**多项式函数拟合**

****

**姓名：张奇**

**学号：1132130124**

**问题背景**

生活中，我们总是在挖掘事物背后的无形力量，称之为规律的东西。

通过对大量数据进行统计，进而构建模型曲线，从而可以利用其去预测下一次的结果。

我们在这里，简单地，对一些离散点，做一条可表达的函数曲线，使这些点与曲线的误差很小，理想地达到重合的程度，即曲线拟合。

已知某个函数（未知）一些样本点，以及其对应的函数值，我们需要了解函数大致的走向，形状，并对一些未知函数值的样本点进行预测。

**实验原理**

在数学中，泰勒公式是一个用函数在某点的信息描述其附近取值的公式。如果函数足够平滑的话，在已知函数在某一点的各阶导数值的情况之下，泰勒公式可以用这些导数值做系数构建一个多项式来近似函数在这一点的邻域中的值。

由泰勒公式可以看出用多项式函数来拟合一个平滑函数的可行性。

我们采用最小二乘法拟合公式进行拟合

添加正则项：根据奥卡姆剃刀原理，在所有的模型中，能很好的解释已知的数据并且十分简单才是最好的模型。因此我们再在优化目标上加上正则项。

**实验步骤：**

**一．样本点的生成：**

在给定的区间（-1，+1）内利用正弦函数，并加上一定量的噪声生成训练集，验证集

训练集用于对参数的训练，验证集用于对不同模型的选择。

**二．不添加正则项**

利用最小二乘公式进行系数的计算。画出图形。

**三．添加正则项**

**1.找到优化目标：**

L(w)=)^2 +λ||w||

其中正则项对应着模型参数的先验分布，选取不同的正则项，对应着对模型参数先验分布的不同预期

本实验中，设置了两种惩罚，分别是L2范数惩罚和平方和形式的惩罚。

（1）平方和形式的惩罚

||w||=

(2)L2范式的惩罚

||w||

**优化方法进行选择**：

牛顿法：需要进行多元函数泰勒展开，展开到第二项。求海森阵，判断是否处于下降方向。个人觉得比较麻烦，没有使用这种方法。

梯度下降法：

沿着负梯度方向在给定的步长下搜索，实现简单。步长确定费时，收敛速度较慢。本实验选取这种方法求解

最速下降法：

梯度下降法的一种改进，每次需要重新确定步长。由于编程语言采用python,所以确定步长所需要求一元函数的极值不易实现。

共轭梯度法：

适用于凸二次函数，收敛较快，在优化带有平方和形式的惩罚时采用这种方法。

故选用梯度下降法以及共轭梯度法

**对优化目标进行计算：**

（1）普通梯度方法

在给定的一组λ，最高阶为9阶的多项式上进行训练。

求梯度:

对于平方和形式的惩罚：

=i\*wi\*+2\*λ\*wi

对于L2范式的惩罚

=i\*wi\* +

步长的选择，步长的选择是一件很棘手的事情，步长如果设置过大，就会偏离超过目标函数的最小值太多，导致目标函数不能达到最优解，选择出来的参数所描述的函数自然也会和样本点偏离较大。

如果步长选择较小，会得到较大的结果。但是收敛速度过慢。

最终经过反复的尝试，将步长确定为10e-5.

