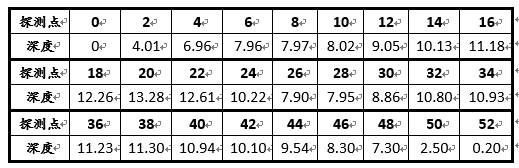
# 2019年 计算方法B 上机报告（python）

报告人：洪振鑫 班级：硕9060

学号：3119105202 联系电话：13759915205

## 第一题

1.**题目内容**

某通信公司在一次施工中，需要在水面宽度为50米左右的河沟底部沿直线走向铺设一条水底光缆。在铺设光缆之前需要对河底的地形进行初步探测，从而估计所需光缆的长度，为工程预算提供依据。假设光缆在铺设时可以比较紧密地贴合河床，并且已通过探测得到一组探测点位置的水的深度数据，相邻两个探测点相距2米，如下表所示：(单位:米)

(1)请用合适的曲线拟合所测数据点；

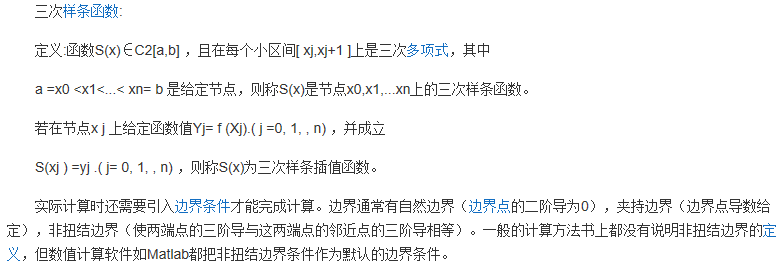
(2)估算所需光缆长度的近似值，精确到小数点后两位小数，并作出铺设河底光缆的曲线图；

**2.实现思想：**

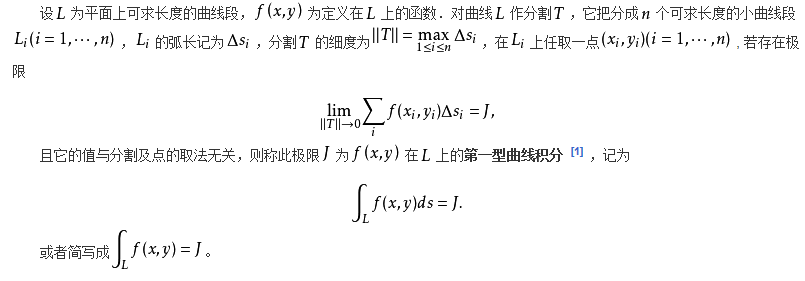
从题目中可以知道，题目需要我们对河底的地形进行一定的探测，并能够大致估算出需要的电缆长度。也就是说，我们需要通过探测出的离散数据点，对连续的河底地形进行一个估计，然后通过这个估计出来的地形，算出我们大致需要的线缆长度，将这个问题抽象化以后就是函数近似拟合以及曲线积分的问题。在确定的问题的类型之后，就是要选择解决问题的办法。高次多项式插值还是不同次数的分段插值。本题中给出了27个数据点，用多项式插值的话，不可避免的会出现龙格现象，从而将给预测带来极大的误差，这个方法实际上是没办法使用的，分段线性插值和分段2次插值的话，则是由于他们在数据点处不能100%保证光滑连续，故而也将给最终的预测带来较大的偏差故而不去选择，而这样分析之后，避免龙格现象，光滑的衔接，自然而然的就是分段三次插值的方法。在预测线缆的总长度的时候，我们则使用第一型曲线积分的方法，从而实现对曲线长度的最终拟合近似。

**3.算法依据**

三次样条插值：



第一型曲线积分：

****

**4.算法实现结构**

读取数据->计算h，l，u，Diag，和自然边界下的d->

编写追赶法函数，将前面求得的参数带入求解M(加入自然边界条件的值)->通过三次样条的通用表达式定义函数->取插值点带入函数求值绘制曲线->

同样基于插值点，以第一型曲线积分的概念，求出曲线的近似长度。

**5.实现代码、输入输出**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import scipy.interpolate as spi

# 读入图像数据

x = np.arange(0,54,2)

y = np.array([0,4.01,6.96,7.96,7.97,8.02,9.05,10.13,11.18,12.26,13.28,12.61,10.22,7.90,7.95,8.86,10.80,10.93,

             11.23,11.30,10.94,10.10,9.54,8.30,7.30,2.50,0.20])

y = -y

# 判断x，y的维度是否相等，如果不等就报出指定的data size错误

assert x.size == y.size, "the origin data is wrong. having different size"

#  in this question, we know that h == 2

# 矩阵维度计算

Dimension = x.size-2

# 由于本题的特殊性，dig=2，lamda=0.5=u,在此处就不需要考虑和原数字的index对位的问题了

# 后面还是需要考虑的

h = 2

Diag = 2\*np.ones(Dimension)

ld = 0.5\*np.ones(Dimension-1)

u3 = 0.5\*np.ones(Dimension-1)

# 通过差商求出自然边界条件下的d的各项值

d = np.zeros(Dimension)

for i in range(Dimension):

    d[i] = 0.75\*(y[i+2]-2\*y[i+1]+y[i])

    pass

# print(d)

# 追赶法函数

def Chasing(*a*,*b*,*c*,*d*,*show*=True):

    # 初始化系数矩阵

    l = np.zeros(a.size)

    u = np.zeros(b.size)

    y = np.ones(d.size)

