## 层次分析法

import numpy as np

from numpy import linalg

def s\_s\_a(n, C):

"""算术平均值求权重"""

w\_1 = sum(C) / n

print("算术平均权重为：\n", w\_1)

print("")

def j\_h\_a(n, A):

"""几何平均值求权重"""

B\_2 = pow(np.prod(A, 1), 1/n) # 0是按行，1是按列

w\_2 = B\_2 / sum(B\_2)

print("几何平均值权重：\n", w\_2)

print("")

def t\_v(V,eig\_max,D):

"""特征值法求权重"""

c = np.where(V == eig\_max)

B\_3 = D[:, c.index(0)]

w\_3 = B\_3 / sum(B\_3)

print("特征值法求权重：\n", w\_3)

def main(A,B\_z,C\_z,D\_z,R\_I,n):

"""主程序"""

if D\_z[1,:].all() == (np.array(B\_z)\*np.array(C\_z)).all():

print("该矩阵是正互反矩阵！\n")

V,D = np.linalg.eig(A)

eig\_max = max(V)

CI = (eig\_max-n) / (n-1)

RI = R\_I[n-1]

CR = CI / RI

if CR < 0.1:

print("一致性检验通过！\n")

B\_1 = np.zeros([n,n])

for i in range(0,n):

B\_1[0:n,i] = A[0:n,i] / sum(A[0:n,i])

C = B\_1.T

s\_s\_a(n, C)

j\_h\_a(n, A)

t\_v(V,eig\_max,D)

else:

print("一致性检验失败，请修改！")

else:

print("您输入的矩阵非正互反矩阵")

R\_I = np.array([0, 0 , 0.52, 0.89, 1.12, 1.26, 1.36, 1.41,

1.46, 1.49, 1.52, 1.54, 1.56, 1.58, 1.59])

A = np.array([[1, 1, 4, 1/3, 3],

[1, 1, 4, 1/3 ,3],

[1/4, 1/4, 1, 1/3, 1/2,],

[3, 3, 3, 1, 3],

[1/3, 1/3, 2, 1/3, 1]])

n = len(A)

B\_z = []

C\_z = []

for i in range(0,n):

for j in range(0,n):

B\_z.append(A[i,j])

C\_z.append(A[j,i])

D\_z = np.ones([len(B\_z), len(B\_z)])

main(A,B\_z,C\_z,D\_z,R\_I,n)

该代码使用的时候只需要改变A矩阵中的数字即可 别的都不需要更改

## Topsis

import numpy as np # 导入numpy包并将其命名为np

##定义正向化的函数

def positivization(x,type,i):

# x：需要正向化处理的指标对应的原始向量

# typ：指标类型（1：极小型，2：中间型，3：区间型）

# i：正在处理的是原始矩阵的哪一列

if type == 1: #极小型

print("第",i,"列是极小型，正向化中...")

posit\_x = x.max(0)-x

print("第",i,"列极小型处理完成")

print("--------------------------分隔--------------------------")

return posit\_x

elif type == 2: #中间型

print("第",i,"列是中间型")

best = int(input("请输入最佳值："))

m = (abs(x-best)).max()

posit\_x = 1-abs(x-best)/m

print("第",i,"列中间型处理完成")

print("--------------------------分隔--------------------------")

return posit\_x

elif type == 3: #区间型

print("第",i,"列是区间型")

a,b = [int(l) for l in input("按顺序输入最佳区间的左右界，并用逗号隔开：").split(",")]

m = (np.append(a-x.min(),x.max()-b)).max()

x\_row = x.shape[0] #获取x的行数

posit\_x = np.zeros((x\_row,1),dtype=float)

for r in range(x\_row):

if x[r] < a:

posit\_x[r] = 1-(a-x[r])/m

elif x[r] > b:

posit\_x[r] = 1-(x[r]-b)/m

else:

posit\_x[r] = 1

print("第",i,"列区间型处理完成")

print("--------------------------分隔--------------------------")

return posit\_x.reshape(x\_row)