（2）共轭梯度法

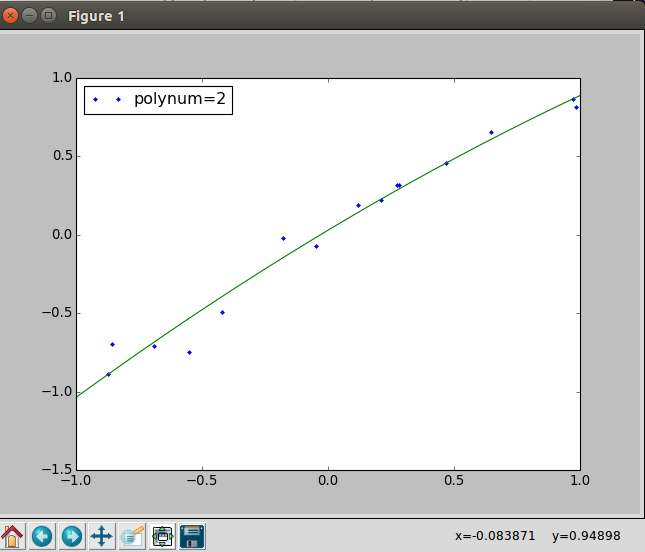
手动计算，将二次函数写成

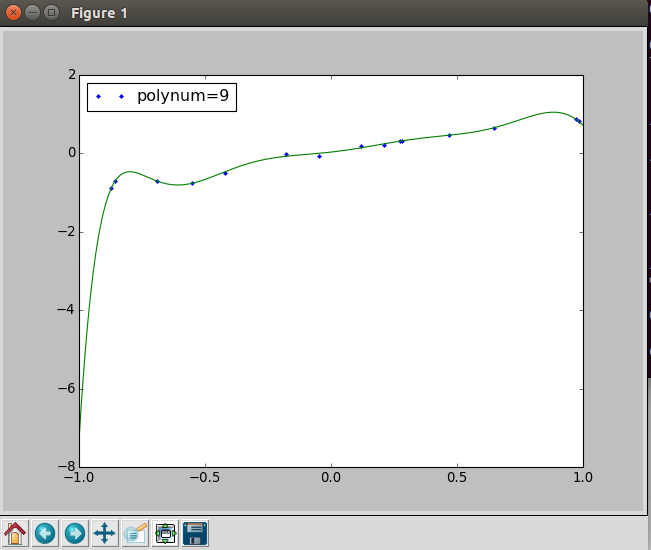
F=的形式。

基于最速下降法，并利用共轭向量调整搜索方向。

**实验结果分析**

**不添加惩罚项**



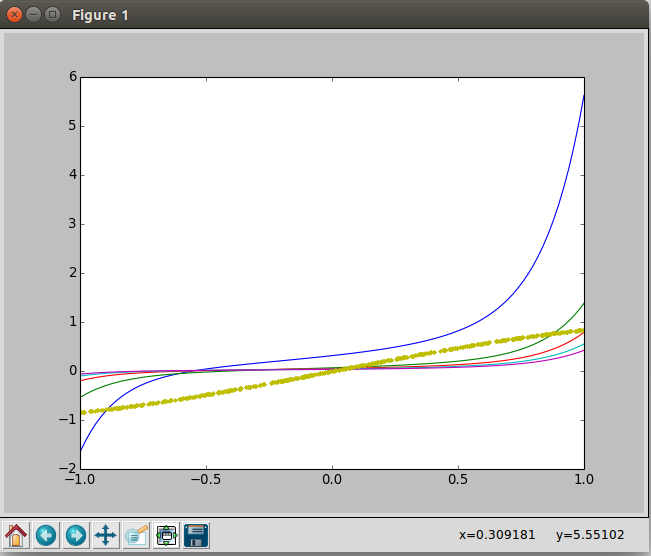


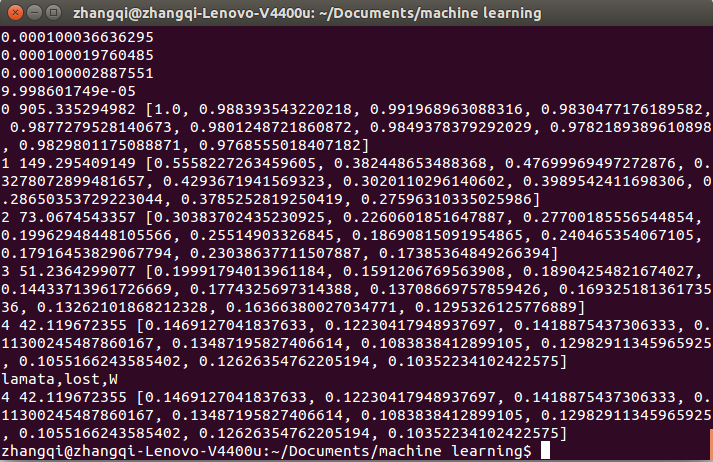
随着阶数的增加，拟合的效果越来越好，同时也将噪声给学习到了模型里，

因此我们可以看出添加正则项的必要

**添加惩罚项**

对于平方和形式的惩罚，当收敛条件设置为0.0001时训练集为500，本身不大，还没有完全收敛（初始值对收敛速度显然也有很大的影响），运行时间已经超过20分钟。

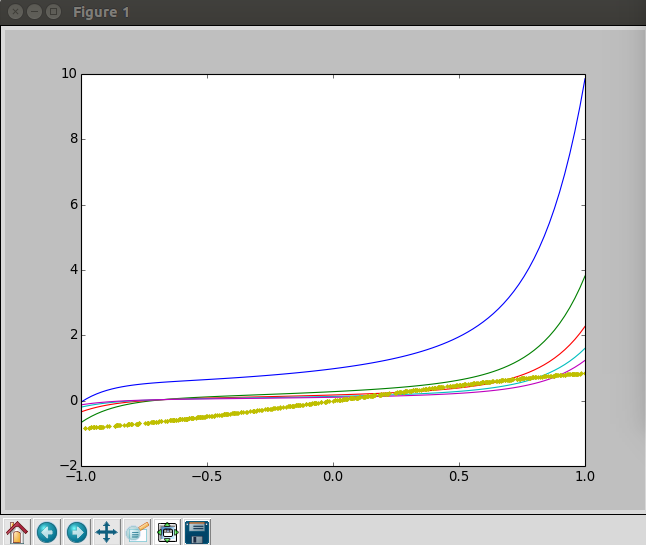


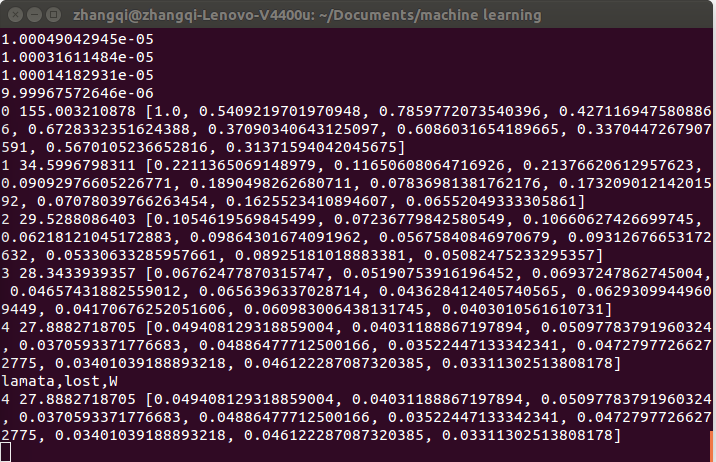


