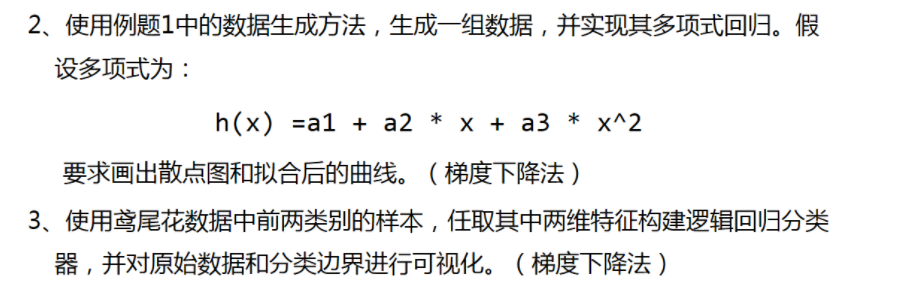
**智能信息系统综合实践**

**实验报告**

|  |  |
| --- | --- |
| **题 目：** | **线性模型回归分析** |
| **年 级：** | **2020** |
| **专 业：** | **软件工程** |
| **姓 名：** | **沈钰斐** |

1. **题目**





1. **解题步骤**
2. 代码如下。利用matplotlib包和numpy包，scipy包应用最小二乘法，**import numpy.linalg as lg确保输入中文不报错**。

* 输入散点数据（时间-速度）。注意这里需要**对变量的量纲进行处理**。
* 定义polyfit\_linear123(t,y,num)函数，应用**np.polgfit函数**得到线性拟合的的系数向量，**np.polyval函数**得到拟合参数多项式的y值，**np.poly1d函数**得到多项式函数式。

【另外，在应用这个函数之前，关于线性拟合的一般过程可以通过推导得到，见注释。具体如下】

#线性拟合 y=a\*t+b

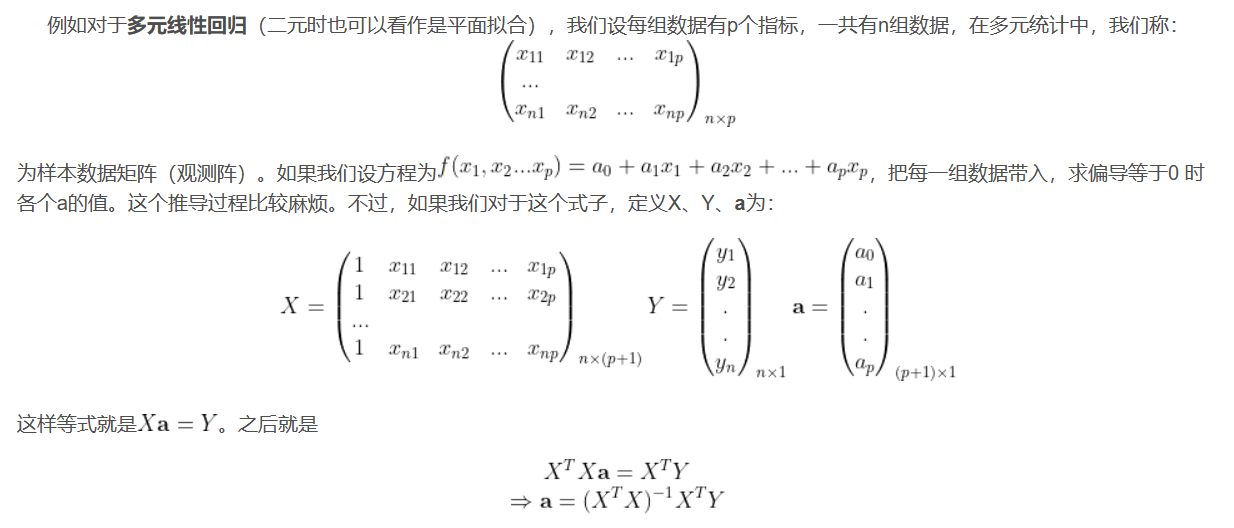
# #使用公式转置

# def linear(t,y):

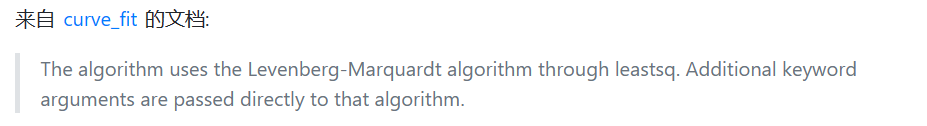
#     A = np.c\_[t,np.ones(t.shape)]

