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： 多项式拟合正弦函数

学号：

姓名：

一、实验目的

掌握最小二乘法求解（无惩罚项的损失函数）、掌握加惩罚项（2范数）的损失函数优化、梯度下降法、共轭梯度法、理解过拟合、克服过拟合的方法(如加惩罚项、增加样本)

二、实验要求及实验环境

实验要求：

1. 生成数据，加入噪声；
2. 用高阶多项式函数拟合曲线
3. 用解析解求解两种loss的最优解（无正则项和有正则项）
4. 优化方法求解最优解（梯度下降，共轭梯度）；
5. 用你得到的实验数据，解释过拟合。
6. 用不同数据量，不同超参数，不同的多项式阶数，比较实验效果。
7. 语言不限，可以用matlab，python。求解解析解时可以利用现成的矩阵求逆。梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不许用现成的平台，例如pytorch，tensorflow的自动微分工具。

实验环境：

Win10 1903; Python 3.7.4; Pycharm Community Edition

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 算法原理

利用多项式函数拟合数据点，多项式函数形式如下：

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

令

$$W=\begin{bmatrix}w_0\\w_1\\\vdots\\w_m\end{bmatrix},X=\begin{bmatrix}1&x_1&\cdots&x_1^m\\1&x_2&\cdots&x_2^m\\\vdots&\vdots&\ddots&\vdots\\1&x_n&\cdots&x_n^m\end{bmatrix}$$

则多项式函数可化为线性代数形式：

$$y(x,W)=XW \tag{2}$$

最小二乘法

为了取得误差函数的最小值，直接令函数的导数等于0，可以得到唯一解。此解称为解析解。

1.1 用高阶多项式函数拟合曲线(不带正则项)

利用训练集，对于每个新的x，预测目标值y。采用多项式函数进行学习，即利用式(1)来确定参数W。

误差函数为：

$$E(W)=\frac{1}{2}\sum_{n=1}^{N}\left(y\left(x_n,W\right)-t_n\right)^2=\frac{1}{2}(XW-T)^{\mathrm{T}}(XW-T) \tag{3}$$

对式(3)求导得：

$$\frac{\partial E(W)}{\partial W}=X^{\mathrm{T}}XW-X^{\mathrm{T}}T \tag{4}$$

令导数等于0得：

$$W = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} T \tag{5}$$

1.2 用高阶多项式函数拟合曲线(带正则项)

为了解决过拟合问题，可以增加样本数据，也可以加入正则项(惩罚项)，降低模型复杂度以减少过拟合。

误差函数变为：

$$E(W) = \frac{1}{2}\sum_{n=1}^{N}\left\{y\left(x_n, W\right) - t_n\right\}^2 + \frac{\lambda}{2}\|W\|_2^2 = \frac{1}{2}[(XW-T)'(XW-T) + \lambda W'W] \tag{6}$$

λ也就是正则系数，是一个超参数，需要设置一个较好的值才能得到较好的结果。

对式(1.2.1)求导得：

$$\frac{\partial E(W)}{\partial W} = \frac{1}{m+1}\left(\left(X^{\mathrm{T}} X + \lambda E_{m+1}\right) W - X^{\mathrm{T}} T\right) \tag{7}$$

令导数等于0得：

$$W = \left(X^{\mathrm{T}} X + \lambda E_{m+1}\right)^{-1} X^{\mathrm{T}} T \tag{8}$$

梯度下降法

梯度是微分中一个很重要的概念：

1. 在单变量函数中，梯度其实就是微分，代表着函数在某个给定点的切线的斜率
2. 在多变量函数中，梯度的方向就指出了函数在给定点的上升最快的方向

对于$f(x_i)$如果在x_i处可微，由于梯度$\nabla f(x_i)$的方向是函数在给定点上升最快的方向，那么梯度的反方向$-\nabla f(x_i)$就是函数在给定点下降最快的方向，因而有式(9)：

$$x_{\mathbf{i+1}} = x_{\mathbf{i}} - \alpha \nabla f\left(x_{\mathbf{i}}\right) \tag{9}$$

其中α在梯度下降法中被称作**学习率**或者**步长**，意味着可以通过α来控制每一步走的距离，如果过大，会错过最低点，如果过小，需要许多步才能到达最低点。因此，α的选择在梯度下降法中极为重要。

定义一个代价函数，选用均方误差代价函数：

$$f(W) = \frac{1}{2m}(XW-T)^T(XW-T) \tag{10}$$

梯度为：

$$\nabla f(W) = \frac{1}{m}X^T(XW-T)$$

共轭梯度法

共轭梯度法是介于梯度下降法(最速下降法)和牛顿法之间的一个方法，是一个一阶方法，仅需利用一阶导数信息，但克服了梯度下降法收敛慢的缺点，又避免了存储和计算牛顿法所需要的二阶导数信息。

共轭梯度法适用于对于形如$Ax = b$的线性方程组解的问题(其中A对称正定)，**主要思想**就是找到n个两两共轭的共轭方向，每次沿着一个方向求取最小值，每一方向求解时不影响其他方向。实现过程主要分为三步：

1. 梯度下降法求得$r^{(0)}$，即$r^{(0)} = -\nabla f(x^{(0)})$，计算$\alpha_0 = \frac{(r^{(0)}, r^{(0)})}{(Ar^{(0)}, r^{(0)})}$，$x^{(1)} = x^{(0)} + \alpha_0 r^{(0)}$，记$r^{(0)} = d^{(0)}$，计算$r^{(1)} = b - Ax^{(1)} \neq 0$，此时$r^{(1)} \perp d^{(0)}$

2. 已知$x^{(1)}, r^{(1)}, d^{(0)}$，从$x^{(1)}$出发，沿$d^{(0)}$的共轭方向$d^{(1)}$做一维搜索，即求
   $\beta \in \mathbb{R}\ s.t.\ d^{(1)} = r^{(1)} + \beta d^{(0)}$与$d^{(0)}$共轭，即$(d^{(1)}, Ad^{(0)}) = (r^{(1)} + \beta d^{(0)}, Ad^{(0)}) = 0$，
   $(r^{(1)}, Ad^{(0)}) + \beta (d^{(0)}, Ad^{(0)}) = 0$，故$\beta = -\frac{(r^{(1)}, Ad^{(0)})}{(d^{(0)}, Ad^{(0)})}$再求$minf(x^{(1)} + \alpha d^{(1)})$得
   $\alpha_1 = \frac{(r^{(1)}, d^{(1)})}{Ad^{(1)}, d^{(1)}}, x^{(2)} = x^{(1)} + \alpha_1 d^{(1)}$
3. 同第2步，得到结论

$$
\begin{cases}
(r^{(i)}, r^{(j)}) = 0 \quad (i \neq j) \\
(d^{(i)}, Ad^{(j)}) = 0 \quad (i \neq j) \\
\{d^{(1)}\}_{i=0}^{k} \quad \text{共轭向量组}\left(\text{线性无关}\right)\left(k \leqslant n,\ k\text{为迭代次数}\right)
\end{cases} \tag{11}
$$

