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： 多项式拟合正弦函数

学号：1170300418
姓名：于新蕊

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

windows64，pycharm，python3.0，anaconda

三、设计思想（本程序中的用到的主要算法及数据结构）

算法原理

最小二乘法求解析解

无正则项

目标函数为：

$$
\begin{aligned}
J(\theta) &= \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2 \\
&= \frac{1}{2m}(X\theta-Y)^T(X\theta-Y) \\
&= \frac{1}{2m}\left(\theta^TX^T-Y^T\right)(X\theta-Y) \\
&= \frac{1}{2m}\left(\theta^TX^TX\theta-\theta^TX^TY-Y^TX\theta+Y^TY\right)
\end{aligned}
$$

我们要最小化$J(\theta)$且$J(\theta)$为凸函数，因此

$$
\frac{\partial J(\theta)}{\partial\theta} = \frac{1}{2m}\left(2X^TX\theta-X^TY-X^TY\right)=0
$$

即

$$\theta=\left(X^TX\right)^{-1}X^TY$$

有正则项

$$\begin{aligned}J(\theta)&=\frac{1}{2m}\left[(X\theta-Y)^T(X\theta-Y)+\lambda\theta^TE_n\theta\right]\\&=\frac{1}{2m}\left(\theta^TX^TX\theta-\theta^TX^TY-Y^TX\theta+Y^TY+\lambda\theta^TE_n\theta\right)\end{aligned}$$

那么

$$\frac{\partial J(\theta)}{\partial\theta}=\frac{1}{2m}\left(2X^TX\theta-X^TY-X^TY+2En\theta\right)=0$$

即

$$\theta=\left(X^TX+\lambda E_n\right)^{-1}X^TY$$

梯度下降法

无正则项

选择一个学习率α，迭代次数$epcho$，每次沿梯度下降最快的方向，即导数方向下降，具体来讲：每次迭代可以表示为$\theta_j=\theta_j-\alpha\frac{\partial J(\theta)}{\theta_j}$。

$$\begin{aligned}&\text{repeat epcho times}\{\\&\qquad\theta_j=\theta_j-\tfrac{\alpha}{m}\textstyle\sum_{i=1}^m\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)x_j^{(i)}\\&\}\text{for } j=0\cdots n,\ \text{simultaneous update}\end{aligned}$$

用向量的形式可表示为

$$\theta=\theta-\frac{\alpha}{m}X^T(X\theta-y)$$

有正则项

如果有正则项的话，导数项会多一项，具体变为

$$\begin{aligned}&\text{repeat epcho times}\{\\&\qquad\theta_j=\theta_j-\frac{\alpha}{m}\sum_{i=1}^m(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}\\&\qquad\theta_j=\theta_j(1-\frac{\alpha\lambda}{m})-\frac{\alpha}{m}\sum_{i=1}^m(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}\\&\}\text{for } j=0\ \cdots\ n,\text{simultaneous update}\end{aligned}$$

用向量的形式可表示为

$$tmp = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1-\frac{\alpha\lambda}{m} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1-\frac{\alpha\lambda}{m} \end{pmatrix}_{(n+1)\times(n+1)}$$

$$\theta = tmp \times \theta - \frac{\alpha}{m} X^T(X\theta - y)$$

另外，加了一个trick，动态改变α的值，当cost function增加了的时候代表学习率该减小了。

共轭梯度法

共轭梯度法可以解决这一类问题：求$Ax=b$的解，其中A正定。

设$f(x)=\frac{1}{2}x^TAx-b^Tx$，$f'(x)=Ax-b$。所以最小化$f(x)$的问题等价于求$Ax=b$的解的问题。

接着我们考虑cost function：

$$\begin{aligned} J(\theta) &= \frac{1}{2m}(X\theta-y)^T(X\theta-y) \\ &= \frac{1}{2m}(\theta^TX^T-y^T)(X\theta-y) \\ &= \frac{1}{2m}(\theta^TX^TX\theta-2y^TX\theta+y^Ty) \end{aligned}$$

那么设$A=X^TX$，$b=X^Ty$，那么最小化$J(\theta)$的θ就是$A\theta=b$的解。（注：A一定正定）

共轭梯度法的主要的思想是，梯度下降法每次沿着一个方向下降，并不会朝这个方向走到最低点走到底，这导致梯度下降法收敛很慢。共轭梯度法找出n个共轭正交向量，每次眼这n个方向走到底，使得剩余残差和之前的所有方向向量正交。理论上n次迭代之后就会找到最优解。

但是事实上并不会真的找到最优解，接着迭代cost function还在下降，应该是因为精度的问题。

伪代码：

> $A=X^TX, b=X^Ty$
>
> $\theta_0=$ any
>
> $r0 := b-A\theta_0$ (r_i为第i次迭代的误差)
>
> $d0 := r0$ (d_i是我们要求的共轭向量，在python实现时别忘copy)
>
> $k := 0$ (k表示第几次迭代)
>
> repeat
>
> $\alpha_k := \frac{r_k^Tr_k}{d_k^TAd_k}$ (该项为学习率，是求出来的)
>
> $x_{k+1} := x_k + \alpha_k d_k$
>
> $r_{k+1} := r_k - \alpha_k Q d_k$
>
> 如果r_{k+1}足够小，则提前退出循环 （也就是认为已经找到最优解了）

