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：1190301610
姓名：王家琪

# 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、Python3.7

# 三、设计思想（本程序中的用到的主要算法及数据结构）

本程序是想要用最小二乘法来拟合多项式函数。对于损失函数，我们设计两种，一种是均方误差，一种是加入惩罚项之后的均方误差。对于参数结果，我们一共通过三种方式来求：解析法、梯度下降法、共轭梯度法。

最小二乘法面对的任务和基本思想是：有 n 个样本$(x_i,y_i)\,(i=1,2,\ldots\ldots,n)$ ，采用多项式 $h_\theta(x)$来拟合它们。

$$h_\theta(x)=w_0+w_1x+\ldots\ldots+w_mx^m$$

设：

$$X=\begin{bmatrix}1 & x_1 & \cdots & {x_1}^m\\ 1 & x_2 & \cdots & {x_2}^m\\ \vdots & \vdots & \ddots & \vdots\\ 1 & x_n & \cdots & {x_n}^m\end{bmatrix},\ W=\begin{bmatrix}w_0\\ w_1\\ \vdots\\ w_m\end{bmatrix},\ Y=\begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_n\end{bmatrix}$$

最小二乘法用损失函数（均方误差）表示预测值和本身值之间的误差，我们的任务就是求$W$，使得$J(w)$最小，$J(w)$的公式如下图所示：

$$J(w)=\frac{1}{2}(XW-Y)^T(XW-Y)$$

导数为：

$$\nabla J_\theta(W)=X^T(XW-Y)$$

令导数为 0 得到了解析解：

$$W=(X^TX)^{-1}X^TY$$

当函数的次数过大之后，容易产生过拟合现象，我们加入惩罚项来降低次数。加入正则项之后的损失函数为：

$$J(w)=\frac{1}{2}(XW-Y)^T(XW-Y)+\frac{\lambda}{2}\|W\|^2$$

导数为：

$$\nabla J_\theta(W)=X^T(XW-Y)+\lambda W$$

得到的解析解为：

$$W=(X^TX+\lambda E)^{-1}X^TY$$

除了直接求出解析解之外，我们还可以通过优化的方法来求最优的参数$W$。由于最小二乘法得到的二次函数是一个凸函数，所以理论上只有一个全局最优解，不用担心陷入局部最优解的问题，优化算法比较经典的有梯度下降法、牛顿梯度法、共轭梯度法等等。

**梯度下降法算法：**

1、随机初始化$W$

2、求当前参数$W$下的$\nabla J(w)$，前文已给出。

3、更新$W$，更新公式为：$W=W-\alpha\frac{\partial J(w)}{w}$，其中$\alpha$为学习率。

4、计算损失函数$J(w)$，如果小于给定值，停止迭代，否则回到步骤 2。

**共轭梯度法算法：**

1、 给定迭代精度$0\leqslant\epsilon\leqslant1$，和初始值$x_0$.计算$g(x_0)=\nabla f(x_0)$.令$k\leftarrow0$

2、 若$\|g_k\|^2\leqslant\epsilon$，停止迭代，输出$x$

3、 计算搜索方向$d_k$

$$d_k=-g_k\ \ k=0$$
$$d_k=-g_k+\alpha_kd_{k-1}\ \ k>0$$

4、 计算$\alpha_k=\frac{{g_k}^Tg_k}{{g_{k-1}}^Tg_{k-1}}$

5、 令$x_{k+1}\leftarrow x_k+a_kd_k$，并计算$g_{k+1}=\nabla f_{k+1}(x)$

6、 $k\leftarrow k+1$,转到第二步。

**具体算法：**

1、 数据加入噪声：

```
x=np.arange(-1,1,0.01)
ys=[np.sin(2*np.pi*i) for i in x]
```

```
y=[]
#加入噪声
for i in range(len(ys)):

    z=np.random.normal(0，0.25)

    y.append(ys[i]+z)
```

