import numpy as np  
from mpl\_toolkits.mplot3d import Axes3D  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import pandas as pd  
  
  
def steps(tn): # вычисление длин участков сплайна и величины шага дискретизации для каждого сплайна  
 N = tn.shape[0]  
 dt = np.zeros(N)  
 for i in range(N - 1):  
 if i == (N - 1):  
 T = tn[N] - tn[N - 1]  
 else:  
 T = tn[i+1] - tn[i]  
 dt[i] = (T\*1.0)/((K)\*1.0)  
 return dt  
  
def get\_t(tn, dt): #вычисление аргументов сплайна (абсчисс точек, по которым будет строиться интерполяционная кривая)  
 # tn - абсциссы точек отсчёта  
 # ве - шаги дискретизации  
 # в общем случае шаг дискретизации меняется от участка сплайна к участку, но не в этот раз, ибо сильно заморочно на поверхности задавать различные шаги дискретизации  
 ind = 0  
 N = dt.shape[0]  
 PointCnt = K \* (N - 1)  
 t = np.zeros(PointCnt + 1)  
 t[PointCnt] = tn[N - 1]  
 for i in range(N - 1):  
 for j in range(K):  
 if ind >= PointCnt:  
 break  
 if j == 0:  
 t[ind] = tn[i]  
 ind = ind + 1  
 continue  
 else:  
 t[ind] = t[ind - 1] + dt[i]  
 ind = ind+1  
 return t  
  
  
def calc\_spline2D(yn, tn, dy): #построение сплайна  
 # yn - значения функции (ординаты в точках отсчёта)  
 # tn - значения аргументов (абсциссы в точках отсчёта)  
 # dy - производные функции в точках отсчёта  
 ind = 0  
 N = yn.shape[0]  
 P = np.zeros(N - 1)  
 D = np.zeros(N - 1)  
 PointCnt = K\*(N-1)  
 dt = steps(tn) # шаги дискретизации  
 points = np.zeros(PointCnt + 1) # абсциссы точек, по которым будет строиться сплайновая кривая  
 points = get\_t(tn, dt)  
 Y = np.zeros(PointCnt + 1) # ординаты точек сплайновой кривой  
 for i in range(N-1):  
 TT = tn[i+1] - tn[i]  
 L = 1.0\*yn[i+1] - 1.0\*yn[i] - 0.5\*TT\*(dy[i+1]+dy[i])  
 P[i] = (-L - np.sign(L)\*np.sqrt(L\*L + 0.25\*TT\*TT\*pow(dy[i+1]-dy[i], 2))) / (0.5\*TT\*TT)  
 if P[i] == 0 or L == 0:  
 D[i] = 0  
 else:  
 D[i] = (dy[i+1]-dy[i]+TT\*P[i]) / (2\*P[i])  
 tp = tn[i + 1] - D[i]  
 for j in range(K):  
 if ind >= PointCnt:  
 break  
 if j == 0:  
 Y[ind] = yn[i]  
 ind = ind + 1  
 continue  
 tL = 1.0 \* (points[ind] - tn[i])  
 tt = 1.0\*(points[ind] - tn[i])  
 if points[ind] < tp:  
 Y[ind] = yn[i] + dy[i] \* tL - (tL \* 1.0 \* tL \* P[i]) / 2.0  
 else:  
 tR = TT - tt  
 if tR < 0:  
 tR = -tR  
 Y[ind] = yn[i + 1] - dy[i + 1] \* tR + pow(tR, 2) \* P[i] / 2.0  
 ind = ind + 1  
 Y[PointCnt] = yn[N - 1]  
 return Y, points  
  
# reading data  
pars\_df = pd.read\_csv("in\_pars.txt", sep=" ")  
valX = pars\_df["x"].values  
valY = pars\_df["y"].values  
print("valX=", valX)  
print("valY=", valY)  
data\_df = pd.read\_csv("in\_data.txt", sep=" ", header=None)  
valZ = data\_df.values  
print("valZ=", valZ)  
# from the above  
K = int(input("Enter amount of points per spline: "))  
  
lx = len(valX)  
ly = len(valY)  
print("lx=", lx)  
print("ly=", ly)  
elemX = int((lx - 1)\*K + 1)  
elemY = int((ly - 1)\*K + 1)  
print("elemX=", elemX)  
print("elemY=", elemY)  
#производные  
derZ = np.zeros((lx, ly))  
resX = np.zeros(elemX)  
resY = np.zeros(elemY)  
  
resOXZ = np.zeros((elemX, ly), dtype=np.float32)  
resOYZ = np.zeros((elemX, elemY), dtype=np.float32)  
  
