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： 限选

实验题目： 多项式拟合正弦函数

学号：1160300601

姓名： 马旭

目录

[1、实验目的 3](#_Toc525759635)

[2、实验要求及实验环境 3](#_Toc525759636)

[2.1.实验要求： 3](#_Toc525759637)

[2.2.实验环境： 3](#_Toc525759638)

[3、设计思想（本程序中的用到的主要算法及数据结构） 3](#_Toc525759639)

[3.1．主要用到的数据结构 3](#_Toc525759640)

[3.2 算法的原理与实现 4](#_Toc525759641)

[3.2.1.使用解析解求解最优参数 4](#_Toc525759642)

[3.2.2.使用梯度下降法求解最优参数 5](#_Toc525759643)

[3.2.3.使用共轭梯度法求解最优参数 5](#_Toc525759644)

[4、实验结果与分析 7](#_Toc525759645)

[4.1.使用不同算法求解不带正则项的向量w 7](#_Toc525759646)

[4.2.使用不同算法求解带正则项的向量w 9](#_Toc525759647)

[4.4.不同数量级样本对过拟合的影响 13](#_Toc525759648)

[4.5.不同多项式阶数对过拟合的影响 14](#_Toc525759649)

[4.6.不同超参数对过拟合的影响 15](#_Toc525759650)

[5、结论 16](#_Toc525759651)

[6、参考文献 16](#_Toc525759652)

[7、附录：源代码（带注释） 16](#_Toc525759653)

[7.1.生成样本数据部分 16](#_Toc525759654)

[7.2一些辅助的函数（用来生成矩阵X,向量t和计算Erms） 17](#_Toc525759655)

[7.3计算W的算法（解析解、梯度下降法、共轭梯度法） 18](#_Toc525759656)

[7.4一个可以修改各种参数，并且打印出对应拟合图像及Erms值的模块 20](#_Toc525759657)

[7.5.运行后可以自动分析Erms（训练集和测试集）与M值关系并展现到图线上的模块 21](#_Toc525759658)

[7.6. 运行后可以自动分析Erms（训练集和测试集）与超参数值关系并展现到图线上的模块 23](#_Toc525759659)

[7.7. 运行后可以自动分析Erms（训练集和测试集）与样本数个数关系并展现到图线上的模块 25](#_Toc525759660)

1、实验目的

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

2、实验要求及实验环境

## 2.1.实验要求：

1. 生成数据，加入噪声；

2. 用高阶多项式函数拟合曲线；

3. 用解析解求解两种loss的最优解（无正则项和有正则项）

4. 优化方法求解最优解（梯度下降，共轭梯度）；

5. 用你得到的实验数据，解释过拟合。

6. 用不同数据量，不同超参数，不同的多项式阶数，比较实验效果。

7. 语言不限，可以用matlab，python。求解解析解时可以利用现成的矩阵求逆。梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不许用现成的平台，例如pytorch，tensorflow的自动微分工具。

## 2.2.实验环境：

1.X86-64 windows

2.使用PyCharm开发工具

3.使用python编程语言

3、设计思想（本程序中的用到的主要算法及数据结构）

## 3.1．主要用到的数据结构

Python中自带的矩阵

## 3.2 算法的原理与实现

### 3.2.1.使用解析解求解最优参数

1.使用解析解求不带正则项的最小E(w)

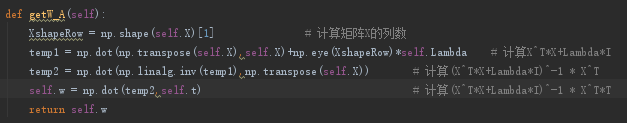
(1).原理

不带正则项的公式为：E(w) =

表达成矩阵形式为：E(w) = ，想要求可对上式求导，得到导数为 = - ,如果有解，就可以求得E(w)的最小值，即当w = 时可求得E(w)的最小值。

(2).实现

该实现为带的通用解法，当其中的设置为0时即时不带正则项的最小值解。



2.使用带解析解求带正则项的最小E(w)

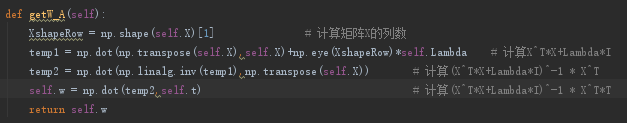
(1).原理

带正则项的公式为：E(w) =

表达成矩阵形式为：E(w) = , 想要求可对上式求导，得到导数为 = ( - , 如果有解，就可以求得E(w)的最小值，即当w = 时可求得E(w)的最小值。

(2).实现

直接通过公式计算



### 3.2.2.使用梯度下降法求解最优参数

1.原理

在微积分里面，对于多元函数可以把求得的各个参数的偏导数以向量的形式写出来，就是梯度，记为▽f(x,y)，梯度向量从几何意义上讲，就是函数变化增加最快的地方，沿着梯度向量的方向就是函数增加最快的地方，就更加容易找到函数的最大值。同时沿着梯度向量相反的方向，也就是-▽f(x,y)的方向，梯度减少最快，也就更容易找到函数的最小值。于是就可以使用梯度下降法一步步来迭代求得最小值。

2.实现

其中定义要解的变量为向量,且E() = -

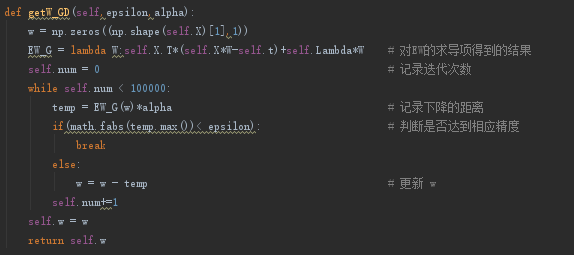
(1).初始化步长α和精度ε

(2).以步长乘以损失的函数的梯度，得到当前位置下降的距离，即E()

(3).确定向量中的每个值，梯度下降的距离都小于ε，如果小于ε则算法终止,否则进入下一步。

(4).更新向量，其更新表达式为：E()，更新完毕后继续转入步骤1

Python代码实现如下：



### 3.2.3.使用共轭梯度法求解最优参数

1.原理

上面讲到加入正则项后的损失函数为E(w) = ，当求导后可以得到 = ( - ，令，可以知道(，这样就可以使用共轭方法求E(w)的最小值。

推导：令目标函数标准形式为f = ,其中对于Q的共轭向量，如果非零向量,则x,y是Q的共轭向量。我们需要找到n个相互的Q的共轭的基向量,,……,它们相互共轭且线性无关，因此空间中任意向量x都可以用这组基向量表示：