#     W = lg.inv(A.T.dot(A)).dot(A.T).dot(y)

#     plt.plot(t,W[0]\*t+W[1],'r')



* 对于非线性拟合：采用**scipy**包里函数来实现。实现非线性拟合有两种方式，一种是scipy的**leastsq函数**，一种是**curve\_fit函数**，关于两者的区别如下：



curve\_fit调用leastsq。在我的解决方案中使用leastsq函数。

* 以对数拟合为例，首先定义**对数拟合函数func1(x,p)**，以及**误差函数residuals(p,y,x)**，func1定义拟合函数的结构，residuals定义误差函数的表示，需要注意的是由于leastsq函数对误差函数的处理本来就是最小二乘法，所以只需要写入（预测值-实际值）。
* 设置拟合函数中的参数初始值，应用**plsq=leastsq(residuals,p0,args=(y,**

**t))**,返回plsp元组，第一项即拟合函数的参数值。

* 进而根据参数值，使用**matplotlib包**可以画出拟合图像。
* 对于误差值的计算，可以使用最小二乘法中预测值减实际值求得。进而，比较得出最合适的拟合函数。
* 问题需要求解200ms的加速度，可以使用**deriv(）函数**求函数导数，再一次求导可以求二阶导等。
* 补充：**关于回归和拟合**，从它们的求解过程以及结果来看，两者似乎没有太大差别。从本质上说，回归属于数理统计问题，研究解释变量与响应变量之间的关系以及相关性等问题。而拟合是把平面的一系列点，用一条光滑曲线连接起来，并且让更多的点在曲线上或曲线附近。更确切的说，拟合是回归用到的一种数学方法，而拟合与回归的应用场合不同。拟合常用的方法有最小二乘法、梯度下降法、高斯牛顿（即迭代最小二乘）、列-马算法。其中最最常用的是最小二乘法。拟合可以分为线性拟合与非线性拟合，非线性拟合比较常用的是多项式拟合。根据自变量的个数，拟合也可以分为曲线拟合与曲面拟合等。而回归大多数采用最小二乘法。回归可以分为一元线性回归、一元非线性回归、多元线性回归、多元非线性回归等。

import numpy as np

import matplotlib.pyplot as plt

import numpy.linalg as lg

from scipy.optimize import leastsq

# 保证输入中文不报错

from pylab import \*

mpl.rcParams['font.sans-serif'] = ['SimHei']

mpl.rcParams['axes.unicode\_minus'] = False

t = np.array([10,30,50,70,90,110,130,150,170,190,210,230,250,

                270,290,310,330,350,370,390,410,430,450,470,490])

# t = np.arange(10/1000,490/1000,25)

y = np.array([0,0.066,0.088,0.1539,0.1979,0.2419,0.2639,0.3079,0.3299,0.3738,0.3958,0.4398,0.4398,

              0.4618,0.4838,0.5498,0.5498,0.5718,0.5938,0.6158,0.6158,0.6377,0.6377,0.6597,0.6597])

#线性拟合 y=a\*t+b

# #使用公式转置

# def linear(t,y):

#     A = np.c\_[t,np.ones(t.shape)]

#     W = lg.inv(A.T.dot(A)).dot(A.T).dot(y)

#     plt.plot(t,W[0]\*t+W[1],'r')

#线性拟合：使用np.polgfit函数 实现最小二乘法拟合

def polyfit\_linear123(t,y,num):

    reg = np.polyfit(t,y,num) # 系数向量

    polyf = np.polyval(reg,t) # 含拟合参数多项式的y值

    Loss = np.sum((polyf - y)\*\*2) # 最小二乘误差

    plot1 = np.poly1d(reg)

    print(num,'元线性回归:')

    print(plot1)

    plt.plot(t,y,'k\*',label='Origin')

    plt.plot(t,plot1(t),'r',label='线性拟合')

    plt.xlabel('Time')

    plt.ylabel('Speed')

    plt.legend()

return Loss

def func1(x,p):

    '''定义对数拟合函数'''

    a,b,c = p

return a\*(np.log(x)/np.log(b))+c

def func2(x,p):

    '''定义幂拟合函数'''

    a,b,c = p

return a\*np.power(x,b)+c

def residuals(p,y,x):

    '''定义误差函数'''

return y-func1(x,p)

def residuals1(p,y,x):