    M = np.zeros(y.size)

    u[0] = b[0]

    y[0] = d[0]

    # 求解系数矩阵

    for i in range(l.size):

        l[i] = a[i]/u[i]

        u[i+1] = b[i+1]-l[i]\*c[i]

    # 求解ly=d

    for i in range(1,y.size):

        y[i] = d[i]-y[i-1]\*l[i-1]

    # 求解UM=y

    M[M.size-1] = y[y.size-1]/u[y.size-1]

    for i in range(M.size-2,-1,-1):

        M[i] = (y[i]-c[i]\*M[i+1])/u[i]

    # 控制参数显示

    if show==True:

        ShowAns(l,u,c,M)

    return M

# 结果展示函数

def ShowAns(*l*,*u*,*c*,*M*):

    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*l\*\*\*\*\*\*\*\*\*\*')

    print(l)

    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*u\*\*\*\*\*\*\*\*\*\*')

    print(u)

    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*c\*\*\*\*\*\*\*\*\*\*')

    print(c)

    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*M\*\*\*\*\*\*\*\*\*\*')

    print(M)

    pass

# 然后建立样条分段函数

# 并绘制图像

# 求出M，再添加自然边界条件（前后加0）

fake\_M = Chasing(u3,Diag,ld,d,False)

M = np.zeros(fake\_M.size+2)

M[1:M.size-1] = fake\_M

# print(M)

# 建立三次样条插值函数（分段）

def MSpline3(*data\_x*,*data\_y*,*M*,*h*,*x*):

    assert x >= 0, "the x value is undefind(wrong)"

    # 以免输入错误的数据

    index = x//2 + 1

    # 对应x的value和矩阵中次序的关系

    # 限制下标的范围（在边界最后一个值会有一个溢出，控制一下）

    index = *int*(index)

    if index==27: index=26

    # 根据M值建立三次样条插值的方程（带入方程表达式）

    V1 = pow(data\_x[index]-x,3)\*M[index-1]/12

    V2 = pow(x-data\_x[index-1],3)\*M[index]/12

    V3 = (data\_y[index-1]-4\*M[index-1]/6)\*(data\_x[index]-x)/2

    V4 = (data\_y[index]-4\*M[index]/6)\*(x-data\_x[index-1])/2

    return V1+V2+V3+V4

# 绘制三次样条插值后的图像

x\_plot = np.linspace(0,52,5000)

y\_plot = np.array([MSpline3(x,y,M,h,t) for t in x\_plot])

LineLens = 0

# Part2 求曲线长度，使用第一型线积分的策略

# 对曲线长度进行分段求和

for i in range(len(x\_plot)-1):

    temp= np.square(x\_plot[i+1]-x\_plot[i])+np.square(y\_plot[i+1]-y\_plot[i])

    V = np.sqrt(temp)

LineLens = LineLens + V

LineLens = round(LineLens, 2)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Lens\*\*\*\*\*\*\*\*\*\*')

print(LineLens)

# 规定横纵轴的单位长度一致，从而使得图像更容易比对和阅读

plt.axis('scaled')

plt.xlim((0,54))

plt.ylim((-15,0))

plt.plot(x\_plot,y\_plot)

# 描出数据点，设置点打大小和颜色

plt.scatter(x,y,10,'r')

# 中文显示问题解决

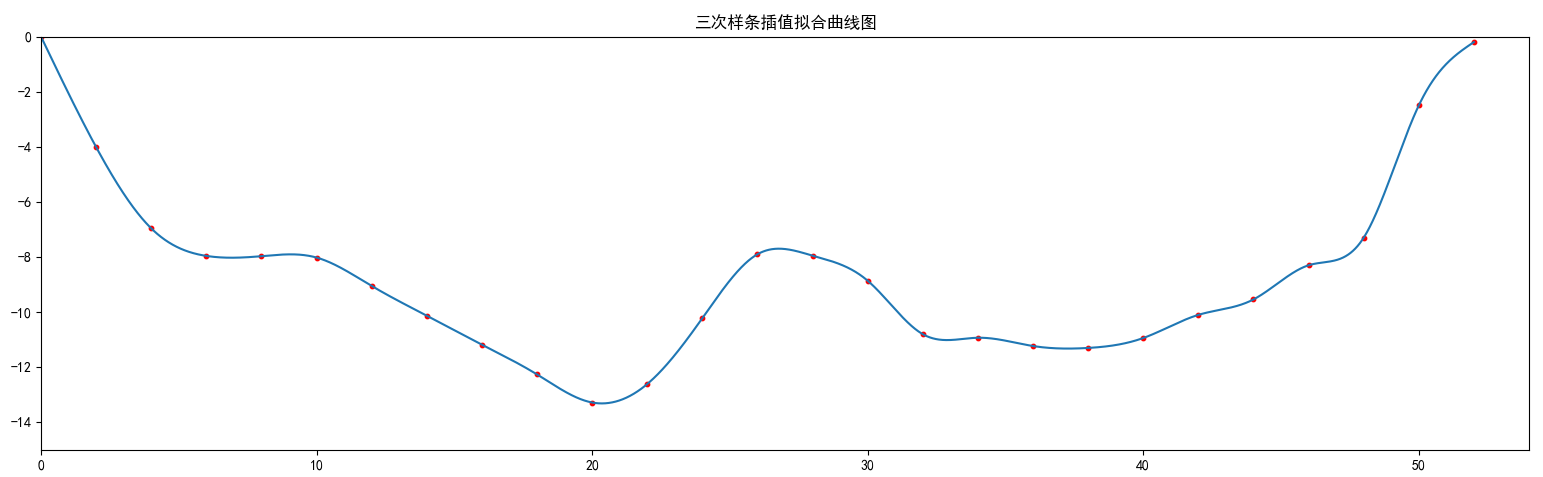
plt.rcParams['font.sans-serif']=['SimHei']