，

因此我们可以将目标函数改写成：

=

因为,,……是共轭的，所以由于=0,如果i!=j,上式可以继续简化为： ,

将求导符号提出来可以得到：

这样我们分别求每一项的最小值，即可，直接求导得：即可，现在我们只需要构建关于Q互相共轭得向量组,,……即可，有一种称为Gram-Schmidt过程的算法可以用来找到这组向量。在实际算法实现中，我们往往是边求,边求

2.实现

令正定矩阵：(，初始向量 = ,向量

伪代码如下：

=b-Q(为第i次迭代的误差)

=(是我们要求的共轭向量)

k:=0(k表示第几次迭代)

repeat

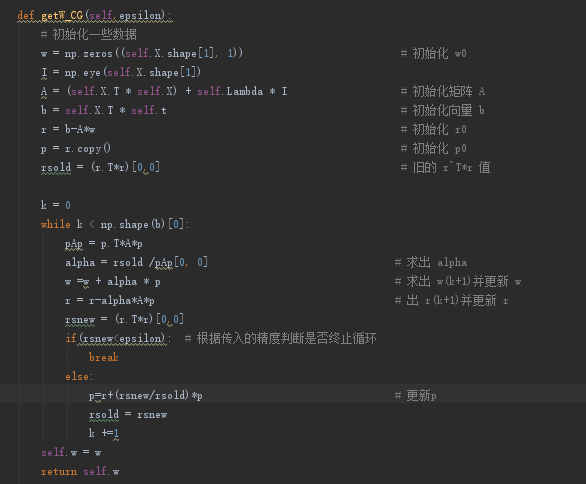
=+

=-

如果足够小，则提前退出循环

:=

Python代码实现如下：



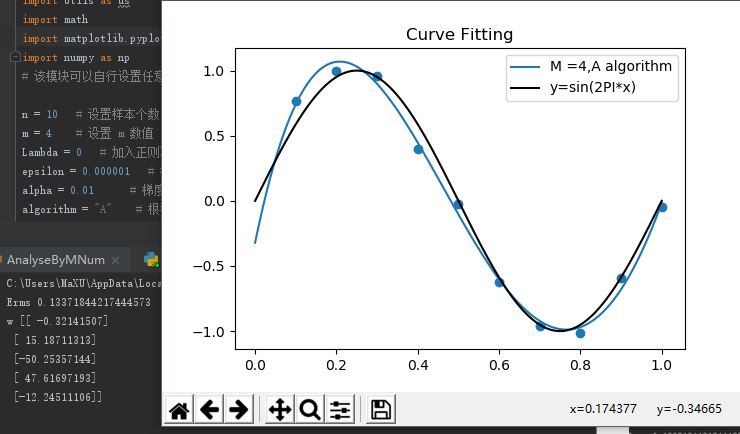
# 4、实验结果与分析

## 4.1.使用不同算法求解不带正则项的向量w

这里选用 n = 10， m = 4的数据进行拟合，求出各个算法的w和E(w)

计算Erms的数据集为与训练集不同的15个随机样本。

(1).使用解析解求解



Erms = 0.13371844217444573

W = [[ -0.32141507]

[ 15.18711313]

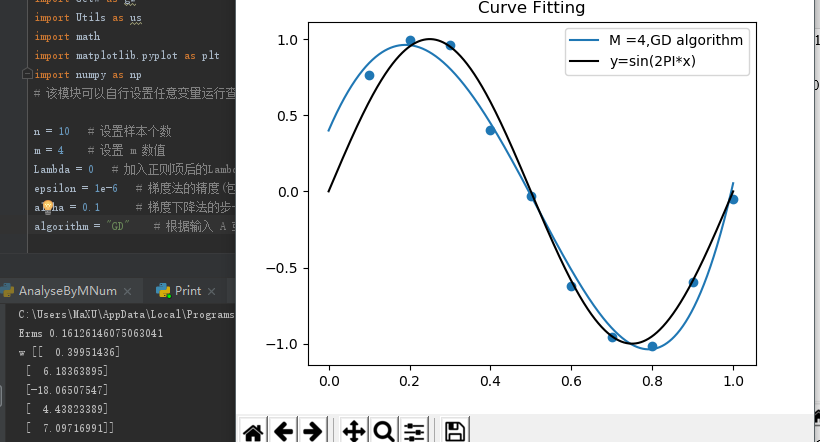
[-50.25357144]

[ 47.61697193]

[-12.24511106]]

(2).使用梯度下降法求解

其中精度为1e-6，步长为0.1



Erms = 0.16126146075063041

w = [[ 0.39951436]

[ 6.18363895]

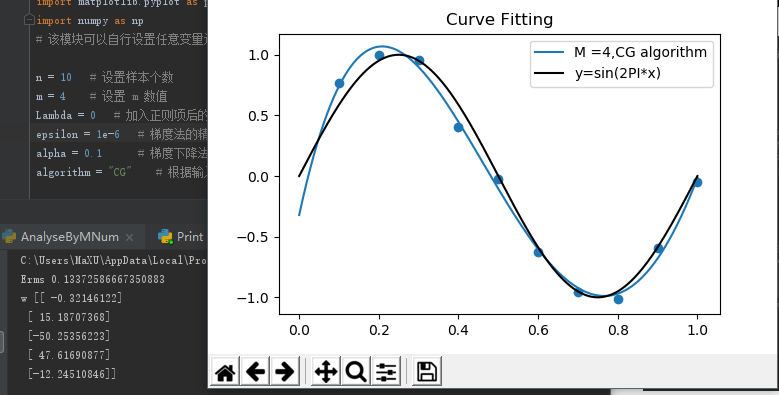
[-18.06507547]

[ 4.43823389]

[ 7.09716991]]

(3).使用共轭梯度法求解

精度为1e-6



Erms = 0.13372586667350883

w = [[ -0.32146122]

[ 15.18707368]

[-50.25356223]

[ 47.61690877]

[-12.24510846]]

分析：三种方法都能求出w解，但是梯度下降法更为耗时，且梯度下降法求得的最好w与设置的精度和步长有很大关系；解析解求出来的解更为准确，但是有条件限制，即必须在有解的情况下才能使用解析解；共轭方法明显优于梯度下降法，因为不需要调试下降的步长，所以剩力，同时迭代最大次数只与w的维度有关，且得到的结果也较为贴近解析解。另外当精度达到一定条件后，梯度下降法和共轭梯度法都不能再降低Erms了。