    '''定义误差函数'''

return y-func2(x,p)

p0 = [1,2,3]

plsq = leastsq(residuals,p0,args=(y,t))

print('对数函数参数：',plsq[0]) # 对数函数拟合的参数

y\_log = func1(t,plsq[0])

loss\_log = np.sum((y\_log - y)\*\*2) # 最小二乘误差

# print(y\_log)

plt.plot(t,y\_log,'r',label='对数拟合')

plt.plot(t,y,'k\*',label='Origin')

plt.xlabel('Time')

plt.ylabel('Speed')

plt.title('对数拟合')

plt.text(0,0.50,'Loss=%.4f'%loss\_log,fontsize=12,c='b')

plt.legend()

plt.figure()

p1 = [1,2,3]

plsq1 = leastsq(residuals1,p1,args=(y,t))

print('幂指数函数参数：',plsq1[0]) # 幂指数函数拟合的参数

y\_pow = func2(t,plsq1[0])

loss\_pow = np.sum((y\_pow - y)\*\*2) # 最小二乘误差

# print(y\_pow)

plt.plot(t,y\_pow,'r',label='幂指数拟合')

plt.plot(t,y,'k\*',label='Origin')

plt.xlabel('Time')

plt.ylabel('Speed')

plt.title('幂指数拟合')

plt.text(0,0.55,'Loss=%.4f'%loss\_pow,fontsize=12,c='b')

plt.legend()

plt.figure()

plt.title('一元线性拟合Polyfit')

Loss1 = polyfit\_linear123(t,y,1)

plt.text(0,0.61,'Loss=%.4f'%Loss1,fontsize=12,c='b')

# plt.show() 如果show了 将会显示多张图 但都是figure1对象 需要×掉上一张才显示下一张

# 但可以最后show这样有多张figure 多个figure对象

plt.figure()

plt.title('二元线性拟合Polyfit')

Loss2 = polyfit\_linear123(t,y,2)

plt.text(0,0.55,'Loss=%.4f'%Loss2,fontsize=12,c='b')

plt.figure()

plt.title('三元线性拟合Polyfit')

Loss3 = polyfit\_linear123(t,y,3)

plt.text(0,0.55,'Loss=%.4f'%Loss2,fontsize=12,c='b')

#单独画出误差最小的拟合图像 并求200的加速度

plt.figure()

reg = np.polyfit(t,y,2) # 系数向量

polyf = np.polyval(reg,t) # 含拟合参数多项式的y值

plt.plot(t,polyf,'r')

plot1 = np.poly1d(reg)

dfx = plot1.deriv() #deriv()函数：求plot的导数 再次使用可以求二阶导

dfx\_200result = dfx(0.2)

text1 = plt.text(80,0.4,'加速度=%.4f'%dfx\_200result,fontsize=12,c='b')

plt.xlabel('Time')

plt.ylabel('Speed')

plt.title('二元线性拟合示例')

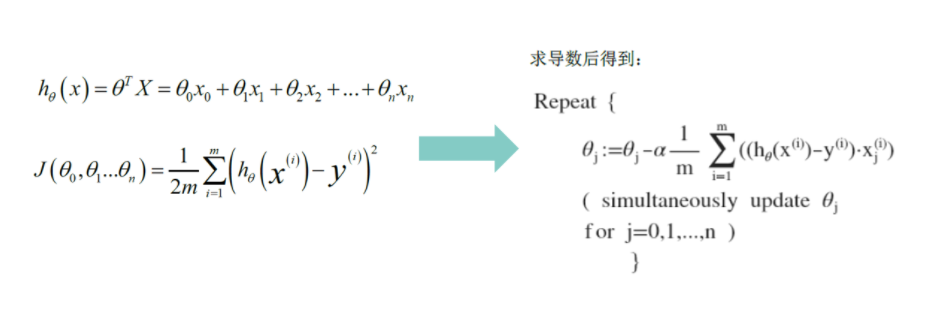
plt.show()

1. 代码如下。利用matplotlib包、numpy包以及random包。

这一问做首先对散点数据初始化，然后利用random包的random函数生成100个数据，其中x和y有一定联系。通过plt.scatter画出散点图。

初始化线性拟合函数参数，赋随机初值。在这里，设置学习率为0.25.需要注意的是，学习率的赋值会影响拟合过程：过小引起迭代次数过多，过大则可能在搜索过程中发生震荡。

利用梯度下降法求线性拟合中三个参数。公式如下：



利用求出来的参数拟合散点数据，在散点图上画出拟合曲线。需要注意的是，在最终plot绘制时，不是直接用散点X，而是自己均匀取点，否则最终画出来不是一条直线，而是很多条。

import random

import numpy as np

import matplotlib.pyplot as plt

# 生成随机一组数据

plt.figure()

x = np.zeros(100)

y = np.zeros(100)

for i in range(100):

    x[i] = random.random()

    y[i] = random.random() \* 0.2 - 0.1 + x[i] \* x[i] + x[i] + 1

plt.scatter(x,y,marker='\*',c='black')

