2020 年春季学期

计算学部《机器学习》课程

Lab 1 实验报告

姓名	余涛
学号	1180300829
班号	1803202
电子邮件	1063695334@qq.com
手机号码	15586430583

目录

1 实验内容

1.1 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)

1.2 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。

1.3 实验环境

Windows 10 专业版；python 3.8.6；PyCharm Community Edition 2020.2.2 x64

2 实验设计思想

2.1 算法原理

2.1.1 生成数据并加入噪声

首先产生[0,1]之间的均匀的 x 样本，利用函数 sin(2πx)计算每一个 x 样本值对应的 t 样本，然后对于每一个 t 样本值，加上一个均值为 0，方差为 0.1 的高斯噪声。

2.1.2 不带惩罚项，利用高阶多项式函数拟合曲线

由于足够高阶的多项式可以拟合任意函数，则可以通过多项式来拟合正弦函数 sin(2πx)。在 m 阶多项式中，存在 m+1 个系数，则问题转变为需要找到 m+1 个系数组成的列向量 w。多项式如下所示:

$$y(x, w) = w_0 + w_1 x + \cdots + w_m x^m = \sum_{i=0}^{m} w_i x^i$$

通过最小二乘法，可以得到误差函数，误差函数可以计算每个 x 样本的目标值 t 与预测的函数 y(x,w)的误差，误差函数如下所示：

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

将此式转化为矩阵形式如下:

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

在上式中，各个参数分别的意义为:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^m \\ 1 & x_2 & \cdots & x_2^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^m \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

对误差函数的矩阵形式进行求导可得：

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

然后令 E 对 w 的偏导数为 0，既可以得到 w*的值：

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

求解出 w*后，就可以得到拟合的多项式进而画出拟合后的图像

2.1.3 带惩罚项，利用高阶多项式函数拟合曲线

不带惩罚项时，多项式拟合的曲线可能会发生过拟合，所以可以通过在误差函数中增加 w 的惩罚项来减少这种过拟合，加上惩罚项后如下所示：

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

将此式转化为矩阵形式如下：

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}[(\mathbf{X}\mathbf{w} - \mathbf{T})'(\mathbf{X}\mathbf{w} - \mathbf{T}) + \lambda\mathbf{w}'\mathbf{w}]$$

对误差函数的矩阵形式进行求导可得：

$$\frac{\partial \widetilde{E}}{\partial \mathbf{w}} = \mathbf{X}'\mathbf{X}\mathbf{w} - \mathbf{X}'\mathbf{T} + \lambda\mathbf{w}$$

然后令 E 对 w 的偏导数为 0，既可以得到 w*的值：

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

求解出 w*后，就可以得到拟合的多项式进而画出拟合后的图像

2.1.4 梯度下降法

数学原理：对于一个函数 f(x)，如果其在某个点 xi 处可微，则顺着该点处的梯度∇f (xi)是函数增长最快的方向，由此可得，梯度的反方向-∇f (xi)是函数下降最快的方向，于是可以通过不断的迭代取 x 的新值来使得 f(xn)收敛到一个期望的最小值，如下所示：

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

则有：

$$f(\mathbf{x_0}) \geq f(\mathbf{x_1}) \geq \ldots$$

因此可以通过对误差函数进行迭代来求得误差函数最小值的近似值，在该处误差最小。

误差函数的矩阵形式求导仍为：

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

可以设置步长（学习率）为 η，然后对 w 用梯度下降，直到完成迭代或者收敛，公式如下：

$$w \longleftarrow w - \eta \cdot \frac{\partial E}{\partial w}$$

求解出 w 后，就可以得到拟合的多项式进而画出拟合后的图像

2.1.5 共轭梯度法

对于共轭梯度法，其不像梯度下降法需要通过很多步的迭代来得到 w 的优化解，而是只需要迭代 m+1 次即可。在共轭梯度法中：

误差函数的矩阵形式求导仍为：

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

令其为 0，即：

$$\frac{\partial E}{\partial w} = 0$$

此时需要求解：

$$(X^T X + \lambda) w = X^T Y$$

可以记

$$Q = X^T X + \lambda$$

$$b = X^T Y$$

则可以把方程求解转化为求解

$$min_{w \in R^n} \frac{1}{2} w^T Q w - b^T w$$

共轭梯度法一般的算法流程为：

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

通过迭代便可以得到 w 的优化解，求解出 w 后，就可以得到拟合的多项式进而画出拟合后的图像

2.2 算法实现

2.2.1 生成数据并加入噪声

作用：

```
'''
    生成sample_num个数据，并加入噪声
    sample_x为[0,0.9]上均匀的sample_num个点
    sample_y为sin(2πx)的值，再加上均值为mu，方差为sigma的高斯噪声
'''
```

具体实现：

通过调用 numpy 库的相关函数给出[0,1]上的 x 样本，然后求得相应的 sin(2πx)的 y 样本值并加上均值为 mu，方差为 sigma 的高斯噪声

```
def CreateData(mu, sigma):
    sample_x = np.arange(0, 1, 1 / sample_num)      #创建[0,0.9]上均匀的
sample_num 个点
    gauss_noise = np.random.normal(mu, sigma, sample_num)  #创建均值为
mu，标准差为 sigma 的高斯噪声
    sample_y = np.sin(sample_x * 2 * np.pi) + gauss_noise  #对于
sample_x 得到每个 sin(2πx)的值，再加上均值为 mu，标准差为 sigma 的高斯噪声
    return sample_x, sample_y
```

2.2.2 将样本 sample_x 进行转换，生成一个 X 矩阵

作用：

```
'''
    将所有的样本sample_x进行转换，生成一个X矩阵
    在CreateData中生成的均匀的sample_x的样本数据是一维向量，需要预处理成为矩阵X
    其中矩阵X的纬度为sample_num * (polynomial_order + 1)
'''
```

具体实现：以注释形式给出

```
def CreateMatrixX(sample_x):
    X = np.zeros((sample_num, polynomial_order + 1))  #先建立一个矩阵，行
数为 sample_num，列数为 polynomial_order + 1
    for i in range(sample_num):  #对矩阵每一行进行赋值
        every_row_i = np.ones(polynomial_order + 1) * sample_x[i]  #先
将每一行全部赋值为 sample_x[i]
        poly_row = np.arange(0, polynomial_order+1)  #得到每一列的阶数
        every_row_i = np.power(every_row_i, poly_row)  #得到这一行所有
sample_x[i]值的阶数值
        X[i] = every_row_i
    return X
```

2.2.3 得到误差函数 E(w)

作用：

```
'''
    误差函数E(w)
    由已知得到误差函数的表达式为E(w) = 1/2 * (Xw - Y)^{T} . (Xw - Y)
'''
```

具体实现：调用公式即可，以注释形式给出

```
def ErrorFunction(sample_x, sample_y, w):
    X = CreateMatrixX(sample_x)   #将 sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维
向量
    temp = X.dot(w) - Y  #X 矩阵与 w 相乘后减去 Y
    ErrorFunction = 1/2 * np.dot(temp.T, temp)  #套用误差函数表达式即可
    return ErrorFunction
```