需要注意的是，最好将 x 的范围设置在[-1,1]之间，如果 x 的初始范围设置的太大，有可能导致梯度下降法在迭代的时候出现梯度爆炸的现象，而导致无法收敛到最优解。这种处理方法就是我们通常所说的数据归一化处理。

2、 定义损失函数，计算误差：

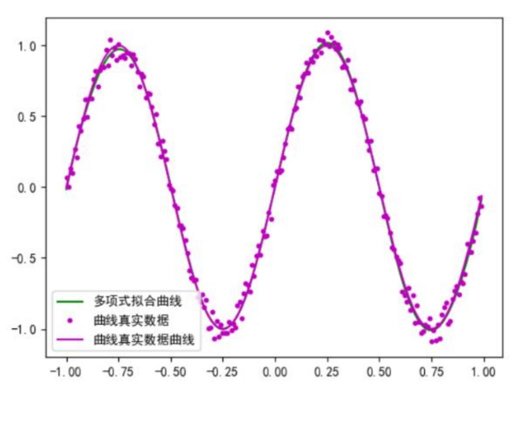
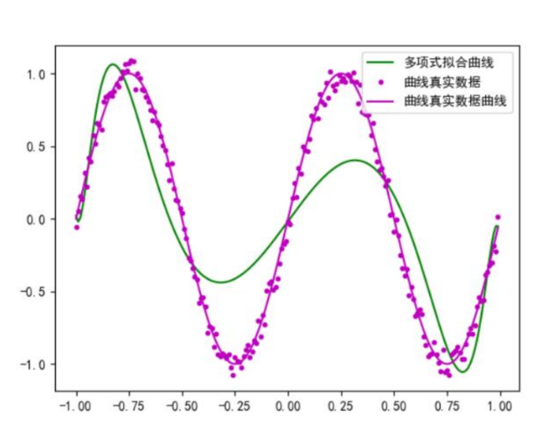
```
def loss(xs,ys,w ):
    error = 0.0
    for i in range(len(x)):
        y1 = 0.0
        for k in range(len(w)):
            y1 += w[k] * x[i] ** k
        error += (y[i] - y1) ** 2
return error/len(xs)
```

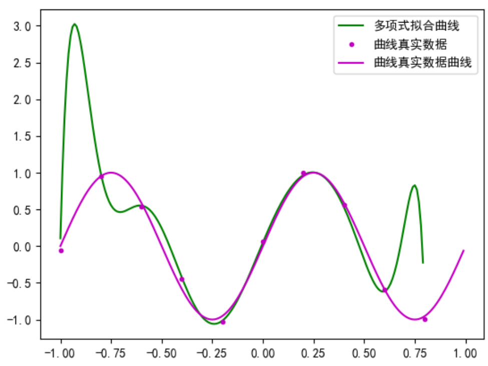
3、 计算解析解、梯度下降法优化算法、共轭梯度优化算法见代码附录。

# 四、实验结果与分析

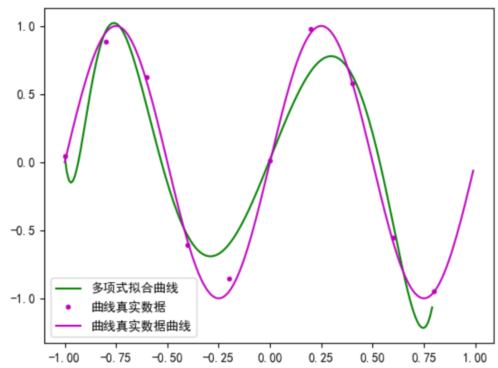
## 4.1 解析解结果

如下图所示，左边为未加入拟合惩罚项的拟合结果，右边是加入惩罚项的拟合结果。次数分别为 5、10 次。可以看出，加入惩罚项之后的拟合结果对于高次有抑制作用，不容易造成过拟合。但是在次数过低的时候，不加入惩罚项的效果比加入惩罚项的效果要好很多，可以推断出虽然加入惩罚性可以防止过拟合，但是次数过低的情况下也会造成拟合不充分等问题。

<table>
<tr>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>5 次多项式，未加入惩罚项</td>
<td>5 次多项式，加入惩罚项</td>
</tr>
</table>




