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：1190200523
姓名：石翔宇

1 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）。

2 实验要求及实验环境

2.1 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。

2.2 实验环境

Windows 11 + Python 3.7.8

3 设计思想

3.1 生成带噪声数据

考虑一个样本集 $[X, T]$ ，其中 $X = [x_1, x_2, \dots, x_n]$ 中的 x_i 均匀分布在 $[0, 1]$ 区间内。

令

$$T' = [t'_1, t'_2, \dots, t'_n] = [f(x_1), f(x_2), \dots, f(x_n)] \tag{1}$$

其中 $f(x) = \sin(2\pi x)$，称为目标函数，则 T' 为不带噪声的数据。

令

$$T = [t_1, t_2, \dots, t_n] = [t'_1 + p_{G_1}, t'_2 + p_{G_2}, \dots, t'_n + g_{G_n}] \tag{2}$$

其中 p_{G_i} 是均值为 0 ，方差为 0.2 的高斯噪声。

至此我们得到了样本集 $[X, T]$ 。

3.2 利用高阶多项式函数拟合曲线（无正则项）

我们将使用多项式函数 $y(x, M)$ 来拟合我们的样本集 $[X, T]$，多项式函数 y 定义为：

$$y(x, W) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{i=0}^{M} w_i x^i \tag{3}$$

其中 M 是多项式的阶数；$w_0, w_1, \ldots, w_M$ 是多项式的系数，记作 $W = [w_0, w_1, \ldots, w_M]^T$。尽管多项式函数 y 是关于 x 的非线性函数，但是关于 W 的线性函数。

将上式矩阵化得到：

$$y(x_k, W) = \sum_{i=0}^{M} w_i x_k^i = X_k W \tag{4}$$

其中，$X_k = [1, x_k^1, x_k^2, \ldots, x_k^M]$。

我们利用均方误差函数来对我们拟合出的多项式进行评估，误差函数如下：

$$E(W) = \frac{1}{2} \sum_{i=1}^{n} (y(x_i, W) - t_i)^2 \tag{5}$$

将上式矩阵化得到：

$$E(W) = \frac{1}{2} (XW - T)^T (XW - T) \tag{6}$$

其中，$X = [X_1, X_2, \ldots, X_n]^T = \begin{bmatrix} 1 & x_1 & \cdots & x_1^M \\ 1 & x_2 & \cdots & x_2^M \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^M \end{bmatrix}$，$T = [t_1, t_2, \ldots, t_n]^T$。

我们把 W 看作自变量，将上式对 W 求导：

$$\begin{aligned} \frac{\partial E}{\partial W} &= \frac{1}{2} \frac{\partial((XW - T)^T (XW - T))}{\partial W} \\ &= \frac{1}{2} \frac{\partial((W^T X^T - T^T)(XW - T))}{\partial W} \\ &= \frac{1}{2} \frac{\partial(W^T X^T X W - W^T X^T T - T^T X W + T^T T)}{\partial W} \\ &= \frac{1}{2} (2X^T X W - X^T T - X^T T + 0) \\ &= X^T X W - X^T T \end{aligned} \tag{7}$$

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

$$W^* = (X^T X)^{-1} X^T T = X^{-1} (X^T)^{-1} X^T T = X^{-1} T \tag{8}$$

3.3 利用高阶多项式函数拟合曲线（有正则项）

我们可以通过正则化的方式来减轻过拟合的影响，即在误差函数中加入一个惩罚项，使得多项式系数被控制在较小的范围。将均方误差函数修改为：

$$\widetilde{E}(W) = \frac{1}{2} \sum_{i=1}^{N} (y(x_i, W) - t_i)^2 + \frac{\lambda}{2} ||W||^2 \tag{9}$$

其中 $||W||^2 = W^T W = w_0^2 + w_1^2 + \cdots + w_M^2$，$\lambda$ 为正则项的平衡系数。

将上式矩阵化得到：

$$\widetilde{E}(W) = \frac{1}{2} ((XW - T)^T (XW - T) + \lambda W^T W) \tag{10}$$

我们把 W 看作自变量，将上式对 W 求导：

$$\begin{aligned}\frac{\partial \widetilde{E}}{\partial W} &= \frac{1}{2}\frac{\partial((XW-T)^T(XW-T)+\lambda W^TW)}{\partial W} \\ &= \frac{1}{2}\frac{\partial(W^TX^TXW-W^TX^TT-T^TXW+T^TT+\lambda W^TW)}{\partial W} \\ &= \frac{1}{2}(2X^TXW-X^TT-X^TT+0+2\lambda W) \\ &= X^TXW-X^TT+\lambda W\end{aligned} \tag{11}$$

令 $\frac{\partial \widetilde{E}}{\partial W}=0$ 可得到 W^*，其中 I 为单位阵：

$$W^*=(X^TX+\lambda I)^{-1}X^TT \tag{12}$$

3.4 梯度下降法求解最优解

考虑一个连续可微函数 $f(x)$，从 x_0 出发想找到局部最低点，可以通过构造一个序列 $x_0, x_1, x_2, \ldots$ 满足 $f(x_i) < f(x_{i+1}), i = 0, 1, 2, \ldots$ ，那么我们就能够通过不断找新的 x_i 来收敛到局部最低点。