2.2.4 直接拟合

作用：

```
'''
    直接拟合数据，通过调用np.polyfit()拟合数据
'''
```

具体实现：调用 np.polyfit()即可，以注释形式给出

```
def FittingData(sample_x, sample_y):
    w = np.polyfit(sample_x, sample_y, polynomial_order)  # 用
polynomial_order 次多项式拟合，
    poly = np.poly1d(w)  #得到多项式系数，按照阶数从高到低排列
    return poly
```

2.2.5 获取均方误差(RMS)

作用：

```
'''
    得到一组测试数据与真实数据的均方误差(RMS)
'''
```

具体实现：使用均方根公式即可，以注释形式给出

```
def Get_RMS(train_num, poly_fit):
    real_x = np.linspace(0,1,train_num)      #真实的[0,1]上 train_num 个
均匀点
    real_y = np.sin(real_x * 2 * np.pi)    #得到对应真实的 sin2pix 值
    fit_y = poly_fit(real_x)         #得到拟合值
    the_loss = real_y-fit_y     #得到拟合值和真实值的差
    the_RMS = np.sqrt((np.dot(the_loss,the_loss.T))/train_num)  #求解均
方根，作为误差
    return the_RMS
```

2.2.6 得到最佳惩罚项系数 lamda

作用：

```
'''
    得到最合适的惩罚项系数lamda
'''
```

具体实现：通过遍历从 e^{-50} 到 e^{0} 的惩罚项系数，每次阶数增加 1，将其带入带惩罚项的高阶多项式中，然后进行比较，找到均方误差最小时对应的惩罚项系数，这就是最佳的惩罚项系数 lamda，以注释形式给出

```
def Get_best_lamda():
    train_num = 100;        #设置测试数据容量为100
    sample_x, sample_y = CreateData(0, 0.5)    #创建数据集
    the_degree = np.zeros(51)     #惩罚项系数阶数集
    the_RMS = np.zeros(51)      #惩罚项每个阶数对应的均方根集
    min = the_degree[0]
    min_num = 10000
    for i in range(0,51):      #对每个阶数进行均方根的求解
        the_degree[i] = -i     #为阶数集赋值
        the_best_lamda = np.exp(the_degree[i])  #得到每一个阶数对应的惩罚项
系数
        poly = Penalty_item_add_ErrorFunction(sample_x, sample_y,
the_best_lamda,sample_num,polynomial_order)  #求解多项式
        the_RMS[i] = Get_RMS(train_num,poly)    #对该多项式求解均方根
        if(min_num>the_RMS[i]):    #得到最小的均方根对应的惩罚项系数阶数
            min = the_degree[i]
            min_num = the_RMS[i]
    plot = matplt.plot(the_degree, the_RMS, 'm', label='lamda line')
    matplt.xlabel('the poly of lamda')
    matplt.ylabel('RMS')
    matplt.legend(loc=1)
    matplt.title("the best poly of lamda is"+str(min))
    matplt.show()
```

2.2.7 不带惩罚项，利用高阶多项式函数拟合曲线

作用：

```
'''
    情况：不加惩罚项
    令误差函数导数等于0，求此时的w，此时的w = (X^{T} * X)^{-1} * X^{T} * Y
'''
```

具体实现：带入此时求 w 的公式即可，以注释形式给出

```
def Penalty_item_ErrorFunction(sample_x,
sample_y,sample_num,polynomial_order):
```

```
    X = CreateMatrixX(sample_x,sample_num, polynomial_order)   #将sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维向量
    w = np.linalg.inv(np.dot(X.T, X)).dot(X.T).dot(Y)   #套用公式 w = (X^{T} * X)^{-1} * X^{T} * Y 得到 w
    poly = np.poly1d(w[::-1].reshape(polynomial_order + 1))  #先将 w 从后往前，得到多项式系数，按照阶数从高到低排列
    return poly
```

2.2.8 带惩罚项，利用高阶多项式函数拟合曲线

作用：

```
'''
    情况：加惩罚项，此时惩罚项系数为lamda
    令误差函数导数等于0，求此时的w，此时的w = (X^{T} * X + lamda)^{-1} * X^{T} * Y
'''
```

具体实现：带入此时求 w 的公式即可（仅仅多了一个惩罚项），以注释形式给出

```
def Penalty_item_add_ErrorFunction(sample_x, sample_y, lamda,sample_num,polynomial_order):
    X = CreateMatrixX(sample_x,sample_num, polynomial_order)   #将sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))   #将 sample_y 变成一个竖着的一维向量
    w = np.linalg.inv(np.dot(X.T, X) + lamda * np.eye(X.shape[1])).dot(X.T).dot(Y)  #套用公式 w = (X^{T} * X + lamda)^{-1} * X^{T} * Y 得到 w
    poly = np.poly1d(w[::-1].reshape(polynomial_order + 1))  #先将 w 从后往前，得到多项式系数，按照阶数从高到低排列
    return poly
```

2.2.9 梯度下降法

作用：

```
'''
    情况：加惩罚项，此时惩罚项系数为lamda
    对误差函数使用梯度下降法，当误差函数收敛到期望的最小值时，得到此时的w
    参数中，最多循环轮次cycle_times，下降步长descending_step_size，迭代误差iteration_error
'''
```

具体实现：通过对迭代公式

$$w \longleftarrow w - \eta \cdot \frac{\partial E}{\partial w}$$

不断迭代，当此时的迭代误差小于规定的迭代误差时终止迭代，如果迭代后的新的迭代误差大于原迭代误差，说明此时的迭代值在解析解左右摇摆，此时需要将步长变为原来的一半，

以注释形式给出

```
def Gradient_descent_method(sample_x, sample_y, lamda, cycle_times, descending_step_size, iteration_error,sample_num,polynomial_order):
    w = np.ones((polynomial_order + 1, 1))   #生成一个行数为 polynomial_order + 1，列数为 1 的矩阵，元素值全为 1
    X = CreateMatrixX(sample_x,sample_num, polynomial_order)   #将 sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维向量
    for i in range(cycle_times):  #迭代 cycle_times 次
        old_ErrorFunction = abs(ErrorFunction(sample_x, sample_y,sample_num,polynomial_order, w))  #求得旧的误差函数值的绝对值
        ErrorFunction_partial_deriv = X.T.dot(X).dot(w) - X.T.dot(Y) + lamda * w  #误差函数对 w 的偏导数公式为 X^{T} * X * w - X^{T} * Y + lamda * w，即梯度
        w = w - descending_step_size * ErrorFunction_partial_deriv  #梯度下降法的迭代公式为 w <--- w - descending_step_size * 误差函数对 w 的偏导数
        new_ErrorFunction = abs(ErrorFunction(sample_x, sample_y,sample_num,polynomial_order, w))  #求得新的误差函数值的绝对值
        if(new_ErrorFunction > old_ErrorFunction): #当新的误差函数值大于旧的误差函数值时，将步长变为原来的一半
            descending_step_size *=0.5
        if(abs(new_ErrorFunction - old_ErrorFunction) < iteration_error): #如果新的误差函数值与旧的误差函数值的差小于迭代误差则终止迭代
            break
    poly = np.poly1d(w[::-1].reshape(polynomial_order + 1)) #先将 w 从后往前，得到多项式系数，按照阶数从高到低排列
    return poly
```

2.2.10 共轭梯度法

作用：

```
'''
    情况：加惩罚项，此时惩罚项系数为lamda
    对误差函数用共轭梯度法，循环迭代polynomial_order+1次，得到此时的w
'''
```