## 第一步：从外部导入数据

#注：保证表格不包含除数字以外的内容

x\_mat = np.loadtxt('river.csv', encoding='UTF-8-sig', delimiter=',') # 推荐使用csv格式文件

## 第二步：判断是否需要正向化

n, m = x\_mat.shape

print("共有", n, "个评价对象", m, "个评价指标")

judge = int(input("指标是否需要正向化处理，需要请输入1，不需要则输入0："))

if judge == 1:

position = np.array([int(i) for i in input("请输入需要正向化处理的指标所在的列，例如第1、3、4列需要处理，则输入1,3,4").split(',')])

position = position-1

typ = np.array([int(j) for j in input("请按照顺序输入这些列的指标类型（1：极小型，2：中间型，3：区间型）格式同上").split(',')])

for k in range(position.shape[0]):

x\_mat[:, position[k]] = positivization(x\_mat[:, position[k]], typ[k], position[k])

print("正向化后的矩阵：", x\_mat)

## 第三步：对正向化后的矩阵进行标准化

tep\_x1 = (x\_mat \* x\_mat).sum(axis=0) # 每个元素平方后按列相加

tep\_x2 = np.tile(tep\_x1, (n, 1)) # 将矩阵tep\_x1平铺n行

Z = x\_mat / ((tep\_x2) \*\* 0.5) # Z为标准化矩阵

print("标准化后的矩阵为：", Z)

## 第四步：计算与最大值和最小值的距离，并算出得分

tep\_max = Z.max(0) # 得到Z中每列的最大值

tep\_min = Z.min(0) # 每列的最小值

tep\_a = Z - np.tile(tep\_max, (n, 1)) # 将tep\_max向下平铺n行,并与Z中的每个对应元素做差

tep\_i = Z - np.tile(tep\_min, (n, 1)) # 将tep\_max向下平铺n行，并与Z中的每个对应元素做差

D\_P = ((tep\_a \*\* 2).sum(axis=1)) \*\* 0.5 # D+与最大值的距离向量

D\_N = ((tep\_i \*\* 2).sum(axis=1)) \*\* 0.5

S = D\_N / (D\_P + D\_N) # 未归一化的得分

std\_S = S / S.sum(axis=0)

sorted\_S = np.sort(std\_S, axis=0)

print(std\_S) # 打印标准化后的得分

## std\_S.to\_csv(std\_S.csv) 结果输出到std\_S.csv文件

## 插值法

### 牛顿插值法

import matplotlib.pyplot as plt

from pylab import mpl

import math

"""

牛顿插值法

插值的函数表为

xi 0.4， 0.55， 0.65， 0.80， 0.90， 1.05

f(xi) 0.41075, 0.57815, 0.69675, 0.88811, 1.02652, 1.25382

"""

x = [0.4, 0.55, 0.65, 0.80, 0.90, 1.05]

y = [0.41075, 0.57815, 0.69675, 0.88811, 1.02652, 1.25382]

"""计算五次差商的值"""

def five\_order\_difference\_quotient(x, y):

# i记录计算差商的次数，这里循环5次，计算5次差商。

i = 0

quotient = [0, 0, 0, 0, 0, 0]

while i < 5:

j = 5

while j > i:

if i == 0:

quotient[j]=((y[j]-y[j-1])/(x[j]-x[j-1]))

else:

quotient[j] = (quotient[j]-quotient[j-1])/(x[j]-x[j-1-i])

j -= 1

i += 1

return quotient;

def function(data):

return x[0]+parameters[1]\*(data-0.4)+parameters[2]\*(data-0.4)\*(data-0.55)+\

parameters[3]\*(data-0.4)\*(data-0.55)\*(data-0.65)\

+parameters[4]\*(data-0.4)\*(data-0.55)\*(data-0.80)

