Anejo 2: Código subprograma 2: Cálculo y ventana de resultados (ISC_v0_calc.py)

```
# -*- coding: utf-8 -*-
"""
Spyder Editor
Python 3.5.2

Segona part del codi del programa Inlet Spacing Calculator (ISC)
Càlcul i finestra resultats
Versió 0

22/09/2017, Barcelona
Autor: Miquel Sàrrias
"""

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from tkinter import PhotoImage
import matplotlib.patches as mpatches
import matplotlib#per importar graf a tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg#per importar graf a tk
from matplotlib.figure import Figure#per importar graf a tk
matplotlib.use('TkAgg')#per importar graf a tk
import time

start_time=time.time()

#-----------------------------------------------------  Lectura de variables

CP=open('ComputingParameters.txt','r')

with open('ComputingParameters.txt') as CP:
    CP_lines = CP.readlines()

CP_lines = [x[7:] for x in CP_lines]

sg=int(CP_lines[1])
so=float(CP_lines[2])
sx1=float(CP_lines[3])
n1=float(CP_lines[4])
w1=float(CP_lines[5])
sx2=float(CP_lines[6])
n2=float(CP_lines[7])
w2=float(CP_lines[8])
k=float(CP_lines[9])
sl=float(CP_lines[10])

ymax=float(CP_lines[13])
vmax=float(CP_lines[14])
vymax=float(CP_lines[15])
vy2max=float(CP_lines[16])

A=float(CP_lines[19])
B=float(CP_lines[20])
cc=float(CP_lines[21])

IM=float(CP_lines[24])
a=float(CP_lines[25])
b=float(CP_lines[26])
c=float(CP_lines[27])
d=float(CP_lines[28])
```

```
65  D=float(CP_lines[29])
66
67  #sg=0
68  #so=0
69  #sx1=0
70  #n1=0
71  #w1=0
72  #sx2=0
73  #n2=0
74  #w2=0
75  #k=0
76  #sl=1000
77  #
78  #ymax=0
79  #vmax=0
80  #vymax=0
81  #vy2max=0
82  #
83  #A=0
84  #B=0
85  #cc=0
86  #
87  #IM=2
88  #a=4477.44
89  #b=19.03
90  #c=1
91  #d=1
92  #D=1
93
94
95
96  #-------------------------------------------------------  Creació vector
    puja
97
98  rfall=[0]*int(D*60/5*3) #temps de calcul: 3 cops l'episodi de pluja
99  At=5                    #l'interval de temps sempre seran 5 minuts
100
101 if IM == 1:
102     #rfall=
    [16.6734,22.5430,32.1641,49.5817,86.2501,186.3271,122.1428,64.1517,39.4692,26.7153,19.2773,1
103     a=4477.44
104     b=19.03
105     c=1
106     d=1
107     D=1
108
109 if IM == 2 or IM == 1: # La idea era definir el T10 BCN mediante un vertor,
    pero así es más versátil
110     rf1=[]
111     n=int(D*60/At)
112
113     for i in range(0,n):
114         I=a/np.power(np.power(At*(i+1),c)+b,d)
115         rf1.append(I*(i+1)-sum(rf1))
116
117     rfall=np.zeros(n)
118     for i in range(0,n):
119         if i == 0:
120             rfall[n//2]=rf1[0]
121         if i%2 == 1:
122             rfall[n//2-i//2-1]=rf1[i]
123         if i%2 == 0:
124             rfall[n//2+i//2]=rf1[i]
125
126 print(rfall)
127
```

```
128   rfall=[x/3600000 for x in rfall]
129
130   add=[0]*(len(rfall)*2)
131   rfall=rfall+add
132
133
134
135   #print(rf)
136   #print('\nrfall =',rfall,'\n')
137   print('Pluja:',int(np.sum(rfall)*300*1000),'l/m2 en',D,'hores.')