具体实现：与梯度下降法一样同样需要迭代，但是迭代次数仅为阶数+1 即可。通过对共轭梯度法的迭代公式进行迭代即可：

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

以注释形式给出

```
def Gradient_conjugate_method(sample_x, sample_y, lamda, sample_num, polynomial_order):
    X = CreateMatrixX(sample_x,sample_num,polynomial_order)    #将sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维向量
    Q = np.dot(X.T, X) + lamda * np.eye(X.shape[1])  #w 的系数 Q 为 X^{T} * X + lamda * 一个主对角线全为 1，其他元素全为 0 的标准矩阵
    w = np.zeros((polynomial_order + 1, 1)) #w 为一个行数为polynomial_order + 1，列数为 1 的矩阵
    Gradient = np.dot(X.T, X).dot(w) - np.dot(X.T, Y) + lamda * w #误差函数对 w 的偏导数公式为 X^{T} * X * w - X^{T} * Y + lamda * w，即梯度
    r = -Gradient #r 为负梯度
    p = r #从 p 为负梯度开始
    for i in range(polynomial_order + 1):  #迭代 polynomial_order + 1 次
        a = (r.T.dot(r)) / (p.T.dot(Q).dot(p))  #a = r^{T} * r / p^{T} * Q * p
        r_prev = r #这个 r
        w = w + a * p  #w = w + a * p
        r = r - (a * Q).dot(p)  #r = r + a * Q * p
        beita= (r.T.dot(r)) / (r_prev.T.dot(r_prev))  #beita = r^{T} * r / r_prev^{T} * r_prev
        p = r + beita * p #p = r + beita * p
    poly = np.poly1d(w[::-1].reshape(polynomial_order + 1))  #先将 w 从后往前，得到多项式系数，按照阶数从高到低排列
    return poly
```

2.2.11 绘制图像

作用:

```
'''
    进行图像的绘制
    sample_x：一维观测数据x
    sample_y：一维观测数据y
    poly_fit：拟合得到的多项式
    title：图像标题
'''
```

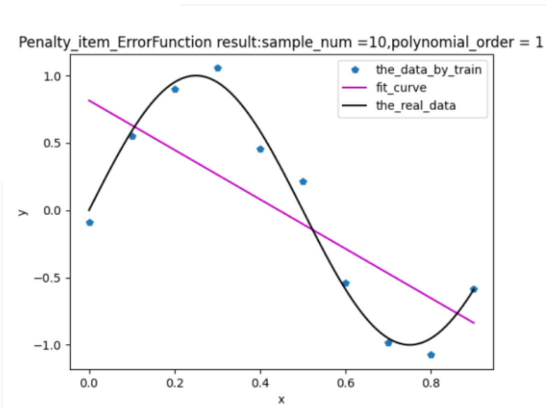
具体实现：以注释形式给出

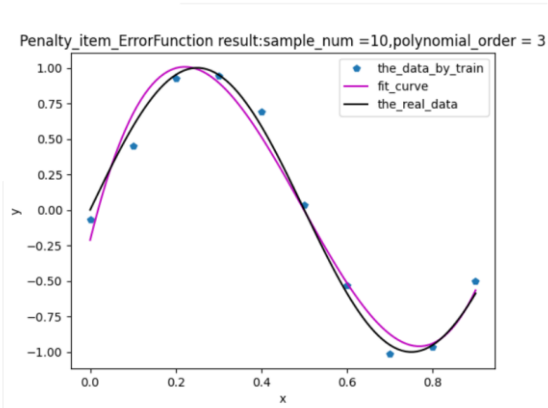
```
def Draw_Images(sample_x, sample_y, poly_fit, title):
    real_x = np.linspace(0, 0.9, 100)
    real_y = np.sin(real_x * 2 * np.pi)
    fit_y = poly_fit(real_x)
    plot1 = matplt.plot(sample_x, sample_y, 'p', label='the_data_by_train')  #测试数据
    plot2 = matplt.plot(real_x, fit_y, 'm', label='fit_curve')  #拟合曲线
    plot3 = matplt.plot(real_x, real_y, 'k', label='the_real_data')  #真实曲线
    matplt.xlabel('x')
    matplt.ylabel('y')
    matplt.legend(loc=1)
    matplt.title(title)
    matplt.show()
```