途中对应的四条曲线分别对应着四种惩罚，在还没有完全收敛的时候，可以看出，惩罚项大的对应的曲线更接近原样本点。

（分析可能还没有完全收敛，而惩罚项越大，每次的学习率越大，能更快的接近结果）

当收敛条件设置为0.00001时，训练集为500，收敛的时间变得更长，60分钟左右。训练的结果变好，更好的描述原样本点。





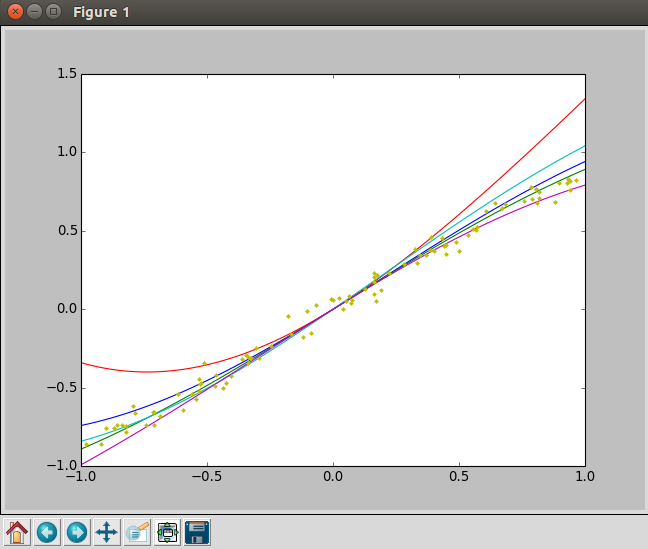
将训练得到的模型预测验证集上的数据，并与真实的值进行比较，得到一个损失函数值。

以上图形输出的是两种迭代收敛值（0.0001，0.00001）对应的惩罚λ（0，1，2，3，4）所对应的曲线。

命令行输出的是惩罚前系数λ其损失函数以及具体的参数值W。可以看出λ大的对应的高次变量前的系数越小，可以看出惩罚项的设置在训练中的约束作用。

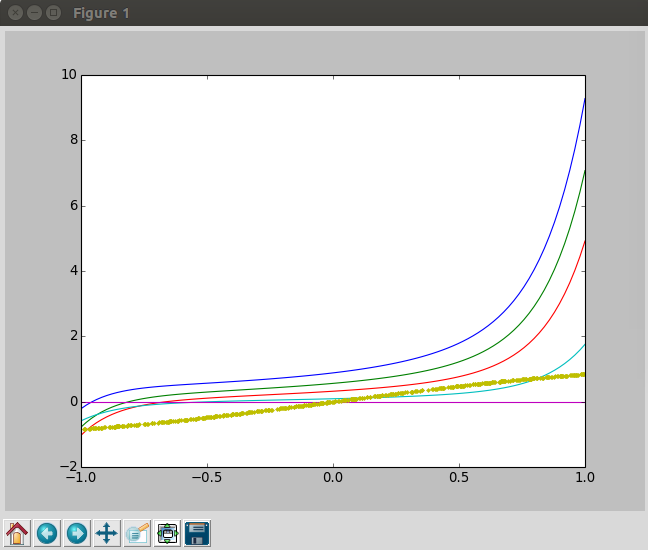
将不同（对应不同模型）的损失函数值做比较，选择最优模型。