> $\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$ $d_{k+1} := r_{k+1} + \beta_k d_k$ （Gram-Schmidt过程求d_k）
>
> $k := k + 1$
>
> end repeat
>
> The result is x_{k+1}

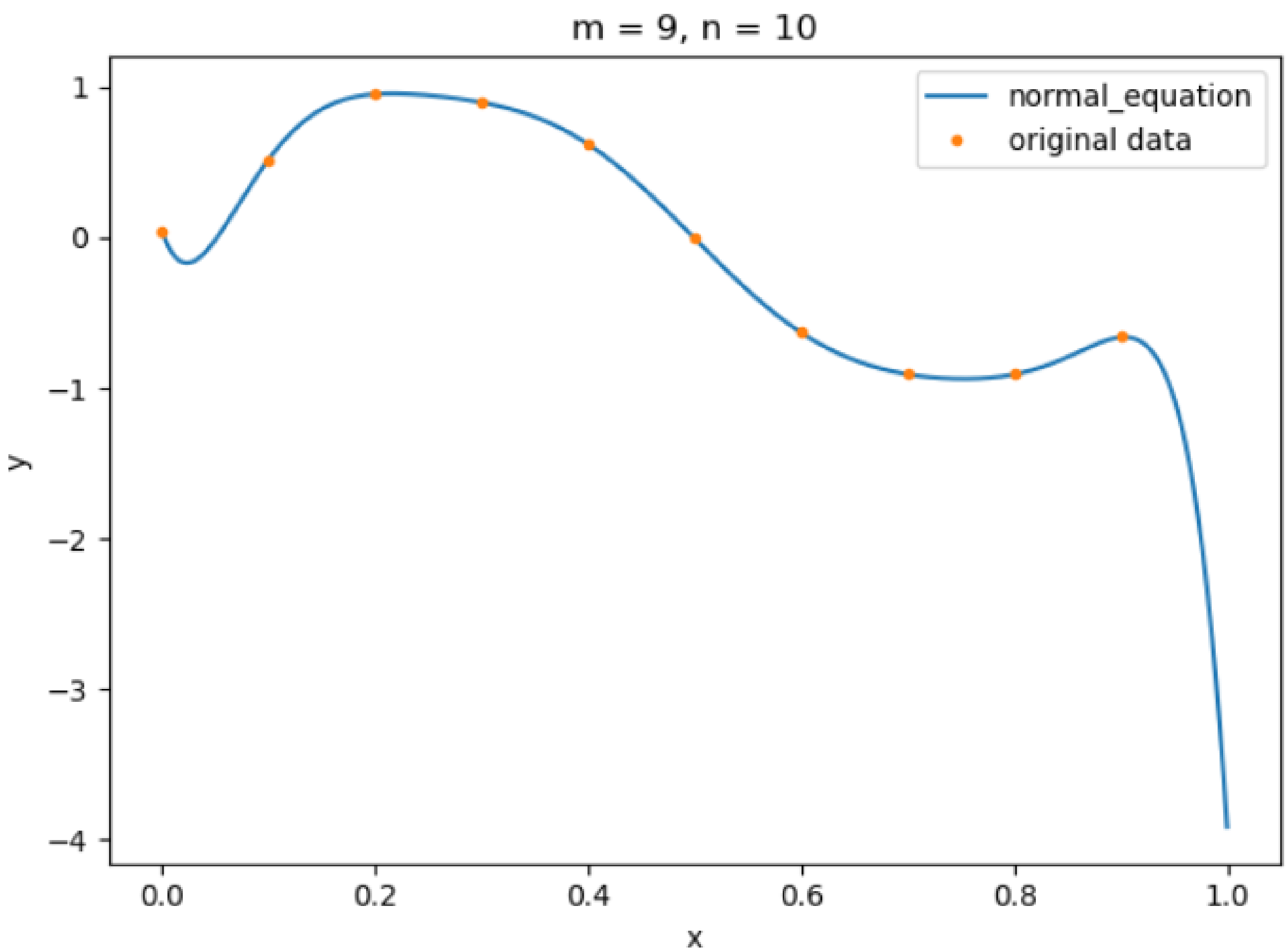
算法的实现