# y=a0+a1\*x+a2\*x^2

# 给系数赋初值

a0 = random.random()

a1 = random.random()

a2 = random.random()

# 计算梯度

h = np.zeros(100)

A = 0.25 # 学习率

for j in range(1000):

    sum\_1,sum\_2,sum\_3 = 0,0,0

    for i in range(100):

        h[i] = a0 + a1 \* x[i] + a2 \* x[i] \* x[i]

        sum\_1 += h[i] - y[i]

        sum\_2 += (h[i] - y[i]) \* x[i]

        sum\_3 += (h[i] - y[i]) \* x[i] \* x[i]

    # 参数迭代

    a0 = a0 - A \* (sum\_1 / 100)

    a1 = a1 - A \* (sum\_2 / 100)

    a2 = a2 - A \* (sum\_3 / 100)

print(a0,a1,a2)

# for i in range(100):

#     h[i] = a0 + a1 \* x[i] + a2 \* x[i] \* x[i] 注意这里不能用散点的X 而应该是自己均匀取点

x0 = np.linspace(0,1,100)

h0 = a0 + a1 \* x0 + a2 \* x0 \* x0

plt.plot(x0,h0,'b')

plt.show()

1. 代码如下。利用matplotlib包和numpy包。

* 定义二分类字典，两种类别的花分别对应一种数字。
* 定义函数**getIndexDict(classDict)**将类别字典和key和value对调，s**igmoid函数**和**classifyVector(x, w)函数**实现一个样本根据特征值向量最终生成标量对类别的映射，**stocGradAscent(xMat, classLabels, numIter=150)函数**实现随机的梯度下降法。这里实现随机，是因为考虑到样本数量庞大，内存资源有限，于是每一次应用一条样本数据，对参数进行更新。
* 为了可测量分类器准确程度，设置了一个阈值，来区分测试集和训练集。在**test(fileName = '', numIter=150, trainRatio = 0.8)函数**中，默认比例为训练集:测试集为0.8。在这个函数中实现了对**.csv文件**数据的读取以及求得回归系数，并依靠系数预测测试集结果。
* 最终可求得测试结果的错误率。
* 为进一步对比迭代次数影响，设置不同上限值，调用test函数来进行训练分类器，比较错误率。
* 最终，将数据和回归曲线可视化。

import numpy as np

import matplotlib.pyplot as plt

# 进行二分类时的类字典

classDict = {'Iris-setosa': 0, 'Iris-versicolor': 1}

# 将类别字典的key和value对调

def getIndexDict(classDict):

    indexDict = {}

    for key, value in classDict.items():

        indexDict[value] = key

return indexDict

# sigmoid函数

def sigmoid(x):

    try:

        return 1.0/(1+np.exp(-x))

    except:

        return 0.0  #溢出时返回0.0

# 随机的梯度下降算法

def stocGradAscent(xMat, classLabels, numIter=150):

    '''

    Parameters:

        xMat: 输入样本矩阵

        classLabels:输入样本类别标签

        numIter:算法迭代次数

    '''

    m,n = np.shape(xMat) # m：样本个数，n：特征数 +1(类别数)

    w = np.ones(n) #初始化一个全1矩阵

    for j in range(numIter): #迭代

        dataIdx = list(range(m)) # 生成样本索引列表

        for i in range(m): #随机遍历一遍样本集

            alpha = 4/(1.0+j+i)+0.01 #步长，不断变化

            randIdx = int(np.random.uniform(0,len(dataIdx))) #随机生成一个样本索引

            #取出对应样本与w做线性运算，求和后通过sigmoid函数生成0-1之间的一个数字

            h = sigmoid(sum(xMat[dataIdx[randIdx]]\*w))

            err = -(classLabels[dataIdx[randIdx]] - h) #计算真实类别(0或1)与预测出数字的差异

            w = w - alpha\*err\*xMat[dataIdx[randIdx]] #更新权重

            del(dataIdx[randIdx]) #删除已访问过的索引

    return w

# 根据权重w对样本x做分类

def classifyVector(x, w):

    prob = sigmoid(sum(x\*w)) #线性模型后经过sigmoid函数值为0-1之间，视为概率

    if prob > 0.5:return 1.0 #概率＞0.5时为正类，否则为负类

    else:return 0.0

# 统计每个类别的个数，返回出现次数多的类别

def majorityCnt(classList):

