哈尔滨工业大学计算学部

实验报告

《机器学习》

实验一：多项式拟合正弦函数

学号：1183710109

姓名：郭茁宁

一、实验目的

掌握最小二乘法求解（无惩罚项的损失函数），掌握增加惩罚项（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、客服过拟合的方法（如增加惩罚项、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: Win 10
- Python 3.6.8

三、算法原理和设计

1、生成数据算法

主要是利用$sin(2\pi x)$函数产生样本，其中x均匀分布在$[0,1]$之间，对于每一个目标值$t=sin(2\pi x)$增加一个0均值，方差为0.25的高斯噪声。

```python
def get_data(
    x_range: (float, float) = (0, 1),
    sample_num: int = 10,
    base_func=lambda x: np.sin(2 * np.pi * x),
    noise_scale=0.25,
) -> "pd.DataFrame":
    X = np.linspace(x_range[0], x_range[1], num=sample_num)
    Y = base_func(X) + np.random.normal(scale=noise_scale, size=X.shape)
    data = pd.DataFrame(data=np.dstack((X, Y))[0], columns=["X", "Y"])
    return data
```

2、利用高阶多项式函数拟合曲线(不带惩罚项)

利用训练集合，对于每个新的$\hat{x}$，预测目标值$\hat{t}$。采用多项式函数进行学习，即利用式(1)来确定参数w，假设阶数m已知。

$$y(x,w)=w_0+w_1x+\cdots+w_mx^m=\sum_{i=0}^{m}w_ix^i \tag{1}$$

采用最小二乘法，即建立误差函数来测量每个样本点目标值t与预测函数$y(x,w)$之间的误差，误差函数即式(2)

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

将上式写成矩阵形式如式(3)

$$E(\mathbf{w}) = \frac{1}{2}(\mathbf{Xw} - \mathbf{T})'(\mathbf{Xw} - \mathbf{T}) \tag{3}$$

其中$\mathbf{X} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^m \\ 1 & x_2 & \cdots & x_2^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^m \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}, \mathbf{T} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$

通过将上式求导我们可以得到式(4)

$$\frac{\partial E}{\partial \mathbf{w}} = \mathbf{X}'\mathbf{Xw} - \mathbf{X}'\mathbf{T} \tag{4}$$

令 $\frac{\partial E}{\partial \mathbf{w}} = 0$ 我们有式(5)即为$\mathbf{w}^*$

$$\mathbf{w}^* = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{T} \tag{5}$$

由(5)实现

```
def get_params(x_matrix, t_vec) -> [float]:
    return np.linalg.pinv(x_matrix.T @ x_matrix) @ x_matrix.T @ t_vec
```

3、带惩罚项的多项式函数拟合曲线

在不带惩罚项的多项式拟合曲线时，在参数多时$\mathbf{w}^*$具有较大的绝对值，本质就是发生了过拟合。对于这种过拟合，我们可以通过在优化目标函数式(3)中增加$\mathbf{w}$的惩罚项，因此我们得到了式(6)。

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\{y(x_i, \mathbf{w}) - t_i\}^2 + \frac{\lambda}{2}||\mathbf{w}||^2 \tag{6}$$

同样我们可以将式(6)写成矩阵形式， 我们得到式(7)

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}[(\mathbf{Xw} - \mathbf{T})'(\mathbf{Xw} - \mathbf{T}) + \lambda\mathbf{w}'\mathbf{w}] \tag{7}$$

对式(7)求导我们得到式(8)

$$\frac{\partial \tilde{E}}{\partial \mathbf{w}} = \mathbf{X}'\mathbf{Xw} - \mathbf{X}'\mathbf{T} + \lambda\mathbf{w} \tag{8}$$

令 $\frac{\partial \tilde{E}}{\partial \mathbf{w}} = 0$ 我们得到$\mathbf{w}^*$即式(9)，其中$\mathbf{I}$为单位阵。

$$\mathbf{w}^* = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}'\mathbf{T} \tag{9}$$

由(9)实现

```python
def get_params_with_penalty(x_matrix, t_vec, lambda_penalty):
    return (
        np.linalg.pinv(
            x_matrix.T @ x_matrix + lambda_penalty *
np.identity(x_matrix.shape[1])
        )
        @ x_matrix.T
        @ t_vec
    )
```

4、梯度下降法求解最优解

对于$f(\mathbf{x})$如果在$\mathbf{x_i}$点可微且有定义，我们知道顺着梯度 $\nabla f(\mathbf{x_i})$为增长最快的方向，因此梯度的反方向 $-\nabla f(\mathbf{x_i})$ 即为下降最快的方向。因而如果有式(10)对于 $\alpha > 0$ 成立，

$$\mathbf{x_{i+1}} = \mathbf{x_i} - \alpha \nabla f(\mathbf{x_i}) \tag{10}$$

那么对于序列$\mathbf{x_0}, \mathbf{x_1}, \ldots$ 我们有$f(\mathbf{x_0}) \geq f(\mathbf{x_1}) \geq \ldots$

> 因此，如果顺利我们可以得到一个 $f(\mathbf{x_n})$ 收敛到期望的最小值，对于此次实验很大可能性可以收敛到最小值。

进而我们实现算法如下，其中δ为精度要求，通常可以设置为$\delta = 1 \times 10^{-6}$:

```python
def gradient_descent_fit(x_matrix, t_vec, lambda_penalty, w_vec_0,
learning_rate=0.1, delta=1e-6):
    loss_0 = calc_loss(x_matrix, t_vec, lambda_penalty, w_vec_0)
    k = 0
    w = w_vec_0
    while True:
        w_ = w - learning_rate * calc_derivative(x_matrix, t_vec, lambda_penalty,
w)
        loss = calc_loss(x_matrix, t_vec, lambda_penalty, w_)
        if np.abs(loss - loss_0) < delta:
            break
        else:
            k += 1
            if loss > loss_0:
                learning_rate *= 0.5
            loss_0 = loss
            w = w_
    return k, w
```

5、共轭梯度法求解最优解

> 共轭梯度法解决的主要是形如$\mathbf{Ax} = \mathbf{b}$的线性方程组解的问题，其中$\mathbf{A}$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，每次都选择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。

对于第 k 步的残差$\mathbf{r_k} = \mathbf{b} - \mathbf{Ax_k}$，我们根据残差去构造下一步的搜索方向$\mathbf{p_k}$，初始时我们令$\mathbf{p_0} = \mathbf{r_0}$。然后利用$Gram - Schmidt$方法依次构造互相共轭的搜素方向$\mathbf{p_k}$，具体构造的时候需要先得到第 k+1 步的残差，即$\mathbf{r_{k+1}} = \mathbf{r_k} - \alpha_k \mathbf{Ap_k}$，其中$\alpha_k$如后面的式$(11)$。

根据第 k+1 步的残差构造下一步的搜索方向$\mathbf{p_{k+1}} = \mathbf{r_{k+1}} + \beta_{k+1}\mathbf{p_k}$，其中$\beta_{k+1} = \dfrac{\mathbf{r_{k+1}^T r_{k+1}}}{\mathbf{r_k^T r_k}}$。

然后可以得到$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k \mathbf{p_k}$，其中$\alpha_k = \frac{\mathbf{p_k}^T(\mathbf{b} - \mathbf{A}\mathbf{x_k})}{\mathbf{p_k}^T\mathbf{A}\mathbf{p_k}} = \frac{\mathbf{p_k}^T\mathbf{r_k}}{\mathbf{p_k}^T\mathbf{A}\mathbf{p_k}}$

