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： 多项式拟合正弦曲线

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：

PyCharm 2020.3.2 x64
Python 3.8　numpy1.20.3　matplotlib 3.4.2

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 生成数据

使用 np.linspace 函数生成 X 的样本点，并用$\mathrm{y} = \sin(2\pi \mathrm{x})$生成数据，样本点 x 均匀分布在[0,1]之间。对生成的样本点添加方差为 Sigma，均值为 0 的高斯噪声。

2. 用高阶多项式函数拟合曲线

采用多项式函数拟合生成数据的样本点的曲线，以预测每个$\hat{x}$对应的值。

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

采用最小二乘法计算每个样本点目标值及预测函数预测值的误差。

$$\mathrm{J}(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{1}$$

假设$X_{(i)}$、W为下式：

$$X_{(i)} = \begin{vmatrix} x_{(i)}^{0} \\ \cdots \\ x_{(i)}^{n} \end{vmatrix}$$

$$\mathrm{W} = \begin{vmatrix} \theta_0 \\ \cdots \\ \theta_n \end{vmatrix}$$

$$h_\theta\left(x^{(i)}\right) = X_{(i)}^{T} W$$

$$\mathrm{J}(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right) - y^{(i)})^2$$

$$\mathrm{J}(\theta) = \frac{1}{2}\begin{pmatrix} h_\theta\left(x^{(0)}\right) - y^{(0)} \\ \cdots \\ h_\theta\left(x^{(m-1)}\right) - y^{(m-1)} \end{pmatrix}^T \begin{pmatrix} h_\theta\left(x^{(0)}\right) - y^{(0)} \\ \cdots \\ h_\theta\left(x^{(m-1)}\right) - y^{(m-1)} \end{pmatrix}$$

将$X_{(i)}$、W带入可得：

$$\mathrm{J}(\theta) = \frac{1}{2}\begin{pmatrix} X_0^T W - y^{(0)} \\ \cdots \\ X_{m-1}^T W - y^{(m-1)} \end{pmatrix}^T \begin{pmatrix} X_0^T W - y^{(0)} \\ \cdots \\ X_{m-1}^T W - y^{(m-1)} \end{pmatrix}$$

$$\mathrm{J}(\theta) = \frac{1}{2}(\begin{pmatrix} X_0^T W \\ \cdots \\ X_{m-1}^T W \end{pmatrix} - \begin{pmatrix} y^{(0)} \\ \cdots \\ y^{(m-1)} \end{pmatrix})^T(\begin{pmatrix} X_0^T W \\ \cdots \\ X_{m-1}^T W \end{pmatrix} - \begin{pmatrix} y^{(0)} \\ \cdots \\ y^{(m-1)} \end{pmatrix})$$

$$\mathrm{J}(\theta) = \frac{1}{2}(XW - Y)^T(XW - Y)$$

$$\mathrm{J}(\theta) = \frac{1}{2}(W^T X^T - Y^T)(XW - Y)$$

$$\mathrm{J}(\theta) = \frac{1}{2}(W^T X^T XW - W^T X^T Y - Y^T XW + Y^T Y) \tag{2}$$

至此，得到公式 2.

$$\text{方程 1：}\ \frac{\partial X^{\mathrm{T}}\mathrm{A}}{\partial \mathrm{X}} = \frac{\partial A^{\mathrm{T}}\mathrm{X}}{\partial \mathrm{X}} = A$$

$$\text{方程 2：}\ \frac{\partial X^{\mathrm{T}}\mathrm{BX}}{\partial \mathrm{X}} = (\mathrm{B} + B^T)\mathrm{X}$$

$$\frac{\partial}{\partial \mathrm{W}}\mathrm{J}(\theta) = X^{\mathrm{T}}XW - X^{\mathrm{T}}Y \tag{3}$$

通过将方程 1 和方程 2 代入，我们可以得到公式 4，

$$\mathrm{W} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}Y \tag{4}$$

3. 用带惩罚项的高阶多项式函数拟合曲线

使用不带惩罚项的高阶多项式函数拟合曲线易使W偏大，发生过拟合。因此

可以使用正则项，保留所有特征的同时，调整W的大小。

对公式 1 引入正则项，可以得到带惩罚项的高阶多项式函数：

$$\mathrm{J}(\theta)=\frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2+\frac{\lambda}{2}\|W\| \tag{5}$$

根据公式 2，可以得到以矩阵形式展开的代价函数：

$$\mathrm{J}(\theta)=\frac{1}{2}(W^TX^TXW-W^TX^TY-Y^TXW+Y^TY)+\frac{\lambda}{2}W^TW \tag{6}$$

对公式 6 求导，我们可以得到：

$$\frac{\partial}{\partial \mathrm{W}}\mathrm{J}(\theta)=X^{\mathrm{T}}XW-X^{\mathrm{T}}Y+\lambda W \tag{7}$$

进一步，令公式 7 为 0，我们可以得到：

$$X^{\mathrm{T}}XW-X^{\mathrm{T}}Y+\lambda W=0 \tag{8}$$

因此有：

$$(X^{\mathrm{T}}X+\lambda I)W-X^{\mathrm{T}}Y=0$$
$$(X^{\mathrm{T}}X+\lambda I)W=X^{\mathrm{T}}Y$$
$$W=(X^{\mathrm{T}}X+\lambda I)^{-1}X^{\mathrm{T}}Y \tag{9}$$

需要选取合适的 lambda 作为惩罚系数。

岭回归一般采用均方误差 MSE 来衡量 lambda：

$$\mathrm{MSE}=\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

4. 优化方法求解最优解（梯度下降法）

梯度下降法是求解无约束优化问题最简单、经典的方法。当n非常大的时候，计算$(X^{\mathrm{T}}X)^{-1}$时间过长，此时使用梯度下降法，效率往往更高。考虑一个无约束优化问题$\min\limits_{\mathrm{x}}\mathrm{f(x)}$， 其中$\mathrm{f(x)}$为连续可微函数， 如果我们能够构造一个序列$x^0,x^1,x^2,\ldots$，并能够满足$\mathrm{f}(x^{t+1})<\mathrm{f}(x^t),\mathrm{t}=0,1,2,\ldots$，则可以收敛到最小值。

我们实现带惩罚项的高阶多项式进行梯度下降法，由公式 7 可得：

$$\frac{\partial}{\partial \mathrm{W}}\mathrm{J}(\theta)=(X^{\mathrm{T}}X+\lambda)W-X^{\mathrm{T}}Y \tag{10}$$

梯度下降法：

$$\mathrm{W}=\mathrm{W}-\alpha\frac{\partial}{\partial \mathrm{W}}\mathrm{J}(\theta) \tag{11}$$

对于公式 11，选择合适的α，使$\mathrm{J}(\theta_{k+1})<\mathrm{J}(\theta_k)$。

因此：

$$\mathrm{W}=\mathrm{W}-\alpha\frac{\partial}{\partial \mathrm{W}}\frac{1}{2}\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right)-y^{(i)})^2$$

$$\mathrm{W}=\mathrm{W}-\alpha\frac{\partial}{\partial \mathrm{W}}\frac{1}{2}\sum_{i=1}^{m}\left(X_{(i)}^{T}W-y^{(i)}\right)^{2}$$

$$\mathrm{W}=\mathrm{W}-\alpha\frac{\partial}{\partial \mathrm{W}}\sum_{i=1}^{m}\left(X_{(i)}^{T}W-y^{(i)}\right)X_{(i)}^{T} \tag{12}$$

通过此时得出的 W 计算代价函数，即 loss 的值。

$$\mathrm{J}(\theta)=\frac{1}{2}(W^{T}X^{T}XW-W^{T}X^{T}Y-Y^{T}XW+Y^{T}Y)+\frac{\lambda}{2}W^{T}W$$

若$\mathrm{J}(\theta_{k+1})<\mathrm{J}(\theta_{k})$，则继续迭代，当$\mathrm{abs}\left(\mathrm{J}(\theta_{k+1})-\mathrm{J}(\theta_{k})\right)<\delta$时，停止迭代。

