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：1190201115
姓名： 陈宇豪

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）

1. 数据生成算法

利用随机函数产生在 0 到 1 之间的 x，再利用 sin(2Πx）和噪声产生 y。

此外，X 是按照阶数产生的如下图的矩阵：

```
function [data_x,data_y,X] = create_data(size,order,average,sigma )
%   输入数据集大小size，阶数order,噪声均值average和标准差sigma
%
%   data_x为在0-1之间的随机数，1*size大小的矩阵。data_y为size*1的矩阵。
%   产生以average为均值，sigma为标准差的噪声noise
%
data_x=rand(1,size);
sin_func=sin(2*pi*data_x);
noise=average+sigma.*randn(1,size);
data_y=sin_func+noise;
data_y=reshape(data_y,size,1);
X=ones(size,order+1);
for i=1:order
  X(:,i+1)=(data_x.^i);
end
end
```

2. 最小二乘法求解析解（无正则项）

已知误差函数如下：

$$\mathbf{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\{y(xi, w) - ti\}^2$$

其中 ti 为数据集中实际取到的第 i 个 y 值，y(xi,w)为通过拟合得到的函数解得的 y 值。

$$y(x, w) = w0 + w1 * x + ... + wm * x^m = \sum_{i=0}^{m} wi * x^i$$

将误差函数写为矩阵形式：

$$E(w) = \frac{1}{2}(Xw - T)'(Xw - T)$$

令上式关于 w 求导，得：

$$\frac{\partial E}{\partial w} = X'Xw - X'T$$

使上式为 0，解得

$$w = ((X'X)^{-1}) * X'Y$$

```
function [ ] = func1_paint( x,X,data_y )
w=pinv(X'*X)*X'*data_y;
```

3. 最小二乘法求解析解（含正则项）

为防止过拟合，在计算误差时加入惩罚项，得到如下形式

$$\mathbf{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\{y(xi, w) - ti\}^2 + \frac{\lambda}{2}|w|^2$$

同样，写为矩阵形式：

$$E(w) = \frac{1}{2}(Xw - T)'(Xw - T) + \frac{\lambda}{2}w'w$$

关于 w 求导，令导数为 0，解得 w 为

$$w = (X'X + \lambda I)^{-1} X'T$$

```
function [ ] = punish_paint( x,X,data_y,lamda )
[~,col]=size(X);
w=pinv(X'*X+lamda*eye(col))*X'*data_y;
```

4. 梯度下降法求优化解

显然代价函数是关于参数 w 得函数，关于每一个参数 w 求对应偏导，可以得到应该如果改变参数 w 的方向才能使代价函数下降。如图

$$
\begin{aligned}
&\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
&\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\
&\theta_0 := \text{temp0} \\
&\theta_1 := \text{temp1}
\end{aligned}
$$

其中，α 表示每一次移动的跨度。如果对 θ0 的偏导为正数，代表朝着 θ0 的轴移动将会使代价函数增大，而我们得目的是时代价函数取到最小，所以前面带着的是负号。

α 的取值如果过小，那么每次移动就会是很小的一步，那么收敛到合适参数的过程将变得漫长。如果 α 取值过大，就有可能直接越过参数最低点。最坏的情况下，这会使代价 J 变得越来越大。

故可以设置一个修正 α 的方法，即当判断出在一次移动后代价函数反而增大了，那么就不进行那次移动，并且将 α 变小为原来的一半。

迭代公式为：

$$w = w - \alpha * \frac{\partial \widetilde{E}}{\partial w}$$

E 为带正则项的代价函数。

```
for i=1:order+1
    tempw(i)=w(i)-train_rate*(sum((hypo_y-data_y).*(data_x.^(i-1))')+lamda*w(i));
end
new_loss=cal_loss(order,tempw,data_x,data_y);
```

5. 共轭梯度法求最优解

梯度下降的法中，用一步步迭代的方法找 E 关于 w 偏导的极小值点。故可以先令：

$$\frac{\partial \widetilde{\mathrm{E}}}{\partial \mathrm{w}} = X'Xw - X'Y + \lambda w = 0$$

记 A=X’X+λ，b=X’Y,即转化为求解 Aw=b。

初始化令 w 为零向量，r=b, p=b.

对于第 k 步的残差，$r_k = b - Ax_k$。要根据残差值找到下一步的搜索方向 p_k，初始为 b。之后利用公式

$$r_{k+1} = r_k - \alpha_k A p_k$$

$$\alpha_k = \frac{r_k' r_k}{p_k' A p_k}$$

求得第 k+1 步的残差。

根据 $p_{k+1} = r_{k+1} + b_{K+1} * p_k$ 求得下一步的搜索方向，其中 $b_k = \frac{r_{k+1}' r_{k+1}}{r_k' r_k}$

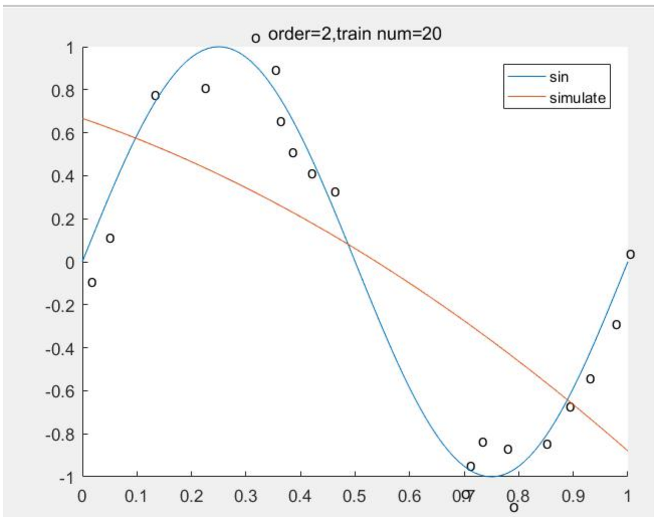
反复迭代直到 r_k 满足精度。

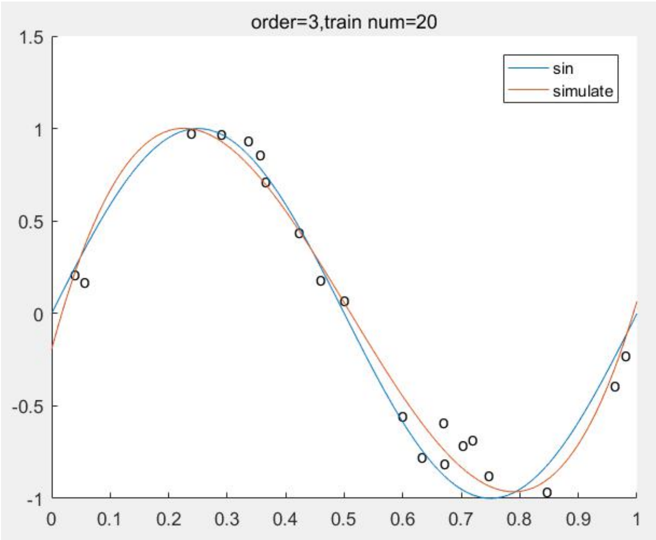
```
while true
    alpha = (r0'* r0)/(p'*A * p);
        w = w + alpha * p;
        r = r0-alpha*A*p;
        if r0'*r0 < delta;
            break;
        end
        beta = (r'* r) / (r0'* r0);
        p = r + beta * p;
        r0 = r;
        k =k+1;
```