    # 类别计数器

    classCount={}

    for c in classList:

        if c not in classCount.keys():

            classCount[c] = 0

        classCount[c] += 1

    # reverse = True 从大到小排列，key x[1]指比较key、value中的value

    sortedClassCount = sorted(classCount.items(),key=lambda x:x[1],reverse=True)

return sortedClassCount[0][0]

# 进行(多分类/)二分类测试

def test(fileName = '', numIter=150, trainRatio = 0.8):

    '''

    Parameters:

        fileName:数据集所在文件

        numIter:迭代次数

        trainRatio:训练集占数据集比例

    Returns:

        errRate:错误率

    '''

    allData = open(fileName)    #打开文件

    allSet = []                 #记录所有数据

    allLab = []                 #记录所有数据的类别

    lines = allData.readlines() #获取文件中所有内容

    head = lines[0].strip().split(',') #得到第一行

    data = lines[1:]            #所有数据

    numFeatures = len(head) - 1 #特征个数，减去的是类别那一列

    for line in data:           #遍历每个数据

        curLine = line.strip().split(',')

        lineArr = []            #记录处理后的每行数据

        for i in range(numFeatures):  # 遍历每个样本的特征

            lineArr.append(0 if curLine[i] == '?' else float(curLine[i]))  #缺失数据补为0

        allSet.append(lineArr)  #将处理后的样本加入样本全集中

        label = curLine[numFeatures] #当前样本类别

        allLab.append(classDict[label])# 加入标签，numFeatures为类别下标

    numExamples = len(data) #样本总数

    numTrain = int(numExamples \* trainRatio) #根据训练集的比例得出训练集总数

    numTest = numExamples - numTrain #测试集总数

    trainSet = np.array(allSet[:numTrain])  # 训练（数据 仅包含特征值部分/标签）,预测

    trainLab = np.array(allLab[:numTrain])

    testSet = np.array(allSet[numTrain:])

testLab = np.array(allLab[numTrain:])

    labels = set(trainLab) #去重样本标签集合

    trainW = [] #记录每个分类器的权重

    labelPairs = [] #记录每个分类器对应的0、1原类别

    for label1 in labels:        #对于二分类，由于labels中只有两个元素，只有一个分类器

        for label2 in labels:    #对于多分类，labels中有多个元素，两两元素之间有一个分类器

            if label1 >= label2: #只允许label1 < label2

                continue

            labelPairs.append([label1, label2]) #记录当前的两个类别

            curTrainLabels = [] #记录标签全集中是当前两个类别的子集标签

            curTrainSet = []    #记录样本全集中是当前两个类别的子集样本

            #遍历每个训练样本，如果属于当前选择的两个类别

            #则将该样本与该样本的标签记录下来

            for i in range(numTrain):

                if trainLab[i] == label1 or trainLab[i] == label2:

                    if trainLab[i] == label1: #属于二分类中的'0'类别

                        curTrainLabels.append(0)

                    if trainLab[i] == label2: #属于二分类中的'1'类别

                        curTrainLabels.append(1)

                    curTrainSet.append(trainSet[i])

            # 对当前分类器通过梯度求回归系数

            curTrainW = stocGradAscent(curTrainSet, curTrainLabels, numIter)

            trainW.append(curTrainW) #记录当前分类器的回归系数

    # ----------------------------- 预测样本集 ------------------------------

    predLab = [] #记录预测结果

    errCount = 0.0 #记录预测错误个数

    numClassifiers = len(labelPairs) #分类器的数目

    indexDict = getIndexDict(classDict) #将类别字典的key与value对调

    # #当执行二分类且类别字典中的类别大于2种时，定义负类的名字就为正类的名字前加个“非”字

    # if numClassifiers == 1 and len(classDict.items()) > 2:

    #     indexDict[0] = '非'+indexDict[1]

    for i in range(numTest):  #依次计算每个预测样本

        curPreLab = [] #当前预测类别

        for j in range(numClassifiers): #遍历每个分类器对预测样本进行预测

            curPartPreLab = classifyVector(testSet[i], trainW[j]) #预测类别

            curPartPreLab = labelPairs[j][int(curPartPreLab)] #将预测的0、1类别还原为原类别

            curPreLab.append(curPartPreLab) #记录当前分类器预测类别

        curPreLab = majorityCnt(curPreLab) #统计每个分类器的预测结果，找出预测次数最大的类别

        predLab.append(curPreLab) #记录当前样本预测类别

        print("分类预测类别为：%s, 真实类别为：%s"%(indexDict[curPreLab], indexDict[testLab[i]]))