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

plt.title('三次样条插值拟合曲线图')

plt.show()

**实现结果：**

****

**长度预测：66.51**

**6.上机中出现的问题（过程总结）：**

在实验中我对多项式和分段线性和分段2次也做了一个测试，测试的最终结果确实出现了龙格现象和曲线不平滑连续的情况。在实验中也没有出现啥其他的问题，就是有一个长度2和矩阵存储index的转换稍微有点东西，以及在图像绘图过程中对坐标轴的拉伸来提高逼真度，其他的都比较按部就班了，没啥特别的。就是按照三次样条和曲线积分的原理对算法进行编写。

# 第二题

**1.题目内容：**

附录1给出了西安市2019年7-9月份每日的天气预报情况，包括每日的最高/最低温度、天气状况、风力风向情况，试使用最小二乘法计算这三个月每个月的平均温度，并说明哪个月最热。那么今年西安的夏天是酷热、挺热、热，还是凉爽一些？

**2.实现思想：**

通过最高温和最低温算出每天的平均温度，然后基于题干给的最小二乘法（分别试试1，2，3次）对曲线数值进行拟合，多项式拟合会龙格，分段在这个情况下也不合适。数值积分后求平均值，算出整个月的近似平均温度。

根据曲线和平均温度判断哪个月凉爽那个月热。

**3.算法依据：**

“最小二乘法”是对线性方程组，即方程个数比未知数更多的方程组，以[回归分析](https://zh.wikipedia.org/wiki/%E8%BF%B4%E6%AD%B8%E5%88%86%E6%9E%90)求得近似解的标准方法。在这整个解决方案中，最小二乘法演算为每一方程式的结果中，将残差平方和的总和最小化。

法方程法求解最小二乘法。

**4.算法结构：**

求出每日的平均平均温度，将三个月的存入三个矩阵中->

编写函数，用法方程的方法求解最小二乘法的各阶系数 ->

编写函数，基于各阶系数，表达最小二乘法的函数形式->

编写函数，基于系数，表达最小二乘法的各阶积分式->

插值、绘图、求平均。

**5.代码和结果：**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.linalg import solve

# 读取文件，计算每日的平均气温

io = r'temperature*.*xlsx' #注意将表格方在同个文件夹中，不然要写绝对地址

d = pd.read\_excel(io,*sheet\_name*='Sheet1',*usecols*=[1,2],*converters*={'max':*float*, 'min':*float*}) #读取文件

# 求出7，8，9月每天的平均温度，用于后续的曲线最小二乘拟合改写

Average = np.zeros([3,31])

for i in range(30):

    Average[0][i] = (d['max'][i] + d['min'][i])/2

    Average[1][i] = (d['max'][i+31] + d['min'][i+31])/2

    Average[2][i] = (d['max'][i+61] + d['min'][i+61])/2

Average[0][30] = (d['max'][30] + d['min'][30])/2

Average[2][30] = (d['max'][91] + d['min'][91])/2

print(Average)

# 最小2×拟合函数编写。1次，2次，3次

# 这个函数主要是求解出各项拟合函数的系数

def myl(*x*,*y*,*k*=1):

# k=1,y = a + bx,g=[1,x]

    if k == 1:

        g = np.ones([x.size,2])

        for i in range(x.size):

            g[i,1] = x[i]

# k=2 y=a+bx+cx\*x

    elif k == 2:

        g = np.ones([x.size,3])

        for i in range(x.size):

            g[i,1]=x[i]

            g[i,2]=np.square(x[i])

# k=3 y=a+bx+cx\*x+dx\*xxx

    elif k == 3:

        g = np.ones([x.size,4])

        for i in range(x.size):

            g[i,1]=x[i]

            g[i,2]=np.square(x[i])

            g[i,3]=pow(x[i],3)

# g\*g'a=g'y

    G = np.transpose(g)

    # 调用函数求解简单的线性方程

    y\_solve = np.dot(G,y)

    y\_solve = np.transpose(y\_solve)

    A = np.dot(G,g)

    para = solve(A,y\_solve)

#     print(para)

    return para

# 最小二乘法的数值积分，直接拿式子手撕就行了

# 手写出积分后的表达式

def interfunc(*x*,*para*,*k*):

    # 多项式的积分表达式，好些的禁

    if k == 1 :

        ans = para[0]\*x + para[1]\*x\*x\*0.5

    elif k == 2:

        ans = para[0]\*x + para[1]\*x\*x\*0.5 + para[2]\*x\*x\*x/3

    elif k == 3:

        ans = para[0]\*x + para[1]\*x\*x\*0.5 + para[2]\*x\*x\*x/3 + para[3]\*x\*x\*x\*x/4

    return ans

# 将各个月的数据输入进行拟合计算

# 通过k选择最小二乘法的次数（阶数）

k = 2

input\_x = np.arange(1,32)

para = myl(input\_x,Average[0][:],k)

def fucx(*x*,*para*,*k*):

    if k == 1 :

        ans = para[0]+para[1]\*x

    elif k == 2:

        ans = para[0]+para[1]\*x+para[2]\*x\*x

    elif k == 3:

        ans = para[0]+para[1]\*x+para[2]\*x\*x+para[3]\*x\*x\*x

    return ans

x\_plot = np.linspace(0,31,3000)

y\_plot = np.array([fucx(t,para,k) for t in x\_plot])

plt.axis('scaled')

plt.xlim((0,31))

plt.ylim((15,40))

plt.scatter(input\_x,Average[0][:])

plt.plot(x\_plot,y\_plot)