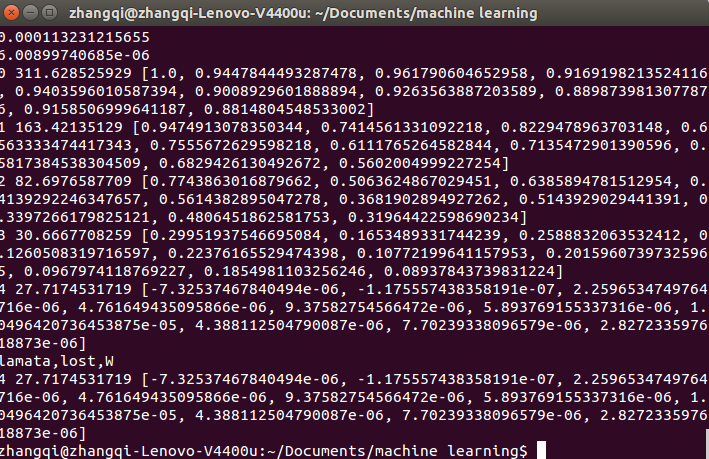
减少训练样本100，并且将收敛值设到10e-20。



把收敛阈值设置的十分小，并且在减少样本的情况下，拟合效果达到了一个理想的状态

L2范式的惩罚训练结果



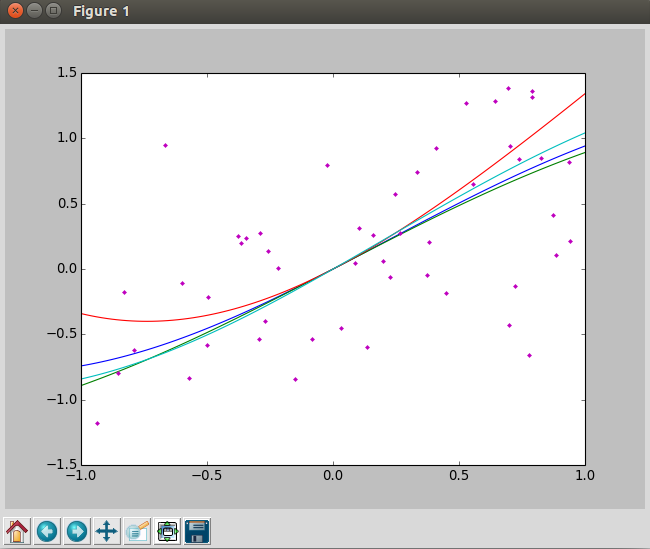


λ越大的对应模型的曲线越平滑，高阶项系数越小。由于梯度下降收敛缓慢。所以训练出来的模型拟合原来的数据样本的结果不是十分理想。

在平方和形式的惩罚模型的训练集上添加噪声的结果：

添加噪声时(噪声设的比较大，惩罚项也设的较大（1，2，3，4）:

惩罚项大的曲线越平滑



**利用共轭梯度法平方和形式的惩罚模型**：

没有很好的理解这种方法,知道实验报告提交之前也没有得出很好的结果。

**附录代码**

梯度下降法求解平方和形式惩罚模型。

import math

import random

import scipy as sp

import matplotlib.pyplot as plt

W=[]