138
139   #-------------------------------------------  Càlcul dels plans
      d'escorrentia
140
141   beta = 0.6
142   At = 300 #segons
143   Ax = 0.5
144
145   print('cas',sg,'\n')
146
147   #-------  Cas (1): Gutter; road
148
149   if sg == 1:
150
151       x1=[] #crear vector espai
152       d=w1
153       while d >= 0:
154           x1.append(w1-d)
155           d=d-Ax
156       if x1[-1] != w1:
157           x1.append(w1)
158       print('x1 =', x1,'\n')
159
160
161       alpha1 = np.power(n1/np.sqrt(sx1),beta) #definir alpha de cada
162
163       Q1=np.zeros((len(x1), len(rfall))) #def matriu flux pla1
164       Q1[1][0]=0.00000000001
165
166       for i in range (1, len(x1)): #calcular matriu flux pla1
167           for j in range (1, len(rfall)):
168               Ax=x1[i]-x1[i-1]
169               term1 = alpha1 * beta * np.power(( Q1[i][j-1] + Q1[i-1][j]) / 2,
      beta-1)
170               term2 = (rfall[j-1] + rfall[j]) / 2
171               Q1[i][j] = (At/Ax*Q1[i-1][j] + Q1[i][j-1]*term1 + At*term2) / (At/
      Ax + term1)
172   #             if i == 1 and j == 1:
173   #                 print(term1)
174   #                 print(term2)
175   #                 print(Q1[i][j])
176
177   #    print(Q1)
178       Qin=np.sum(rfall)*At*w1
179       Qout=np.sum(Q1[-1])*300
180       print('Qin ',np.sum(rfall)*At*w1)
181       print('Qout',np.sum(Q1[-1])*300)
182       print('>>',np.abs((np.sum(rfall)*At*w1-np.sum(Q1[-1])*300))/(np.sum(Q1
      [-1])*300/100),'%\n')
183   #    print(max(Q1[5]))
184
185       q=Q1[-1]
186
187
188
189   #-------  Caso (2): Gutter with sidewalk >> dos planos (calzada y acera)
```

```
190
191 if sg == 2: # sg = tipo de sección transversal
192
193     alpha1 = np.power(n1/np.sqrt(sx1),beta)
194     alpha2 = np.power(n2/np.sqrt(sx2),beta)
195
196     x1=[] #crear vector espacial (dividir ancho calzada en tramos de 0.5m)
197     d=w1
198     while d >= 0:
199         x1.append(w1-d)
200         d=d-Ax
201     if x1[-1] != w1:
202         x1.append(w1)
203     print('x1 =', x1,'\n')      #control de ejecución en consola
204
205     Q1=np.zeros((len(x1), len(rfall))) #definición matriz flujo plano 1
206                                        #dimensiones espaciales y temporales
207     Q1[1][0]=0.00000000001 #Definir primer flujo no nulo (inestabilidades)
208
209     for i in range (1, len(x1)): #cálculo onda cinemática plano 1
210         for j in range (1, len(rfall)):
211             Ax=x1[i]-x1[i-1]
212             term1 = alpha1 * beta * np.power(( Q1[i][j-1] + Q1[i-1][j]) / 2,
    beta-1)
213             term2 = (rfall[j-1] + rfall[j]) / 2
214             Q1[i][j] = (At/Ax*Q1[i-1][j] + Q1[i][j-1]*term1 + At*term2) / (At/
    Ax + term1)
215
216     x2=[] #crear vector espacial plano 2 (dividir ancho acera en tramos de
    0.5m)
217     d=w2
218     while d >= 0:
219         x2.append(w2-d)
220         d=d-Ax
221     if x2[-1] != w2:
222         x2.append(w2)
223     print('x2 =', x2,'\n')      #control de ejecución en consola
224
225     Q2=np.zeros((len(x2), len(rfall))) #definición matriz flujo plano 2
226                                        #dimensiones espaciales y temporales
227     Q2[1][0]=0.00000000001 #Definir primer flujo no nulo (inestabilidades)
228