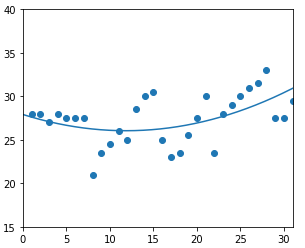
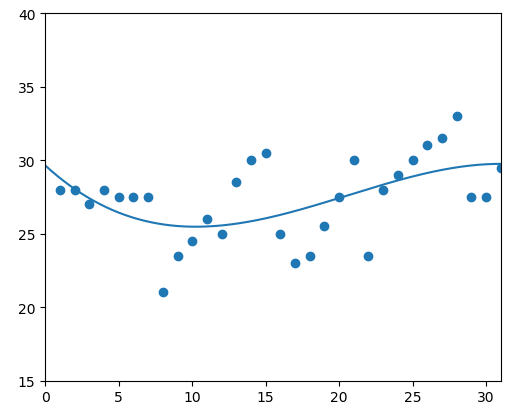
plt.show()

Ans = interfunc(31,para,k)-interfunc(1,para,k)

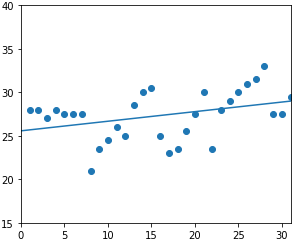
Ans = Ans/31

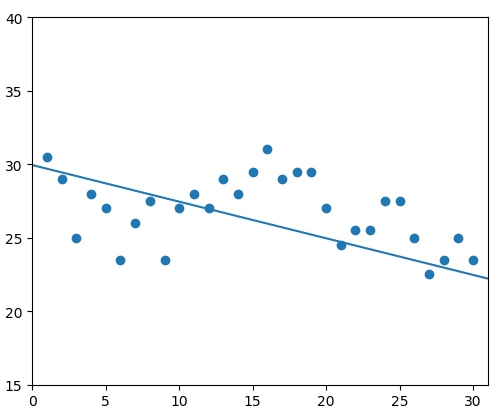
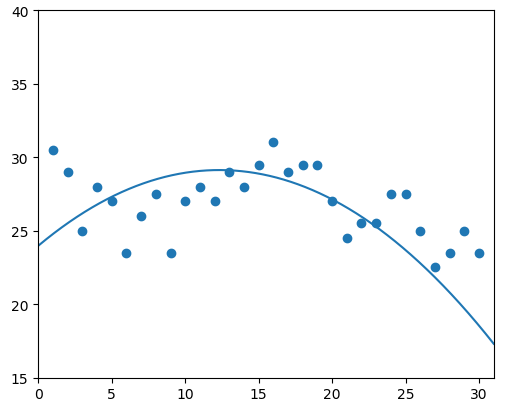
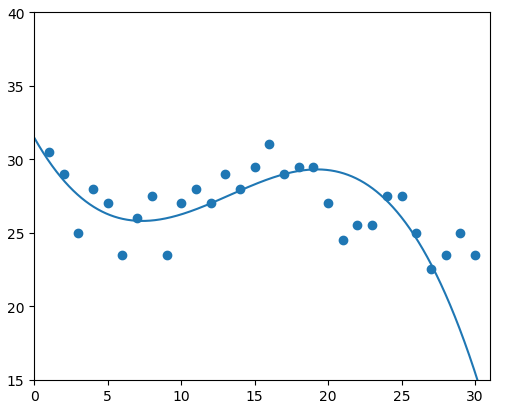
print(Ans)

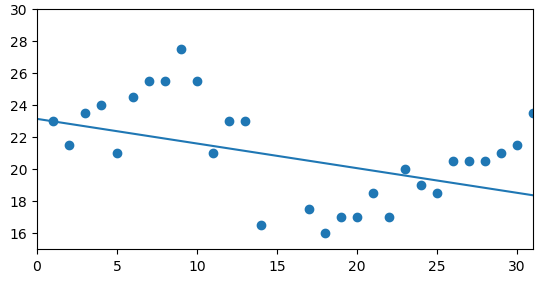
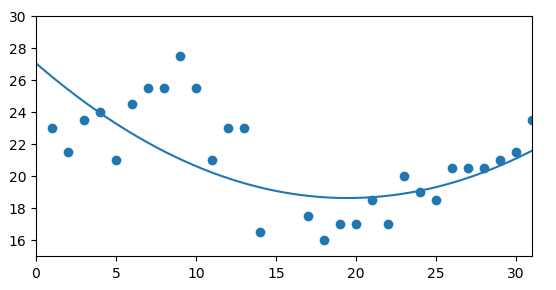
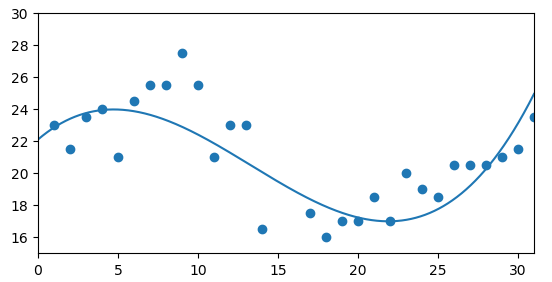
输出结果：

