



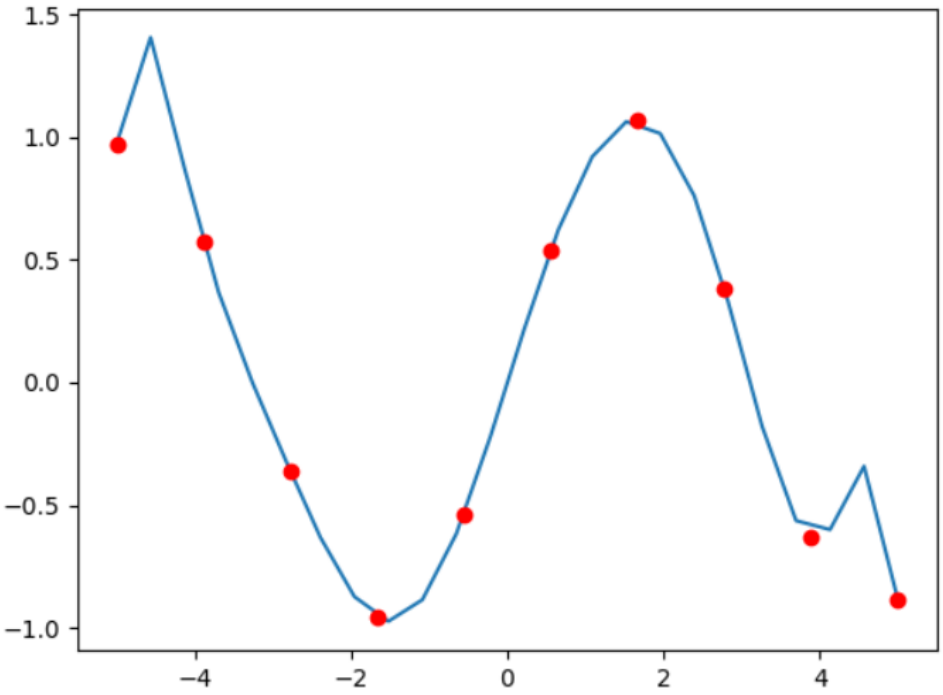
(5) max_exp = 6



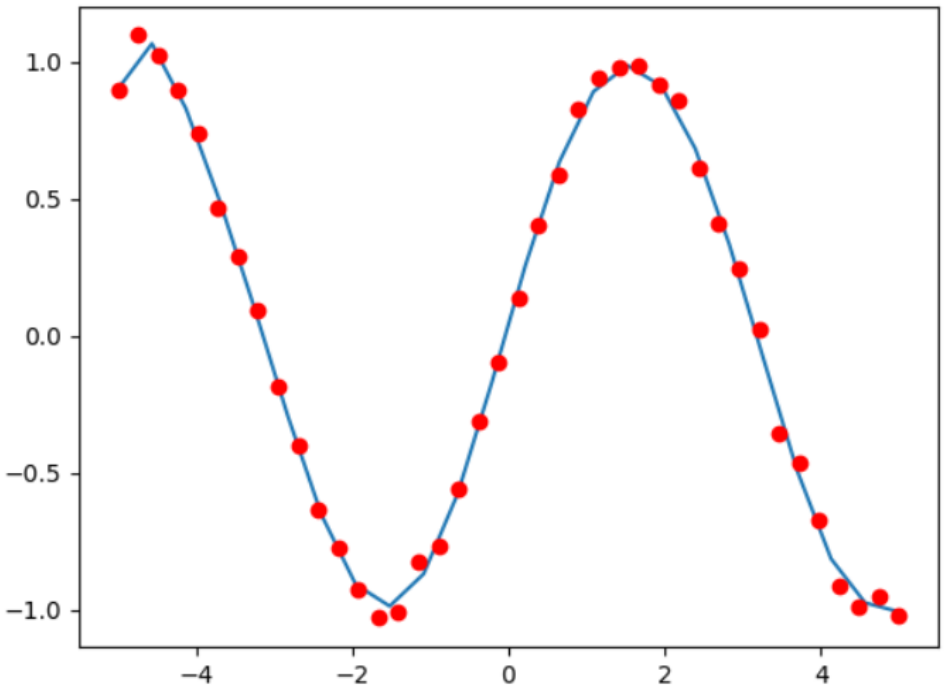
(6) max_exp = 9



(7) max_exp = 15



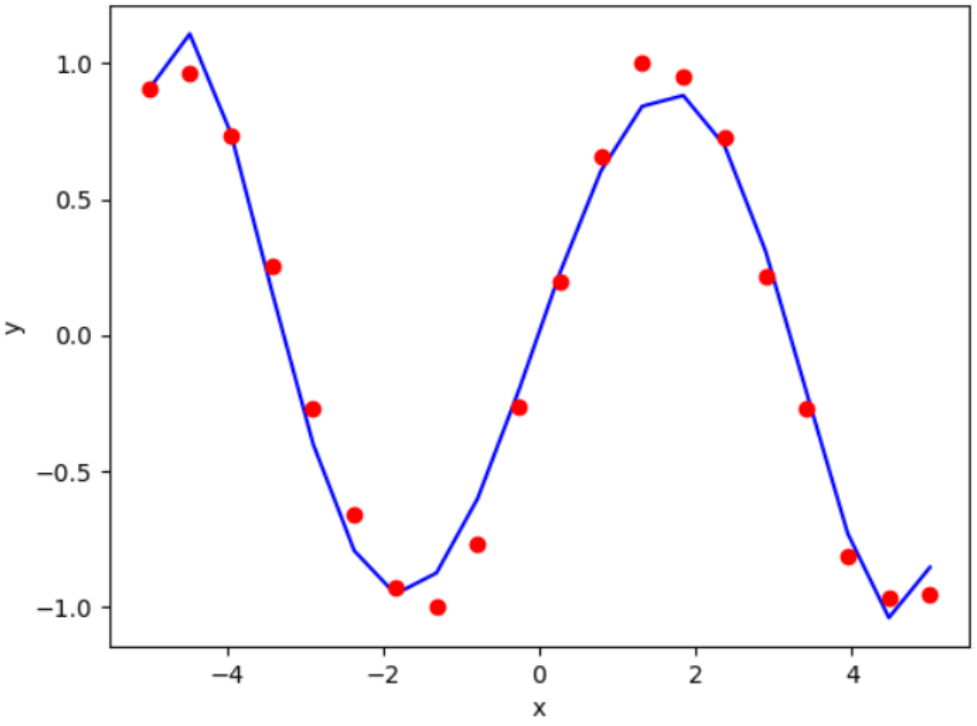
会缓解过拟合的问题，但是仍需要加大数据使现象变得明显

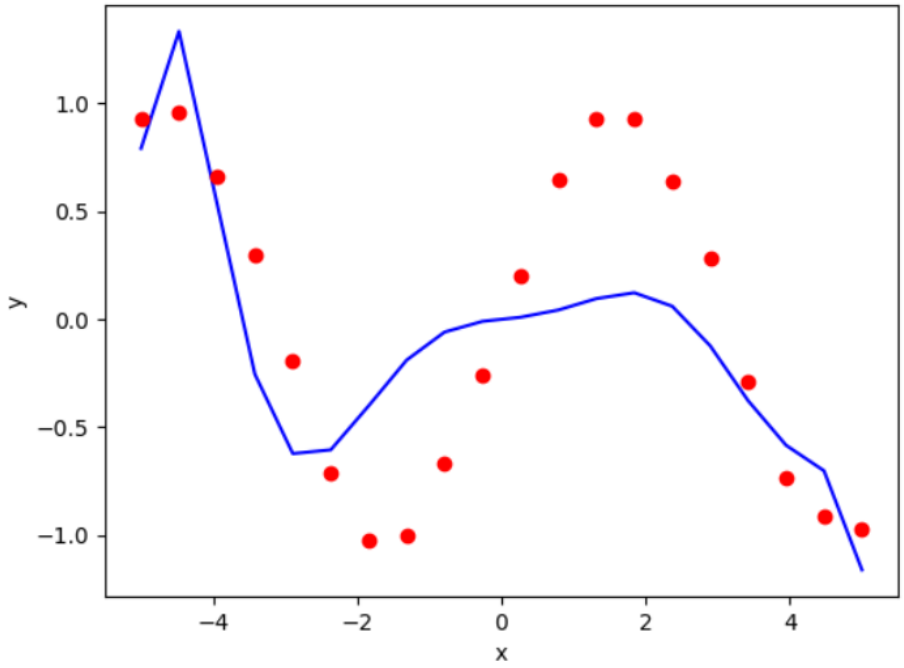


数据从 10 个增加到 40 个的时候，过拟合现象会变得不那么严重

5. 共轭梯度法

这里就不过多的放图了，只选了 max_exp = 6 与 max_exp = 9 时的情况





五、结论

对于梯度下降法来说，我们要选取合适的学习率，这样会使我们的学习效率更高。模型复杂度与模型参数个数和模型参数的绝对值（范数）有关。由后验概率的推导公式，和最大似然（数据）和模型参数有关，所以，加大数据样本，增加参数正则项都对模型有较好的改进。

对于多项式来说，项数越高，需要的学习率越小，即步长会越来越小。对于阶数高的多项式来说，容易出现过拟合的情况，这个时候正则项（惩罚项）也就变得更加重要了。在阶数低，样本比较多时，正则项的存在对最终拟合的结果，影响不大。

从性能方面考虑，梯度下降需要下降很多次，甚至是 10 的 7 次方。在速度方面，共轭梯度与正规方程要强于梯度下降法。

六、参考文献

《机器学习》 - 周志华

机器学习网课 - 吴恩达

七、附录：源代码（带注释）

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt


def compute_cost(theta, X, y):
    # 计算非正则项的代价函数
    inner = np.power((X @ theta.T) - y, 2)
    return np.sum(inner) / (2 * X.shape[0])


def compute_cost_regularization(theta, X, y, lm):
    # 计算正则项的代价函数
    inner = np.power((X @ theta.T) - y, 2) + lm * (theta * theta)