利用numpy array中的二维数组进行矩阵的乘法、加法、数乘等操作，按照上述介绍的算法思想来实现代码。具体参见源代码。

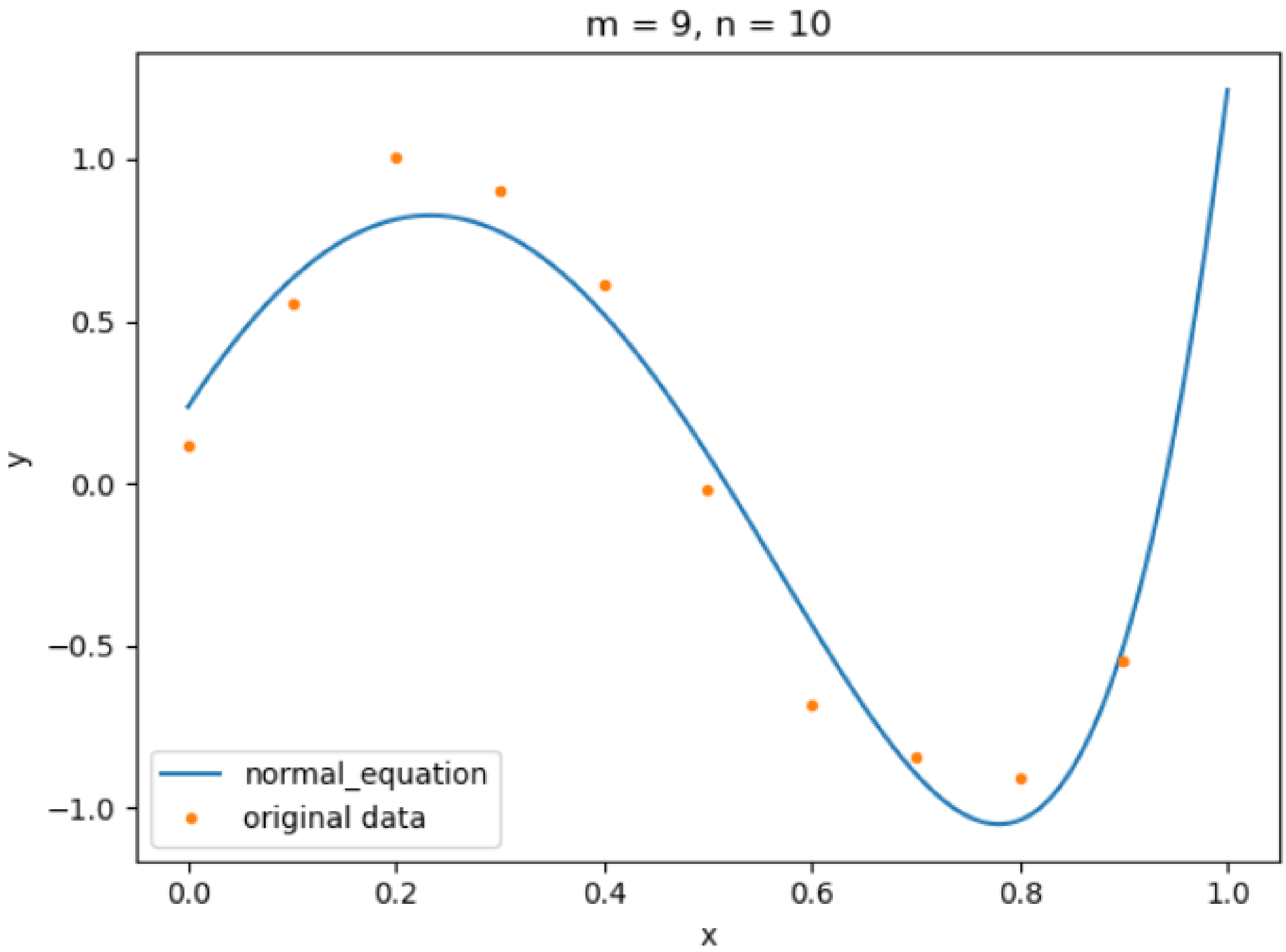
四、实验结果与分析

最小二乘法求解析解

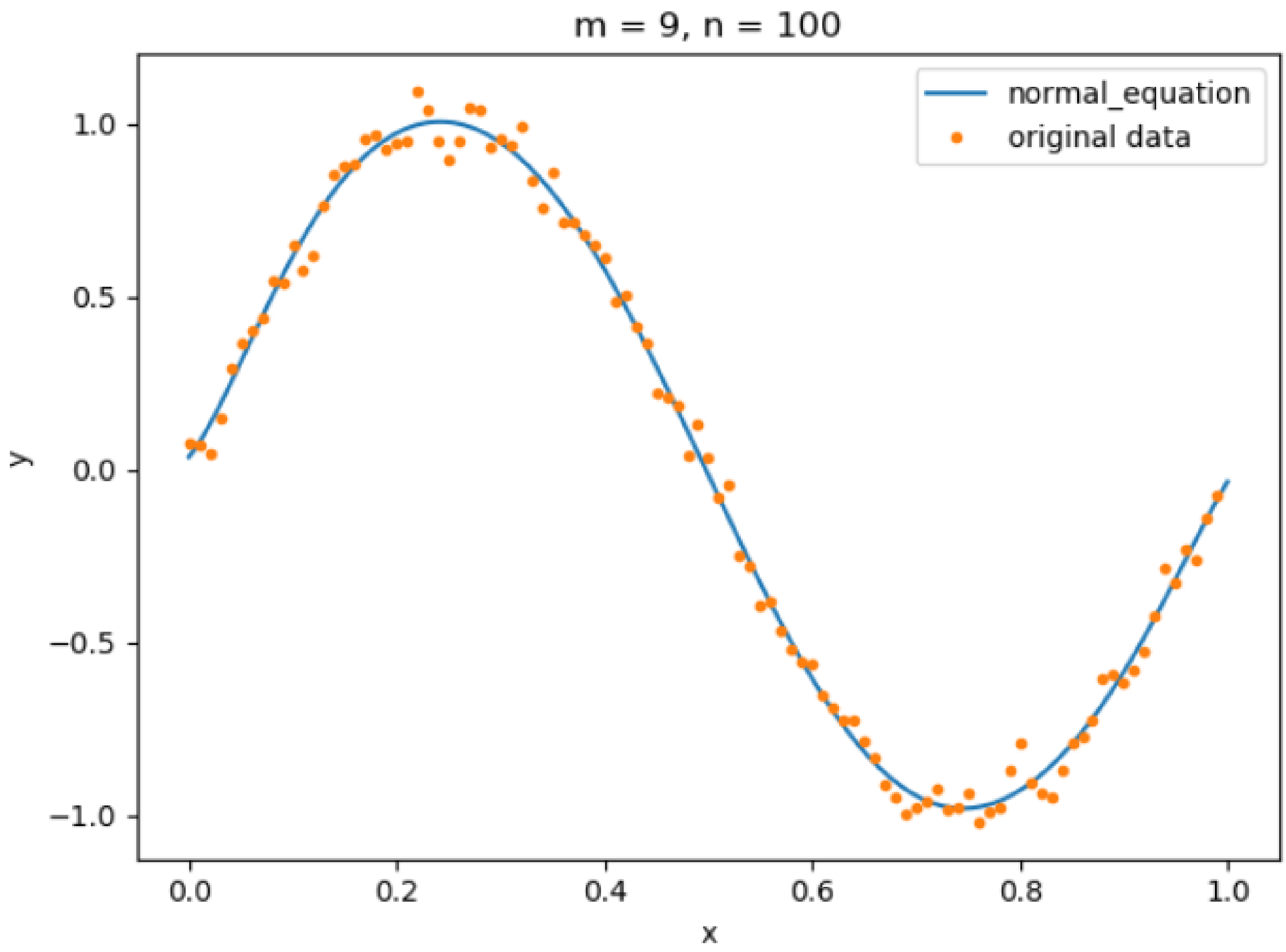
若无正则项，求出来的θ就是满足$J(\theta)$最小的θ。当n和m很接近时，会出现过拟合的现象，如图