7月：k=1，2，3（图下方是平均温度）

26.4568、 26.3933、 26.3925

八月：k=1，2，3（图下方是平均温度）

25.1145 25.2786 25.2786

九月：k=1，2，3（图下方是平均温度）

19.9948 19.8872 19.8872

无论是从点分布还是函数拟合，还是最终的平均气温估计结果来看，今年最热的月份都是7月，而9月则是最凉爽的一个月。

同时同样，对以上的几个指标进行分析，我们可以知道，今年的西安相较于往年来说，可以说是比较凉爽的了。

**6.上机中出现的问题（过程总结）：**

感觉这样的多项式拟合的话，还是需要结合我们对气象的先验知识，才比较好选择哪一条曲线是更优的，如果单纯从单个月的数据点，以及曲线的拟合结果来看的话，很容易选择出错误的最优曲线，其实相对而言，如果要选择一个较为通用的拟合函数的话，在这题中应该是一次的最小二乘。（但是其实这样的结果并不是很让人满意）

在实验后，感觉其实可以将这样的最小二乘过程写成一个更为通用函数，但是由于刚写的时候，没有认真的规划，所以其实写出来的算法太过specific。还有改进的空间，其次，解最小二乘中，可能QR方法才是更好的选择，但是在尝试了法方程法，发现这样的问题好像只需要法方程法就已经足够了，就没有在编写。

# 第三题

**1.题目内容：**

在国产影片中，《战狼2》以56.39亿的票房收入稳居中国票房第一位。《流浪地球》最终只以46.19亿的票房屈居第三位。附录2给出了这两部电影的票房情况，试用插值法和数值积分进行以下的探讨：

（1）《流浪地球》作为国内第一部制作精良的科幻大片，在延长档期的情况下，最终票房收入也仅为46.19亿元，若想让它的票房突破50亿元，那么大概还需要多长时间才能达到这一目标。

（2）根据票房的增长数据，试判断一下，这两部片子哪一部更引人入胜；

**2.实现思想：**

要解决的问题：预测突破50e的时间，以及判断电影的受欢迎程度。针对前一个问题就是一个曲线拟合预测，函数拟合和数值积分的问题，和前面的分析一样，在这里尝试用最小二乘拟合和分段样条插值（观察曲线取趋势，可以的话将分段函数向区间外拓展），通过变上限的数值积分的方法，判断出什么时候能够突破50e。

而第二个问题，判断电影的吸引力，很大的一个因素就是口碑和票房的变化趋势，从趋势上可以看出这个电影到底是被人们所推崇还是只是宣发以及明星效应等等。

**3.算法依据：**

变上限积分、最小二乘法、三次样条插值、数值积分

**4.算法结构：（这里只写三次样条的方法，最小二乘大体一致）**

导入票房数据->

定义（前面已经写过）追赶法->

定义specific的三次样条计算需要的一些基本数据->

拟合函数，插值绘制曲线->

观察曲线结果发现最后区域平缓，故而延长最后的函数，用于预测->

编写变上限的数值积分函数，用while控制算法在50e跳出，以及设置上限。

**5.代码和结果：**

Version 三次样条：（只给出了预测的函数，绘图就是稍微修改一下输入，所以就没有另外贴出）

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import scipy.interpolate as spi

from scipy.linalg import solve

# 票房数据单位 万/周

weekZL\_x = np.arange(12)

weekZL\_y = np.array([99681.0, 217126.0, 140241.0, 56448.0, 26080.0, 17194.0, 5564.0,

                    2289.0, 1197.0, 619.0, 1029.0, 210.0])

assert weekZL\_x.size == weekZL\_y.size, "the size of ZL data is wrong"

weekLL\_x = np.arange(14)

weekLL\_y = np.array([0,202074.0, 177540.0, 56644.0, 17872.0, 6617.0, 2514.0, 1100.0,

                    462.0, 289.0, 127.0, 150.0, 41.0, 47.0])

assert weekLL\_x.size == weekLL\_y.size, "the size of LL data is wrong"

def Chasing(*a*,*b*,*c*,*d*,*show*=True):

    # 初始化系数矩阵

    # # 仅支持上下宽为1的追赶法，后续可能需要修改一下这个方法。

    l = np.zeros(a.size)

    u = np.zeros(b.size)

    y = np.ones(d.size)

    M = np.zeros(y.size)

    u[0] = b[0]

    y[0] = d[0]

    # 求出系数矩阵

    for i in range(l.size):

        l[i] = a[i]/u[i]

        u[i+1] = b[i+1]-l[i]\*c[i]

    # 求解Ly=d

    for i in range(1,y.size):

        y[i] = d[i]-y[i-1]\*l[i-1]

    # 求解UM=y

    M[M.size-1] = y[y.size-1]/u[y.size-1]

    for i in range(M.size-2,-1,-1):

        M[i] = (y[i]-c[i]\*M[i+1])/u[i]

    # 控制参数显示

    if show==True:

        ShowAns(l,u,c,M)

    return M

# 结果展示函数

def ShowAns(*l*,*u*,*c*,*M*):

    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*l\*\*\*\*\*\*\*\*\*\*')

    print(l)

    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*u\*\*\*\*\*\*\*\*\*\*')

    print(u)

    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*c\*\*\*\*\*\*\*\*\*\*')

    print(c)

