

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



2020-05-22

Procesamiento de señales, fundamentos


 **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



repaso DFT

$e^{j2\pi ft}$ modulado por $\sin(t)$ y centro de masas en f , DFT?



2020-05-22

Procesamiento de señales, fundamentos

└ repaso DFT

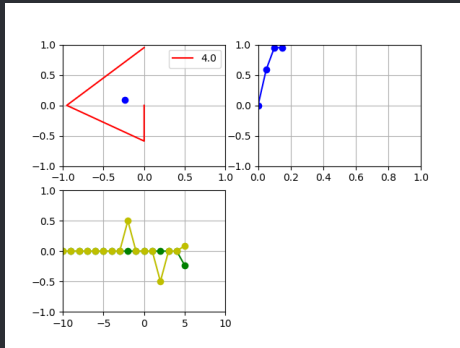


```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure

#-----
fig = plt.figure()
fs = 50
N = 50
#-----CONJUGADO-----
#conjugado=np.zeros(100,dtype=complex)
#conjugado+=0.2
#N=len(conjugado)
#conjugado[6]=0.5*N
#conjugado[100-6]=0.5*N
#def signal(f,n):
#    return conjugado[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)
#-----
signalAxe = fig.add_subplot(2,2,2)
signalLn, = plt.plot([],[],'b-o')
signalAxe.grid(True)
signalAxe.set_xlim(0,N/fs)
signalAxe.set_ylim(-1,1)
signalLn.set_label('')
def signal(f,n):
    return np.cos(2*np.pi*f*n*1/fs)
```

```
    return np.cos(2*np.pi*f*n*1/fs)
#-----
promAxe = fig.add_subplot(2,2,3)
promRLn,promILn,promMagLn,promPhaseLn = plt.plot([],[],'b-o',
    [],[],'r-o',[],[],'k-o',[],[],'y-')
promAxe.grid(True)
promAxe.set_xlim(-fs/2,fs/2)
promAxe.set_ylim(-1,1)
promData=np.zeros(N,dtype=complex)
#-----
tData=np.arange(0,N/fs,1/fs)
def init():
    return circleLn,circleLg,signalLn,massLn,promRLn,promILn
def update(n):
    global circleData,signalData,promData,frecIter,circleFrec,
        circleLg
    circleData.append(circle(circleFrec[frecIter],n)*signal(
        signalFrec,n))
    mass=np.average(circleData)
    massLn.set_data(np.real(mass),
        np.imag(mass))
    circleLn.set_data(np.real(circleData),
        np.imag(circleData))
    signalData.append(signal(signalFrec,n))
    signalLn.set_data(tData[:n+1],signalData)
    promData[frecIter]=mass
    promRLn.set_data(circleFrec[:frecIter+1],np.real(promData[:
        frecIter+1]))
    promILn.set_data(circleFrec[:frecIter+1],np.imag(promData[:
        frecIter+1]))
    # promMagLn.set_data(circleFrec[:frecIter+1],np.abs(promData
        [:frecIter+1])**2)
    # promPhaseLn.set_data(circleFrec[:frecIter+1],np.angle(
        promData[:frecIter+1])/np.pi)
    circleLn.set_label(circleFrec[frecIter])
    circleLg=figureAxe.legend()
    if n==N-1:
        circleData = []
        signalData = []
        if frecIter == N-1:
            ani.repeat=False
        else:
            frecIter+=1
    return circleLn,circleLg,signalLn,massLn,promRLn,promILn,
        promMagLn,promPhaseLn,
```

PDF MSE2020



- retomar DFT con el concepto de centro de masas
- lanzar euler4.py de la clase3 con dft en modo rectangular
- destacar DFT parecido entre la señal y el círculo y viceversa
- Por lo tanto lo que tenemos son relojes girando