229     for i in range (1, len(x2)): #cálculo onda cinemática plano 2
230         for j in range (1, len(rfall)):
231             Ax=x2[i]-x2[i-1]
232             term1 = alpha2 * beta * np.power(( Q2[i][j-1] + Q2[i-1][j]) / 2,
    beta-1)
233             term2 = (rfall[j-1] + rfall[j]) / 2
234             Q2[i][j] = (At/Ax*Q2[i-1][j] + Q2[i][j-1]*term1 + At*term2) / (At/
    Ax + term1)
235
236     print('Q1in ',np.sum(rfall)*At*w1)       #control de ejecución en
    consola
237     print('Q1out',np.sum(Q1[-1])*300)
238     print('>>',np.abs((np.sum(rfall)*At*w1-np.sum(Q1[-1])*300))/(np.sum(Q1
    [-1])*300/100),'%')
239
240     print('Q2in ',np.sum(rfall)*At*w2)
241     print('Q2out',np.sum(Q2[-1])*300)
242     print('>>',np.abs((np.sum(rfall)*At*w2-np.sum(Q2[-1])*300))/(np.sum(Q2
    [-1])*300/100),'%')
243
244     q=[sum(x) for x in zip(Q1[-1],Q2[-1])] #suma del flujo que amandona los
    dos planos
245                                            #(fujo de entrada para el càlculo
    del canal
```

```
246                                                 #de escorrentía)
247
248     Qin=np.sum(rfall)*At*w1+np.sum(rfall)*At*w2
249     Qout=np.sum(Q1[-1])*300+np.sum(Q2[-1])*300 #control de volúmenes y errores
250
251
252
253
254 #-------  Cas (3): v-section; road
255
256 if sg == 3:
257
258     x1=[] #crear vector espai
259     d=w1
260     while d >= 0:
261         x1.append(w1-d)
262         d=d-Ax
263     if x1[-1] != w1:
264         x1.append(w1)
265     print('x1 =', x1,'\n')
266
267
268     alpha1 = np.power(n1/np.sqrt(sx1),beta) #definir alpha de cada
269
270     Q1=np.zeros((len(x1), len(rfall))) #def matriu flux pla1
271     Q1[1][0]=0.00000000001
272
273     for i in range (1, len(x1)): #calcular matriu flux pla1
274         for j in range (1, len(rfall)):
275             Ax=x1[i]-x1[i-1]
276             term1 = alpha1 * beta * np.power(( Q1[i][j-1] + Q1[i-1][j]) / 2,
    beta-1)
277             term2 = (rfall[j-1] + rfall[j]) / 2
278             Q1[i][j] = (At/Ax*Q1[i-1][j] + Q1[i][j-1]*term1 + At*term2) / (At/
    Ax + term1)
279
280     q=[x * 2 for x in Q1[-1]]
281
282 #-------  Cas (4): v-section with sidewalk; road + sidewalk
283
284 if sg == 4:
285
286     alpha1 = np.power(n1/np.sqrt(sx1),beta)
287     alpha2 = np.power(n2/np.sqrt(sx2),beta)
288
289     x2=[] #crear vector espai
290     d=w2
291     while d >= 0:
292         x2.append(w2-d)
293         d=d-Ax
294     if x2[-1] != w2:
295         x2.append(w2)
296     print('x2 =', x2,'\n')
297
298     Q2=np.zeros((len(x2), len(rfall))) #def matriu flux pla1
299     Q2[1][0]=0.00000000001
300
301     for i in range (1, len(x2)): #calcular matriu flux pla1
302         for j in range (1, len(rfall)):
303             Ax=x2[i]-x2[i-1]
304             term1 = alpha2 * beta * np.power(( Q2[i][j-1] + Q2[i-1][j]) / 2,
    beta-1)
305             term2 = (rfall[j-1] + rfall[j]) / 2
306             Q2[i][j] = (At/Ax*Q2[i-1][j] + Q2[i][j-1]*term1 + At*term2) / (At/
    Ax + term1)
307
```

```
308       x1=[] #crear vector espai
309       d=w1
310       while d >= 0:
311           x1.append(w1-d)
312           d=d-Ax
313       if x1[-1] != w1:
314           x1.append(w1)
315       print('x1 =', x1,'\n')
316
317       Q1=np.zeros((len(x1), len(rfall))) #def matriu flux pla1
318       Q1[0]=Q2[-1]
319
