```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 27 10:44:19 2016

@author: Dom
"""

import numpy.fft as npf
import matplotlib.pyplot as plt
import numpy as np
import random as rd

###################################
# Configuration

N=2**10 # nbre d'échantillons
Duree=10 # Durée du signal -> Fe = N/Duree
t=np.linspace(0,Duree,N) # liste de la variable temps
Dt=Duree/N # pas temporel


#####################################################################
# Quelques signaux qui peuvent servir ########################

# Impulsion
def impuls():
    'crée un signal impulsionnel'
    x=[0 for n in t]
    x[0]=1
    return x

#plt.plot(t,impuls())

# Peigne
def peigne(T=1):
    'crée un peigne de Dirac d"amplitude 1 de période T'
    x=[0 for n in t]
    for i in range(int(Duree/T)):
        x[int(i*T/Dt)]=1
    return x
#plt.plot(t,peigne())

# Bruit
def bruit(Amp=1):
    'engendre un signal bruité d"amplitude max Amp'
    x=[rd.uniform(-Amp,Amp) for i in t]
    return(x)
# C'est pas la peine de tracer du bruit, y'en a assez comme ça !

# Sinus
def sinus(F=1,Amp=1):
    return [Amp*np.sin(2*np.pi*z*F) for z in t]

plt.plot(t,sinus(F=1,Amp=1))

# Cosinus
def cosinus(F=1,Amp=1):
    return [Amp*np.cos(2*np.pi*z*F) for z in t]

plt.plot(t,cosinus(F=1,Amp=1))
```

```
# Sinus redressé
def sinr(F=1,Amp=1):
    return [Amp*abs(np.sin(np.pi*z*F)) for z in t]

plt.plot(t,sinr(F=1,Amp=1))

# Triangle
def triangle(F=1,Amp=1):
    return [Amp*np.arcsin(np.sin(2*np.pi*z*F)) for z in t]

plt.plot(t,triangle(F=1,Amp=1))

# Créneau
def creneau(F=1,Amp=1):
    'génère un signal créneau symétrique'
    x=[]
    for n in t:
        if F*n-int(F*n) < .5:
            x.append(Amp)
        else:
            x.append(-Amp)
    return x

plt.plot(t,creneau(F=1,Amp=1))

# Créneau à rapport cyclique a
def creneau_a(alpha=1,F=1,Amp=1):
    'génère un signal créneau dissymétrique, de rapport alpha'
    x=[]
    for n in t:
        if F*n-int(F*n) < alpha:
            x.append(Amp)
        else:
            x.append(-Amp)
    return x

plt.plot(t,creneau_a(alpha=0.3,F=1,Amp=1))

# Rampe
def rampe(F=1,Amp=1):
    return [(n*F-int(n*F))*Amp for n in t]

plt.plot(t,rampe(F=1,Amp=1))

# Sinus amorti
def sin_amorti(tau=1,F=1,Amp=1):
    return [np.exp(-n/tau)*np.cos(2*np.pi*n*F)*Amp for n in t]

plt.plot(t,sin_amorti(tau=2,F=1,Amp=1))

####################################
# Filtres ##########################

# Dérivateur
def deriv(signal):
    ' dérive les valeurs du signal de type liste'
    long=len(signal)
    y=[signal[0]]
    for i in range(1,long):
```

```
        y.append(signal[i]-signal[i-1])
    return y

# Intégrateur
def integr(signal):
    'intègre le signal'
    s=0
    y=[]
    for i in signal:
        s+=i
        y.append(s)
    return y

# Amplificateur
def ampli(signal,A):
    'amplifie le signal par A'
    return [A*x for x in signal]

# Décaler
def decal(signal,s=0):
    'ajoute s au signal'
    return [x+s for x in signal]

# Moyenneur
def moyenne(signal):
    'moyenne le signal sur 4 valeurs glissantes'
    long=len(signal)
    y=[signal[0]/4,(signal[0]+signal[1])/4,(signal[0]+signal[1]+signal[2])/4]
    for i in range(3,long):
        y.append((signal[i]+signal[i-1]+signal[i-2]+signal[i-3])/4)
    return y

# Sommateur
def somme(signal1,signal2):
    'somme les deux signaux'
    s=[]
    for n in range(len(signal1)):
        s.append(signal1[n]+signal2[n])
    return s

# Médiane
def mediane(signal):
    'extrait la médiane de cinq valeurs glissantes'
    N=len(signal)
    x=[]
    for i in range(N-5):
        x.append(np.median(signal[i:i+5]))
    for i in range(5):
        x.append(np.median(signal[N-6+i:N-1]))
    return x

# Passe-bas
def passe_bas(signal,Fc=20):
    'filtre passe-bas équivalent à un premier ordre analogique'
    long=len(signal)
    y=[(signal[0]*2*np.pi*Fc*t[1])/(1+2*np.pi*Fc*t[1])]
    for n in range(1,long):
        y.append((signal[n]*2*np.pi*Fc*t[1]+y[n-1])/(1+2*np.pi*Fc*t[1]))