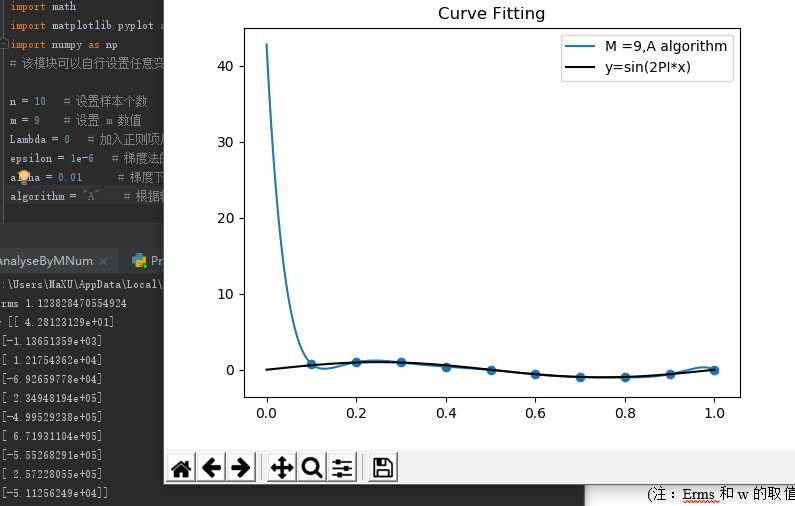
## 4.2.使用不同算法求解带正则项的向量w

这里选用 n = 10， m = 4，=exp(-15)的数据进行拟合，求出各个算法的w和E(w)

计算Erms的数据集为与训练集不同的20个随机测试数据集计算得到的结果。

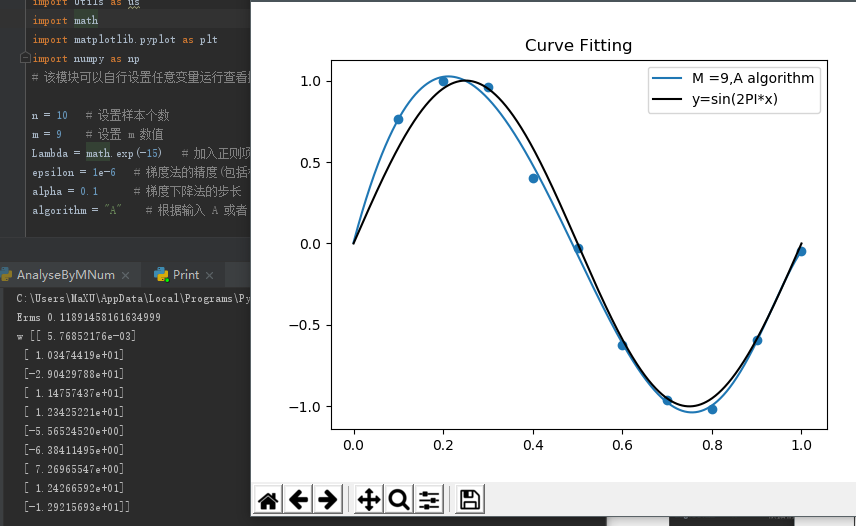
(1).使用解析解法求解

①．当不含有正则项时，如下：



(注：Erms和w的取值已经在图左下角)

②．当含有正则项时如下：

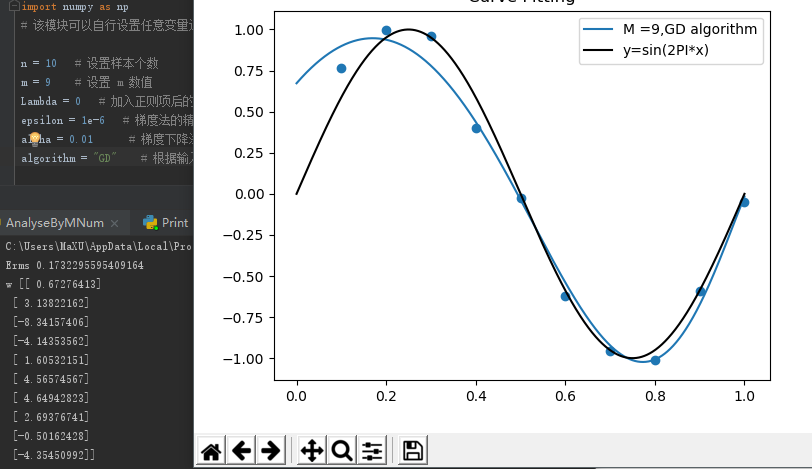


(注：Erms和w的取值已经在图左下角)

(2).使用梯度下降法求解

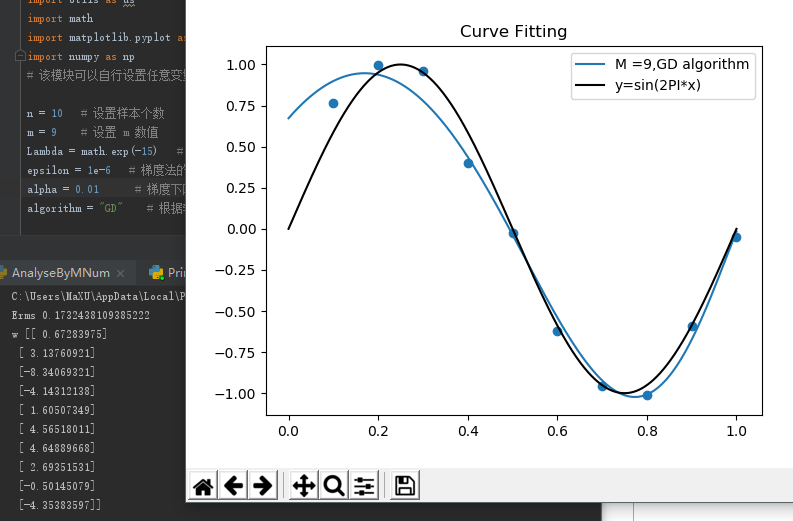
精度取1e-6，步长取0.01

①．当不含有正则项时，如下：



(注：Erms和w的取值已经在图左下角)