"""计算插值多项式的值和相应的误差"""

def calculate\_data(x,parameters):

returnData=[];

for data in x:

returnData.append(function(data))

return returnData

"""画函数的图像

newData为曲线拟合后的曲线

"""

def draw(newData):

plt.scatter(x,y,label="离散数据",color="red")

plt.plot(x,newData,label="牛顿插值拟合曲线",color="black")

plt.scatter(0.596,function(0.596),label="预测函数点",color="blue")

plt.title("牛顿插值法")

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

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

plt.legend(loc="upper left")

plt.show()

parameters=five\_order\_difference\_quotient(x,y)

yuanzu=calculate\_data(x,parameters)

draw(yuanzu)

### 拉格朗日插值法

import matplotlib.pyplot as plt

from pylab import mpl

"""

用拉格朗日插值法拟合数据。

"""

"""

功能：计算插值多项式的系数。

参数：data\_x为数据的x坐标，data\_y为数据的y坐标，size为插值基函数的个数。

返回值：插值函数的系数。

"""

x = [100, 121, 144]

y = [10,11, 12]

def ParametersOfLagrangeInterpolation(data\_x,data\_y,size):

parameters=[]

#i用来控制参数的个数

i=0;

while i < size:

#j用来控制循环的变量做累乘

j = 0;

temp = 1;

while j < size:

if(i != j):

temp\*=data\_x[i]-data\_x[j]

j+=1;

parameters.append(data\_y[i]/temp)

i += 1;

return parameters

"""

功能：计算拉格朗日插值法公式的值。

参数：data\_x为原始数据的横坐标，x为要用拉格朗日插值函数计算数据，parameters为拉格朗日插值函数的系数。

返回值：经拉格朗日插值公式计算后的值。

"""

def CalculateTheValueOfLarangeInterpolation(data\_x,parameters,x):

returnValue=0

i = 0;

while i < len(parameters):

temp = 1

j = 0;

while j< len(parameters):

if(i!=j):

temp \*=x-data\_x[j]

j+=1

returnValue += temp \* parameters[i]

i += 1

return returnValue

"""

功能：将函数绘制成图像

参数：data\_x,data\_y为离散的点.new\_data\_x,new\_data\_y为由拉格朗日插值函数计算的值。x为函数的预测值。

返回值：空

"""

def Draw(data\_x,data\_y,new\_data\_x,new\_data\_y):

plt.plot(new\_data\_x, new\_data\_y, label="拟合曲线", color="black")

plt.scatter(data\_x,data\_y, label="离散数据",color="red")

plt.scatter(115, 10.723805294764, label="真实数据", color="green")

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

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

plt.title("拉格朗日插值拟合数据")

plt.legend(loc="upper left")

plt.show()

"""

由于三个点绘制的拟合曲线效果太差，所以采用这样的方法来进行数据拟合。

1>利用原数据计算出拉格朗日插值多项式的函数，分别在10-150区间内，每10个数取一个点，计算相应的值，绘制函数图像。

2>将源数据以点的形式画在图像上，

3>将115代入拉格朗日函数计算出相应的值，绘制在图像上。点为红色。并将真实的值也绘制在图像上点为绿色。看红色的点和绿色的点是否重合。

4>结果证明红色的点和绿色的点重合，说明拉格朗日函数插值效果较好。

"""

parameters=ParametersOfLagrangeInterpolation(x,y,3)

datax=[10,20,30,40,50,60,70,80,90,100,110,120,130,140,150]

datay=[]

for temp in datax:

datay.append(CalculateTheValueOfLarangeInterpolation(x,parameters,temp))

x.append(115)

y.append(CalculateTheValueOfLarangeInterpolation(x,parameters,115))

Draw(x,y,datax,datay)

### 埃尔米特插值

import numpy as np

from sympy import \*

import matplotlib.pyplot as plt

def f(x):

return 1 / (1 + x \*\* 2)

def cal(begin, end):

by = f(begin)

ey = f(end)

