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：　专业选修

实验题目：　多项式拟合正弦函数

学号：120L030501
姓名：　张明远

一、实验目的

目标：

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

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。

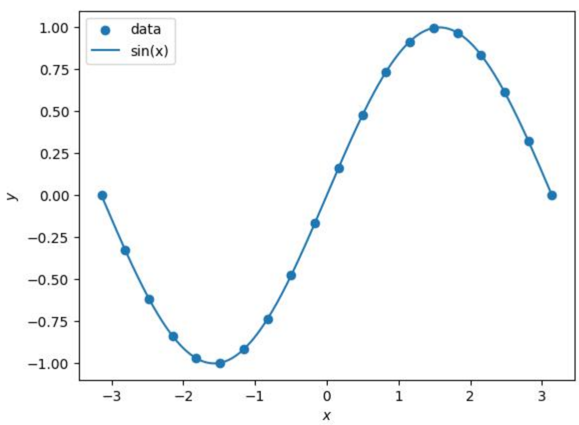
环境：

python3.10

三、设计思想（本程序中的用到的主要算法及数据结构）

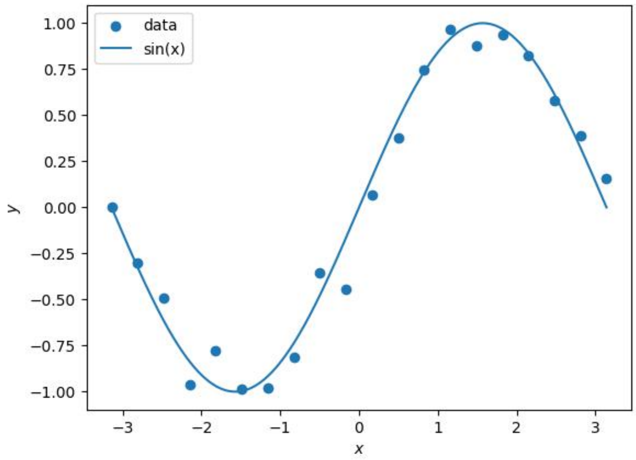
3.1 生成数据

本实验要求对正弦曲线进行多项式拟合。在这里选取区间为[-π， π]的正弦曲线进行实验。先选取样本点 20 个生成数据如下：



然后对数据添加噪声处理，这里选用最常用的高斯噪声，对采样数据添加 N

（0，0.1^2）的高斯噪声进行处理，结果如下：



可见通过噪声处理样本点对于正弦曲线有了不同程度的偏移，较为符合现实中的分布情况，接下来以这份数据进行实验，该数据备份保存在/data_reserved/data.csv

3.2 解析解求解无正则项 loss 的最优解

解析解，是指通过严格的公式所求得的解。对于多项式拟合曲线问题，即利用训练集，拟合

$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$

使得拟合曲线与实际曲线最为接近。上式写成矩阵为 **y = θx**
其中θ = [a0,a1,a2,...,an],x =[1,x,x^2,...,x^n].

误差函数通常写为

$$J(\theta) = \frac{1}{2m}(X*\theta - Y)^T(X*\theta - Y)$$

对于本问题，只存在一个参数θ。通过最小化误差函数，我们可以得到相应的最优解。一般的，采用误差函数而此最小化误差函数存在解析解：

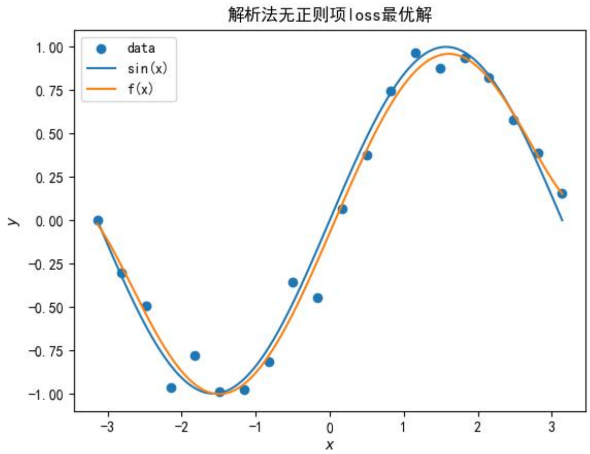
$$\theta = (X^T X)^{-1} X^T Y$$

python 代码实现：

```
polyFeat_x = np.array([x ** i for i in range(dimension + 1)])

theta1 = np.linalg.pinv(polyFeat_x @ polyFeat_x.T) @ polyFeat_x @ y
```