②．当含有正则项时如下：

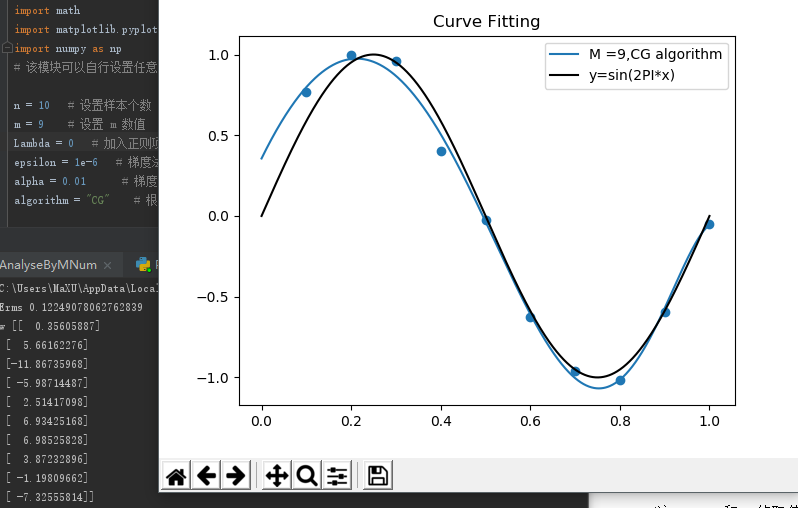


(注：Erms和w的取值已经在图左下角)

(3). 使用共轭梯度法求解

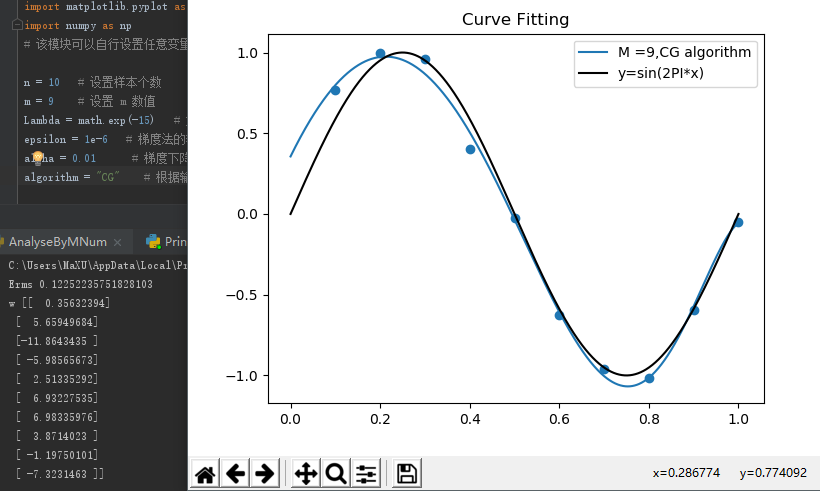
精度取1e-6

①．当不含有正则项时，如下：



(注：Erms和w的取值已经在图左下角)

②．当含有正则项时如下：



(注：Erms和w的取值已经在图左下角)

分析：通过上面的图线拟合对比，可以发现，对于解析解，如果不加正则项，则会出现很严重的过拟合现象，加上正则项后将有效提高函数的拟合程度，降低损失；但是当使用优化方法，梯度下降法和共轭梯度法时，即使不添加正则项也能拟合的很好，几乎无影响。其中Erms均为使用测试集数据进行计算的结果。