Y = 1 / (1 + n \*\* 2)

df = diff(Y)

dfb = df.subs(n, begin)

dfe = df.subs(n, end)

oldFrac = ((n - end) / (begin - end))

newFrac = ((n - begin) / (end - begin))

I = (oldFrac \*\* 2) \* (1 + 2 \* newFrac) \* by + (newFrac \*\* 2) \* (1 + 2 \* oldFrac) \* ey + (oldFrac \*\* 2) \* (

n - begin) \* dfb + (newFrac \*\* 2) \* (n - end) \* dfe

return I

def calnf(x):

nf = []

for i in range(len(x) - 1):

nf.append(cal(x[i], x[i + 1]))

return nf

def calf(f, x):

y = []

for i in x:

y.append(f.subs(n, i))

return y

def nfSub(x, nf):

tempx = np.array(range(11)) - 5

dx = []

for i in range(10):

labelx = []

for j in range(len(x)):

if x[j] >= tempx[i] and x[j] < tempx[i + 1]:

labelx.append(x[j])

elif i == 9 and x[j] >= tempx[i] and x[j] <= tempx[i + 1]:

labelx.append(x[j])

dx = dx + calf(nf[i], labelx)

return np.array(dx)

def draw(nf):

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

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

x = np.linspace(-5, 5, 101)

y = f(x)

Ly = nfSub(x, nf)

plt.plot(x, y, label='原函数')

plt.plot(x, Ly, label='分段Hermite插值函数')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.savefig('1.png')

plt.show()

def lossCal(nf):

x = np.linspace(-5, 5, 101)

y = f(x)

Ly = nfSub(x, nf)

Ly = np.array(Ly)

temp = Ly - y

temp = abs(temp)

print(temp.mean())

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

x = np.array(range(11)) - 5

y = f(x)

n, m = symbols('n m')

init\_printing(use\_unicode=True)

nf = calnf(x)

draw(nf)

lossCal(nf)

### 三次样条插值

**边界条件1：已知两端一阶导数值，即多项式最左右两端导数值等于某个已知值。**

def spline\_boundary\_one\_sovle(self, b1, b2):

self.spline[0][0], self.spline[0][1] = 2, 1

self.spline[-1][-1], self.spline[-1][-2] = 2, 1

index = 0

for k in range(1, (self.n - 1)):

u = (self.x[k] - self.x[k - 1]) / (self.x[k + 1] - self.x[k - 1])

lam = 1 - u

self.spline[k][index:index + 3] = (u, 2, lam)

index += 1

self.d[0] = 6 / (self.x[1] - self.x[0]) \* (self.df[0][0] - b1)

self.d[-1] = 6 / (self.x[-1] - self.x[-2]) \* (b2 - self.df[0][-1])

flag = 1

for p in range(1, (self.n - 1)):

self.d[p] = 6 \* self.df[1][flag]

flag += 1

print(self.d)

eq = np.dot(self.spline, self.M) - self.d

solver = solve(eq)

for x in solver:

self.results.append(solver[x])

return self.results

**边界条件2：已知两端二阶导数值，即多项式最左右两端二阶导数值等于某个已知值，当左右二阶导数都等于0时称为自然边界条件。**

def spline\_boundary\_two\_solve(self, a, b):

self.spline\_boundary\_one\_sovle(a, b)

self.spline[0][2] = 0

self.spline[-1][-2] = 0

self.d[0] = 2 \* a

self.d[-1] = 2 \* b

eq = np.dot(self.spline, self.M) - self.d

solver = solve(eq)

self.results = []

for x in solver:

self.results.append(solver[x])

return self.results

**边界条件3：左右两端函数值、一阶导数、二阶导数值相等，这样的样条函数称为周期样条函数。**

def spline\_boundary\_three\_solve(self):

self.spline = np.zeros((self.n - 1, self.n - 1))

self.spline[0][0] = 2

self.spline[0][3] = (self.x[2] - self.x[1]) / (self.x[2] - self.x[0])

self.spline[0][-1] = 1 - self.spline[0][4]