对于第 k 步的残差 $\mathbf{r_k} = \mathbf{b} - \mathbf{A}\mathbf{x}$，$\mathbf{r_k}$为$\mathbf{x} = \mathbf{x_k}$时的梯度反方向。由于我们仍然需要保证 $\mathbf{p_k}$ 彼此共轭。因此我们通过当前的残差和之前所有的搜索方向来构建$\mathbf{p_k}$，得到式(11)

$$\mathbf{p_k} = \mathbf{r_k} - \sum_{i<k} \frac{\mathbf{p_i}^T\mathbf{A}\mathbf{r_k}}{\mathbf{p_i}^T\mathbf{A}\mathbf{p_i}}\mathbf{p_i} \tag{11}$$

进而通过当前的搜索方向$\mathbf{p_k}$得到下一步优化解$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k \mathbf{p_k}$，其中
$\alpha_k = \frac{\mathbf{p_k}^T(\mathbf{b} - \mathbf{A}\mathbf{x_k})}{\mathbf{p_k}^T\mathbf{A}\mathbf{p_k}} = \frac{\mathbf{p_k}^T\mathbf{r_k}}{\mathbf{p_k}^T\mathbf{A}\mathbf{p_k}}$

> 求解的方法就是我们先猜一个解$\mathbf{x_0}$，然后取梯度的反方向 $\mathbf{p_0} = \mathbf{b} - \mathbf{A}\mathbf{x}$，在 n 维空间的基中$\mathbf{p_0}$要与其与的基共轭并且为初始残差。

对于共轭梯度下降，算法实现如下，初始时取$\mathbf{w_0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

由上文$(8)(9)$得

$$(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})\mathbf{w}^* = \mathbf{X}'\mathbf{T} \tag{12}$$

令

$$\begin{cases} \mathbf{A} = \mathbf{X}'\mathbf{X} + \lambda\mathbf{I} \\ \mathbf{b} = \mathbf{X}'\mathbf{T} \end{cases} \tag{13}$$

```python
def switch_deri_func_for_conjugate_gradient(x_matrix, t_vec, lambda_penalty, w_vec):
    A = x_matrix.T @ x_matrix - lambda_penalty * np.identity(len(x_matrix.T))
    x = w_vec
    b = x_matrix.T @ t_vec
    return A, x, b
```

通过共轭梯度下降法求解$Ax = b$

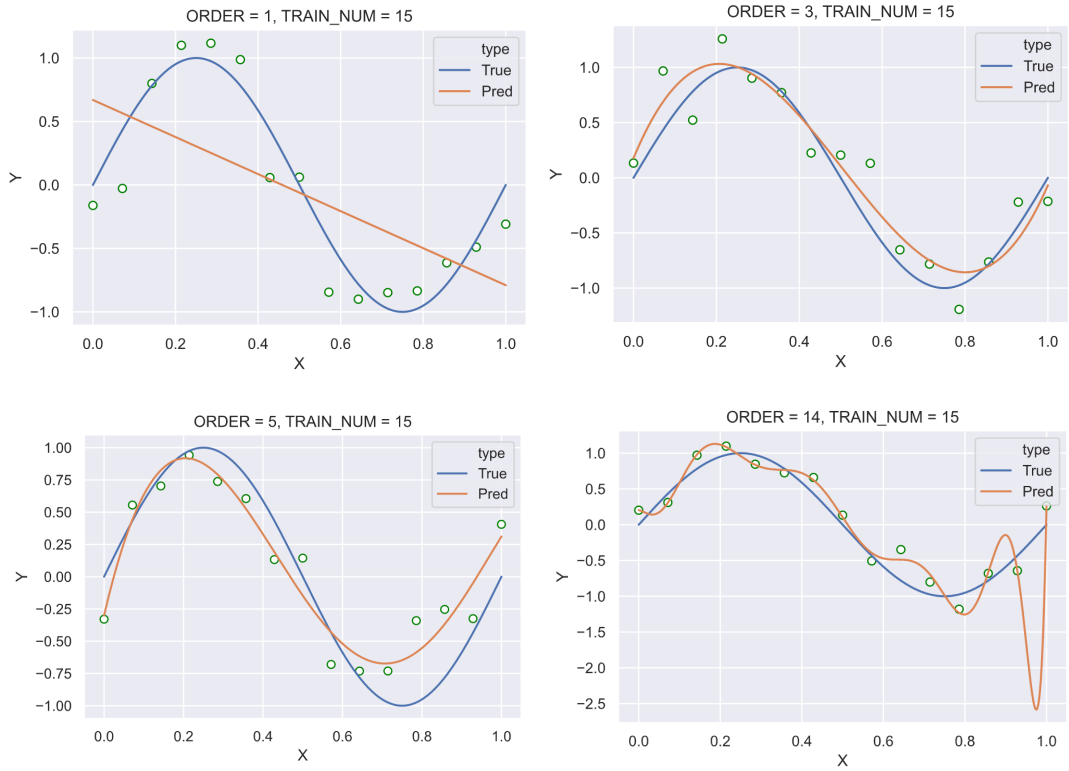
```python
def conjugate_gradient_fit(A, x_0, b, delta=1e-6):
    x = x_0
    r_0 = b - A @ x
    p = r_0
    k = 0
    while True:
        alpha = (r_0.T @ r_0) / (p.T @ A @ p)
        x = x + alpha * p
        r = r_0 - alpha * A @ p
        if r_0.T @ r_0 < delta:
            break
        beta = (r.T @ r) / (r_0.T @ r_0)
        p = r + beta * p
        r_0 = r
        k += 1
    return k, x
```

