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： 多项式拟合曲线

学号：1190200208
姓名：李旻翀

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. 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。

2. 实验环境

VS code 2021 + Python

三、设计思想（本程序中的用到的主要算法及数据结构）

多项式拟合曲线，即给定一些数据点，用一个多项式尽可能好的拟合出这些点排布的轨迹，并给出解析解。

接下来针对每个部分分别阐述设计思想。

1. 生成数据，加入噪声；

在这次实验中，我采用$y=\sin(2\pi x),\ x\in[0,1]$作为需要拟合的曲线。

为生成的数据点添加$\mu=0,\sigma=0.1$的高斯噪声，最终生成训练集包括 N

个样本点：$(x_1,t_1),\ (x_2,t_2),\ \ldots,\ (x_n,t_n)$。

2. 用高阶多项式函数拟合曲线；

已知训练集的 N 个样本点$(x_1,y_1),\ (x_2,y_2),\ \ldots,\ (x_n,y_n)$，高阶多项式函数的表达式为：

$$y(x,\boldsymbol{w})=\sum_{i=0}^{m}w_i x^i$$

作为拟合曲线的标准，我们采用 loss 函数作为衡量拟合曲线好坏的标准。loss 函数的形式如下：

$$E(\boldsymbol{w})=\frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\boldsymbol{w})-t_n\}^2$$

构造如下向量和矩阵：

$$w=\begin{pmatrix}w_1\\w_2\\\cdots\\w_n\end{pmatrix}，\ \text{表示多项式系数的列向量}$$

$$T=\begin{pmatrix}t_1\\t_2\\\cdots\\t_n\end{pmatrix}，\ \text{表示样本点纵坐标的列向量}$$

$$X=\begin{pmatrix}1 & {x_1}^1 & \cdots & {x_1}^n\\1 & {x_2}^1 & \cdots & {x_2}^n\\\vdots & \vdots & \vdots & \vdots\\1 & {x_n}^1 & \cdots & {x_n}^n\end{pmatrix},\ \ x_i\text{构成的范德蒙矩阵}$$

则 loss 函数可以写成如下的矩阵形式：

$$E(\boldsymbol{w})=\frac{1}{2}(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})^T(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})$$

另外，带有正则项的 loss 函数的形式为：

$$\widetilde{E}(\boldsymbol{w})=\frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\boldsymbol{w})-t_n\}^2+\frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

其矩阵形式为：

$$\widetilde{E}(\boldsymbol{w})=\frac{1}{2}[(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})^T(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})+\lambda\boldsymbol{w}^T\boldsymbol{w}]$$

其中的 λ 为超参数，需要人工设定。

3. 用解析解求解两种 loss 的最优解（无正则项和有正则项）

求 loss 函数的最优解，即求w，使得$\dfrac{\partial E}{\partial w}$（$\dfrac{\partial \widetilde{E}}{\partial w}$）为 0，此时 loss 函数得以取到最小值，对应的w即为最小值。下面分别阐述用解析解求解两种 loss 的最优解的方法。

1) 无正则项

根据 loss 函数的矩阵形式：

$$E(\boldsymbol{w})=\frac{1}{2}(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})^T(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{T})$$

我们可以将其展开为如下形式：

$$E(\boldsymbol{w})=\frac{1}{2}(\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}-2\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{T}+\boldsymbol{T}^T\boldsymbol{T})$$

（注意，因为$E(w)$是标量，所以$w^TX^TT=T^TXw$，可以合并）

展开式对w求导，可以得到：

$$\frac{\partial E}{\partial \boldsymbol{w}} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{T}$$

令$\dfrac{\partial E}{\partial w}=0$，可以最终解得：

$$\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{T} = \boldsymbol{X}^{-1}\boldsymbol{T}$$

这便是最小二乘法求得的无正则项的解析解。

2) 有正则项

根据有正则项的 loss 函数形式，同样对$\widetilde{E}(w)$进行求导可得：

$$\frac{\partial \widetilde{E}}{\partial \boldsymbol{w}} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{T} + \lambda\boldsymbol{w}$$