若出现$\mathrm{J}(\theta_{k+1})>\mathrm{J}(\theta_{k})$的情况，则可能学习率选择过大，此时令$\alpha=\alpha/2$. 可使其正常迭代。

5. 优化方法求解最优解（共轭梯度）

由公式 10，我们可以得到需要解决的任务如下：

$$(X^{\mathrm{T}}X+\lambda I)W-X^{\mathrm{T}}Y=0 \tag{13}$$

由共轭梯度下降法的方法表述：

即对线性系统求解：

$$\mathrm{Ax}=\mathrm{b} \tag{14}$$

由公式 13、公式 14 有：

$$\mathrm{A}=X^{\mathrm{T}}X+\lambda I$$
$$\mathrm{b}=X^{\mathrm{T}}Y$$

共轭梯度下降法的算法描述：

$$r_0 := b-Ax_0$$
$$p_0 := r_0$$
$$\mathrm{k} := 0$$
$$\text{repeat}$$
$$\alpha_k := \frac{r_k^T r_k}{P_k^T A P_K}$$
$$x_{k+1} := x_k+\alpha_k P_K$$
$$r_{k+1} := r_k-\alpha_k A P_K$$
$$\text{if } r_{k+1}\ is\ sufficiently\ small, then\ exit\ loop$$
$$\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$
$$P_{K+1} := r_{k+1}+\beta_k P_K$$
$$\mathrm{k} := \mathrm{k}+1$$
$$\text{end repeat}$$

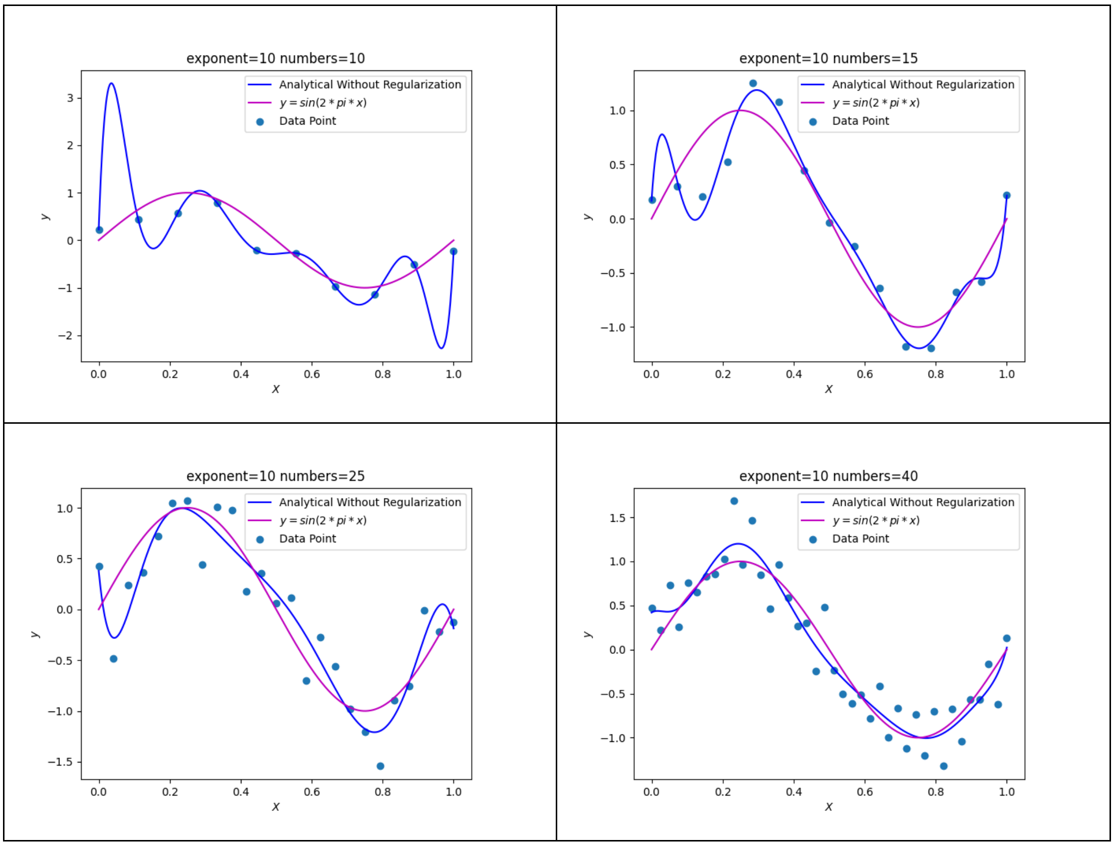
结果为x_{k+1}，即$\mathrm{W}=x_{k+1}$。

四、实验结果与分析

注：exponent 指多项式的系数的矩阵的规模为 exponent*1，即多项式的最高次为 exponent-1.

(一) 不带正则项的高阶多项式拟合

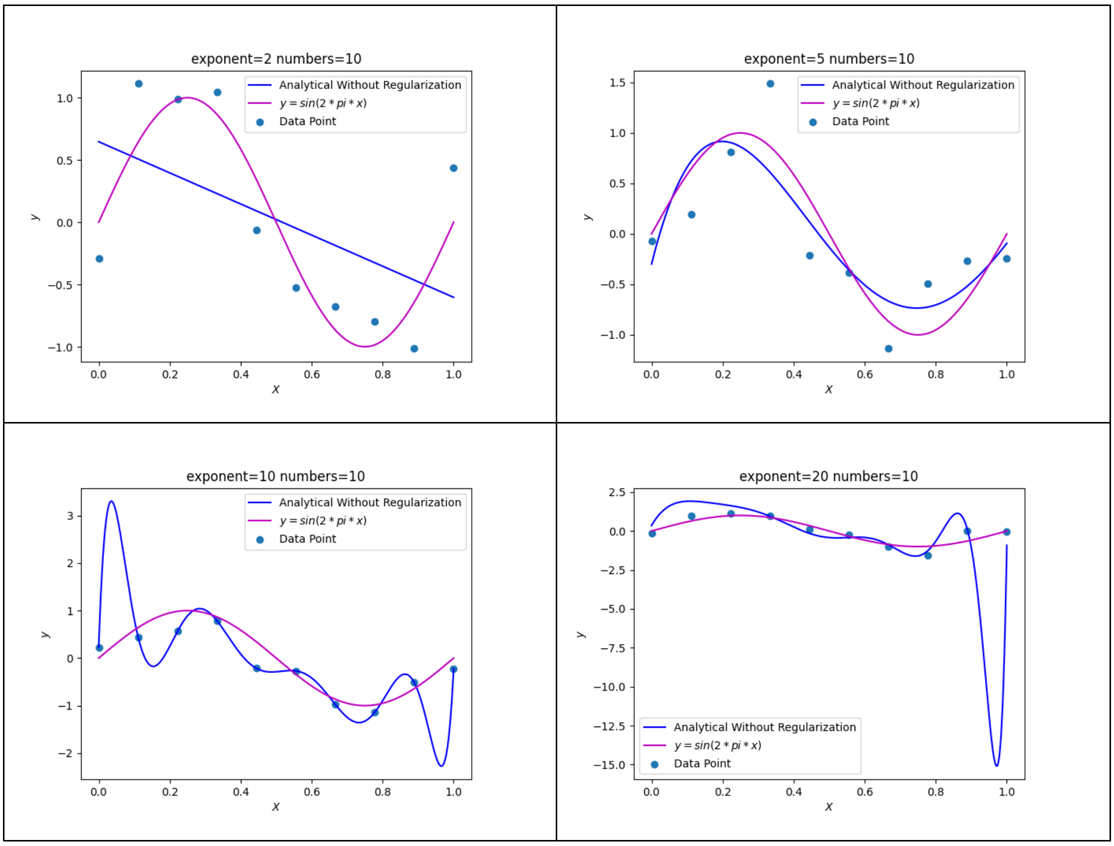
1. 固定多项式阶数，使用不同训练样本数量，样本的噪声的方差为 0.3.