解得x，并记录迭代次数k

四、实验结果分析

1、不带惩罚项的解析解

> 固定训练样本的大小为 15，分别使用不同多项式阶数，测试：

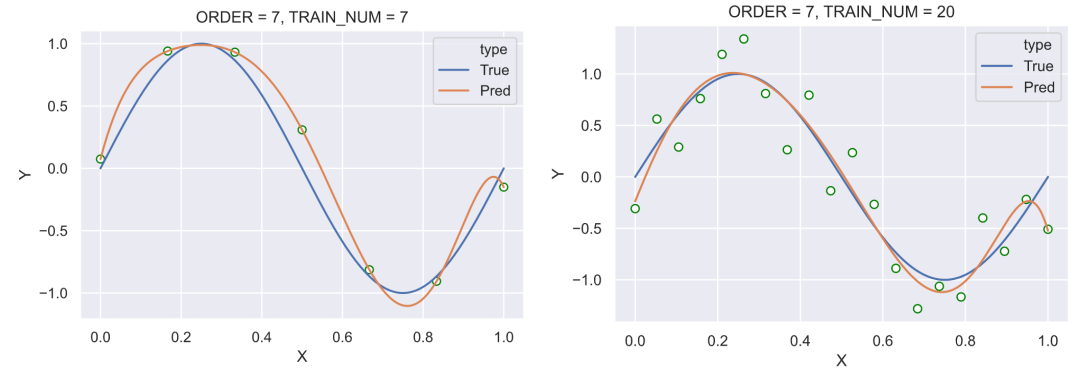


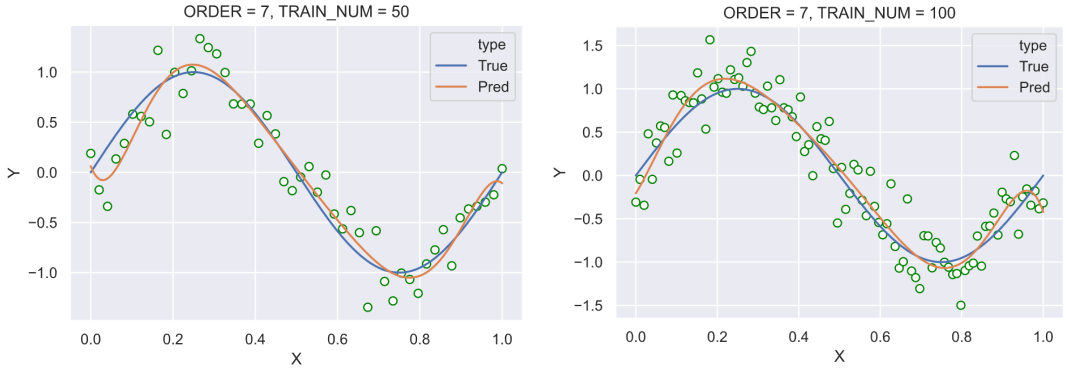
我们可以看到在固定训练样本的大小之后，在多项式阶数为 3 时的拟合效果已经很好。继续提高多项式的阶数，尤其在阶数为 14 的时候曲线"完美的"经过了所有的节点，这种剧烈的震荡并没有很好的拟合真实的背后的函数$sin(2\pi x)$，反而将所有噪声均很好的拟合，即表现出来一种过拟合的情况。

其出现过拟合的本质原因是，在阶数过大的情况下，模型的复杂度和拟合的能力都增强，因此可以通过过大或者过小的系数来实现震荡以拟合所有的数据点，以至于甚至拟合了所有的噪声。在这里由于我们的数据样本大小只有 15，所以在阶数为 14 的时候，其对应的系数向量$\mathbf{w}$恰好有唯一解，因此可以穿过所有的样本点。

对于过拟合我们可以通过增加样本的数据或者通过增加惩罚项的方式来解决。增加数据集样本数量，使其超过参数向量的大小，就会在一定程度上解决过拟合问题。

> 固定多项式阶数，使用不同数量的样本数据，测试

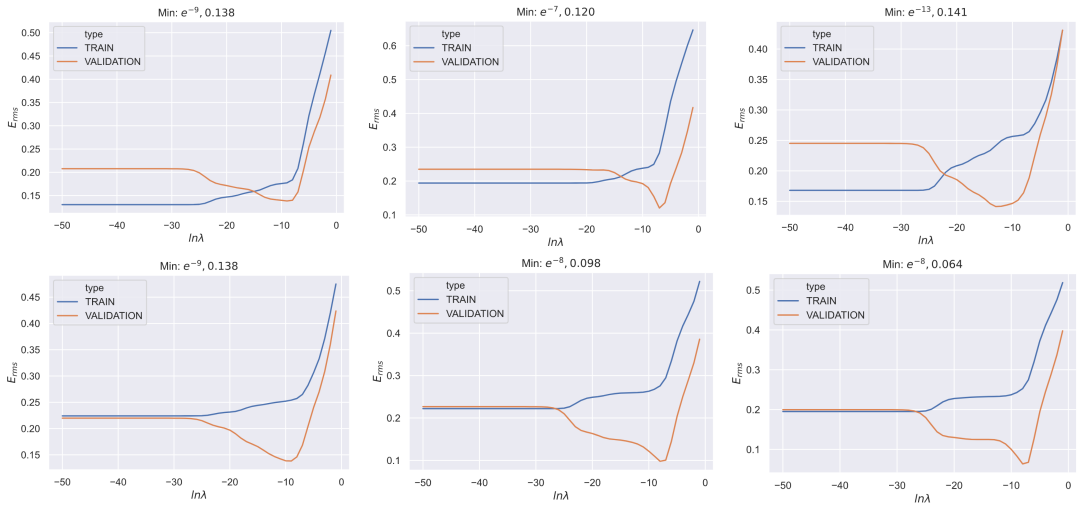




可以看到在固定多项式阶数为 7 的情况下，随着样本数量逐渐增加，过拟合的现象有所解决。特别是对比左上图与右下图的差距，可以看到样本数量对于过拟合问题是有影响的。

2、带惩罚项的解析解

首先根据式(6)我们需要确定最佳的超参数λ，因此我们通过根均方(RMS)误差来确定



观察上面的四张图，我们可以发现对于超参数λ的选择，在(e^{-50}, e^{-30})左右保持一种相对稳定的错误率；但是在(e^{-30}, e^{-5})错误率有一个明显的下降，所以下面在下面的完整 100 次实验中我们可以看到最佳参数的分布区间也大都在这个范围内；在大于e^{-5}的区间内，错误率有一个急剧的升高。

> 对于不同训练集，超参数λ的选择不同。
> 因此每生成新训练集，则通过带惩罚项的解析解计算出当前的$best_lambda$，用于带惩罚项的解析解、梯度下降和共轭梯度下降。

一般来说，最佳的超参数范围在(e^{-10}, e^{-6})之间。

> 比较是否带惩罚项的拟合曲线

3、优化解

实验利用梯度下降(Gradient descent)和共轭梯度法(Conjugate gradient)两种方法来求优化解。由于该问题是有解析解存在，并且在之前的实验报告部分已经列出，所以在此处直接利用上面的解析解进行对比。**此处使用的学习率固定为$learning_rate = 0.01$，停止的精度要求为1×10^{-6}。**

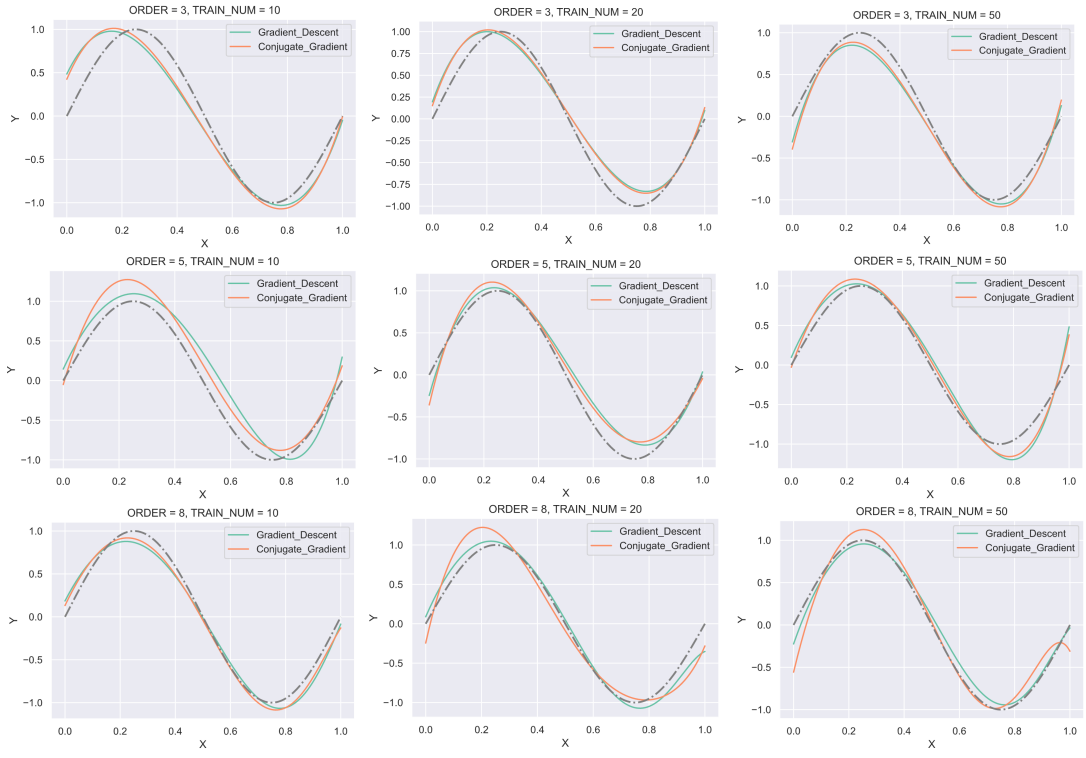
阶数	训练集规模	梯度下降迭代次数	共轭梯度下降迭代次数
3	10	110939	4
3	20	92551	4
3	50	56037	4
5	10	38038	4
5	20	22867	5
5	50	118788	5
8	10	81387	5
8	20	71743	7
8	50	47445	8