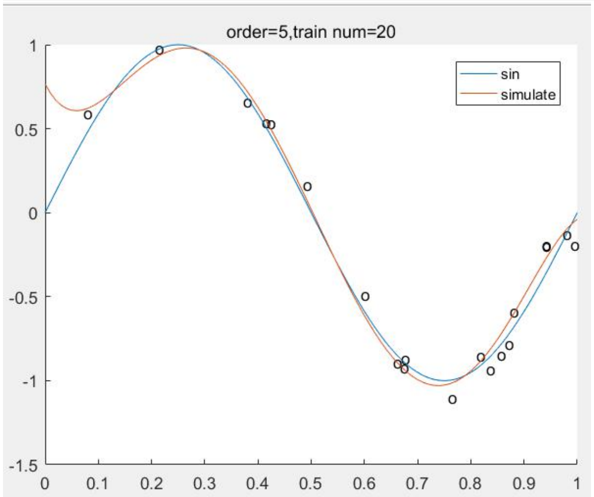
四、实验结果与分析

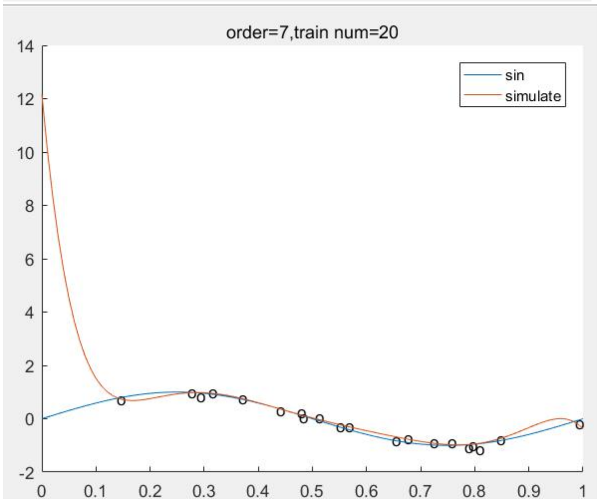
4.1 不带惩罚项的解析解

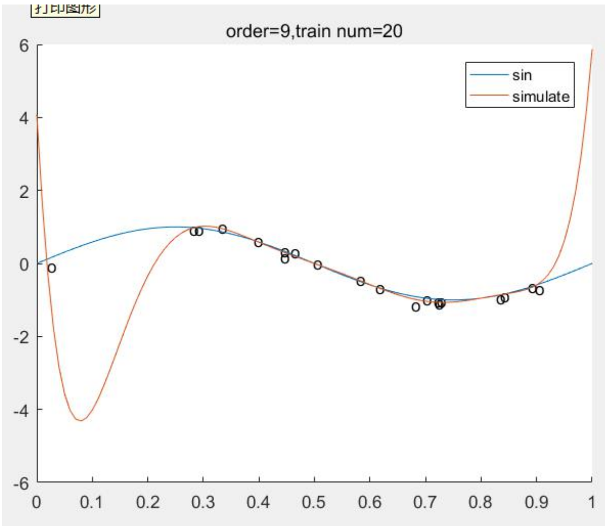
设定训练样本数为 20，测试当阶数为 2，3，5，7，9，19 时的函数拟合情况。

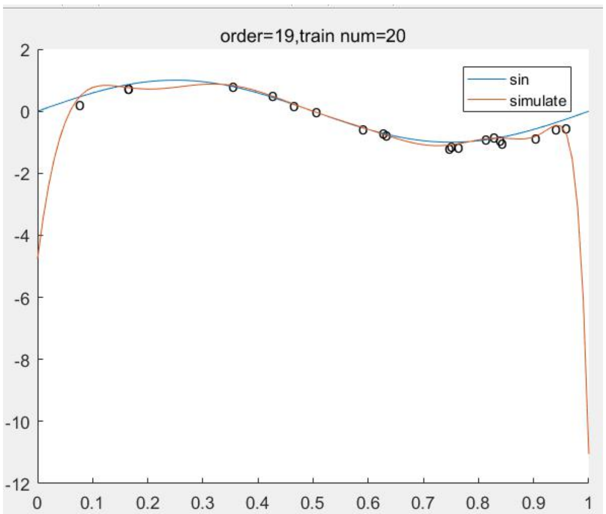



order=3,train num=20
sin
simulate
1.5
1
0.5
0
-0.5
-1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


order=5,train num=20
sin
simulate
1
0.5
0
-0.5
-1
-1.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

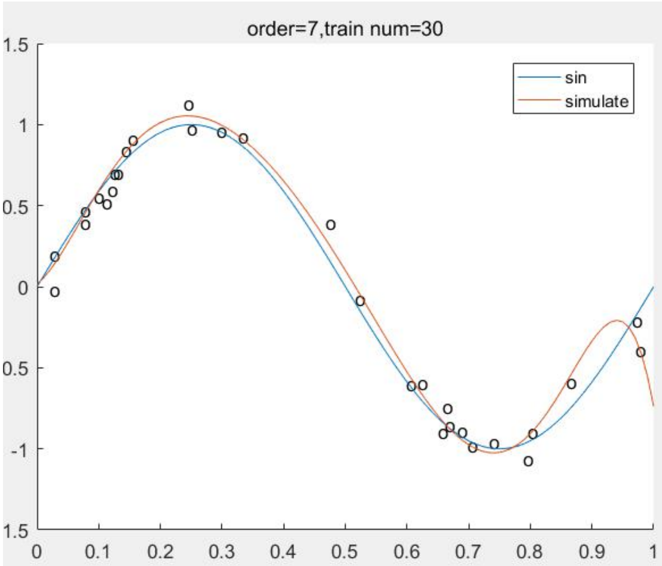

order=7,train num=20
sin
simulate
14
12
10
8
6
4
2
0
-2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

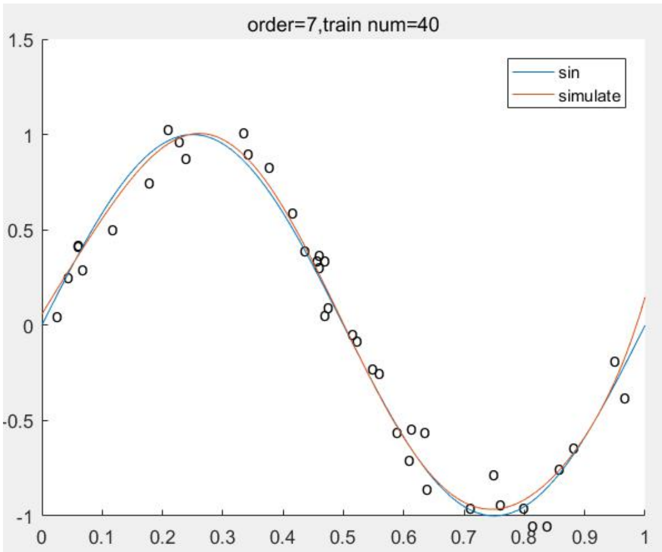




可以发现，当阶数为 3 时已经能够较好地拟合 sin 函数了，但是当阶数不断提高时，函数为了更好地拟合所有点，表现出了一定变形，在阶数达到 7 以上时，虽然模拟函数很好地经过了各点，但却已经和要模拟的函数 sin(2Πx)相差甚远了。这就是过拟合现象。在阶数很大时，函数很好地模拟了所有点，包括噪声，当阶数达到 19 时，对于 20 个样本的方程恰好有唯一解，最终产生的就是非常扭曲的函数图像。