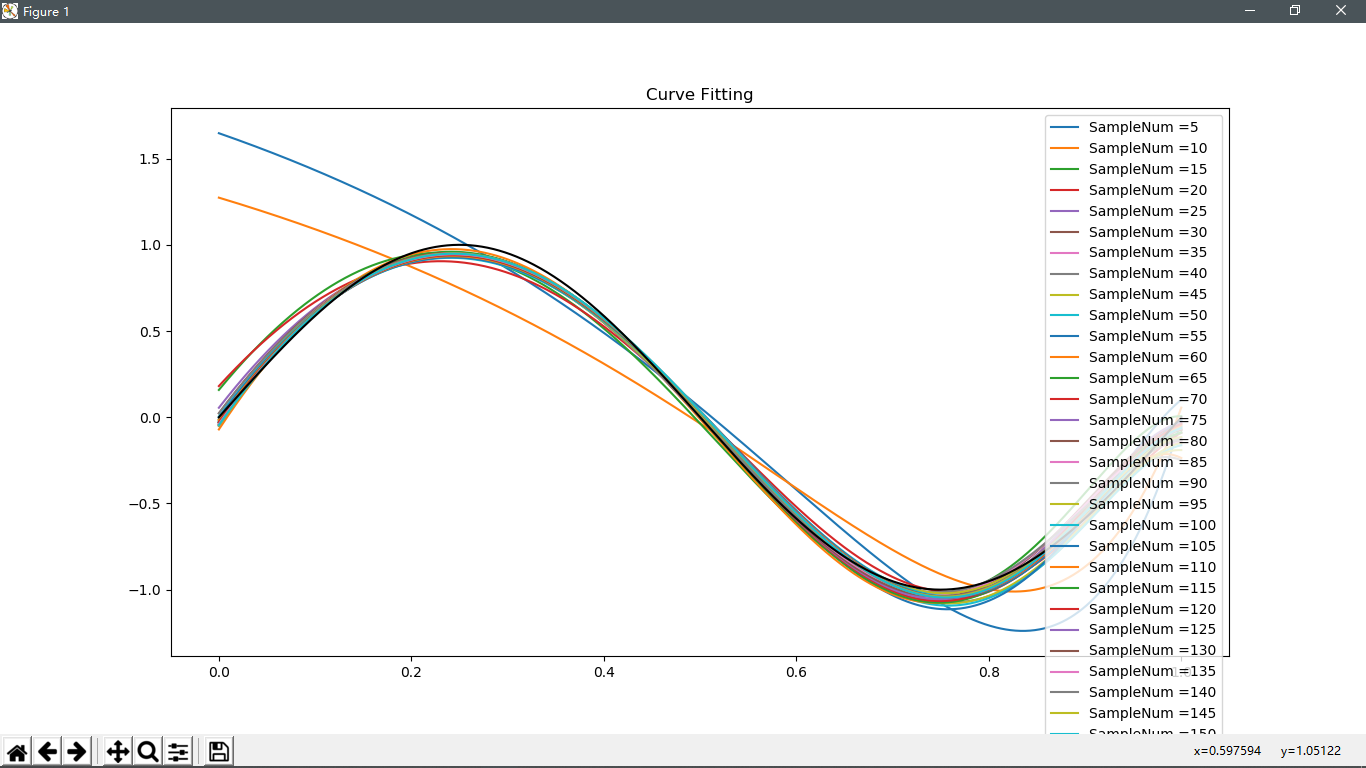
## 4.4.不同数量级样本对过拟合的影响

这里选用 m = 9，=0的数据进行拟合，求出n从5变化到150的Erms对比

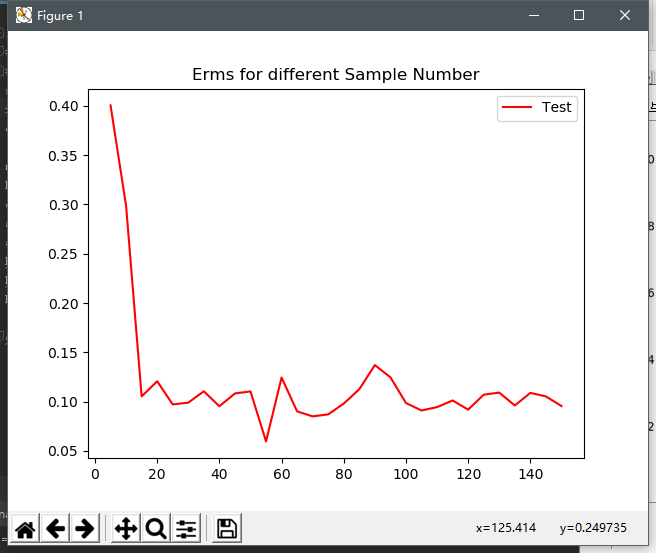
计算Erms的数据集为与训练集不同的15个随机测试数据集计算得到的结果。

。

拟合的函数图像为：



Erms与样本数量间的关系为：



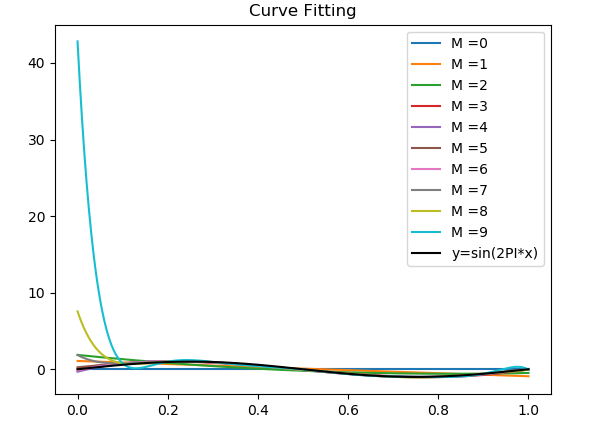
分析：通过图像可以发现，随着训练样本的增加，可以有效的防止过拟合，使Erms下降，但是当训练样本增加到一定程度时，效果就不是太大了。

## 4.5.不同多项式阶数对过拟合的影响

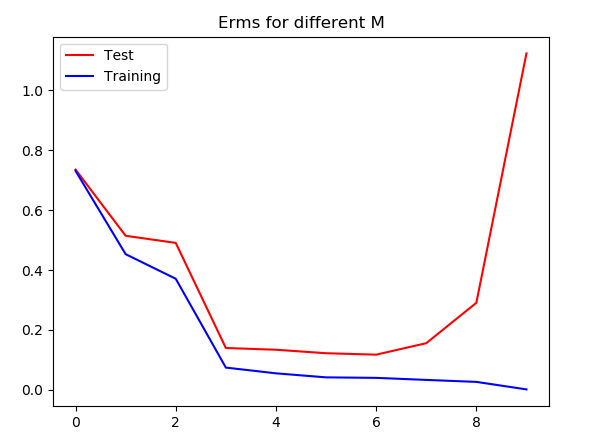
这里选用n = 10，=0的数据进行拟合，求出m从0变化到9的Erms对比

计算Erms的数据集为与训练集不同的15个随机测试数据集计算得到的结果。

1.拟合的函数图像为：



2.Erms与多项式之间的阶数关系：



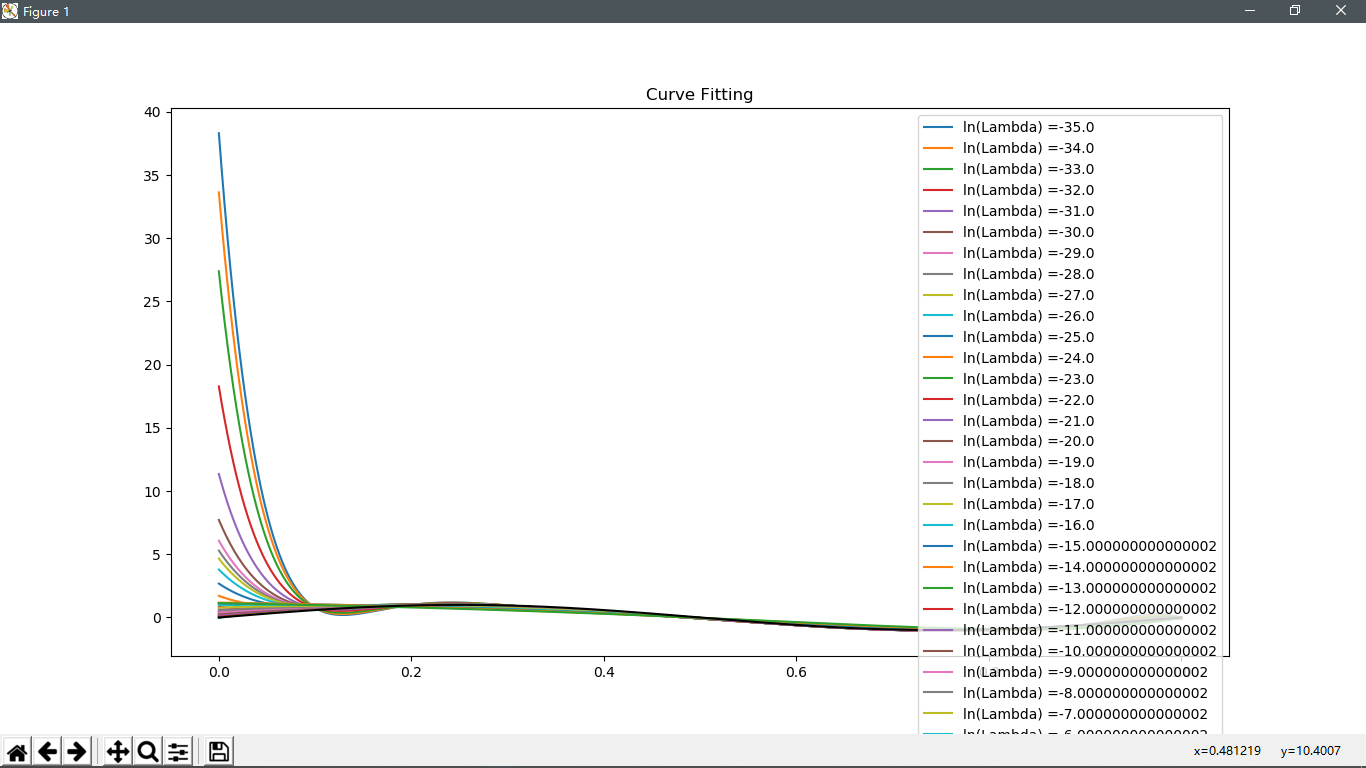
分析：对于测试集的Erms开始时随着M的增加降低，然后在3-7之间趋于稳定，在8-9时急速上升，而对于训练集的Erms开始随着M的增加降低，然后再3-7之间趋于稳定，在9时Erms为0，此时得到的拟合函数完全穿过训练集的10个样本。