        if curPreLab != testLab[i]: #如果预测类别!=实际类别，错误量+1

            errCount += 1.0

    errRate = float(errCount) / numTest #错误率

    print('错误率为：%f, 总测试集样本数为：%d,预测错误数为:%d' % (errRate, numTest, errCount))

return errRate,trainW

if \_\_name\_\_ == '\_\_main\_\_':

    minErrRate= 100 #记录最小错误率

    bestNumIter = 0 #记录最佳迭代次数

    for numIter in range(50, 200, 10): #试验不同迭代次数对算法的影响

        curErrRate = test(fileName = 'Iris\_01.csv', numIter = numIter, trainRatio = 0.67)

        if curErrRate[0] < minErrRate : #记录最小错误率以及最佳迭代次数

            minErrRate = curErrRate[0]

            bestNumIter = numIter

    print('最佳训练时迭代次数为%d次,最小错误率为%f'%(bestNumIter, minErrRate))

    print ('权重：',curErrRate[1])

    #画图

    plt.figure()

    allData = open('Iris\_01.csv')    #打开文件

    allSet = []                 #记录所有数据

    allLab = []                 #记录所有数据的类别

    lines = allData.readlines() #获取文件中所有内容

    head = lines[0].strip().split(',') #得到第一行

    data = lines[1:]            #所有数据

    numFeatures = len(head) - 1 #特征个数，减去的是类别那一列

    for line in data:           #遍历每个数据

        curLine = line.strip().split(',')

        lineArr = []            #记录处理后的每行数据

        for i in range(numFeatures):  # 遍历每个样本的特征

            lineArr.append(0 if curLine[i] == '?' else float(curLine[i]))  #缺失数据补为0

        allSet.append(lineArr)  #将处理后的样本加入样本全集中

        label = curLine[numFeatures] #当前样本类别

        allLab.append(classDict[label])# 加入标签，numFeatures为类别下标

    x = [i[0] for i in allSet] #取数据第一列

    y = [j[1] for j in allSet]

    # print(x)

    # print(y)

    scatter\_01 = plt.scatter(x[0:51],y[0:51],marker='\*',c='black')

    scatter\_02 = plt.scatter(x[51:],y[51:],marker='+',c='red')

    plt.legend(\*scatter\_01.legend\_elements(num=1),\*scatter\_02.legend\_elements(num=1),

                    labels=['Iris-setosa','Iris-versicolor'],loc='upper left',title='Class')

    x1 = np.linspace(0,6,256)

    y0 = curErrRate[1][0]

    y1 = (0.5 - y0[0] \* x1) / y0[1]

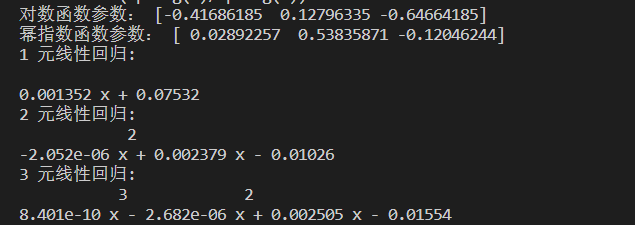
    plt.plot(x1,y1,'b')