2. 算法的实现

梯度下降法

利用带惩罚项的代价函数进行梯度下降法的实现

$$
J(W) = \left(X^T X + \lambda I\right) W - X^T T \tag{12}
$$

设置精度要求当$\delta \leqslant 1^{-6}$时说明已经进入了比较平滑的状态，这个时候可以退出循环。

$Algorithm\ 1$ 梯度下降法

$$
\begin{aligned}
&rate = 0.01 \\
&k = 0 \\
&\mathbf{W}_0 = \mathbf{0} \\
&\mathbf{E}(\mathbf{W}) = \tfrac{1}{2m}[(\mathbf{XT} - \mathbf{T})^{\mathrm{T}}(\mathbf{XW} - \mathbf{T}) + \lambda \mathbf{W}^{\mathrm{T}}\mathbf{W}] \\
&\mathbf{J}(\mathbf{W}) = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I})\mathbf{W} - \mathbf{X}^{\mathrm{T}}\mathbf{T} \\
&\mathbf{loss}_0 = \mathbf{E}(\mathbf{W}_0) \\
&\mathbf{repeat:} \\
&\quad \mathbf{W}_{k+1} = \mathbf{W}_k - rate * \mathbf{J}(\mathbf{W_k}) \\
&\quad \mathbf{loss}_{k+1} = \mathbf{E}(\mathbf{W}_{k+1}) \\
&\quad \mathbf{if}: \mathbf{abs}(\mathbf{gradient}_{k+1}) < \delta\ \mathbf{then\ break\ loop} \\
&\quad k = k + 1 \\
&\mathbf{end\ repeat}
\end{aligned}
$$

共轭梯度法

形如$Ax = b$的线性方程组解的问题，实现步骤与算法原理中的实现步骤相同，下面给出伪代码，其中

$$
\begin{cases}
A = X^{\mathrm{T}} X \\
b = X^{\mathrm{T}} T
\end{cases} \tag{13}
$$

使得A是对称正定的。

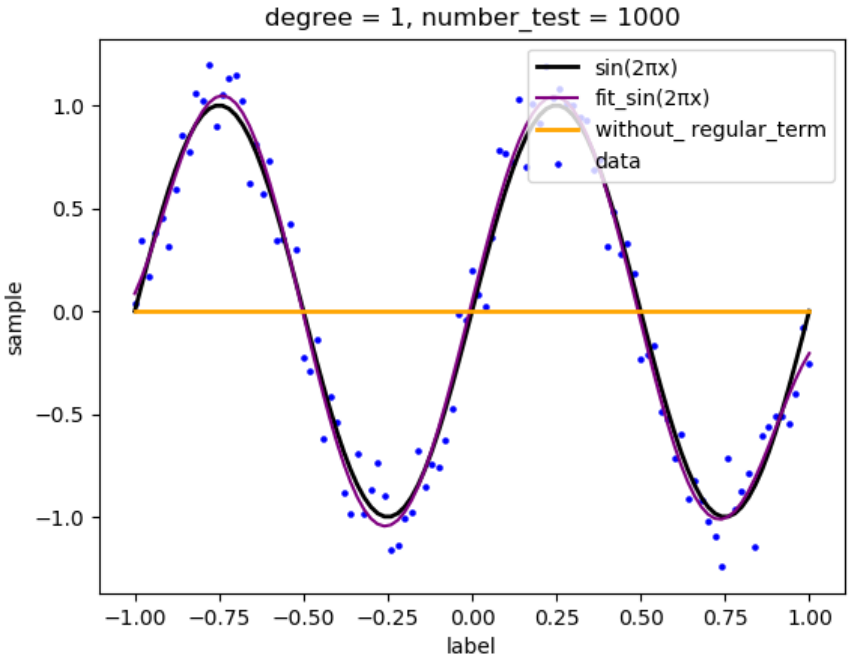
$Algorithm\ 2$ 共轭梯度法

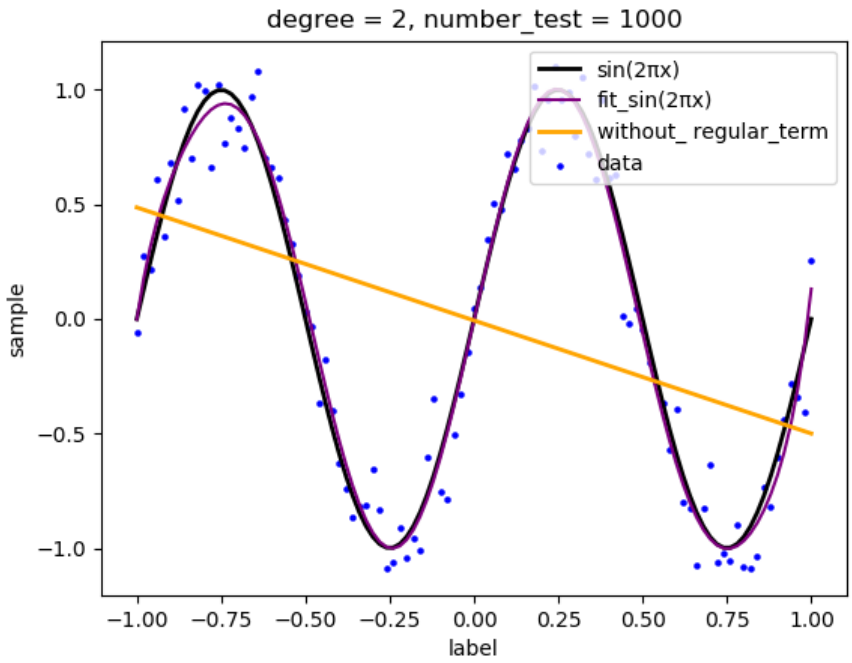
$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
$k = 0$
repeat
$\alpha_k = \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$
$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$
$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
if r_{k+1} **is sufficiently small, then exit loop**
$\beta_k = \frac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$
$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
$k = k + 1$
end repeat