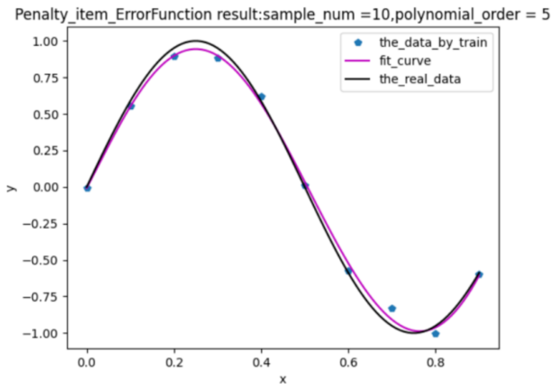
3 实验结果分析

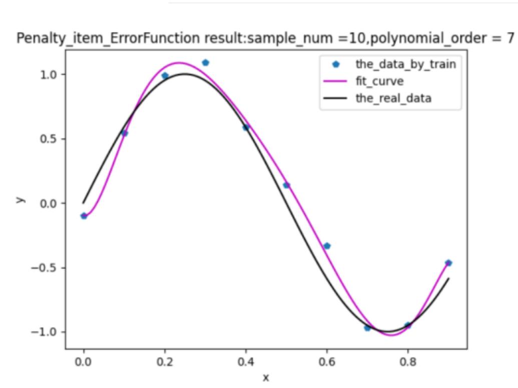
3.1 不带惩罚项，利用高阶多项式函数拟合曲线

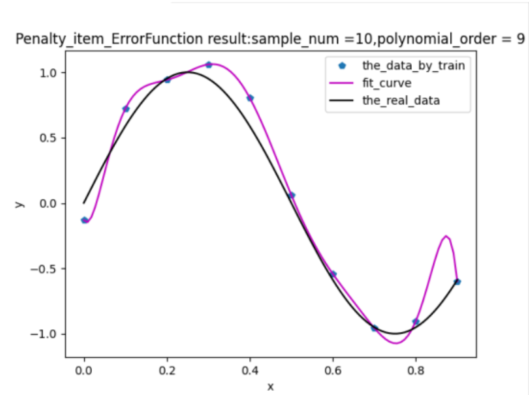
（1）此时训练样本的大小为 10，改变多项式的阶数可以得到这些曲线：

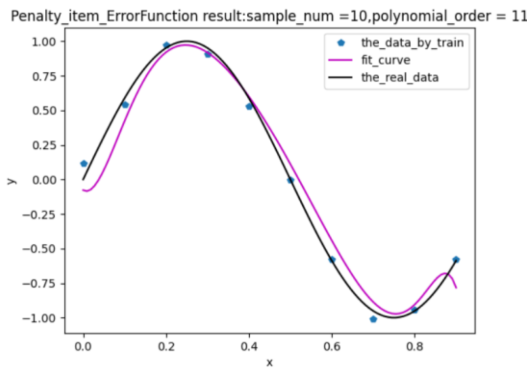












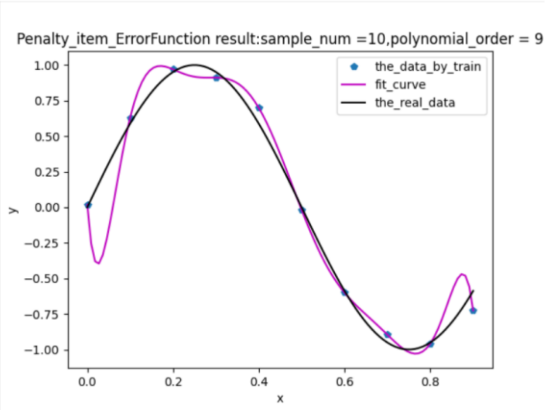
通过对图像进行分析可以得出以下结论：

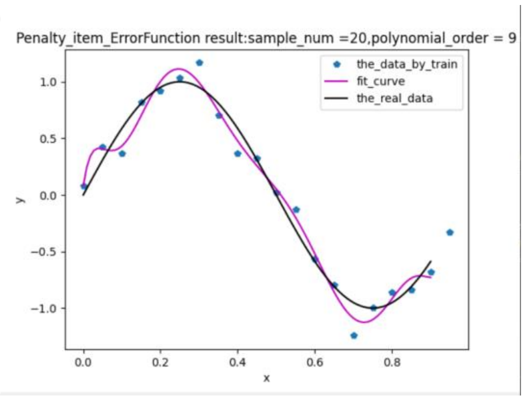
1.多项式阶数=3 时，图像的拟合程度已经很好。

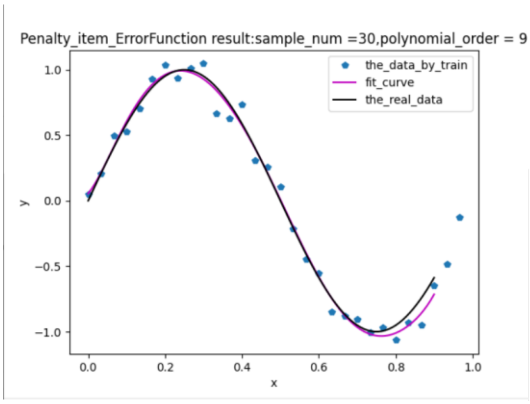
2.通过提高多项式的阶数，能够更好的拟合图像

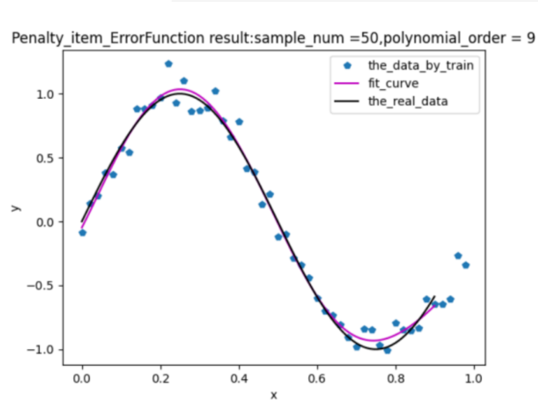
3.当多项式阶数=9 时，拟合的曲线经过了所有的样本点，但是图像却与真实图像差别很大，表现出了过拟合的现象。原因在于阶数很大时，模型拟合的能力得到了增强，但是可以通过增大系数或者减小系数来拟合所有数据点，甚至拟合了噪声。为了解决过拟合的为问题，可以通过增加样本的数量或者增加惩罚项来解决。

（2）此时固定多项式阶数，使用不同数量的样本数据可以得到以下曲线：







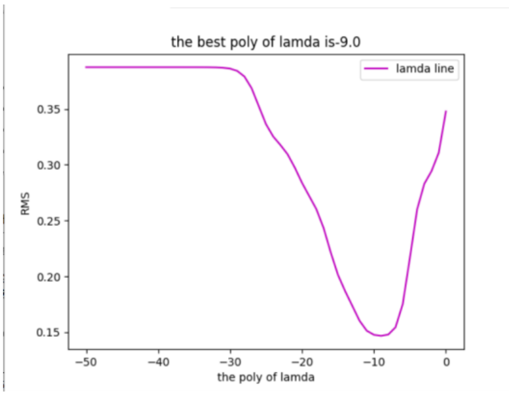


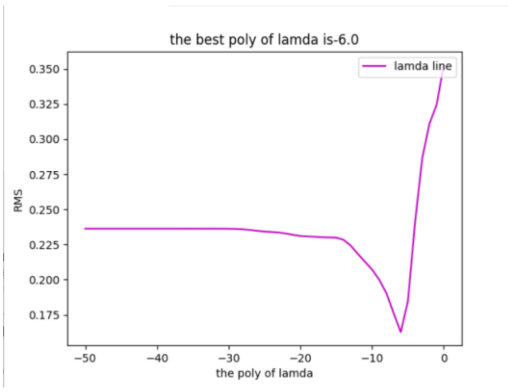
通过对图像进行分析可以得出以下结论：

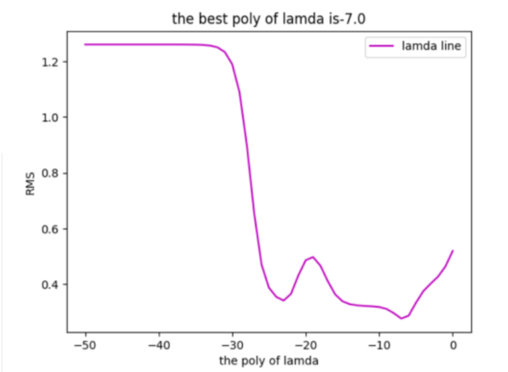
固定多项式阶数的情况下，随着样本数量的增加，可以缓解过拟合的现象。

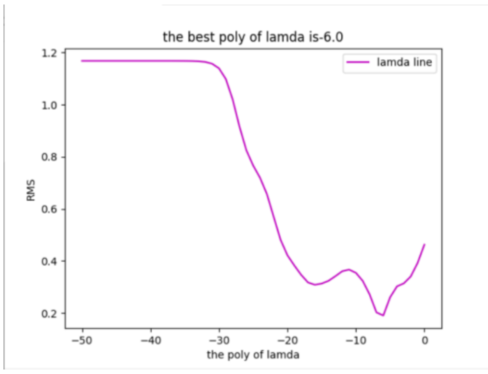
3.2 得到最佳惩罚项系数 lamda

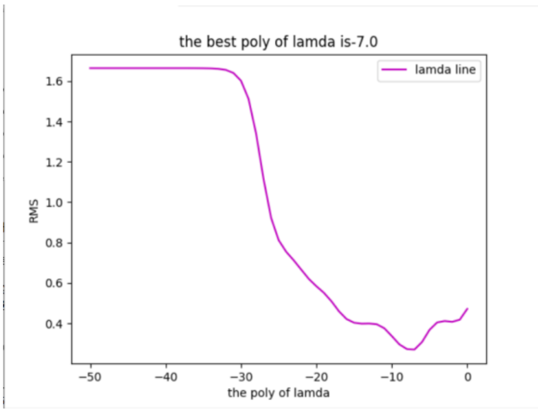
通过多次执行 Get_best_lamda()方法观察可得，比较合适的 lamda 范围在(e^{-9},e^{-6})之间，最佳的 lamda 结果为 e^{-7}，选取六次结果如下所示：