    return y
```

```
#####################################
zero=[0 for n in t]

# Trace les signaux x (et y) sur une duree D, tmax par défaut

def trace(x,y=zero,D=Duree):
    'trace les signaux x et y de type liste'
    N=len(t)
    Fe=N/t[N-1]
    f=np.linspace(0,Fe//2,N//2)
    zx=npf.fft(x)
    # Tracé des signaux
    plt.clf()
    # Signaux
    plt.subplot(211)
    plt.tight_layout()
    xmin,xmax=min(x),max(x) # x min et max, pour les échelles des axes
    plt.axis([0,D,xmin-.5,xmax+.5])
    tt=t[0:int(N//Duree*D)]
    xx=x[0:int(N//Duree*D)]
    #plt.title('Signaux')
    plt.xlabel('Temps en s')
    s1=plt.plot(tt,xx,color='blue',label='Entrée')
    #plt.legend(s1,'Signal d"entrée')
    if y != zero:
        ymin,ymax=min(y),max(y)
        plt.axis([0,D,min(xmin,ymin)-.5,max(xmax,ymax)+.5])
        yy=y[0:int(N//Duree*D)]
        plt.plot(tt,yy,color='red',label='Sortie')
    plt.grid(True)
    #plt.legend(loc='lower right')
    # Spectres
    plt.subplot(212)
    plt.tight_layout()
    plt.title('Spectres')
    plt.xlabel('Fréquences')
    plt.ylabel('Amplitudes')
    if max(abs(zx[1:]))>200:
        maxi=200
    else:
        maxi=max(abs(zx[1:]))
    plt.axis([-1,Fe//2,0,1+maxi])
    plt.plot(f,abs(zx[0:N//2]),color='blue',label='Entrée')
    if y != zero:
        zy=npf.fft(y)
        plt.plot(f,abs(zy[0:N//2]),color='red',label='Sortie')
        if max(max(abs(zx[1:])),max(abs(zy[1:])))> 200:
            maxi=200
        else:
            maxi=max(max(abs(zx[1:])),max(abs(zy[1:])))
        plt.axis([-1,Fe//2,0,1+maxi])
    plt.legend(loc='upper right')
    plt.grid(True)
    plt.show()


x1=somme(cosinus(F=20,Amp=1),cosinus(F=30,Amp=1))
x2=somme(cosinus(F=2,Amp=1),bruit(Amp=1))
x=somme(x1,x2)
trace(x,y=zero,D=Duree)
```

```
xf=passe_bas(x,Fc=2)
trace(x,y=xf,D=Duree)

# Tracé en ordonnée logarithmique pour les spectres
def tracelog(x,y=zero,D=Duree):
    'trace les signaux x et y de type liste'
    N=len(t)
    Fe=N/t[N-1]
    f=np.linspace(0,Fe//2,N//2)
    zx=npf.fft(x)
    # Tracé des signaux
    plt.clf()
    # Signaux
    plt.subplot(211)
    plt.tight_layout()
    xmin,xmax=min(x),max(x) # x min et max, pour les échelles des axes
    plt.axis([0,D,xmin-.5,xmax+.5])
    tt=t[0:int(N//Duree*D)]
    xx=x[0:int(N//Duree*D)]
    plt.title('Signaux')
    plt.xlabel('Temps')
    s1=plt.plot(tt,xx,color='blue',label='Entrée')
    plt.legend(s1,'Signal d"entrée')
    if y != zero:
        ymin,ymax=min(y),max(y)
        plt.axis([0,D,min(xmin,ymin)-.5,max(xmax,ymax)+.5])
        yy=y[0:int(N//Duree*D)]
        plt.plot(tt,yy,color='red',label='Sortie')
    plt.grid(True)
    plt.legend(loc='lower right')
    # Spectres
    plt.subplot(212)
    plt.tight_layout()
    plt.semilogy()
    plt.title('Spectres')
    plt.xlabel('Fréquences')
    plt.ylabel('Amplitudes')
    if max(abs(zx[1:]))>200:
        maxi=200
    else:
        maxi=max(abs(zx[1:]))
    plt.axis([-1,Fe//2,0,1+maxi])
    plt.plot(f,abs(zx[0:N//2]),color='blue',label='Entrée')
    if y != zero:
        zy=npf.fft(y)
        plt.plot(f,abs(zy[0:N//2]),color='red',label='Sortie')
        if max(max(abs(zx[1:])),max(abs(zy[1:])))> 200:
            maxi=200
        else:
            maxi=max(max(abs(zx[1:])),max(abs(zy[1:])))
        plt.axis([-1,Fe//2,0,1+maxi])
    plt.legend(loc='upper right')
    plt.grid(True)
    plt.show()



################################################################################"
# Fonction de transfert - transmittance
```

```
def H(filtre):
    'trace la transmittance du filtre'
    N=len(t)
    Fe=N/t[N-1]
    f=np.linspace(0,Fe//2,N//2)
    y=filtre(impuls())
    zy=npf.fft(y)
    plt.clf()
    plt.xlabel('Fréquences')
    plt.ylabel('Amplitudes')
    plt.plot(f,abs(zy[0:N//2]),color='green',label='|H(f)|',linewidth=2)
    plt.axis([0,Fe//2,0,.2+max(abs(zy[1:]))])
    plt.legend(loc='upper right')
    plt.grid(True)
    plt.show()

H(passe_bas)
```