要减小过拟合现象的影响，除了接下来要做的增加惩罚项以外，还可以通过增加样本数的方式，上图中当样本数为 20，阶数为 7 时，已经有了明显的过拟合现象。以下做两组测试实验



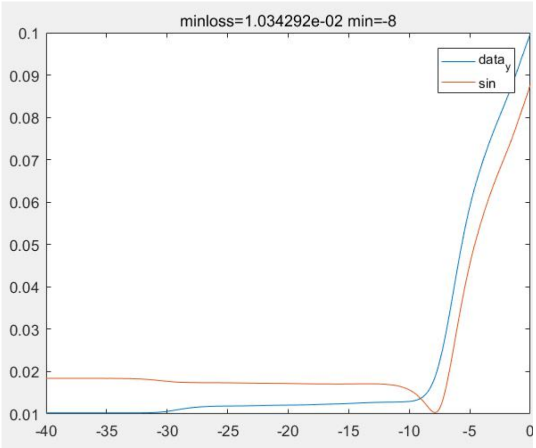


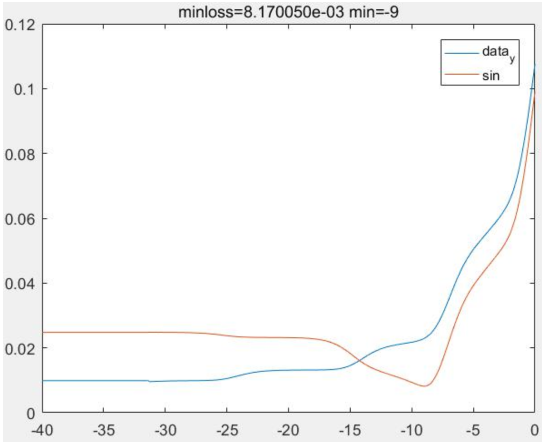
可以发现，当阶数保持为 7 时，适当增加样本数，的确有助于减小过拟合现象。

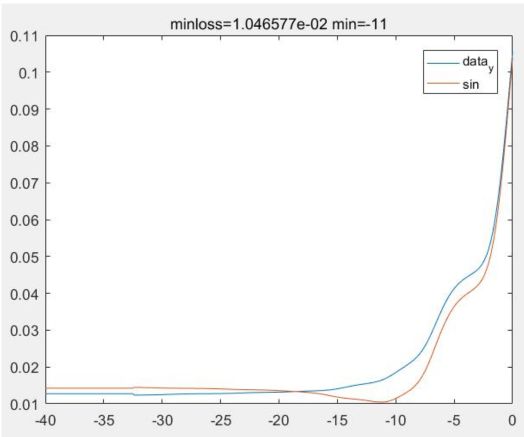
4.2 带惩罚项的解析解

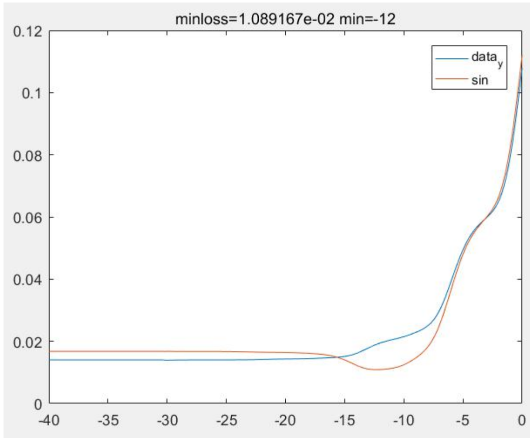
首先要确定合适的λ值。设计算法如下：

首先让 lnλ在-40 到 0 之间以 0.1 的间距分布，得到一个一位数组。对于数组中的每一个值，求得对应的λ后，带入到带惩罚项的解析解公式中，求得系数向量 w。分别计算拟合函数和 sin(2Πx)和数据点的代价 J。最终得到以 lnλ为横轴，J 为纵轴的函数图像如下：



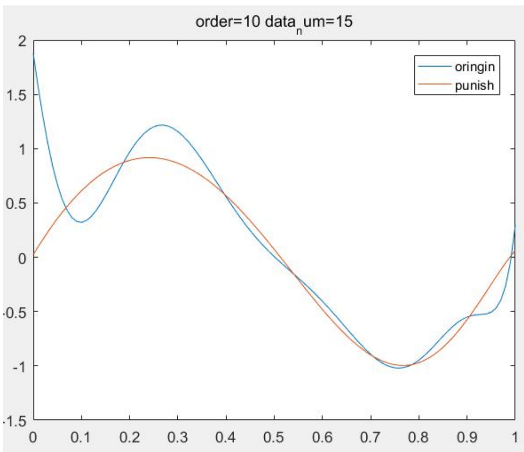


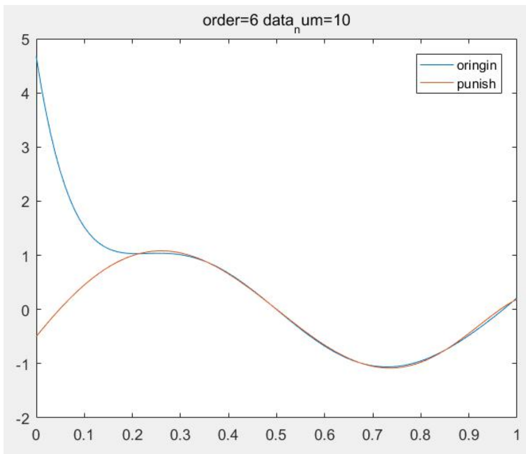


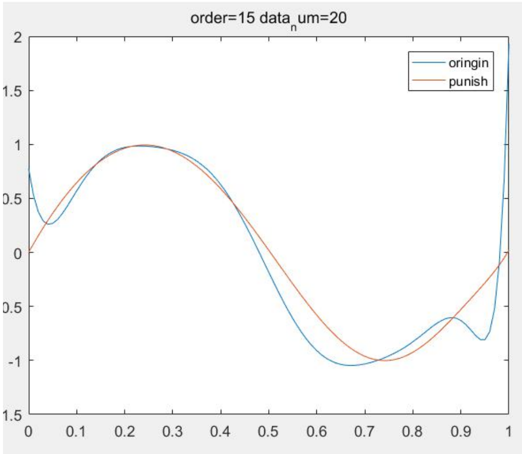


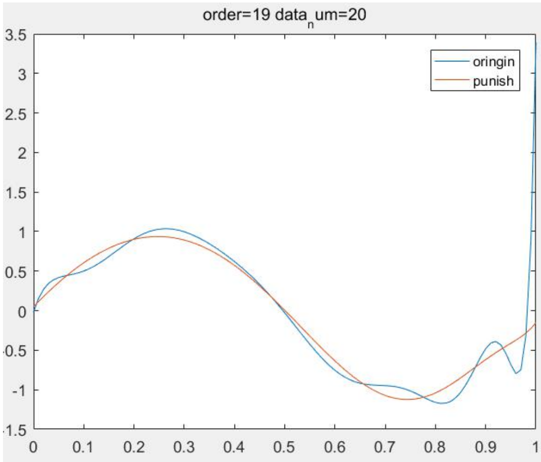
以上数据均取 15 个数据点进行 10 阶拟合，可以观察到λ在 e^-12 阶到 e^-8 阶之间时，拟合曲线和原函数 sin(2Πx)的误差有一定的减小，在该实验条件下，可以认为一般λ取值在 eˆ-15˜eˆ-5 之间时效果较好。