320       for i in range (1, len(x1)): #calcular matriu flux pla1
321           for j in range (1, len(rfall)):
322               Ax=x1[i]-x1[i-1]
323               term1 = alpha1 * beta * np.power(( Q1[i][j-1] + Q1[i-1][j]) / 2,
    beta-1)
324               term2 = (rfall[j-1] + rfall[j]) / 2
325               Q1[i][j] = (At/Ax*Q1[i-1][j] + Q1[i][j-1]*term1 + At*term2) / (At/
    Ax + term1)
326
327       print('Q1in ',np.sum(rfall)*At*w1)
328       print('Q1in2',np.sum(rfall)*At*w2,'   Total =',np.sum(rfall)*At*w1+np.sum
    (rfall)*At*w2)
329       print('Q1out',np.sum(Q1[-1])*300)
330       print('>>',np.abs((np.sum(rfall)*At*w1+np.sum(rfall)*At*w2-np.sum(Q1
    [-1])*300))/(np.sum(Q1[-1])*300/100),'%')
331
332       print('Q2in ',np.sum(rfall)*At*w2)
333       print('Q2out',np.sum(Q2[-1])*300)
334       print('>>',np.abs((np.sum(rfall)*At*w2-np.sum(Q2[-1])*300))/(np.sum(Q2
    [-1])*300/100),'%')
335
336       q=[x * 2 for x in Q1[-1]]
337
338
339
340   ###############################################################################
341   ###############################################################################
342
343   #-------------------------------------------------------  Cálculo caudales
    máximos
344
345   Qlimv   = np.power(vmax,4)*np.power(n1,3)/16/sx1/np.power(0.376,3)/np.power
    (so,3/2)
346   Qlimy   = 0.376*np.sqrt(so)*np.power(ymax,8/3)/n1/sx1
347   Qlimvy  = np.power(vymax,8/5)*np.power(n1,3/5)/np.power(2,8/5)/sx1/np.power
    (0.376,3/5)  \
348             /np.power(so,3/10)
349   Qlimvy2 = vy2max/2/sx1
350
351   print('Lims:',Qlimv,Qlimy,Qlimvy,Qlimvy2) # control de ejecución en consola
352
353   Qlim = min(Qlimv,Qlimy,Qlimvy,Qlimvy2)
354
355   print('Qlim:',Qlim,'\n') # control de ejecución en consola
356
357   #-------------------------------------------------------------  Cálculo
    canal
358
359   dist_total=sl
360   Ax=1 # Resolución espacial de cálculo
361
362   xc=np.arange(0,dist_total,Ax) # Vector discretización espacial
363
364   def calc(Dist, xc, rfall, alpha1, beta, q, At,     # definición función
```

```
    cálculo
365          Ax, A, B, n1, sx1, so, cc):
366
367      Qc=np.zeros((len(xc), len(rfall)))  # Definición matriz canal
368      Qc[1][0]=0.000001                   # Definir primer flujo no nulo
    (inestabilidades)
369      Qcap_t=0                            # Contador caudal captado
370
371
372      for i in range (1, len(xc)): # Cálculo recurrente matriz canal
373
374          for j in range (1, len(rfall)):
375              term1 = alpha1 * beta * np.power( (Qc[i][j-1] + Qc[i-1][j]) / 2,
    beta-1)
376              term2 = (q[j-1] + q[j]) / 2
377              Qc[i][j] = (At/Ax*Qc[i-1][j] + Qc[i][j-1]*term1 + At*term2) / (At/
    Ax + term1)
378
379              if i*Ax % Dist == 0 and Qc[i][j]>0.005: # Condición para cálculo
    sumidero
380                                                      # caudal mínimo para
    estabilidad
381                  E=(1-cc)*A*np.power(np.power(Qc[i][j],5/8)*np.power(0.376/n1/
    sx1,3/8)* \
382                  np.power(so,3/16),-B)   # Cálculo eficiencia
383                  if E>1:
384                      E=1
385                  Qcap=Qc[i][j]*E             # Cálculo caudal captado
386                  Qc[i][j]=Qc[i][j]-Qcap      # Cálculo caudal que pasa
