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：必修

实验题目：多项式拟合正弦函数

学号：1190200501

姓名：林燕燕

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

OS： Windows 11

Python： 3.7.9

三、设计思想

1. 生成数据并加入噪声

函数 $f(x)=sin(2\pi x)$，取训练集 $[X,T]$， $X=[x_1,x_2,...,x_N]$，其中 x_i 均匀分布于 $[0,1]$ ，分别带入函数得到，

$$Y=[y_1,y_2,...,y_N]=[f(x_1),f(x_2),...,f(x_N)]$$

加入均值为 0，标准差为 0.05 的高斯噪声 p_{Gi} ，得到:

$$T=[t_1,t_2,...,t_N]=[f(x_1)+p_{G1},f(x_2)+p_{G2},...,f(x_N)+p_{GN}]$$

即可得到加入噪声的训练集 $[X,T]$。

2. 利用高阶多项式函数拟合曲线（无正则项）

使用多项式函数来拟合曲线，定义多项式函数 $y(x,M)$:

$$y(x,M)=w_0+w_1x+w_2x^2+...+w_Mx^M=\sum_{i=0}^{M}w_ix^i$$

其中 M 为多项式阶数， $w_0,w_2,...,w_M$ 为多项式系数，

记 $W=[w_0,w_2,...,w_M]^T$ ， $X_k=[1,x_k,x_k^2,...,x_k^M]$ ，则将多项式函数矩阵化为:

$$y(x,M)=\sum_{i=0}^{M}w_ix^i=X_kW$$

定义误差函数

$$E(W)=\frac{1}{2}\sum_{j=1}^{N}(y(x_j,W)-t_j)^2$$

记 $X=[X_1,X_2,\ldots,X_N]^T=\begin{bmatrix}1 & x_1 & \cdots & x_1^M\\ 1 & x_2 & \cdots & x_2^M\\ \vdots & \vdots & \ddots & \vdots\\ 1 & x_N & \cdots & x_N^M\end{bmatrix}$ ， $T=[t_1,t_2,\ldots,t_N]^T$ ，矩阵化误差函数为：

$$E(W)=\frac{1}{2}(XW-T)^T(XW-T)$$

对数据进行拟合需要使误差函数 $E(W)$ 取值达到最小，则 $E(W)$ 对 W 求导：

$$\begin{aligned}\frac{\partial E}{\partial W}&=\frac{1}{2}\frac{\partial((XW-T)^T(XW-T))}{\partial W}\\&=\frac{1}{2}\frac{\partial((W^TX^T-T^T)(XW-T))}{\partial W}\\&=\frac{1}{2}\frac{\partial(W^TX^TXW-W^TX^TT-T^TXW+T^TT)}{\partial W}\\&=\frac{1}{2}(2X^TXW-X^TT-X^TT)\\&=X^TXW-X^TT\end{aligned}$$

令 $\frac{\partial E}{\partial W}=0$ 得到：

$$\begin{aligned}W^*&=(X^TX)^{-1}X^TT\\&=X^{-1}(X^T)^{-1}X^TT\\&=X^{-1}T\end{aligned}$$

代入 X,T 即可得到 W^* 获得拟合曲线。

3.利用高阶多项式函数拟合曲线（有正则项）

通过正则化减轻过拟合影响，为误差函数增加一个惩罚项如下：

$$\widetilde{E}(W)=\frac{1}{2}\sum_{j=1}^{N}(y(x_j,W)-t_j)^2+\frac{\lambda}{2}||W||^2$$

其中 $||W||^2=W^TW=w_1^2+w_2^2+\ldots+w_M^2$ ，将 $\widetilde{E}(W)$ 矩阵化得：

$$\begin{aligned}\widetilde{E}(W)&=\frac{1}{2}(XW-T)^T(XW-T)+\frac{\lambda}{2}W^TW\\&=\frac{1}{2}(W^TX^TXW-W^TX^TT-T^TXW+T^TT+\lambda W^TW)\end{aligned}$$

对 $\widetilde{E}(W)$ 求导：

$$\begin{aligned}\frac{\partial\widetilde{E}}{\partial W}&=\frac{1}{2}\frac{\partial(W^TX^TXW-W^TX^TT-T^TXW+T^TT+\lambda W^TW)}{\partial W}\\&=\frac{1}{2}(2X^TXW-X^TT-X^TT+2\lambda W)\\&=X^TXW-X^TT+\lambda W\end{aligned}$$

令 $\frac{\partial E}{\partial W}=0$ 得到：

$$W^* = (X^TX + \lambda I)^{-1}X^TT$$

其中 I 为单位阵，代入 X, T 可得 W^* .

4.梯度下降法求解最优解

对连续可微函数 $J(\Theta)$ ，$\Theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}$ ，其梯度为：

$$\nabla J(\Theta) = (\frac{\partial J}{\partial \theta_0}, \frac{\partial J}{\partial \theta_1}, \ldots, \frac{\partial J}{\partial \theta_n})$$

- 在单变量的函数中，梯度其实就是函数的微分，代表着函数在某个给定点的切线的斜率
- 在多变量函数中，梯度是一个向量，向量有方向，**梯度的方向就指出了函数在给定点的上升最快的方向；对应的，梯度的反方向就是给定点的下降最快的方向。**

梯度下降公式为 $\Theta_{i+1} = \Theta_i - \alpha \nabla J(\Theta)$ ，其中 α 为学习率或步长

对有惩罚项的误差函数：

$$\begin{aligned} \widetilde{E}(W) &= \frac{1}{2}\sum_{j=1}^{N}(y(x_j, W) - t_j)^2 + \frac{\lambda}{2}||W||^2 \\ &= \frac{1}{2}(XW - T)^T(XW - T) + \frac{\lambda}{2}W^TW \end{aligned}$$

其梯度为： $\nabla \widetilde{E}(W) = X^TXW - X^TT + \lambda W$

则 $W_{i+1} = W_i - \alpha \nabla \widetilde{E}(W)$ ，获得 W^* 使 $\widetilde{E}(W)$ 最小化，实现拟合。

5.共轭梯度法求解最优解

考虑线性对称正定方程组： $Ax = b$

构造二次函数：

$$\phi(x) = \frac{1}{2}x^TAx - b^Tx$$

对其求导得：

$$\nabla \phi(x) = \frac{\partial \phi(x)}{\partial x} = Ax - b$$

则求 $Ax = b$ 的解即为求函数 $\phi(x)$ 的极小值点

对误差函数：

$$\begin{aligned} \widetilde{E}(W) &= \frac{1}{2}\sum_{j=1}^{N}(y(x_j, W) - t_j)^2 + \frac{\lambda}{2}||W||^2 \\ &= \frac{1}{2}(XW - T)^T(XW - T) + \frac{\lambda}{2}W^TW \end{aligned}$$