令$\dfrac{\partial \widetilde{E}}{\partial w}=0$，可以最终解得：

$$\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{T}$$

4. 优化方法求解最优解（梯度下降，共轭梯度）；

1) 梯度下降

在多变量函数中，梯度的方向代表着函数在给定点处上升最快的方向；梯度的反方向就是给定点的下降最快的方向。由此，我们可以给定一组初始的w，不断地寻找其下降最快的方向，由此找到函数最小值对应的点，这便是梯度下降的原理[1]。

对于多项式拟合函数的具体问题而言，loss 函数对应的梯度为：

$$\frac{\partial E}{\partial \boldsymbol{w}} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{T}$$

记$\dfrac{\partial E}{\partial w}$为$\nabla E(w)$。梯度下降法的迭代式为：

$$w_{i+1} = w_i - \alpha \nabla E(w_i)$$

其中α为学习率，需要人工设定。α过小则会造成迭代次数过多，得到最终结果的速度非常慢；而α过大则可能导致无法收敛。

我们可以设定一个小值ϵ，当前一次计算出的$E(w_{i-1})$与后一次计算出的$E(w_i)$之差的绝对值小于ϵ时，停止迭代，认为其已经找到了极小值。

另外，若后一次迭代比前一次迭代的 loss 值更大，则将学习率减半。

2) 共轭梯度

共轭梯度发在解空间的每一维分别求解最优解，每一维上的求解不会影响到其他维，对于 n 维空间的函数，最多迭代 n 次即可得到结果[2]。

具体到方法来讲，对于第 k 步的残差$r_k = b - AX_k$，根据残差，构造下一步的搜索方向p_k。具体的操作方法是：

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

$$p_{k+1} = r_{k+1} + \beta_{k+1} p_k$$

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

5. **用你得到的实验数据，解释过拟合。**
   具体对于多项式拟合正弦函数，过拟合是指拟合函数阶数过高，样本点过少时，拟合函数可以经过所有训练集上的点，但因此导致函数的波动较大，对真正需拟合函数拟合效果并不好的情况。
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。**