如果 $f(x)$ 在 x_i 点可微，则其在该点的梯度被记作 $\nabla f(x_i)$，梯度 $\nabla f(x_i)$ 的反方向即为 $f(x)$ 下降最快的方向。由此，每一步我们都按照

$$x_{i+1}=x_i-\alpha\nabla f(x) \tag{13}$$

来更新 x ，其中 α 被称作为学习率或者步长，满足 $\alpha > 0$ 。

现在我们将上面提到的加入正则项的均方误差函数 $\widetilde{E}(W)$ 当作考虑的 $f(x)$ 。由式11可知 $\widetilde{E}(W)$ 是可微的，且可知 $\nabla\widetilde{E}(W)=X^TXW-X^TT+\lambda W$ ，则我们可以利用上述方法，最小化 $\widetilde{E}(W)$ 从而实现对曲线的拟合。

3.5 共轭梯度法求解最优解

考虑求解一个线性方程 $Ax=b$ ，我们先额外构造一个二次函数

$$\phi(x)=\frac{1}{2}x^TAx-b^Tx \tag{14}$$

对该函数求导，并令导数为 0 可得

$$\nabla\phi(x)=Ax-b=0 \tag{15}$$

不难发现，$\phi(x)$ 的极小值点恰好是线性方程 $Ax=b$ 的解。于是我们可以把求解线性方程的问题转化为求二次函数极值的问题。

对于一组向量 $\{p_0, p_1, \ldots, p_n\}$ ，如果任意两个向量间满足

$$p_i^TAp_j=0 \tag{16}$$

则称这组向量与对称正定矩阵 A 共轭。这组向量即是共轭梯度法每次迭代的方向。那么给定任意一个对称正定矩阵，如何求取一组共轭向量？

假设起始点为 x_0 ，我们将从 $p_0=-\nabla\phi(x_0)$ 开始，推导出后续每次迭代所需的 p_k 。

在 $k=1$ 时刻，我们已知 p_0 和 $\nabla\phi(x_1)$ ，注意到这两个向量一定是线性无关的。在这两个向量构成的平面上生成新的 p_1 ，令

$$p_1=-\nabla\phi(x_1)+\beta_1p_0 \tag{17}$$

将 p_1 和 p_0 代入式16，可得 $p_1^T A p_0 = 0$ ，代入式17可得：

$$\begin{aligned}\beta_1 &= \frac{p_1 + \nabla\phi(x_1)}{p_0} \\ &= \frac{p_1^T A(p_1 + \nabla\phi(x_1))}{p_1^T A p_0} \\ &= \frac{p_1^T A \nabla\phi(x_1)}{p_1^T A p_0}\end{aligned} \tag{18}$$

推广到一般情况，

$$p_k = -\nabla\phi(x_k) + \beta_k p_{k-1} \tag{19}$$

$$\beta_k = \frac{p_k^T A \nabla\phi(x_k)}{p_k^T A p_{k-1}} \tag{20}$$

至此我们得到了一组共轭向量 $\{p_0, p_1, \ldots, p_n\}$ ，代表每次迭代的方向。

接下来计算各个方向上的最优步长 $\{\alpha_0, \alpha_1, \ldots, \alpha_n\}$ 。用 α_k 替换 $\phi(x_{k+1})$ 中的 x_{k+1} ，得到 $\phi(x_k + \alpha_k p_k)$ ，将其对 α_k 求导，并令导数为 0 可得

$$\alpha_k = -\frac{\nabla\phi(x_k)^T p_k}{p_k^T A p_k} \tag{21}$$

于是，整个共轭梯度法的流程整理如下：

$$\begin{aligned}\alpha_k &\leftarrow -\frac{r_k^T p_k}{p_k^T A p_k} \\ x_{k+1} &\leftarrow x_k + \alpha_k p_k \\ r_{k+1} &\leftarrow A x_{k+1} - b \\ \beta_{k+1} &\leftarrow \frac{r_{k+1}^T A p_k}{p_k^T A p_k} \\ p_{k+1} &\leftarrow -r_{k+1} + \beta_{k+1} p_k \\ k &\leftarrow k+1\end{aligned} \tag{22}$$

其中，用 r_k 替代了 $\nabla\phi(x_k)$ 。从 $k=0$ 开始, $r_0 = Ax_0 - b, p_0 = -r_0$ 依次迭代下去。

实际中, 为了降低计算量, 一般会对式22稍做改动, 得到下面的算法流程

$$\begin{aligned}\alpha_k &\leftarrow \frac{r_k^T r_k}{p_k^T A p_k} \\ x_{k+1} &\leftarrow x_k + \alpha_k p_k \\ r_{k+1} &\leftarrow r_k + \alpha_k A p_k \\ \beta_{k+1} &\leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\ p_{k+1} &\leftarrow -r_{k+1} + \beta_{k+1} p_k \\ k &\leftarrow k+1\end{aligned} \tag{23}$$

现在回到我们的问题上来,将上面提到的均方误差函数 $\widetilde{E}(W)$ 最小化。可知 $\nabla\widetilde{E}(W) = X^T XW - X^T T + \lambda W$ ，则我们利用上述方法，最小化 $\widetilde{E}(W)$ 从而实现对曲线的拟合。

令 $\nabla\widetilde{E}(W) = X^T XW - X^T T + \lambda W = 0$ ，不难看出上式是一个关于 W 的一阶线性方程，则 $A = X^T X + \lambda I, b = X^T T$ ，带入式23即可求解。