其梯度为： $\nabla \widetilde{E}(W) = X^TXW - X^TT + \lambda W$

令其为 0 ，则可得到： $(X^TX + \lambda)W = X^TT$ ，记 $A = X^TX + \lambda$，$b = X^TT$ ，求 W

若 r_k 不满足精度，继续第 k 次循环：

$$a_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

$$W_k = w_{k-1} + \alpha_k p_k$$

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

$$b_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

$$p_{k+1} = r_{k+1} + b_k p_k$$

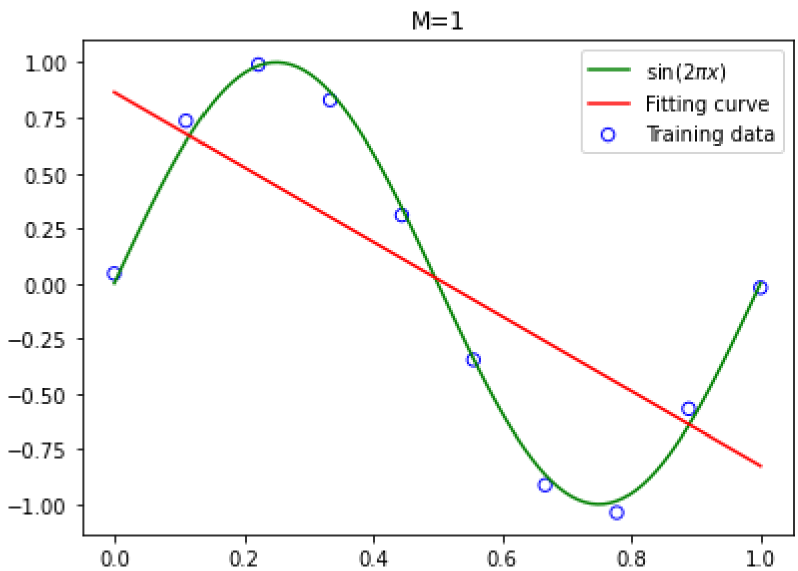
直到满足精度，退出循环，得到 W^* 用于拟合。

四、实验结果分析

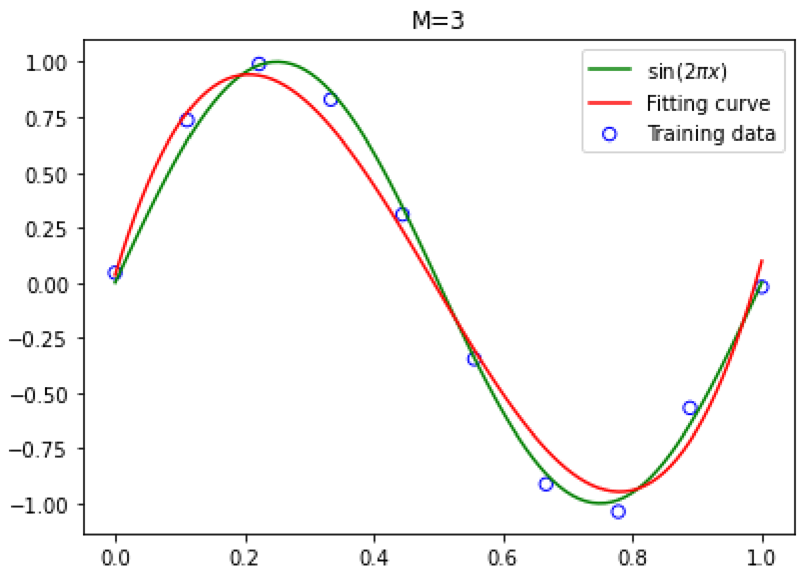
1.不带正则项的解析解

设定训练集大小为 10， 在不同阶数下的拟合结果：

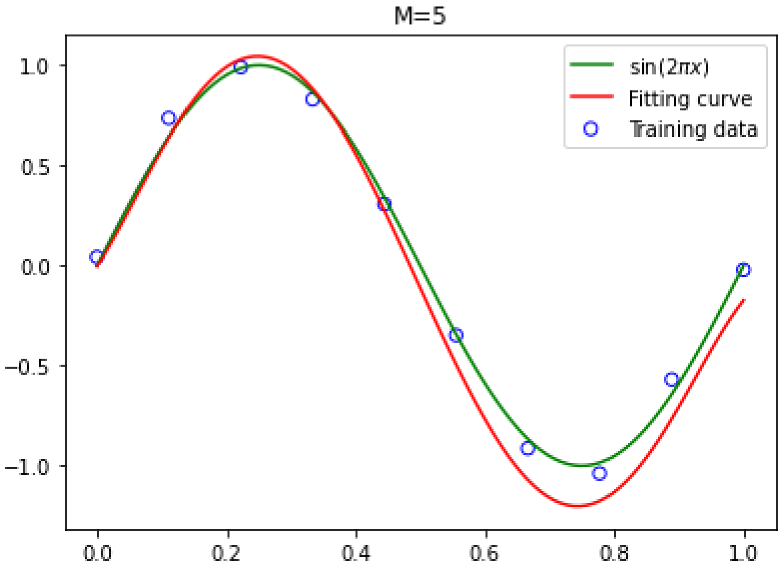
- 阶数为 1 时：



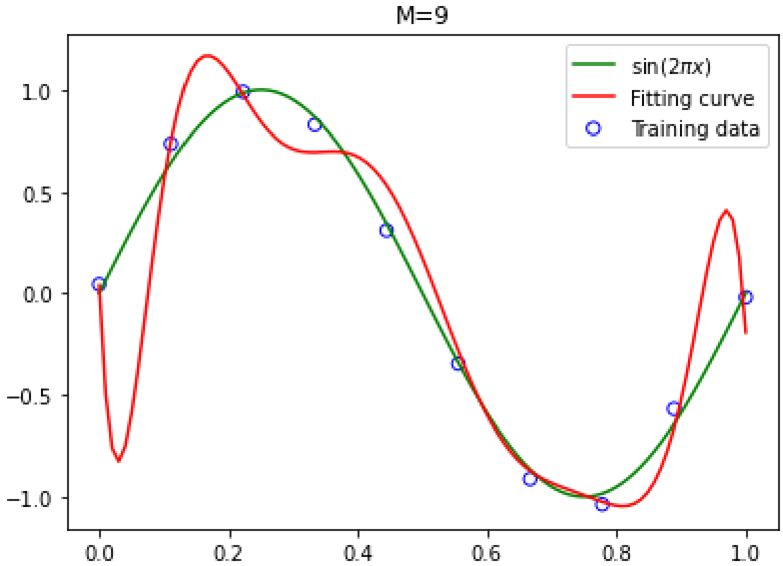
- 阶数为 3 时：



- 阶数为 5 时：



- 阶数为 9 时：



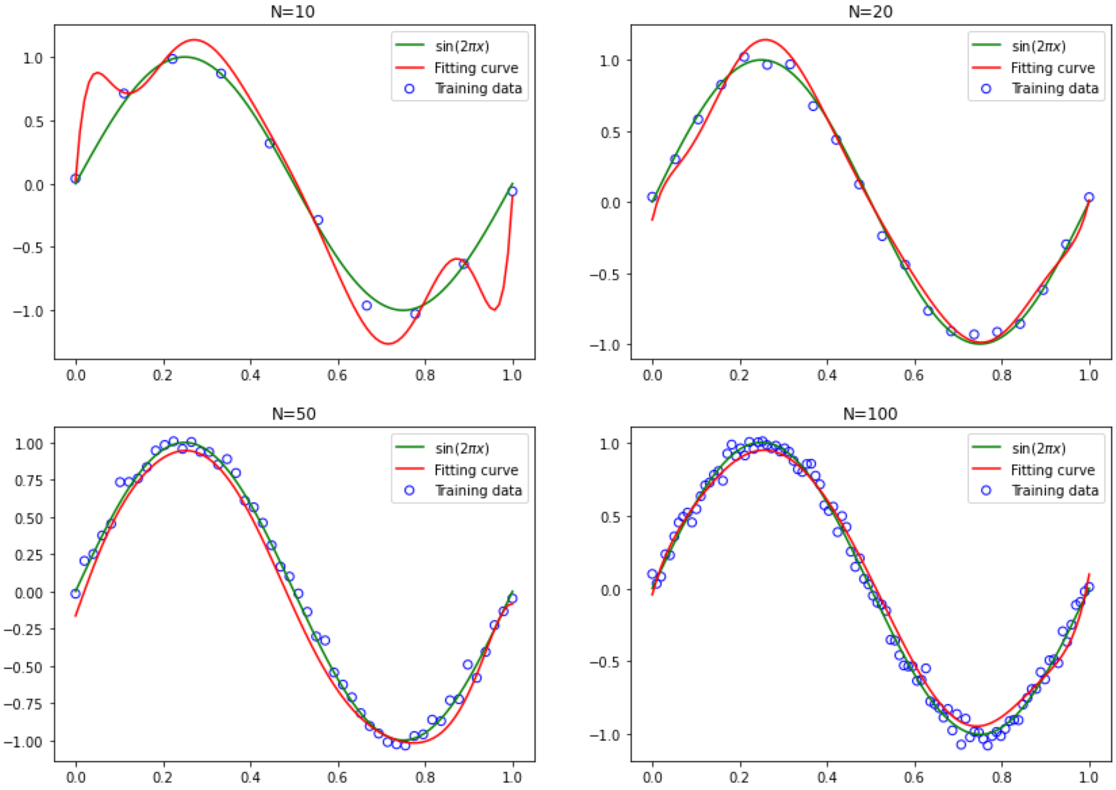