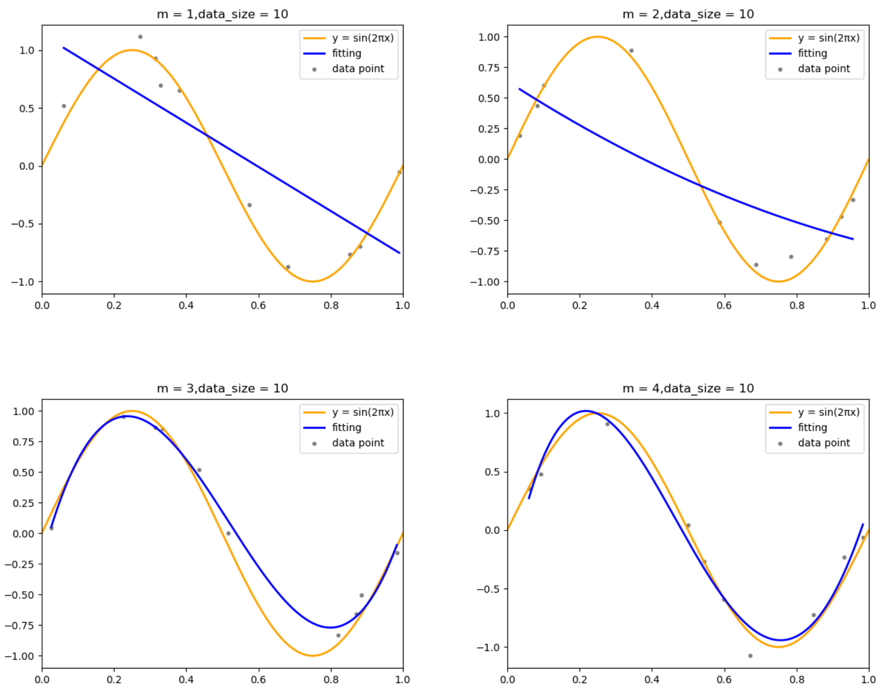
   对 5.6.7 的具体展示放在**四、实验结果与分析**中

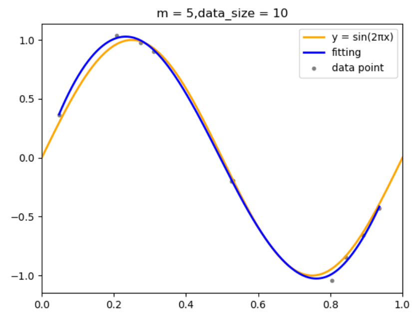
四、实验结果与分析

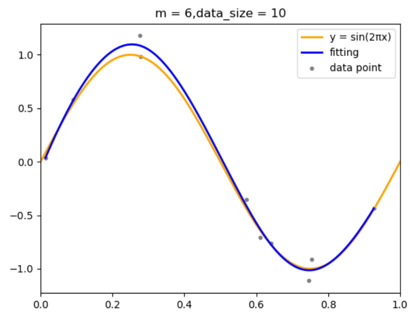
1. 最小二乘法求解析解

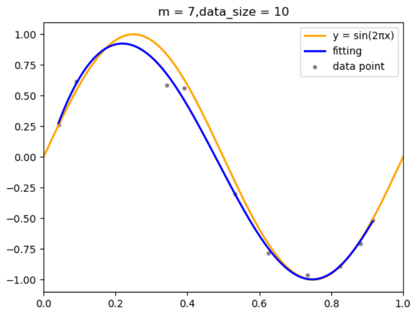
1） 固定训练集大小，改变阶数

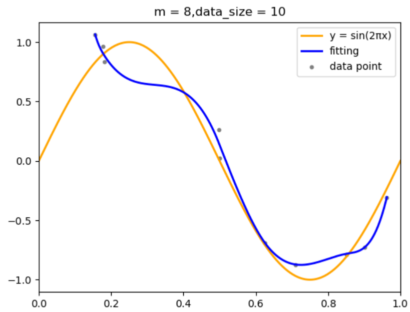
保持$datasize\ =\ 10$，改变阶数从 1 升至 9，拟合曲线如下：

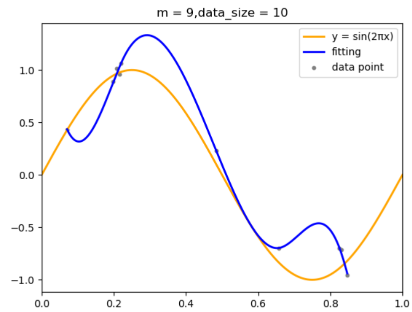










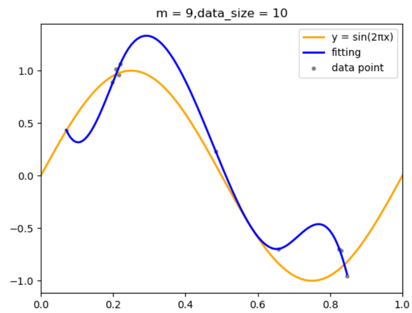


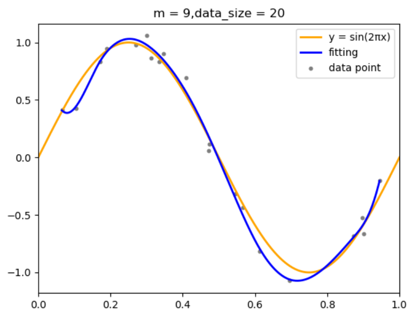
可以得出以下结论：

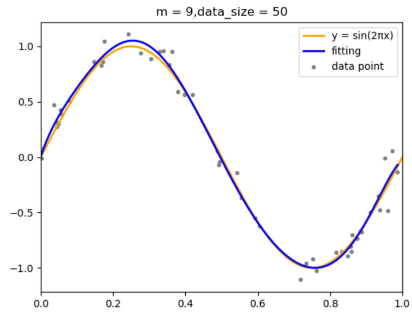
- 阶数很小时，模型的复杂度低，拟合效果并不好，不能很好地拟合真实曲线。
- 阶数适当时，模型的复杂度有所提高，拟合效果较好。此时模型泛化能力增强，能够较好地拟合真实曲线。
- 阶数过大时，模型可以很完美地经过样本点，但在样本点之外的区域与真实曲线差异过大，发生了过拟合现象。