## 4.6.不同超参数对过拟合的影响

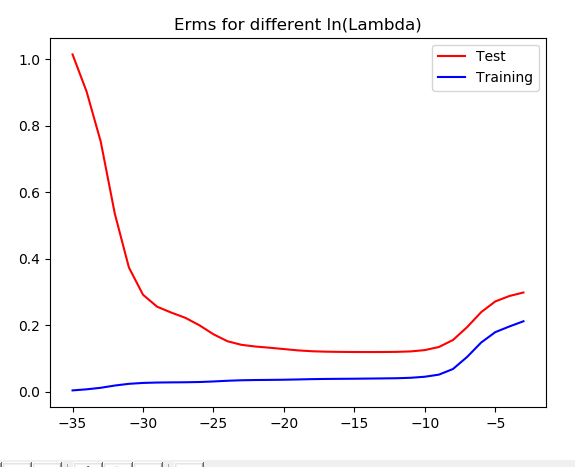
这里选用n = 10，m = 9的数据进行拟合，求出从-35变化到-5的Erms对比

计算Erms的数据集为与训练集不同的15个随机测试数据集计算得到的结果。

1.拟合的函数图像为：



2.Erms与超参数的关系：



分析：随着的逐渐增大，测试集上的Erms会出现一个突降，而训练集的Erms一直在缓慢上升，然后两者有一段相对稳定的时期后，两者又会上升到一定水平后保持稳定。这样说明随着一定范围的增大，会对过拟合有着很好矫正作用，而当超过这个范围继续增加后，反而会对数据的拟合造成一定负面影响，而后对于拟合的影响就不大了，这时主要通过样本数量来减小Erms。

# 5、结论

1. 优化的算法可以有效的矫正过拟合的问题

2. 当训练样本数据增加时，可以有效的矫正过拟合问题

3. 当在损失函数中添加正则项时，如果超参数在一定合理范围里取值，对于矫正过拟合有很好的作用。

4. 随着多项式阶数的增加，如果阶数过于接近训练集大小，则可能造成过拟合。

5. 拟合的函数与采集的数据也有很大关系，如果是关于自变量均匀采集的，对于函数拟合，将会得到很清晰的关系图像，如果不是均匀采集，则对函数拟合有一定影响，得到的关系图也可能不是很清晰。

# 6、参考文献

[1].刘建平博客《梯度下降小结》 <https://www.cnblogs.com/pinard/p/5970503.html>

[2].维基百科 《Conjugate gradient method》

<https://en.wikipedia.org/wiki/Conjugate_gradient_method>

[3].shjyoudp博客《共轭梯度法》 <https://blog.csdn.net/qq547276542/article/details/78186050>

[4].廖雪峰官方网站《Python教程》

<https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000>

[5].周志华.《机器学习》[M]，清华大学出版社，2016-1-1，54-56

# 7、附录：源代码（带注释）

## 7.1.生成样本数据部分

import numpy as np

import math

# 该部分生成实验的样本数据

class CreateSample(object):

x = [] # 自变量数组x

y = [] # 生成的因变量数组y

sampleNum = 10 # 生成数据的个数

mu = 0 # 正态分布的mu

sigma = 0.1 # 正态分布的sigma

seed = 3 # 设置的随机数种子

def \_\_init\_\_(self,sampleNum): # 参数为样本数量

self.sampleNum = sampleNum

def createData(self):

np.random.seed(self.seed)

self.x = np.linspace(1.0/self.sampleNum,1,self.sampleNum)

self.y = np.sin(2.0\*np.pi\*self.x)

for i in range(self.x.size):

self.y[i]+=np.random.normal(self.mu,self.sigma)

def setNormal(self,mu,sigma): # 可设置正态分布的 mu 和 sigma

self.mu = mu

self.sigma = sigma

def setSeed(self,seed): # 可设置随机数种子

self.seed = seed

## 7.2一些辅助的函数（用来生成矩阵X,向量t和计算Erms）

import math

import numpy as np

import CreateSample as cs

# 该部分用来根据传入的 M,x,y,和Lambda来计算需要的矩阵 X 和向量 t

# 及计算Erms

class Utils(object):

M = 0 # 记录M的取值

X = [] # 记录得到的X矩阵

t = [] # 记录得到的t向量

Lambda = 0

ERMS = 0

def \_\_init\_\_(self,M,x,y,Lambda): # 参数为 M,x,y,Lambda

self.M = M

self.t = np.mat(y)

self.Lambda = Lambda

self.X = self.generateX(x)

self.t = self.generateT(y)

def generateX(self,x): # 生成X矩阵的方法

X = 0

for i in x:

j = 0

row = [];

while j <= self.M:

row.append(math.pow(i, j))

j = j + 1

if (i == x[0]):

X = np.mat(row)

else:

X = np.row\_stack((X, row))

return X

def generateT(self,t): # 生成T向量的方法

return np.transpose(np.mat(t))

def calculateE(self,w): # 计算测试集的Erms

a = cs.CreateSample(15)

a.setSeed(346)

a.createData()

X = self.generateX(a.x)

t = self.generateT(a.y)

EW = lambda W:0.5\*((X\*W-t).T\*(X\*W-t))

N = np.shape(X)[0]

self.ERMS = math.pow(2\*EW(w)/N,0.5)

return self.ERMS

def caculateTraining(self,w): # 计算训练集的Erms

EW = lambda W:0.5\*((self.X\*W-self.t).T\*(self.X\*W-self.t))

N = np.shape(self.X)[0]

self.ERMS = math.pow(2\*EW(w)/N,0.5)

return self.ERMS

## 7.3计算W的算法（解析解、梯度下降法、共轭梯度法）

import numpy as np

import math

# 该部分为计算w的主要函数部分

class GetW(object):