未加惩罚项（origin)和加惩罚项（punish)的拟合函数图像对比：



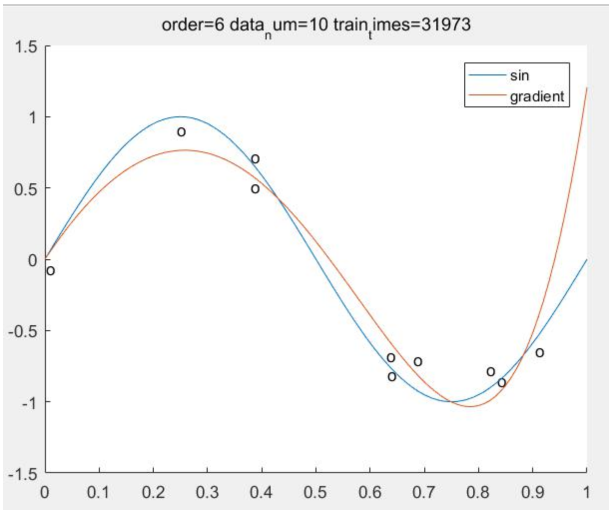


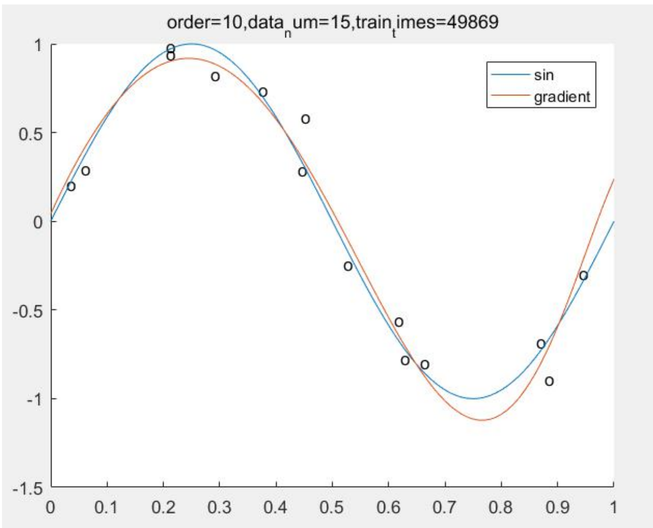


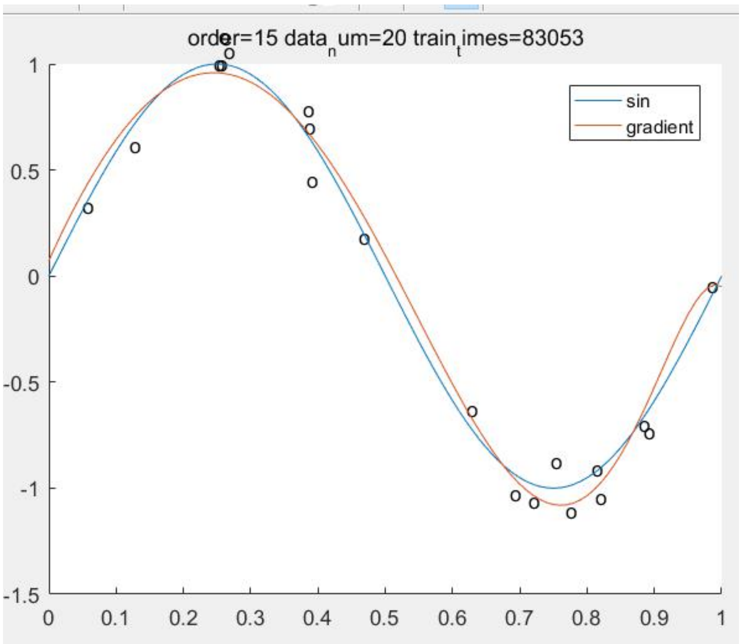


4.3 梯度下降法得优化解

梯度下降法中，我设置的初始学习率为 0.02，当一次下降运算导致代价增大时，就将学习率变为原来的二分之一。设置的停止条件为，两次下降对代价的减小值小于 10^-6。