    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*M\*\*\*\*\*\*\*\*\*\*')

    print(M)

    pass

# 三次样条插值part

# 特定问题的三次样条系数设置

Dimension = weekLL\_x.size -2

h = 1

Diag = 2\*np.ones(Dimension)

ld = 0.5\*np.ones(Dimension-1)

u3 = 0.5\*np.ones(Dimension-1)

# 通过差商求出自然边界条件下的d的各项值

# 这一部分和上面那一部分讲道理可以改写成函数的。

d = np.zeros(Dimension)

for i in range(Dimension):

    d[i] = 3\*(weekLL\_y[i+2]-2\*weekLL\_y[i+1]+weekLL\_y[i])

    pass

# print(d)

# 然后建立样条分段函数

# 并绘制图像

# 求出M

fake\_M = Chasing(u3,Diag,ld,d,False)

M = np.zeros(fake\_M.size+2)

M[1:M.size-1] = fake\_M

# 建立三次样条插值函数（分段）

def MSpline3(*data\_x*,*data\_y*,*M*,*h*,*x*):

    assert x >= 0, "the x value is undefind(wrong)"

    index = x + 1

    index = *int*(index)

    if index>=13: index=13

    V1 = pow(data\_x[index]-x,3)\*M[index-1]/6

    V2 = pow(x-data\_x[index-1],3)\*M[index]/6

    V3 = (data\_y[index-1]-M[index-1]/6)\*(data\_x[index]-x)

    V4 = (data\_y[index]-M[index]/6)\*(x-data\_x[index-1])

    return V1+V2+V3+V4

# 绘制三次样条插值后的图像

x\_plot = np.linspace(0,13,1000)

y\_plot = np.array([MSpline3(weekLL\_x,weekLL\_y,M,h,t) for t in x\_plot])

LineLens = 0

plt.plot(x\_plot,y\_plot)

plt.scatter(weekLL\_x,weekLL\_y)

plt.show()

# 对三次样条插值函数分析后发现最后的曲线趋于平缓，

# 故延长曲线，作为再利用数值积分的方法，来控制计算到达550e的时间

def my3ChIntegral(*func*,*x\_plot*,*threshold*=500000):

    Ans = 0

    h = x\_plot[1]-x\_plot[0]

    # 计算出插值点之间的步长

    # 通过数值积分方法来计算近似总票房结果

    for i in range(x\_plot.size-1):

        Value1 = func(weekLL\_x,weekLL\_y,M,h,x\_plot[i]) + func(weekLL\_x,weekLL\_y,M,h,x\_plot[i+1])

        temp = (x\_plot[i]+x\_plot[i+1])/2

        Value2 = func(weekLL\_x,weekLL\_y,M,h,temp)

        Ans = Ans + (Value1 + 4\*Value2)\*h/6

    print(Ans)

    # 延长上映时间，并进行数值累计，进行到达50e票房的时间预测，

    index = x\_plot.size -1

    x\_1 = x\_plot[index]

    x\_2 = x\_plot[index] + h

    times = 0

    while(Ans<threshold):

        Value1 = func(weekLL\_x,weekLL\_y,M,h,x\_1) + func(weekLL\_x,weekLL\_y,M,h,x\_2)

        temp = (x\_1+x\_2)/2

        Value2 = func(weekLL\_x,weekLL\_y,M,h,temp)

        Ans = Ans + (Value1 + 4\*Value2)\*h/6

        # 通过x来控制上映时间的向前延伸，也能就算出来后面每周的票房，从而实现累计的预测

        x\_1 = x\_2

        x\_2 = x\_2 + h

        # 通过times 控制算法跳出，避免出现永远到达不了50e算法不结束的情况发生

        times = times+1

        if times > 5000:

            print(x\_2)

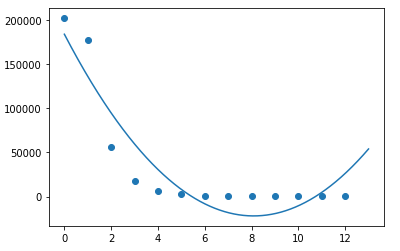
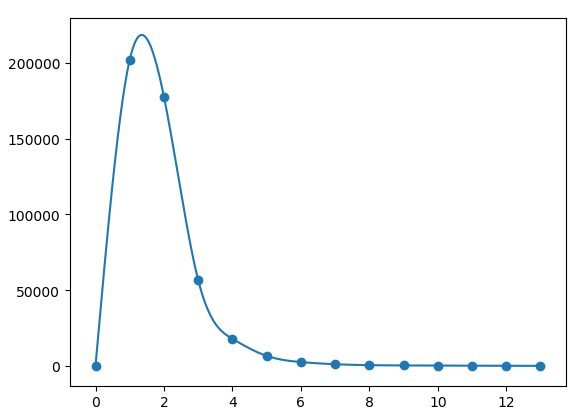
            break

    print(x\_2)

    return x\_2

Ans = my3ChIntegral(MSpline3,x\_plot)

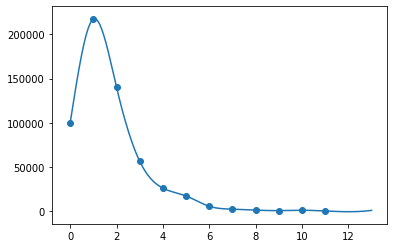
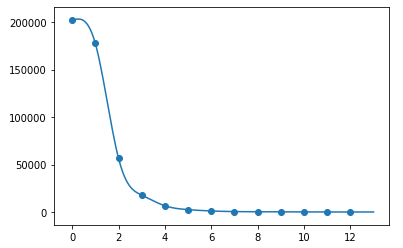
结果展示：1）流浪，总共需要78周到达50e，也就是还需要55周。（拟合的时候在数据前面加了（0，0），为了更平滑的计算出总票房的预测，）