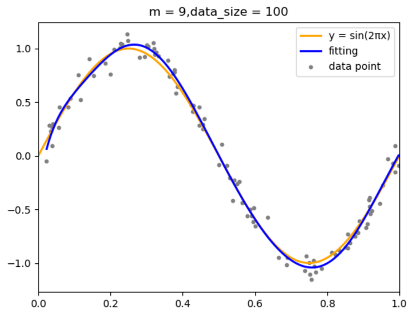
2） 固定阶数，改变训练集大小

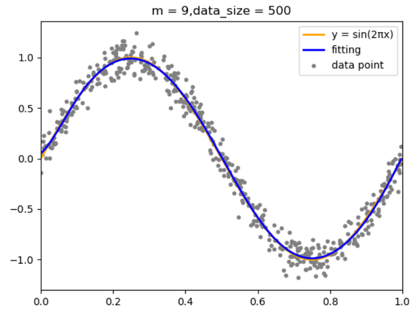
固定阶数为 9，改变训练集大小，结果如下：











不难看出，随着训练集样本数的增大，拟合曲线与真实曲线越发贴合。

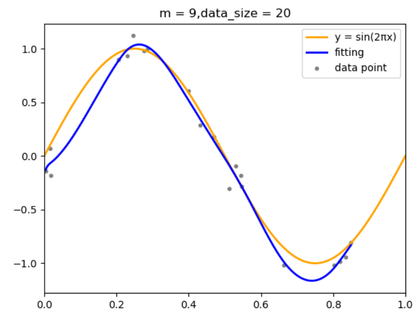
3） 对过拟合的说明

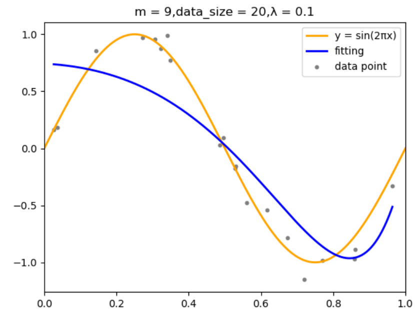
过拟合是指拟合函数阶数过高，样本点过少时，拟合函数可以经过所有训练集上的点，但因此导致函数起伏较大，对真实函数拟合效果反而不好的情况。