在1阶时，无法拟合，属于欠拟合，需要提高阶数；

在3阶时，拟合效果较好，阶数提高到5阶，拟合效果更好；

阶数提高到9阶，拟合函数图像能穿过大多数训练集上的点，但波动较大，出现过拟合现象。

可以通过增加训练集大小降低过拟合的影响：

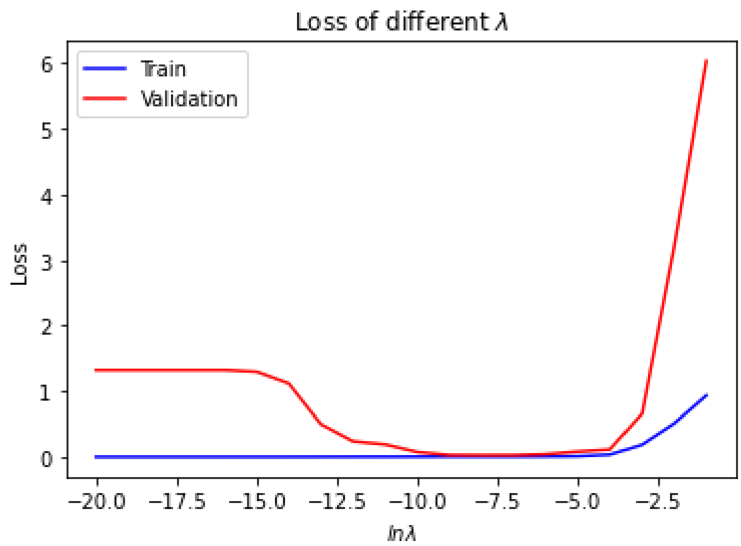
设定阶数为 9 阶，在训练集大小分别为 10，20，50，100 下拟合结果：

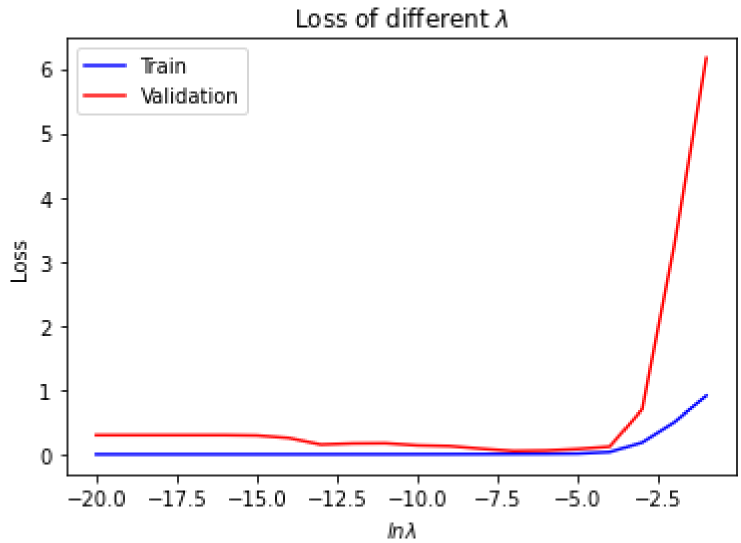


随着训练集增大，过拟合现象逐渐消失，拟合函数很好的拟合到原函数上。

2.带正则项的解析解

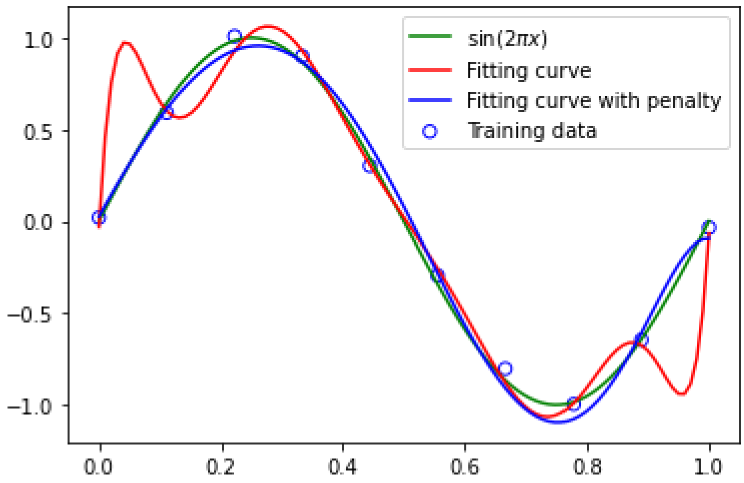
取训练集大小为 10，验证集大小为 20，阶数为 9，确定最优 λ





根据多次运行得到最佳 λ 取值范围为 $(10^{-8}, 10^{-5})$

取λ 为 10^{-6} ，在训练集大小为 10 ，阶数为 9 的条件下的带惩罚项和不带惩罚项的拟合图像比较



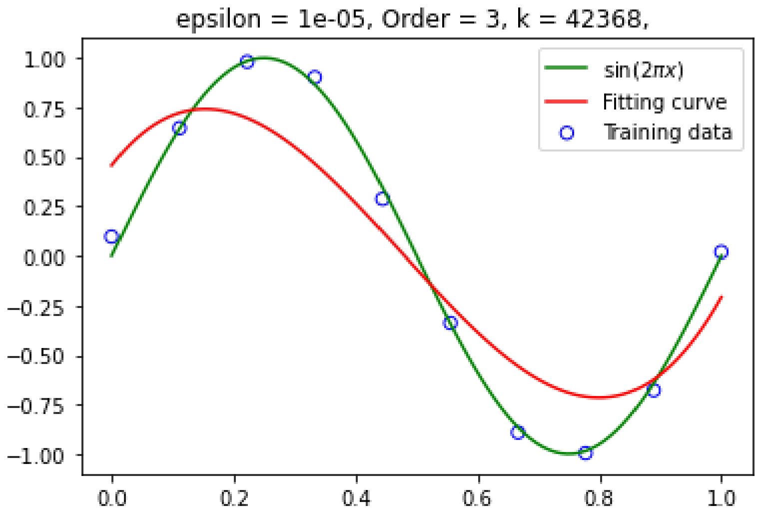
可以看出，加入惩罚项有效降低过拟合现象。

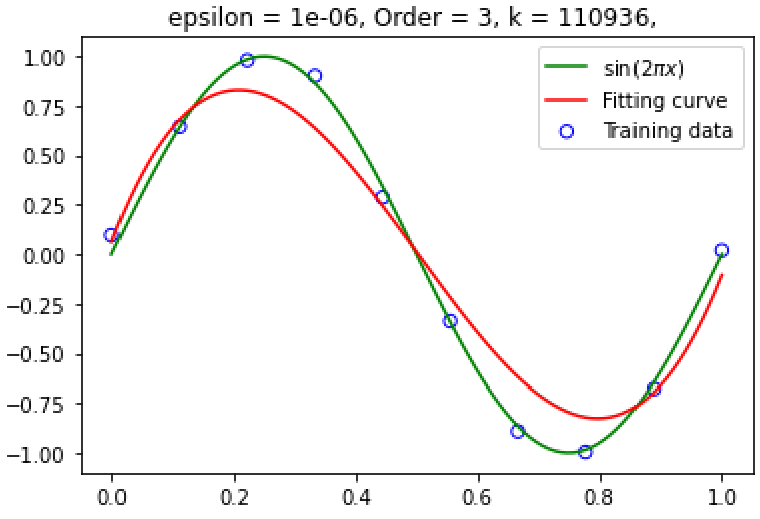
3.梯度下降求得优化解

设定 λ 为 10^{-6}，学习率 α 为 0.01，（左图精度为 10^{-5}，右图精度为 10^{-6} ）

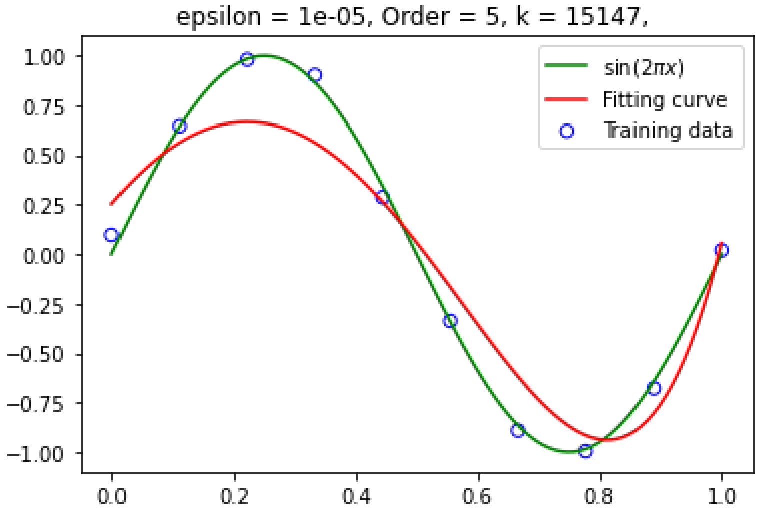
设定训练集大小为 10，在不同阶数下拟合函数:

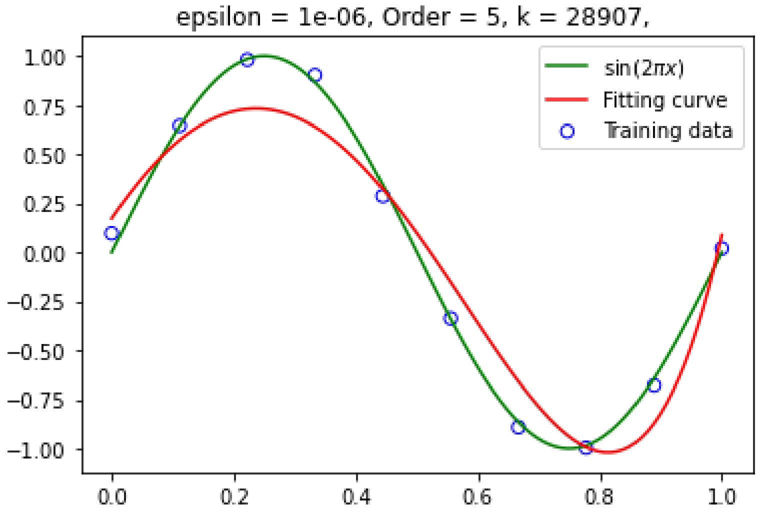
- 阶数为 3：



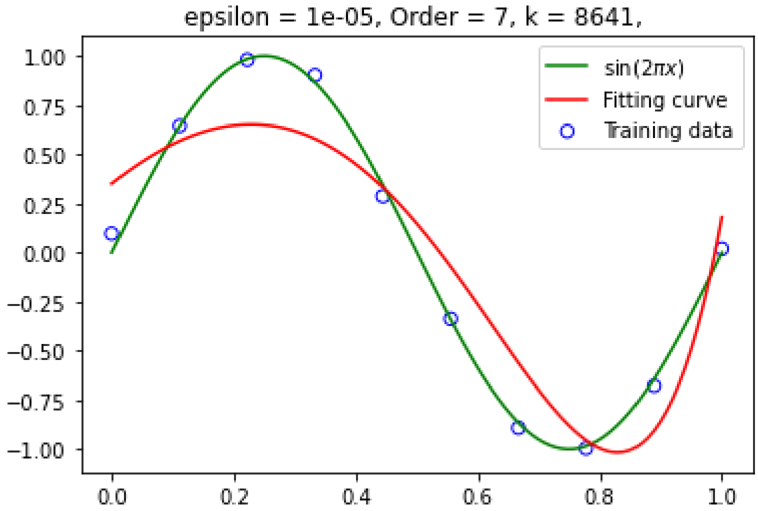


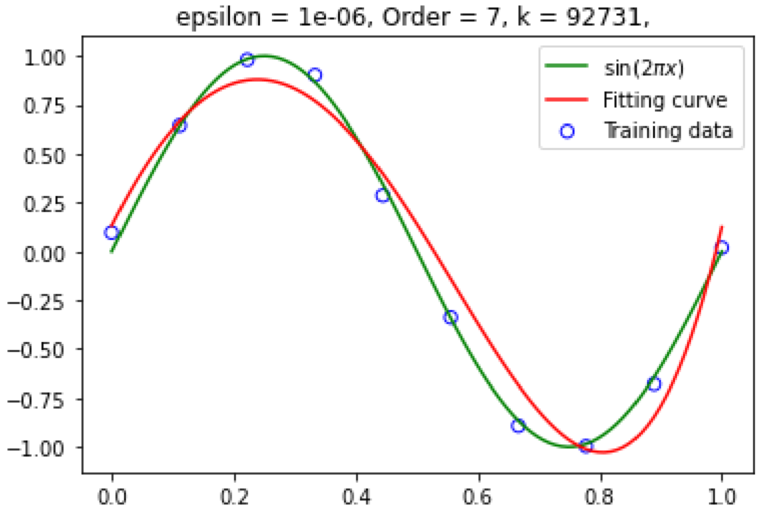
- 阶数为 5:


epsilon = 1e-05, Order = 5, k = 15147,
sin(2πx)
Fitting curve
Training data

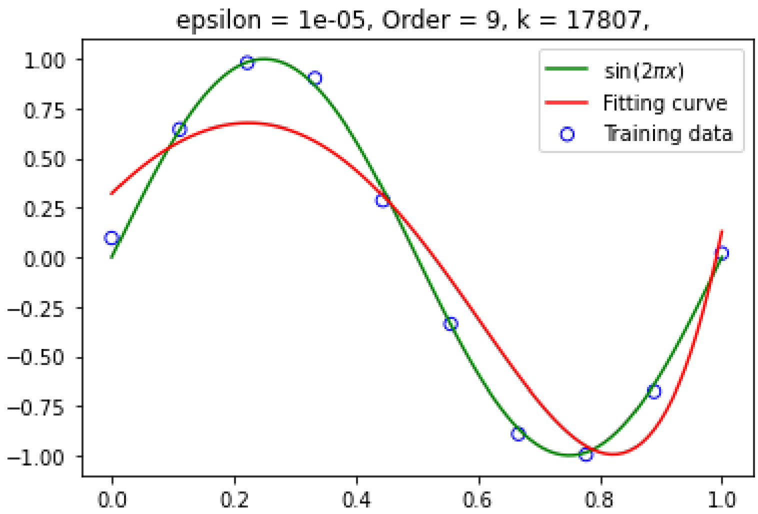

epsilon = 1e-06, Order = 5, k = 28907,
sin(2πx)
Fitting curve
Training data

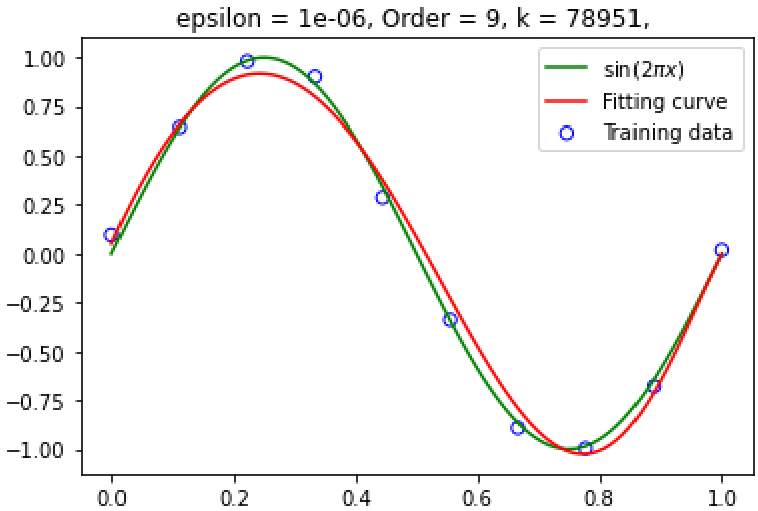
- 阶数为 7:


epsilon = 1e-05, Order = 7, k = 8641,
sin(2πx)
Fitting curve
Training data
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0


epsilon = 1e-06, Order = 7, k = 92731,
sin(2πx)
Fitting curve
Training data
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

