FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Procesamiento de señales, fundamentos

Maestría en sistemas embebidos
Universidad de Buenos Aires
MSE 5Co2020

Clase 4 - Euler | Fourier - IDFT

Ing. Pablo Slavkin
slavkin.pablo@gmail.com
wapp:011-62433453

Síntesis

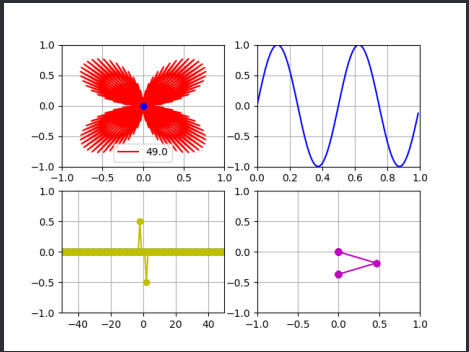
Como reconstruyo la señal en el tiempo

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
#---------------------------------------
fig         = plt.figure()
fs          = 100
N           = 100
#---------------------------------------
circleAxe   = fig.add_subplot(2,2,1)
circleLn,massLn,  = plt.plot([],[],'r-',[],[],'bo')
circleAxe.grid(True)
circleAxe.set_xlim(-1,1)
circleAxe.set_ylim(-1,1)
circleFrec = np.arange(-fs/2,fs/2,fs/N)
circleLn.set_label(circleFrec[0])
circleLg=circleAxe.legend()
circleData = []
mass        = 0
frecIter    = 0
def circle(f,n):
    return np.exp(-1j*2*np.pi*f*n*1/fs)
def circleInv(f,n,c):
    return c*np.exp(-1j*2*np.pi*f*n*1/fs)
#---------------------------------------
signalAxe   = fig.add_subplot(2,2,2)
signalLn,   = plt.plot([],[],'b-')
signalAxe.grid(True)
signalAxe.set_xlim(0,N/fs)
signalAxe.set_ylim(-1,1)
signalFrec = 2
signalData=[]
def signal(f,n):
    return np.sin(2*np.pi*f*n*1/fs)
    return np.sin(2*np.pi*f*n*1/fs)
#---------------------------------------
promAxe  = fig.add_subplot(2,2,3)
promRLn,promILn,  = plt.plot([],[],'g-o',[],[],'y-o')
promAxe.grid(True)
promAxe.set_xlim(-fs/2,fs/2)
promAxe.set_ylim(-1,1)
promData=np.zeros(N,dtype=complex)
#---------------------------------------
inversaAxe      = fig.add_subplot(2,2,4)
inversaLn, = plt.plot([],[],'m-o')
inversaAxe.grid(True)
inversaAxe.set_xlim(-1,1)
inversaAxe.set_ylim(-1,1)
inversaData = []
#---------------------------------------
tData=np.arange(0,N/fs,1/fs)
def init():
    return circleLn,circleLg,signalLn,massLn,promRLn,promILn,inversaLn
def updateF(n):
    global promData,fData,inversaData,frecIter
    if aniT.repeat==True:
        return inversaLn,
    inversaData=[0]
    for f in range(N):
        inversaData.append(inversaData[-1]+circleInv(circleFrec[f],frecIter,promData[f]))
    inversaLn.set_data(np.imag(inversaData),np.real(inversaData))
    frecIter+=1
    if frecIter==N:
        frecIter=0
    return inversaLn,
```

Síntesis

Como reconstruyo la señal en el tiempo

```
def updateT(nn):
    global circleData,signalData,promData,frecIter,circleFrec,circleLg

    circleData = []
    signalData = []
    for n in range(N):
        circleData.append(circle(circleFrec[frecIter],n)*signal(signalFrec,n))
        mass=np.average(circleData)
        signalData.append(signal(signalFrec,n))
        promData[frecIter]=mass
    massLn.set_data(np.real(mass),
                    np.imag(mass))
    circleLn.set_data(np.real(circleData),
                      np.imag(circleData))
    signalLn.set_data(tData[:n+1],signalData)
    promRLn.set_data(circleFrec[:frecIter+1],np.real(promData[:frecIter+1]))
    promILn.set_data(circleFrec[:frecIter+1],np.imag(promData[:frecIter+1]))
    circleLn.set_label(circleFrec[frecIter])
    circleLg=circleAxe.legend()