387                  Qcap_t=Qcap_t+Qcap          # Actualización caudal captado
    acumulado
388      return(Qc, Qcap_t)
389
390  # Cálculo de la onda cinemática si no se incluye el uso de sumideros:
391  (Qc,Qcap_t)=calc(dist_total, xc, rfall, alpha1, beta, q, At, Ax, A, B, n1,
    sx1, so, cc)
392
393  print('El Qmaxdist_total es',Qc.max())  # Control de ejecución en consola
394
395  if Qc.max() < Qlim:             # Si no se superan los límites de seguridad:
396      print('no al posar reixes') # Control de ejecución en consola
397      situation=1                 # No es necesario utilizar drenaje superficial
398  else:
399      # Cálculo de la onda cinemática con imbornales cada 3 metros:
400      (Qc,Qcap_t)=calc(3*Ax, xc, rfall, alpha1, beta, q, At, Ax, A, B, n1, sx1,
    so, cc)
401      print('El QmaxAx es',Qc.max())
402      if Qc.max() > Qlim:  # Si no se consigue rebajar los caudales máximos por
    debajo los límites:
403          print('No es pot asolir objectiu')  # Control de ejecución en consola
404          situation=2      # No es posible alcanzar el objetivo
405      else:
406          situation=3          # Situación para el cálculo del espaciamiento
    óptimo
407          DistA=3*Ax             # Se define una distancia A (inferior a la
    dist. óptima)
408          DistB=dist_total     # Y una distancia B (superior a la dist. óptima)
409          dist=(((DistA+DistB)/2)//Ax)*Ax  # Y la distancia media entre las dos
410          print('\nDistA:',DistA)
411          print('DistB:',DistB,'\n')# Control de ejecución en consola
412
413
414  if situation == 3:          # Se procede al cálculo del espaciamiento óptimo
415
416      c=0##                   # Contador de iteraciones para control interno
417      while dist != DistA:    # Algoritmo para hallar el espaciamiento óptimo
```

```
418         (Qc,Qcap_t)=calc(dist, xc, rfall, alpha1, beta, q, At, Ax, A, B, n1,
    sx1, so, cc)
419         if Qc.max() > Qlim:         #    Ajuste de las distancias A i B según
    si los caudales
420             DistB=dist              # máximos producidos se encuentran por
    arriba o por
421         else:                       # debajo del límite establecido.
422             DistA=dist              #    Cuando la nueva distancia propuesta
    es la misma que
423                                     # la distancia A, significa que ya no se
    puede ajustar
424                                     # más y que esa es la mayor distancia
    entre sumideros
425                                     # que no sobrepasa el límite.
426
427         print('iteració',c,'distància',dist)    # Control de ejecución en
    consola
428         print('A:',DistA,'B:',DistB)
429         print('Qmax:',Qc.max(),'\n')
430         dist=(((DistA+DistB)/2)//Ax)*Ax     # Actualización de la distancia
    de cálculo
431         c+=1##
432
433     print('>>>>   La distància optima és',DistA)    # Control de ejecución en
    consola
434
435     # Calculo final con la distancia calculada como óptima:
436     (Qc,Qcap_t)=calc(DistA, xc, rfall, alpha1, beta, q, At, Ax, A, B, n1,
    sx1, so, cc)
437
438 QcTmax=np.where(Qc == Qc.max()) #    Obtención de coordenadas de la matriz
    donde se produce el
439                                 # mayor caudal.