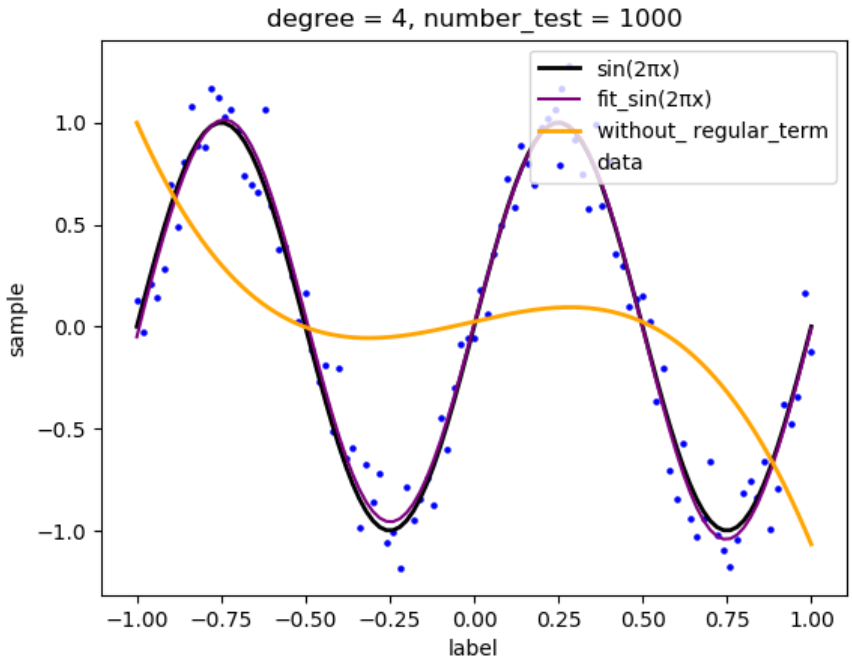
四、实验结果与分析

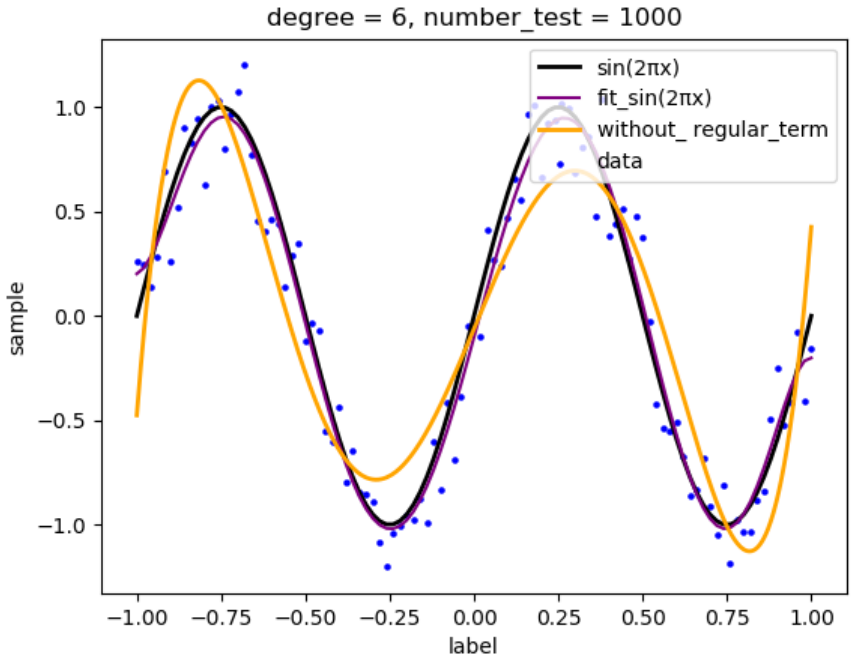
1. 不带正则项的解析解

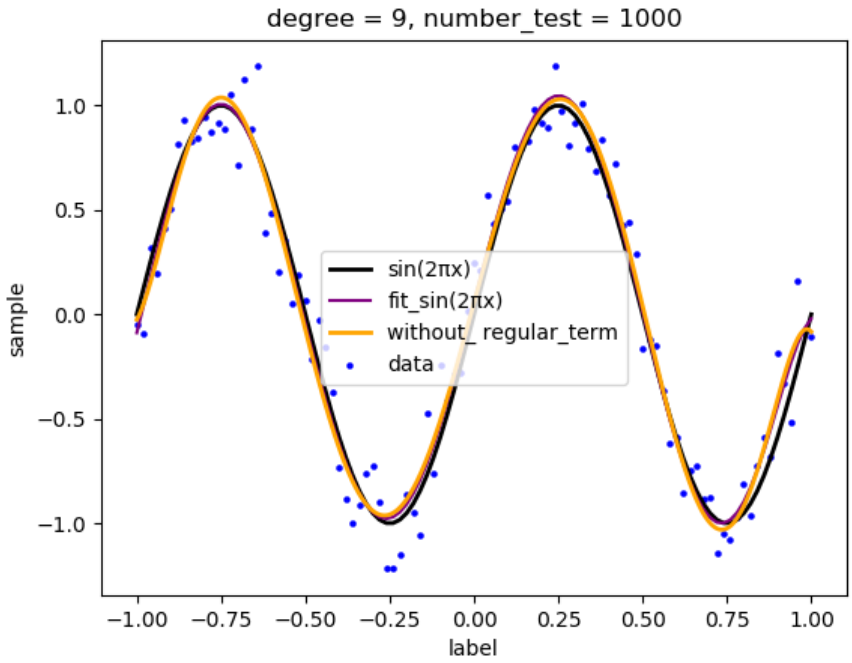
黑色表示$sin(2\pi x)$的图像，紫色为使用numpy.polyfit包拟合的曲线阶数为9，橙色为不带正则项的多项式拟合曲线。

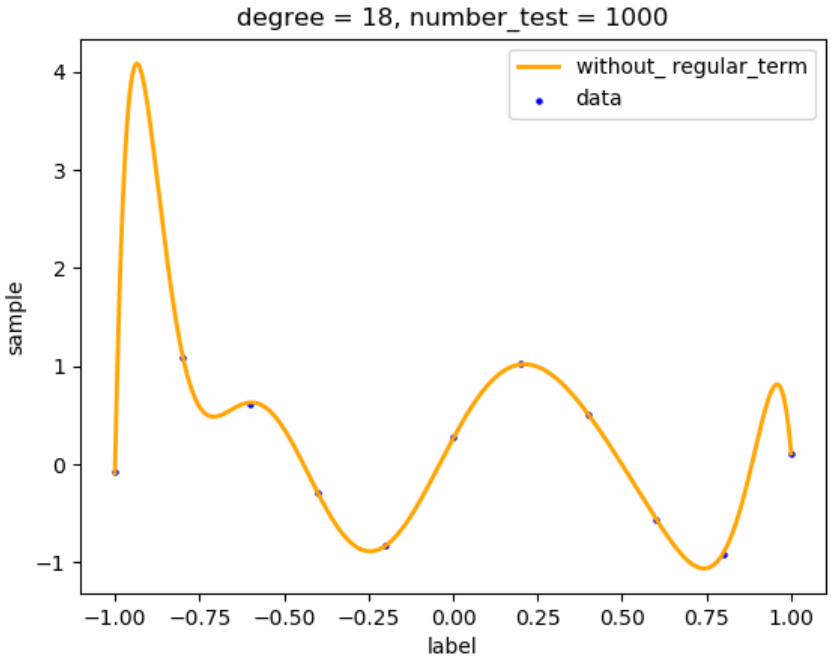



degree = 2, number_test = 1000
sin(2πx)
fit_sin(2πx)
without_ regular_term
data
sample
label
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


degree = 4, number_test = 1000
sin(2πx)
fit_sin(2πx)
without_ regular_term
data
sample
label
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


degree = 6, number_test = 1000
sin(2πx)
fit_sin(2πx)
without_ regular_term
data
sample
label
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