X=[]

Y=[]

step=0.00001

trainnum=50

testnum=10

polynum=10

X1=[]

Y1=[]

select={}

for i in range(polynum):

W.append(1);

for i in range(trainnum):

X.append(random.uniform(-1,1))

Y.append(math.sin(X[i]))

for i in range(testnum):

X1.append(random.uniform(-1,1))

Y1.append(math.sin(X[i]))

def qiupiandao(i,W,lamata,X,Y):

sumh=2\*lamata\*W[i]

for j in range(trainnum):

sum0=0

for q in range(polynum):

sum0=sum0+W[q]\*(X[j]\*\*q)

if q==0:

sum0=sum0-Y[j]

sumh=sumh+sum0\*i\*W[i]\*X[j]\*\*(i-1)/trainnum

return sumh

def lostfuction(lamata,W,X,Y):

suml=0

for i in range(polynum):

suml=suml+lamata\*(W[i]\*\*2)

for j in range(trainnum):

sum0=0

for q in range(polynum):

sum0=sum0+W[q]\*(X[j]\*\*q)

if q==0:

sum0=sum0-Y[j]

suml=suml+sum0\*\*2/(2\*trainnum)

return suml

def error(f, x, y):

return sp.sum((f(x)-y)\*\*2)

for lamata in range(5):

while True:

prelost=lostfuction(lamata,W,X,Y)

for i in range(polynum):

W[i]=W[i]-step\*qiupiandao(i,W,lamata,X,Y)

postlost=lostfuction(lamata,W,X,Y)

print prelost-postlost

if prelost-postlost<0.000001:

break

f1 = sp.poly1d(W)

fx=sp.linspace(-1,1,100)

er=error(f1, X1, Y1)

plt.plot(fx, f1(fx))

#plt.legend(["lamata=%i" % lamata], loc="upper left")

select[lamata]=[]

select[lamata].append(er)

select[lamata].append(list(W))

for i in range(polynum):

W[i]=1

for lamata in range(5):

miner=0

print lamata,select[lamata][0],select[lamata][1]

if select[lamata][0] < select[miner][0]:

miner=lamata

print "lamata,lost,W"

print miner,select[miner][0],select[miner][1]

plt.plot(X,Y,'.')

plt.show()

梯度下降法求解L2范式惩罚模型。

import math

import random

import scipy as sp

import matplotlib.pyplot as plt

W=[]

X=[]

Y=[]

step=0.00001

trainnum=500

testnum=100

polynum=10

X1=[]

Y1=[]

select={}

for i in range(polynum):

W.append(1);

for i in range(trainnum):

X.append(random.uniform(-1,1))

Y.append(math.sin(X[i]))

for i in range(testnum):

X1.append(random.uniform(-1,1))

Y1.append(math.sin(X[i]))

def qiupiandao(i,W,lamata,X,Y):

suml=0

for q in range(polynum):

suml=suml+W[q]\*\*2

suml=math.sqrt(suml)

sumh=lamata\*W[i]/suml

for j in range(trainnum):

sum0=0

for q in range(polynum):

sum0=sum0+W[q]\*(X[j]\*\*q)

if q==0:

sum0=sum0-Y[j]

sumh=sumh+sum0\*i\*W[i]\*X[j]\*\*(i-1)/trainnum

return sumh

def lostfuction(lamata,W,X,Y):

suml=0

for i in range(polynum):

suml=suml+W[i]\*\*2

suml=lamata\*math.sqrt(suml)

for j in range(trainnum):

sum0=0

for q in range(polynum):

sum0=sum0+W[q]\*(X[j]\*\*q)

if q==0:

sum0=sum0-Y[j]

suml=suml+sum0\*\*2/(2\*trainnum)

return suml

def error(f, x, y):

return sp.sum((f(x)-y)\*\*2)

for lamata in range(5):

while True:

prelost=lostfuction(lamata,W,X,Y)

for i in range(polynum):

W[i]=W[i]-step\*qiupiandao(i,W,lamata,X,Y)

postlost=lostfuction(lamata,W,X,Y)

print prelost-postlost

if prelost-postlost<0.0001:

break

f1 = sp.poly1d(W)

fx=sp.linspace(-1,1,100)

er=error(f1, X1, Y1)

plt.plot(fx, f1(fx))