实验拟合结果如图，f(x)即为拟合曲线



3.3 解析解求解正则项 loss 的最优解

为了避免过拟合现象的发生，我们在进行多项式拟合时倾向于得到更加平滑的解，即向量内各元素都较小的学习参数θ，为此在误差函数中加入正则项（θ的 L2 范数的λ倍），得到新的误差函数：

$$J(\theta)=\frac{1}{2m}(X*\theta-Y)^T(X*\theta-Y)+\lambda\|w\|_2^2$$

该函数的解析解为

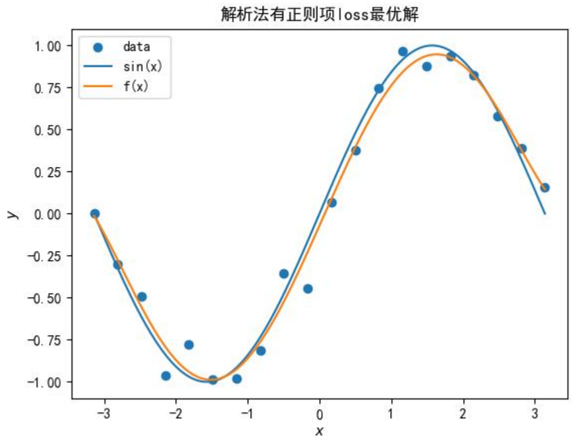
$$\theta=(X^TX+\lambda E)^{-1}X^TY$$

python 代码实现：

```
polyFeat_x = np.array([x ** i for i in range(dimension + 1)])

theta = np.linalg.pinv(np.eye(polyFeat_x.shape[0], dtype=float) * lambda_analytical + polyFeat_x @ polyFeat_x.T) @ polyFeat_x @ y
```

实验拟合结果如图，f(x)即为拟合曲线



3.4 梯度下降求解 loss 的最优解

在机器学习算法中，在最小化损失函数时，可以通过梯度下降法来一步步的迭代求解，得到最小化的损失函数，和模型参数值，即在函数中，找到给定的梯度，朝着梯度相反的方向更新变量，反复迭代，从而得到函数达到最小值时所对应的变量。

通过对损失函数 J(θ)求导，可以得到

$$\frac{\mathrm{d}}{\mathrm{d}\theta}J(\theta)=\frac{1}{n}(X\theta-y)X^T$$

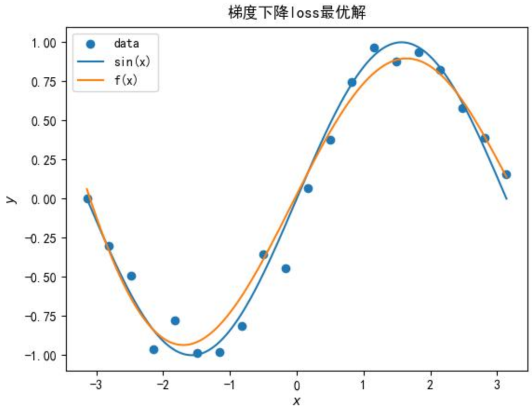
通过梯度对θ进行迭代：

$$\theta:=\theta_0-\alpha\frac{\mathrm{d}}{\mathrm{d}\theta}J(\theta_0)$$

python 代码实现：

```
def getLoss(yo, t, polyFeat):
    return 0.5 * np.linalg.norm(t @ polyFeat - yo)


times = 0
theta = np.array([np.random.random() for i in range(polyFeat_x.shape[0])])
loss = getLoss(y, theta, polyFeat_x)
regular_param = 0.01
delta_loss = 1
while delta_loss > 10 ** -8:
    derivative = (theta @ polyFeat_x - y) @ polyFeat_x.T + regular_param * theta
    theta -= derivative * learning_rate / np.linalg.norm(derivative)
    loss_0 = loss
    loss = getLoss(y, theta, polyFeat_x)
    delta_loss = abs(loss_0 - loss)
    print('Turn {} Loss = {}'.format(times, loss))
    times += 1
    if times % 70 == 0:
        learning_rate *= 0.96
print('Training completed.\nFinal Loss = {}'.format(loss))
```