首先在固定多项式阶数的情况下，**随着训练样本的增加，梯度下降的迭代次数均有所下降，但是对于共轭梯度迭代次数变化不大。**

其次在固定训练样本的情况下，**梯度下降迭代次数的变化，对于 3 阶的情况下多于 8 阶的情况对于共轭梯度的而言，迭代次数仍然较少。**

总的来说，**对于梯度下降法，迭代次数在 10000 次以上；而对于共轭梯度下降，则需要的迭代次数均不超过 10 次(即小于解空间的维度 M)。**

> 下面是不同阶数和不同训练集规模上，两种方法的比较

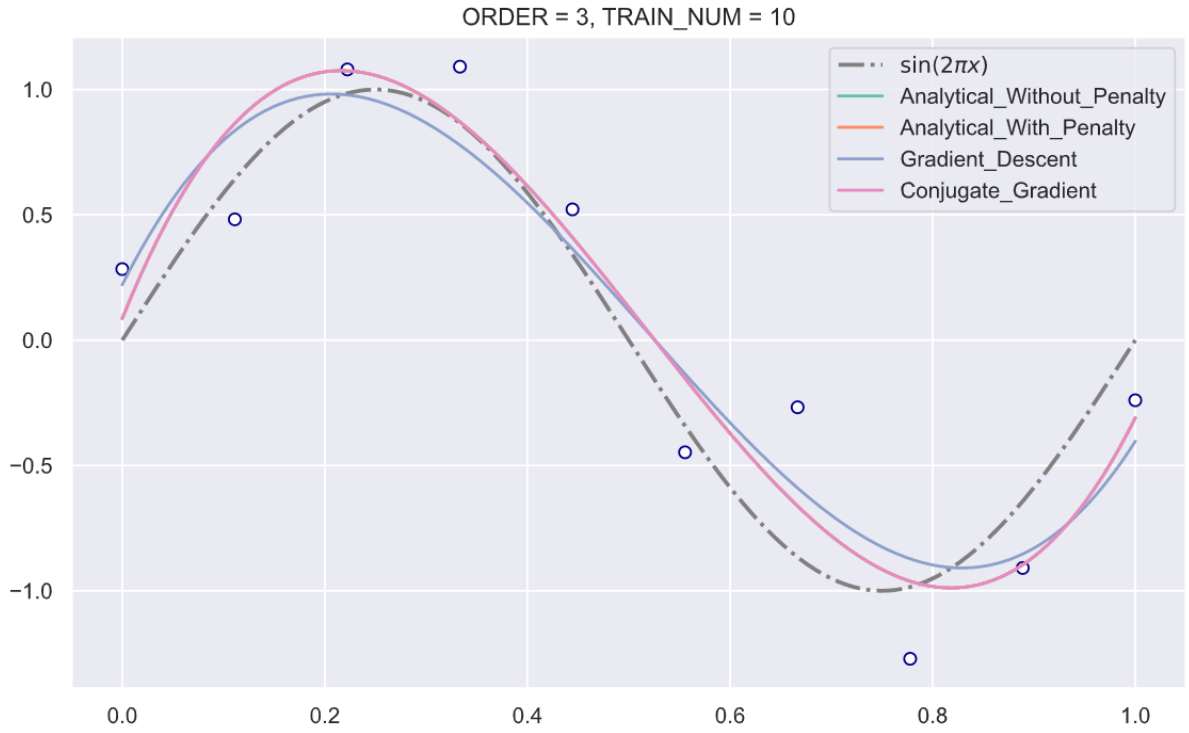


梯度下降和共轭梯度下降两种方法拟合结果相似，但共轭梯度下降收敛速度却远优于梯度下降、梯度下降稳定程度略优于共轭梯度下降。

4、四种拟合对比

> 3 阶，训练集规模 10

w	Analytical_Without_Penalty	Analytical_With_Penalty	Gradient_Descent	Conjugate_Gradient
0	0.086879	0.086879	0.221466	0.086879
1	10.030641	10.030641	8.037745	10.030641
2	-29.339163	-29.339163	-24.321837	-29.339163
3	18.911480	18.911480	15.658977	18.911480