实验分别取训练样本点数量为 10、15、25、40.

通过实验，我们可以很明显的看到，在训练样本数量过少的时候，可能存在拟合的多项式经过所有的点，但很容易发生过拟合。如图 1，其经过完美的经过所有的点，但发生了剧烈的震荡，不能够很好的拟合$y = \sin(2\pi x)$. 因此，图 1 拟合的函数出现了过拟合的现象，不适用于预测其他样本点。如图 2、图 3、图 4，随着训练样本点的逐渐增多，过拟合的现象得到改善，说明样本点的数量对于过拟合问题有较大影响。图 4 已经可以较好的拟合$y = \sin(2\pi x)$函数。

2. 固定训练样本数量，使用不同的多项式阶数拟合，样本的噪声的方差为 0.3.



实验取 exponent 分别为 2、5、10、20，所对应的多项式的阶数分别为 1、4、9、19.

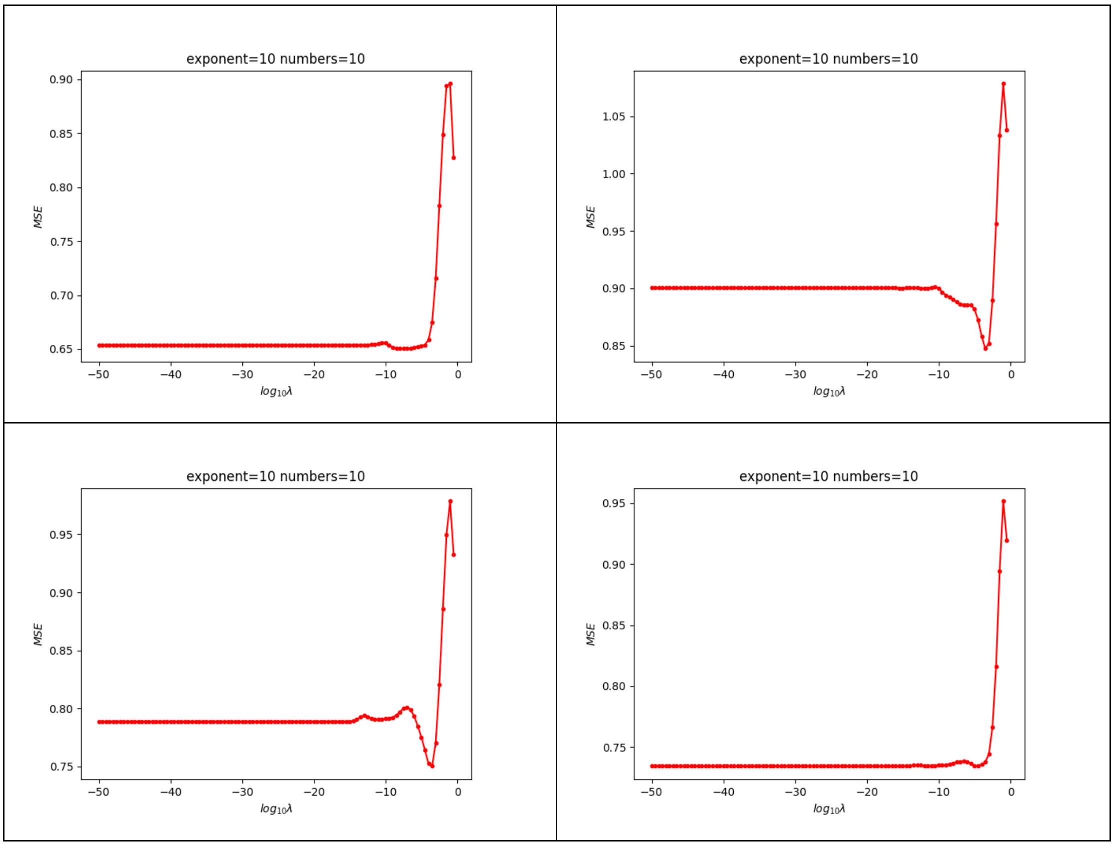
随着多项式的阶数的提高，多项式的拟合能力逐渐增强，如图 1，当 exponent=2，阶数为 1 的时候，拟合效果很差。但当 exponent=5，即阶数为 4 的时候，如图 2，函数已经较好的拟合了$y = \sin(2\pi x)$，此时继续增长多项式的阶数，可能会造成过拟合，即多项式函数可能接近每一个点，但其预测效果很差，容易发生剧烈抖动。如图 3 函数发生了剧烈的抖动，不能很好的预测$y = \sin(2\pi x)$.当 exponent=20，阶数为 19 时，函数发生了非常剧烈的抖动，在第九个和第十个样本点之间，出现了远低于$y = \sin(2\pi x)$预测的数，因此拟合效果很差。因此多项式函数的阶数会很大程度上影响函数的拟合效果。

(二) 带正则项的高阶多项式拟合

1. 首先选取合适的 lambda 参数。

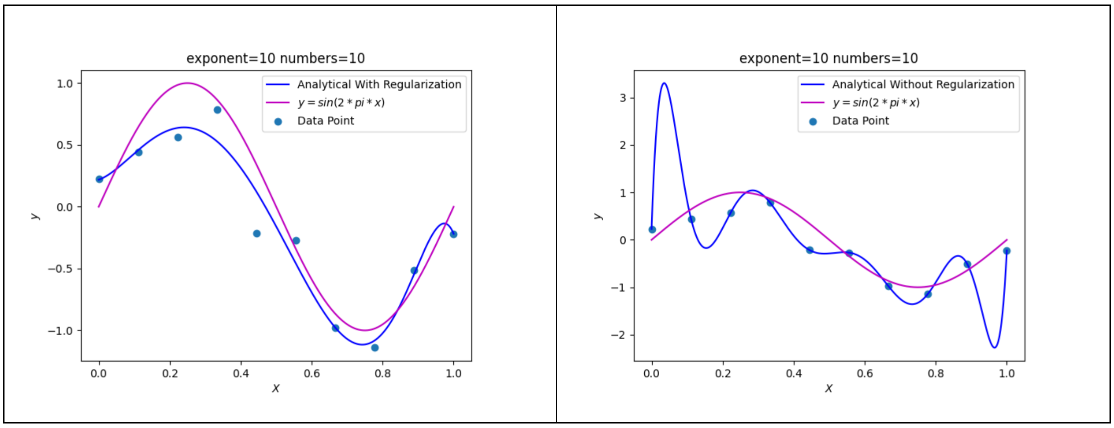
计算不同 lambda 对应的 MSE 的值，若 lambda 过小则可能导致函数过拟合，若 lambda 过大，则可能导致多项式函数出现欠拟合现象，因此选取合适的 lambda 非常重要。