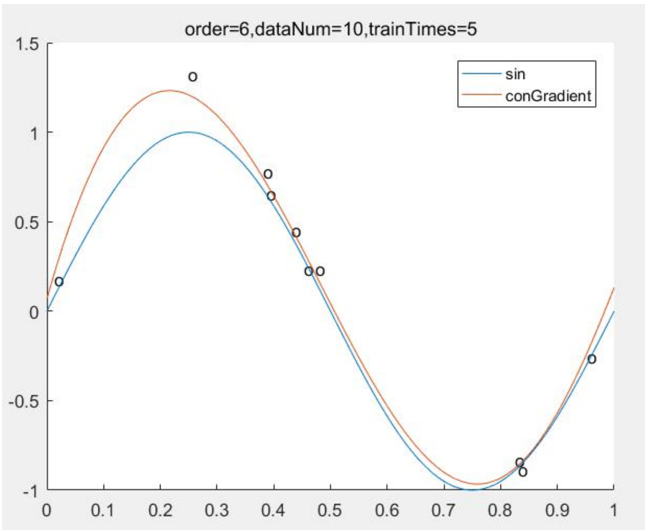


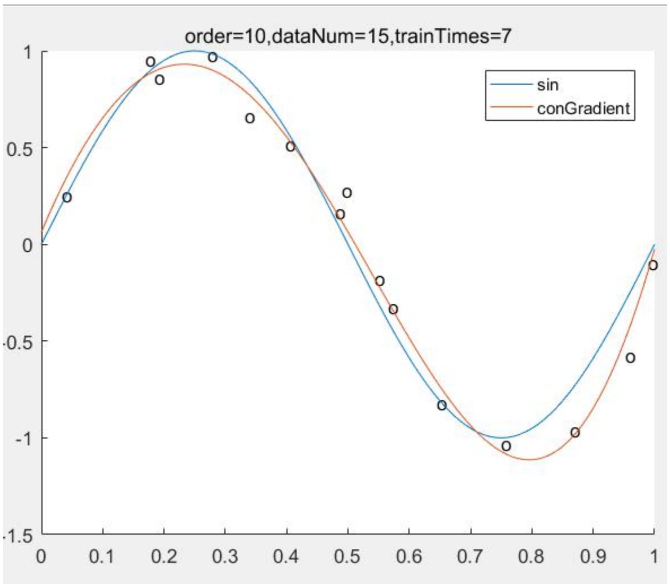


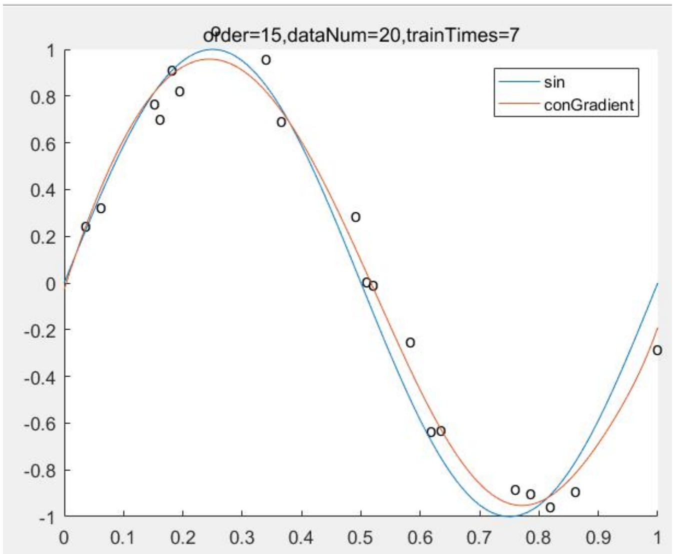


4.4 共轭梯度法得优化解

设置精度为 10^-6，λ 为 e^-10，实验结果如下：



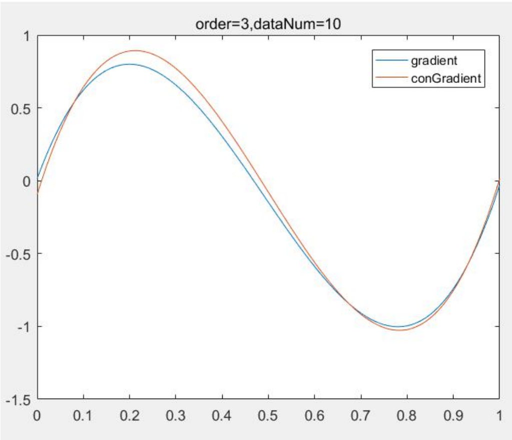


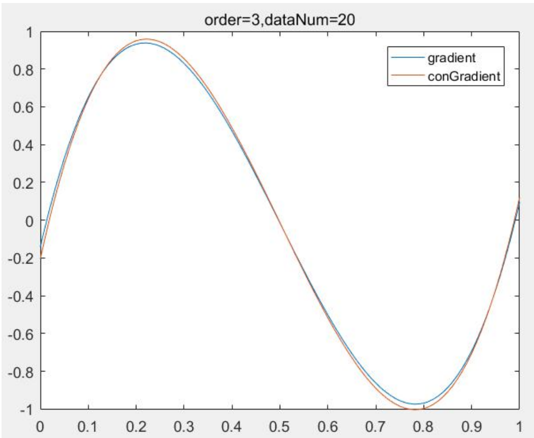


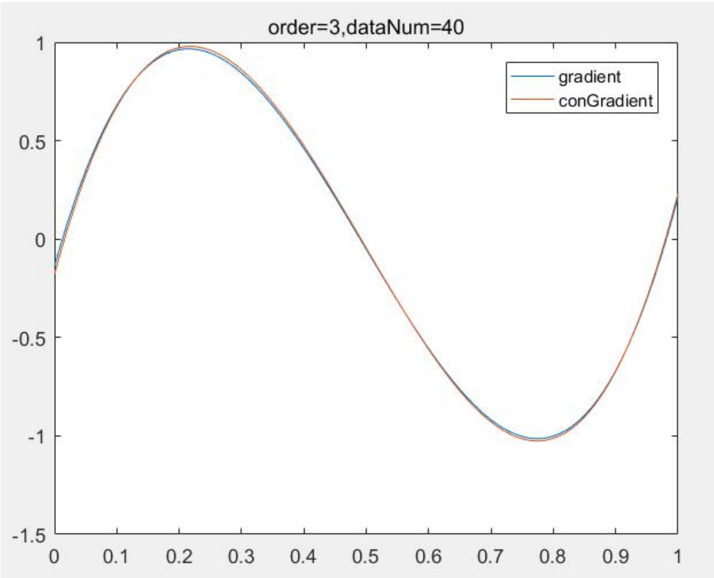
4.5 阶数与样本数对梯度下降法与共轭梯度法的影响

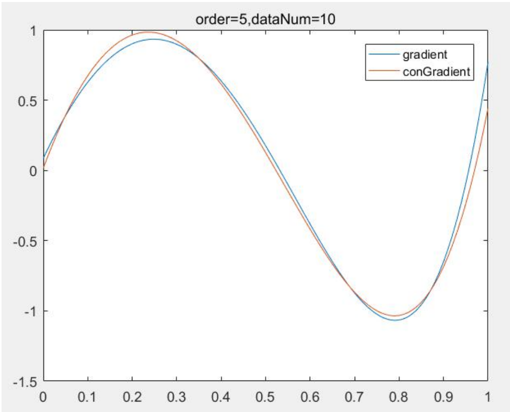
阶数	样本数	梯度下降法迭代次数	共轭梯度法迭代次数
3	10	123303	4
3	20	74316	4
3	40	52212	4
5	10	33096	5

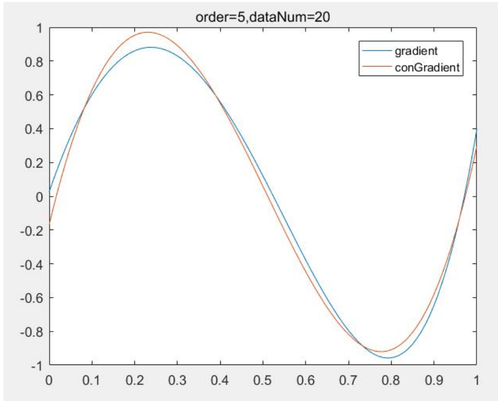
5	20	102968	5
5	40	133181	5
9	10	70113	5
9	20	71941	5
9	40	36572	7