    if frecIter == N-1:
        aniT.repeat=False
    else:
        frecIter+=1
    return circleLn,circleLg,signalLn,massLn,promRLn,promILn,

aniT=FuncAnimation(fig,updateT,N,init,interval=10  ,blit=True,repeat=True)
aniF=FuncAnimation(fig,updateF,N,init,interval=30 ,blit=True,repeat=True)
plt.show()
```



Síntesis, Transformada inversa discreta de Fourier

IDFT

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0, \ldots, N-1$$

IDFT

Señales complejas como entrada?

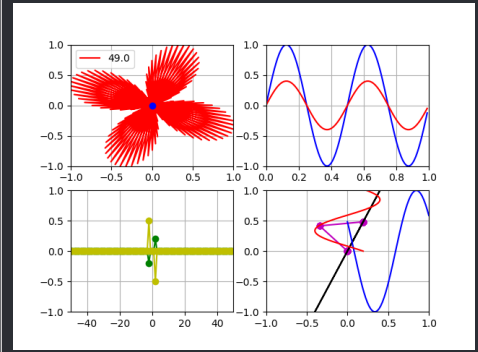
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
#--------------------------------------
fig         = plt.figure()
fs          = 100
N           = 100
#--------------------------------------
circleAxe  = fig.add_subplot(2,2,1)
circleLn,massLn,  = plt.plot([],[],'r-',[],[],'bo')
circleAxe.grid(True)
circleAxe.set_xlim(-1,1)
circleAxe.set_ylim(-1,1)
circleFrec = np.arange(-fs/2,fs/2,fs/N)
circleLn.set_label(circleFrec[0])
circleLg=circleAxe.legend()
circleData = []
mass       = 0
frecIter   = 0
def circle(f,n):
    return np.exp(-1j*2*np.pi*f*n*1/fs)
def circleInv(f,n,c):
    return c*np.exp(-1j*2*np.pi*f*n*1/fs)
#--------------------------------------
signalAxe  = fig.add_subplot(2,2,2)
signalRLn,signalILn  = plt.plot([],[],'b-',[],[],'r-')
signalAxe.grid(True)
signalAxe.set_xlim(0,N/fs)
signalAxe.set_ylim(-1,1)
signalFrec = 2
signalData=[]
def signal(f,n):
    return np.sin(2*np.pi*f*n*1/fs)+0.4j*np.sin(2*np.pi*f*n*1/fs)
```

```
    return np.sin(2*np.pi*f*n*1/fs)+0.4j*np.sin(2*np.pi*f*n*1/fs)
#--------------------------------------
promAxe  = fig.add_subplot(2,2,3)
promRLn,promILn,  = plt.plot([],[],'g-o',[],[],'y-o')
promAxe.grid(True)
promAxe.set_xlim(-fs/2,fs/2)
promAxe.set_ylim(-1,1)
promData=np.zeros(N,dtype=complex)
#--------------------------------------
inversaAxe          = fig.add_subplot(2,2,4)
inversaLn,penLn,penRLn,penILn = plt.plot([],[],'m-o',[],[],'k-',[],[],'b-',[],[],'r-')
inversaAxe.grid(True)
inversaAxe.set_xlim(-1,1)
inversaAxe.set_ylim(-1,1)
inversaData,penData= [],[]
#--------------------------------------
tData=np.arange(0,N/fs,1/fs)

def init():
    return circleLn,circleLg,signalRLn,signalILn,massLn,promRLn,promILn,inversaLn,penILn,penRLn,

def updateF(n):
    global promData,fData,vectorData,frecIter,penData
    if aniT.repeat==True:
        return inversaLn,
    vectorData=[0]
    for f in range(N):
        vectorData.append(vectorData[-1]+circleInv(circleFrec[f],frecIter,promData[f]))
    inversaLn.set_data(np.imag(vectorData),np.real(vectorData))
    penData.insert(0,vectorData[-1])
    traceData=penData[0:N//2]
    t=np.linspace(0,1,len(traceData))
    penRLn.set_data(t,np.real(traceData))
    penILn.set_data(np.imag(traceData),t)
```