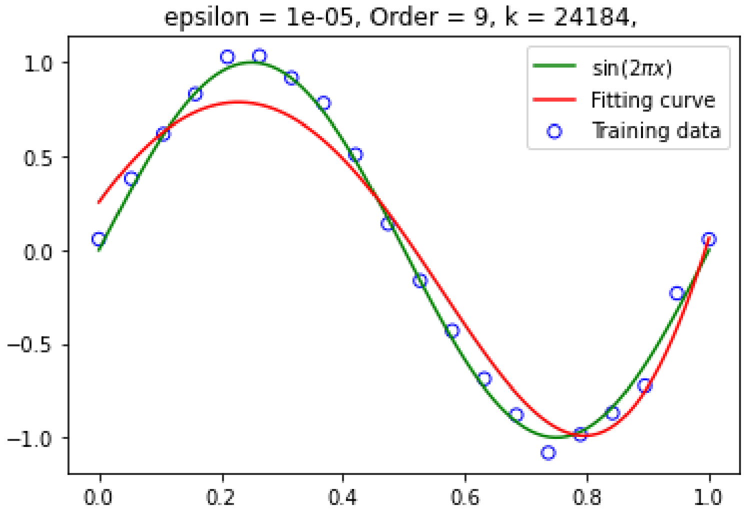
- 阶数为 9：



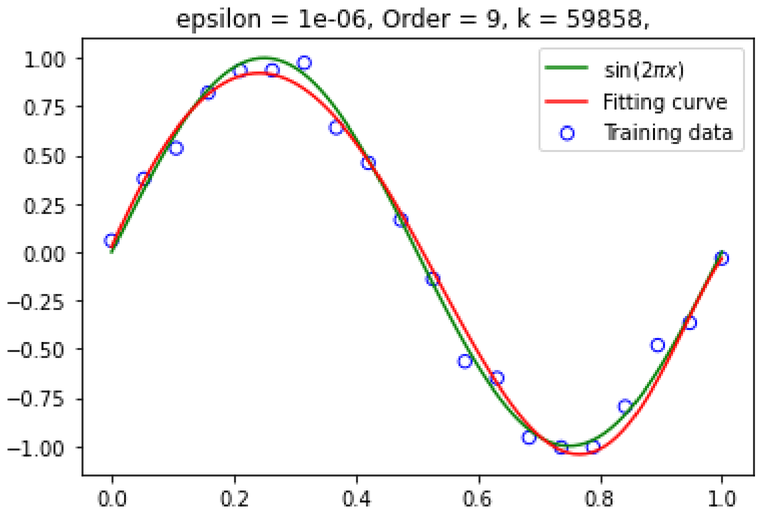


设定阶数为 9，在不同训练集大小下拟合函数

- 训练集大小为 20：

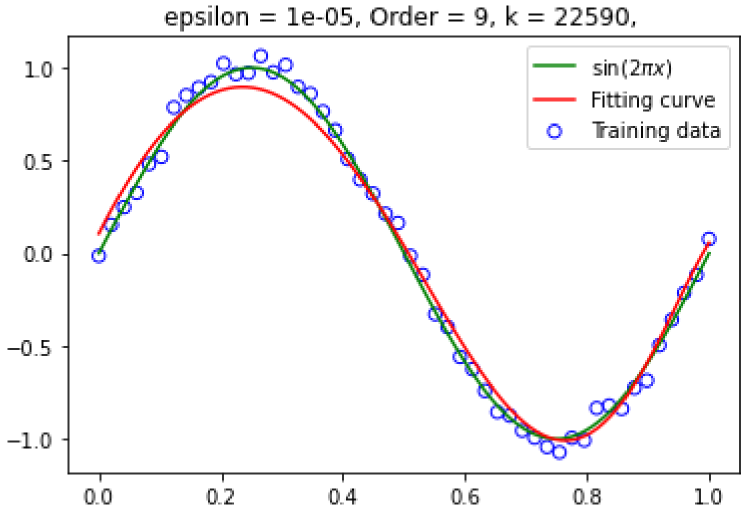


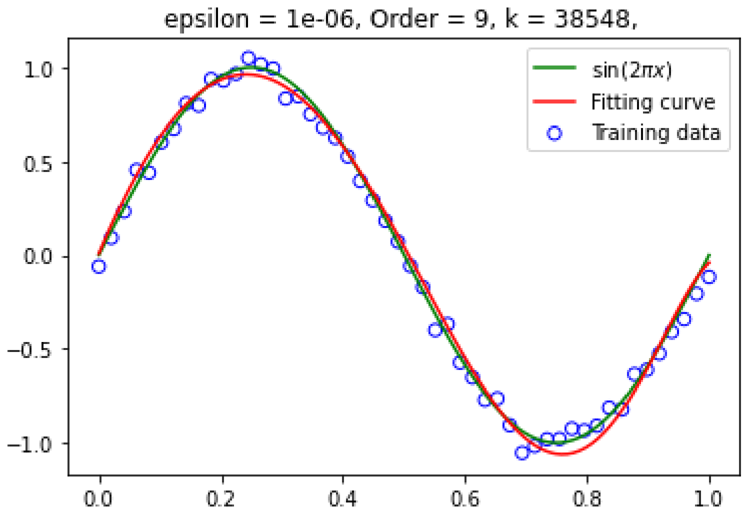
epsilon = 1e-05, Order = 9, k = 24184,
sin(2πx)
Fitting curve
Training data
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0



epsilon = 1e-06, Order = 9, k = 59858,
sin(2πx)
Fitting curve
Training data
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

- 训练集大小为 50：





从上图可看出，同条件下精度为 10^{-6} 的右图的拟合效果要显著强于精度为 10^{-5}下的拟合效果。

精度升高，迭代次数变大；多项式阶增大，迭代次数呈现下降趋势；

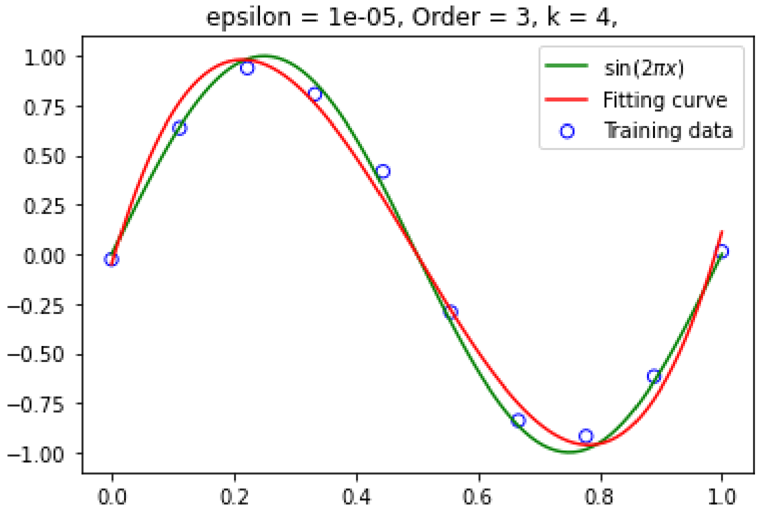
而训练集的大小对于迭代次数几乎没有影响，但仍然满足训练集越大拟合效果也好的结论。

4.共轭梯度求得优化解

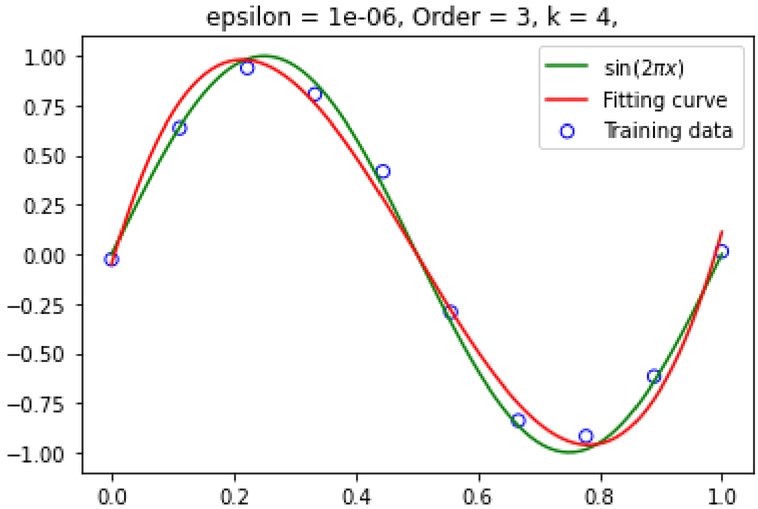
设定 λ 为 10^{-6}， （左图精度为 10^{-5}，右图精度为 10^{-6} ）

设定训练集大小为 10 ，在不同阶数下拟合函数:

- 阶数为 3：

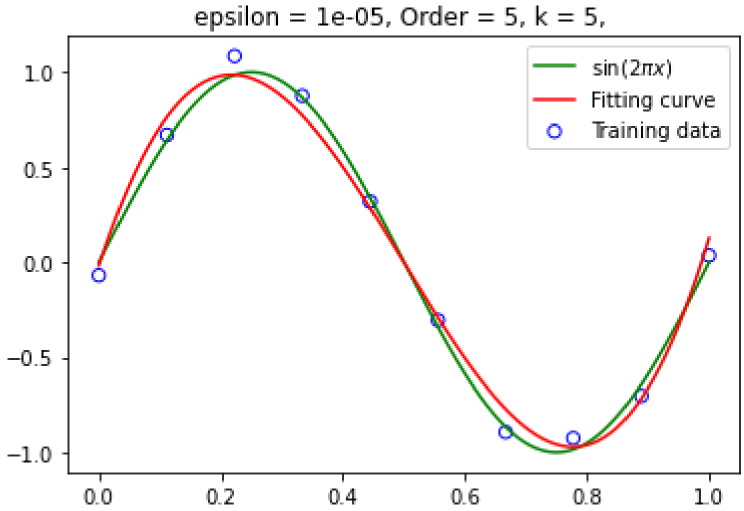


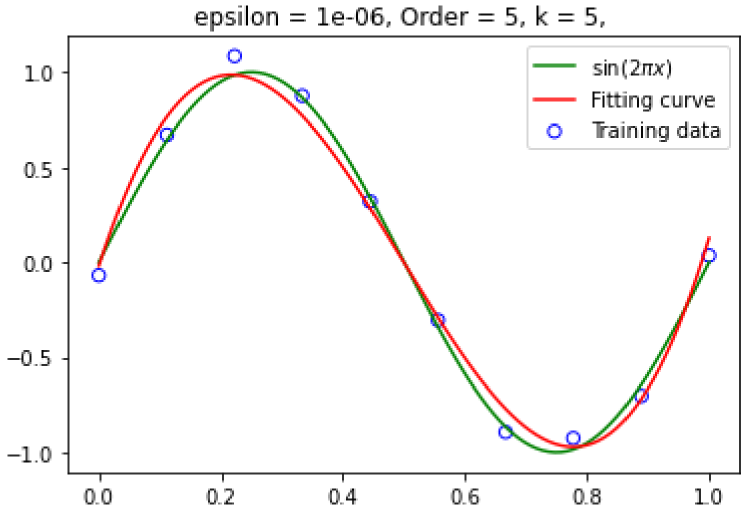
epsilon = 1e-05, Order = 3, k = 4,
sin(2πx)
Fitting curve
Training data
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0