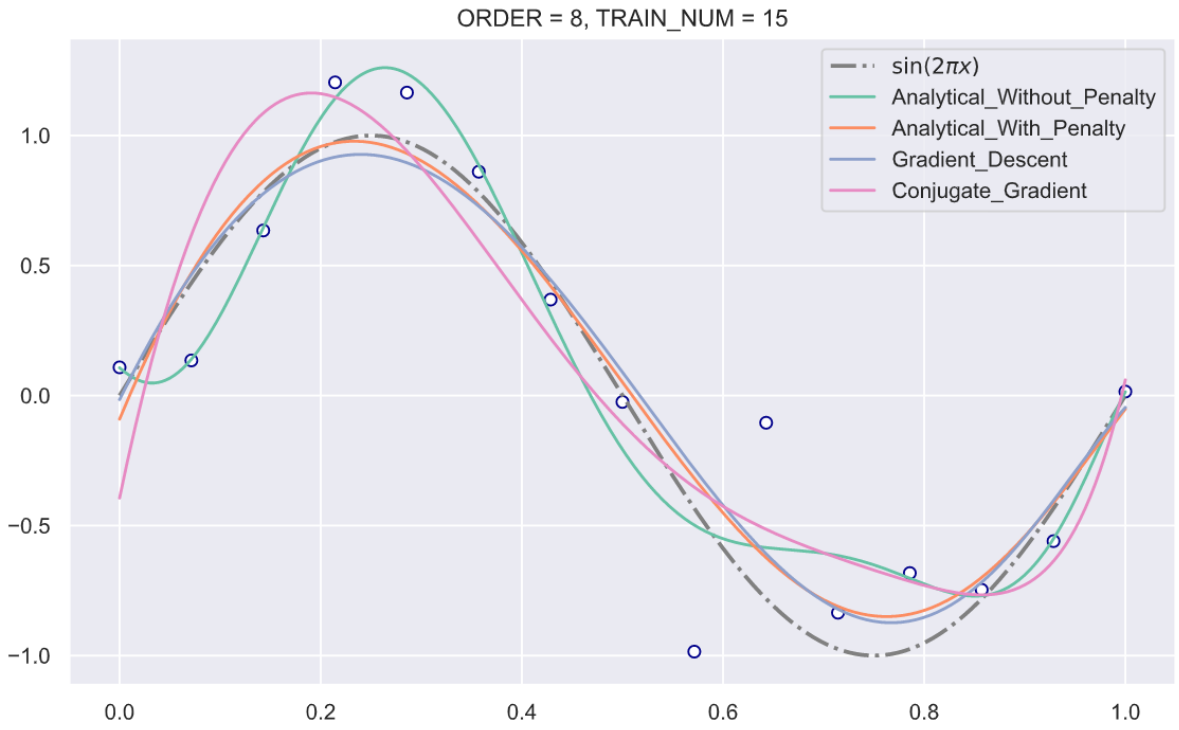


> 5 阶，训练集规模 15

w	Analytical_Without_Penalty	Analytical_With_Penalty	Gradient_Descent	Conjugate_Gradient
0	0.179098	0.179098	0.252970	0.127280
1	6.186679	6.186679	6.120921	8.802575
2	-4.197192	-4.197192	-14.547026	-24.093028
3	-47.562082	-47.562082	-0.720464	6.293567
4	73.581708	73.581708	6.179084	13.416870
5	-28.074476	-28.074476	2.945509	-4.398728

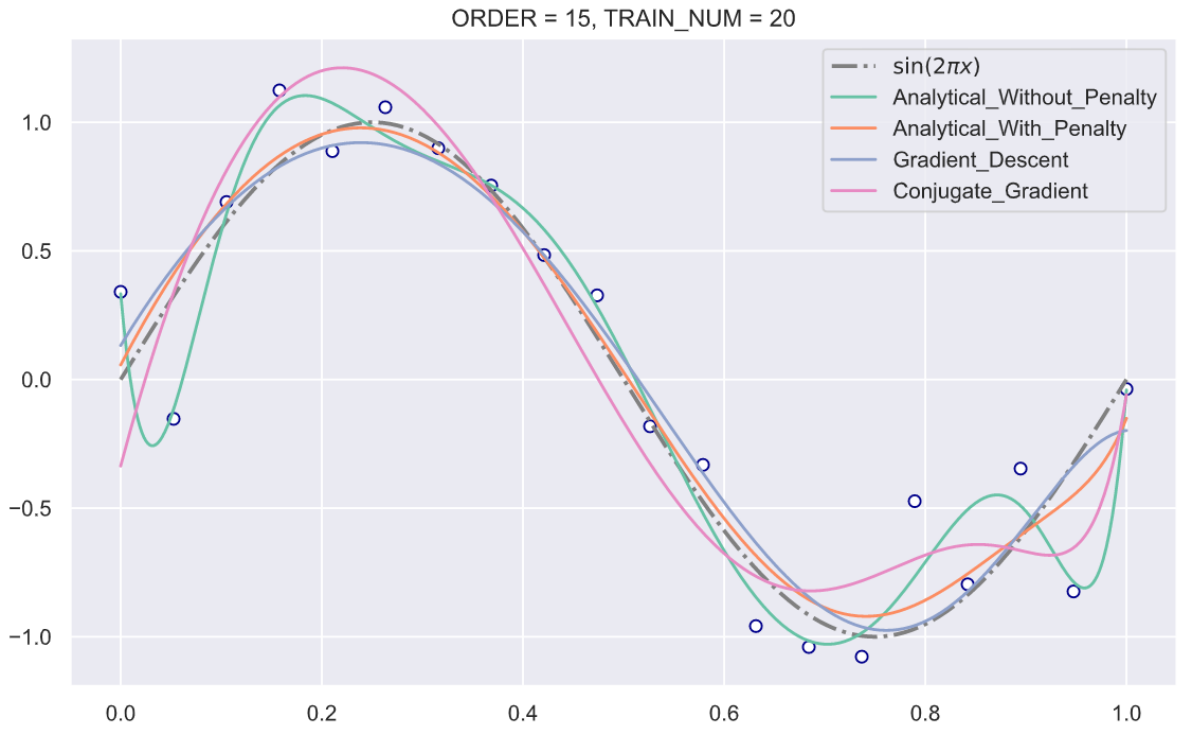
> 8 阶，训练集规模 15

w	Analytical_Without_Penalty	Analytical_With_Penalty	Gradient_Descent	Conjugate_Gradient
0	0.106875	-0.090665	-0.015363	-0.394036
1	-3.384695	9.291901	7.889612	18.583773
2	37.377871	-19.656246	-16.016235	-65.870523
3	451.762357	-6.583293	-3.968438	54.941302
4	-3775.172514	16.087370	5.885670	25.695923
5	10453.330011	10.996164	7.857581	-17.630151
6	-13828.893913	-3.553833	4.750929	-30.267690
7	8931.020178	-9.096546	-0.458928	-12.072342
8	-2266.131841	2.553330	-5.970533	27.074041



> 15 阶，训练集规模 20

w	Analytical_Without_Penalty	Analytical_With_Penalty	Gradient_Descent	Conjugate_Gradient
0	0.333171	0.056738	0.132317	-0.336080
1	-43.880559	7.449353	6.433572	14.722545
2	1033.334292	-13.398546	-12.067511	-36.803428
3	-8505.117304	-8.201093	-5.256489	3.989214
4	34811.021660	3.048536	2.141618	17.244891
5	-75466.068594	8.612557	5.268958	13.535997
6	75267.404153	8.529704	5.150508	5.089805
7	786.304140	5.220844	3.410453	-2.234687
8	-53324.724219	0.884139	1.230193	-6.603244
9	-2109.290031	-2.981579	-0.689982	-8.038983
10	42045.345941	-5.509223	-1.994826	-7.225030
11	14623.791181	-6.291021	-2.546300	-4.952217
12	-33191.453129	-5.219866	-2.332749	-1.904370
13	-22428.414650	-2.369747	-1.410270	1.399483
14	38870.202234	2.086673	0.133219	4.598064
15	-12368.829171	7.931668	2.199092	7.459104



五、结论

- 增加训练样本的数据可以有效的解决过拟合的问题。
- 对于训练样本限制较多的问题，通过增加惩罚项仍然可以有效解决过拟合问题。
- 对于梯度下降法和共轭梯度法而言，梯度下降收敛速度较慢，共轭梯度法的收敛速度快；且二者相对于解析解而言，共轭梯度法的拟合效果解析解的效果更好。
- 相比之下，共轭梯度下降能够有效的解决梯度下降法迭代次数多，和复杂度高的有效方法。

六、参考文献

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

七、附录（代码）

> 源代码：实验一 GitHub 仓库

`main.ipynb`

> Jupyter Notebook 运行效果：主程序

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

from DataGenerator import *
from AnalyticalSolution import *
from Switcher import *
from Calculator import *
```