IDFT

Señales complejas como entrada?

```
        penILn.set_data(np.imag(traceData),t)
        penLn.set_data(np.imag(penData),np.real(penData))
        frecIter+=1
        if frecIter==N:
            frecIter=0
        return inversaLn,penLn,penILn,penRLn,

    def updateT(nn):
        global circleData,signalData,promData,frecIter,circleFrec,circleLg

        circleData = []
        signalData = []
        for n in range(N):
            circleData.append(circle(circleFrec[frecIter],n)*signal(signalFrec,n))
            mass=np.average(circleData)
            signalData.append(signal(signalFrec,n))
            promData[frecIter]=mass
        massLn.set_data(np.real(mass),
                        np.imag(mass))
        circleLn.set_data(np.real(circleData),
                          np.imag(circleData))
        signalRLn.set_data(tData[:n+1],np.real(signalData))
        signalILn.set_data(tData[:n+1],np.imag(signalData))
        promRLn.set_data(circleFrec[:frecIter+1],np.real(promData[:frecIter+1]))
        promILn.set_data(circleFrec[:frecIter+1],np.imag(promData[:frecIter+1]))
        circleLn.set_label(circleFrec[frecIter])
        circleLg=circleAxe.legend()

        if frecIter == N-1:
            aniT.repeat=False
        else:
            frecIter+=1
        return circleLn,circleLg,signalRLn,signalILn,massLn,promRLn,promILn,

    aniT=FuncAnimation(fig,updateT,N,init,interval=10  ,blit=True,repeat=True)
    aniF=FuncAnimation(fig,updateF,N,init,interval=30 ,blit=True,repeat=True)
    plt.show()
```



IDFT

Un conejo como entrada?

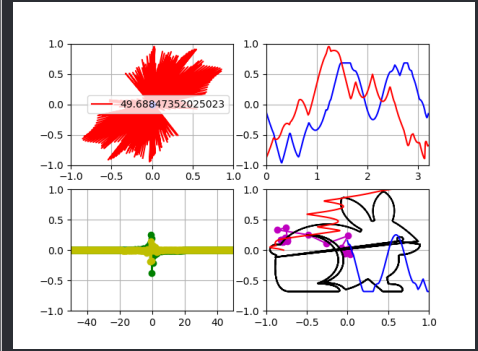
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
#-------------------------------------
fig         = plt.figure()
fs          = 100
N           = 100
#-------------------------------------
conejo=np.load("conejo.npy")[::1]
N=len(conejo)
def signal(f,n):
    return conejo[n]
#-------------------------------------
circleAxe  = fig.add_subplot(2,2,1)
circleLn,massLn,  = plt.plot([],[],'r-',[],[],'bo')
circleAxe.grid(True)
circleAxe.set_xlim(-1,1)
circleAxe.set_ylim(-1,1)
circleFrec = np.arange(-fs/2,fs/2,fs/N)
circleLn.set_label(circleFrec[0])
circleLg=circleAxe.legend()
circleData = []
mass       = 0
frecIter   = 0
def circle(f,n):
    return np.exp(-1j*2*np.pi*f*n*1/fs)
def circleInv(f,n,c):
    return c*np.exp(-1j*2*np.pi*f*n*1/fs)
#-------------------------------------
signalAxe  = fig.add_subplot(2,2,2)
signalRLn,signalILn  = plt.plot([],[],'b-',[],[],'r-')
signalAxe.grid(True)
signalAxe.set_xlim(0,N/fs)
```

```
signalAxe.set_xlim(0,N/fs)
signalAxe.set_ylim(-1,1)
signalFrec = 2
signalData=[]
#def signal(f,n):
#    return np.sin(2*np.pi*f*n*1/fs)+0.4j*np.sin(2*np.pi*f*n*1/fs)
#-------------------------------------
promAxe  = fig.add_subplot(2,2,3)
promRLn,promILn,  = plt.plot([],[],'g-o',[],[],'y-o')
promAxe.grid(True)
promAxe.set_xlim(-fs/2,fs/2)
promAxe.set_ylim(-1,1)
promData=np.zeros(N,dtype=complex)
#-------------------------------------
inversaAxe          = fig.add_subplot(2,2,4)
inversaLn,penLn,penRLn,penILn = plt.plot([],[],'m-o',[],[],'k-',[],[],'b-',[],[],'r-')
inversaAxe.grid(True)
inversaAxe.set_xlim(-1,1)
inversaAxe.set_ylim(-1,1)
inversaData,penData= [],[]
#-------------------------------------
tData=np.arange(0,N/fs,1/fs)