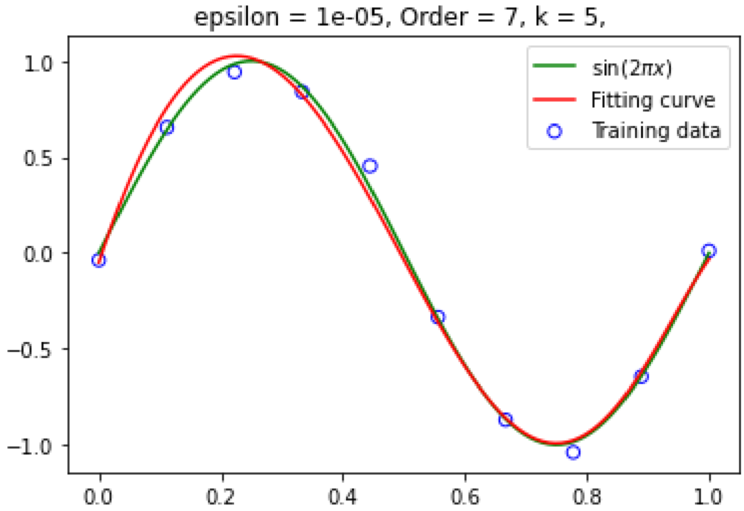
epsilon = 1e-06, Order = 3, k = 4,
sin(2πx)
Fitting curve
Training data
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

- 阶数为 5：

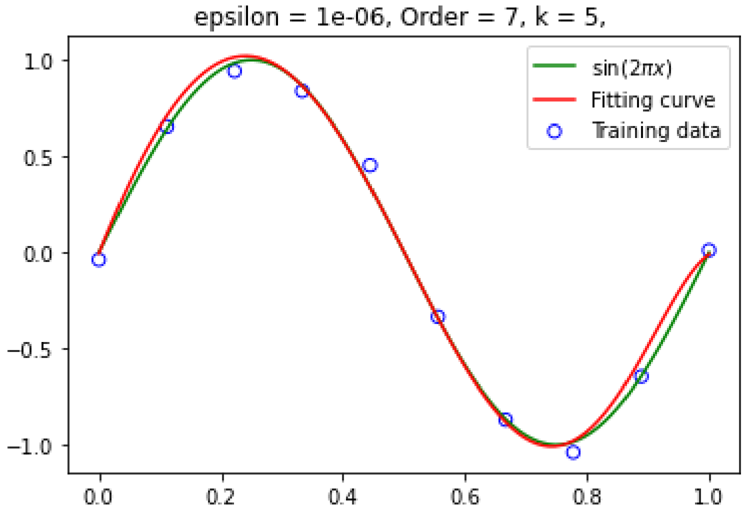

epsilon = 1e-05, Order = 5, k = 5,
sin(2πx)
Fitting curve
Training data
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0


epsilon = 1e-06, Order = 5, k = 5,
sin(2πx)
Fitting curve
Training data
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

- 阶数为 7:

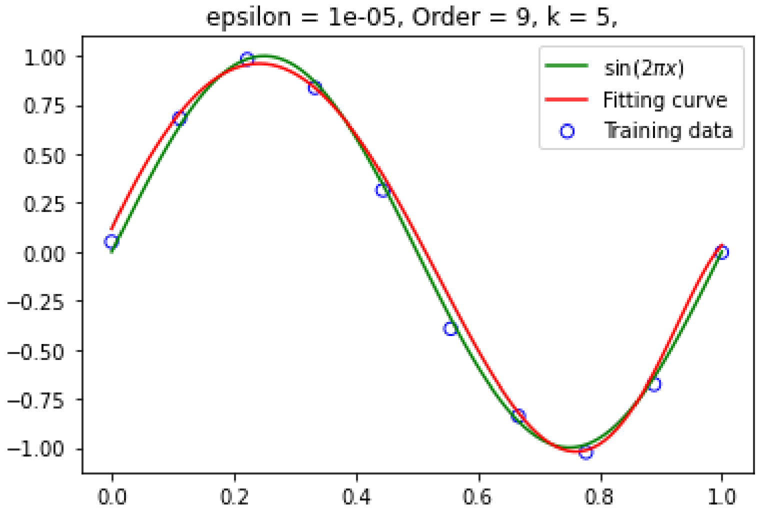


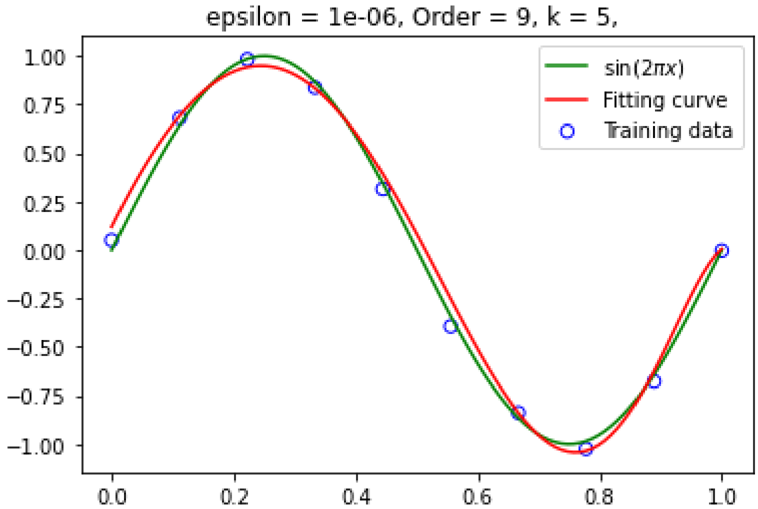
epsilon = 1e-05, Order = 7, k = 5,
sin(2πx)
Fitting curve
Training data
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0



epsilon = 1e-06, Order = 7, k = 5,
sin(2πx)
Fitting curve
Training data
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