3.5 共轭梯度下降求解 loss 的最优解

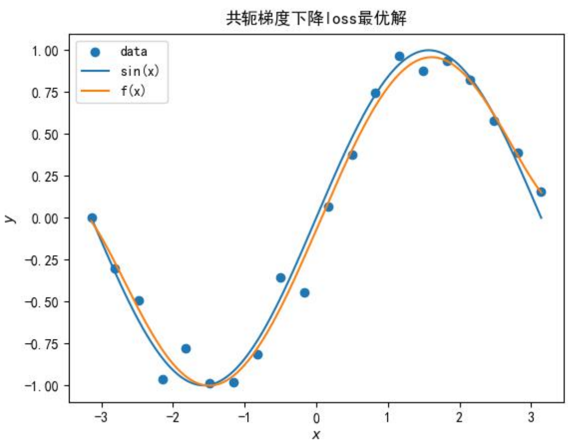
共轭梯度法解决的主要是形如 A x = b \bold{Ax} = \bold{b} Ax=b 的线性方程组解的问题，其中 A \bold{A} A 必须是对称的、正定的。大概来说，共轭梯度下降就是在解空间的每一个维度分别取求解最优解的,每一维单独去做的时候不会影响到其他维，这与梯度下降方法，每次都选泽梯度的反方向去迭代，梯度

下降不能保障每次在每个维度上都是靠近最优解的,这就是共轭梯度优于梯度下降的原因。

python 代码实现:

```
times = 0
theta = np.array([np.random.random() for i in range(polyFeat_x.shape[0])])
loss = getLoss(y, theta, polyFeat_x)
regular_param = 0.01
A = polyFeat_x @ polyFeat_x.T + regular_param * np.eye(polyFeat_x.shape[0])
b = polyFeat_x @ y
r_0 = b - theta @ A
p = r_0
while True:
    alpha = (r_0.T @ r_0) / (p.T @ A @ p)
    theta = theta + p * alpha
    r = r_0 - alpha * A @ p
    if r_0 @ r_0 < 10 ** -8:
        break
    beta = (r.T @ r) / (r_0.T @ r_0)
    p = r + beta * p
    r_0 = r
    loss = getLoss(y, theta, polyFeat_x)
    times += 1
    print('Turn {} Loss = {}'.format(times, loss))

print('Training completed.\nFinal Loss = {}'.format(loss))
```



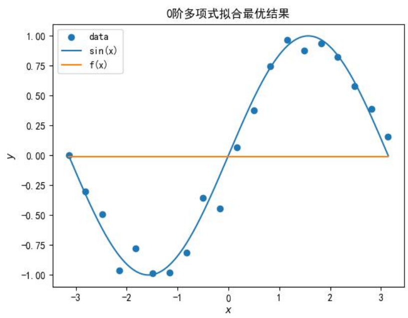
四、实验结果与分析

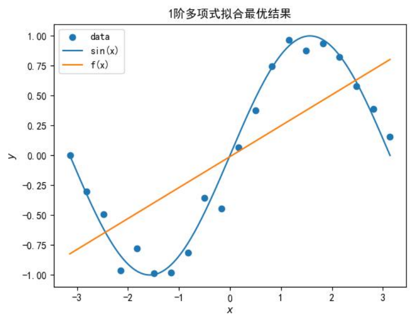
4.1 对过拟合现象的解释

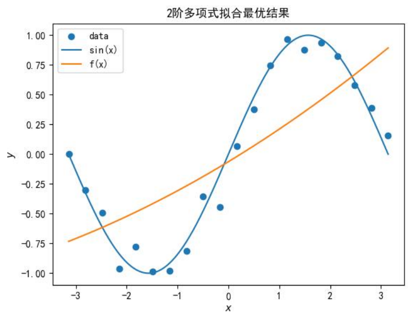
由于多项式拟合问题存在解析解，这里直接使用解析解确定的结果对过拟合现象进行分析，因而不需要对学习率进行讨论，只需要关注数据点数 points 和多项式阶数 n。

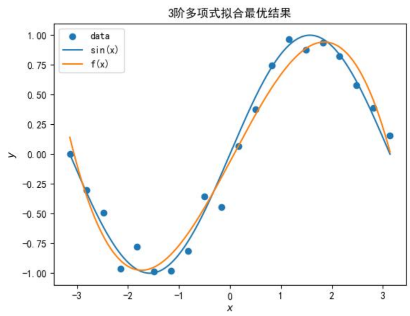
首先进行对多项式阶数的分析，固定数据点数为 20 个，对 n=0,1,2,3,4,5,12,13 分别运行程序，