def init():
    return circleLn,circleLg,signalRLn,signalILn,massLn,promRLn,promILn,inversaLn,penILn,penRLn,

def updateF(n):
    global promData,fData,vectorData,frecIter,penData
    if aniT.repeat==True:
        return inversaLn,
    vectorData=[0]
    for f in range(N):
        vectorData.append(vectorData[-1]+circleInv(circleFrec[f],frecIter,promData[f]))
    inversaLn.set_data(np.imag(vectorData),np.real(vectorData))
```

IDFT

Un conejo como entrada?

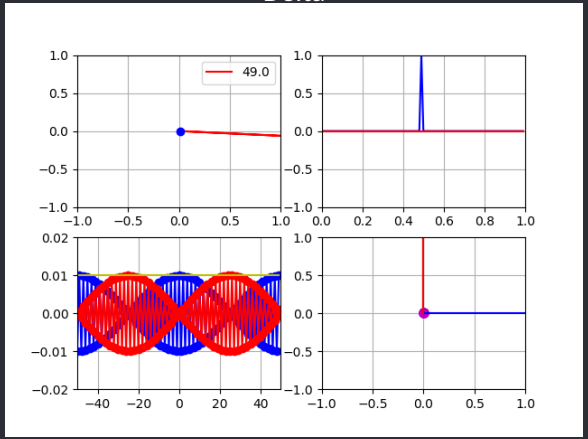
```
        inversaLn.set_data(np.imag(vectorData),np.real(vectorData))
        penData.insert(0,vectorData[-1])
        traceData=penData[0:N//2]
        t=np.linspace(0,1,len(traceData))
        penRLn.set_data(t,np.real(traceData))
        penILn.set_data(np.imag(traceData),t)
        penLn.set_data(np.imag(penData),np.real(penData))
        frecIter+=1
        if frecIter==N:
            frecIter=0
        return inversaLn,penLn,penILn,penRLn,

    def updateT(nn):
        global circleData,signalData,promData,frecIter,circleFrec,circleLg

        circleData = []
        signalData = []
        for n in range(N):
            circleData.append(circle(circleFrec[frecIter],n)*signal(signalFrec,n))
            mass=np.average(circleData)
            signalData.append(signal(signalFrec,n))
            promData[frecIter]=mass
        massLn.set_data(np.real(mass),
                        np.imag(mass))
        circleLn.set_data(np.real(circleData),
                          np.imag(circleData))
        signalRLn.set_data(tData[:n+1],np.real(signalData))
        signalILn.set_data(tData[:n+1],np.imag(signalData))
        promRLn.set_data(circleFrec[:frecIter+1],np.real(promData[:frecIter+1]))
        promILn.set_data(circleFrec[:frecIter+1],np.imag(promData[:frecIter+1]))
        circleLn.set_label(circleFrec[frecIter])
        circleLg=circleAxe.legend()

        if frecIter == N-1:
            aniT.repeat=False
        else:
            frecIter+=1
        return circleLn,circleLg,signalRLn,signalILn,massLn,promRLn,promILn,

