**### Lecture/écriture et l’intégration numérique**

### Fonction de formatage

def formate(c):

 cmod=""

 for x in c:

 if x==",":

 cmod=cmod+"."

 else:

 cmod=cmod+x

 return cmod

# Pour qui connaît la commande replace, on écrit simplement :

# return c.replace(',','.')

### Analyse des données du fichier

f = open("MesureBrute.txt")

temps=[]

listv=[]

listv\_plateau=[]

for k in range(22): # saut de 22 lignes

 f.readline()

for ligne in f :

 lignemod=formate(ligne)

 t=lignemod[0:8]

 tf=float(t)

 temps.append(tf)

 v=lignemod[9:9+9]

 vf=float(v)

 listv.append(vf)

# Problème pour récupérer les vitesses du plateau car elles ont 8 caractères quand elles sont positives, 9 quand elles sont négatives. Solution : utiliser la commande split (vue en sup). Après la ligne 8, insérer :

"""

liste\_3donnees=formate(ligne.split('\t'))

listv\_plateau.append(float(liste\_3donnees[2]))

"""

### Intégration numérique par la méthode des rectangles (à gauche)

def int\_rect(listx,listv,y0):

 """

 Retourne une liste de couple[listx,listy]

 où listy contient une intégration du signal numérique

 """

 listy=[y0]

 n=len(listx)

 for i in range(0,n-1):

 y=listy[-1]+(listx[i+1]-listx[i])\* listv[i]

 # avec la méthode des trapèzes, écrire :

 # y=listy[-1]+(listx[i+1]-listx[i])\* (listv[i]+listv[i+1])/2

 listy.append(y)

 return listy

### Test :

import pylab as pl

pl.close()

pl.figure()

pl.plot(temps,int\_rect(temps,listv,0), label="déplacement du maneton en fonction du temps")

pl.legend()

"""

pl.figure()

pl.plot(temps,int\_rect(temps,listv\_plateau,0), label="déplacement du plateau en fonction du temps")

pl.legend()

"""

### Lissage

def val\_moy\_n(temps,tab,m):

 """

 Retourne une liste de couple[temps,tab\_f]

 ou tab\_f est une valeur moyenne de tab sur m éléments

 """

 tab\_f=[]

 for i in range(0,len(temps)-m,m):

 tmp=0

 for j in range(i,i+m):

 tmp = tmp+tab[j]

 tab\_f.append([temps[i],tmp/n])

 return tab\_f

##################################################################################

**### Correction du Sudoku**

def affiche(s):

 for i in range(9):

 print(s[9\*i:9\*i+9])

def possibles(k,s):

 i,j=divmod(k,9)

 l=27\*(i//3)+3\*(j//3)

 p=s[9\*i:9\*i+9]+ s[j:80:9]+s[l:l+3]+s[l+9:l+12]+s[l+18:l+21]

 return [x for x in range(1,10) if x not in p]

def soltou(s):

 if 0 not in s: #cas de base : s est maximal

 affiche(s) # donc on l'affiche

 else:

 k=0 #on calcule l'index du premier 0 de s

 while s[k]!=0:

 k+=1

 #remarque : k=s.index(0) donne directement l'index du premier 0 de s

 p=possibles(k,s) # liste des valeurs possibles de s[k]

 for i in p: #exploration

 s[k]=i

 soltou(s)

 s[k]=0 #retour sur trace (backtracking)

### Test de soltou

sudoku=[8]+[0]\*10+[3,6]+[0]\*6+[7,0,0,9,0,2,0,0,0,5,0,0,0,7]+[0]\*7+[4,5,7]+[0]\*5+[1,0,0,0,3,0,0,0,1,0,0,0,0,6,8,0,0,8,5,0,0,0,1,0,0,9,0,0,0,0,4,0,0]

soltou(sudoku)

### Fonction modifiée sol1

def sol1(s):

 if 0 not in s: #cas de base : s est maximal

 affiche(s) # donc on l'affiche

 else:

 k=0 #on calcule l'index du premier 0 de s

 while s[k]!=0:

 k+=1

 #remarque : k=s.index(0) donne directement l'index du premier 0 de s

 p=possibles(k,s) # liste des valeurs possibles de s[k]

 for i in p: #exploration

 s[k]=i

 sol1(s)

 if 0 not in s:

 break

 s[k]=0 #retour sur trace (backtracking)

sol1(sudoku)