# считывание данных о производных в граничных точках. Врочем, мы всё равно считаем эти производные как левые и правые соответственно, но пусть полежит тут на всякий случай, вдруг понадобится  
derZ[0] = pars\_df["bot"].values  
derZ[-1] = pars\_df["top"].values  
  
# Построим одномерные кривые в плоскостях OXZ  
for iy in range(ly):  
 for i in range(lx):  
 if i == 0:  
 derZ[i][iy] = (valZ[i + 1][iy] - valZ[i][iy]) / (valX[i + 1] - valX[i])  
 continue  
 if i == (lx - 1):  
 derZ[i][iy] = (valZ[i][iy] - valZ[i - 1][iy]) / (valX[i] - valX[i - 1])  
 continue  
 if(((valZ[i - 1][iy] <= valZ[i][iy]) and (valZ[i + 1][iy] <= valZ[i][iy])) or ((valZ[i - 1][iy] >= valZ[i][iy])and(valZ[i + 1][iy] >= valZ[i][iy]))):  
 derZ[i, iy] = 0  
 else:  
 derZ[i][iy] = 0.5 \* ((valZ[i][iy] - valZ[i - 1][iy]) / (valX[i] - valX[i - 1]) + (valZ[i + 1][iy] - valZ[i][iy]) / (valX[i + 1] - valX[i]))  
 print("derZ=", derZ)  
 resOXZ[:, iy], resX = calc\_spline2D(valZ[:, iy], valX, derZ[:, iy])  
derZ[:, 0] = pars\_df["left"].values  
derZ[:, -1] = pars\_df["right"].values  
print("resOXZ=", resOXZ)  
derZ2 = np.zeros((elemX, elemY))  
print("derZ2=", derZ)  
  
# Построим одномерные кривые в плоскостях OYZ  
# Данные для построения берутся из OXZ - результатов вычислений на предыдущем шаге  
for ix in range(elemX):  
 derZ2[ix, 0] = derZ[round(ix/K), 0]  
 derZ2[ix, ly-1] = derZ[round(ix/K), ly-1]  
 for i in range(ly):  
 if i==0:  
 derZ2[ix][i] = (resOXZ[ix][i+1] - resOXZ[ix][i])/(valY[i+1] - valY[i]) # правая производная  
 continue  
 elif i==(ly-1):  
 derZ2[ix][i] = (resOXZ[ix][i] - resOXZ[ix][i-1])/(valY[i] - valY[i-1]) # левая производная  
 continue  
 if(((resOXZ[ix][i - 1] <= resOXZ[ix][i])and(resOXZ[ix][i + 1] <= resOXZ[ix][i]))or((resOXZ[ix][i - 1] >= resOXZ[ix][i])and(resOXZ[ix][i + 1] >= resOXZ[ix][i]))):  
 derZ2[ix, i] = 0 # на случай точки экстремума  
 else:  
 derZ2[ix][i] = 0.5 \* ((resOXZ[ix][i] - resOXZ[ix][i-1])/(valY[i] - valY[i-1]) + (resOXZ[ix][i+1] - resOXZ[ix][i])/(valY[i+1] - valY[i])) # центральная производная  
 # print("resY.shape[0]=", resY.shape[0], ", resOXZ[ix].shape[0]=", resOXZ[ix].shape[0])  
 resOYZ[ix], resY = calc\_spline2D(resOXZ[ix], valY, derZ2[ix])  
  
# Массив resX содержит координаты Х плоскостей OXZ  
# Массив resY содержит координаты Y плоскостей OYZ  
# Массив resOYZ содержит значения узлов интерполированной поверхности  
print("resOYZ=", resOYZ)  
print("resX=", resX)  
print("resY=", resY)  
  
X, Y = np.meshgrid(resX, resY)  
  
fig = plt.figure()  
ax = fig.gca(projection='3d')  
ax.set\_xlabel('X axis')  
ax.set\_ylabel('Y axis')  
ax.set\_zlabel('Z axis')  
surf = ax.plot\_surface(X, Y, resOYZ, cmap=cm.bone, linewidth=0, antialiased=True)  
Xp, Yp = np.meshgrid(valX, valY)  
ax.scatter(Xp, Yp, valZ, c="r", s=20)  
plt.show()