 三次样条； 最小2乘（1，2，3）都及其不靠谱

如下图，就没有继续使用

以下是没有添加零点，对其进行曲线拟合（判断趋势）

战狼 流浪地球

2）从票房趋势来看的话，战狼基于良好的口碑所以，在上映一段时间内，观影票房是逐步上升才达到一个峰值，而流浪地球则是上映即巅峰，说明流浪地球的声势大于他的实力，而战狼则是更受观众欢迎和喜欢，所以我认为从数据来看的话，还是战狼更加引人入胜一点。

**6.上机中出现的问题（过程总结）**

在这一题中运用到的这些方法其实都是在之前两问中编写过的，也就是一个重新整合再带入的过程而已，只是其中由于票房数据的特殊性，所以选择再进行数值积分的时候，在数据之前加入一个（0，0），让其积分曲线更平滑合理。这题主要的问题就是对于数据的理解把，还有就是先对数据进行分析观察，是一个比较重要的步骤，在分析之前茫然动手的话，其实并不是一个最有效率的解决问题的方式。

# 第四题

**1.题目内容：**

线性方程组求解方法一般是以高斯消去法为主，也可以使用列主元高斯消去法。当方程组中的系数矩阵是严格对角占优矩阵时，直接使用高斯消去法就可以得到比较准确的方程组的解。

通常可以将方程组的系数矩阵和右端常量一起存储在数据文件中。在进行求解时，从文件中读出系数矩阵和右端常量，然后使用高斯消去法进行求解。该数据文件为二进制文件，其存储结构见附件2。

数据文件共有4个，其说明如下表所示，并且各数据文件中的系数矩阵均为严格对角占优的带状矩阵：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **数据文件** | **规模** | **类型** | **根** | **说明** |
| 1 | data20191.dat | 10阶 | 非压缩格式 | 1 | 用于测试程序 |
| 2 | data20192.dat | 20阶 | 压缩格式 | 1 | 用于测试程序 |
| 3 | data20193.dat | 3000阶左右 | 非压缩格式 | 待解 |  |
| 4 | data20194.dat | 50000阶左右 | 压缩格式 | 待解 |  |

对于压缩格式的方程组，在进行求解时可以只对带状区域中的元素进行处理，这样可以大大减少计算量。

（1）请试编写一个统一的程序，实现从4个数据文件中读入方程组的数据，再使用高斯消去法进行求解。

（2）针对本专业中所碰到的实际问题，提炼一个使用方程组进行求解的例子，并对求解过程进行分析、求解。

**2.实现思想：**

对于大规模带状矩阵，减少计算量是关键，减少计算量一个在结构的存储，这个是文件给定的，也就没有必要去考虑，唯一可能需要考虑的是是否需要逐行读取之类的。第二点就在计算上了，对带状矩阵的计算，每次消去的过程中只需要对下带宽行进行运算即可，同时处理也只需要上带宽列，也就是近似只需要对一个p\*q的矩阵进行消去分析就行，这样的话就能极大的降低计算需求和运算过程中的临时存储。（由于题目中带状矩阵的优越：严格对角占优）就考虑简单的高斯消去法就ok了。

**3.算法依据：**

高斯消去法，带状矩阵的分析，数据存储格式，

**4.算法结构：（这里只写三次样条的方法，最小二乘大体一致）**

根据数据的存储类型（int、float），分别读取出header，和data->

将压缩格式数据的存储格式修改变成矩阵中添的零全在右侧->

根据我修改的格式，定义offset 1\*n，能算出压缩格式的每行与实际坐标的偏差->

编写带状矩阵的高斯消去法，（只对p\*q处理），在每次运算过程中通过offset转换成真实坐标，就可以写一个统一的消去法函数->

带入数据求得结果->输出

**5.代码和结果：**

'''

通过对矩阵的最终结果的分析，我认为最后的结果应该是一个稳定值，但是由于高斯消去法的舍去误差

'''

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import struct

# import scipy.linalg

# 文件读取，可以通过这里选择文件读取

# 对所有的文件都可以进行处理，改变一个数字即可

# 解析文件头，并输出

filename = 'data20194.dat'

Headinfo = np.fromfile(filename,*dtype*=np.int32)

head = Headinfo[0:5].astype(np.int32)

print(head)

print('``````````````````````head↑```````````````````````````')

# 获取文件头以及一些基本的数据参量

Version = head[1]

n = head[2]

p = head[3]

q = head[4]

offset =[]

# 针对带状矩阵的压缩存储格式，给出了（i，j）中不同i

# 所会带来的矩阵横向坐标上的偏差

for i in range(n):

    if i>q:offset.append(i-q)

    else: offset.append(0)

offset = np.array(offset)

# 读取矩阵数据

data = np.fromfile(filename,*dtype*=np.float32)

# 针对不同格式的矩阵进行不同类型的读取

# 压缩格式的带状矩阵的0值在后续消去的时候进行判断后填充

if Version == 258:

    start = 5

    end = n\*n+5

    A = data[start:end].astype(np.float32)

    b = data[end:end+n].astype(np.float32)

    A = A.reshape((n,n))

    pass

elif Version == 514:

    start = 5

    end = n\*(p+q+1)+5

    A = data[start:end].astype(np.float32)

    b = data[end:end+n].astype(np.float32)

    num = p+q+1

    A = A.reshape((n,num))