15 次多项式，未加入惩罚项

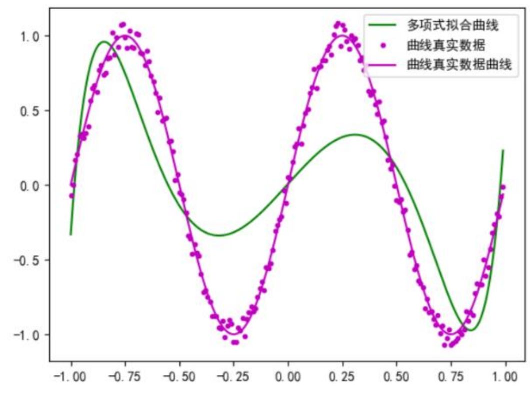



15 次多项式，加入惩罚项

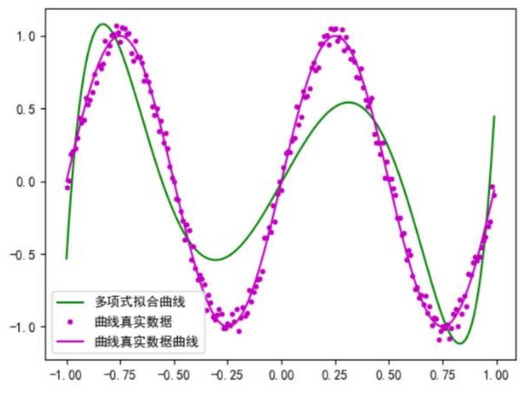
## 4.2 优化方法:

### 4.2.1 梯度下降法

下图左边为迭代 1000 次的结果，右边是迭代 80000 次的结果。可以看到，在学习率一定的情况下，迭代的次数越多，拟合的结果也就越好。



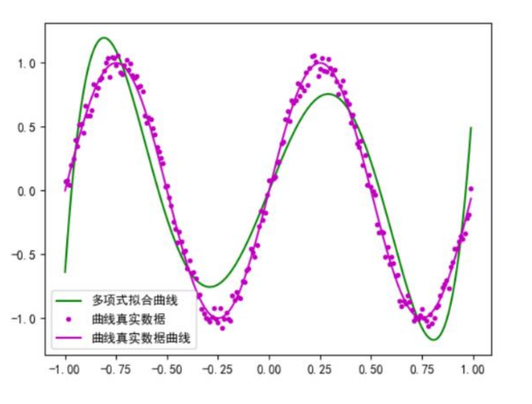
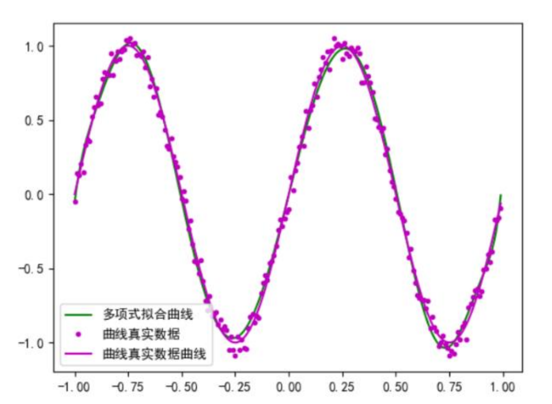