若加入正则项，会有效防止过拟合，如图，$\lambda = 0.001$

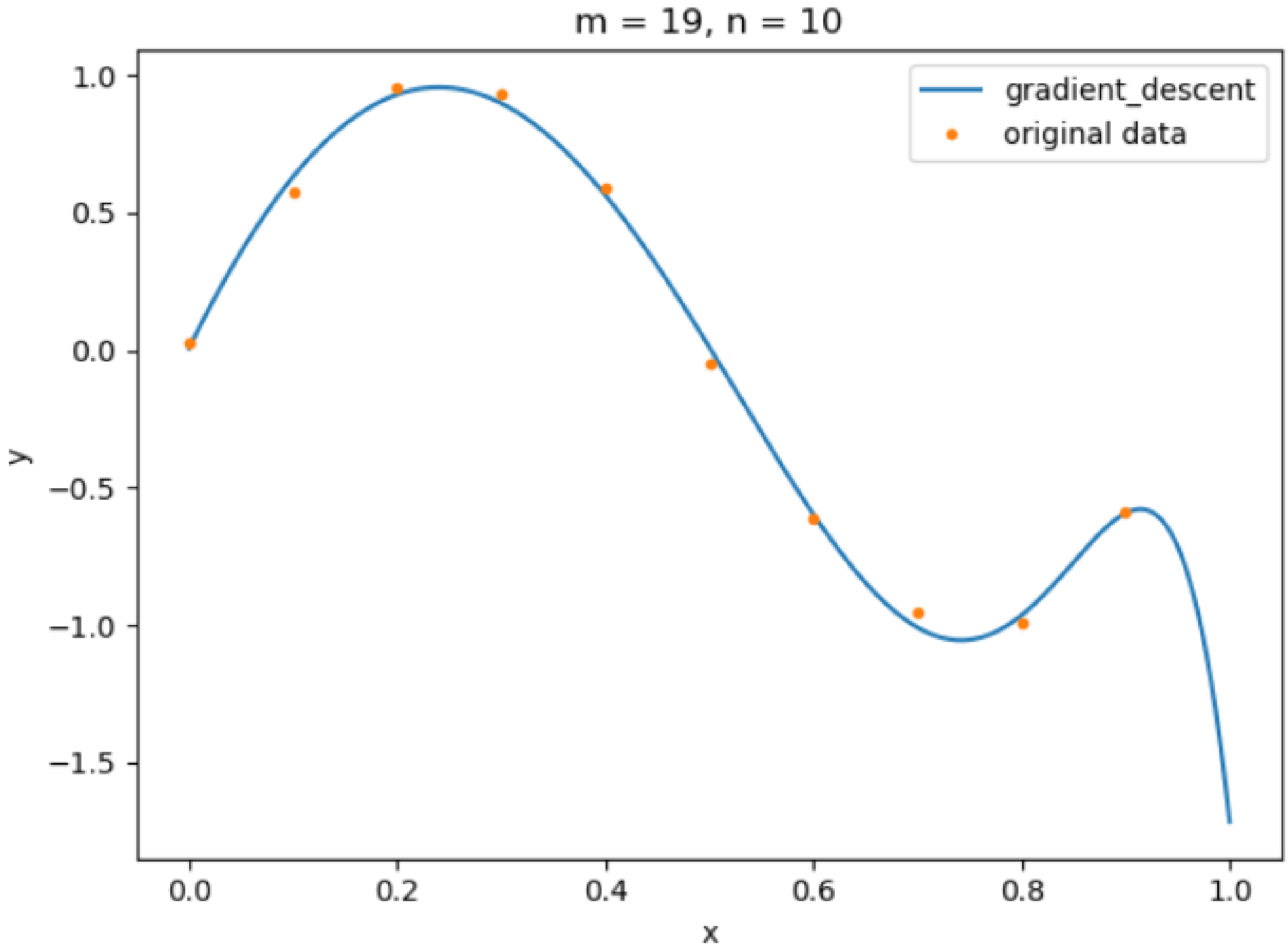


增加样本数量也会防止过拟合，如图