X = 0

t = 0

w = 0

Lambda = 0

num = 0

def \_\_init\_\_(self,X,t,Lambda):

self.X = X

self.t = t

self.Lambda = Lambda

def haveSolution(self): # 判断是否有解，有解返回1，没有返回0

XshapeRow = np.shape(self.X)[1] # 计算矩阵X的列数

temp1 = np.dot(np.transpose(self.X), self.X) + np.eye(XshapeRow) \* self.Lambda

temp2 = np.dot(np.transpose(self.X),self.t)

temp3 = np.column\_stack((temp1,temp2))

if(np.linalg.matrix\_rank(temp1) == np.linalg.matrix\_rank(temp3)):

return 1

else:

return 0

def getW\_A(self): # 使用解析解计算

XshapeRow = np.shape(self.X)[1] # 计算矩阵X的列数

temp1 = np.dot(np.transpose(self.X),self.X)+np.eye(XshapeRow)\*self.Lambda # 计算X^T\*X+Lambda\*I

temp2 = np.dot(np.linalg.inv(temp1),np.transpose(self.X)) # 计算(X^T\*X+Lambda\*I)^-1 \* X^T

self.w = np.dot(temp2,self.t) # 计算(X^T\*X+Lambda\*I)^-1 \* X^T\*T

return self.w

def getW\_GD(self,epsilon,alpha): #使用梯度下降法计算

w = np.zeros((np.shape(self.X)[1],1))

EW\_G = lambda W:self.X.T\*(self.X\*W-self.t)+self.Lambda\*W # 对EW的求导项得到的结果

self.num = 0 # 记录迭代次数

while self.num < 100000:

temp = EW\_G(w)\*alpha # 记录下降的距离

if(math.fabs(temp.max())< epsilon): # 判断是否达到相应精度

break

else:

w = w - temp # 更新 w

self.num+=1

self.w = w

return self.w

def getW\_CG(self,epsilon): # 使用共轭梯度法计算

# 初始化一些数据

w = np.zeros((self.X.shape[1], 1)) # 初始化 w0

I = np.eye(self.X.shape[1])

A = (self.X.T \* self.X) + self.Lambda \* I # 初始化矩阵 A

b = self.X.T \* self.t # 初始化向量 b

r = b-A\*w # 初始化 r0

p = r.copy() # 初始化 p0

rsold = (r.T\*r)[0,0] # 旧的 r^T\*r 值

k = 0

while k < np.shape(b)[0]:

pAp = p.T\*A\*p

alpha = rsold /pAp[0, 0] # 求出 alpha

w =w + alpha \* p # 求出 w(k+1)并更新 w

r = r-alpha\*A\*p # 出 r(k+1)并更新 r

rsnew = (r.T\*r)[0,0]

if(rsnew<epsilon): # 根据传入的精度判断是否终止循环

break

else:

p=r+(rsnew/rsold)\*p # 更新p

rsold = rsnew

k +=1

self.w = w

return self.w

## 7.4一个可以修改各种参数，并且打印出对应拟合图像及Erms值的模块

import CreateSample as cs

import GetW as gw

import Utils as us

import math

import matplotlib.pyplot as plt

import numpy as np

# 该模块可以自行设置任意变量运行查看拟合函数和得到Erms值

n = 10 # 设置样本个数

m = 9 # 设置 m 数值

Lambda = 0 # 加入正则项后的Lambda取值

epsilon = 1e-6 # 梯度法的精度(包括梯度下降法和共轭梯度法)

alpha = 0.01 # 梯度下降法的步长

algorithm = "A" # 根据输入 A 或者 CG 或者 GD 选择不同算法

# 下面两句用来产生数据

a = cs.CreateSample(n)

a.createData()

# 使用Utils工具，得到 X 矩阵,t 向量

utils = us.Utils(m, a.x, a.y, Lambda)

# 计算得到 W ,可通过修改getW的使用方法，选择使用

# 解析解方法 getW.getW\_A

# 还是梯度下降法 getW.getW\_GD

# 还是共轭梯度法求解 getW.CG

getW = gw.GetW(utils.X,utils.t,utils.Lambda)

if algorithm == "A":

w = getW.getW\_A()

elif algorithm == "CG":

w = getW.getW\_CG(epsilon)

elif algorithm == "GD":

w = getW.getW\_GD(epsilon, alpha)

# 计算 ERMS

ERMS = utils.calculateE(w)

print("Erms",ERMS)

# 下面几句用来生成拟合的曲线

x = np.linspace(0,1,1000)

y = 0\*x

for i in range(np.shape(w)[0]):

y += pow(x,i)\*w[i,0]

print("w",w)

plt.title("Curve Fitting")

plt.plot(x,y,label='M ='+str(m)+","+algorithm+" algorithm")

# 画原函数曲线

xO = np.linspace(0,1,1000)

yO = np.sin(2\*math.pi\*xO)

plt.plot(xO,yO,color="black",label="y=sin(2PI\*x)")

plt.scatter(a.x,a.y) # 用来 生成描绘生成的数据样本

plt.legend()

plt.show()

## 7.5.运行后可以自动分析Erms（训练集和测试集）与M值关系并展现到图线上的模块

import CreateSample as cs

import GetW as gw

import Utils as us

import math

import matplotlib.pyplot as plt

import numpy as np

# 改模块运行会比较不同 M 对于拟合的影响,先后显示两个图像,分别是不同M值拟合的所有图像与不同M值对应得Erms(训练集和测试集)图像

# 通过修改循环的初始值和叠加值及终止值控制 M 的循环

start = 0

feed = 1 #每步加1

end = 9

n = 10 # n 的取值

Lambda = 0 # 加入正则项后的Lambda取值

epsilon = 0.001 # 梯度法的精度(包括梯度下降法和共轭梯度法)

alpha = 0.002 # 梯度下降法的步长

algorithm = "A" # 根据输入 A 或者 CG 或者 GD 选择不同算法

ErmsTest = []

ErmsTraining = []

M = []

def mytest(a,m):

# 得到 X 矩阵 ， t 向量

utils = us.Utils(m,a.x,a.y,Lambda)

# 选择算法

getW = gw.GetW(utils.X,utils.t,utils.Lambda)

if algorithm == "A":

w = getW.getW\_A()

elif algorithm =="CG":

w = getW.getW\_CG(epsilon)

elif algorithm == "GD":

w = getW.getW\_GD(epsilon,alpha)

# 计算 Erms

ermsTest = utils.calculateE(w)

ErmsTest.append(ermsTest)

ermsTraining = utils.caculateTraining(w)

ErmsTraining.append(ermsTraining)

# 下面几句用来生成拟合的曲线

x = np.linspace(0,1,1000)

y = 0\*x

for i in range(np.shape(w)[0]):

y += pow(x,i)\*w[i,0]

plt.title("Curve Fitting")

plt.plot(x,y,label='M ='+str(m))

print('M ='+str(m)+" ,ErmsTest:",ermsTest)

print('M ='+str(m)+" ,ErmsTraining:",ermsTraining)