迭代 1000 次




迭代 80000 次

### 4.2.2 共轭梯度法

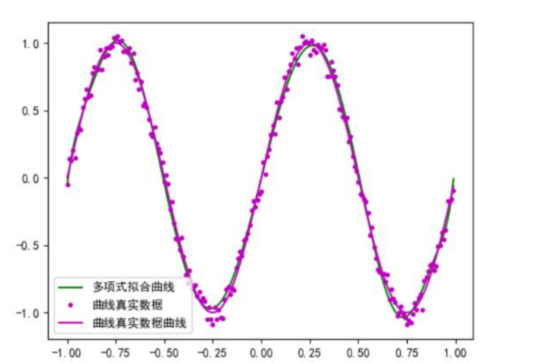
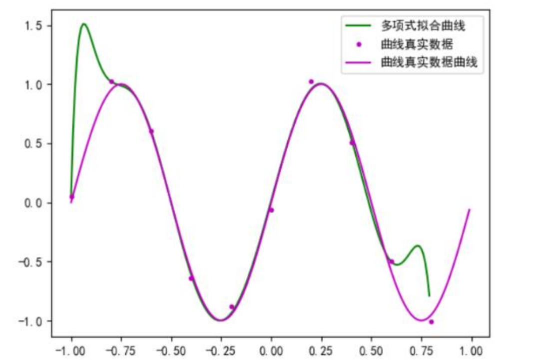
应用共轭梯度法的好处之一就是不需要进行几万次的迭代，如果搜索空间的相互正交的方向向量有 m 个，那么至多进行 m 次迭代就可以得到最好的结果。我们可以看到下图，一共有 200 组数据，随着多项式的次数越高，拟合的结果也就越好。

| | |
|---|---|
|  |  |
| 5 次多项式 | 20 次多项式 |

## 1.3 结果分析:

### 1.3.1 数据少而导致过拟合

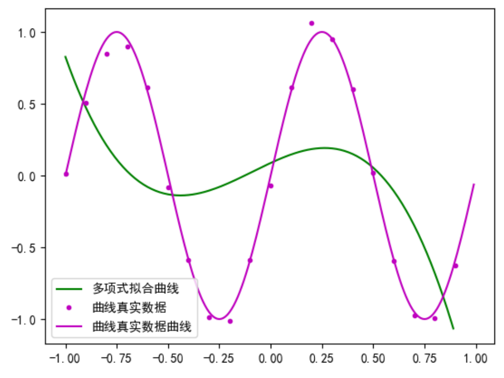
当数据过少而多项式的次数很高的时候，容易出现过拟合的现象，如下图所示，左边有两百组数据，即使是 20 次的多项式也可以很好的拟合。但是右边的图只有 10 组数据，当拟合 20 次多项式的时候，虽然数据也都在多项式拟合曲线上，但是出现了过拟合的现象，导致预测的不准确，离正弦函数较远。

| | |
|---|---|
|  |  |
| 200 组数据 20 次多项式 | 10 组数据 20 次多项式 |

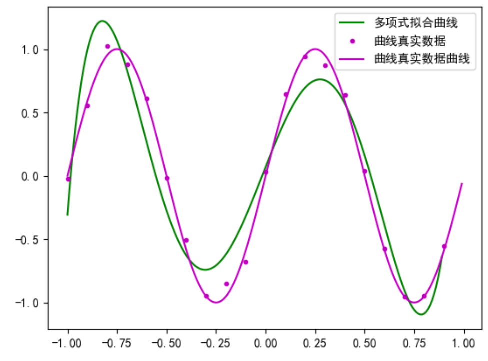
### 1.3.2 不同次数比较

如下图所示，多项式的次数依次为 3、5、10、15。我们发现多项式的次数太低或者太高都没法很好的拟合数据。次数太低容易造成误差，次数太高容易过拟合。比较好的次数是 10 次，此时拟合的函数与正弦函数重合度较高。