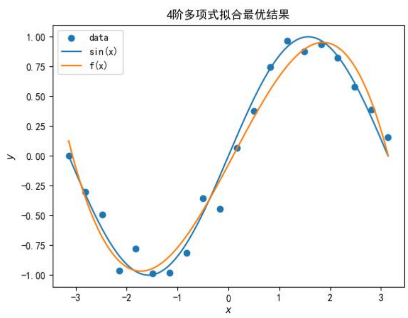
得到如下结果：

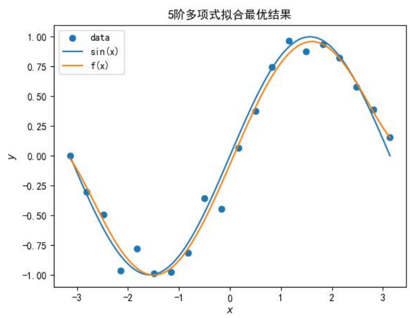


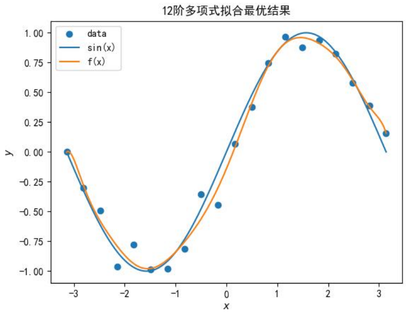


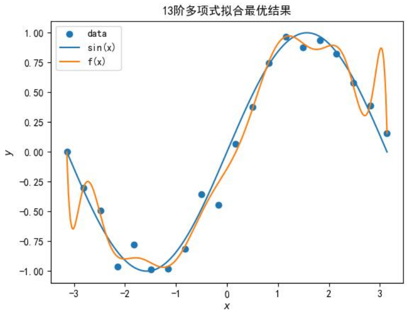








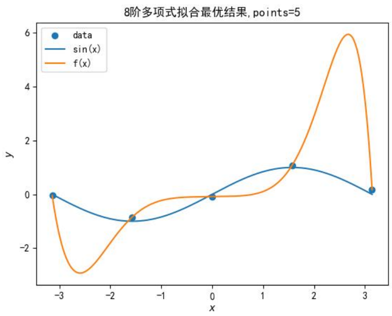


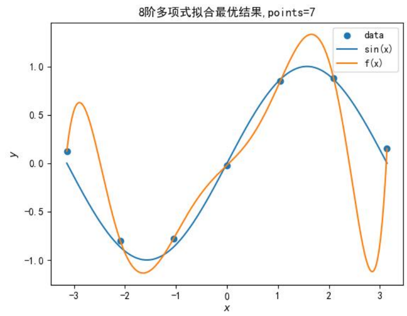


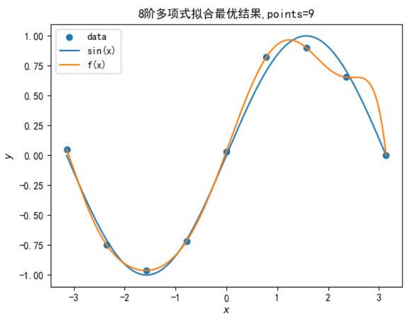
可以看到，对于 n 阶多项式，当 n 由 0 增大到 5 的过程中，多项式逐渐逼近正弦曲线，在 n 增大到 12 时，曲线形状保持基本不变，而当 n=13 时，曲线发生过拟合现象，和正弦曲线的形状有了明显的偏差。

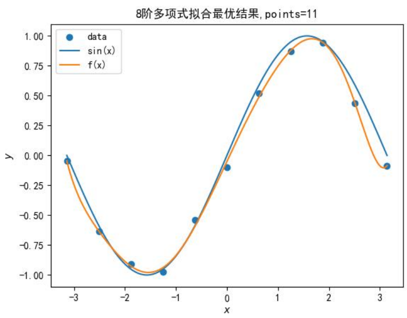
然后进行对数据点数的分析，固定 n=6。由于正弦曲线定形需要至少一定量的点，从 points=5 开始增加，对 n=5,7,9,11,13,15,30,45 分析。