4 实验结果分析

4.1 利用高阶多项式函数拟合曲线（无正则项）

我们将训练样本的数目固定为 10，分别将多项式阶数设为 2、5、7、9，测试结果如图 **1**所示。

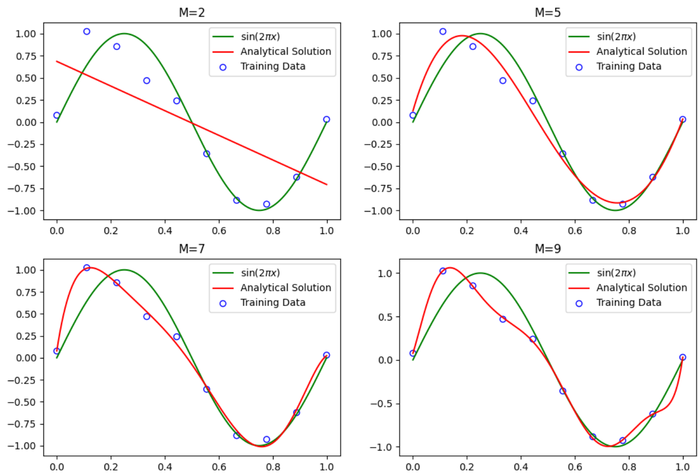

图 1

观察图 **1**，我们可以看到当阶数为 2 时，多项式函数还并不能拟合给定的曲线，这时多项式函数曲线无法经过几乎所有的训练数据点，拟合能力还很差。当阶数增加到 5 时，多项式函数就能很好地拟合了。当阶数继续增加，到 7 时，就出现了过拟合的趋势；到 9 时，多项式函数曲线几乎经过了所有的样本数据点，将所有噪声也进行了很好的拟合，表现出了过拟合的情况。

这也间接表明，多项式函数的拟和能力随着阶数的增加而增强。但不能盲目追求增强拟和能力，出现过拟合会导致模型的泛化能力下降，我们在图 **2**中展示相关数据。

图 **2**为在测试样本集上，不同阶数的高阶多项式函数拟合结果的均方误差 $E(W)$ 值。由上图可以看出，阶数过小或过大时，误差都很大；只有当阶数适中，才能够找到拟合效果较好的解。

4.2 利用高阶多项式函数拟合曲线（有正则项）

与上面相同地，我们将训练样本的数目固定为 10，将惩罚项的系数 λ 固定为 10^{-7}，分别将多项式阶数设为 2、5、7、9，测试结果如图 **3**。

我们可以看到，正则项对控制过拟合起到了积极作用。图 **4**可以更好地表现出正则项的积极作用。

下面我们将探讨惩罚项系数 λ 对拟合的影响。我们将训练样本的数目固定为 10，将多项式阶数固定为 7，分别测试 $(10^{-30}, 10^{-0})$ 不同的 λ，如图 **5**所示。

可以看出，在 λ 处于 $(10^{-30}, 10^{-8})$ 之间时，保持着一种相对较稳定的误差；$(10^{-8}, 10^{-4})$ 之间时，误差有轻微的下降，表明此时正则项权重较为合适；$(10^{-4}, 10^{-0})$ 之间时，误差急剧上升，这表明此时正则项权重太大，影响了拟合效果。

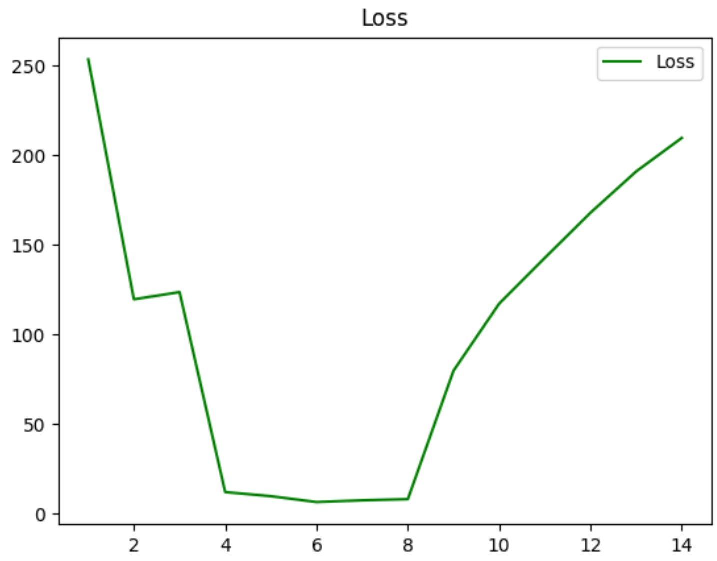

图 2

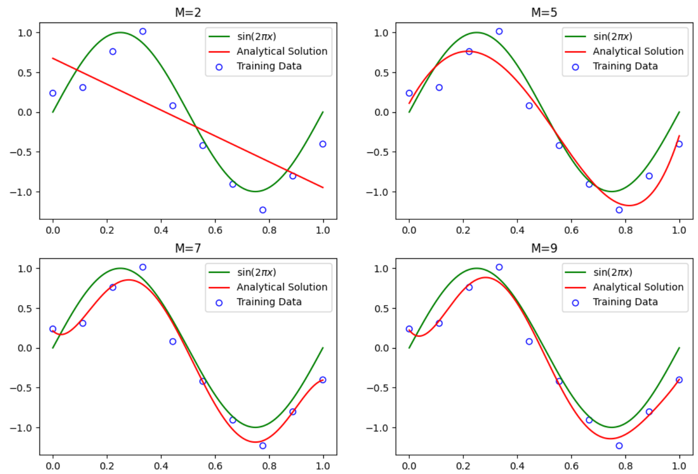

图 3

4.3 梯度下降法求解最优解

我们将训练样本的数目固定为 10 ，加入早停策略和学习率下降策略，固定学习率为 10^{-2} ，最大迭代次数为 70000 ，早停策略的 δ 为 10^{-7} ，正则化项的系数 λ 为 10^{-7} ，分别将多项式阶数设为 2 、5 、7 、9 ，测试结果如图 **6**。