```
from GradientDescent import *
from ConjugateGradient import *

%matplotlib inline
sns.set(palette="Set2")
```

```
# 超参数
TRAIN_NUM = 10 # 训练集规模
VALIDATION_NUM = 100 # 验证集规模
TEST_NUM = 1000 # 测试集规模
ORDER = 7 # 阶数
X_LEFT = 0 # 左界限
X_RIGHT = 1 # 右界限
NOISE_SCALE = 0.25 # 噪音标准差
LEARNING_RATE = 0.01 # 梯度下降学习率
DELTA = 1e-6 # 优化残差界限
W_SOLUTION = pd.DataFrame() # 不同方法的w解
LAMBDA = np.power(np.e, -7)

def base_func(x):
    """原始函数
    """
    return np.sin(2 * np.pi * x)

# 训练集
train_data = get_data(
    x_range=(X_LEFT, X_RIGHT),
    sample_num=TRAIN_NUM,
    base_func=base_func,
    noise_scale=NOISE_SCALE,
)
X_TRAIN = train_data["X"]
Y_TRAIN = train_data["Y"]

# 验证集
X_VALIDATION = np.linspace(X_LEFT, X_RIGHT, VALIDATION_NUM)
Y_VALIDATION = base_func(X_VALIDATION)

# 测试集
X_TEST = np.linspace(X_LEFT, X_RIGHT, TEST_NUM)
Y_TEST = base_func(X_TEST)

train_data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	X	Y
0	0.000000	0.087444
1	0.111111	0.667822
2	0.222222	0.826025
3	0.333333	0.563550
4	0.444444	0.548170
5	0.555556	-0.388687
6	0.666667	-0.803972
7	0.777778	-1.049368
8	0.888889	-0.648516
9	1.000000	0.234352

可视化拟合结果

```
def show_train_data(x_train, y_train):
    plt.scatter(x=x_train, y=y_train, color="white", edgecolors="darkblue")
```

```
def show_test_data(x_test, y_test):
    plt.plot(x_test, y_test, linewidth=2, color="gray", linestyle="-.")
```

```
def show_order_and_train_num():
    plt.title("ORDER = " + str(ORDER) + ", TRAIN_NUM = " + str(TRAIN_NUM))
```

```
def show_comparation(x, y1, label1, y2, label2):
    """可视化两个函数
    """
    d = pd.DataFrame(
        np.concatenate(
            (
                np.transpose([X_TEST, y1, [label1] * TEST_NUM]),
                np.transpose([X_TEST, y2, [label2] * TEST_NUM]),
            )
        ),
        columns=["X", "Y", "type"],
    )
    d[["X", "Y"]] = d[["X", "Y"]].astype("float")
    sns.lineplot(x="X", y="Y", data=d, hue="type")
    show_order_and_train_num()
```

解析解

```python
def get_y_pred_by_analytical(x_train, y_train, x, with_penalty=False, lambda_penalty=None):
    assert not with_penalty or lambda_penalty is not None  # 有惩罚项时，lambda不为空
    x_vec, t_vec = x_train, y_train
    w_vec = (
        get_params_with_penalty(
            x_matrix=get_x_matrix(x_vec, order=ORDER),
            t_vec=t_vec,
            lambda_penalty=lambda_penalty,
        )
        if with_penalty
        else get_params(x_matrix=get_x_matrix(x_vec, order=ORDER), t_vec=t_vec)
    )
    return np.dot(get_x_matrix(x, order=ORDER), w_vec), w_vec
```

不带惩罚项解析解

```python
y_pred_without_penalty, w_wec_without_penalty = get_y_pred_by_analytical(
    x_train=X_TRAIN, y_train=Y_TRAIN, x=X_TEST, with_penalty=False
)
W_SOLUTION["Analytical_Without_Penalty"] = w_wec_without_penalty
```

```python
show_comparation(x=X_TEST, y1=Y_TEST, label1="True", y2=y_pred_without_penalty, label2="Pred")
show_train_data(x_train=X_TRAIN, y_train=Y_TRAIN)
```

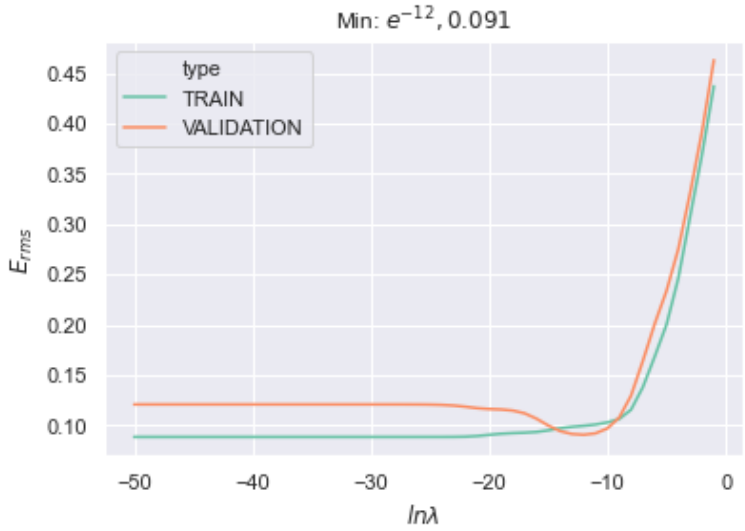
带惩罚项解析解

```python
def show_lambda_error(error_ln_lambda):
    data = pd.DataFrame(error_ln_lambda, columns=["$ln{\lambda}$", "$E_{rms}$", "type"])
    sns.lineplot(x="$ln{\lambda}$", y="$E_{rms}$", data=data, hue="type")
    idxmin = error_ln_lambda[data[data["type"]=="VALIDATION"]["$E_{rms}$"].idxmin()]
    plt.title(label=("Min: $e^{" + str(int(idxmin[0])) + "}, " + "{:.3f}".format(idxmin[1]) + "$"))
    return np.power(np.e, idxmin[0]), idxmin[1]
```

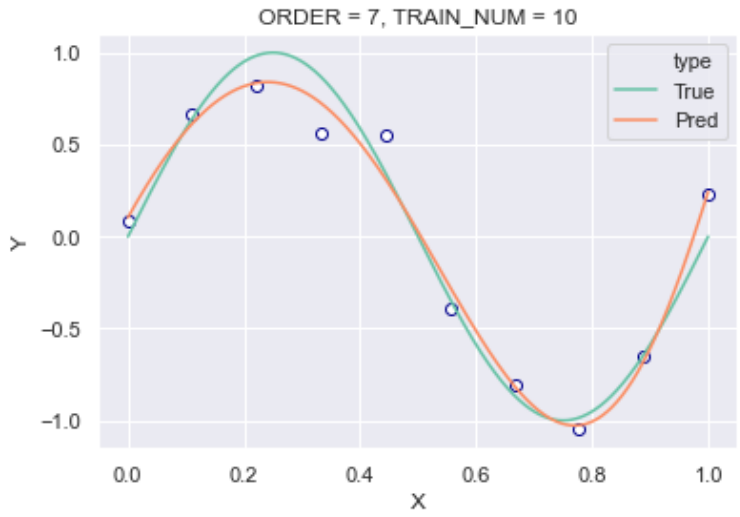
```python
error_ln_lambda = []
for i in range(-50, 0):
    y_pred, w_vec = get_y_pred_by_analytical(
        x_train=X_TRAIN,
```

```
        y_train=Y_TRAIN,
        x=X_TRAIN,  # 训练集
        with_penalty=True,
        lambda_penalty=np.exp(i),
    )
    error_ln_lambda.append(
        [i, calc_e_rms(y_pred=y_pred, y_true=Y_TRAIN), "TRAIN"]
    )  # 训练集上的根均方误差
    y_pred, w_vec = get_y_pred_by_analytical(
        x_train=X_TRAIN,
        y_train=Y_TRAIN,
        x=X_VALIDATION,  # 验证集
        with_penalty=True,
        lambda_penalty=np.exp(i),
    )
    error_ln_lambda.append(
        [i, calc_e_rms(y_pred=y_pred, y_true=Y_VALIDATION), "VALIDATION"]
    )  # 测试集上的根均方误差
```

```
best_lambda, least_loss = show_lambda_error(error_ln_lambda)
LAMBDA = best_lambda
```




```
y_pred_with_penalty, w_wec_with_penalty = get_y_pred_by_analytical(
    x_train=X_TRAIN, y_train=Y_TRAIN, x=X_TEST, with_penalty=True, lambda_penalty=LAMBDA,
)
W_SOLUTION["Analytical_With_Penalty"] = w_wec_with_penalty
show_comparation(x=X_TEST, y1=Y_TEST, label1="True", y2=y_pred_with_penalty, label2="Pred")
show_train_data(x_train=X_TRAIN, y_train=Y_TRAIN)
```