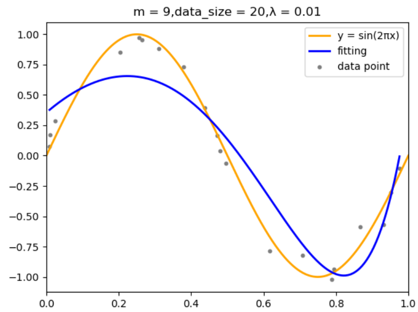
由上述说明可以看出，过拟合可以通过选择合适的拟合函数阶数与增大样本数来解决。

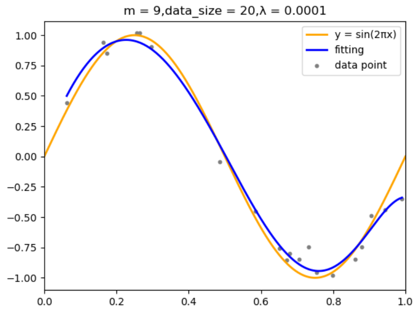
2. 最小二乘法求解析解（带正则项）

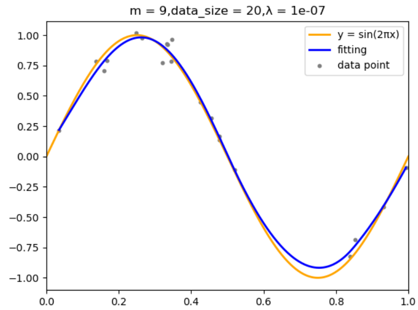
选取阶数为 9，样本数为 20 的情况，观察有无正则项对拟合函数带来的效果：

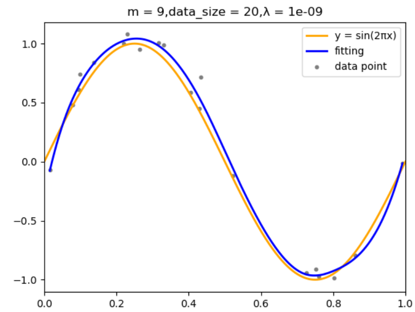


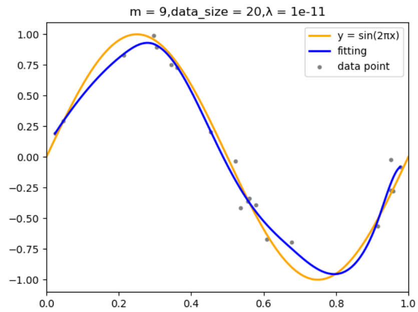










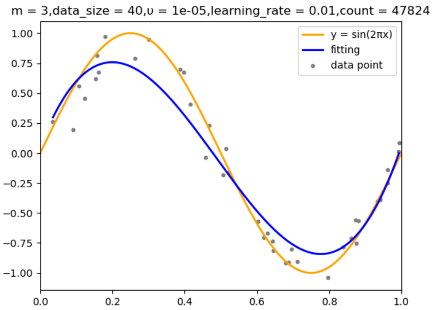


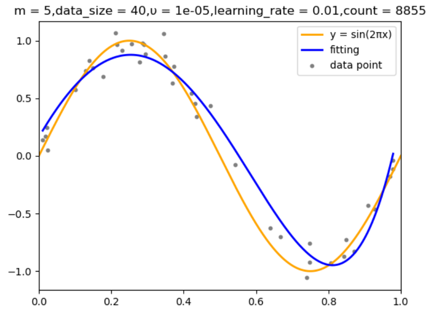
可以看出，当λ取值合适时，正则项可以提高模型的泛化能力，合适的超参数λ可以使拟合函数更接近真实值，loss 函数更小。通过上述实验，我们可以选取10^{-7}作为λ，取得较好的泛化效果。