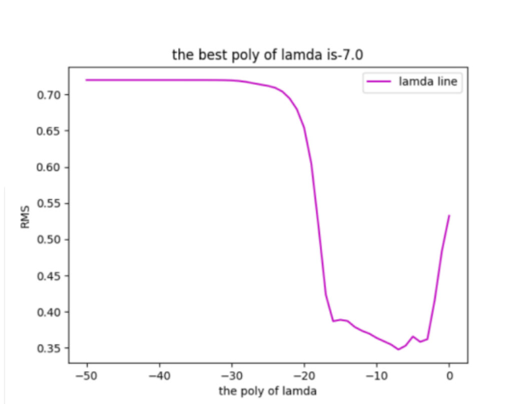










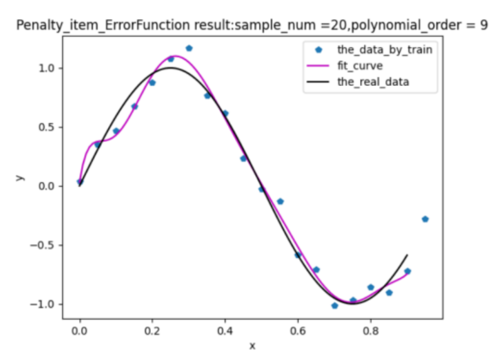


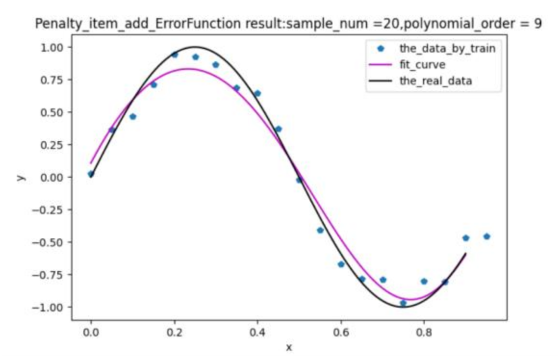
通过对图像进行分析可以得出以下结论：

经过多次实验可以得到出现次数最多的阶数为-7，则最佳的 lamda 结果为 e^{-7}，对于梯度下降法和共轭梯度法，所得的惩罚项系数 lamda 和这个通用

3.3 带惩罚项，利用高阶多项式函数拟合曲线

测试训练样本为 20，多项式阶数为 9 时无惩罚项和有惩罚项的函数拟合曲线，如下所示：





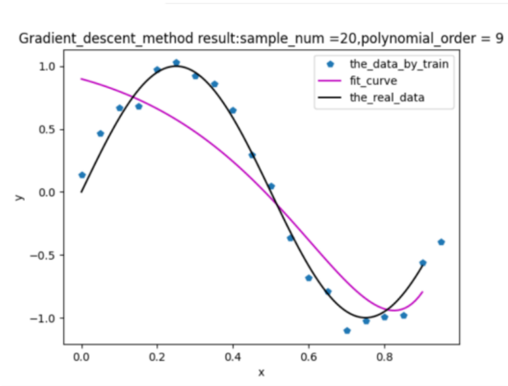
通过对图像进行分析可以得出以下结论：

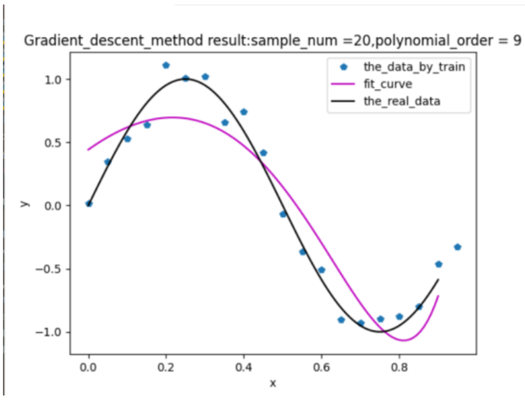
通过增加惩罚项能够降低过拟合，使拟合更加精确。

3.4 梯度下降法

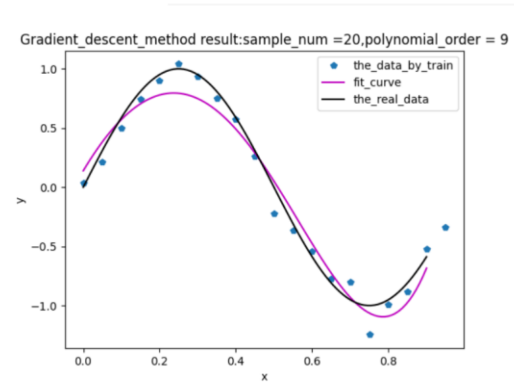
测试训练样本为 20，多项式阶数为 9 时不同迭代次数的图像如下：

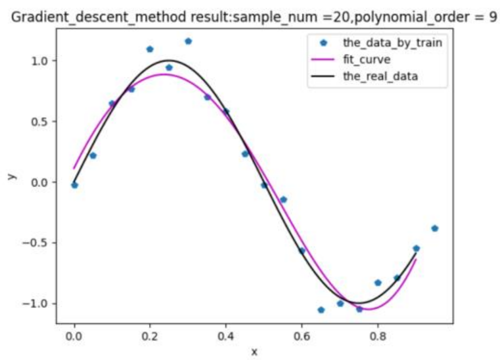
迭代次数为 100： 迭代次数为 1000





迭代次数为 10000： 迭代次数为 100000



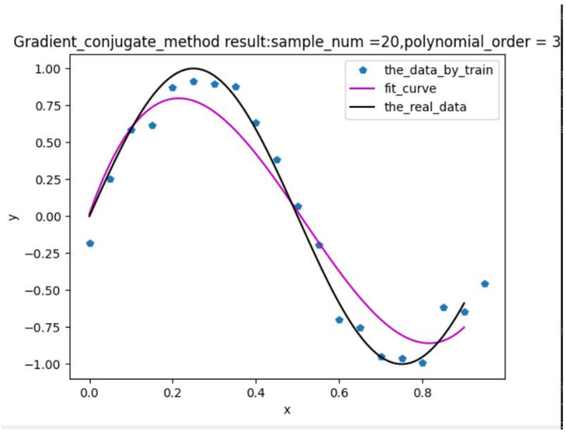


通过对图像进行分析可以得出以下结论：

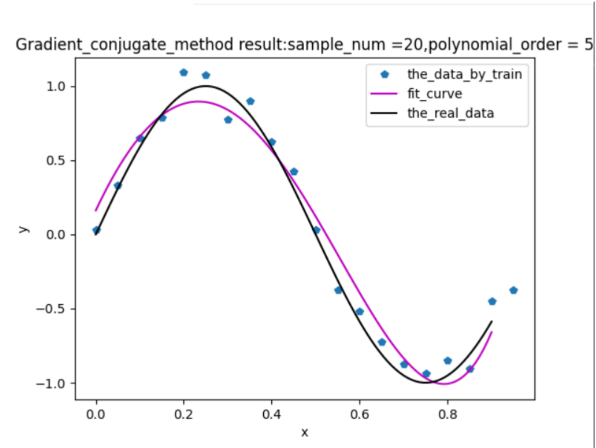
随着迭代次数的增加，图像的拟合程度增加。可以看出，梯度下降法得到比较准确地图像所需要的迭代次数非常多，并且图像的拟合程度并不是很优秀