    return np.sum(inner) / (2 * X.shape[0])


# def func(theta, X, y):
#     inner = np.power((X @ theta.T) - y, 2)
#     return inner / (2 * X.shape[0])


def gradient(theta, X, y):
    # 求一个多项式函数的梯度
    temp = (X @ theta.T - y).T @ X / X.shape[0]
    return temp


def dimension(data, dim):
    # 由于是多项式的线性拟合，对数据进行处理
    data = data[None, :]
    data = np.vstack((np.ones_like(data), data)).T
    for i in range(2, dim):
        data = np.insert(data, data.shape[1], np.power(data[:, 1], i), axis=1)
    return data


def armijo_backtrack(theta, X, y, grad, alpha):
    # 自适应学习率算法
    c = 0.3
```

```
41.     now = compute_cost(theta, X, y)
42.     next_v = compute_cost(theta - alpha * grad, X, y)
43.     count = 30
44.     while next_v < now:
45.         alpha = alpha * 2
46.         next_v = compute_cost(theta - alpha * grad, X, y)
47.         count = count - 1
48.         if count == 0:
49.             break
50.     count = 50
51.     while next_v > now - c * alpha * np.dot(grad, grad.T):
52.         alpha = alpha / 2
53.         next_v = compute_cost(theta - alpha * grad, X, y)
54.         count = count - 1
55.         if count == 0:
56.             break
57.     return alpha
58.
59.
60. def normal_equ(X, y, max_exp, regularization, lm):
61.     # 正规方程法，如果 regularization 为 0，则是没有惩罚项的正规矩阵。如果是 1，则是带有惩罚项的正规矩阵
62.     y = y.reshape(len(y), 1)
63.     X = dimension(X, max_exp)
64.     theta = np.zeros(max_exp)
65.     theta = theta.reshape(1, len(theta))
66.
67.     reg_matrix = np.identity(max_exp)
68.     reg_matrix[0, 0] = 0
69.     lm = 1e-4
70.
71.     if regularization == 1:
72.         theta = (np.linalg.pinv(X.T @ X - lm * reg_matrix) @ X.T @ y).T  # 正则化
73.         print(compute_cost_regularization(theta, X, y, lm))
74.     else:
75.         theta = (np.linalg.pinv(X.T @ X) @ X.T @ y).T  # 非正则化
76.         print(compute_cost(theta, X, y))
77.
78.     X_pred = np.linspace(-5, 5, 24)
79.     Y_pred = theta @ dimension(X_pred, max_exp).T
80.
81.     plt.plot(X_pred, Y_pred[0, :])
82.     plt.plot(X[:, 1], y, 'ro')
```