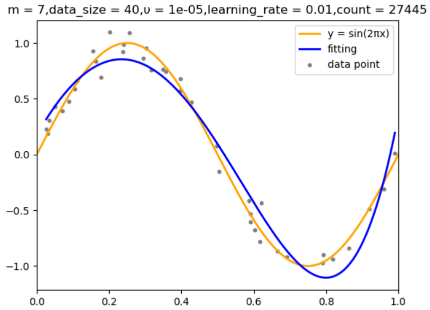
3. 梯度下降求优化解

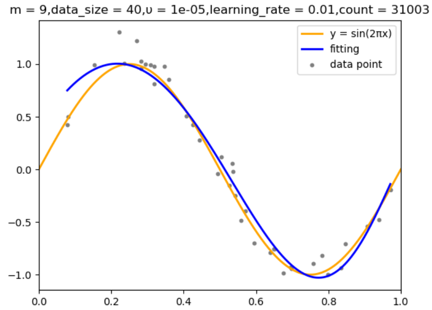
- 固定误差值、学习率，观察阶数的影响

固定误差为 1e-5，学习率 1e-2，观察阶数造成的影响：





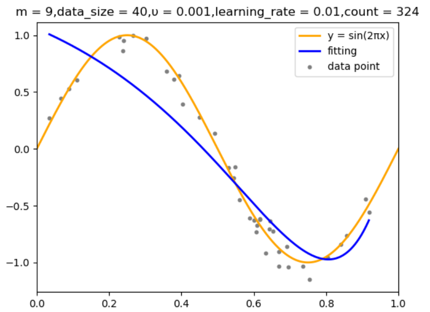


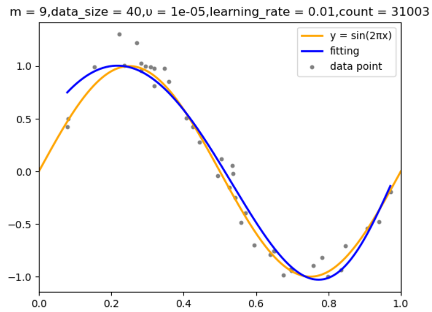


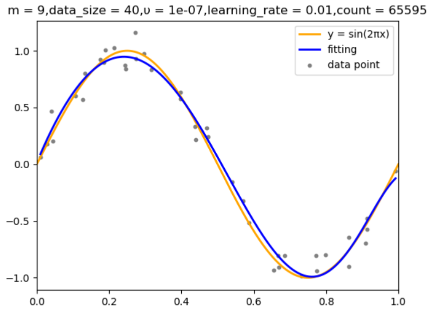
可以看出，在一定范围内，阶数越高（越合适），拟合效果越好。

- 固定阶数、学习率，观察误差值的影响

固定阶数为 9，学习率 1e-2，观察误差值造成的影响：



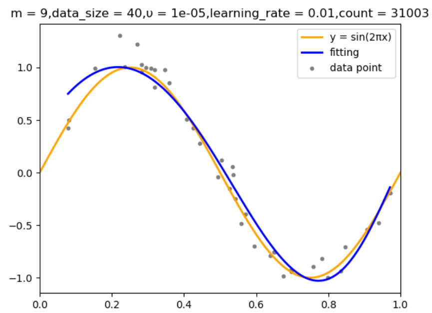


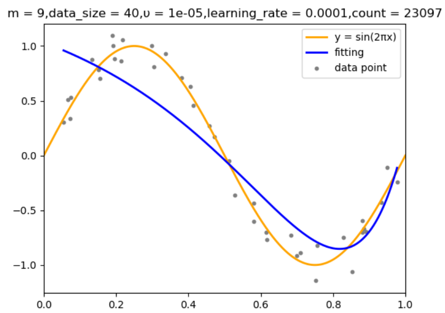


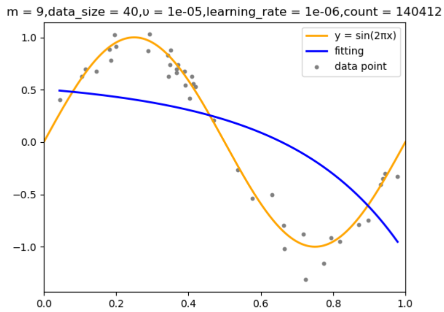
可以观察出，误差值设定越小，迭代次数越多，拟合效果越好。

- 固定阶数、误差值，观察学习率的影响

固定阶数为 9，误差为 1e-5,观察学习率造成的影响：



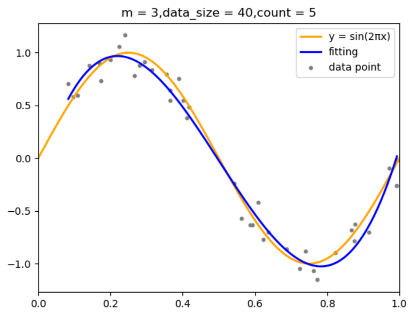


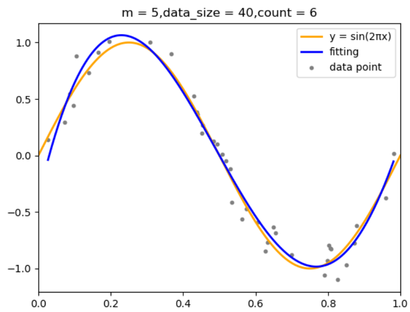


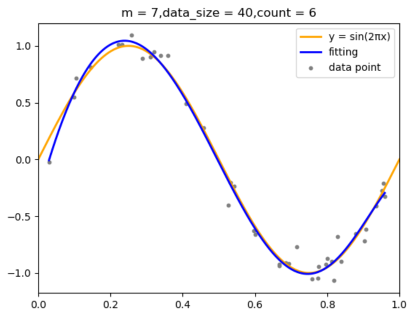
可以观察出，学习率设置也对拟合效果有影响。

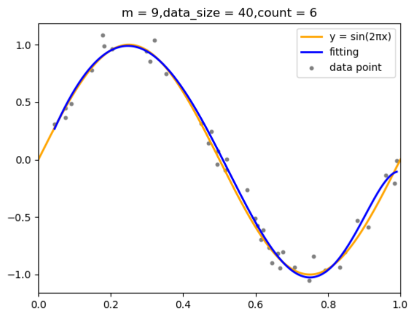
4. 共轭梯度法求优化解

设置λ为 1e-4，观察共轭梯度法的效果：









可以看出，共轭梯度法的迭代次数大大减少，也能取得很好的拟合效果。

五、结论

在多项式拟合正弦函数的问题中，多项式的次数越高，其拟合能力越强，但

在不加正则项的情况下，过高的次数与过小的样本点会出现过拟合的情况。

对于过拟合的情况，我们可以采取两种解决方法：增大数据集或者调整拟合多项式的次数。经过验证，这两种方法都可以有效解决过拟合的问题。

除此之外，还可以考虑在拟合函数中增加正则项来解决过拟合的问题。加入参数的正则项后，过拟合现象得到明显改善，对于训练样本数有限的情况，可以考虑增加正则项来解决过拟合的问题。

在使用梯度下降时。给定一组初始的w，不断地寻找其下降最快的方向，由此找到函数最小值对应的点。需要注意的是，学习率需要人工设定为合适的值。过小会造成迭代次数过多，过大则可能导致无法收敛。另外，若后一次迭代比前一次迭代的 loss 值更大，则将学习率减半。

梯度下降的缺点是收敛速度很慢，在题目所设定的场景中，迭代次数往往需要上万次。而共轭梯度法的迭代次数大大减少，仅需几次迭代便可迅速得到拟合结果。

六、参考文献

[1] 博客园.梯度下降（Gradient Descent）小结[EB/OL].2016[2021-10-2].https://www.cnblogs.com/pinard/p/5970503.html.

[2] CSDN.共轭梯度（CG）算法[EB/OL].2017[2021-10-4].https://blog.csdn.net/lusongno1/article/details/78550803.

七、附录：源代码（带注释）

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