440
441
442 print(QcTmax)
443 print('Volum pluja:',Qin*dist_total)
444 print('Volum superficie:',np.sum(Qc[-1])*300)
445 print('Volum subterrani:',Qcap_t*300)
446 print('Error:',np.abs(np.sum(Qc[-1])*300+Qcap_t*300-Qin*dist_total)/
    (Qin*dist_total)*100,'%\n')
447
448 #-------------------------------------------------------   Cálculo de
    estabilidad
449
450 if situation == 3:                      # Situación en que se aplica algoritmo
451     graf = Qc[:,QcTmax[1][0]]           # Vector de caudales en función del
    espacio
452     vect=[]
453
454     i=1
455     while i*DistA < sl:
456         vect.append(graf[i*DistA/Ax-1])     # Vector caudales anteriores a
    sumideros
457         i+=1
458     print(vect)                             # Control de ejecución en consola
459
460     vect = sorted(vect, reverse=True)                   # Orden inverso de
    caudales
461     vect = [ (vect[0]-x)/vect[0]*100 for x in vect]     # Diferencias
    respecto al mayor (%)
462     print(vect)                                         # Control de
    ejecución en consola
463
464     steady=0                # Definición contador
465     while vect[steady]<1:
```

```
466             steady+=1           # Bucle para contar diferencias menores al 1%
467         print(steady)           # Control de ejecución en consola
468
469
470     ### Grafica
471
472     #t=np.arange(0,21600,300)
473     #plt.plot(t,q,'r--',t,rfall,'b--')
474     #plt.show()
475
476     plt.pcolor(Qc)
477     plt.show
478
479     #plt.plot(t,Qc[-1],'k',t,Qc[500],'b',t,Qc[250],'g')
480     #plt.show()
481
482     #plt.plot(Qc[:,QcTmax[1][0]],'r')
483     #plt.show()
484
485     #i=0
486     #while i < 10:
487     #    plt.plot(Qc[:,i],'r')
488     #    i+=1
489     #plt.show()
490
491     elapsed_time=time.time()-start_time
492     print(elapsed_time)
493
494     ##########################################################################################
495     #############                          D E F I N I C I O N S   W I D G E T
        S                                  #############
496     ##########################################################################################
497
498     root = Tk()
499     root.title("Results - Inlet spacing calculator")
500     #root.geometry("620x455+300+300") #num auri (1+sqrt5)/2=1.6180
501     #root.geometry("1240x800+300+300")
502
503     ###############################################################################
        Creació frame principal
504
505     mainframe = ttk.Frame(root, padding="15 15 15 15")
506     mainframe.grid(column=0, row=0, sticky=('N, W, E, S'))
507     mainframe.columnconfigure(0, weight=1)
508     mainframe.rowconfigure(0, weight=1)
509
510     frame_width=460
511     frame_heigth=220
512
513     frame1 = ttk.LabelFrame(mainframe, padding="10 10 10 10", text='Rainfall and
        flow')
514     frame1.grid(column=0, row=0)
515     frame1['borderwidth'] = 2
516     frame1['relief'] = 'groove'
517     frame1['width'] = frame_width
518     frame1['height'] = frame_heigth
519     frame1.grid_propagate(False)
520
521     frame2 = ttk.LabelFrame(mainframe, padding="10 10 10 10", text='Flow limits')
522     frame2.grid(column=0, row=1)
523     frame2['borderwidth'] = 2
524     frame2['relief'] = 'groove'
525     frame2['width'] = frame_width
526     frame2['height'] = frame_heigth
527     frame2.grid_propagate(False)
528
```

```
529   frame3 = ttk.LabelFrame(mainframe, padding="10 10 10 10", text='Spacing')