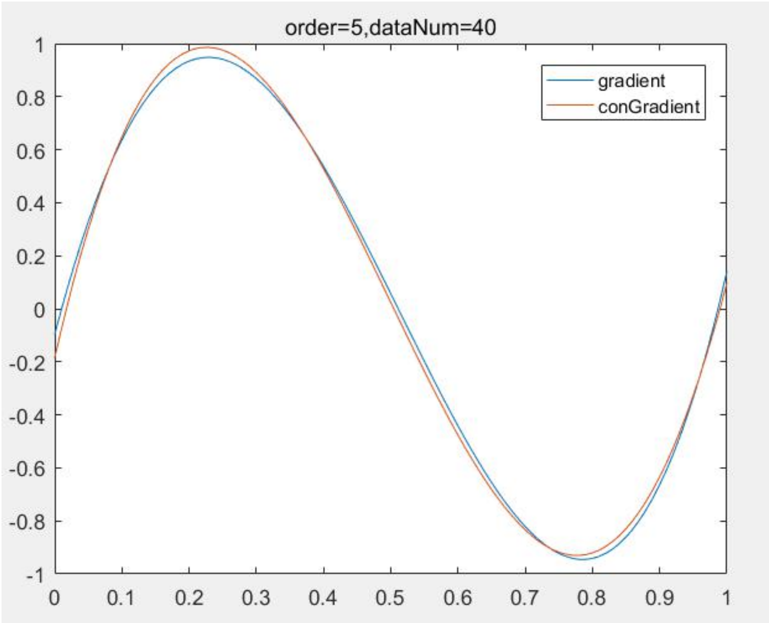


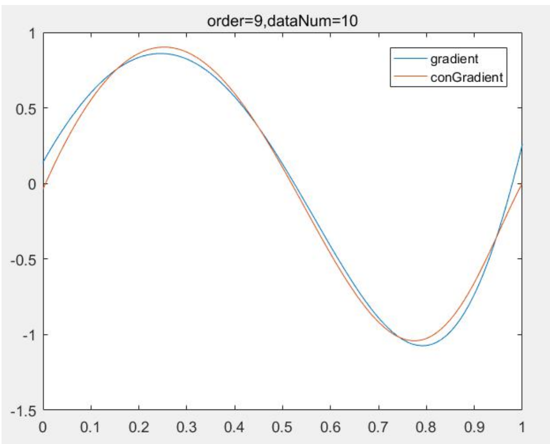


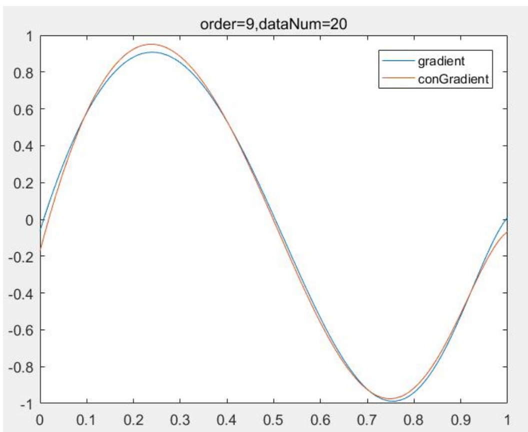


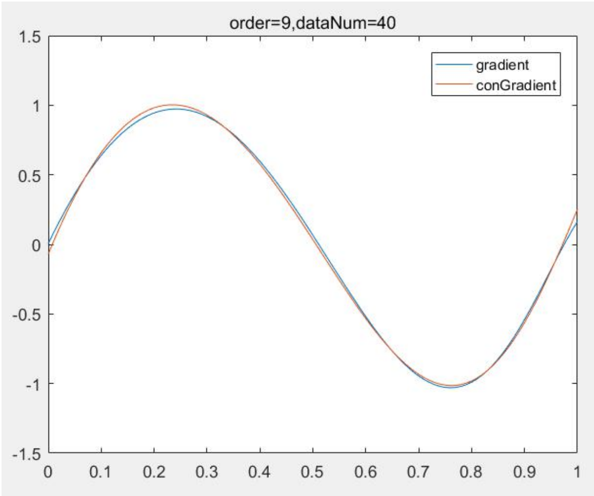

order=5,dataNum=10
gradient
conGradient


order=5,dataNum=20
gradient
conGradient


order=5,dataNum=40
gradient
conGradient


order=9,dataNum=10
gradient
conGradient

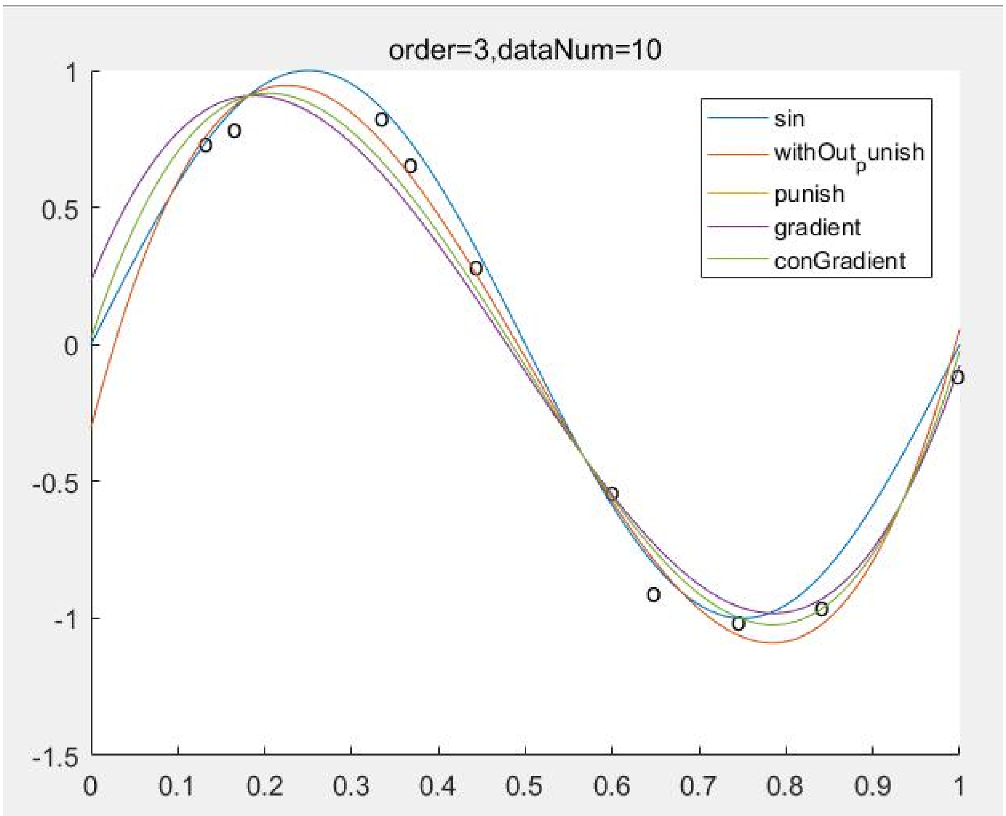

order=9,dataNum=20
gradient
conGradient



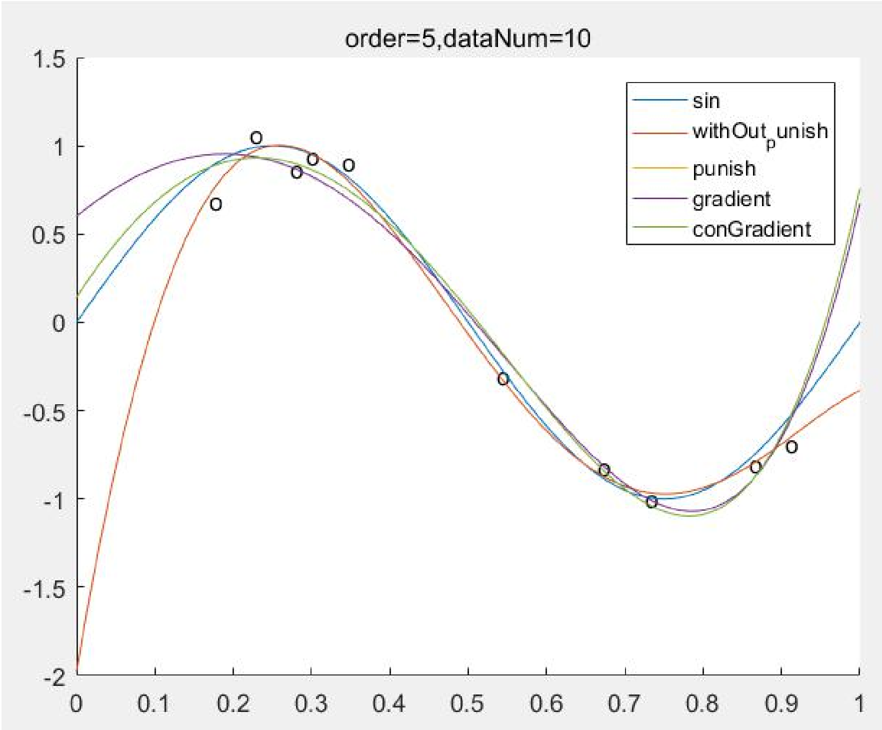
梯度下降法的迭代次数在 30000 到 140000 之间，共轭梯度法迭代次数在 3 到 7 之间。两种方法拟合结果相似，共轭梯度法速度要优于梯度下降法，准确度要稍优于梯度下降法。

4.6 四种拟合方法对比

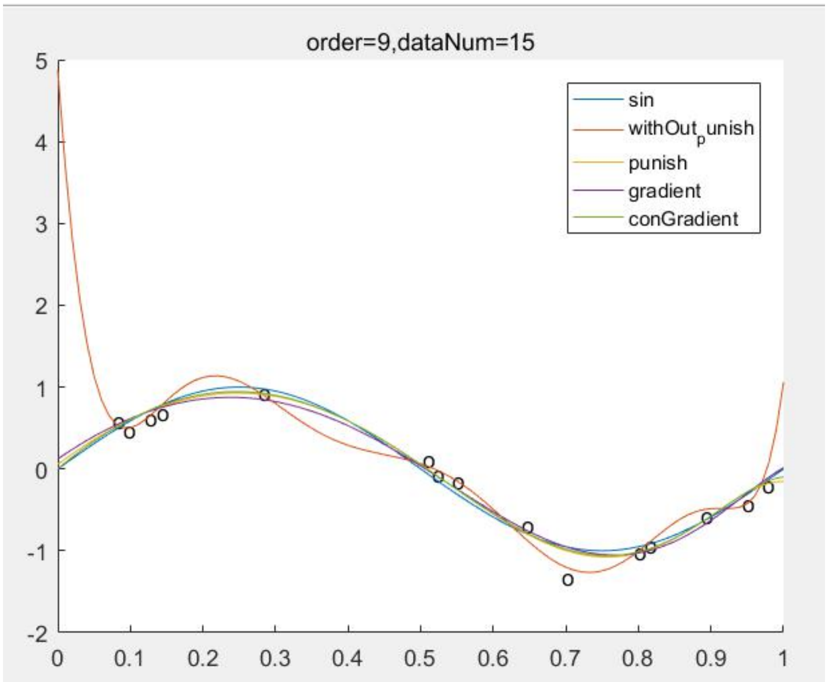