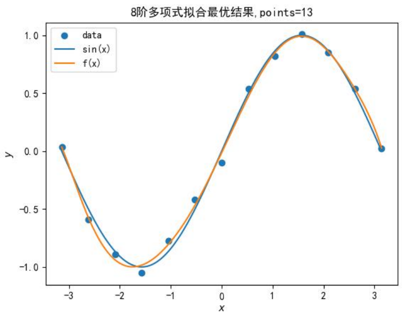
运行程序得到如下结果：

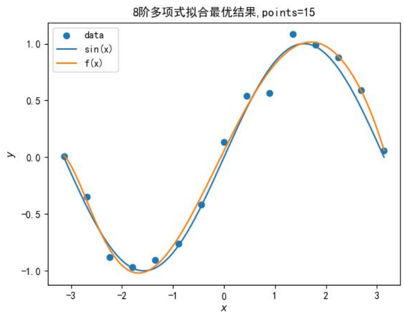


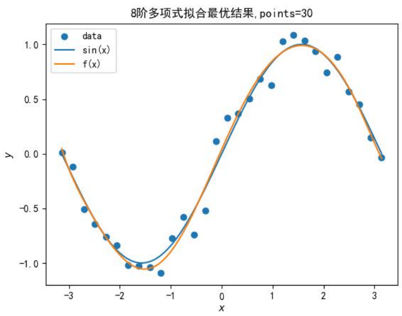


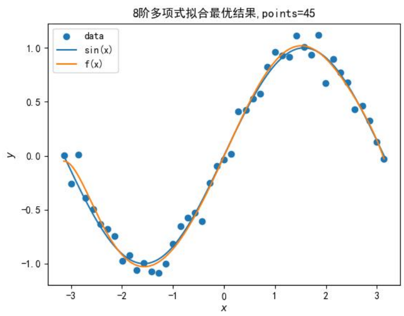












可以看到，对于 8 阶多项式，当样本点较少，为 5，7，9 时拟合曲线均发生不同程度的过拟合，但随着样本点数量的增加，曲线的形状逐渐被固定为近似正弦曲线，当样本点足够多为 90 时，多项式曲线可以很好的拟合正弦曲线。

综上所述，n 值过大或样本点较少都会导致过拟合现象的发生。

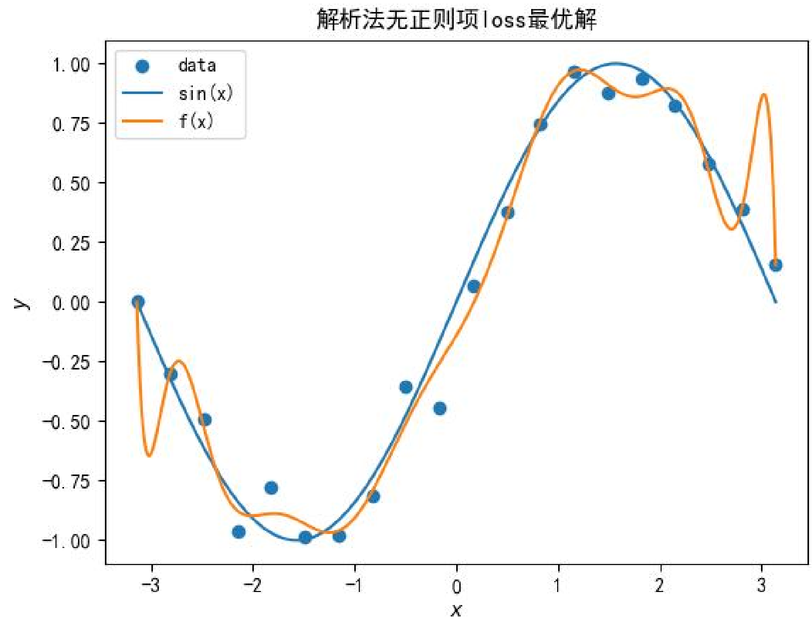
当 n 值过大时，拟合曲线可以变得更加复杂，逐渐对样本点中的随机误差进行了更多的学习，使曲线开始距待拟合曲线产生误差，形成过拟合。

当样本点过少时，样本点难以准确表示曲线的几何特征，同时随机误差的影响也凸显出来，产生过拟合。而随着样本点的增加，正弦曲线的形状越发明显，随机误差也被大量的点所稀释，过拟合逐渐消失。