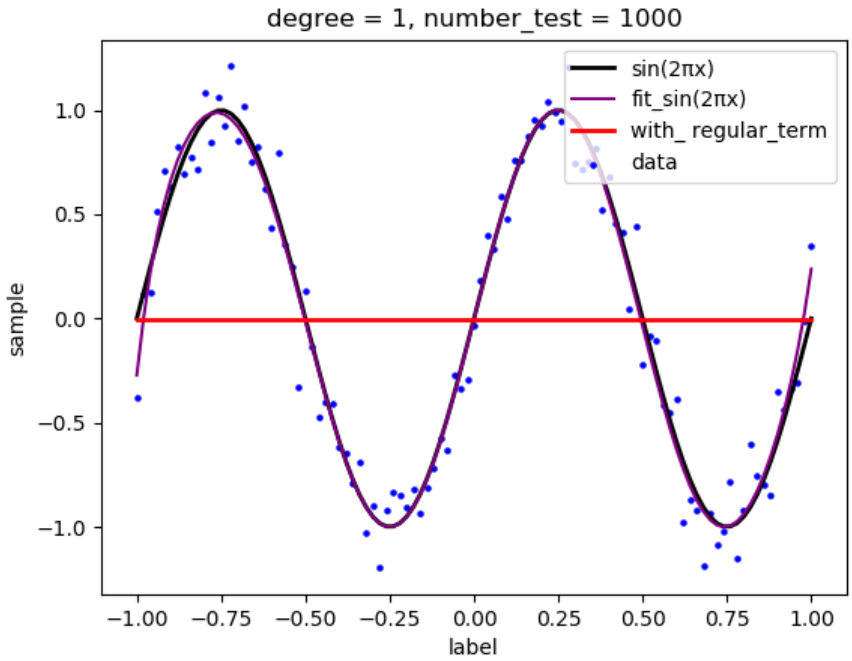

degree = 9, number_test = 1000
sin(2πx)
fit_sin(2πx)
without_ regular_term
data
sample
label
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

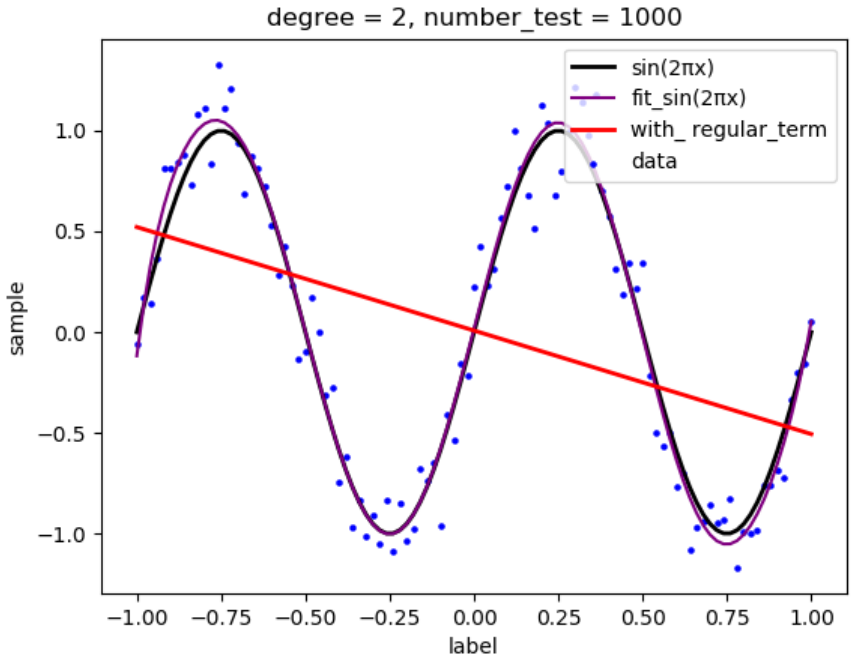


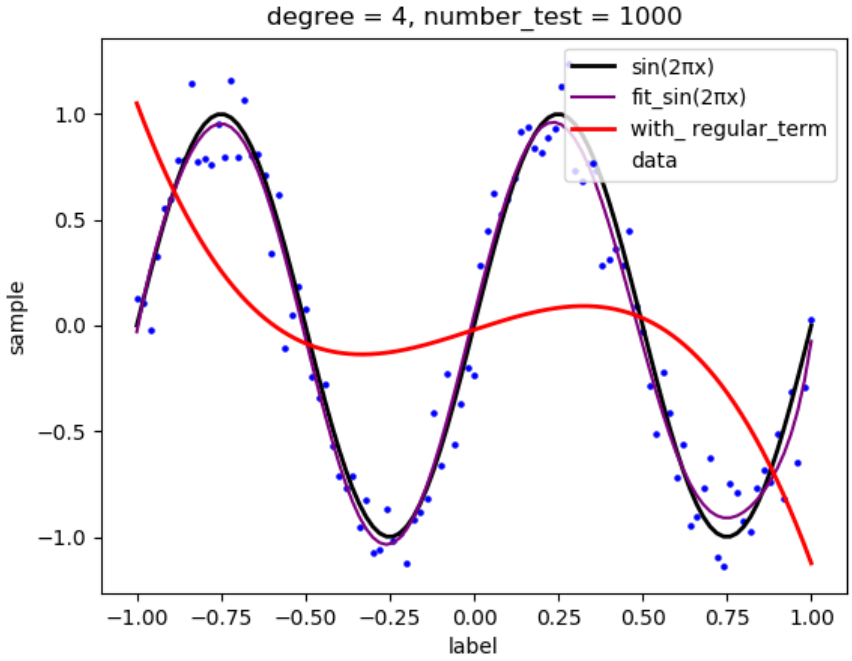
可以看出当阶数较小时(M=1 4)时可以看出，模型表达能力有限，曲线欠拟合；当阶数M=6 13时，拟合效果较好，当阶数M⩾15时，曲线过拟合如上图6所示，为了使过拟合效果明显，将训练数据减少为10个，但是当依旧使用大小为100的训练集时，阶数为18时过拟合效果并不明显，因此对于过拟合我们可以通过增加样本的数据或者通过增加正则项的方式来解决。