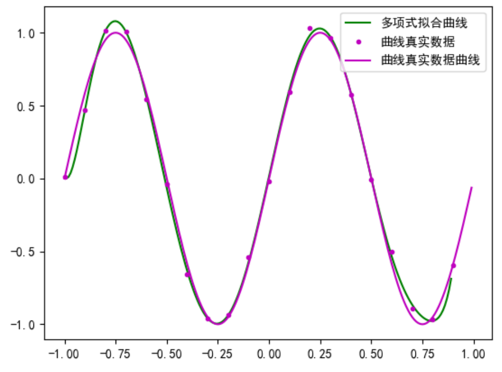
| | |
|---|---|
| | |



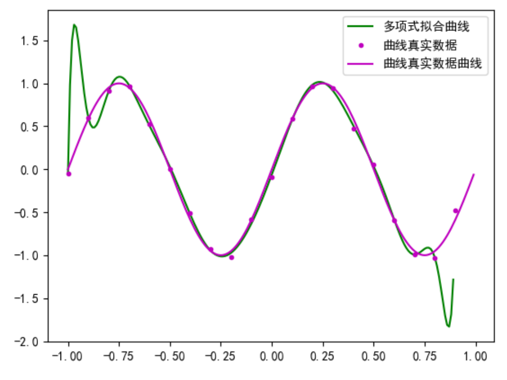

三次多项式




五次多项式




十次多项式




十五次多项式

# 五、结论

1、无正则项的解析解来拟合时，若阶数过大，会发生了过拟合现象。
2、增加正则项是克服过拟合现象的一个有效手段。
3、增加样本数量也是克服过拟合现象的一个有效手段。
4、梯度下降法求解的迭代次数往往很大（大于 10000）。

5、共轭梯度法求解的迭代次数一定≤解空间维数。

6、当样本数量相同，阶数过少容易导致拟合不充分，阶数过大容易导致过拟合

# 六、参考文献

# 七、附录：源代码（带注释）

## 7.1 计算解析解

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

```
import math
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
x=np.arange(-1,1,0.01)
ys=[np.sin(2*np.pi*i) for i in x]
y=[]
#加入噪声
for i in range(len(ys)):
    z=np.random.randint(low=-10,high=10)/100
    y.append(ys[i]+z)
#损失函数，无惩罚项
def loss(xs,ys,w ):
    error = 0.0
    for i in range(len(x)):
        y1 = 0.0
        for k in range(len(w)):
            y1 += w[k] * x[i] ** k
        error += (y[i] - y1) ** 2
    return error/len(xs)
def lossPunish(xs,ys,w,lamda=2):
    error = 0.0
    for i in range(len(x)):
        y1 = 0.0
        for k in range(len(w)):
            y1 += w[k] * x[i] ** k
        error += (y[i] - y1) ** 2
    for j in range(len(w)):
        error+=lamda*w[j]**2
    return error/len(xs)
def straight(x,y,deg):
    '''
    :param x:
    :param y: 因变量 自变量个数*1
    :return: w 新的矩阵参数
    '''
    y=np.mat(y)
    x = np.vander(x, deg + 1,increasing=True)
    a=np.dot(x.T ,x)
    a=np.matrix(a)
    a=a.I
```

```
    b=np.dot(a,x.T)
    w=np.dot(b,y.T)
    a=[0.0]*(deg+1)
    for i in range(deg+1):
        a[i]=w[i][0].item()
    print(a)
    return a
def straightPunish(x,y,deg,lamda=0.2):
    '''
    :param x:
    :param y: 因变量 自变量个数*1
    :return: w 新的矩阵参数
    '''
    y=np.mat(y)
    x = np.vander(x, deg + 1,increasing=True)
    a=np.dot(x.T ,x)
    l=lamda*np.eye(deg+1)#惩罚项
    a=a+l
    a=np.matrix(a)
    a=a.I
    b=np.dot(a,x.T)
    w=np.dot(b,y.T)
    a = [0.0] * (deg + 1)
    for i in range(deg + 1):
        a[i] = w[i][0].item()
    print(a)
    return a
def draw(x,y,w,order):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    fit_xs, fit_ys = np.arange(min(x), max(x) , 0.01), []
    for i in range(0, len(x)):
        ys = 0.0
        for k in range(0, order + 1):
            ys += (w[k] * fit_xs[i] ** k)
        fit_ys.append(ys)
    ax.plot(fit_xs, fit_ys, color='g', linestyle='-', marker='', label='多项式拟合曲线')
    ax.plot(x, y, color='m', linestyle='', marker='.', label='曲线真实数据')
    x1 = np.arange(-1, 1, 0.01)
```

```
    ax.plot(x1, np.sin(2*np.pi*x1), color='m', linestyle='-', marker='', label='曲线真实数据曲线')

    # plt.title(s='最小二乘法拟合多项式 N={}的函数曲线 f(x)'.format(order))

    plt.legend()
    plt.show()
deg = 50

# w1 = straight(x, y, deg)   # 解析解求参数

# print(loss(x,y,w1))
# draw(x,y,w1,deg)
w2 = straightPunish(x, y, deg)
print(lossPunish(x,y,w2))
draw(x,y,w2,deg)
```