    aniT=FuncAnimation(fig,updateT,N,init,interval=10  ,blit=True,repeat=True)
    aniF=FuncAnimation(fig,updateF,N,init,interval=30 ,blit=True,repeat=True)
    plt.show()
```



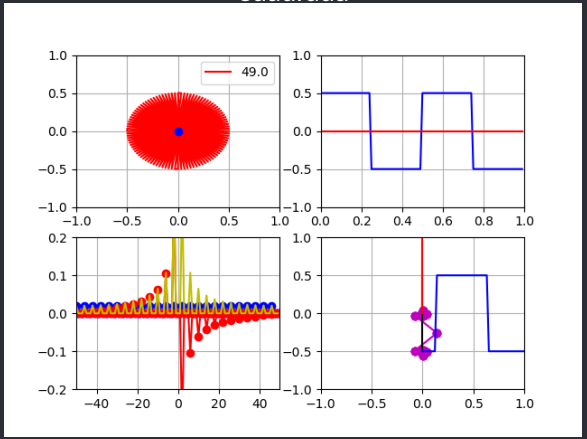
DFT<>IDFT

Transformadas relevantes

Delta



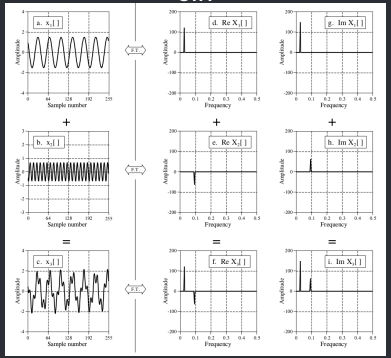
Cuadrada



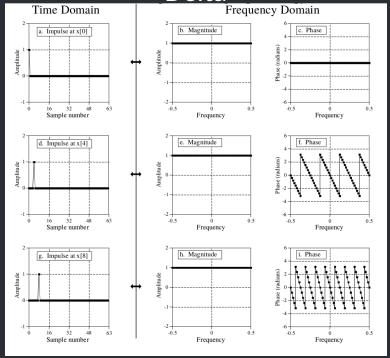
DFT<>IDFT

Transformadas relevantes

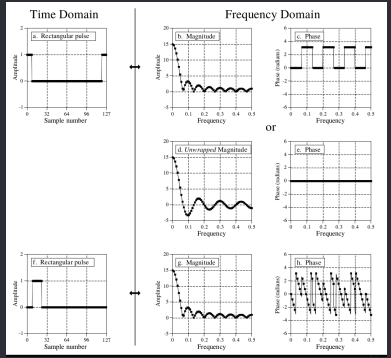
Sin



Delta



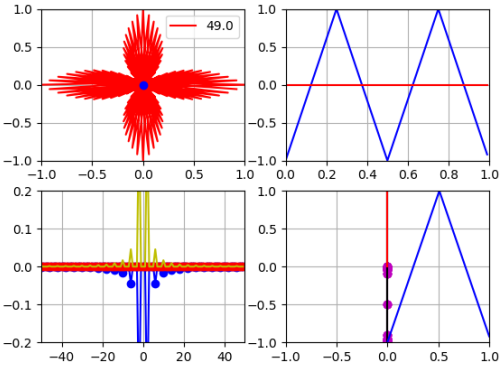
Cuadrada



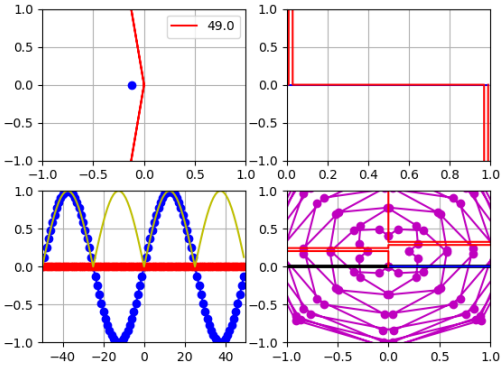
DFT<>IDFT

Transformadas relevantes

Triangular

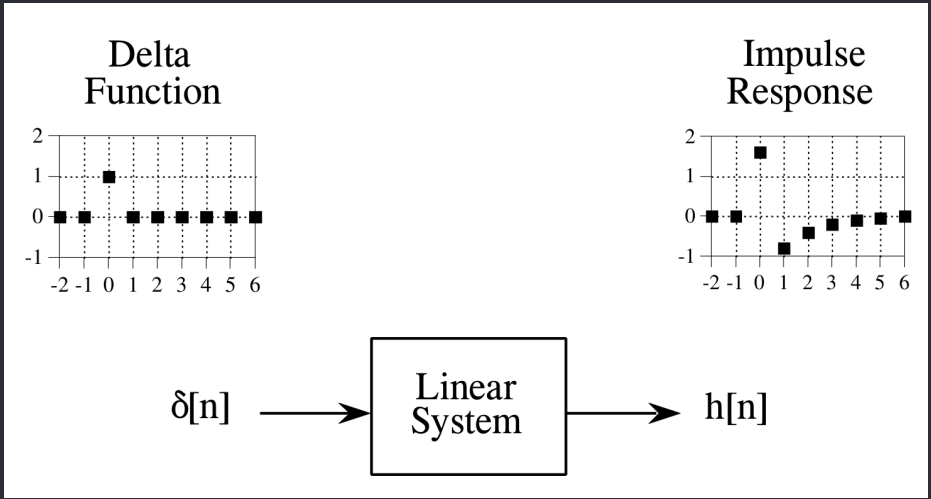


Conjugado



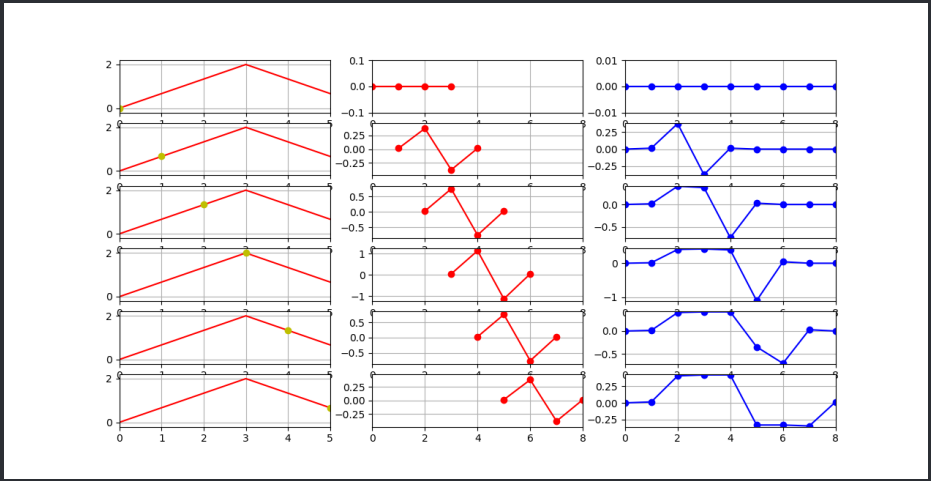
Funcion delta

Respuesta al impulso



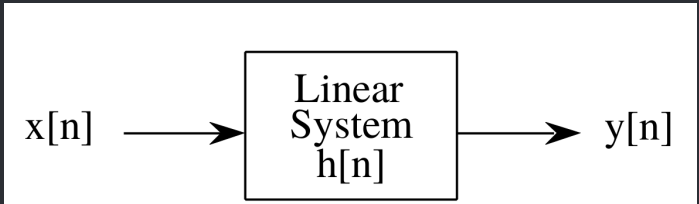
Convolucion

Descomposicion Delta



Funcion delta

Respuesta al impulso



$$x[n] * h[n] = y[n]$$

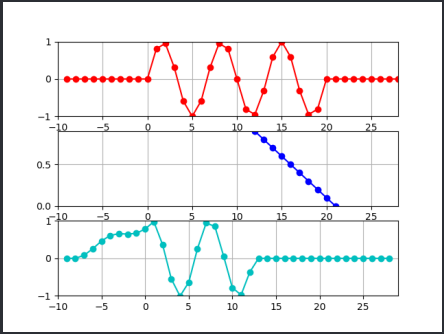
$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Convolucion

Respuesta al impulso

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.animation import FuncAnimation
#--------------------------------------
fig = plt.figure()
fs  = 20
N   = 20
M=10
#--------------------------------------
xFrec = 3
def x(f,n):