530   frame3.grid(column=0, row=2)
531   frame3['borderwidth'] = 2
532   frame3['relief'] = 'groove'
533   frame3['width'] = frame_width
534   frame3['height'] = frame_heigth
535   frame3.grid_propagate(False)
536
537   frame4 = ttk.LabelFrame(mainframe, padding="10 10 10 10", text='Charts')
538   frame4.grid(column=1, row=0, rowspan=3)
539   frame4['borderwidth'] = 2
540   frame4['relief'] = 'groove'
541   frame4['width'] = 500
542
543   ##############################################################################
      Creació frame RAINFALL / FLOW
544
545   trf = np.sum(rfall)*300*1000
546   Vin = Qin*dist_total
547   Vdr = Qcap_t*300
548   Vst = np.sum(Qc[-1])*300
549   Err = np.round(np.abs(np.sum(Qc[-1])*300+Qcap_t*300-Qin*dist_total)/
      (Qin*dist_total)*100, 2)
550   p1  = int(Qcap_t*300/(Qcap_t*300+np.sum(Qc[-1])*300)*100)
551   p2  = 100-p1
552
553
554   label01=ttk.Label(frame1, text='The total rainfall discharge has been %d mm.
      \n' % trf)
555   label01.grid(column=0, row=0, sticky='NW')
556
557   label02=ttk.Label(frame1, text='The total water volume entering the system
      has been %d m3.' % Vin)
558   label02.grid(column=0, row=1, pady=(5), sticky='NW')
559
560   label03=ttk.Label(frame1, text='The total water volume that left the system
      through \n   the drainage system has been %d m3 (%s%%).' % (Vdr, p1))
561   label03.grid(column=0, row=2, pady=(5), sticky='NW')
562
563   label04=ttk.Label(frame1, text='The total water volume that left the system
      as \n   overflow has been %d m3 (%s%%).' % (Vst, p2))
564   label04.grid(column=0, row=3, pady=(5), sticky='NW')
565
566   label05=ttk.Label(frame1, text='\nError based on volume: %s %%.' % Err)
567   label05.grid(column=0, row=4, sticky='NW')
568
569   ##############################################################################
      Creació frame FLOW LIMITS
570
571   qlim = np.round(Qlim,4)
572   qlimv = np.round(Qlimv,4)
573   qlimy = np.round(Qlimy,4)
574   qlimvy = np.round(Qlimvy,4)
575   qlimvy2 = np.round(Qlimvy2,4)
576
577
578   label06=ttk.Label(frame2, text='Maximum flows due to specified hazard
      criteria:')
579   label06.grid(column=0, row=0, pady=(3), sticky='NW')
580
581   label07=ttk.Label(frame2, text='   - Due to flow velocity (v):   %s m3/s' %
      qlimv)
582   label07.grid(column=0, row=1, pady=(3), sticky='NW')
583
584   label08=ttk.Label(frame2, text='   - Due to flow depth (y):      %s m3/s' %
      qlimy)
```

```
585   label08.grid(column=0, row=2, pady=(3), sticky='NW')
586
587   label07=ttk.Label(frame2, text='    - Due to v·y:                        %s m3/
      s' % qlimvy)
588   label07.grid(column=0, row=3, pady=(3), sticky='NW')
589
590   label08=ttk.Label(frame2, text='    - Due to v·y2:                      %s m3/s
      \n' % qlimvy2)
591   label08.grid(column=0, row=4, pady=(3), sticky='NW')
592
593   label09=ttk.Label(frame2, text='The maximum allowed flow has been %s m3/s.' %
      qlim)
594   label09.grid(column=0, row=5, sticky='NW')
595
596
597
598   ##############################################################################
      Creació frame SPACINGS
599
600   ymaxcalc = np.round(np.power(Qc.max()*n1*sx1/0.376/np.sqrt(so),3/8),3)
601   vmaxcalc = np.round(2*np.power(Qc.max()*sx1,1/4)*np.power(0.376,3/4)*np.power
      (so,3/8)/np.power(n1,3/4),3)
602   diststeady = int((len(vect)-steady)*DistA)
603
604   if situation == 1:
605       label13=ttk.Label(frame3, text='There is no need to use any inlet.