7.2 梯度下降法

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
'''

最小二乘法多项式拟合曲线

'''

# 生成带有噪点的待拟合的数据集合

def init_fx_data():

    # 待拟合曲线 f(x) = sin2x * [(x^2 - 1)^3 + 0.5]

    xs = np.arange(-1, 1, 0.01)   # 200 个点

    ys = [((x ** 2 - 1) ** 3 + 0.5) * np.sin(x * 2) for x in xs]
    ys1 = []
    for i in range(len(ys)):

        z = np.random.randint(low=-10, high=10) / 100   # 加入噪点

        ys1.append(ys[i] + z)
    return xs, ys1

# 计算最小二乘法当前的误差

def last_square_current_loss(xs, ys, A):
    error = 0.0
    for i in range(len(xs)):
        y1 = 0.0
```

```
            for k in range(len(A)):
                y1 += A[k] * xs[i] ** k
            error += (ys[i] - y1) ** 2
        return error
```

# 迭代解法：最小二乘法+梯度下降法

```
def last_square_fit_curve_Gradient(xs, ys, order, iternum=1000, learn_rate=0.001):
    A = [0.0] * (order + 1)
    for r in range(iternum + 1):
        for k in range(len(A)):
            gradient = 0.0
            for i in range(len(xs)):
                y1 = 0.0
                for j in range(len(A)):
                    y1 += A[j] * xs[i]**j

                gradient += -2 * (ys[i] - y1) * xs[i]**k   # 计算 A[k]的梯度

            A[k] = A[k] - (learn_rate * gradient)   # 更新 A[k]的梯度

        # 检查误差变化

        if r % 100 == 0:
            error = last_square_current_loss(xs=xs, ys=ys, A=A)

            print('最小二乘法+梯度下降法：第{}次迭代，误差下降为：{}'.format(r, error))

    return A
```

# 迭代解法：最小二乘法+正则惩罚项+梯度下降法

```
def last_square_fit_punish_curve_Gradient(xs, ys, order, iternum=1000,
learn_rate=0.001,lamda=0.1):
    A = [0.0] * (order + 1)
    for r in range(iternum + 1):
        for k in range(len(A)):
            gradient = 0.0
            for i in range(len(xs)):
                y1 = 0.0
                for j in range(len(A)):
                    y1 += A[j] * xs[i]**j

                gradient += -2 * (ys[i] - y1) * xs[i]**k   # 计算 A[k]的梯度

            gradient+=lamda*A[k]

            A[k] = A[k] - (learn_rate * gradient)   # 更新 A[k]的梯度
```

```
        # 检查误差变化
        if r % 100 == 0:
            error = last_square_current_loss(xs=xs, ys=ys, A=A)
            print('最小二乘法+梯度下降法：第{}次迭代，误差下降为：{}'.format(r, error))
    return A

# 可视化多项式曲线拟合结果
def draw_fit_curve(xs, ys, A, order):

    fig = plt.figure()
    ax = fig.add_subplot(111)
    fit_xs, fit_ys = np.arange(min(xs) * 0.8, max(xs) * 0.8, 0.01), []
    for i in range(0, len(fit_xs)):
        y = 0.0
        for k in range(0, order + 1):
            y += (A[k] * fit_xs[i] ** k)
        fit_ys.append(y)
    ax.plot(fit_xs, fit_ys, color='g', linestyle='-', marker='', label='多项式拟合曲线')

    ax.plot(xs, ys, color='m', linestyle='', marker='.', label='曲线真实数据')
    x=np.arange(-1,1,0.01)
    ax.plot(x, ((x ** 2 - 1) ** 3 + 0.5) * np.sin(x * 2), color='m', linestyle='-', marker='', label='曲
线真实数据曲线')