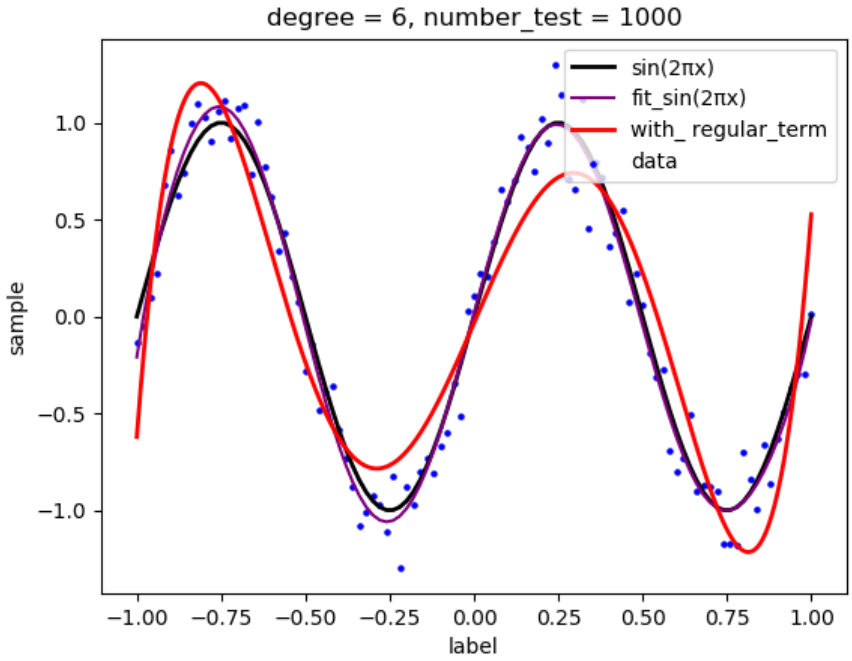
2.带正则项的解析解

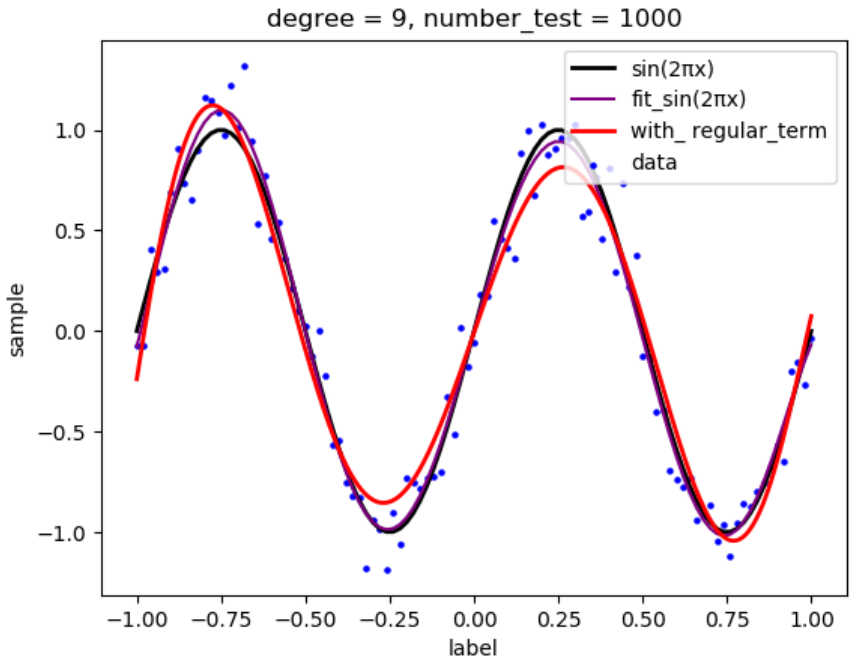
黑色表示$sin(2\pi x)$的图像，紫色为使用numpy.polyfit包拟合的曲线阶数为9，红色为带正则项的多项式拟合曲线。**超参数**$\lambda = \mathbf{1e^{-3}}$

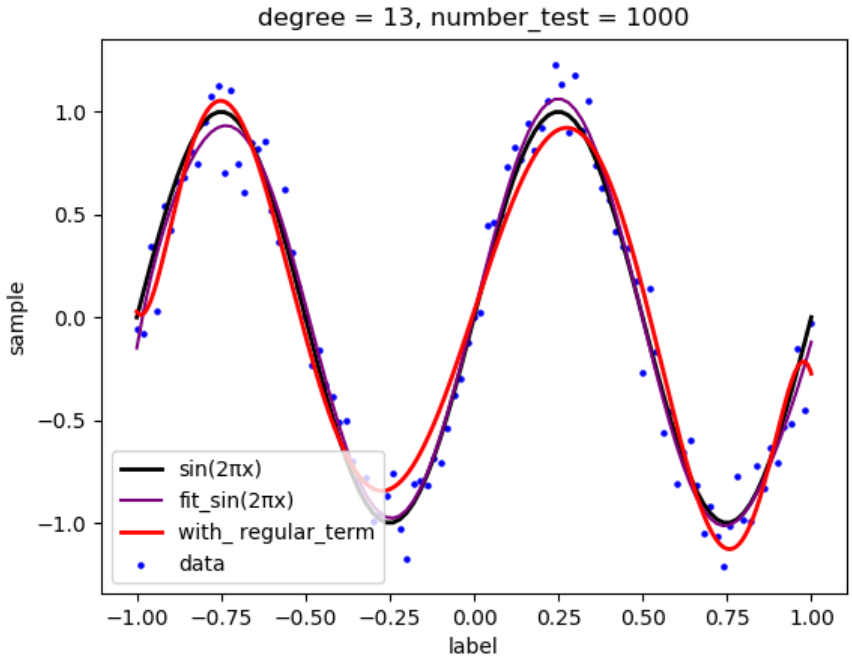


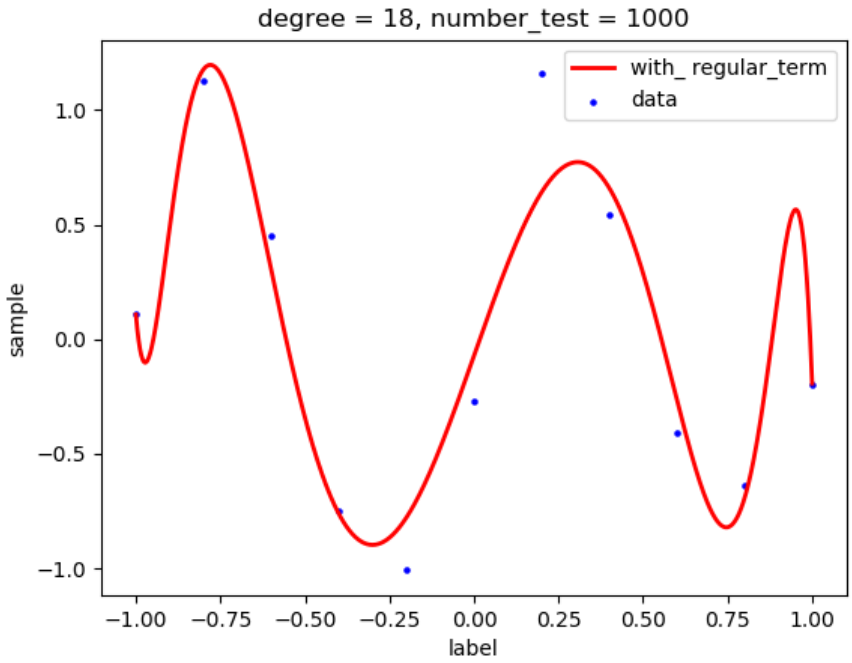

degree = 2, number_test = 1000
sin(2πx)
fit_sin(2πx)
with_ regular_term
data
sample
label
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


degree = 4, number_test = 1000
sin(2πx)
fit_sin(2πx)
with_ regular_term
data
sample
label
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

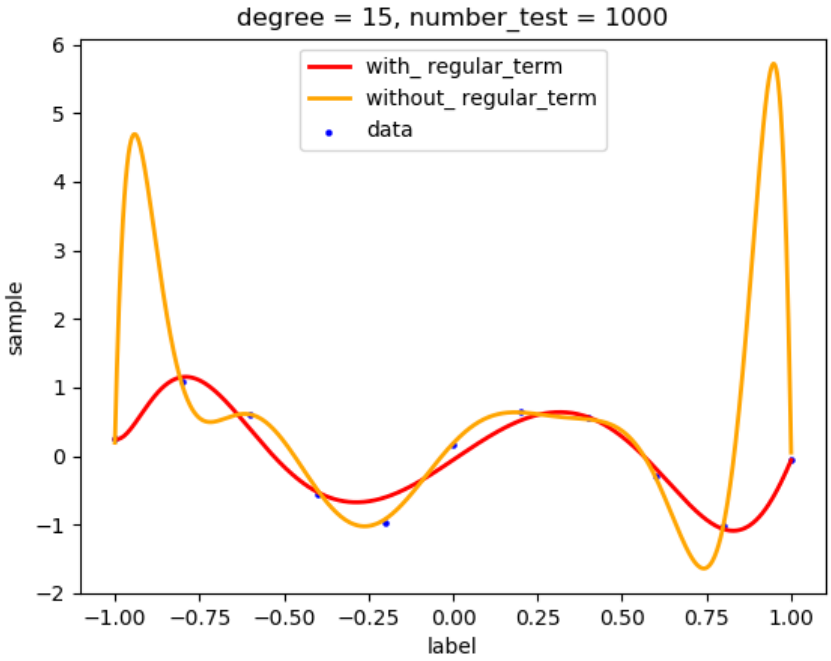

degree = 6, number_test = 1000
sin(2πx)
fit_sin(2πx)
with_ regular_term
data
sample
label
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


degree = 9, number_test = 1000
sin(2πx)
fit_sin(2πx)
with_ regular_term
data
sample
label
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00