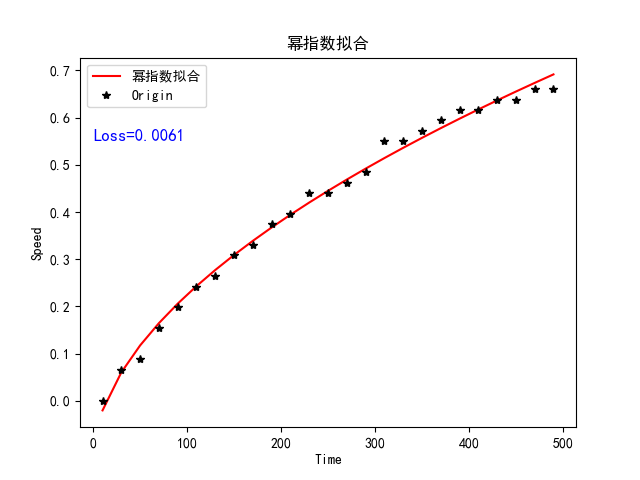
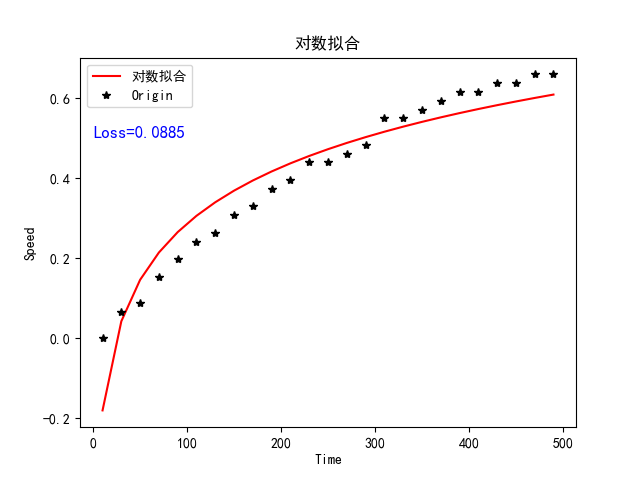
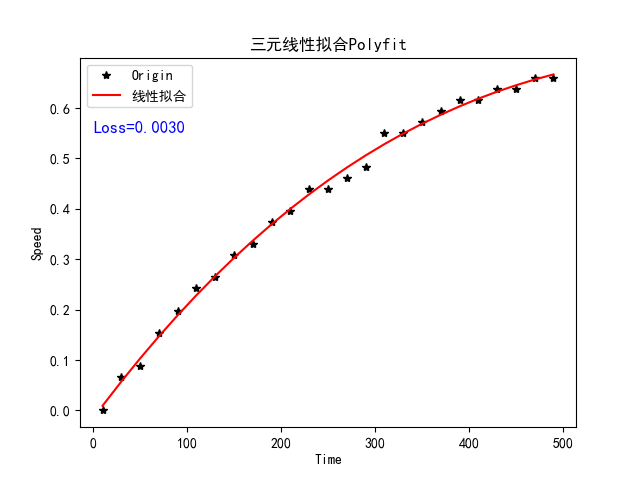
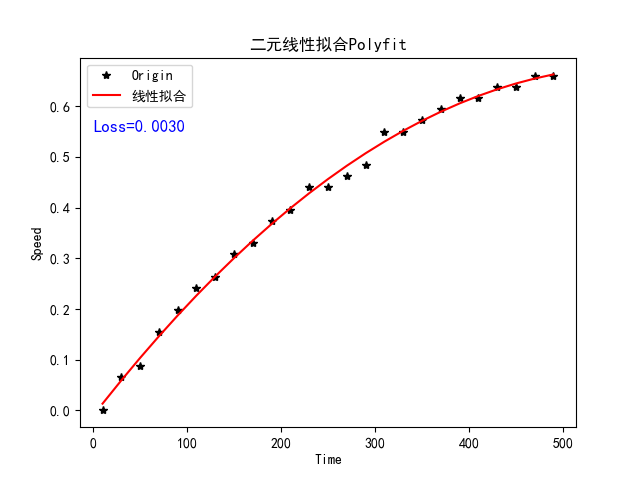
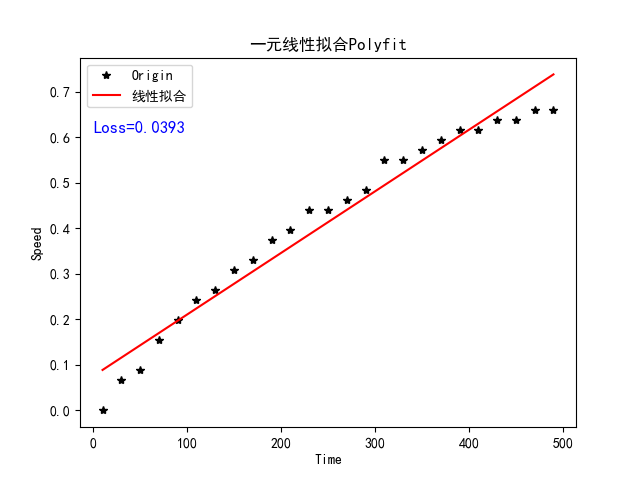
    plt.xlabel('petal length')

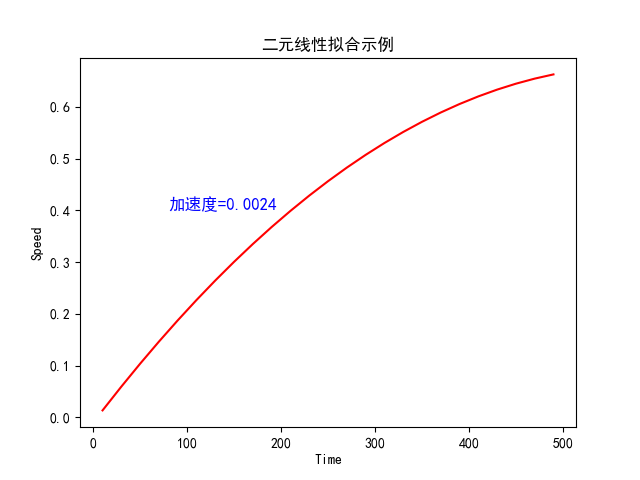
    plt.ylabel('petal width')

    plt.show()