    #plt.title(s='最小二乘法拟合多项式 N={}的函数曲线 f(x)'.format(order))
    plt.legend()
    plt.show()
if __name__ == '__main__':
    order = 10   # 拟合的多项式项数

    xs, ys = init_fx_data()   # 曲线数据
    A = last_square_fit_punish_curve_Gradient(xs=xs, ys=ys, order=order, iternum=500,
learn_rate=0.001,lamda=0.1)
    draw_fit_curve(xs=xs, ys=ys, A=A, order=order)   # 可视化多项式曲线拟合结果
```

7.3 共轭梯度法

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
#最小二乘法

def init_fx_data():
    # 待拟合曲线 f(x) = sin2x * [(x^2 - 1)^3 + 0.5]
    xs = np.arange(-1, 1, 0.01)   # 200 个点
    ys = [((x ** 2 - 1) ** 3 + 0.5) * np.sin(x * 2) for x in xs]
    ys1 = []
    for i in range(len(ys)):
        z = np.random.randint(low=-10, high=10) / 100   # 加入噪点
        ys1.append(ys[i] + z)
    return xs, ys1

x,y=init_fx_data();
degree=5
X=np.vander(x,degree+1,increasing=True)
x=np.mat(x).T
y=np.mat(y).T

def fun(w):
    a=np.dot(X,w)-y
    return np.dot(a.T,a)


def gfun(w):
    a = np.dot(X, w) - y
    return np.dot(X.T,a)


def frcg(x0):
    #用 FR 共轭梯度法求解无约束问题
    #x0 是初始点，fun 和 gfun 分别是目标函数和梯度
    #x,val 分别是近似最优点和最优值，k 是迭代次数
    maxk = 5000
```

```
    rho = 0.6
    sigma = 0.4
    k = 0
    epsilon = 1e-5
    n = np.shape(x0)[0]
    itern = 0
    while k < maxk:
        gk = gfun(x0)
        itern += 1
        itern %= n

        #求 dk

        if itern == 1:
            dk = -gk
        else:
            beta = 1.0*np.dot(gk.T,gk)/np.dot(g0.T,g0)
            dk = -gk + beta[0][0].item()*d0
            gd = np.dot(gk.T,dk)
            if gd >= 0.0:
                dk = -gk

        #判断精度

        if np.linalg.norm(gk) < epsilon:
            break

        #求 ak

        m = 0
        mk = 0
        while m < 20:
            if fun(x0+rho**m*dk) < fun(x0) + sigma*rho**m*np.dot(gk.T,dk):
                mk = m
                break
            m += 1
        x0 += rho**mk*dk
        g0 = gk
        d0 = dk
        k += 1

    return x0,fun(x0),k

def draw_fit_curve(xs, ys, A, order):
    xs=np.array(xs)
    ys=np.array(ys)
    A=np.array(A)
```

```
    fig = plt.figure()
    ax = fig.add_subplot(111)
    fit_xs, fit_ys = np.arange(min(xs) * 0.8, max(xs) * 0.8, 0.01), []
    for i in range(0, len(fit_xs)):
        y = 0.0
        for k in range(0, order + 1):
            y += (A[k] * fit_xs[i] ** k)
        fit_ys.append(y)

    ax.plot(fit_xs, fit_ys, color='g', linestyle='-', marker='', label='多项式拟合曲线')

    ax.plot(xs, ys, color='m', linestyle='', marker='.', label='曲线真实数据')

    x=np.arange(-1,1,0.01)

    ax.plot(x, ((x ** 2 - 1) ** 3 + 0.5) * np.sin(x * 2), color='m', linestyle='-', marker='', label='曲
线真实数据曲线')

    #plt.title(s='最小二乘法拟合多项式 N={}的函数曲线 f(x)'.format(order))

    plt.legend()
    plt.show()

w=np.zeros(degree+1)
data=np.mat(w).T
w,f,k= frcg(data)
print(np.shape(w))
draw_fit_curve(x,y,w,degree)
```