可以看出总体规律同不带正则项的曲线拟合规律基本相似，阶数低时欠拟合，当阶数M=6~13时效果较好，当阶数M过大时曲线过拟合。但是对比过拟合曲线时可以发现，加正则项的过拟合曲线是有影响的(见下图)，会降低模型的复杂度。

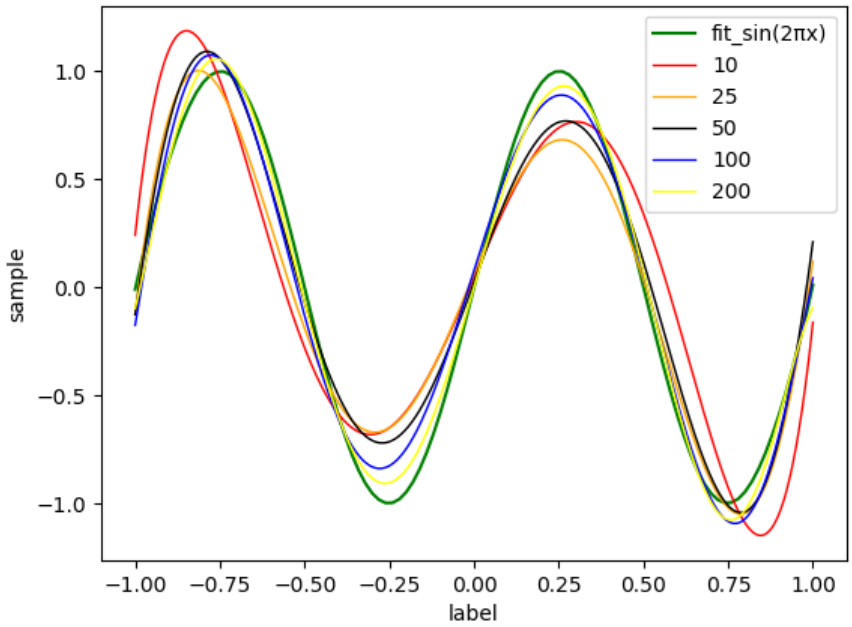


采用带正则项的解析解，比较实验结果

通过上述实验可发现当阶数为9是拟合效果最好，以下实验都在阶数为9的条件下进行。

不同数据量

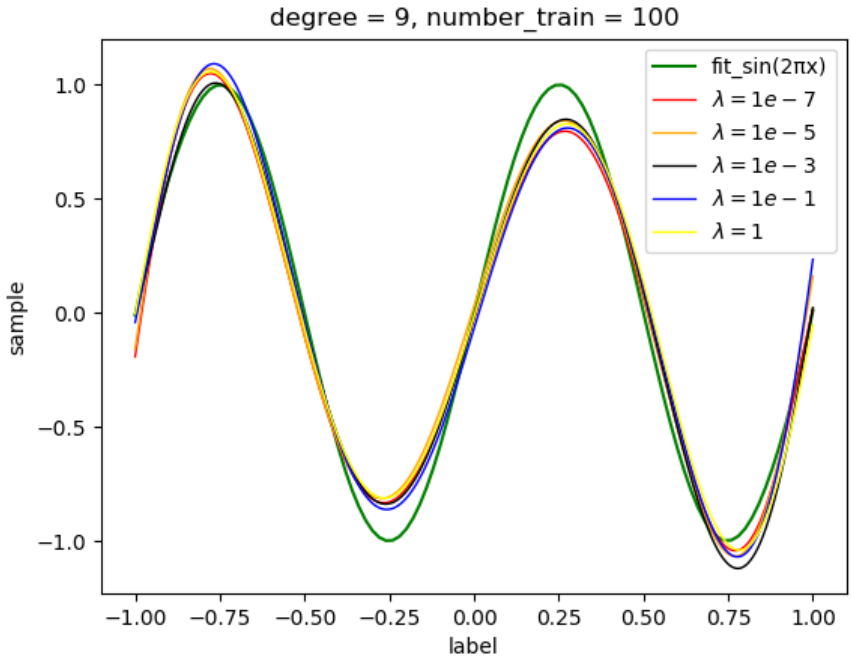
选取不同规模的训练集进行对比：



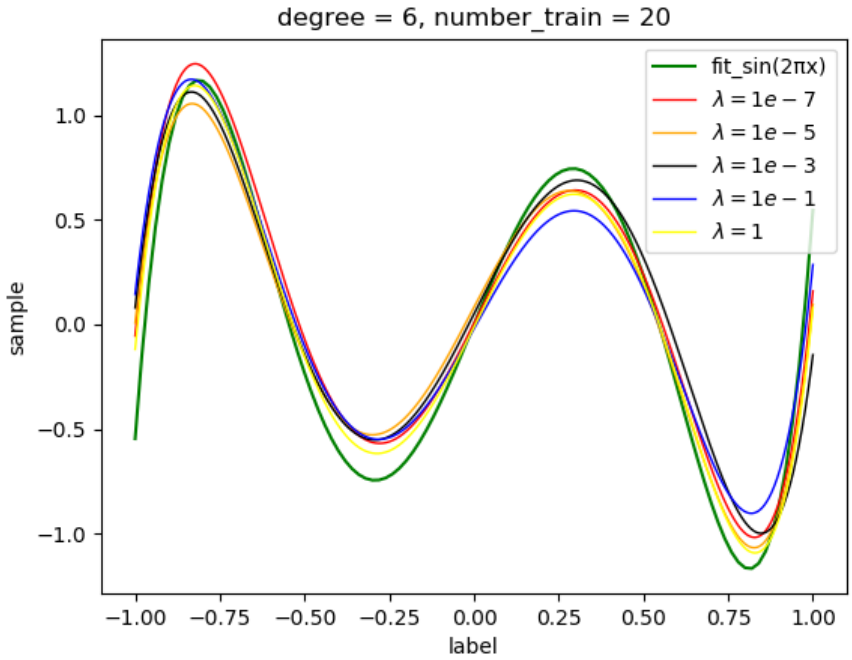
由上图可以得知，当曲线阶数选择适当时，**训练数据越多，拟合效果就越好。**

不同超参数

对于不同的超参λ进行实验对比，得到结果如下：



此处实验发现不同超参下，图像的差别并不明显，因此更改了多项式阶数并且减少了训练数据集大小进行了第二次实验，结果如下：



图像对比依旧不明显，根据Pattern Recognition and Machine Learning中提到的跟均方误差E_{RMS}来区别，则有以下公式：

$$E(W^*) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, W) - t_n\}^2 + \frac{\lambda}{2}\|W\|^2 = \frac{1}{2}[(XW-T)'(XW-T) + \lambda W'W] \quad (14)$$