图 **6**中每个子图标题处还标注了使用早停策略后实际的迭代轮数，可以看出阶数为 2 时，受拟合能力影响，效果并不好，且在很早就停止了；当阶数为 5 、7 、9 时，可以看出阶数对拟合效果影响并不大，甚至没有影响。

我们将固定阶数为 7 ,探究学习率对拟合效果的影响。我们将学习率分别设置为 $[5*10^{-5}, 10^{-4}, 5*10^{-4}, 10^{-3}, 5*10^{-3}, 10^{-2}, 5*10^{-2}, 10^{-1}]$ ，结果如图 **7**所示：

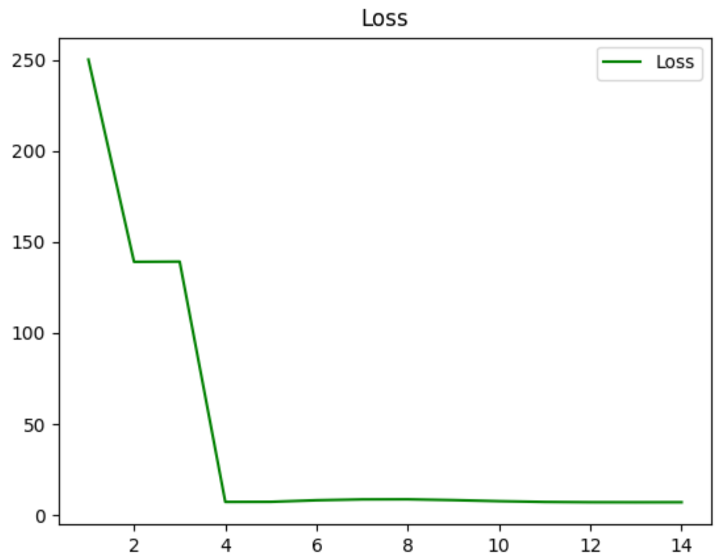

图 4

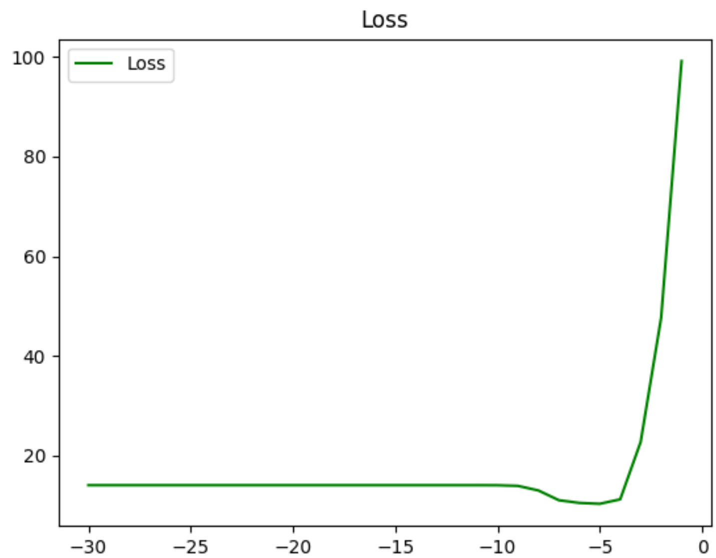

图 5

我们可以看到，在早停策略和学习率下降策略的帮助下，随着学习率的有限增大，拟合效果大致呈变好的趋势，实际的迭代轮数大致呈下降的趋势。

4.4 共轭梯度法求解最优解

我们将训练样本的数目固定为 10，加入早停策略，固定最大迭代次数为 20，早停策略的 δ 为 10^{-7}，正则化项的系数 λ 为 10^{-7}，分别将多项式阶数设为 2、5、7、9，测试结果如图 **8**。

从图 **8**可看出，当阶数为 2 时，受拟合能力影响，拟合效果并不好；当阶数大于等于 5 时，拟合效果变化并不大，但是迭代次数随着阶数的增长而增长。

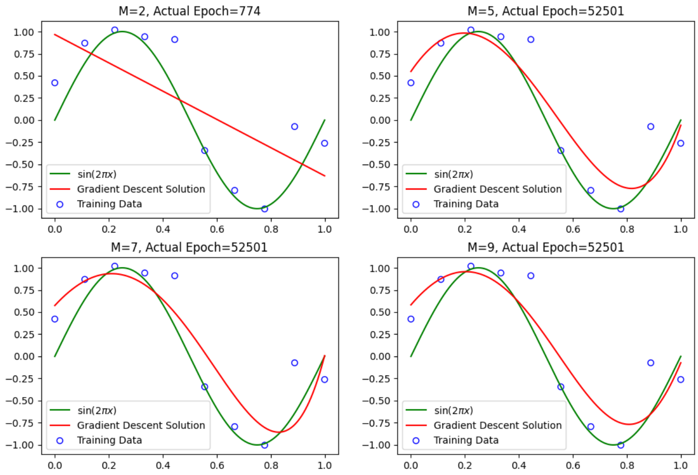
M=2, Actual Epoch=774
M=5, Actual Epoch=52501
M=7, Actual Epoch=52501
M=9, Actual Epoch=52501
sin(2πx)
Gradient Descent Solution
Training Data