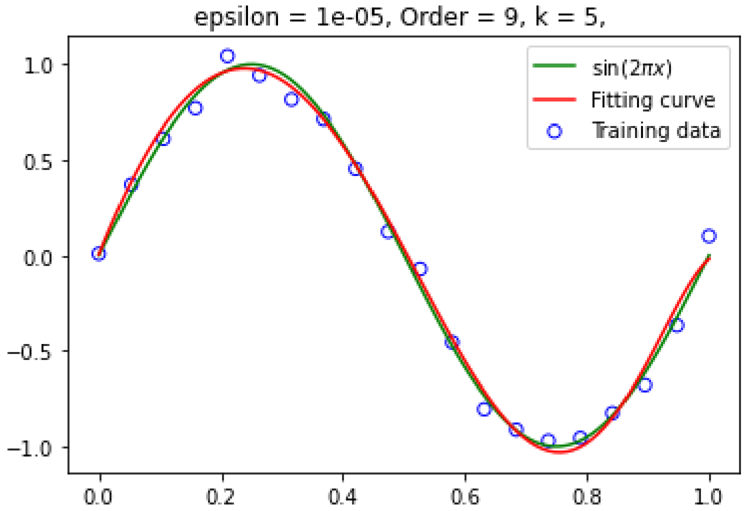
- 阶数为 9：



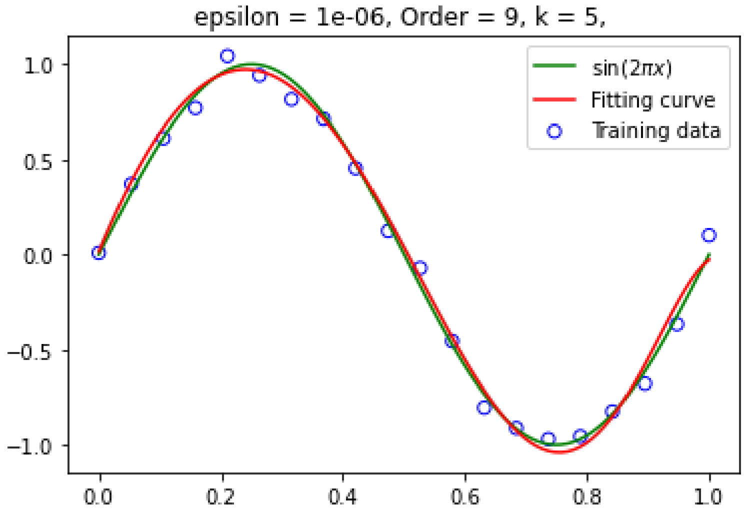


设定阶数为 9，在不同训练集大小下拟合函数

- 训练集大小为 20：

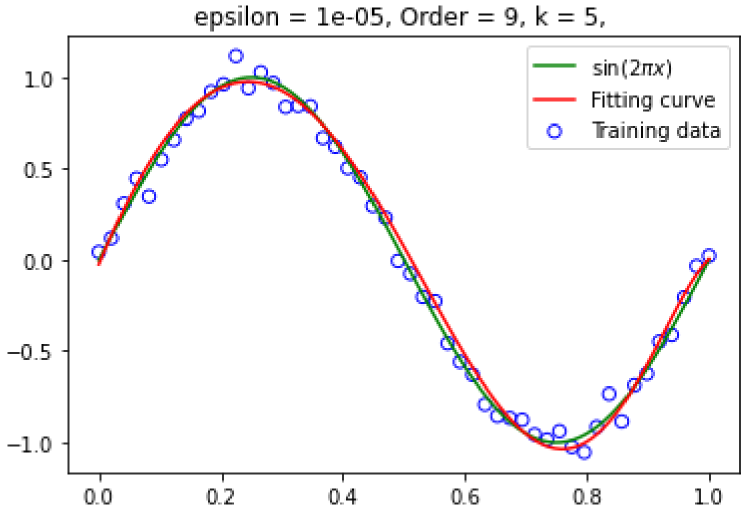


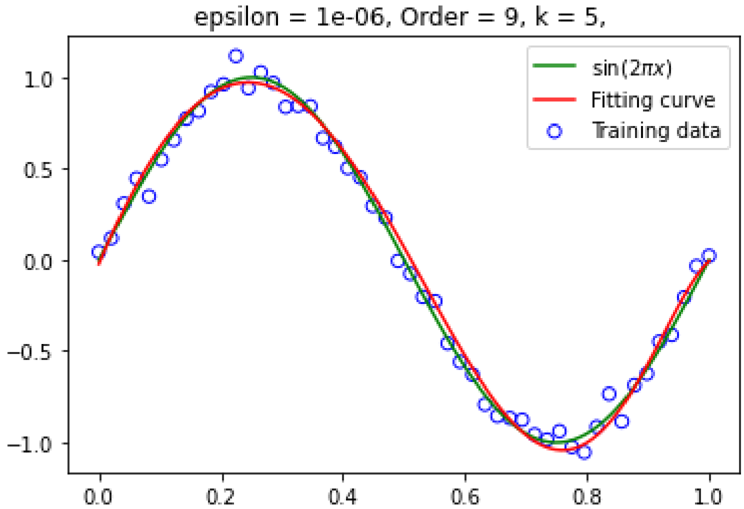
epsilon = 1e-05, Order = 9, k = 5,
sin(2πx)
Fitting curve
Training data
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0



epsilon = 1e-06, Order = 9, k = 5,
sin(2πx)
Fitting curve
Training data
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

- 训练集大小为 50：





从上面的图可以看出，同条件下精度为 10^{-6} 的右图的拟合效果并未显著强于精度为 10^{-5}下的拟合效果。

共轭梯度法的迭代次数受精度、多项式阶、训练集的大小的影响并不大，一直在 $(0, 10)$

五、结论

- 在对正弦函数的多项式拟合中，不加惩罚项时，多项式的次数越高，拟合得越好，阶数较高时出现过拟合现象，是由于样本数量少但模型能力强，模型拟合结果过分依赖数据集，这种强拟合能力可能无法拟合出正弦曲线的效果。所以增大数据集可以有效地解决过拟合问题。
- 加入惩罚项后，过拟合现象得到改善。加入惩罚项可以有效地降低参数的绝对值，从而使模型复杂度与问题匹配。所以对于训练样本限制较多的问题，增加惩罚项是解决过拟合问题的有效手段。
- 在使用梯度下降时，由于我们的目标函数是二次的，只有一个极值点，即最值点，所以梯度下降的初值选取并不很重要。如果梯度下降步长设置的比较大，那么下降结果将在目标函数最值附近逐渐向上跳动，从而无法收敛。
- 梯度下降相比共轭梯度收敛速度很慢，迭代次数很大，而共轭梯度的稳定性较差，更容易出现过拟合现象，但对于数据量较大复杂度较高的情况，共轭梯度显然要比梯度下降来的更优。

六、参考文献

《模式识别与机器学习》

《机器学习》

[矩阵求导、几种重要的矩阵及常用的矩阵求导公式]

[梯度下降算法推导](#)

[梯度下降算法原理讲解——机器学习](#)

七、附录:源代码(带注释)

```
############ 初始数据 ###############

import numpy as np
import matplotlib.pyplot as plt

def sin_func(x):
    return np.sin(2*np.pi*x)  # 返回初始正弦函数

def poly_func(W,x):  #多项式函数
    return x @ W


def get_point(Data_amount):
    x = np.linspace(0,1,Data_amount)  # 取Data_amount个样本
    y = sin_func(x)+np.random.normal(scale=0.05, size=x.shape)  # 加上高斯噪声
    return x, y

def get_param(Data_amount, Order):
    x_train = np.linspace(0,1,Data_amount)  # 取Data_amount个样本
    T = (sin_func(x_train)+np.random.normal(scale=0.05, size=x_train.shape)).T
# 加上高斯噪声
    X = []
    for i in range(0,Data_amount):
        x = [1.]
        for j in range(Order):
            x.append(x[-1] * x_train[i])  # x_i = [1  x_i  x_i^2  ...  x_i^M]
        X.append(x)  # X = [X_1  X_2  ...  X_i  ...  X_N]
    X = np.array(X)
    #print(X)
    return X,T

def draw_data(Data_amount):
    # sin函数
    x_sin = np.linspace(0,1,100)
    y_sin = sin_func(x_sin)
    # Data_amount个样本
    x_train, y_train = get_point(Data_amount)
    # 画图
    # plt.scatter(x_train,
y_train,edgecolors="b",facecolor="none",s=40,label="Training data")
    # plt.plot(x_sin, y_sin,c="g",label="$\sin(2\pi x)$")
    # plt.legend()
    # plt.show()
    return x_sin, y_sin, x_train, y_train #返回绘图数据
#draw_data(10)


################ 多项式拟合 #################

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

```python
from Data import *


################ 无正则项 #################
def get_poly(Data_amount, Order):
    X, T = get_param(Data_amount, Order)
    W = np.linalg.pinv(X) @ T  # 多项式函数系数 W
    #print(W.shape, T.shape)
    X_test, _ = get_param(100, Order)  # 多项式函数数据 X_test
    x = np.linspace(0,1,100)
    y = poly_func(W,X_test)
    return x, y   # 获取多项式函数x,y

def draw_poly(Data_amount):
    x_sin, y_sin, x_train, y_train = draw_data(Data_amount)
    plt.figure(figsize=(9,8))
    for i,order in enumerate([0, 1, 3, 9]):
        plt.subplot(2,2,i+1)

        x, y = get_poly(Data_amount,order)

        plt.scatter(x_train,
y_train,edgecolors="b",facecolor="none",s=40,label="Training data")
        plt.plot(x_sin, y_sin,c="g",label="$\sin(2\pi x)$")
        plt.plot(x, y,c="r",label="Fitting curve")
        plt.title("M={}".format(order))
        plt.legend()
    plt.show()

################ 有正则项 #################
def get_poly_with_penalty(Data_amount, Order, _lambda):
    X, T = get_param(Data_amount, Order)
    W = np.linalg.pinv(X.T @ X + _lambda * np.identity(X.shape[1])) @ X.T @ T  #
多项式函数系数 W

    #print(W.shape)
    X_test, _ = get_param(100, Order)  # 多项式函数数据 X_test
    x = np.linspace(0,1,100)
    y = poly_func(W,X_test)
    return x, y   # 获取多项式函数x,y


def draw_poly_with_penalty(Data_amount, _lambda):
    x_sin, y_sin, x_train, y_train = draw_data(Data_amount)
    plt.figure(figsize=(9,8))
    for i,order in enumerate([0, 1, 3, 9]):
        plt.subplot(2,2,i+1)
        print(x_train.shape)
        x, y = get_poly_with_penalty(Data_amount,order, _lambda)

        plt.scatter(x_train,
y_train,edgecolors="b",facecolor="none",s=40,label="Training data")
        plt.plot(x_sin, y_sin,c="g",label="$\sin(2\pi x)$")
        plt.plot(x, y,c="r",label="Fitting curve")
        plt.title("M={}".format(order))
        plt.legend()
    plt.show()
```

```python
#draw_poly(10)  # 无正则项
#draw_poly_with_penalty(10)   # 有正则项


############## 计算误差 #############

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

from Data import *

def cal_loss(X, W, T, _lambda):  #### 计算误差
    return 0.5 * ((X @ W - T).T @ (X @ W - T) + (10 ** _lambda) * W.T @ W)