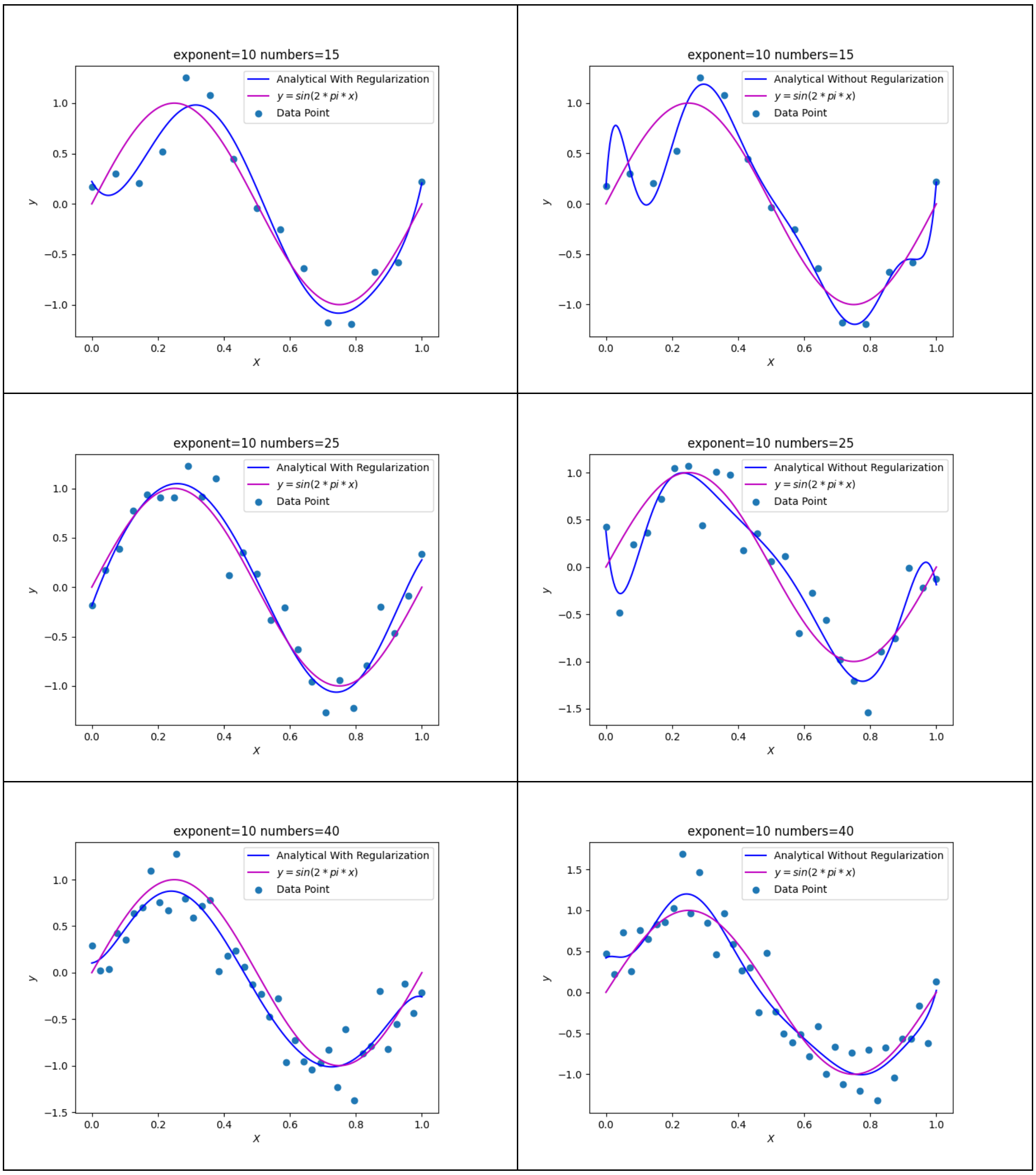
通过计算 MSE 选取合适的 lambda 的值。固定样本的阶数为 10 阶，样本的数量为 10.



通过上述实验过程，可以观察到，MSE 在处于$(10^{-10}, 10^{-5})$会出现最小值，但由于在最低点之后，MSE 增长速度过快，因此通过大量实验，选取较为合适的 lambda 的值为10^{-7}.

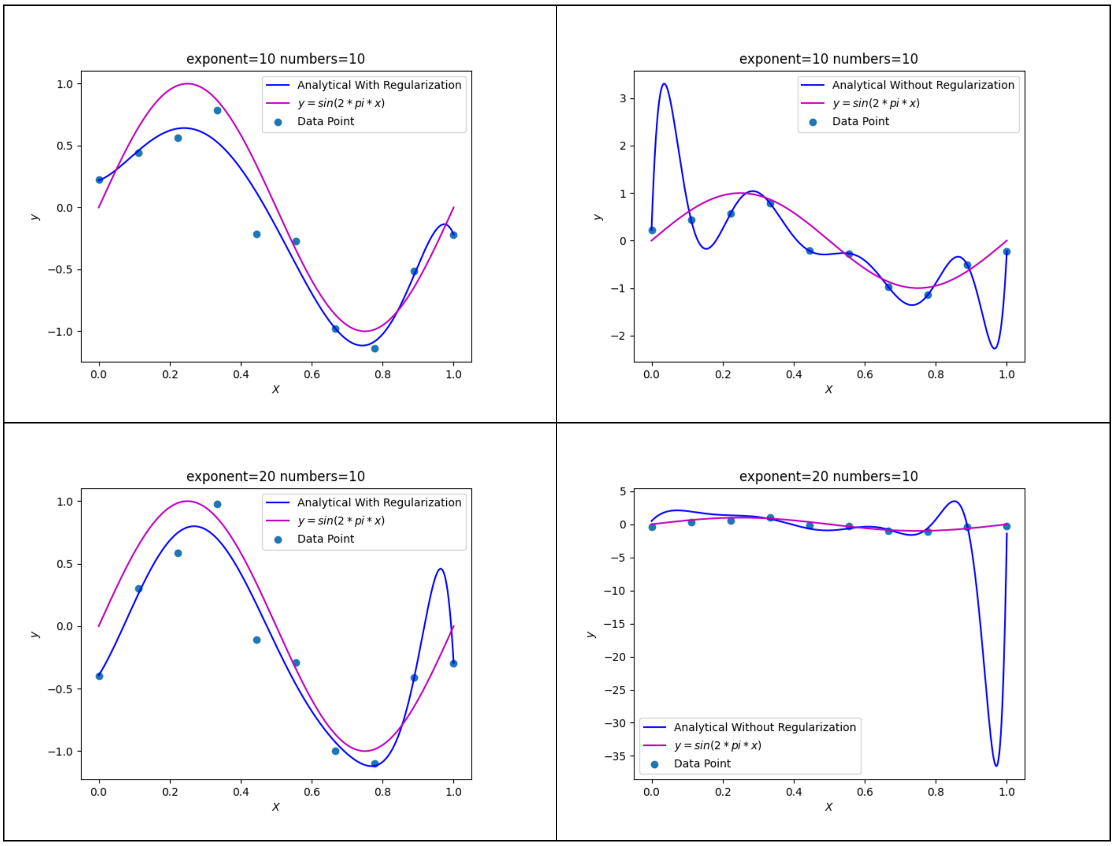
2. 固定多项式阶数，固定惩罚项系数为10^{-7}使用不同训练样本数量，样本的噪声的方差为 0.3。展开有正则项和无正则项的对比实验





上图左侧为带正则项的高阶多项式，右侧为不带正则项的高阶多项式，可以看到，在选择合适的超参数λ后，函数过拟合的情况大大缓和。实验验证正则项可以大大缓和过拟合的情况。

3. 固定训练样本数量，使用不同的多项式阶数拟合，样本的噪声的方差为 0.3.

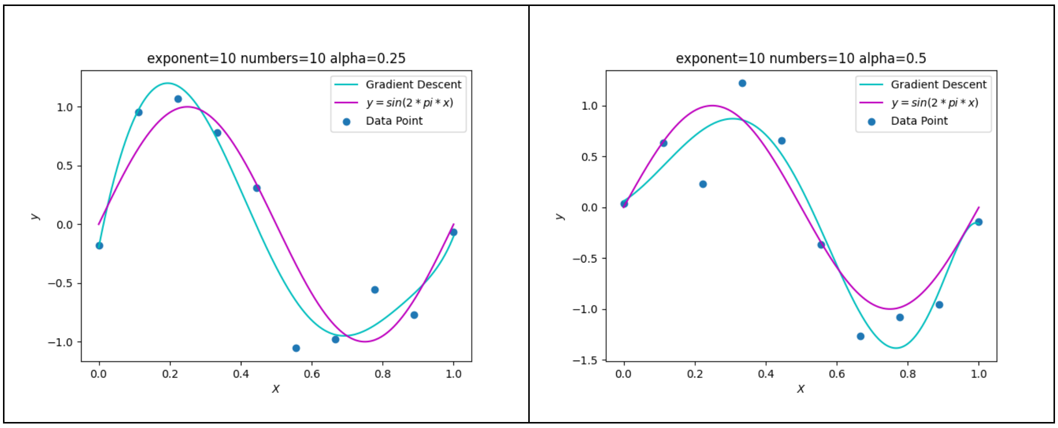


实验取 exponent 分别为 10、20，所对应的多项式的阶数分别为 9、19. 右侧不带正则项函数拟合曲线明显过拟合，左侧带正则项函数过拟合程度大大减缓，实验验证正则项可以大大缓和过拟合的情况。

(三) 优化方法梯度下降法

此处设置的停止的精度要求为10^{-8}.