图 6

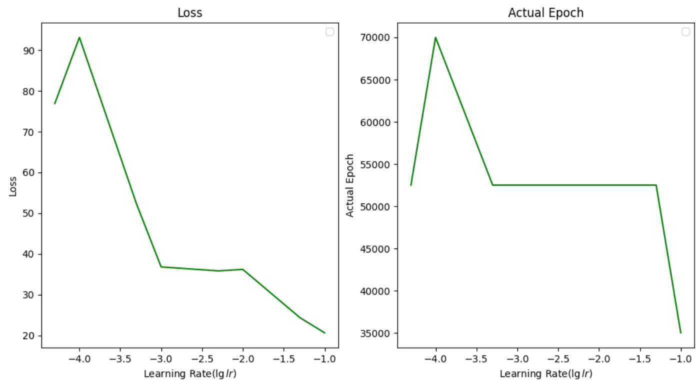
Loss
Actual Epoch
Learning Rate(lg lr)

图 7

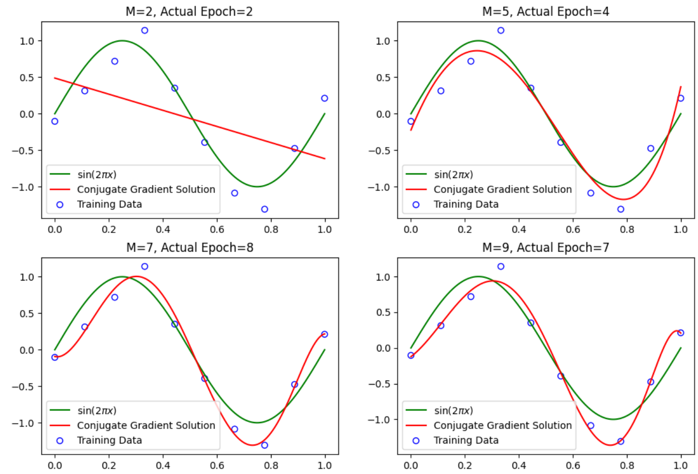
M=2, Actual Epoch=2
M=5, Actual Epoch=4
M=7, Actual Epoch=8
M=9, Actual Epoch=7
sin(2πx)
Conjugate Gradient Solution
Training Data

图 8

4.5 解析法（有正则项）、梯度下降法、共轭梯度法的比较

我们将训练样本的数目固定为 10，梯度下降法中加入早停策略和学习率下降策略，共轭梯度法中加入早停策略，早停策略的 δ 为 10^{-7}，正则化项的系数 λ 为 10^{-7}，分别将多项式阶数设为 2、5、7、9，测试结果如图 9。