################### 无正则项 ########################
def get_loss_with_Order():  # 阶数对损失的影响
    Data_amount = 10
    Order = []
    Loss = []
    for order in range(15):
        Order.append(order)
        X, T = get_param(Data_amount, order)
        W = np.linalg.pinv(X) @ T
        loss = 0.5 * (X @ W - T).T @ (X @ W - T)
        Loss.append(loss)

    plt.title("Loss of different order")
    plt.xlabel("Order")
    plt.ylabel("Loss")
    plt.plot(Order, Loss,c="b")
    plt.show()

def get_loss_with_DataAmount():
    Order = 10
    Data_amount = []
    Loss = []
    for amount in range(5, 20, 1) :
        Data_amount.append(amount)
        X, T = get_param(amount, Order)
        W = np.linalg.pinv(X) @ T
        loss = 0.5 * (X @ W - T).T @ (X @ W - T)
        Loss.append(loss)

    plt.title("Loss of different data amout")
    plt.xlabel("Data amount")
    plt.ylabel("Loss")
    plt.plot(Data_amount, Loss,c="b")
    plt.show()
################### 有正则项 ########################
def get_loss_with_lambda():
    Order = 10
    Data_amount = 10
    Loss = []
    ln_lambda = -30
    ln = []
    while ln_lambda <= 0:
```

```
        ln.append(ln_lambda)
        X, T = get_param(Data_amount, Order)
        W = np.linalg.pinv(X) @ T
        loss = cal_loss(X, W, T, ln_lambda)
        Loss.append(loss)
        ln_lambda += 1

    plt.title("Loss of different lambda")
    plt.xlabel("$ln\lambda$")
    plt.ylabel("Loss")
    plt.plot(ln, Loss,c="b")
    plt.show()


# get_loss_with_Order()
# get_loss_with_DataAmount()
# get_loss_with_lambda()

#################### 梯度下降 ################

import numpy as np
import matplotlib.pyplot as plt

from Data import *
from Loss import *

def cal_gradient(X, W, T, _lambda):  ####  计算梯度值
    return X.T @ X @ W - X.T @ T + (10 ** _lambda) * W

def gradient_descent(Data_amount, Order, _lambda, times, _alpha, epsilon):
    X, T = get_param(Data_amount, Order)

    new_W = np.zeros((Order + 1))   #### 初始化 W
    new_loss = abs(cal_loss(X, new_W, T, _lambda))
    k = 0
    for i in range(times):
        old_loss = new_loss
        gradient_loss = cal_gradient(X, new_W, T, _lambda)
        old_W = new_W
        new_W -= gradient_loss * _alpha   # W_i+1 = W_i - _alpha * gradient_loss
        new_loss = abs(cal_loss(X, new_W, T, _lambda))
        if old_loss < new_loss: #不下降了，说明步长过大
            new_W = old_W
            _alpha /= 2
        if old_loss - new_loss < epsilon:
            k = i
            break
    X_test, _ = get_param(100, Order)  # 多项式函数数据 X_test
    x = np.linspace(0,1,100)
    y = poly_func(new_W,X_test)
    return x, y, k

################### 共轭梯度 ################

import numpy as np
import matplotlib.pyplot as plt

from Data import *
```

```python
def conjugate_gradient(X, T, Order, _lambda, epsilon, times):
    A = np.transpose(X) @ X - (10 ** _lambda) *
np.identity(len(np.transpose(X)))
    b = np.transpose(X) @ T
    x = np.random.normal(size=(A.shape[1]))
    r_0 = A @ x - b
    p = -r_0
    k = times
    for i in range(times):
        _alpha = (r_0.T @ r_0) / (p.T @ A @ p)
        x = x + _alpha * p
        r = r_0 + _alpha * A @ p
        if (r_0.T @ r_0) < epsilon:
            k = i
            break
        _beta = r.T @ r / (r_0.T @ r_0)
        p = -r + _beta * p
        r_0 = r
    X_test, _ = get_param(100, Order)
    x_ = np.linspace(0,1,100)
    y = poly_func(x,X_test)
    return x_, y, k

############ main ############################

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

from Data import *
from AnalyticalSolution import *
from Loss import *
from GradientDescent import *
from ConjugateGradient import *


X_SIN = np.linspace(0,1,100)
Y_SIN = sin_func(X_SIN)
X_TRAIN, Y_TRAIN = get_point(10) # 10个数据


##### 训练集大小为 10 ，不同阶数下拟合 ######
Data_amount = 10
plt.figure(figsize=(14,10))
for i,order in enumerate([1, 3, 5, 9]):
    plt.subplot(2,2,i+1)
    x, y = get_poly(Data_amount,order)
    plt.scatter(X_TRAIN,
Y_TRAIN,edgecolors="b",facecolor="none",s=40,label="Training data")
    plt.plot(X_SIN,Y_SIN,c="g",label="$\sin(2\pi x)$")
    plt.plot(x, y,c="r",label="Fitting curve")
    plt.title("M={}".format(order))
    plt.legend()
plt.show()


####### 阶数为 9 ，不同训练集大小下拟合 #####
```

```python
Order = 9
plt.figure(figsize=(14,10))
for i,Data_amount in enumerate([10, 20, 50, 100]):
    plt.subplot(2,2,i+1)
    x_train, y_train = get_point(Data_amount)
    x, y = get_poly(Data_amount,Order)
    plt.scatter(x_train,
y_train,edgecolors="b",facecolor="none",s=40,label="Training data")
    plt.plot(X_SIN, Y_SIN,c="g",label="$\sin(2\pi x)$")
    plt.plot(x, y,c="r",label="Fitting curve")
    plt.title("N={}".format(Data_amount))
    plt.legend()
plt.show()


def cal_W(X, T, _lambda):
    return np.linalg.pinv(X.T @ X + (10 ** _lambda) * np.identity(X.shape[1])) @
X.T @ T

## 训练集为 10，验证集为 20 ###
Data_amount = 10
Validation = 100
Order = 9
X, T = get_param(Data_amount, Order)
###### 验证集 无噪声 #######   获得 X_val, Y_val
x_train = np.linspace(0,1,Validation)
Y_val = (sin_func(x_train)).T
X_val = []
for i in range(0,Validation):
    x = [1.]
    for j in range(Order):
        x.append(x[-1] * x_train[i])  # X_i = [1  x_i  x_i^2  ...  x_i^M]
    X_val.append(x)  # X = [X_1  X_2  ...  X_i  ...  X_N]
X_val = np.array(X_val)
##########################
Lambda = []
Loss = []
Loss_val = []
for _lambda in range(-20, 0):
    Lambda.append(_lambda)
    W = cal_W(X, T, _lambda)
    loss = cal_loss(X, W, T, _lambda)
    loss_val = cal_loss(X_val, W, Y_val, _lambda)
    Loss.append(loss)
    Loss_val.append(loss_val)
plt.title("Loss of different $\lambda$")
plt.xlabel("$ln\lambda$")
plt.ylabel("Loss")
plt.plot(Lambda, Loss,c="b",label = "Train")
plt.plot(Lambda, Loss_val,c="r",label = "Validation")
plt.legend()
plt.show()

Data_amount = 10
Order = 9
_lambda = 1e-6
x, y = get_poly(Data_amount,Order)
x_penalty, y_penalty = get_poly_with_penalty(Data_amount, Order, _lambda)
```

```python
plt.scatter(X_TRAIN,
Y_TRAIN,edgecolors="b",facecolor="none",s=40,label="Training data")
plt.plot(X_SIN, Y_SIN,c="g",label="$\sin(2\pi x)$")

plt.plot(x, y,c="r",label="Fitting curve")
plt.plot(x_penalty, y_penalty,c="b",label="Fitting curve with penalty")

plt.legend()
plt.show()

Data_amount = 20
_lambda = -6
_alpha = 0.01
times = 100
epsilon_1 = 1e-5 # 精度
epsilon_2 = 1e-6
Order = 9
X, T = get_param(Data_amount, Order)
x_train, y_train = get_point(Data_amount)
x, y, k = conjugate_gradient(X, T, Order, _lambda, epsilon_1,times)
plt.scatter(x_train,
y_train,edgecolors="b",facecolor="none",s=40,label="Training data")
plt.plot(X_SIN,Y_SIN,c="g",label="$\sin(2\pi x)$")
plt.plot(x, y,c="r",label="Fitting curve")
plt.title("epsilon = {}, Order = {}, k = {}, ".format(epsilon_1,Order,k))
plt.legend()
plt.show()
######################
x, y, k = conjugate_gradient(X, T, Order, _lambda, epsilon_2,times)
plt.scatter(x_train,
y_train,edgecolors="b",facecolor="none",s=40,label="Training data")
plt.plot(X_SIN,Y_SIN,c="g",label="$\sin(2\pi x)$")
plt.plot(x, y,c="r",label="Fitting curve")
plt.title("epsilon = {}, Order = {}, k = {}, ".format(epsilon_2,Order,k))
plt.legend()
plt.show()
```