3.5 共轭梯度法

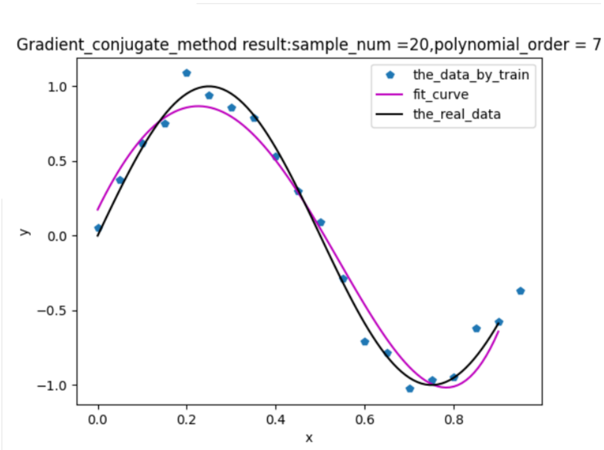
测试训练样本为 10，多项式阶数为 3 时不同迭代次数的图像如下，所需迭代次数为 4：



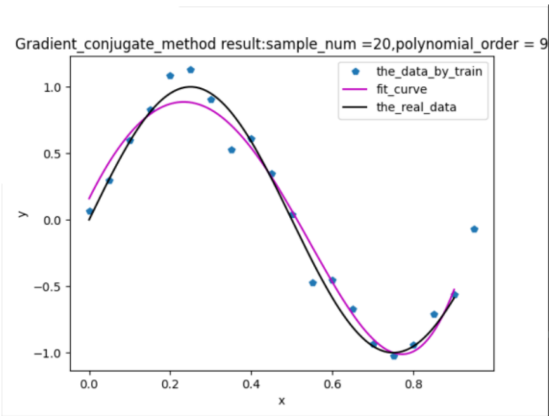
测试训练样本为 10，多项式阶数为 5 时不同迭代次数的图像如下，所需迭代次数为 6：



测试训练样本为 10，多项式阶数为 7 时不同迭代次数的图像如下，所需迭代次数为 8:



测试训练样本为 10，多项式阶数为 9 时不同迭代次数的图像如下，所需迭代次数为 10:



通过对图像进行分析可以得出以下结论：
共轭梯度法得到比较准确地图像所需要的迭代次数很少，仅为多项式阶数+1，并且图像的拟合程度很优秀

4 结论

1.正弦函数的多项式拟合中，多项式的次数越高，其模型能力越强
2.不加正则项的情况下，高次多项式的拟合效果出现了过拟合，过拟合是由于样本数量过

少，模型能力过强导致的,增加训练样本的数据可以有效的解决过拟合的问题。
3.增加惩罚项可以有效解决过拟合问题。
4.梯度下降法收敛速度较慢,所需要的迭代次数多，并且图像的拟合程度并不是很优秀。
5. 共轭梯度法收敛速度很快,所需要的迭代次数少，仅为多项式阶数+1，并且图像的拟合程度并不是很好。

5 源代码

Lab1-1180300829.py

```
import numpy as np
import matplotlib.pyplot as matplt
from Gradient_conjugate_method import Gradient_conjugate_method
from Gradient_descent_method import Gradient_descent_method
from Penalty_item_ErrorFunction import Penalty_item_ErrorFunction
from Penalty_item_add_ErrorFunction import
Penalty_item_add_ErrorFunction

global sample_num  #样本点的数量
global polynomial_order  #多项式的阶数
sample_num = 50
polynomial_order = 9



'''
    生成 sample_num 个数据，并加入噪声
    sample_x 为[0,0.9]上均匀的 sample_num 个点
    sample_y 为 sin(2πx)的值，再加上均值为 mu，方差为 sigma 的高斯噪声
'''
def CreateData(mu, sigma):
    sample_x = np.arange(0, 1, 1 / sample_num)      #创建[0,0.9]上均匀的
sample_num 个点
    gauss_noise = np.random.normal(mu, sigma, sample_num)  #创建均值为
mu，标准差为 sigma 的高斯噪声
    sample_y = np.sin(sample_x * 2 * np.pi) + gauss_noise  #对于
sample_x 得到每个 sin(2πx)的值，再加上均值为 mu，标准差为 sigma 的高斯噪声
    return sample_x, sample_y

'''
   将所有的样本 sample_x 进行转换，生成一个 X 矩阵
   在 CreateData 中生成的均匀的 sample_x 的样本数据是一维向量，需要预处理成为矩阵
X
   其中矩阵 X 的纬度为 sample_num * (polynomial_order + 1)
'''
```

```
def CreateMatrixX(sample_x):
    X = np.zeros((sample_num, polynomial_order + 1))  #先建立一个矩阵，行数为 sample_num，列数为 polynomial_order + 1
    for i in range(sample_num):  #对矩阵每一行进行赋值
        every_row_i = np.ones(polynomial_order + 1) * sample_x[i]  #先将每一行全部赋值为 sample_x[i]
        poly_row = np.arange(0, polynomial_order+1)  #得到每一列的阶数
        every_row_i = np.power(every_row_i, poly_row)  #得到这一行所有 sample_x[i]值的阶数值
        X[i] = every_row_i
    return X

'''
   误差函数 E(w)
   由已知得到误差函数的表达式为 E(w) = 1/2 * (Xw - Y)^{T} . (Xw - Y)
'''
def ErrorFunction(sample_x, sample_y, w):
    X = CreateMatrixX(sample_x)   #将 sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维向量
    temp = X.dot(w) - Y  #X 矩阵与 w 相乘后减去 Y
    ErrorFunction = 1/2 * np.dot(temp.T, temp)  #套用误差函数表达式即可
    return ErrorFunction

'''
   直接拟合数据，通过调用 np.polyfit()拟合数据
'''
def FittingData(sample_x, sample_y):
    w = np.polyfit(sample_x, sample_y, polynomial_order)  # 用 polynomial_order 次多项式拟合，
    poly = np.poly1d(w)  #得到多项式系数，按照阶数从高到低排列
    return poly

'''
    得到一组测试数据与真实数据的均方误差(RMS)
'''
def Get_RMS(train_num, poly_fit):
    real_x = np.linspace(0,1,train_num)       #真实的[0,1]上 train_num 个均匀点
    real_y = np.sin(real_x * 2 * np.pi)    #得到对应真实的 sin2pix 值
    fit_y = poly_fit(real_x)          #得到拟合值
    the_loss = real_y-fit_y      #得到拟合值和真实值的差
    the_RMS = np.sqrt((np.dot(the_loss,the_loss.T))/train_num)  #求解均方根，作为误差
```

```
    return the_RMS

'''
    得到最合适的惩罚项系数 lamda
'''
def Get_best_lamda():
    train_num = 100;       #设置测试数据容量为 100
    sample_x, sample_y = CreateData(0, 0.5)    #创建数据集
    the_degree = np.zeros(51)     #惩罚项系数阶数集
    the_RMS = np.zeros(51)      #惩罚项每个阶数对应的均方根集
    min = the_degree[0]
    min_num = 10000
    for i in range(0,51):      #对每个阶数进行均方根的求解
        the_degree[i] = -i     #为阶数集赋值
        the_best_lamda = np.exp(the_degree[i])  #得到每一个阶数对应的惩罚项
系数
        poly = Penalty_item_add_ErrorFunction(sample_x, sample_y,
the_best_lamda,sample_num,polynomial_order)  #求解多项式
        the_RMS[i] = Get_RMS(train_num,poly)   #对该多项式求解均方根
        if(min_num>the_RMS[i]):  #得到最小的均方根对应的惩罚项系数阶数
            min = the_degree[i]
            min_num = the_RMS[i]
    plot = matplt.plot(the_degree, the_RMS, 'm', label='lamda line')
    matplt.xlabel('the poly of lamda')
    matplt.ylabel('RMS')
    matplt.legend(loc=1)
    matplt.title("the best poly of lamda is"+str(min))
    matplt.show()

'''
   进行图像的绘制
    sample_x: 一维观测数据 x
    sample_y: 一维观测数据 y
   poly_fit: 拟合得到的多项式
    title: 图像标题
'''
def Draw_Images(sample_x, sample_y, poly_fit, title):
    real_x = np.linspace(0, 0.9, 100)
    real_y = np.sin(real_x * 2 * np.pi)
    fit_y = poly_fit(real_x)
    plot1 = matplt.plot(sample_x, sample_y, 'p',
label='the_data_by_train')  #测试数据
    plot2 = matplt.plot(real_x, fit_y, 'm', label='fit_curve')  #拟合曲
线
```

```
    plot3 = matplt.plot(real_x, real_y, 'k', label='the_real_data')  #
真实曲线
    matplt.xlabel('x')
    matplt.ylabel('y')
    matplt.legend(loc=1)
    matplt.title(title)
    matplt.show()

Get_best_lamda()  #得到最佳的惩罚项系数

sample_x, sample_y = CreateData(0, 0.1) #生成 sample_num 数据，并加入噪声

# 情况：用 np.polyfit 直接拟合
case1 = FittingData(sample_x, sample_y)
Draw_Images(sample_x, sample_y, case1, 'np.polyfit
result:'+'sample_num ='+str(sample_num)+',polynomial_order =
'+str(polynomial_order))
print(case1)

#情况：不加惩罚项
case2 = Penalty_item_ErrorFunction(sample_x,
sample_y,sample_num,polynomial_order)
Draw_Images(sample_x, sample_y, case2, 'Penalty_item_ErrorFunction
result:'+'sample_num ='+str(sample_num)+',polynomial_order =
'+str(polynomial_order))
print(case2)

# 情况：加惩罚项
lamda1 = np.exp(-7)
case3 = Penalty_item_add_ErrorFunction(sample_x, sample_y,
lamda1,sample_num,polynomial_order)
Draw_Images(sample_x, sample_y, case3,
'Penalty_item_add_ErrorFunction result:'+'sample_num
='+str(sample_num)+',polynomial_order = '+str(polynomial_order))
print(case3)

# 情况：加惩罚项,对误差函数使用梯度下降法
lamda2 = np.exp(-7)
descending_step_size= 0.05
cycle_times = 100000
iteration_error1 = 1e-5
case4 = Gradient_descent_method(sample_x, sample_y, lamda2,
cycle_times, descending_step_size,
iteration_error1,sample_num,polynomial_order)
```

```
Draw_Images(sample_x, sample_y, case4, 'Gradient_descent_method
result:'+'sample_num ='+str(sample_num)+',polynomial_order =
'+str(polynomial_order))
print(case4)

# 情况：加惩罚项,对误差函数使用共轭梯度法
lamda3 = np.exp(-7)
case5 = Gradient_conjugate_method(sample_x, sample_y,
lamda3,sample_num,polynomial_order)
Draw_Images(sample_x, sample_y, case5, 'Gradient_conjugate_method
result:'+'sample_num ='+str(sample_num)+',polynomial_order =
'+str(polynomial_order))
print(case5)
```