1. 固定训练样本数量、多项式阶数，使用不同的学习率。



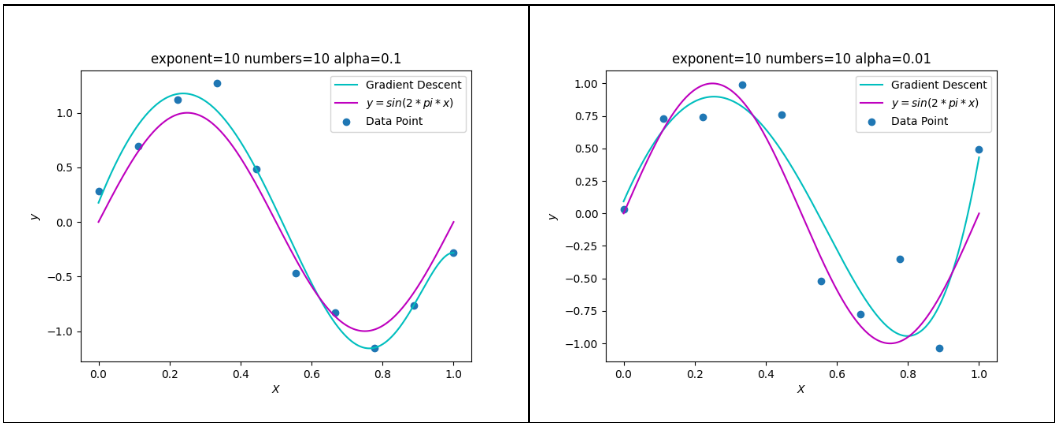
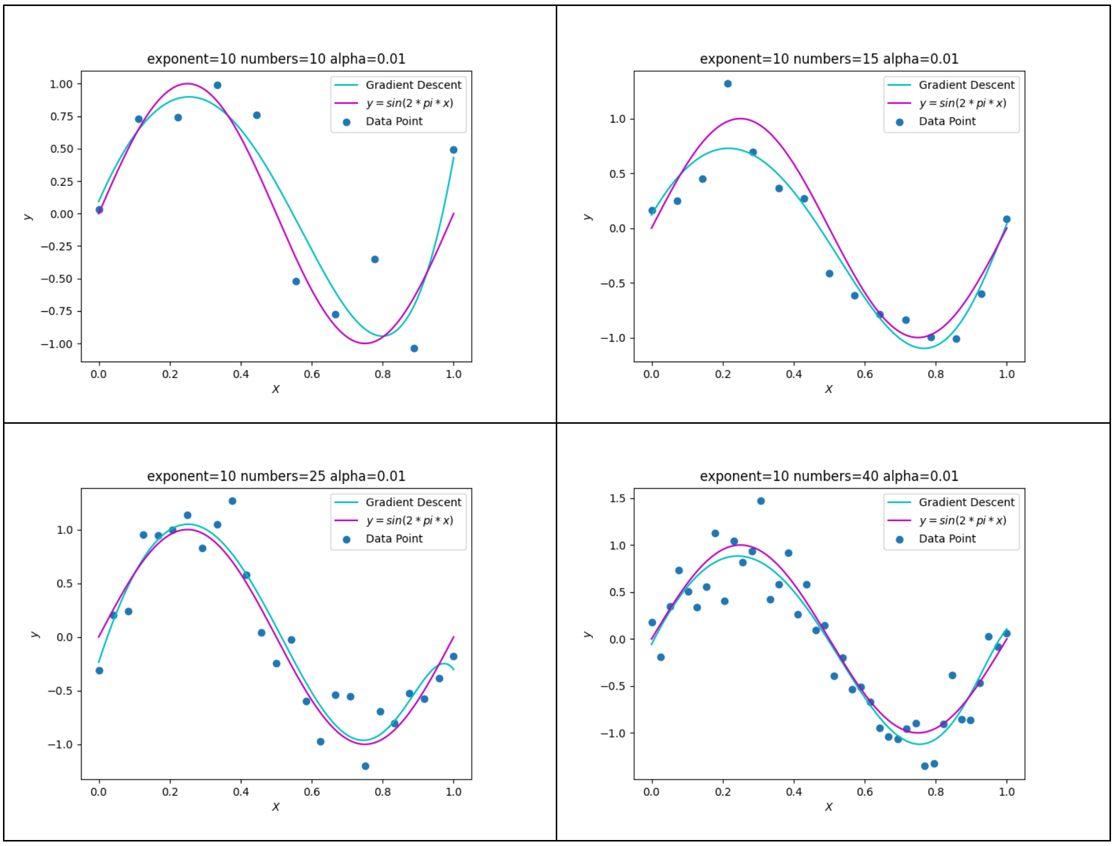


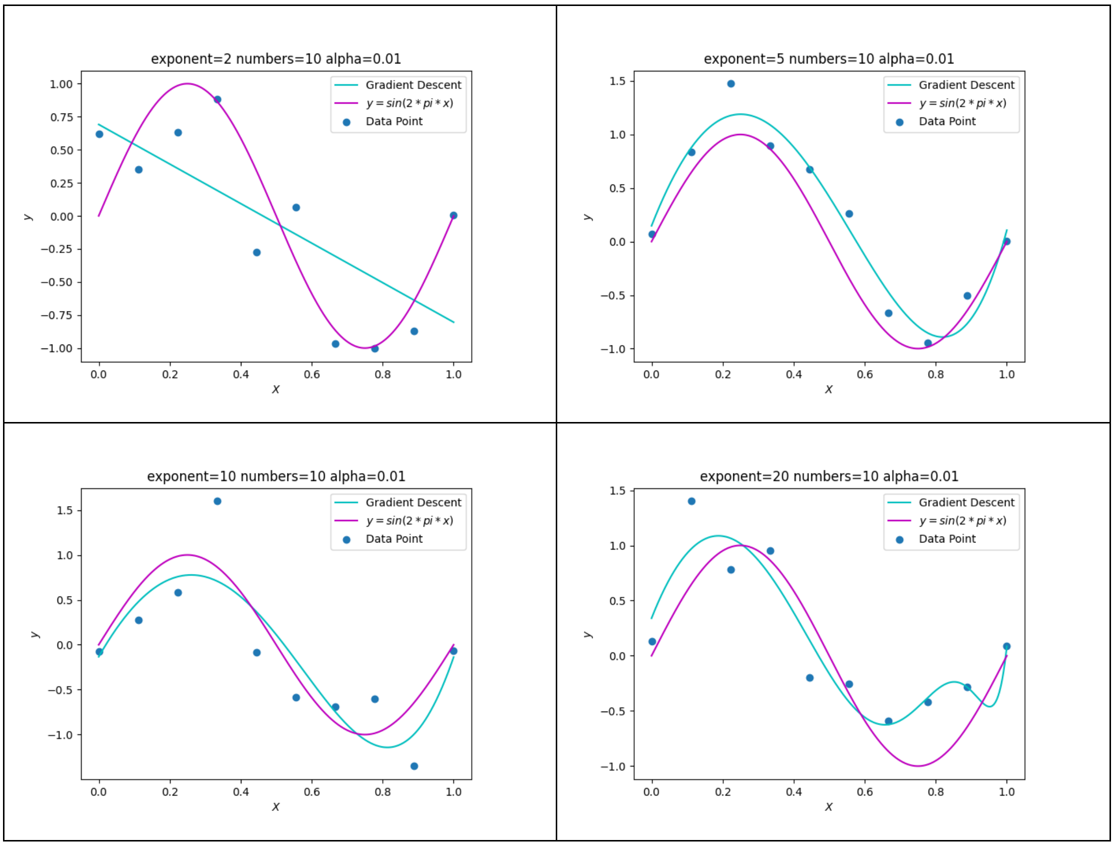
图 1 设置的学习率为 1，但由于当学习率为 1 的时候，梯度下降无法收敛，因此在迭代的过程中，自动将学习率下调，当下调到 0.25 的时候，梯度下降可以收敛。图 2 为使用 0.5 的学习率迭代，图 3 使用 0.1 的学习率迭代，图 4 使用 0.01 的学习率迭代。由于精度的要求较高，并且函数有正则项，因此拟合效果较好。