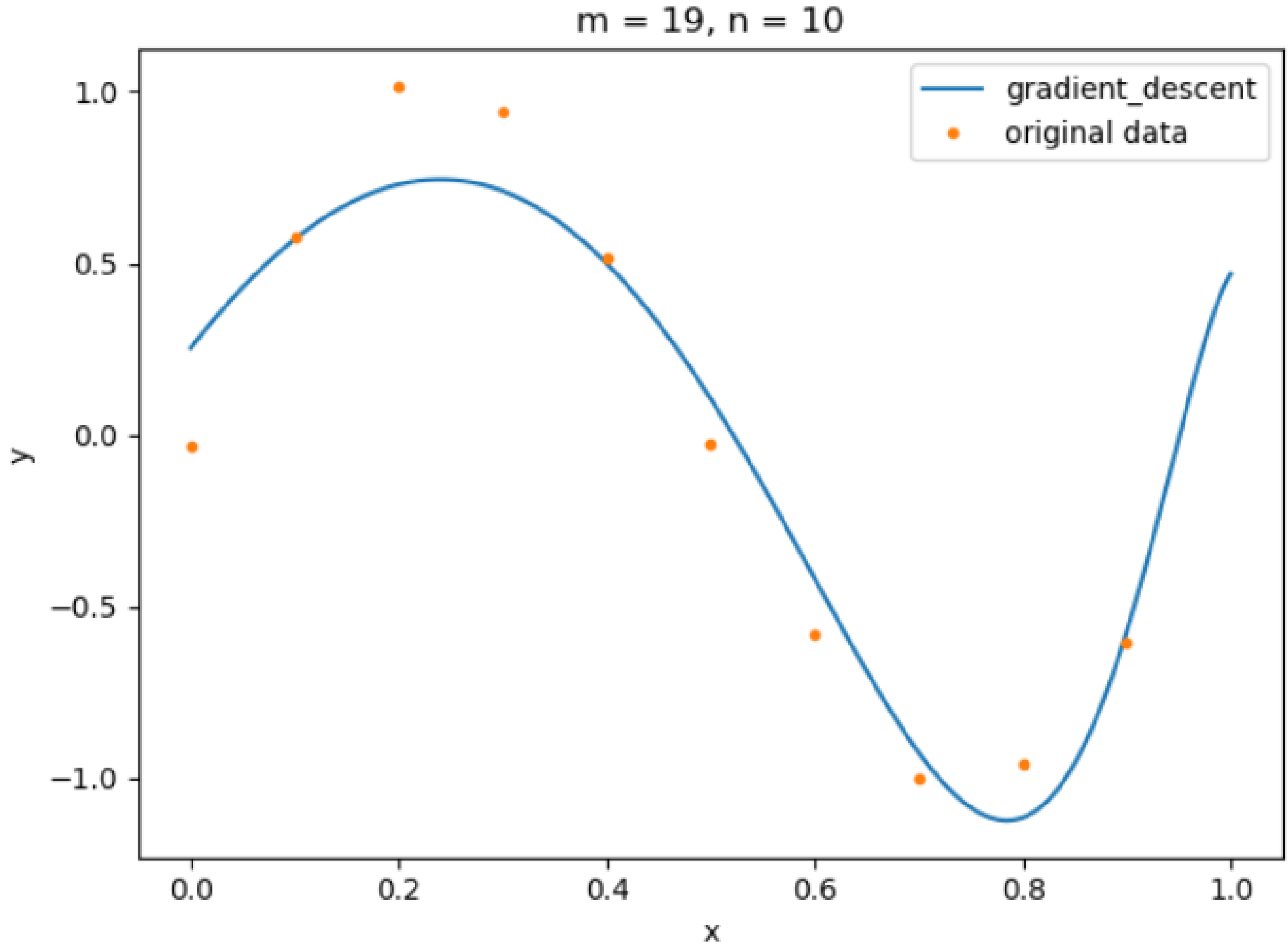


梯度下降法

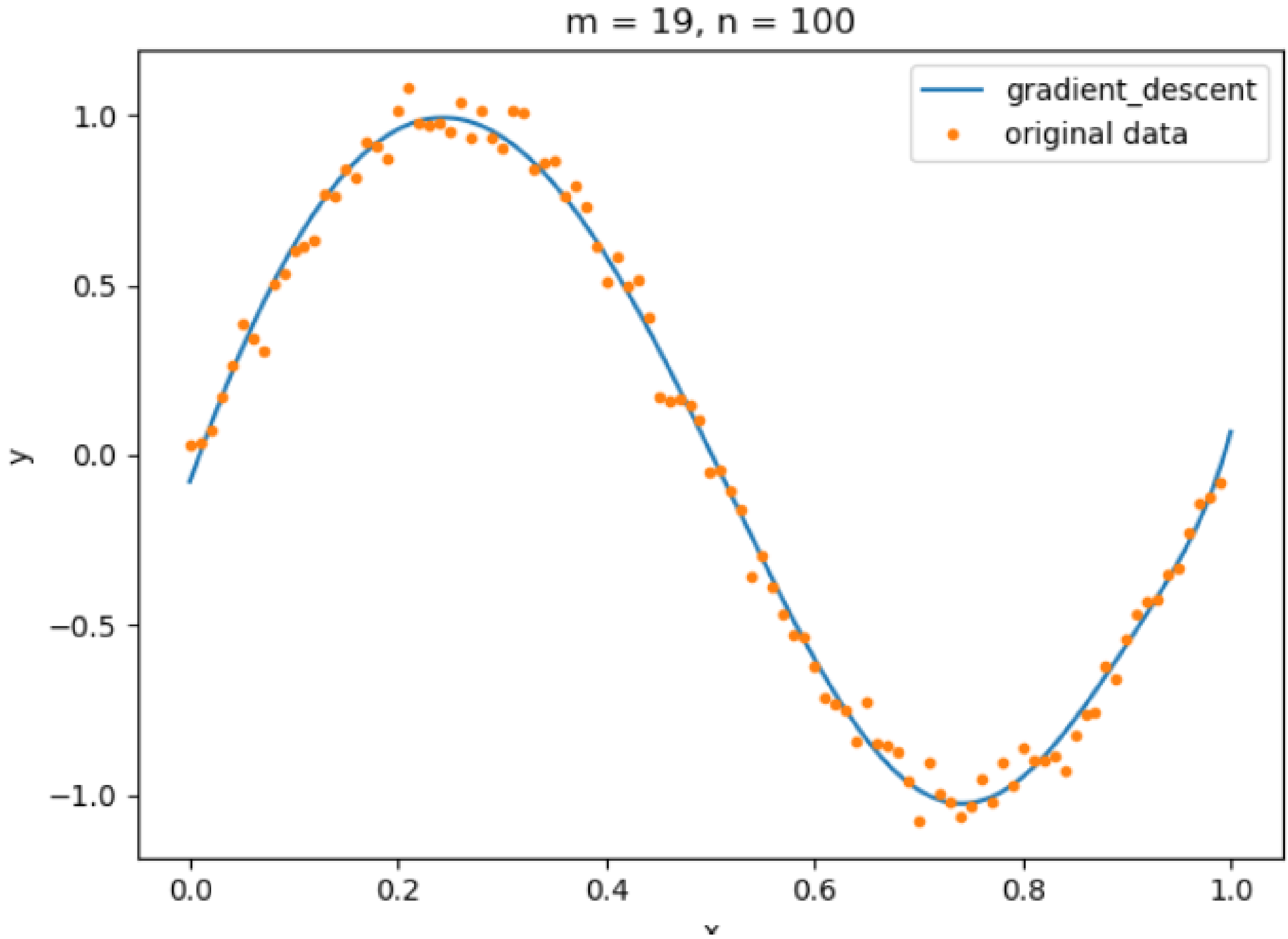
无正则项，当阶数过大时会出现过拟合现象



有正则项，阶数很大，调参为$\alpha = 1, \lambda = 0.003$

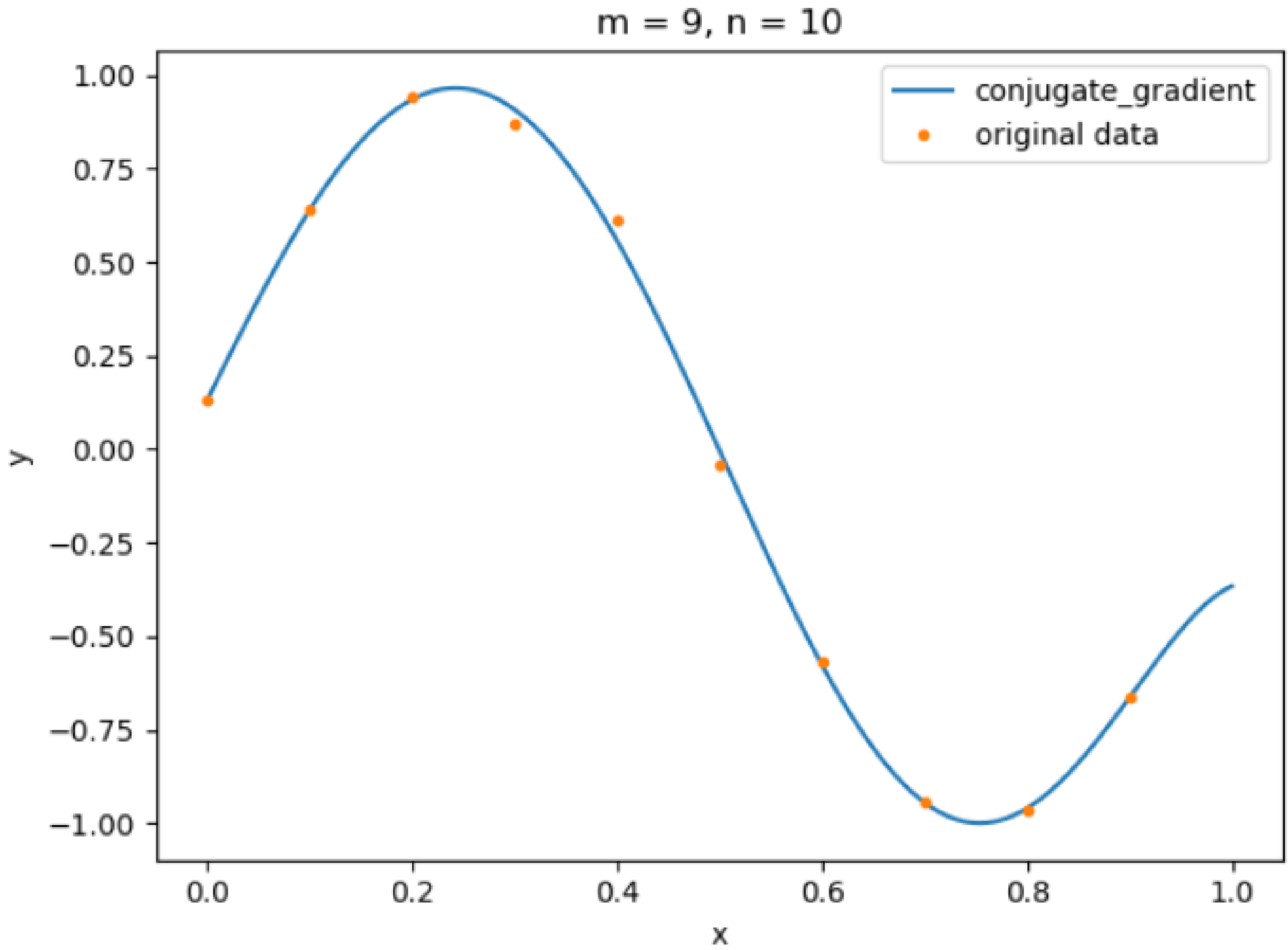


当增加数据样本数量时，拟合效果很好

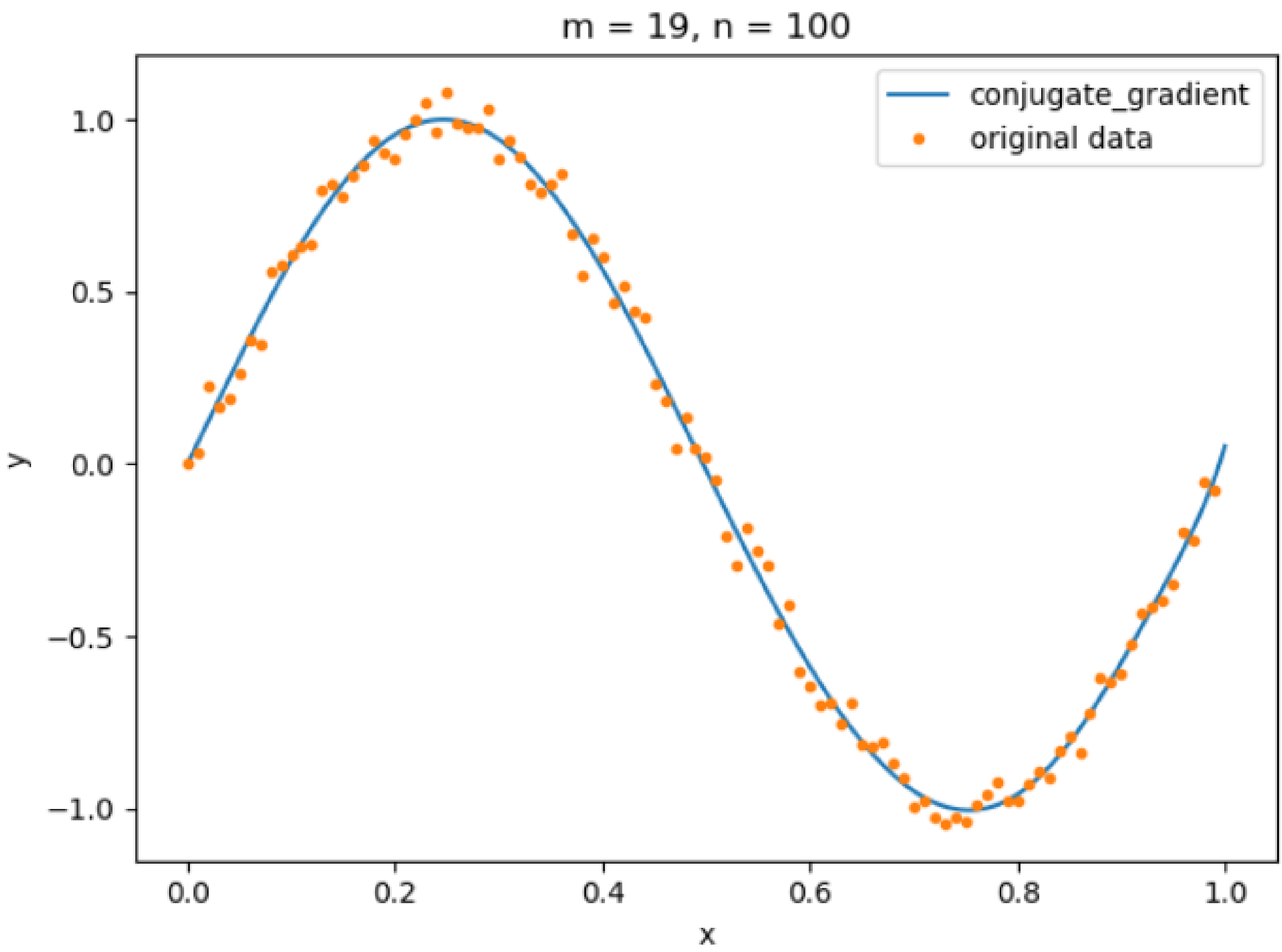

m = 19, n = 100
gradient_descent
original data
y
x
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