    return np.sin(2*np.pi*f*n*1/fs)
tData=np.arange(-(M-1),N+(M-1),1)
xData=np.zeros(N+2*(M-1))
xData[M-1:M-1+N]=x(xFrec,tData[M-1:M-1+N])
xAxe  = fig.add_subplot(3,1,1)
xLn,xHighLn  = plt.plot(tData,xData,'r-o',[],[],'y-
        o')
xAxe.grid(True)
xAxe.set_xlim(-M,M+N-2)
xAxe.set_ylim(np.min(xData),np.max(xData))
#--------------------------------------
hData=[0.1*n for n in range(M)]
hAxe  = fig.add_subplot(3,1,2)
hLn,  = plt.plot([],[],'b-o')
```

```
hAxe.grid(True)
hAxe.set_xlim(-M,M+N-2)
hAxe.set_ylim(np.min(hData),np.max(hData))
#--------------------------------------
yAxe  = fig.add_subplot(3,1,3)
yLn,  = plt.plot([],[],'c-o')
yAxe.grid(True)
yAxe.set_xlim(-M,M+N-1)
yAxe.set_ylim(np.min(xData),np.max(xData))
yData=[]
#--------------------------------------
def init():
    global yData
    yData=np.zeros(N+2*(M-1))
    return hLn,xLn,xHighLn,yLn,

def update(i):
    global yData
    t=np.linspace(-(M-1)+i,i,M,endpoint=True)
    yData[i]=np.sum(xData[i:i+M]*hData[::-1])
    xHighLn.set_data(t,xData[i:i+M])
    hLn.set_data(t,hData[::-1])
    yLn.set_data(tData,yData)
    return hLn,xLn,xHighLn,yLn,

ani=FuncAnimation(fig,update,M+N-1,init,interval
        =1000 ,blit=True,repeat=True)
plt.show()
```