      Maximum flow is not surpased along the street.')
606       label13.grid(column=0, row=0, sticky='NW')
607   if situation == 2:
608       label14=ttk.Label(frame3, text='There is too much rain to comupute a
      inlet spacing, beeing less than %s meters the space required between inlets.'
      % (Ax))
609       label14.grid(column=0, row=0, sticky='NW')
610   if situation == 3:
611       label15=ttk.Label(frame3, text='The optimal computed space between inlets
      is %s meters.\n' % DistA)
612       label15.grid(column=0, row=0, sticky='NW')
613
614
615   label10=ttk.Label(frame3, text='The maximum street flow is %s m3/s, meaning:'
      % np.round(Qc.max(),4))
616   label10.grid(column=0, row=1, pady=(3), sticky='NW')
617
618   label11=ttk.Label(frame3, text='    - Maximum depth: %s m' % (ymaxcalc))
619   label11.grid(column=0, row=2, pady=(3), sticky='NW')
620
621   label12=ttk.Label(frame3, text='    - Maximum velocity: %s m/s2\n' %
      (vmaxcalc))
622   label12.grid(column=0, row=3, pady=(3), sticky='NW')
623
624   if situation == 3:
625       if steady != 1:
626           situation3 = 1
627           label13=ttk.Label(frame3, text='The overland flow becomes steady from
      a distance\n   of %s meters (black dashed line).' % diststeady)
628           label13.grid(column=0, row=4, sticky='NW')
629
630       if steady == 1:
631           situation3 = 2
632           label14=ttk.Label(frame3, text='The street is not long enough to
      stabilize the overland flow.')
633           label14.grid(column=0, row=4, sticky='NW')
634
635
636
637
```

```
638 ################################################################################
    Creació frame PLOTS
639
640
641 #######  F1
642
643 rfallgraf=[x*3600000 for x in rfall]
644 outflowgraf = Qc[QcTmax[0][0]]
645 t=np.arange(0,3*D,1/12)
646
647 f1 = Figure(figsize=(6, 3.5), dpi=90)
648
649 a11 = f1.add_subplot(111)
650 a11.bar(t, rfallgraf, width = 1/12)#
651 a11.set_ylabel('Rainfall intensity (mm/h)', color='b')
652 a11.tick_params('y', colors='b')
653
654 a12 = a11.twinx()
655 a12.plot(t, outflowgraf, 'r')
656 a12.set_ylabel('Maximum street flow (m3/s)', color='r')
657 a12.tick_params('y', colors='r')
658
659 a11.set_xlabel('Time (h)', fontsize=10)
660 a11.grid(True)
661 canvas = FigureCanvasTkAgg(f1, frame4)
662 canvas.show()
663 canvas.get_tk_widget().grid(column=0, row=0, pady=(0,10))
664
665 #######  F2
666
667 f2 = Figure(figsize=(6, 3.5), dpi=90)
668 a2 = f2.add_subplot(111)
669 a2.plot(xc, Qc[:,QcTmax[1][0]], 'r')
670 if situation3 == 1:
671     a2.axvline(x=diststeady, linewidth=1, color = 'k', linestyle = '--')
672 a2.set_ylabel('Maximum street flow (m3/s)', color='r')
673 a2.tick_params('y', colors='r')
674 a2.set_xlabel('Distance (m)', fontsize=10)
675 a2.grid(True)
676 canvas = FigureCanvasTkAgg(f2, frame4)
677 canvas.show()
678 canvas.get_tk_widget().grid(column=0, row=1, pady=(10,0))
679
680
681
682
683
684
685
686 for child in mainframe.winfo_children(): child.grid_configure(padx=5, pady=5)
687
688
689 root.mainloop()
690
```