from scipy.interpolate import make_interp_spline

# 全局变量
sigma = 0.1  # 生成数据的标准差
data_size = 40  # 数据点数量
lambda_ML = 1e-11  # 超参数 λ
m = 7  # 拟合多项式的阶数
learning_rate = 1e-6  # 学习率
epsilon = 1e-5 # 允许的误差值


# 产生带0均值高斯噪声的数据
def generate_data():
    x = np.random.random(data_size)
    x = np.sort(x)
    noise = np.random.normal(0, sigma, data_size)
    y = np.sin(2 * np.pi * x) + noise
    x = x.reshape(data_size, 1)
```

```python
    y = y.reshape(data_size, 1)
    return x, y


# 构造 x 的 Vandermonde 矩阵（横着的）
def generate_vandermonde(x):
    van = np.ones((x.shape[0], 1))
    for i in range(1, m + 1):
        van = np.hstack((van, x**i))
    return van


# 最小二乘法求解 w
def least_square(van, y):
    w = np.linalg.inv((van.T @ van)) @ van.T @ y
    return w


# 带正则项的最小二乘法求解 w
def least_square_regular(van, y):
    w = np.linalg.inv(
        (van.T @ van + np.eye(van.T.shape[0]) * lambda_ML)) @ van.T @ y
    return w


# 确定最佳的超参数λ
def RMS(van, y):
    ln_lambda = np.linspace(0, 10, 101).reshape(1, 101).T
    E_RMS = np.empty((1, 101))
    for i in range(0, 101):
        real_lambda = np.log10(1.0 / ln_lambda[i])
        w = np.linalg.inv(
            van.T @ van + np.eye(van.T.shape[0]) * real_lambda) @ van.T
 @ y
        Ew = 0.5 * (van @ w - y).T @ (van @ w - y)
        E_RMS[0, i] = (np.sqrt(2 * Ew / data_size))
    E_RMS = E_RMS.T
    # 绘图
    plt.figure(2)
    plt.xlim(0, 10)
    plt.ylim(0, 1)
    plt.plot(ln_lambda, E_RMS, c='orange', linewidth=2,
             label='E_RMS')
    plt.xlabel("10^(-x)")
    plt.ylabel("E_RMS")
```