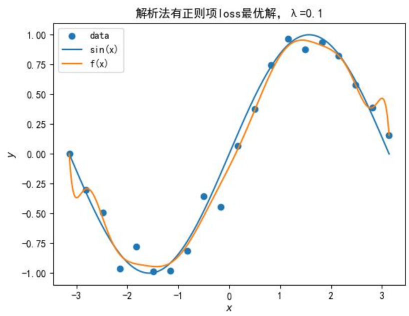
4.2 不同参数对实验结果的影响

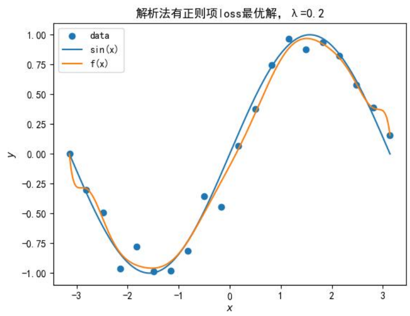
在本次实验中一共有数据量、超参数、多项式阶数等因素对拟合曲线的效果产生影响。其中数据量和多项式阶数已经在 4.1 重点讨论过，接下来就超参数进行讨论。这里的超参数包括正则化参数λ和学习率。

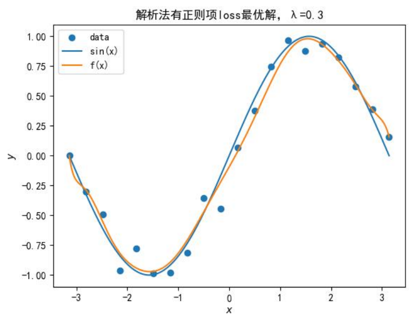
正则化参数的讨论基于解析解，这样可以忽略学习率的影响。我们选取一个 4.1 出现的会发生过拟合的参数，即 n=13，points=20。在没有正则项的情况下，或者说λ=0 时，如图：

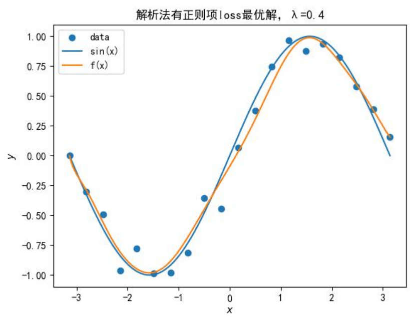


曲线发生了明显的过拟合。取正则化参数λ=0.1，0.2，0.3，0.4 可得下列结果：

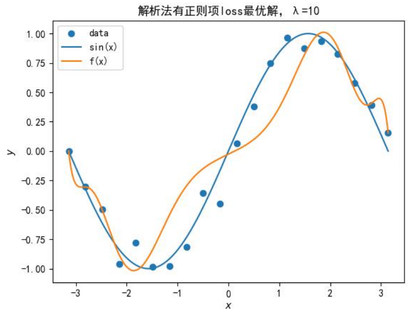


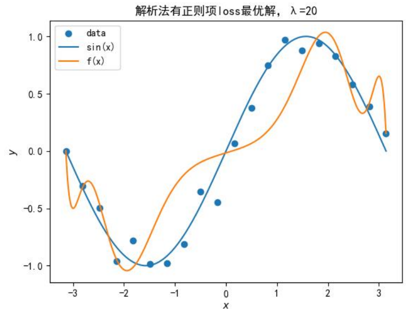




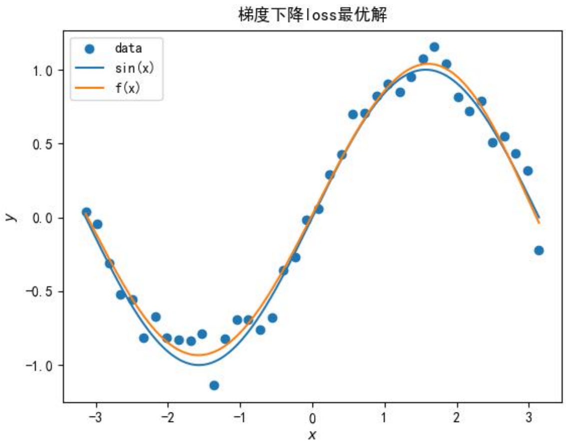


可见，有了正则项，相比于无正则项的情况，过拟合现象随着λ的增大被逐渐消除，直到λ=0.4 时不再有过拟合现象。至于λ取得过大时，则是会导致欠拟合的发生。下为λ=10，20 的拟合曲线：