2. 固定学习率为 0.01、多项式阶数为 9，调整训练样本个数。



分别设置训练样本数量为 10、15、25、40. 在不同训练数量的情况下，都有比较好的结果。

3. 固定学习率为 0.01、训练样本个数为 10，调整多项式阶数。

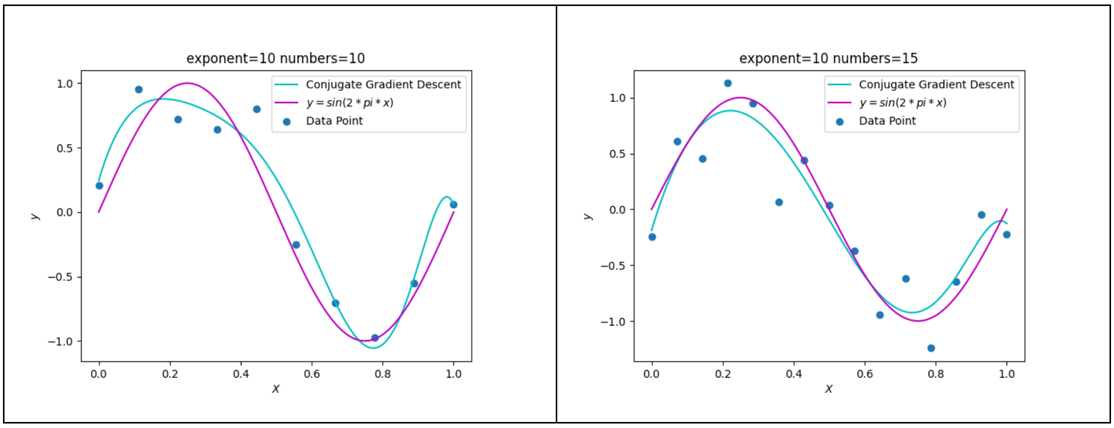


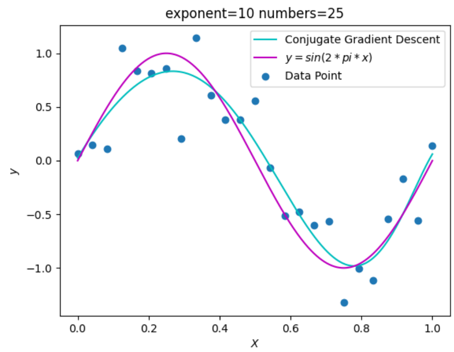
分别将 exponent 设为 2、5、10、20，如图 2，在阶数为 4 的时候，拟合曲线逐渐符合$y = \sin(2\pi x)$。如图 4，在多项式阶数过高的时候发生了过拟合。

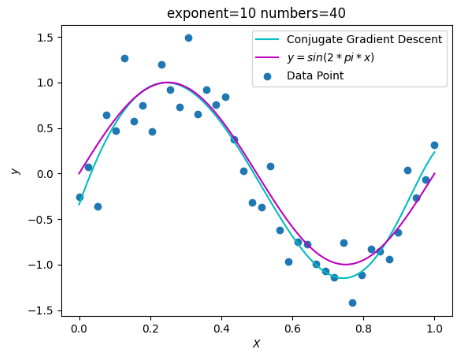
(四)优化方法共轭梯度法

此处设置的停止的精度要求为10^{-8}.

1. 多项式阶数为 9，调整训练样本个数。

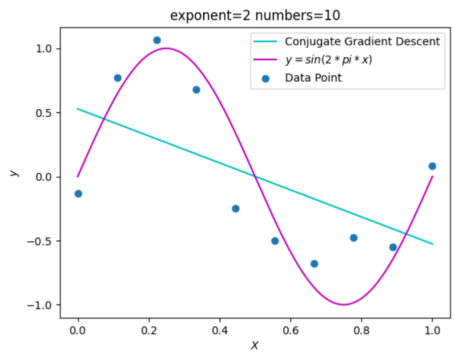


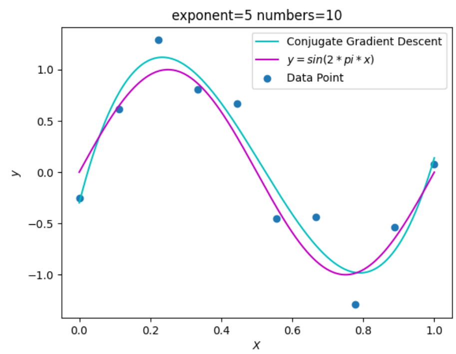


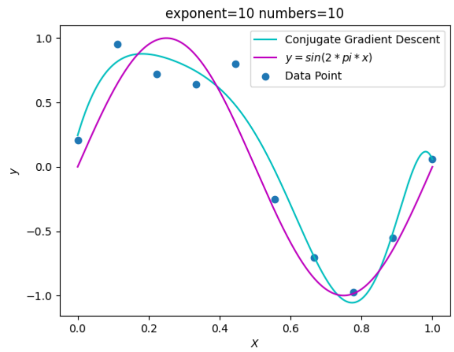


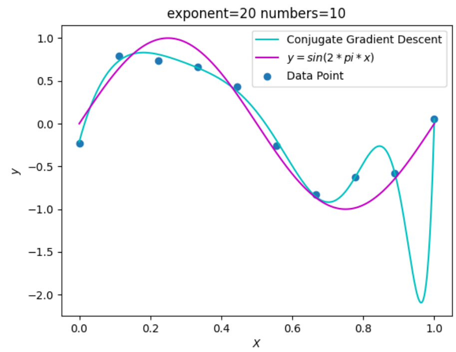
分别设置训练样本数量为 10、15、25、40. 随着训练样本数量的增多，拟合曲线逐渐贴近$y = \sin(2\pi x)$。拟合效果较好。

2. 训练样本个数为 10，调整多项式阶数。









分别将 exponent 设为 2、5、10、20，如图 2，在阶数为 4 的时候，拟合曲线逐渐符合$y = \sin(2\pi x)$。如图 4，在多项式阶数过高的时候发生了过拟合。

(五)梯度下降法和共轭梯度法比较

迭代次数比较：

1. 当训练样本个数固定时

	n=10 exponent=2	n=10 exponent=5	n=10 exponent=10	n=10 exponent=20
梯度下降法	7095	785681	1289810	2855686