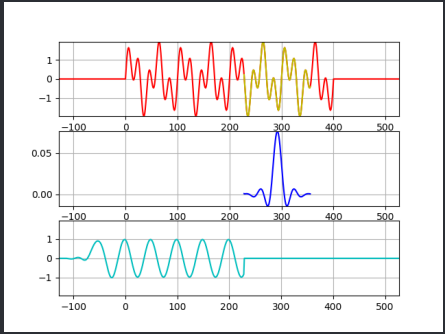


Filtrado

Pasa bajos

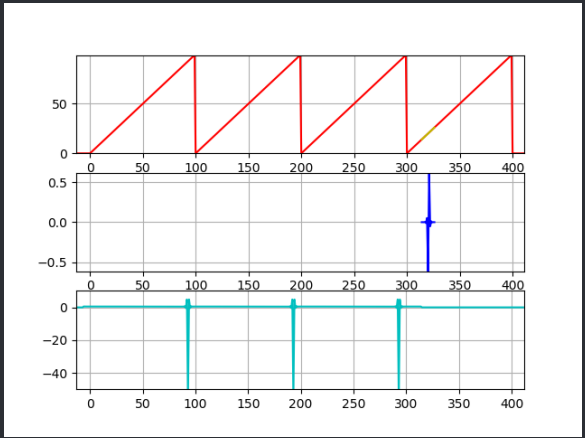
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.animation import FuncAnimation
#-------------------------------------
fig = plt.figure()
fs  = 100
N   = 400
fir,=np.load("low_pass.npy").astype(float)
M=len(fir)
#fir,=np.load("diferenciador.npy").astype(float)
#M=len(fir)
#-------------------------------------
def x(f,n):
    return np.sin(2*np.pi*2*n*1/fs)+\
           np.sin(2*np.pi*5*n*1/fs)

xFrec = 3
tData=np.arange(-(M-1),N+(M-1),1)
xData=np.zeros(N+2*(M-1))
xData[M:M+N]=x(xFrec,tData[M:M+N])
xAxe  = fig.add_subplot(3,1,1)
xLn,xHighLn  = plt.plot(tData,xData,'r-',[],[],'y-'
               )
xAxe.grid(True)
xAxe.set_xlim(-M,M+N-2)
xAxe.set_ylim(np.min(xData),np.max(xData))
#-------------------------------------
hData=fir
```

```
hAxe  = fig.add_subplot(3,1,2)
hLn,  = plt.plot([],[],'b-')
hAxe.grid(True)
hAxe.set_xlim(-M,M+N-2)
hAxe.set_ylim(np.min(hData),np.max(hData))
#-------------------------------------
yAxe  = fig.add_subplot(3,1,3)
yLn,  = plt.plot([],[],'c-')
yAxe.grid(True)
yAxe.set_xlim(-M,M+N-1)
yAxe.set_ylim(np.min(xData),np.max(xData))
yData=[]
#-------------------------------------
def init():
    global yData
    yData=np.zeros(N+2*(M-1))
    return hLn,xLn,xHighLn,yLn,

def update(i):
    global yData
    t=np.linspace(-(M-1)+i,i,M,endpoint=True)
    yData[i]=np.sum(xData[i:i+M]*hData[::-1])
    xHighLn.set_data(t,xData[i:i+M])
    hLn.set_data(t,hData[::-1])
    yLn.set_data(tData,yData)
    return hLn,xLn,xHighLn,yLn,

ani=FuncAnimation(fig,update,M+N-1,init,interval=10
        ,blit=True,repeat=True)
plt.get_current_fig_manager().window.showMaximized
        ()
plt.show()
```



Filtrado

diferenciador



Bibliografía

Libros, links y otro material

[1] *ARM CMSIS DSP*.
link

[2] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. Second Edition, 1999.

[3] *Interactive Mathematics Site Info*.

[4] *Grant Sanderson*
link

[5] *Interactive Mathematics Site Info*.
link