1. **总结**
2. 结果：

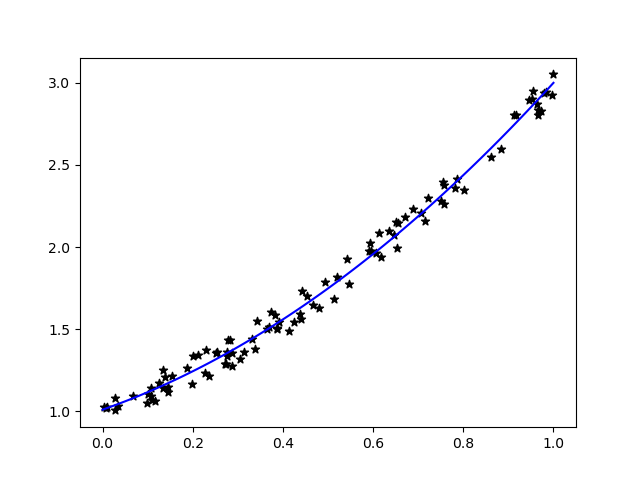




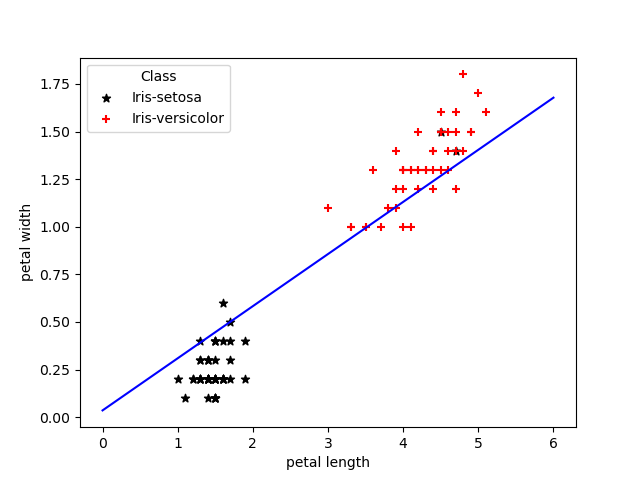


其中二元和三元线性拟合效果更好，根据奥卡姆剃刀原理，最终对图像进行拟合应用的函数是二元线性函数，最终得到加速度为0.0024米/秒²。

1. 结果：



3. 结果：

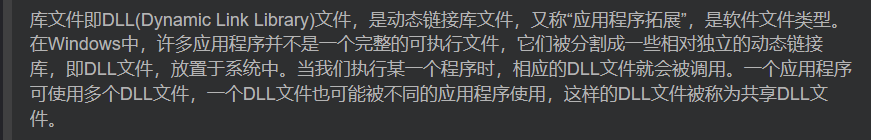




在拟合直线上部为Iris-versicolor的领域，下部为Iris=setosa的领域。

记录：

在本次实验中，耗费时间最久的并非解决基本的问题，而是环境的配置。在本机中，我使用annaconda管理python环境，但对于scipy包的导入总存在“ImportError DLL load failed”的错误。查找问题原因，是由于自己的这个包存在问题，运行时才报错。



于是找了大量解决问题的博客，将问题锁定在两种原因：1.函数库本身有问题，导致调用异常。2.依赖包版本不符，导致调用异常。

解决的方法有两种：1.重新安装该函数库。2.检查依赖包的版本（过高或过低）。

由于重新安装函数库对当时的我来说要在系统而非anaconda prompt中需要更改环境，而更改环境是我不愿意做的，于是选择了看起来简单但实际难度巨大的一个方法。下载了github上dependencies程序，希望应用该程序查看scipy dll文件是否缺失，若缺失了就进行下一步操作。而dependencies的运行需要依赖VC++环境，我并没有配置，但我有VSCode，于是进一步配置VC++环境。尝试3天后，以失败告终。最终决定重装函数库。

于是cmd进入终端，激活base环境，直接在C盘用conda语句卸载和重新安装了scipy包。途中，遇到权限不够的问题，这是因为在anaconda存储位置对User没有完全控制的权限，更改即可。

但最后下载完，发现相关matplotlib包变得不可导入，于是去查conda list是否有matplotlib包，果然没有，最后重新下载，程序可运行。一波三折。