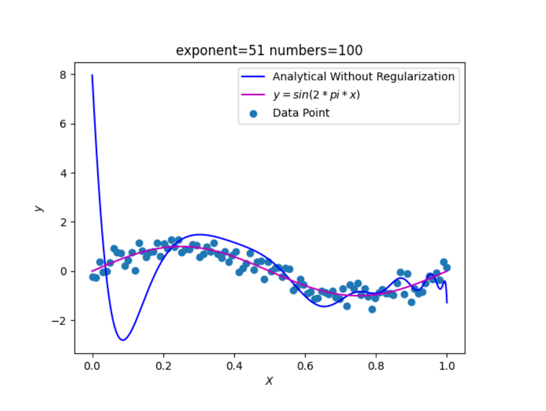
共轭梯度法	1	4	10	11

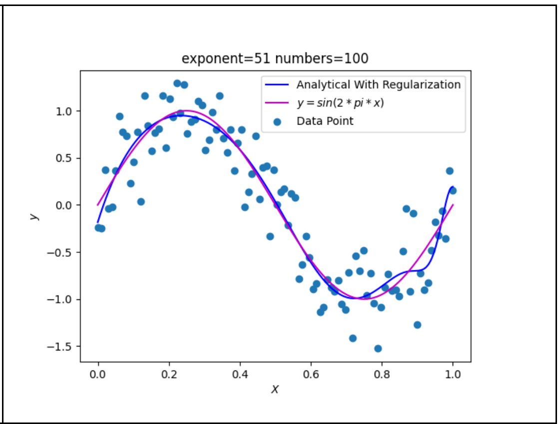
2. 当多项式阶数固定时

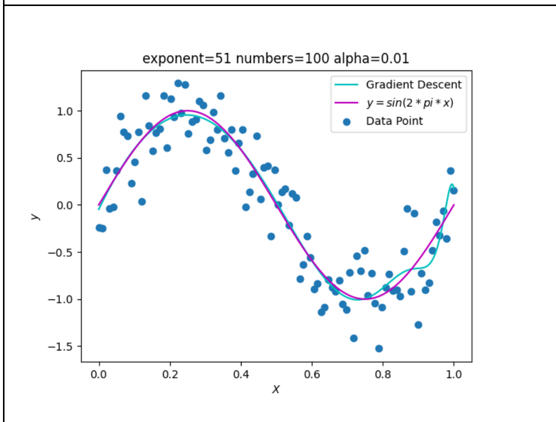
	n=10 exponent=10	n=15 exponent=10	n=25 exponent=10	n=40 exponent=10
梯度下降法	1289810	1580094	1668467	2042393
共轭梯度法	10	14	14	18

从表中可以看出，梯度下降法的迭代次数远远大于共轭梯度法，由于梯度下降法，在靠近极值点的时候收敛速度降低，可能会造成速度较慢，实际实验中，共轭梯度法的速度也远远高于梯度下降法。

(六) 训练样本为 100，多项式的阶数为 50







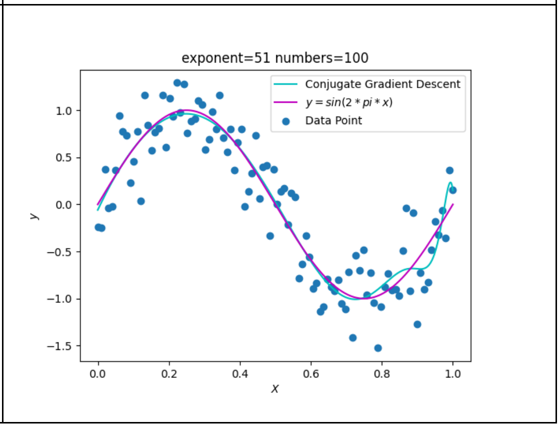


图 1 为不带正则项的高阶多项式拟合结果，图 2 为带正则项的高阶多项式拟合结果，图 3 为梯度下降法的拟合曲线，图 4 为共轭梯度法的拟合曲线，可以看到图 1 曲线不拟合的情况，此时通过判断，当前矩阵为病态矩阵，即没有实际的逆，但 inv 函数给出了伪逆，因此实际拟合曲线结果较差，其他三种情况拟合效果都比较好。

```
if np.linalg.cond(np.matmul(X_train.T, X_train)) > np.finfo(np.matmul(X_train.T, X_train).dtype).eps:
    print("病态矩阵")
```

五、结论

1. 拟合函数阶数过高容易导致过拟合；
2. 训练样本数量太少容易导致过拟合；
3. 解析解易过拟合时，可以添加正则项，减缓过拟合的情况。
4. 对于 n 过大的情况，求逆运算会很慢，可以使用梯度下降法或共轭梯度法，梯度下降法的迭代次数较多，远远慢于共轭梯度法，但二者都有较好的拟合能力。
5. 学习率的选择不宜过大，易发生不收敛的情况。

六、参考文献

[1] Pattern Recognition and Machine Learning
[2] 机器学习 周志华著 北京：清华大学出版社，2016 年 1 月.
[3] https://zhuanlan.zhihu.com/p/36564434

七、附录：源代码（带注释）

主函数 main.py

```python
from generateData import *
from gradientDescent import *
from conjugateGradientDescent import *
from leastSquareMethod import *

if __name__ == '__main__':
    # 训练样本个数
    numbers = 25
    # 噪声 Sigma
```

```python
    Sigma = 0.3
    # 多项式的阶数
    exponent = 10
    # 惩罚系数 Lambda
    Lambda = 1e-7
    # 梯度下降学习率
    alpha = 0.01
    # 梯度下降迭代次数
    GDiterNum = 800000000
    # 共轭梯度下降迭代次数
    CGiterNum = 100
    # 迭代精度
    precision = 1e-8
    # 生成数据
    X, X_train, y, y_noise = addNoise(numbers, exponent, Sigma)

    # fittingNoRegular(X, X_train, y_noise, exponent)
    # fittingRegular(X, X_train, y_noise, exponent, Lambda)
    # Drawlambda(X, X_train, y_noise, exponent)
    ConjugateGradientDescent(X, X_train, y_noise, exponent, CGiterNum, Lambda, precision)
    GradientDescent(X, X_train, y_noise, exponent, alpha, GDiterNum, Lambda, precision)
```

生成数据 generateData.py

```python
import numpy as np
def generateNoise(Sigma, Size):
    Noise = np.random.normal(0, scale=Sigma, size=Size)
    return Noise

def addNoise(numbers,exponent,Sigma):
    X = np.linspace(start=0, stop=1, num=numbers)
    GuassNoise = np.random.normal(0, scale=Sigma, size=X.shape)
    y = np.sin(2 * np.pi * X)
    y_noise = y + GuassNoise
    # plt.scatter(X, y_noise)
    # plt.show()
    row = np.ones(numbers, dtype=np.float64) * X
    X_train = row ** 0
    # 计算不同阶对应 X 值
    for i in range(1, exponent):
        row = np.ones(numbers, dtype=np.float64) * X
        row = row ** i
```

```python
        X_train = np.dstack((X_train, row))
    X_train = np.reshape(X_train, (numbers, exponent))
    return X,X_train,y, y_noise


def generatePlotdata(numbers,exponent):
    X = np.linspace(start=0, stop=1, num=numbers)
    row = np.ones(numbers, dtype=np.float64) * X
    X_train = row ** 0
    # 计算不同阶对应 X 值
    for i in range(1, exponent):
        row = np.ones(numbers, dtype=np.float64) * X
        row = row ** i
        X_train = np.dstack((X_train, row))
    X_train = np.reshape(X_train, (numbers, exponent))
    return X,X_train
```

绘制sin(2πx)曲线 drawSin2pix.py

```python
import numpy as np
import matplotlib.pyplot as plt
def drawSin2pix(numbers):
    X = np.linspace(start=0, stop=1, num=numbers)
    y = np.sin(2 * np.pi * X)
    plt.plot(X,y,"m",label = "$y=sin(2*pi*x)$")
```