    # 对压缩型的带状矩阵进行重构，重组其压缩形式，将所有需要填充的0都放在右侧

    # 以此用来匹配offset的坐标偏差

    for i in range(p):

        index = -1

        # 找到第一个非0 元所在的index

        for j in range(num):

            if A[i][j] !=0 and index == -1:

                index = j

        # 重构矩阵进行替换

        if index != -1:

            for k in range(num-index):

                A[i][k] = A[i][k+index]

            A[i][num-index:num] = 0

print(A)

print('````````````````````reshpae A↑`````````````````````````````')

print(b)

print('````````````````````reshape b↑`````````````````````````````')

# 接下来编写高斯消去法（带状矩阵专用）

# 消去的过程中只需要有限的取值就行

def GaussEli(*A*,*b*,*Version*,*n*,*q*,*p*,*offset*):

    if Version == 258: offset[:]=0

    '''高斯消去法，编写上没有什么需要注意的地方，就是普通的逐行进行消去叭了

    通过我们对压缩格式的重组，我们可以将非压缩格式的坐标也变换成普通格式，所以不需要重复编写

    但是要注意到的是，对于带状矩阵，需要计算的只有两个方向上的p和q个单位长度就行了，

    由于带外元素都是0，多余的计算是没必要进行的'''

    for index in range(n):

        for i in range(1,q+1):

            if index+i<n:

                # 先求出l

                l = A[index+i,index-offset[index+i]]/A[index,index-offset[index]]

                for j in range(p+1):

                    if index+j<n:

                        # 逐行进行消去化简

                        A[index+i,index+j-offset[index+i]] -= A[index,index+j-offset[index]]\*l

                        # 对右侧参数也进行变换

                b[index+i] -= b[index]\*l

    # 上面得到了A，b消去以后的矩阵，后面通过回带求解x[]

    x = np.zeros(n)

    # 回代过程的编写

    # 统一的形式 b-（a？\*x？）/aii，然后通过反向计算即可

    # 由于初始化的x value是0，所以用同一的形式反向回代就可以了

    # 需要注意的是在编程的过程中不要让矩阵的index越界就可以了

    for i in range(n-1,-1,-1):

        sumA = 0

        for j in range(p):

            index = i+j

            if index<n: sumA += A[i,index-offset[i]]\*x[index]

        for j in range(q):

            index = i-j

            if index>0: sumA += A[i,index-offset[i]]\*x[index]

        x[i] = round((b[i]-sumA)/A[i,i-offset[i]],5)

    # 输出结果

    print(x)

    print('`````````````````````result↑````````````````````````````')

    print(A)

    print('`````````````````````gauss a↑````````````````````````````')

    print(b)

    print('`````````````````````gauss b↑````````````````````````````')

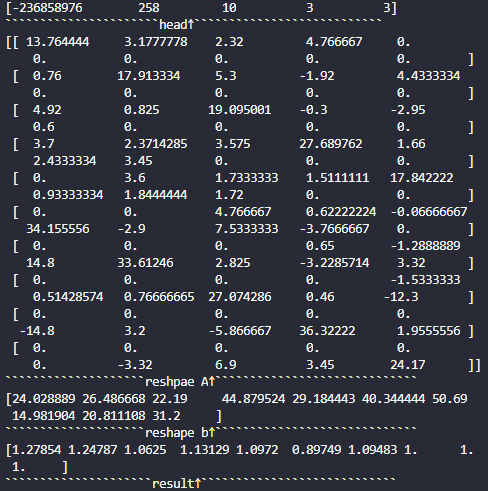
    print(offset)

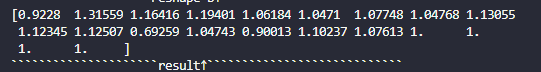
    print('``````````````````````offset↑```````````````````````````')

    pass

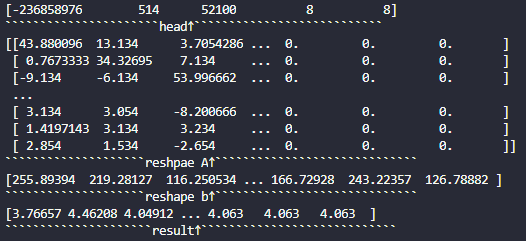
GaussEli(A,b,Version,n,q,p,offset)

结果显示：（结果在最后由于舍入误差的传播，从后往前逐渐偏移，但是可以看出准确解）（可能与python的计算精度有关，这题其实应该用matlab做结果可能会更好一些）

 Data1：准确解：1

Data2：准确解：1

Data3：准确解：2.019

 Data4：准确解：4.063

**6. 上机中出现的问题（过程总结）：**

数据读取的过程中一开始采取了其他方法，比较不好用，后来转投了老师给出的参考操作，舒服了很多。第二点就是在计算真实坐标，用offset对应真实坐标计算的时候，编写统一的处理Gauss消去的时候，矩阵的对应，比较纠结一些。然后就是最后结果可以看到，最后几个维度的答案都是正确的，但是随着往回代的过程，误差逐渐的大了起来，不知道是不是python计算精度的问题，因为严格对角占优的高斯消去法，理应是稳定算法，能控制误差才对，对L的验证，也确实在过程中都是远小于1的，设置断点验证计算过程的时候也没有问题。所以可能这种对精度要求更高的算法的话，用matlab之类的偏数学的工具会好一些。