#plt.legend(["lamata=%i" % lamata], loc="upper left")

select[lamata]=[]

select[lamata].append(er)

select[lamata].append(list(W))

for i in range(polynum):

W[i]=1

for lamata in range(5):

miner=0

print lamata,select[lamata][0],select[lamata][1]

if select[lamata][0] < select[miner][0]:

miner=lamata

print "lamata,lost,W"

print miner,select[miner][0],select[miner][1]

plt.plot(X,Y,'.')

plt.show()

共轭梯度法求解平方和形式惩罚模型

梯度下降法求解求解平方和形式惩罚模型（样本带有噪声）

import math

import random

import scipy as sp

import matplotlib.pyplot as plt

import numpy as np

W=[]

X=[]

Y=[]

step=0.00001

trainnum=100

testnum=10

polynum=10

X1=[]

Y1=[]

select={}

Sigma=0.05

for i in range(polynum):

W.append(1);

for i in range(trainnum):

X.append(random.uniform(-1,1))

Y.append(math.sin(X[i])+np.random.normal()\*Sigma)

for i in range(testnum):

X1.append(random.uniform(-1,1))

Y1.append(math.sin(X[i])+np.random.normal()\*Sigma)

def qiupiandao(i,W,lamata,X,Y):

sumh=2\*lamata\*W[i]

for j in range(trainnum):

sum0=0

for q in range(polynum):

sum0=sum0+W[q]\*(X[j]\*\*q)

if q==0:

sum0=sum0-Y[j]

sumh=sumh+sum0\*i\*W[i]\*X[j]\*\*(i-1)/trainnum

return sumh

def lostfuction(lamata,W,X,Y):

suml=0

for i in range(polynum):

suml=suml+lamata\*(W[i]\*\*2)

for j in range(trainnum):

sum0=0

for q in range(polynum):

sum0=sum0+W[q]\*(X[j]\*\*q)

if q==0:

sum0=sum0-Y[j]

suml=suml+sum0\*\*2/(2\*trainnum)

return suml

def error(f, x, y):

return sp.sum((f(x)-y)\*\*2)

for lamata in range(5):

while True:

prelost=lostfuction(lamata,W,X,Y)

for i in range(polynum):

W[i]=W[i]-step\*qiupiandao(i,W,lamata,X,Y)

postlost=lostfuction(lamata,W,X,Y)

print prelost-postlost

if prelost-postlost<2.4e-10:

break

f1 = sp.poly1d(W)

fx=sp.linspace(-1,1,100)

er=error(f1, X1, Y1)

plt.plot(fx, f1(fx))

#plt.legend(["lamata=%i" % lamata], loc="upper left")

select[lamata]=[]

select[lamata].append(er)

select[lamata].append(list(W))

for i in range(polynum):

W[i]=1

for lamata in range(5):

miner=0

print lamata,select[lamata][0],select[lamata][1]

if select[lamata][0] < select[miner][0]:

miner=lamata

print "lamata,lost,W"

print miner,select[miner][0],select[miner][1]

plt.plot(X,Y,'.')

plt.show()

公式法求解不加正则项的拟合模型

import math

import random

import numpy

import scipy as sp

import matplotlib.pyplot as plt

import numpy as np

X=[]

Y=[]

trainnum=100

polynum=10

for i in range(trainnum):

X.append(random.uniform(-1,1))

Y.append(math.sin(X[i]))

M=np.zeros((polynum+1,polynum+1))

for i in range(polynum+1):

for j in range(polynum+1):

if j==0 and i==0:

M[i,j]=trainnum

else:

suml=0

for l in range(trainnum):

suml=suml+X[l]\*\*(i+j)

M[i,j]=suml

N=np.zeros((polynum+1,1))

W=np.zeros((polynum+1,1))

for j in range(polynum+1):

sumh=0

for l in range(trainnum):

sumh=sumh+(X[l]\*\*j)\*Y[l]

N[j,0]=sumh

W=np.linalg.inv(M)\*N

wa=[]

for i in range(polynum+1):

wa.append(W[i,0])

f1=sp.poly1d(wa)

fx=sp.linspace(-1,1,100000)

plt.plot(X,Y,'.')

plt.plot(fx, f1(fx), linewidth=3)

plt.show()