共轭梯度法

m=9，n=10


m = 9, n = 10
conjugate_gradient
original data
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0.0
0.2
0.4
0.6
0.8
1.0
x

m=19， n=100



五、结论

对于最小二乘法求解析解，其代码量小，但是时间复杂度为$O(n^3)$，当样本个数很大时其效率过低，样本个数不大时可以采用这个方法；

对于梯度下降法，我们设置一个迭代次数epoch，时间复杂度就是$O(epoch)$，但是梯度下降法需要自己设置一个参数α，参数过大会不收敛，参数过小收敛的会很慢，所以需要多次尝试不同的α；

对于共轭梯度法找出n个共轭正交向量，每次眼这n个方向走到底，使得剩余残差和之前的所有方向向量正交。理论上n次迭代之后就会找到最优解，总体来讲，共轭梯度法较为优秀。

六、参考文献

[吴恩达机器学习(Coursera)]

[最小二乘法推导]

[共轭梯度法]

七、附录：源代码（带注释）

```
def generate_data(m, n, begin=0, end=1):
    """
    生成数据
    :param m: 多项式阶数
    :param n: 训练集大小
```

```python
    :param begin: 区间起点
    :param end: 区间终点
    :return: x, y, x = (n, m+1), y = (n, 1)
    """
    generate_x = np.arange(begin, end, (end - begin) / n)
    generate_y = np.sin(2 * np.pi * generate_x)
    noise = np.random.normal(0, 0.05, n)
    generate_y = generate_y + noise
    data_set = np.empty((n, m + 1))
    for i in range(n):
        data_set[i][0] = 1
        for j in range(1, m + 1):
            data_set[i][j] = data_set[i][j - 1] * generate_x[i]
    return data_set, generate_y.reshape(n, 1)


def normal_equation(x, y, _lambda=0):
    """
    正规方程即最小二乘法求解解析解，默认lambda=0无正则项
    :param x: 矩阵x
    :param y: 列向量y
    :param _lambda: 正则项系数，默认无正则项
    :return: 系数向量theta
    """
    m = x.shape[1] - 1
    # 正则项为 sigma(theta_i ** 2) i = 1 to m，与老师讲的二范数略有区别，这个正则项是参考吴恩达讲的的
    regular = np.eye(m + 1)
    regular[0][0] = 0
    theta = np.dot(np.dot(np.linalg.pinv(np.dot(x.T, x) + _lambda * regular), x.T), y)
    print(cost_function(theta, x, y))
    return theta


def cost_function(theta, x, y):
    """
    计算损失函数
    :param theta: 系数矩阵
    :param x: x矩阵
    :param y: y向量
    :return: 损失函数值
    """
    hx = np.dot(x, theta) - y
    cost = np.sum(np.power(hx, 2)) / (2 * y.size)
    return cost


def gradient_descent(x, y, alpha=0.05, epochs=500000, _lambda=0.0):
    """
    梯度下降法
    :param x: x矩阵
    :param y: y向量
    :param alpha: 学习率
    :param epochs: 迭代次数
```

```python
    :param _lambda: 正则项系数，默认无正则项
    :return: 系数向量theta
    """
    # 正则项为 sigma(theta_i ** 2) i = 1 to m，与老师讲的二范数略有区别，这个正则项是参考吴恩达讲的的
    n, m = x.shape
    theta = np.zeros(m).reshape(m, 1)
    cost0 = 100000
    # y1 = np.array([])
    while epochs > 0:
        hx = (np.dot(x, theta) - y).T
        tmp_theta = (1 - _lambda * alpha / n) * theta - alpha / n * np.dot(hx, x).T
        theta = tmp_theta
        epochs -= 1
        cost1 = cost_function(theta, x, y)
        if cost1 > cost0:
            alpha /= 2
        # print(cost1-cost0)
        cost0 = cost1
    return theta


def conjugate_gradient(x_train, y_train, epochs=10000, eps=1e-10):
    """
    共轭梯度法
    :param x_train: X矩阵
    :param y_train: y向量
    :param epochs: 迭代次数
    :param eps: 精度
    :return: 系数向量theta
    """
    m, n = x_train.shape
    theta = (np.zeros(n)).reshape(n, 1)
    a = np.dot(x_train.T, x_train)
    b = np.dot(x_train.T, y_train)
    r0 = b - np.dot(a, theta)
    d0 = np.copy(r0)
    for i in range(epochs):
        alpha = np.sum(np.dot(r0.T, r0) / np.dot(np.dot(d0.T, a), d0))
        theta = theta + alpha * d0
        r1 = r0 - alpha * np.dot(a, d0)
        beta = np.sum(np.dot(r1.T, r1) / np.dot(r0.T, r0))
        if np.dot(r1.T, r1) < eps:
            return theta, i + 1
        d1 = r1 + beta * d0
        d0 = np.copy(d1)
        r0 = np.copy(r1)
        # print(cost_function(theta, x_train, y_train))
    return theta, epochs
```