Penalty_item_ErrorFunction.py

```
import numpy as np


'''
   将所有的样本 sample_x 进行转换，生成一个 X 矩阵
   在 CreateData 中生成的均匀的 sample_x 的样本数据是一维向量，需要预处理成为矩阵
X
   其中矩阵 X 的纬度为 sample_num * (polynomial_order + 1)
'''
def CreateMatrixX(sample_x,sample_num,polynomial_order):
    X = np.zeros((sample_num, polynomial_order + 1))  #先建立一个矩阵，行
数为 sample_num，列数为 polynomial_order + 1
    for i in range(sample_num):  #对矩阵每一行进行赋值
        every_row_i = np.ones(polynomial_order + 1) * sample_x[i]  #先
将每一行全部赋值为 sample_x[i]
        poly_row = np.arange(0, polynomial_order+1)  #得到每一列的阶数
        every_row_i = np.power(every_row_i, poly_row)  #得到这一行所有
sample_x[i]值的阶数值
        X[i] = every_row_i
    return X


'''
    情况：不加惩罚项
    令误差函数导数等于 0，求此时的 w，此时的 w = (X^{T} * X)^{-1} * X^{T} * Y
'''
def Penalty_item_ErrorFunction(sample_x,
sample_y,sample_num,polynomial_order):
    X = CreateMatrixX(sample_x,sample_num, polynomial_order)   #将
```

```
sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维
向量
    w = np.linalg.inv(np.dot(X.T, X)).dot(X.T).dot(Y)   #套用公式 w =
(X^{T} * X)^{-1} * X^{T} * Y 得到 w
    poly = np.poly1d(w[::-1].reshape(polynomial_order + 1))  #先将 w 从
后往前，得到多项式系数，按照阶数从高到低排列
    return poly
```

Penalty_item_add_ErrorFunction.py

```
import numpy as np

'''
   将所有的样本 sample_x 进行转换，生成一个 X 矩阵
   在 CreateData 中生成的均匀的 sample_x 的样本数据是一维向量，需要预处理成为矩阵
X
   其中矩阵 X 的纬度为 sample_num * (polynomial_order + 1)
'''
def CreateMatrixX(sample_x,sample_num, polynomial_order):
    X = np.zeros((sample_num, polynomial_order + 1))  #先建立一个矩阵，行
数为 sample_num，列数为 polynomial_order + 1
    for i in range(sample_num):  #对矩阵每一行进行赋值
        every_row_i = np.ones(polynomial_order + 1) * sample_x[i]  #先
将每一行全部赋值为 sample_x[i]
        poly_row = np.arange(0, polynomial_order+1)  #得到每一列的阶数
        every_row_i = np.power(every_row_i, poly_row)  #得到这一行所有
sample_x[i]值的阶数值
        X[i] = every_row_i
    return X


'''
    情况：加惩罚项，此时惩罚项系数为 lamda
    令误差函数导数等于 0，求此时的 w，此时的 w = (X^{T} * X + lamda)^{-1} *
X^{T} * Y
'''
def Penalty_item_add_ErrorFunction(sample_x, sample_y,
lamda,sample_num,polynomial_order):
    X = CreateMatrixX(sample_x,sample_num, polynomial_order)    #将
sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))   #将 sample_y 变成一个竖着的一
维向量
    w = np.linalg.inv(np.dot(X.T, X) + lamda *
np.eye(X.shape[1])).dot(X.T).dot(Y)  #套用公式 w = (X^{T} * X +
lamda)^{-1} * X^{T} * Y 得到 w
```

```
    poly = np.poly1d(w[::-1].reshape(polynomial_order + 1))  #先将 w 从后往前，得到多项式系数，按照阶数从高到低排列
    return poly
```