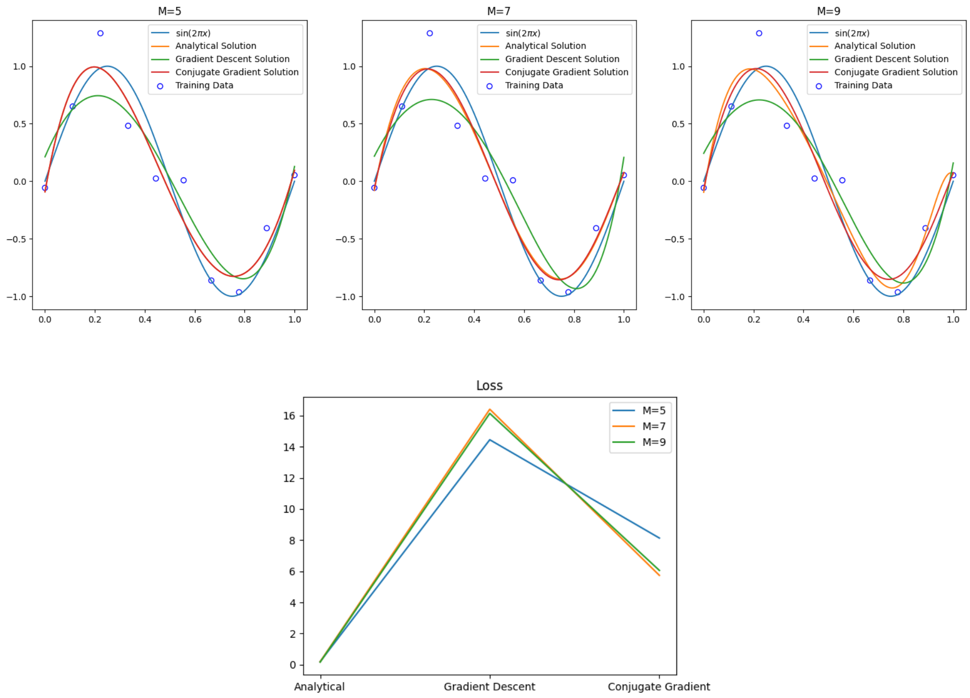

图 9

从图 9中可以看出，在上述超参数设置的情况下，三种方法的拟合效果排序为解析法（有正则项）> 梯度下降法 > 共轭梯度法。

5 结论

对于正弦函数的拟合任务，加入正则项的解析法的拟合效果优于其他两种迭代方法，代码简单，但是需要对矩阵进行求逆运算，在数据量较小时有很好的应用价值。

两种迭代方法中，共轭梯度法要优于梯度下降法，无论是迭代次数还是拟合效果，但是代码实现较为复杂。相比之下，共轭梯度法对于解决数据量较多的任务较为合适。

A 源代码（带注释）

A.1 生成带噪声数据

Listing 1: data.py

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

```python
from utils import *

def training_data_generator(x_range=(0.0, 1.0), sample_num=10, scale=0.2):
    """
    生成训练样本

    Args:
        x_range (tuple, optional): X的范围. Defaults to (0.0, 1.0).
        sample_num (int, optional): 训练样本数量. Defaults to 10.
        scale (float, optional): 噪声标准差. Defaults to 0.2.

    Returns:
        tuple: 训练样本(X, Y)
    """

    X = np.linspace(x_range[0], x_range[1], sample_num)
    Y = np.sin(2 * np.pi * X) + np.random.normal(scale=scale, size=X.shape)
    return X, Y

def testing_data_generator(x_range=(0.0, 1.0), sample_num=1000):
    """
    生成测试样本

    Args:
        x_range (tuple, optional): X的范围. Defaults to (0.0, 1.0).
        sample_num (int, optional): 测试样本数量. Defaults to 1000.

    Returns:
        tuple: 测试样本(X, Y)
    """

    X = np.linspace(x_range[0], x_range[1], sample_num)
    Y = np.sin(2 * np.pi * X)
    return X, Y

# 画图
def show(data_trian, data_test):
    plt.scatter(data_trian[0], data_trian[1], facecolor="none",  edgecolor="b",  s=50, label="training data")
    plt.plot(data_test[0], data_test[1], c="g", label="$\sin(2\pi x)$")
    plt.ylabel("y", size=20)
    plt.xlabel("x", size=20)
    plt.legend()
    plt.show()

if __name__ == '__main__':
    data_train, data_test = training_data_generator(), testing_data_generator()
    show(data_train, data_test)
```

A.2 利用高阶多项式函数拟合曲线

Listing 2: analytical solution.py

```
import numpy as np
import matplotlib.pyplot as plt
from data import *
from utils import *
import math

class analytical_solution:
    def __init__(self, degree, lambda_, data_train):
        """
        求解析解初始化

        Args:
            degree (int): 维度
            lambda_ (float): 正则项系数
            data_train (tuple)): 训练样本
        """
        self.degree = degree
        self.lambda_ = lambda_
        self.data_train = data_train
        self.data_train_X_transformed = transform(data_train[0], degree)

    def analytical_solution(self):
        """
        求不带正则项的解析解

        Returns:
            list: W
        """
        self.analytical_solution_W = np.linalg.pinv(self.data_train_X_transformed) @ self.
            data_train[1]
        return self.analytical_solution_W

    def analytical_solution_normalized(self):
        """
        求带正则项的解析解

        Returns:
            list: W
        """
        data_train_X_t_t = np.transpose(self.data_train_X_transformed)
        self.analytical_solution_normalized_W = np.linalg.pinv(data_train_X_t_t @ self.
            data_train_X_transformed + self.lambda_* np.identity(
```

```python
            len(data_train_X_t_t))) @ data_train_X_t_t @ self.data_train[1]
        return self.analytical_solution_normalized_W

    def calc_loss(self):
        """
        计算不带正则项的loss

        Returns:
            float: loss
        """
        return calc_loss(self.data_train_X_transformed, self.analytical_solution_W, self.
            data_train[1])

    def calc_loss_normalized(self):
        """
        计算带正则项的loss

        Returns:
            float: loss
        """
        return calc_loss(self.data_train_X_transformed, self.analytical_solution_normalized_W,
            self.data_train[1])


# 画图
def draw(data_train, data_test, lambda_=1e-7, normalized=False, degrees=[2, 5, 7, 9]):
    plt.figure(figsize=(12, 8))
    for i, degree in  enumerate(degrees):
        plt.subplot(2, 2, i + 1)

        ana = analytical_solution(degree, lambda_, data_train)
        if normalized:
            W = ana.analytical_solution_normalized()
        else:
            W = ana.analytical_solution()
        data_test_X_transformed = transform(data_test[0], degree)
        plt.scatter(data_train[0], data_train[1], facecolor="none", edgecolor="b", label="
            Training Data")
        plt.plot(data_test[0], data_test[1], "g", label="$\sin(2\pi x)$")
        plt.plot(data_test[0], np.dot(data_test_X_transformed, W), "r", label="Analytical
            Solution")
        plt.title("M={}". format(degree))
        plt.legend()
    plt.show()

# 画图
def draw_loss(data_train, data_test, lambda_=1e-7, normalized=False, degrees= range(1, 15)):
    losses = []
```

```python
    for degree in degrees:
        ana = analytical_solution(degree, lambda_, data_train)
        if normalized:
            W = ana.analytical_solution_normalized()
            loss = get_test_loss_normalized(W, data_test, lambda_)
        else:
            W = ana.analytical_solution()
            loss = get_test_loss(W, data_test)
        losses.append(loss)
    plt.plot(degrees, losses, "g", label="Loss")
    plt.title("Loss")
    plt.legend()
    plt.show()

# 画图
def draw_lambda(data_train, data_test, lambda_exp=(0, 30), normalized=True, degree=7):
    losses = []
    lambdas = [10 ** (-x) for x in  range(lambda_exp[1], lambda_exp[0], -1)]
    for lambda_ in lambdas:
        ana = analytical_solution(degree, lambda_, data_train)
        if normalized:
            W = ana.analytical_solution_normalized()
            loss = get_test_loss_normalized(W, data_test, lambda_)
        else:
            W = ana.analytical_solution()
            loss = get_test_loss(W, data_test)
        losses.append(loss)
    lambdas = [math.log10(x) for x in lambdas]
    plt.plot(lambdas, losses, "g", label="Loss")
    plt.title("Loss")
    plt.legend()
    plt.show()


if __name__ == "__main__":
    sample_num = 10
    data_train, data_test = training_data_generator(sample_num=sample_num),
        testing_data_generator()
    draw(data_train, data_test, normalized=False)
    draw_loss(data_train, data_test, normalized=False)
    draw(data_train, data_test, normalized=True)
    draw_loss(data_train, data_test, normalized=True)
    draw_lambda(data_train, data_test)
```