结束了对正则化参数的讨论，我们接着讨论学习率的影响，设定正则化参数λ为 0.4，多项式阶数 n=8，points=40。当学习率为 0.01 时，梯度下降法迭代 742507 轮得到与其他方式相似的结果。



而当学习率变小时，如学习率为 0.005 时，为了得到同样的结果，程序迭代了 1340051 轮，学习率变小降低了学习速率。而当学习率过大时，如学习率为 0.1 时，程序便陷入了死循环，无法收敛了。

```
Turn 164246 Loss = 57.087681818837744
Turn 164247 Loss = 40.929364178528914
Turn 164248 Loss = 57.08768172919054
Turn 164249 Loss = 40.92936408887152
```

五、结论

5.1 过拟合现象

过拟合是指在模型参数拟合过程中的问题，由于训练数据包含抽样误差，训练时，复杂的模型将抽样误差也考虑在内，将抽样误差也进行了很好的拟合。n 值过大或样本点较少都会导致过拟合现象的发生。当 n 值过大时，拟合曲线可以变得更加复杂，逐渐对样本点中的随机误差进行了更多的学习，使曲线开始距待

拟合曲线产生误差，形成过拟合。当样本点过少时，样本点难以准确表示曲线的几何特征，同时随机误差的影响也凸显出来，产生过拟合。而随着样本点的增加，正弦曲线的形状越发明显，随机误差也被大量的点所稀释，过拟合逐渐消失。

5.2 不同参数对实验结果的影响

n 过大，样本点过少，λ值过小都会导致过拟合现象出现。而 n 过小，λ值过大会导致欠拟合出现。学习率过大会导致不能收敛。因此选择更多的样本点，适当的λ值、n 值和学习率可以很好的模拟待拟合曲线。

六、参考文献

1）Pattern Recognition and Machine Learning.
2）Gradient descent wiki
3）Conjugate gradient method wiki
4）Shewchuk J R. An introduction to the conjugate gradient method without the agonizing pain[J]. 1994.

七、附录：源代码（带注释）

文件名	说明
analytical_solution.py	解析解
config.py	参数设置
conjugate_gradient.py	共轭梯度下降
dataGenerator.py	数据生成器
gradient_descent.py	梯度下降

analytical_solution.py

```
import numpy as np
from matplotlib.pyplot import *
from config import *

# 设置字形
rc('font', family='SimHei')
rc('axes', unicode_minus=False)

# 读取数据
data = np.loadtxt(fname="data.csv", dtype=float, delimiter=",")
x = data[1, :]
y = data[0, :]
```

```
# 计算无正则项解析解
polyFeat_x = np.array([x ** i for i in range(dimension + 1)])
theta1 = np.linalg.pinv(polyFeat_x @ polyFeat_x.T) @ polyFeat_x @ y
# 画图
a = np.linspace(left, right, 2000)
polyFeat_a = np.array([a ** i for i in range(dimension + 1)])
b1 = theta1 @ polyFeat_a
subplot(1, 2, 1)
scatter(x, y)
plot(a, np.sin(a))
plot(a, b1)
legend(['data', 'sin(x)', 'f(x)'])
xlabel('$x$')
ylabel('$y$')
title('解析法无正则项 loss 最优解')
# 计算有正则项解析解
theta2 = np.linalg.pinv(
    np.eye(polyFeat_x.shape[0], dtype=float) * lambda_analytical +
polyFeat_x @ polyFeat_x.T) @ polyFeat_x @ y
b2 = theta2 @ polyFeat_a
# 画图
subplot(1, 2, 2)
scatter(x, y)
plot(a, np.sin(a))
plot(a, b2)
xlabel('$x$')
ylabel('$y$')
legend(['data', 'sin(x)', 'f(x)'])
title('解析法有正则项 loss 最优解, λ={}'.format(lambda_analytical))
show()
```

config.py

```
import numpy as np

left = -np.pi  # 生成数据左端点
right = np.pi  # 生成数据右端点
dimension = 6  # 多项式维数
points = 40  # 样本点数
theta_normal = 0.1  # 高斯噪声方差
lambda_analytical = 0.4  # 正则化参数
learning_rate = 1  # 学习率
```