Gradient_descent_method.py

```
import numpy as np

'''
   将所有的样本 sample_x 进行转换，生成一个 X 矩阵
   在 CreateData 中生成的均匀的 sample_x 的样本数据是一维向量，需要预处理成为矩阵 X
   其中矩阵 X 的纬度为 sample_num * (polynomial_order + 1)
'''
def CreateMatrixX(sample_x,sample_num,polynomial_order):
    X = np.zeros((sample_num, polynomial_order + 1))  #先建立一个矩阵，行数为 sample_num，列数为 polynomial_order + 1
    for i in range(sample_num):  #对矩阵每一行进行赋值
        every_row_i = np.ones(polynomial_order + 1) * sample_x[i]  #先将每一行全部赋值为 sample_x[i]
        poly_row = np.arange(0, polynomial_order+1)  #得到每一列的阶数
        every_row_i = np.power(every_row_i, poly_row)  #得到这一行所有 sample_x[i]值的阶数值
        X[i] = every_row_i
    return X

'''
   误差函数 E(w)
   由已知得到误差函数的表达式为 E(w) = 1/2 * (Xw - Y)^{T} . (Xw - Y)
'''
def ErrorFunction(sample_x, sample_y,sample_num,polynomial_order, w):
    X = CreateMatrixX(sample_x,sample_num,polynomial_order)   #将 sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维向量
    temp = X.dot(w) - Y  #X 矩阵与 w 相乘后减去 Y
    ErrorFunction = 1/2 * np.dot(temp.T, temp)  #套用误差函数表达式即可
    return ErrorFunction

'''
    情况：加惩罚项，此时惩罚项系数为 lamda
    对误差函数使用梯度下降法，当误差函数收敛到期望的最小值时，得到此时的 w
    参数中，最多循环轮次 cycle_times，下降步长 descending_step_size，迭代误差 iteration_error
'''
```

```
def Gradient_descent_method(sample_x, sample_y, lamda, cycle_times, descending_step_size, iteration_error,sample_num,polynomial_order):
    w = np.ones((polynomial_order + 1, 1))  #生成一个行数为 polynomial_order + 1，列数为 1 的矩阵，元素值全为 1
    X = CreateMatrixX(sample_x,sample_num, polynomial_order)  #将 sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维向量
    for i in range(cycle_times):  #迭代 cycle_times 次
        old_ErrorFunction = abs(ErrorFunction(sample_x, sample_y,sample_num,polynomial_order, w))  #求得旧的误差函数值的绝对值
        ErrorFunction_partial_deriv = X.T.dot(X).dot(w) - X.T.dot(Y) + lamda * w  #误差函数对 w 的偏导数公式为 X^{T} * X * w - X^{T} * Y + lamda * w，即梯度
        w = w - descending_step_size * ErrorFunction_partial_deriv  #梯度下降法的迭代公式为 w <--- w - descending_step_size * 误差函数对 w 的偏导数
        new_ErrorFunction = abs(ErrorFunction(sample_x, sample_y,sample_num,polynomial_order, w))  #求得新的误差函数值的绝对值
        if(new_ErrorFunction > old_ErrorFunction): #当新的误差函数值大于旧的误差函数值时，将步长变为原来的一半
            descending_step_size *=0.5
        if(abs(new_ErrorFunction - old_ErrorFunction) < iteration_error): #如果新的误差函数值与旧的误差函数值的差小于迭代误差则终止迭代
            break
    poly = np.poly1d(w[::-1].reshape(polynomial_order + 1)) #先将 w 从后往前，得到多项式系数，按照阶数从高到低排列
    return poly
```

Gradient_conjugate_method.py

```
import numpy as np

'''
  将所有的样本 sample_x 进行转换，生成一个 X 矩阵
  在 CreateData 中生成的均匀的 sample_x 的样本数据是一维向量，需要预处理成为矩阵 X
  其中矩阵 X 的纬度为 sample_num * (polynomial_order + 1)
'''
def CreateMatrixX(sample_x,sample_num,polynomial_order):
    X = np.zeros((sample_num, polynomial_order + 1))  #先建立一个矩阵，行数为 sample_num，列数为 polynomial_order + 1
    for i in range(sample_num):  #对矩阵每一行进行赋值
        every_row_i = np.ones(polynomial_order + 1) * sample_x[i]  #先将每一行全部赋值为 sample_x[i]
        poly_row = np.arange(0, polynomial_order+1)  #得到每一列的阶数
```

```
        every_row_i = np.power(every_row_i, poly_row)  #得到这一行所有sample_x[i]值的阶数值
        X[i] = every_row_i
    return X

'''
   误差函数 E(w)
   由已知得到误差函数的表达式为 E(w) = 1/2 * (Xw - Y)^{T} . (Xw - Y)
'''
def ErrorFunction(sample_x, sample_y,sample_num, w):
    X = CreateMatrixX(sample_x)   #将 sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维向量
    temp = X.dot(w) - Y  #X 矩阵与 w 相乘后减去 Y
    ErrorFunction = 1/2 * np.dot(temp.T, temp)  #套用误差函数表达式即可
    return ErrorFunction

'''
    情况：加惩罚项，此时惩罚项系数为 lamda
    对误差函数用共轭梯度法，循环迭代 polynomial_order+1 次，得到此时的 w
'''
def Gradient_conjugate_method(sample_x, sample_y, lamda, sample_num, polynomial_order):
    X = CreateMatrixX(sample_x,sample_num,polynomial_order)    #将sample_x 进行转换，生成一个 X 矩阵
    Y = sample_y.reshape((sample_num, 1))  #将 sample_y 变成一个竖着的一维向量
    Q = np.dot(X.T, X) + lamda * np.eye(X.shape[1])  #w 的系数 Q 为 X^{T} * X + lamda * 一个主对角线全为 1，其他元素全为 0 的标准矩阵
    w = np.zeros((polynomial_order + 1, 1)) #w 为一个行数为polynomial_order + 1，列数为 1 的矩阵
    Gradient = np.dot(X.T, X).dot(w) - np.dot(X.T, Y) + lamda * w #误差函数对 w 的偏导数公式为 X^{T} * X * w - X^{T} * Y + lamda * w，即梯度
    r = -Gradient #r 为负梯度
    p = r #从 p 为负梯度开始
    for i in range(polynomial_order + 1):  #迭代 polynomial_order + 1 次
        a = (r.T.dot(r)) / (p.T.dot(Q).dot(p))  #a = r^{T} * r / p^{T} * Q * p
        r_prev = r #这个 r
        w = w + a * p  #w = w + a * p
        r = r - (a * Q).dot(p)  #r = r + a * Q * p
        beita= (r.T.dot(r)) / (r_prev.T.dot(r_prev))  #beita = r^{T} * r / r_prev^{T} * r_prev
        p = r + beita * p #p = r + beita * p
```

```
    poly = np.poly1d(w[::-1].reshape(polynomial_order + 1))  #先将 w 从后往前，得到多项式系数，按照阶数从高到低排列
    return poly
```