三阶，样本数为 10，精度为 10^-6, λ 取 e^-9。拟合效果均较好



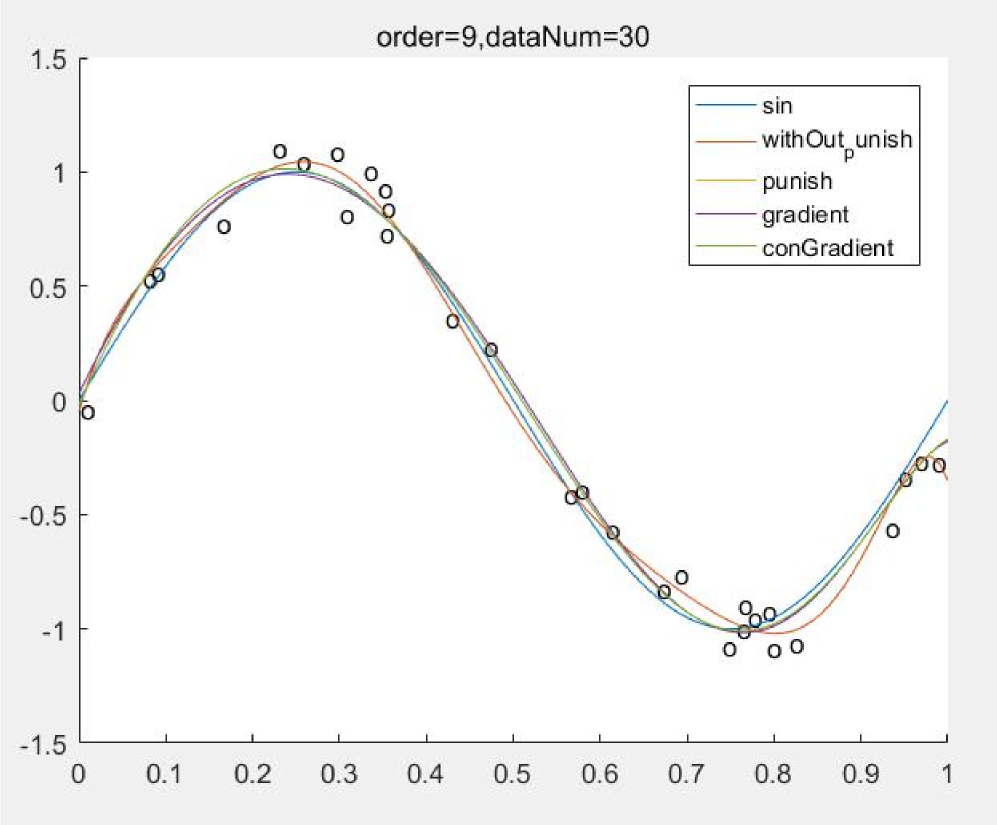
阶数为 5，样本数为 10，其他参数不变。无惩罚项的解析解图像拟合程度较差



9 阶，样本数为 15，无惩罚项的解析解的拟合图像出现了明显的过拟合现象，其他方法表现均较好



阶数为 9，样本数为 30，拟合程度均较好。



五、结论

在对正弦函数的多项式拟合中，多项式的次数越高，拟合能力越强，如果不加正则项的话，高次多项式会将噪声也一并拟合，进而产生过拟合现象。为了减弱过拟合现象，除了增加正则项以外，还可以适当扩大数据集。

对于惩罚系数的选取，不同的拟合参数有不同的可能，通过比较在取不同 λ 值时拟合曲线和原函数曲线的代价值，可以得到一个较为合适的取值区间。

在使用梯度下降法时，如果梯度下降步长设置的比较大，那么在最坏的情况下，取到的参数会让代价越来越大。所以在适当的情况下要让函数自己调整步长。如果步长太小，则训练次数会太多。

梯度下降相比共轭梯度收敛速度很慢，迭代次数很大，当一次迭代产生的效果很小时（如小于 10^-6），则可以停止迭代。共轭梯度法的迭代次数相比之下小得多，效果与梯度下降法也十分接近。故对于复杂度较高的训练集，共轭梯度法比梯度下降法要合适得多。