self.spline[-1][0] = (self.x[1] - self.x[0]) / (self.x[-1] - self.x[-2] + self.x[1] - self.x[0])

self.spline[-1][-2] = 1 - self.spline[-1][0]

self.spline[-1][-1] = 2

index = 0

for k in range(1, (self.n - 2)):

u = (self.x[k + 1] - self.x[k]) / (self.x[k + 2] - self.x[k])

lam = 1 - u

self.spline[k][index:index + 3] = (u, 2, lam)

index += 1

self.d = np.ones(self.n - 1)

for i in range(1, (self.n - 1)):

self.d[i] = 6 \* self.df[1][i]

self.d[-1] = 6 \* (self.df[0][0] - self.df[0][-1]) / (self.x[1] - self.x[0] + self.x[-1] - self.x[-2])

M0 = self.M[0]

del self.M[0]

eq = np.dot(self.spline, self.M) - self.d

solver = solve(eq)

self.results = []

for x in solver:

self.results.append(solver[x])

self.results = [self.results[-1]] + self.results

return self.results

**表达式函数**

# 表达式函数

def spline\_get\_expression(self):

self.spline\_expression = []

for j in range((self.n - 1)):

hj = self.x[j + 1] - self.x[j]

exp1 = self.results[j] \* (self.x[j + 1] - self.t) \*\* 3 / 6

exp2 = self.results[j + 1] \* (self.t - self.x[j]) \*\* 3 / 6

exp3 = (self.y[j] - self.results[j] \* hj \*\* 2 / 6) \* (self.x[j + 1] - self.t)

exp4 = (self.y[j + 1] - self.results[j + 1] \* hj \*\* 2 / 6) \* (self.t - self.x[j])

exp = exp1 + exp2 + exp3 + exp4

exp = exp / hj

exp = expand(exp)

self.spline\_expression.append(exp)

return self.spline\_expression

#求值函数，不同区间会有不同的表达式，所以需要先确定给的点的所属区间。

def spline\_GetValue(self, value):

idx = self.n - 2

for index, element in enumerate(self.x):

if value < element:

idx = index - 1

break

idx = max(idx, 0)

f = self.spline\_expression[idx]

f = f.subs(self.t, value)

return f

## 拟合算法

多项式拟合

import numpy as np

from matplotlib import pyplot as plt

def f(x):

return x\*\*2+1

def f\_fit(x,y\_fit):

a,b,c=y\_fit.tolist()

return a\*x\*\*2+b\*x+c

x=np.linspace(-5,5)

y=f(x)+np.random.randn(len(x))#加入噪音

y\_fit=np.polyfit(x,y,2)#二次多项式拟合

y\_show=np.poly1d(y\_fit)#函数优美的形式

print(y\_show)#打印

y1=f\_fit(x,y\_fit)

plt.plot(x,f(x),'r',label='original')

plt.scatter(x,y,c='g',label='before\_fitting')#散点图

plt.plot(x,y1,'b--',label='fitting')

plt.title('polyfitting')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()#显示标签

plt.show()

三角函数拟合

import numpy as np

from matplotlib import pyplot as plt

from scipy.optimize import curve\_fit

def f(x):

return 2\*np.sin(x)+3

def f\_fit(x,a,b):

return a\*np.sin(x)+b

def f\_show(x,p\_fit):

a,b=p\_fit.tolist()

return a\*np.sin(x)+b

x=np.linspace(-2\*np.pi,2\*np.pi)

y=f(x)+0.5\*np.random.randn(len(x))#加入了噪音

p\_fit,pcov=curve\_fit(f\_fit,x,y)#曲线拟合

print(p\_fit)#最优参数

print(pcov)#最优参数的协方差估计矩阵

y1=f\_show(x,p\_fit)

plt.plot(x,f(x),'r',label='original')

plt.scatter(x,y,c='g',label='before\_fitting')#散点图

plt.plot(x,y1,'b--',label='fitting')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()