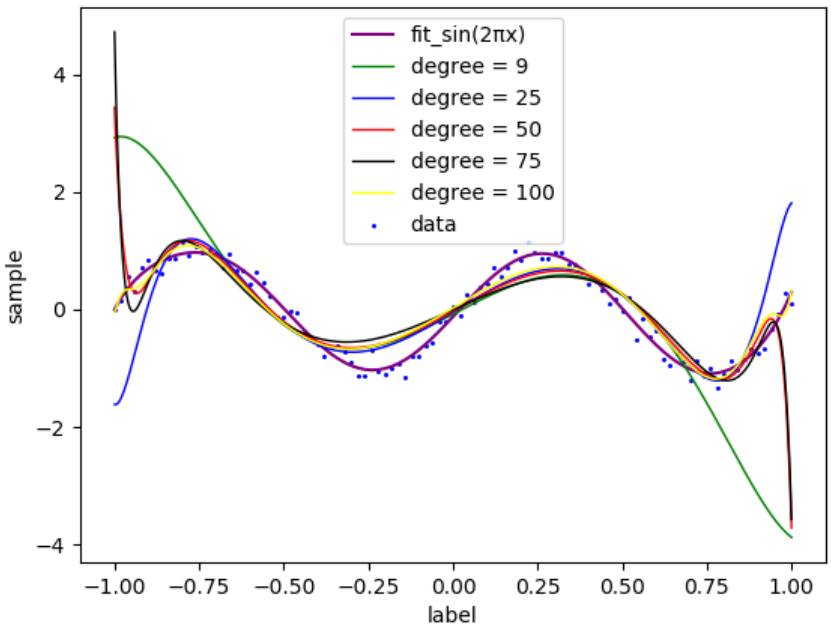
$$E_{RMS} = \sqrt{\frac{2E(W^*)}{N}}$$

λ	1e-7	1e-5	1e-3	1e-1	1
E_{RMS}	0.179	0.188	0.241	1.451	4.401

可以发现当超参λ越小时，效果越好。

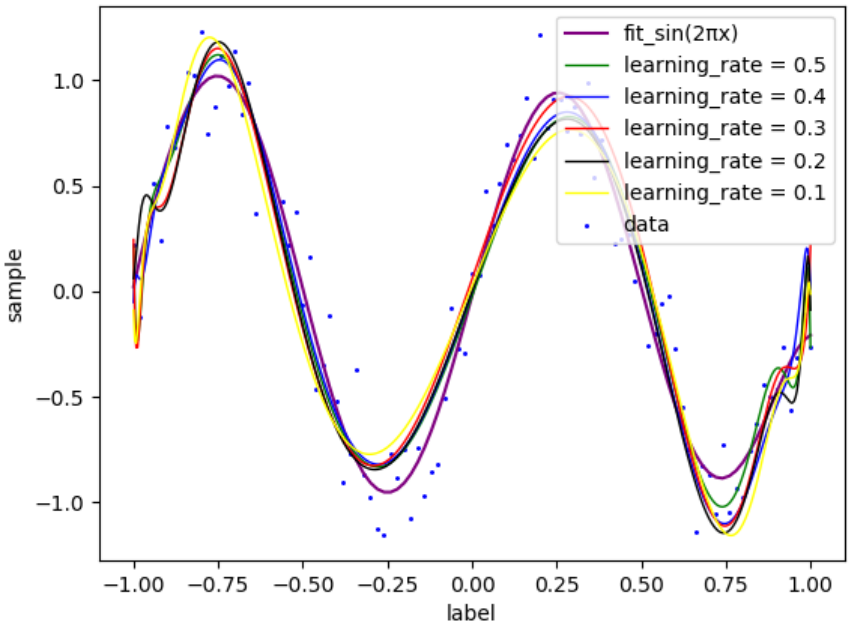
梯度下降法

在学习率α = 0.01，停止精度$\delta \leqslant 1e-5$条件下进行实验，测试不同多项式阶数拟合情况



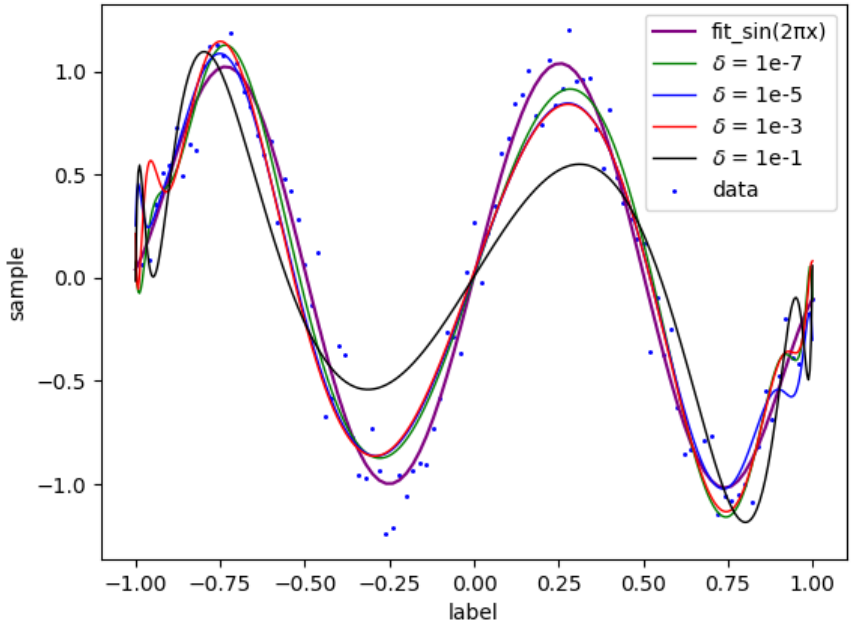
根据结果可以发现，当degree = 100时效果最好，造成这样结果的原因可能参数矩阵维数和样本维数基本相同，超高次多项式的过拟合效应在所取样本点之外效果明显，导致两端出现过拟合强烈的现象，因此当多项式阶数与系数矩阵维数相等时效果最好是正确的。

在degree = 100，停止精度$\delta \leqslant 1e-5$条件下进行实验，测试不同学习率拟合情况



根据结果可知，当α=0.3时的拟合效果在以上实验结果中表现最好，经分析可知，当α大时，步长过大，可能错过最低点，因此效果不佳；当α小时，根据程序迭代次数发现，学习率小的时候迭代次数很少，分析可能是由于学习率过小，在某一次迭代中，梯度小，小于停止精度，就跳出循环。在改进时不能只判断停止精度，应当加上次数限制，当连续多少次均小于停止精度时确保已经收敛才可以停止。