conjugate_gradient.py

```
    import numpy as np
from matplotlib.pyplot import *
from config import *

# 设置字形
rc('font', family='SimHei')
rc('axes', unicode_minus=False)
title('n = {}'.format(dimension))

# 读取数据
data = np.loadtxt(fname="data.csv", dtype=float, delimiter=",")
x = data[1, :]
y = data[0, :]
polyFeat_x = np.array([x ** i for i in range(dimension + 1)])


# 定义 loss 函数
def getLoss(yo, t, polyFeat):
    return 0.5 * np.linalg.norm(t @ polyFeat - yo)


times = 0  # 迭代次数
# 随机生成初始矩阵
theta = np.array([np.random.random() for i in
range(polyFeat_x.shape[0])])
# 计算初始 loss
loss = getLoss(y, theta, polyFeat_x)
lambda_analytical = 0.01
# 初始化共轭矩阵计算
A = polyFeat_x @ polyFeat_x.T + lambda_analytical *
np.eye(polyFeat_x.shape[0])
b = polyFeat_x @ y
r_0 = b - theta @ A
p = r_0
# 共轭矩阵迭代循环
while True:
    alpha = (r_0.T @ r_0) / (p.T @ A @ p)
    theta = theta + p * alpha
    r = r_0 - alpha * A @ p
    if r_0 @ r_0 < 10 ** -8:
```

```
        break
    beta = (r.T @ r) / (r_0.T @ r_0)
    p = r + beta * p
    r_0 = r
    loss = getLoss(y, theta, polyFeat_x)
    times += 1
    print('Turn {} Loss = {}'.format(times, loss))

print('Training completed.\nFinal Loss = {}'.format(loss))

# 画图
a = np.linspace(left, right, 2000)
polyFeat_a = np.array([a ** i for i in range(dimension + 1)])
b1 = theta @ polyFeat_a
scatter(x, y)
plot(a, np.sin(a))
plot(a, b1)
legend(['data', 'sin(x)', 'f(x)'])
xlabel('$x$')
ylabel('$y$')
title('共轭梯度下降 loss 最优解')
show()
```

dataGenerator.py

```
    import numpy as np
from config import *
from matplotlib.pyplot import *

# 生成 x
x = np.linspace(left, right, points)
y = np.sin(x)  # 计算 y
# 加入高斯噪声
yg = y + np.array([np.random.normal(0, theta_normal) for i in range(0,
points)])
data = np.append(yg, x).reshape(2, points)
# 画图
scatter(x, yg)
plot(np.linspace(left, right, 2000), np.sin(np.linspace(left, right,
2000)))
legend(['data', 'sin(x)'])
```

```
xlabel('$x$')
ylabel('$y$')
show()
# 保存
np.savetxt(fname="data.csv", X=data, fmt="%f", delimiter=",")
```

gradient_descent.py

```
    import numpy as np
from matplotlib.pyplot import *
from config import *

# 设置字形
rc('font', family='SimHei')
rc('axes', unicode_minus=False)

# 读取数据
data = np.loadtxt(fname="data.csv", dtype=float, delimiter=",")
x = data[1, :]
y = data[0, :]
polyFeat_x = np.array([x ** i for i in range(dimension + 1)])


# 定义 loss 函数
def getLoss(yo, t, polyFeat):
    return 0.5 * np.linalg.norm(t @ polyFeat - yo)


times = 0  # 迭代次数
# 随机生成初始矩阵
theta = np.array([np.random.random() for i in
range(polyFeat_x.shape[0])])
# 计算初始 loss
loss = getLoss(y, theta, polyFeat_x)
# 初始化 loss 变化
delta_loss = 1
# 梯度下降迭代循环
while loss > 1 or delta_loss > 10 ** -8:
    derivative = (theta @ polyFeat_x - y) @ polyFeat_x.T +
lambda_analytical * theta
    theta -= derivative * learning_rate / np.linalg.norm(derivative)
```

```
    loss_0 = loss
    loss = getLoss(y, theta, polyFeat_x)
    delta_loss = abs(loss_0 - loss)
    print('Turn {} Loss = {}'.format(times, loss))
    times += 1
    if times % 5000 == 0:
        learning_rate *= 0.96

print('Training completed.\nFinal Loss = {}'.format(loss))

# 画图
a = np.linspace(left, right, 2000)
polyFeat_a = np.array([a ** i for i in range(dimension + 1)])
b1 = theta @ polyFeat_a
scatter(x, y)
plot(a, np.sin(a))
plot(a, b1)
legend(['data', 'sin(x)', 'f(x)'])
xlabel('$x$')
ylabel('$y$')
title('梯度下降 loss 最优解')
show()
```