使用解析解求拟合函数 leastSquareMethod.py

```python
import numpy as np
import matplotlib.pyplot as plt
from generateData import generatePlotdata
from drawSin2pix import drawSin2pix
from calculateLambda import *
def fittingNoRegular(X,X_train,y_noise,exponent):
    W = np.matmul(np.matmul(np.linalg.inv(np.matmul(X_train.T, X_train)), X_train.T), y_noise)
    W = np.reshape(W, (exponent, 1))
    Y_predict = np.matmul(X_train, W)
    # plt.plot(X, Y_predict,format('b.-'))


    title = 'exponent={} numbers={}'.format(exponent,X.shape[0])
    plt.title(title)
    plt.xlabel('$X$', fontsize=10)
```

```
    plt.ylabel('$y$', fontsize=10)
    plt.scatter(X, y_noise, marker='o', label='Data Point')
    # numbers 为绘图的点数
    X,X_train = generatePlotdata(numbers=1000,exponent=exponent)
    Y_predict = np.matmul(X_train, W)
    plt.plot(X,Y_predict,format('b'),label='Analytical Without Regularization')
    drawSin2pix(1000)
    plt.legend()
    plt.savefig("./" + "picture/LS/NoRegular/" + title)
    plt.show()


def fittingRegular(X,X_train,y_noise,exponent,Lambda):
    W = np.matmul(np.matmul(np.linalg.inv((np.matmul(X_train.T, X_train) + np.eye(exponent) * Lambda)), X_train.T),y_noise)
    # W = np.reshape(W, (exponent, 1))
    Y_predict = np.matmul(X_train, W)
    # plt.plot(X, Y_predict)

    title = 'exponent={} numbers={}'.format(exponent,X.shape[0])
    plt.title(title)
    plt.xlabel('$X$', fontsize=10)
    plt.ylabel('$y$', fontsize=10)
    plt.scatter(X, y_noise, marker='o', label='Data Point')
    # numbers 为绘图的点数
    X, X_train = generatePlotdata(numbers=1000, exponent=exponent)
    Y_predict = np.matmul(X_train, W)
    plt.plot(X, Y_predict, format('b'), label='Analytical With Regularization')
    drawSin2pix(1000)
    plt.legend()
    plt.savefig("./" + "picture/LS/Regular/" + title)
    plt.show()
```

计算代价函数的值 computeCost.py

```
import numpy as np
# 计算代价函数
def ComputeCost(X, y, theta,Lambda):
    hypthesis = np.dot(X, np.transpose(theta))
    # 先转置再做矩阵乘法
    cost = np.dot(np.transpose(hypthesis - y), (hypthesis - y))
    cost = cost / 2 + Lambda*np.dot(theta,np.transpose(theta))/2
```

```
    return cost
```

求梯度下降法的系数 gradientDescent.py

```
import numpy as np
from computeCost import ComputeCost
import matplotlib.pyplot as plt
from generateData import generatePlotdata
from drawSin2pix import drawSin2pix
def GradientDescent(X,X_train,y,exponent,alpha,iterNum,Lambda,precision):
    # 用于存储代价值
    costStore = np.zeros(iterNum)
    # 定义 X 数据的大小
    Xsize = X_train.shape[0]
    # 设置 theta 初始值
    theta = np.zeros(exponent)
    costStore[0] = ComputeCost(X_train, y, theta, Lambda)
    predict = np.matmul(X_train, theta.T)
    theta = theta - alpha * np.dot(X_train.T, (predict - y).T) / Xsize
    print("当前迭代的 alpha={}".format(alpha))
    for num in range(1,iterNum):
        # 计算当前代价值并保存
        costStore[num] = ComputeCost(X_train,y,theta,Lambda)
        predict = np.matmul(X_train,theta.T)
        theta = theta - alpha *np.dot(X_train.T,(predict-y).T)/Xsize
        if abs(costStore[num]-costStore[num-1]) <= precision :
            print("迭代次数：{}".format(num))
            print("当前迭代的 alpha={}".format(alpha))
            break
        if costStore[num]-costStore[num-1] >0:
            alpha = alpha/2
            print("当前迭代的 alpha={}".format(alpha))
    Y_predict = np.matmul(X_train, theta)
    # plt.plot(X, Y_predict,format('y'))

    # # numbers 为绘图的点数
    # X,X_train = generatePlotdata(numbers=100,exponent=exponent)
    # Y_predict = np.matmul(X_train, theta)
    # plt.plot(X, Y_predict, format('y'),label = 'Gradient Descent')

    title = 'exponent={} numbers={} alpha={}'.format(exponent,X.shape[0],alpha)
    plt.title(title)
    plt.xlabel('$X$', fontsize=10)
```

```python
    plt.ylabel('$y$', fontsize=10)
    plt.scatter(X, y, marker='o', label='Data Point')
    # numbers 为绘图的点数
    X, X_train = generatePlotdata(numbers=1000, exponent=exponent)
    Y_predict = np.matmul(X_train, theta)
    plt.plot(X, Y_predict,format('c'),label='Gradient Descent')
    drawSin2pix(1000)
    plt.legend()
    # plt.savefig("./" + "picture/GradientDescent/" + title+".png")
    plt.show()
```

计算合适的 lambda 的值 caculateLambda.py

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

from computeCost import ComputeCost
def CalculateLambda(X, y, theta,Lambda):
    hypthesis = np.matmul(X, theta)
    y = np.reshape(y,(X.shape[0],-1))
    # 先转置再做矩阵乘法
    cost = np.dot(np.transpose(hypthesis - y), (hypthesis - y))
    cost = (cost / 2 + Lambda*np.dot(np.transpose(theta),theta)/2)[0][0]
    # cost = (cost / 2)[0][0]
    MSE = math.sqrt(cost*2/X.shape[0])
    return MSE


def Drawlambda(X,X_train, y,exponent):

    # 迭代次数
    RangeLeft = -100
    RangeRight = 0
    ErmsStore = np.zeros(RangeRight-RangeLeft)
    LambdaStore = np.zeros(RangeRight-RangeLeft)
    for num in range(RangeLeft,RangeRight):

        LambdaStore[num-RangeLeft] = num/2
        Lambda = 10**(num/2)
        W = np.matmul(np.matmul(np.linalg.inv((np.matmul(X_train.T, X_train)
 + np.eye(exponent) * Lambda)), X_train.T),y)
        MSE = CalculateLambda(X_train,y,W,Lambda)

```

```python
        # Cost = ComputeCost(X_train,y,theta,Lambda)

        ErmsStore[num-RangeLeft] = MSE
    title = 'exponent={} numbers={}'.format(exponent, X.shape[0])
    plt.title(title)
    plt.xlabel('$log_{10}\lambda$', fontsize=10)
    plt.ylabel('$MSE$', fontsize=10)
    plt.plot(LambdaStore, ErmsStore,'r.-')
    print(LambdaStore)
    print(LambdaStore.shape)
    print(ErmsStore)
    # plt.legend()
    plt.savefig("./" + "picture/MSE/LS/" + title)
    plt.show()
```

共轭梯度下降法 conjugateGradientDescent.py

```python
import numpy as np
from generateData import generatePlotdata
import matplotlib.pyplot as plt
from drawSin2pix import drawSin2pix
def ConjugateGradientDescent(X, X_train, y, exponent, CGiterNum, Lambda, precision):
    A = np.dot(np.transpose(X_train), X_train) + np.eye(exponent) * Lambda
    b = np.dot(np.transpose(X_train), y)
    b = np.reshape(b,(exponent,-1))
    Wk = np.ones(exponent)*0
    Wk = np.reshape(Wk,(exponent,-1))

    test = np.dot(A, Wk)

    rk = b - np.dot(A, Wk)
    rk = np.reshape(rk,(exponent,-1))
    Pk = rk
    precisionMatrix = np.ones(exponent)*precision
    for num in range(0, CGiterNum):

        alpha = ((np.dot(np.transpose(rk),rk))/(np.dot((np.dot(np.transpose(Pk),A)),Pk)))[0][0]
        Wk = Wk + alpha*Pk
        rkadd1 = rk - alpha*np.dot(A,Pk)
        if np.all(rkadd1 <= precisionMatrix):
            print("第{}次迭代收敛".format(num))
            break
```

```
        beta = (np.dot(np.transpose(rkadd1),rkadd1)/np.dot(np.transpose(rk),
rk))[0][0]
        Pk = rkadd1+beta*Pk
        rk = rkadd1

    title = 'exponent={} numbers={}'.format(exponent,X.shape[0])
    plt.title(title)
    plt.xlabel('$X$', fontsize=10)
    plt.ylabel('$y$', fontsize=10)
    plt.scatter(X, y, marker='o', label='Data Point')
    # numbers 为绘图的点数
    X, X_train = generatePlotdata(numbers=1000, exponent=exponent)
    Y_predict = np.matmul(X_train, Wk)
    plt.plot(X, Y_predict, format('c'), label='Conjugate Gradient Descent')

    drawSin2pix(1000)
    plt.legend()
    # plt.savefig("./" + "picture/ConjugateGradientDescent/" + title)
    plt.show()
```