# /plt.scatter(a.x,a.y)

def sim(start,feet,end):

m = start

a = cs.CreateSample(n)

a.createData()

while m <= end:

M.append(m)

mytest(a,m)

m += feet

sim(start,feed,end)

# 画原函数曲线

xO = np.linspace(0,1,1000)

yO = np.sin(2\*math.pi\*xO)

plt.plot(xO,yO,color="black",label="y=sin(2PI\*x)")

# print(Erms)

plt.legend()

plt.show()

# 下面是画 M 取值与Erms的关系

plt.title("Erms for different M")

plt.plot(M,ErmsTest,color="red",label="Test")

plt.plot(M,ErmsTraining,color="blue",label="Training")

plt.legend()

plt.show()

## 7.6. 运行后可以自动分析Erms（训练集和测试集）与超参数值关系并展现到图线上的模块

import CreateSample as cs

import GetW as gw

import Utils as us

import math

import matplotlib.pyplot as plt

import numpy as np

# 改模块运行会比较不同超参数对于拟合的影响,先后显示两个图像,分别是不同Lambda值拟合的所有图像与不同M值对应得Erms(训练集和测试集)图像

# 通过修改循环的初始值和叠加值及终止值控制 超参数 的循环

start = math.exp(-35)

feed = math.e # 每次乘e

end = math.exp(-3)

n = 10 # 设置 n 的值

m = 9 # 设置 m 数值

epsilon = 0.001 # 梯度法的精度(包括梯度下降法和共轭梯度法)

alpha = 0.002 # 梯度下降法的步长

algorithm = "A" # 根据输入 A 或者 CG 或者 GD 选择不同算法

ErmsTest = []

ErmsTraining = []

LAM = []

def mytest(a,Lambda):

# 得到 X 矩阵 ， t 向量

utils = us.Utils(m,a.x,a.y,Lambda)

# 选择算法

getW = gw.GetW(utils.X,utils.t,utils.Lambda)

if algorithm == "A":

w = getW.getW\_A()

elif algorithm =="CG":

w = getW.getW\_CG(epsilon)

elif algorithm == "GD":

w = getW.getW\_GD(epsilon,alpha)

# 计算 Erms

ermsTest = utils.calculateE(w)

ErmsTest.append(ermsTest)

ermsTraining = utils.caculateTraining(w)

ErmsTraining.append(ermsTraining)

# 下面几句用来生成拟合的曲线

x = np.linspace(0,1,1000)

y = 0\*x

for i in range(np.shape(w)[0]):

y += pow(x,i)\*w[i,0]

plt.title("Curve Fitting")

plt.plot(x,y,label='ln(Lambda) ='+str(math.log(Lambda)))

print('M =' + str(m) + " ,ErmsTest:", ermsTest)

print('M =' + str(m) + " ,ErmsTraining:", ermsTraining)

# /plt.scatter(a.x,a.y)

def sim(start,feet,end):

Lambda = start

a = cs.CreateSample(n)

a.createData()

while Lambda <=end:

LAM.append(math.log(Lambda))

mytest(a,Lambda)

Lambda \*= feet

sim(start,feed,end)

# 画原函数曲线

xO = np.linspace(0,1,1000)

yO = np.sin(2\*math.pi\*xO)

plt.plot(xO,yO,color="black",label="y=sin(2PI\*x)")

plt.legend()

plt.show()

# 下面是画超参数与Erms的关系

plt.title("Erms for different ln(Lambda)")

plt.plot(LAM,ErmsTest,color="red",label="Test")

plt.plot(LAM,ErmsTraining,color="blue",label="Training")

plt.legend()

plt.show()

## 7.7. 运行后可以自动分析Erms（训练集和测试集）与样本数个数关系并展现到图线上的模块

import CreateSample as cs

import GetW as gw

import Utils as us

import math

import matplotlib.pyplot as plt

import numpy as np

# 改模块运行会比较不同超参数对于拟合的影响,先后显示两个图像,分别是不同Lambda值拟合的所有图像与不同M值对应得Erms(训练集和测试集)图像

# 通过修改循环的初始值和叠加值及终止值控制 超参数 的循环

start = math.exp(-35)

feed = math.e # 每次乘e

end = math.exp(-3)

n = 10 # 设置 n 的值

m = 9 # 设置 m 数值

epsilon = 0.001 # 梯度法的精度(包括梯度下降法和共轭梯度法)

alpha = 0.002 # 梯度下降法的步长

algorithm = "A" # 根据输入 A 或者 CG 或者 GD 选择不同算法

ErmsTest = []

ErmsTraining = []

LAM = []

def mytest(a,Lambda):

# 得到 X 矩阵 ， t 向量

utils = us.Utils(m,a.x,a.y,Lambda)

# 选择算法

getW = gw.GetW(utils.X,utils.t,utils.Lambda)

if algorithm == "A":

w = getW.getW\_A()

elif algorithm =="CG":

w = getW.getW\_CG(epsilon)

elif algorithm == "GD":

w = getW.getW\_GD(epsilon,alpha)

# 计算 Erms

ermsTest = utils.calculateE(w)

ErmsTest.append(ermsTest)

ermsTraining = utils.caculateTraining(w)

ErmsTraining.append(ermsTraining)

# 下面几句用来生成拟合的曲线

x = np.linspace(0,1,1000)

y = 0\*x

for i in range(np.shape(w)[0]):

y += pow(x,i)\*w[i,0]

plt.title("Curve Fitting")

plt.plot(x,y,label='ln(Lambda) ='+str(math.log(Lambda)))

print('M =' + str(m) + " ,ErmsTest:", ermsTest)

print('M =' + str(m) + " ,ErmsTraining:", ermsTraining)

# /plt.scatter(a.x,a.y)

def sim(start,feet,end):

Lambda = start

a = cs.CreateSample(n)

a.createData()

while Lambda <=end:

LAM.append(math.log(Lambda))

mytest(a,Lambda)

Lambda \*= feet

sim(start,feed,end)

# 画原函数曲线

xO = np.linspace(0,1,1000)

yO = np.sin(2\*math.pi\*xO)

plt.plot(xO,yO,color="black",label="y=sin(2PI\*x)")

plt.legend()

plt.show()

# 下面是画超参数与Erms的关系

plt.title("Erms for different ln(Lambda)")

plt.plot(LAM,ErmsTest,color="red",label="Test")

plt.plot(LAM,ErmsTraining,color="blue",label="Training")

plt.legend()

plt.show()