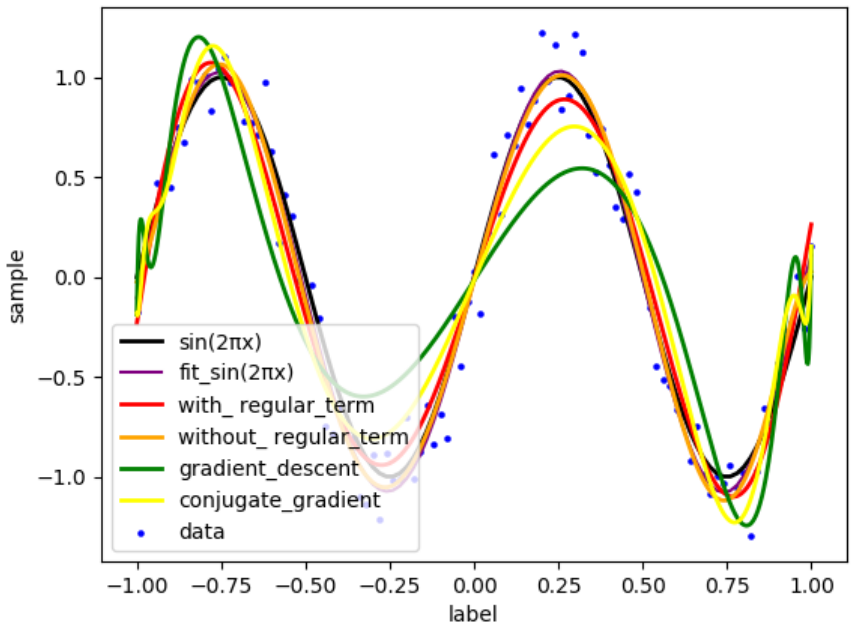
在degree = 100，学习率α = 0.01条件下进行实验，测试不同停止精度拟合情况



根据结果可以看到，当停止精度越小时，拟合效果越好，这也与梯度下降法的原理相吻合。

共轭梯度法

从图像中可以得到结果，在多项式阶数小的时候，会出现欠拟合现象，在高阶多项式中会出现过拟合现象且在边界处较为明显。通过对比训练集规模不难发现，当训练集的规模越大时，拟合出的曲线效果越好越接近解析解。并且在运行时，共轭梯度法相较于梯度下降法来说节省了大量的时间，并且在同等条件下拟合效果相较于梯度下降法更好，如下图所示。



五、结论

1. **增大训练集的规模可以解决过拟合问题**
2. 若更改训练集的规模不能很好地解决过拟合问题，**可以通过增加正则项的方法来降低模型复杂度从而解决过拟合问题**

3. 梯度下降法相较于共轭梯度法而言，**所需时间更长，需要更多的迭代次数，并且就拟合效果来说，共轭梯度法的效果更好，因此共轭梯度法相对来讲由于梯度下降法**
4. 不同的训练集，超参的变量会导致拟合效果不同，**需要权衡各个参数的影响选取最佳的参数组合**

六、参考文献

[1] 《机器学习》周志华 《机器学习》周志华 .北京 .清华大学出版社

[2] 梯度下降法 wiki

[3] 共轭梯度法 wiki

[4] Pattern Recognition and Machine Learning

七、附录：源代码（带注释）

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


def without_regular_term(x, Y):
    '''
    不带正则项的解析解
    W = (X.T * X).I * X.T * T
    '''
    A = x.T * x
    A = A.I
    w = A * x.T * Y
    return w


def with_regular_term(x, Y, degree):
    '''
    带正则项的解析解
    W = (X.T * X + lambda * E(m+1)).I * X.T * T
    '''
    A = x.T * x + 0.001 * np.eye(degree)
    A = A.I
    w = A * x.T * Y
    return w


def loss(W, X, Y, lamda):
    '''
    损失函数
    '''
    return (1 / X.shape[0]) * ((X * W - Y).T * (X * W - Y) + lamda * W.T * W)


def gradient(W, X, Y, lamda):
    '''
    计算带正则项的梯度
    '''
    return (1 / X.shape[0]) * (X.T * X * W - X.T * Y + lamda * W)


```

```python
def gradient_descent(X, Y, learning_rate, lamda):
    time = 0
    W = np.mat(np.zeros(X.shape[0])).T
    grad = gradient(W, X, Y, lamda)
    while np.all(np.absolute(loss(W, X, Y, lamda)) > 1e-5):
        W = W - learning_rate * grad
        grad = gradient(W, X, Y, lamda)
        time += 1
        if time >= 100000:
            break
    return W


def conjugate_gradient(X, Y, W):
    '''
    共轭梯度法
    形如Ax=b，其中A是对称正定的
    '''
    A = X.T * X + 0.01 * np.eye(101)
    b = X.T * Y
    r = b - A * W
    p = r
    for i in range(1, 100):
        alpha = (r.T * r) / (p.T * A * p)
        r_prev = r
        W = W + np.multiply(alpha, p)
        r = r - np.multiply(alpha , A * p)
        if all(abs(r)) <= 0.01:
            break
        beta = (r.T * r) / (r_prev.T * r_prev)
        p = r + np.multiply(beta, p)
    return W


if __name__ == '__main__':
    X = np.arange(-1, 1.02, 0.02)   #生成x，y
    Y = np.sin(2 * np.pi * X)
    plt.plot(X, Y, color="black", linewidth=2, label='sin(2πx)')
    mu = 0
    sigma = 0.15
    scale = X.size
    degree = 9
    learning_rate = 0.01
    lamda = 0.001
    W = np.mat(np.zeros((101, 1)))

    for i in range(scale):
        Y[i] += random.gauss(mu, sigma)     #加高斯噪声
    plt.scatter(X, Y, marker='.', color='blue', s=20, label='data')

    poly = np.polyfit(X, Y, deg=9)      #numpy拟合函数，用作对比
    z = np.polyval(poly, X)
    plt.plot(X, z, color='purple', label='fit_sin(2πx)')

    X = np.mat(X).T
    y = np.mat(Y).T
    x = np.power(X, 0)
    x_0 = np.power(X, 0)
```

```
100
101     test_set = np.linspace(-1, 1, 1000)
102
103     for i in range(1, degree):
104         temp = np.power(X, i)
105         x_0 = np.hstack((x_0, temp))         #范德蒙德行列式
106
107     cofficient_matrix = with_regular_term(x_0, y, degree)
108     func = 0 * test_set
109     for i in range(0, degree):
110         func += (np.double(cofficient_matrix[i])) * (test_set ** (i))
111     plt.plot(test_set, func, color="red", linewidth=2, label='with_
    regular_term')  #带正则项的曲线
112
113     cofficient_matrix = without_regular_term(x_0, y)
114     func = 0 * test_set
115     for i in range(0, degree):
116         func += (np.double(cofficient_matrix[i])) * (test_set ** (i))
117     plt.plot(test_set, func, color="orange", linewidth=2, label='without_
    regular_term')    #不带正则项的曲线
118
119     for i in range(1, x.shape[0]):
120         temp = np.power(X, i)
121         x = np.hstack((x, temp))
122
123     W = gradient_descent(x, y, learning_rate, lamda)
124
125     func = 0 * test_set
126     for i in range(0, x.shape[0]):
127         func += (np.double(W[i])) * (test_set**i)
128     plt.plot(test_set, func, color="green", linewidth=2,
    label='gradient_descent')  #梯度下降法
129
130     W = np.mat(np.zeros((101, 1)))
131     W = conjugate_gradient(x, y, W)
132     func = 0 * test_set
133     for i in range(0, x.shape[0]):
134         func += (np.double(W[i])) * (test_set ** i)
135     plt.plot(test_set, func, color="yellow", linewidth=2,
    label='conjugate_gradient')    #共轭梯度法
136
137
138     plt.xlabel("label")
139     plt.ylabel("sample")
140     plt.legend(loc='best')
141     plt.show()
142
```