对比是否带惩罚项的拟合结果

```
show_comparation(
    x=X_TEST,
    y1=y_pred_without_penalty,
    label1="Analytical_Without_Penalty",
    y2=y_pred_with_penalty,
    label2="Analytical_With_Penalty",
)
plt.legend(["Analytical_Without_Penalty", "Analytical_With_Penalty"])
```

```
<matplotlib.legend.Legend at 0x24269fcaa90>
```

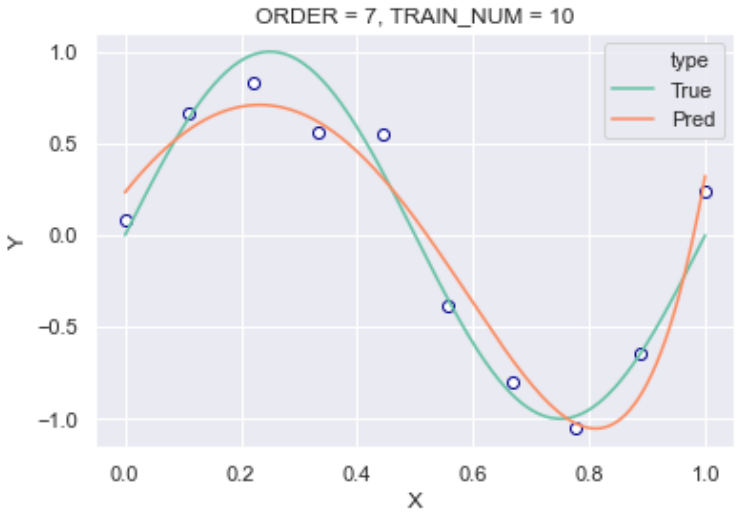
梯度下降法

```python
def get_y_pred_by_gradient_descent(x_train, y_train, x, lambda_penalty):
    x_vec, t_vec = x_train, y_train
    k, w_vec = gradient_descent_fit(
        x_matrix=get_x_matrix(x_vec, order=ORDER),
        t_vec=t_vec,
        lambda_penalty=lambda_penalty,
        w_vec_0=np.zeros(ORDER + 1),
        learning_rate=LEARNING_RATE,
        delta=DELTA,
    )
    return k, np.dot(get_x_matrix(x, order=ORDER), w_vec), w_vec
```

```python
k_gradient_descent, y_pred_gradient_descent, w_wec_gradient_descent =
get_y_pred_by_gradient_descent(
    x_train=X_TRAIN, y_train=Y_TRAIN, x=X_TEST, lambda_penalty=LAMBDA
)
W_SOLUTION["Gradient_Descent"] = w_wec_gradient_descent
k_gradient_descent
```

```
52893
```

```python
show_comparation(x=X_TEST, y1=Y_TEST, label1="True", y2=y_pred_gradient_descent,
label2="Pred")
show_train_data(x_train=X_TRAIN, y_train=Y_TRAIN)
```



共轭梯度法

```python
def get_y_pred_by_conjugate_gradient(x_train, y_train, x, lambda_penalty):
    x_vec, t_vec = x_train, y_train
    A, x_0, b = switch_deri_func_for_conjugate_gradient(
        x_matrix=get_x_matrix(x_vec, order=ORDER),
        t_vec=t_vec,
        lambda_penalty=lambda_penalty,
        w_vec=np.zeros(ORDER + 1),
    )
    k, w_vec = conjugate_gradient_fit(A=A, x_0=x_0, b=b, delta=DELTA)
    return k, np.dot(get_x_matrix(x, order=ORDER), w_vec), w_vec
```

```
k_conjugate_gradient, y_pred_conjugate_gradient, w_wec_conjugate_gradient =
get_y_pred_by_conjugate_gradient(
    x_train=X_TRAIN, y_train=Y_TRAIN, x=X_TEST, lambda_penalty=LAMBDA,
)
W_SOLUTION["Conjugate_Gradient"] = w_wec_conjugate_gradient
k_conjugate_gradient
```

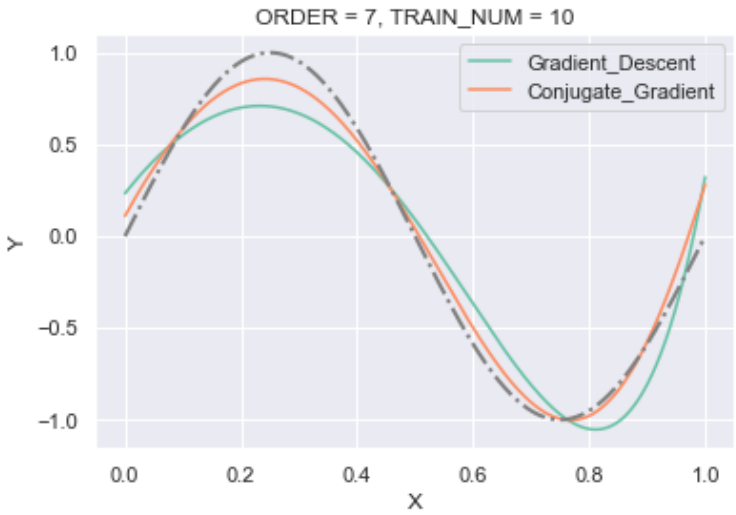
```
5
```

```
show_comparation(x=X_TEST, y1=Y_TEST, label1="True", y2=y_pred_conjugate_gradient,
label2="Pred")
show_train_data(x_train=X_TRAIN, y_train=Y_TRAIN)
```

对比梯度下降和共轭梯度的拟合结果

```
show_comparation(
    x=X_TEST,
    y1=y_pred_gradient_descent,   label1="Gradient_Descent",
    y2=y_pred_conjugate_gradient, label2="Conjugate_Gradient",
)
show_test_data(x_test=X_TEST, y_test=Y_TEST)
plt.legend(["Gradient_Descent", "Conjugate_Gradient"])
```

```
<matplotlib.legend.Legend at 0x24269d1cc88>
```