A.3 梯度下降法求解最优解

Listing 3: gradient descent.py

```python
import numpy as np
import matplotlib.pyplot as plt
from data import *
from utils import *
import math

class gradient_descent:
    def __init__(self, degree, data_train, lr=1e-2, epochs=70000, delta=1e-7, lambda_=1e-7):
        """
        梯度下降法初始化

        Args:
            degree (int)): 维度
            data_train (tuple): 训练样本
            lr (float), optional): 学习率. Defaults to 1e-2.
            epochs (int, optional): 最大迭代次数. Defaults to 70000.
            delta (float), optional): 早停策略的参数值. Defaults to 1e-7.
            lambda_ (float), optional): 正则项系数. Defaults to 1e-7.
        """
        self.degree = degree
        self.lambda_ = lambda_
        self.data_train = data_train
        self.lr = lr
        self.epochs = epochs
        self.delta = delta
        self.data_train_X_transformed = transform(data_train[0], degree)

    def gradient_descent(self):
        """
        梯度下降法

        Returns:
            tuple: (W, 总迭代次数)
        """
        X, T = self.data_train_X_transformed, self.data_train[1]
        W = np.random.normal(size=(X.shape[1]))
        lr_scheduler = lr_scheduler_MultiStep(self.epochs, self.lr)
        loss_last = loss = calc_loss_normalized(X, W, T, self.lambda_)
        last_epoch = self.epochs
        for epoch in  range(self.epochs):
            W = W - self.lr * calc_gradient_normalized(X, W, T, self.lambda_)
            loss = calc_loss_normalized(X, W, T, self.lambda_)
            if early_stop(loss, loss_last, self.delta):
                last_epoch = epoch
                break
            else: loss_last = loss
            self.lr = lr_scheduler.step(epoch)
```

```python
        return W, last_epoch

# 画图
def draw(data_train, data_test, lr=1e-2, epochs=70000, delta=1e-7, lambda_=1e-7, degrees=[2, 5,
     7, 9]):
    plt.figure(figsize=(12, 8))
    for i, degree in  enumerate(degrees):
        plt.subplot(2, 2, i + 1)
        gd = gradient_descent(degree, data_train, lr=lr, epochs=epochs, delta=delta, lambda_=
            lambda_)
        W, actual_epoch = gd.gradient_descent()
        data_test_X_transformed = transform(data_test[0], degree)
        plt.scatter(data_train[0], data_train[1], facecolor="none", edgecolor="b", label="
            Training Data")
        plt.plot(data_test[0], data_test[1], "g", label="$\sin(2\pi x)$")
        plt.plot(data_test[0], np.dot(data_test_X_transformed, W), "r", label="Gradient Descent
             Solution")
        plt.title("M={}, Actual Epoch={}". format(degree, actual_epoch))
        plt.legend()
    plt.show()

# 画图
def draw_loss_lr(data_train, data_test, degree=7, epochs=70000, delta=1e-7, lambda_=1e-7, lrs
    =[5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1]):
    losses, actual_epochs = [], []
    plt.figure(figsize=(12, 6))
    for lr in lrs:
        gd = gradient_descent(degree, data_train, lr=lr, epochs=epochs, delta=delta, lambda_=
            lambda_)
        W, _, actual_epoch = gd.gradient_descent()
        loss = get_test_loss_normalized(W, data_test, lambda_)
        losses.append(loss)
        actual_epochs.append(actual_epoch)
    lrs = [math.log10(x) for x in lrs]
    plt.subplot(1, 2, 1)
    plt.plot(lrs, losses, "g")
    plt.xlabel("Learning Rate($\lg lr$)")
    plt.ylabel("Loss")
    plt.title("Loss")
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(lrs, actual_epochs, "g")
    plt.xlabel("Learning Rate($\lg lr$)")
    plt.ylabel("Actual Epoch")
    plt.title("Actual Epoch")
    plt.legend()
    plt.show()
```

```python


if __name__ == "__main__":
    sample_num = 10
    data_train, data_test = training_data_generator(sample_num=sample_num),
        testing_data_generator()
    draw(data_train, data_test)
    draw_loss_lr(data_train, data_test)
```