六、参考文献

梯度下降算法结束条件：
https://blog.csdn.net/Amigo_1997/article/details/84110835

吴恩达机器学习笔记：
https://blog.csdn.net/qq_45832325/article/details/120535731

七、附录：源代码（带注释）

源代码地址：
https://github.com/1190201115/HIT-machineLearningLab/tree/main/lab1

主函数

```
%%主函数部分，各个函数的汇总。解除掉对应注释即可运行

%%x:自变量范围，num:数据集数量，order:阶数，average:噪声平均值,sigma：噪声标准差
%%lamda:惩罚项系数
x=0:0.01:1;
num=15;
order=6;
average=0;
sigma=0.1;
lamda=exp(-9);

%%数据产生函数
%%output:
%%data_x:产生的 x 轴坐标向量，大小为 1*num
%%data_y:产生的 y 轴坐标向量，大小为 num*1
%%X:以[1，xi,xi^2,...,xi^order]为行，共 num 列的矩阵，i 的范围为 1~num。大小为
num*(order+1)
[data_x,data_y,X]=create_data(num,order,average,sigma);

%%寻找合适的 lamda（惩罚系数）
%%find_lamda(X,data_x,data_y,num);

%%绘制 sin(2pix)的函数图像
%{
origin_paint(x,data_x,data_y);
hold on;
%}

%%最小二乘法求解析解（无正则项）
%{
func1_paint(x,X,data_y)
hold on;
%}

%%最小二乘法求解析解（有正则项）
%{
punish_paint(x,X,data_y,lamda);
hold on;
```

```
%}

%%梯度下降法
%%output:
%%train_times:当满足精度时，梯度下降的迭代次数
%{
train_times=gradient_paint(x,order,data_x,data_y,lamda);
hold on;
%}

%%共轭梯度法
%%output
%%trainTimes:当满足精度时，共轭梯度的迭代次数
%{
trainTimes=con_gradient(X,lamda,data_y,x);
%}

%%图像说明
%{
legend('sin','withOut_punish','punish','gradient','conGradient');
string1=sprintf('%s%d,%s%d','order=',order,'dataNum=',num);
title(string1);
%}
```

数据产生函数：

```
function [data_x,data_y,X] = create_data(size,order,average,sigma )
%     输入数据集大小 size，阶数 order,噪声均值 average 和标准差 sigma
%
%     data_x 为在 0-1 之间的随机数，1*size 大小的矩阵。data_y 为 size*1 的矩阵。
%     产生以 average 为均值，sigma 为标准差的噪声 noise
%
data_x=rand(1,size);
sin_func=sin(2*pi*data_x);
noise=average+sigma.*randn(1,size);
data_y=sin_func+noise;
data_y=reshape(data_y,size,1);
X=ones(size,order+1);
for i=1:order
   X(:,i+1)=(data_x.^i);
end
end
```

寻找合适的 lamda 值：

```
%%寻找合适的惩罚系数
function [ ] = find_lamda(X,data_x,data_y,num)
siny=sin(2*pi*data_x)';

%%lamda 范围为 exp(-40),到 exp(0)
lnlamda=(-40:0.1:0);
lamda=exp(lnlamda);

%%存储代价值
rms=zeros(401,1);
rms2=zeros(401,1);

[~,col]=size(X);
order=col-1;
minloss=1;
minlamda=0;

for i=1:401
    w=pinv(X'*X+lamda(i)*eye(col))*X'*data_y;
    rms2(i)=((cal_loss(order,w,data_x,siny))^(1/2))/num;
    %%记录代价最小时 lamda 的值，转化为以 e 为底的指数
    if rms2(i)<minloss
        minlamda=round(lnlamda(i));
        minloss=rms2(i);
    end
    rms(i)=((cal_loss(order,w,data_x,data_y))^(1/2))/num;
end
plot(lnlamda,rms);
hold on;
plot(lnlamda,rms2);
legend('data_y','sin');
string1=sprintf('%s%d %s%d','minloss=',minloss,'min=',minlamda);;
title(string1);
end
```

绘制初始函数图像：

```
%%绘制 sin(2pix)的函数图像
function [ ] = origin_paint( x,data_x,data_y )
text(data_x,data_y,'o');
hold on;
y=sin(2*pi*x);
plot(x,y);
```

```
end
```

不含正则项的解析解

```
function [   ] = func1_paint( x,X,data_y )
w=pinv(X'*X)*X'*data_y;
[~,col]=size(X);
paint(w,col-1,x);
end
```

含正则项的解析解

```
function [   ] = punish_paint( x,X,data_y,lamda )
[~,col]=size(X);
w=pinv(X'*X+lamda*eye(col))*X'*data_y;
paint(w,col-1,x);
end
```

梯度下降算法

```
function [ train_times ] = gradient_paint(x,order,data_x,data_y,lamda)
%初始系数设置为全 0
w = zeros(order+1, 1);
%计算当前代价
old_loss=cal_loss(order,w,data_x,data_y);
tempw=w;
train_rate=0.02;
train_times=1000000;%%最大训练次数
for j=1:train_times

    hypo_y=cal_hypoY(order,w,data_x);
    for i=1:order+1
        tempw(i)=w(i)-train_rate*(sum((hypo_y-data_y).*(data_x.^(i-1))')+lamda*w(i));
    end
    new_loss=cal_loss(order,tempw,data_x,data_y);
    %%满足精度时，记录训练次数，退出迭代
    if abs(old_loss-new_loss)<10^-6
        train_times=j;
        break;
    end
    if new_loss<old_loss
        w=tempw;
        old_loss=new_loss;
```

```
        continue;
    end
    if new_loss>=old_loss
        train_rate=train_rate/2;
    end
end
paint(w,order,x);
end
```

共轭梯度算法

```
function [trainTimes ] =con_gradient(X,lamda,data_y,x)
[~,col]=size(X);
A=X'*X+lamda*eye(col);
b=X'*data_y;
w = zeros(col, 1);
r0=b-A*w;
p=r0;
k=0;
delta=10^-6;
while true
    alpha = (r0'* r0)/(p'*A * p);
        w = w + alpha * p;
        r = r0-alpha*A*p;
        %%满足精度时退出
        if r0'*r0 < delta;
            break;
        end
        beta = (r'* r) / (r0'* r0);
        p = r + beta * p;
        r0 = r;
        k =k+1;
end
%%记录训练次数
trainTimes=k;
paint(w,col-1,x);
end
```

计算拟合曲线的 Y 坐标向量

```
function [ hypo_y ] = cal_hypoY(order,w,data_x)
%根据输入的数据集大小，阶数，系数，x 矩阵产生对应的 y 矩阵
[~,num]=size(data_x);
hypo_y=zeros(1,num);
for j=1:num
    for i=1:order+1
        hypo_y(j)=w(i)*(data_x(j).^(i-1))+hypo_y(j);
    end
end
hypo_y=reshape(hypo_y,num,1);
end
```

计算 loss

```
function [ loss ] = cal_loss( order,w,data_x,data_y)
%计算当前代价
hypo_y=cal_hypoY(order,w,data_x);
loss=sum((hypo_y-data_y).^2);
end
```