```
83.     plt.show()
84.
85.
86. def gradient_descent(X, y, alpha, iters, max_exp, regularization,
    lm):
87.     # 梯度下降法，如果 regularization 为 0，则是无正则项的梯度下降，
88.     # 如果为 1，则为惩罚项系数为 lm 的带有正则项的梯度下降
89.     X = dimension(X, max_exp)
90.     y = y.reshape(len(y), 1)
91.     theta = np.zeros(max_exp)
92.     theta = theta[None, :]
93.     cost = np.zeros(iters)
94.     count = np.zeros(iters)
95.
96.     if regularization == 1:
97.         for i in range(iters):
98.             cost[i] = compute_cost_regularization(theta, X, y, lm)
99.             count[i] = i
100.                print(cost[i])
101.                theta = (1 - (alpha * lm / X.shape[0])) * theta -
    alpha * gradient(theta, X, y)  # 正则化
102.
103.        else:
104.            for i in range(iters):
105.                cost[i] = compute_cost(theta, X, y)
106.                count[i] = i
107.                print(cost[i])
108.                grad = gradient(theta, X, y)
109.                alpha = armijo_backtrack(theta, X, y, grad, alpha)
110.                theta = theta - alpha * grad  # 非正则化
111.
112.        X_pred = np.linspace(-5, 5, 100)
113.        Y_pred = theta @ dimension(X_pred, max_exp).T
114.
115.        plt.figure()
116.        plt.xlabel('x')
117.        plt.ylabel('y')
118.        plt.plot(X_pred, Y_pred[0, :])
119.        plt.plot(X[:, 1], y, "ro")
120.        plt.legend()
121.        plt.grid(True)
122.
123.        plt.figure()
124.        plt.xlabel('iterations')
```

```
125.        plt.ylabel('cost function')
126.        plt.plot(count, cost)
127.        plt.show()
128.
129.
130.    def conjugate_gradient(X, y, max_exp, lm):
131.        # 共轭梯度法
132.        X = dimension(X, max_exp)
133.        y = y.reshape((len(y), 1))
134.        Q = X.T @ X + lm * np.eye(X.shape[1])
135.        theta = np.zeros((max_exp, 1))
136.        grad = X.T @ X @ theta - X.T @ y + lm * theta
137.        r = -grad
138.        p = r
139.        for i in range(max_exp):
140.            pdm = (r.T.dot(r)) / (p.T.dot(Q).dot(p))
141.            r_prev = r
142.            theta = theta + pdm * p
143.            r = r - (pdm * Q).dot(p)
144.            beta = (r.T.dot(r)) / (r_prev.T.dot(r_prev))
145.            p = r + beta * p
146.
147.        ratio = np.poly1d(theta[::-1].reshape(max_exp))
148.        X_real = np.linspace(-5, 5, 20)
149.        Y_real = np.sin(X_real) + np.random.randn(X_real.shape[0])
    * 0.05
150.        Y_fit = ratio(X_real)
151.        plt.plot(X_real, Y_fit, 'b', label='fit_result')
152.        plt.plot(X_real, Y_real, 'ro', label='real_data')
153.        plt.xlabel('x')
154.        plt.ylabel('y')
155.        plt.show()
156.
157.
158.    def main():
159.        noise = 0.05
160.        X = np.linspace(-5, 5, 24)
161.        y = np.sin(X) + np.random.randn(X.shape[0]) * noise
162.        alpha = 1e-4
163.        iters = int(1e7)
164.        max_exp = 6
165.        lm = 1e-3
166.        gradient_descent(X, y, alpha, iters, max_exp + 1, 0, lm)
    # 梯度下降法
```

```
167.         # normal_equ(X, y, max_exp + 1, 0, lm)
    # 正规方程法
168.         # conjugate_gradient(X, y, max_exp + 1, lm)
    # 共轭梯度法
169.
170.
171.     if __name__ == '__main__':
172.         main()
```