四种拟合方法汇总

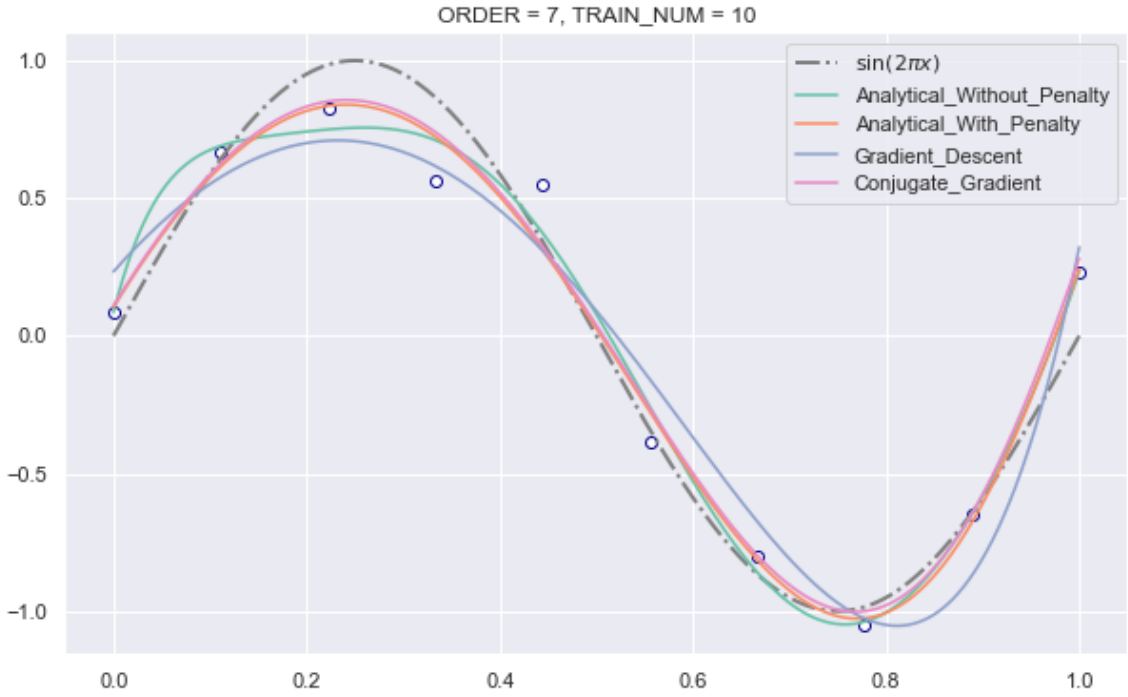
```
W_SOLUTION
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Analytical_Without_Penalty	Analytical_With_Penalty	Gradient_Descent	Conjugate_Gradient
0	0.084599	0.103922	0.233123	0.110288
1	13.779036	5.694382	3.978839	5.693982
2	-125.344309	-8.782488	-7.604623	-8.090070
3	601.300442	-9.708832	-3.345961	-12.785307
4	-1540.525403	-1.287225	0.884203	2.779658
5	2029.729045	14.740948	2.743226	14.930732
6	-1308.321291	13.958697	2.525709	10.353532
7	329.531802	-14.476569	0.907662	-12.711493

```
plt.figure(figsize=(10, 6))
show_train_data(x_train=X_TRAIN, y_train=Y_TRAIN)
show_test_data(x_test=X_TEST, y_test=Y_TEST)
for column in W_SOLUTION.columns:
    sns.lineplot(
        x=X_TEST,
        y=[W_SOLUTION[column] @ get_x_series(x=x, order=ORDER) for x in X_TEST],
    )
plt.legend(["$\sin (2 \pi x)$"] + list(W_SOLUTION.columns))
show_order_and_train_num()
```



DataGenerator.py

```python
import numpy as np
import pandas as pd


def get_data(
    x_range: (float, float) = (0, 1),
    sample_num: int = 10,
    base_func=lambda x: np.sin(2 * np.pi * x),
    noise_scale=0.25,
) -> "pd.DataFrame":
    X = np.linspace(x_range[0], x_range[1], num=sample_num)
    Y = base_func(X) + np.random.normal(scale=noise_scale, size=X.shape)
    data = pd.DataFrame(data=np.dstack((X, Y))[0], columns=["X", "Y"])
    return data

```

AnalyticalSolution.py

```python
import numpy as np


def get_params(x_matrix, t_vec) -> [float]:
    return np.linalg.pinv(x_matrix.T @ x_matrix) @ x_matrix.T @ t_vec


def get_params_with_penalty(x_matrix, t_vec, lambda_penalty):
    return (
        np.linalg.pinv(
            x_matrix.T @ x_matrix + lambda_penalty * np.identity(x_matrix.shape[1])
        )
        @ x_matrix.T
        @ t_vec
    )

```

GradientDescent.py

```python
import numpy as np
from Calculator import *


def gradient_descent_fit(
    x_matrix, t_vec, lambda_penalty, w_vec_0, learning_rate=0.1, delta=1e-6
):
    loss_0 = calc_loss(x_matrix, t_vec, lambda_penalty, w_vec_0)
    k = 0
    w = w_vec_0
    while True:
        w_ = w - learning_rate * calc_derivative(x_matrix, t_vec, lambda_penalty, w)
        loss = calc_loss(x_matrix, t_vec, lambda_penalty, w_)
        if np.abs(loss - loss_0) < delta:
            break
        else:
            k += 1
            if loss > loss_0:
                learning_rate *= 0.5
            loss_0 = loss
            w = w_
    return k, w

```

ConjugateGradient.py

```python
import numpy as np


def conjugate_gradient_fit(A, x_0, b, delta=1e-6):
    """解Ax=b
    """
    x = x_0
    r_0 = b - A @ x
    p = r_0
    k = 0
    while True:
        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 < delta:
            break
        beta = (r.T @ r) / (r_0.T @ r_0)
        p = r + beta * p
        r_0 = r
        k += 1
    return k, x


```

Calculator.py

```python
import numpy as np

```

```
3
4  def calc_e_rms(y_true, y_pred):
5      return np.sqrt(np.mean(np.square(y_true - y_pred)))
6
7
8  def calc_loss(x_matrix, t_vec, lambda_penalty, w_vec):
9      Xw_T = x_matrix @ w_vec - t_vec
10     return 0.5 * np.mean(Xw_T.T @ Xw_T + lambda_penalty * w_vec @ np.transpose([w_vec]))
11
12
13 def calc_derivative(x_matrix, t_vec, lambda_penalty, w_vec):
14     return x_matrix.T @ x_matrix @ w_vec - x_matrix.T @ t_vec + lambda_penalty * w_vec
15
```

Switcher.py

```
1  import numpy as np
2
3
4  def get_x_series(x, order) -> [float]:
5      """return 1, x^1, x^2,..., x^n, n = order
6      """
7      series = [1.0]
8      for _ in range(order):
9          series.append(series[-1] * x)
10     return series
11
12
13 def get_x_matrix(x_vec, order: int = 1) -> [[float]]:
14     x_matrix = []
15     for i in range(len(x_vec)):
16         x_matrix.append(get_x_series(x_vec[i], order))
17     return np.asarray(x_matrix)
18
19
20 def switch_deri_func_for_conjugate_gradient(x_matrix, t_vec, lambda_penalty, w_vec):
21     A = x_matrix.T @ x_matrix - lambda_penalty * np.identity(len(x_matrix.T))
22     x = w_vec
23     b = x_matrix.T @ t_vec
24     return A, x, b
25
```