```python
    plt.legend()
    plt.show()
    return 0


# 损失函数
def loss(van, y, w):
    diff = van @ w - y
    loss = 0.5 * diff.T @ diff
    return loss


# 梯度函数
def gradient_function(van, y, w):
    grad = van.T @ van @ w - van.T @ y + lambda_ML * w
    return grad


# 梯度下降法
def gradient_descent(van, y, learning_rate):
    w = np.zeros((m + 1, 1))
    grad = gradient_function(van, y, w)
    loss0 = 0
    loss1 = loss(van, y, w)
    count = 0
    xw = np.linspace(0, 1, 1000)
    while abs(loss1 - loss0) > epsilon:  # 误差值，小于该值时停止迭代
        count += 1
        w = w - learning_rate * grad
        loss0 = loss1
        loss1 = loss(van, y, w)
        if(loss1 > loss0):  # loss 不降反增，则减半学习率
            learning_rate *= 0.5
        grad = gradient_function(van, y, w)
        print(count)
    return w, count


# 共轭梯度法
def conjugate_gradient(van, y):
    c_lambda = 1e-4
    A = van.T @ van + c_lambda
    b = van.T @ y
    w = np.zeros((van.shape[1], 1))
    r = b
```

```python
    p = b
    count = 0
    while True:
        count += 1
        if r.T @ r < epsilon:
            break
        norm = r.T @ r
        a = norm / (p.T @ A @ p)
        w = w + a * p
        r = r - (a * A @ p)
        b = (r.T @ r) / norm
        p = r + b * p
    return w, count

# 根据已经求得的w 计算实际的 x, y 坐标值


def fitting(van, w):
    xw = van[:, 1]
    yw = van @ w
    return xw, yw


x, y = generate_data()
# print(x)
# print(y.shape)
van = generate_vandermonde(x)
# print(van)
# print(van.shape)

method = ['least_square', 'least_square_regular',
          'gradient_descent', 'conjugate_gradient']

choose_method = method[3]  # 选择方法

count = 0

if choose_method == "least_square":
    w = least_square(van, y)
elif choose_method == "least_square_regular":
    w = least_square_regular(van, y)
elif choose_method == "gradient_descent":
    w, count = gradient_descent(van, y, learning_rate)
elif choose_method == "conjugate_gradient":
```

```python
    w, count = conjugate_gradient(van, y)
else:
    w = 0
# print(w)
# print(w.shape)

xw, yw = fitting(van, w)
# print(xw.shape)
# print(yw.shape)


# 绘图部分
plt.xlim(0, 1)
plt.scatter(x, y, c='grey', s=10, label='data point')  # 生成的数据点
temp = np.linspace(0, 1, 10000)
plt.plot(temp, np.sin(2 * np.pi * temp), c='orange',
         linewidth=2, label='y = sin(2πx)')  # 标准的 y = sin(2πx) 函数
xw_smooth = np.linspace(xw.min(), xw.max(), 300)
yw_smooth = make_interp_spline(xw, yw)(xw_smooth)
plt.plot(xw_smooth, yw_smooth, c='blue', linewidth=2,
         label='fitting')  # 拟合得到的函数（进行了平滑）
# plt.title('m = ' + str(m) + ',' + 'data_size = ' + str(data_size))
# plt.title('m = ' + str(m) + ',' + 'data_size = ' + str(data_size) + '
,' + 'λ = ' + str(lambda_ML)) # 解析解（带正则项）
# plt.title('m = ' + str(m) + ',' + 'data_size = ' + str(data_size) + '
,' + 'υ = ' + str(epsilon) + ',' + 'learning_rate = ' + str(learning_ra
te) + ',' + 'count = ' + str(count) )  # 梯度下降
plt.title('m = ' + str(m) + ',' + 'data_size = ' + str(data_size) + ','
 + 'count = ' + str(count) )  # 共轭梯度
plt.legend()
plt.show()
```