A.4 共轭梯度法求解最优解

Listing 4: conjugate gradient.py

```python
import numpy as np
import matplotlib.pyplot as plt
from data import *
from utils import *

class conjugate_gradient:
    def __init__(self, degree, data_train, lr=1e-2, epochs=20, delta=1e-7, lambda_=1e-7):
        """
        共轭梯度法初始化

        Args:
            degree (int)): 维度
            data_train (tuple): 训练样本
            lr (float), optional): 学习率. Defaults to 1e-2.
            epochs (int, optional): 最大迭代次数. Defaults to 20.
            delta (float), optional): 早停策略的参数值. Defaults to 1e-7.
            lambda_ (float), optional): 正则项系数. Defaults to 1e-7.
        """
        self.degree = degree
        self.lambda_ = lambda_
        self.data_train = data_train
        self.lr = lr
        self.epochs = epochs
        self.delta = delta
        self.data_train_X_transformed = transform(data_train[0], degree)

    def conjugate_gradient(self):
        """
        共轭梯度法

        Returns:
            tuple: (W, 总迭代次数)
        """
        X, T = self.data_train_X_transformed, self.data_train[1]
```

```python
        A = np.transpose(X) @ X - self.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
        last_epoch = self.epochs
        for i in  range(self.epochs):
            alpha = np.transpose(r_0) @ r_0 / (np.transpose(p) @ A @ p)
            x = x + alpha * p
            r = r_0 + alpha * A @ p
            if np.transpose(r_0) @ r_0 < self.delta:
                last_epoch = i
                break
            beta = np.transpose(r) @ r / (np.transpose(r_0) @ r_0)
            p = -r + beta * p
            r_0 = r
        return x, last_epoch

# 画图
def draw(data_train, data_test, lr=1e-2, epochs=20, delta=1e-7, lambda_=1e-7, degrees=[2, 5, 7,
     9]):
    plt.figure(figsize=(12, 8))
    for i, degree in  enumerate(degrees):
        plt.subplot(2, 2, i + 1)
        cg = conjugate_gradient(degree, data_train, lr=lr, epochs=epochs, delta=delta, lambda_=
            lambda_)
        W, actual_epoch = cg.conjugate_gradient()
        data_test_X_transformed = transform(data_test[0], degree)
        plt.scatter(data_train[0], data_train[1], facecolor="none", edgecolor="b", label="
            Training Data")
        plt.plot(data_test[0], data_test[1], "g", label="$\sin(2\pi x)$")
        plt.plot(data_test[0], np.dot(data_test_X_transformed, W), "r", label="Conjugate
            Gradient Solution")
        plt.title("M={}, Actual Epoch={}". format(degree, actual_epoch))
        plt.legend()
    plt.show()


if __name__ == "__main__":
    sample_num = 10
    data_train, data_test = training_data_generator(sample_num=sample_num),
        testing_data_generator()
    draw(data_train, data_test)
```

A.5 辅助代码

Listing 5: utils.py

```python
import numpy as np
import matplotlib.pyplot as plt
from data import *

def transform(X, degree):
    """
    将样本自变量转为矩阵

    Args:
        X (list): 样本自变量
        degree (int): 维度

    Returns:
        list: 矩阵
    """
    X_transformed = np.zeros((X.shape[0], degree))
    for i in  range(X.shape[0]):
        X_transformed[i][0] = 1
        for j in  range(1, degree):
            X_transformed[i][j] = np.multiply(X[i], X_transformed[i][j - 1])
    return X_transformed

def calc_loss(X, W, T):
    """
    计算不带正则项的loss

    Args:
        X (list): X
        W (list): W
        T (list): T

    Returns:
        float: loss
    """
    loss = 0.5 * np.mean(np.transpose(X @ W - T) @ (X @ W - T))
    return loss

def calc_loss_normalized(X, W, T, lambda_):
    """
    计算带正则项的loss

    Args:
        X (list): X
        W (list): W
        T (list): T
        lambda_ (float): 正则项系数

```

```
    Returns:
        float: loss
    """
    loss = 0.5 * np.mean(np.transpose(X @ W - T) @ (X @ W - T) + lambda_ * np.transpose(W) @ W)
    return loss

def calc_gradient(X, W, T):
    """
    计算不带正则项的梯度

    Args:
        X (list): X
        W (list): W
        T (list): T

    Returns:
        list: 梯度
    """
    gradient = np.transpose(X) @ X @ W - np.transpose(X) @ T
    return gradient

def calc_gradient_normalized(X, W, T, lambda_):
    """
    计算带正则项的梯度

    Args:
        X (list): X
        W (list): W
        T (list): T
        lambda_ (float): 正则项系数

    Returns:
        list: 梯度
    """
    gradient = np.transpose(X) @ X @ W - np.transpose(X) @ T + lambda_ * W
    return gradient

def get_test_loss(W, data_test):
    """
    计算测试集上的loss（不带正则项）

    Args:
        W (list): W
        data_test (tuple): 测试样本

    Returns:
        float: loss
    """
```

```python
    X = transform(data_test[0], W.shape[0])
    return calc_loss(X, W, data_test[1])

def get_test_loss_normalized(W, data_test, lambda_):
    """
    计算测试集上的loss（带正则项）

    Args:
        W (list): W
        data_test (tuple): 测试样本
        lambda_ (float): 正则项系数

    Returns:
        float: loss
    """
    X = transform(data_test[0], W.shape[0])
    return calc_loss_normalized(X, W, data_test[1], lambda_)

def early_stop(loss, loss_last, delta):
    """
    早停策略

    Args:
        loss (float): 当前的loss
        loss_last (float): 上一个loss
        delta (float): 早停策略参数

    Returns:
        bool: 是否停止
    """
    return  abs(loss - loss_last) < delta


class lr_scheduler_MultiStep:
    def __init__(self, epochs, lr):
        """
        学习率下降策略

        Args:
            epochs (int): 总迭代轮数
            lr (float): 初始学习率
        """
        self.epochs = epochs
        self.lr = lr
        self.gamma = 0.1
        self.milestones = [ int(epochs * 0.5),  int(epochs * 0.75)]
    def step(self, epoch):
        """
```

```
        更新学习率

        Args:
            epoch (int): 当前迭代轮数

        Returns:
            float: 新的学习率
        """
        if epoch in self.milestones:
            self.lr *= self.gamma
        return self.lr
```