Repaso DFT

Analisis, Transformada discreta de Fourier

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

2020-05-22

Procesamiento de señales, fundamentos

Repaso DFT

Repaso DFT

Analisis, Transformada discreta de Fourier

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

Síntesis

Como reconstruyo la señal en el tiempo



2020-05-22

Síntesis

- lanzar euler6 y explicar que pasa con sin y cos
- explicar el concepto de relojes
- explicar el concepto de frecuencia negativa
- comentar lo de hermitica cuando la salida es real y dft complejo conjugado



```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure

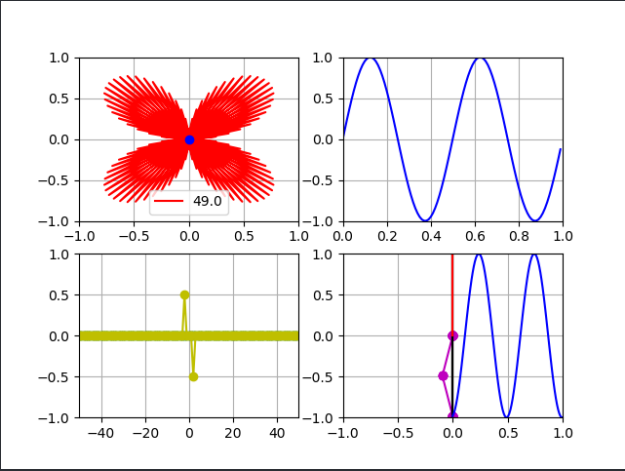
#-----
fig = plt.figure()
fs = 100
N = 100
#-----
circleAxe = fig.add_subplot(2,2,1)
circleLn,massLn, = plt.plot([],[],'r-',[],[],'bo')
circleAxe.grid(True)
circleAxe.set_xlim(-1,1)
circleAxe.set_ylim(-1,1)
circleFreq = np.arange(-fs/2,fs/2,fs/N)
circleLn.set_label(circleFreq[0])
circleLg=circleAxe.legend()
circleData = []
mass = 0
freqIter = 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)
signalFreq = 2
signalData=[]
def signal(f,n):
    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,penLn,penRLn,penILn = plt.plot([],[],'m-o',[],[],'k-',[],[],'b-',[],[],'r-')
inversaAxe.grid(True)
inversaAxe.set_xlim(-1,1)
inversaAxe.set_ylim(-1,1)
penData= []
#-----
tData=np.arange(0,N/fs,1/fs)
def init():
    return circleLn,circleLg,signalLn,massLn,promRLn,promILn,inversaLn
def updateF(n):
    global promData,tData,freqIter,penData
    if aniT.repeat==True:
        return inversaLn,
        inversaData=[0]
    for f in range(N):
        inversaData.append(inversaData[-1]+circleInv(circleFreq[f],freqIter,promData[f]))
    inversaLn.set_data(np.imag(inversaData),np.real(inversaData))
    penData.insert(0,inversaData[-1])
    penData=penData[0:N]
    t=np.linspace(0,1,len(penData))
    penRLn.set_data(t,np.real(penData))
```

Síntesis

Como reconstruyo la señal en el tiempo



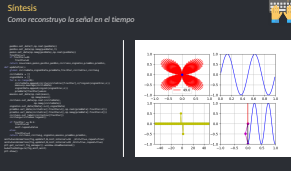
```
penRLn.set_data(t,np.real(penData))
penILn.set_data(np.imag(penData),t)
penLn.set_data(np.imag(penData),np.real(penData))
frecIter+=1
if frecIter==N:
    frecIter=0
    return inversaLn,penLn,penILn,penRLn,sigallLn,promRLn,promILn,
def updateT(nn):
    global circleData,sigallData,promData,frecIter,circleFrec,circleLg
    circleData = []
    sigallData = []
    for n in range(N):
        circleData.append(circle(circleFrec[frecIter],n)*signal(signalFrec,n))
        mass=np.average(circleData)
        sigallData.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))
        sigallLn.set_data(tData[:n+1],sigallData)
        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:
        frecIter=0
        aniT.repeat=False
    else:
        frecIter+=1
    return circleLn,circleLg,sigallLn,massLn,promRLn,promILn,
aniT=FuncAnimation(fig,updateT,N,init,interval=10 ,blit=True,repeat=True)
aniF=FuncAnimation(fig,updateF,N,init,interval=20 ,blit=True,repeat=True)
plt.get_current_fig_manager().window.showMaximized()
b=buttonOnFigure(fig,aniT,aniF)
plt.show()
```



2020-05-22

Síntesis

- de nuevo la formula la calculamos a mano y arrivamos a lo que viene en la filmina siguiente



Síntesis, Transformada inversa discreta de Fourier

IDFT

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

2020-05-22

Procesamiento de señales, fundamentos

└ Síntesis, Transformada inversa discreta de Fourier

Síntesis, Transformada inversa discreta de Fourier
IDFT

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

- comentar tambien lo del $1/N$, $1/\sqrt[2]{N}$

IDFT

Señales complejas como entrada?



2020-05-22

IDFT

Procesamiento de señales, fundamentos



- lanzar euler7
- hacer algunas pruebas con diferentes valores.
- destacar que esto es la DFT compleja de lado a lado

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure
#-----
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):
```

```
def signal(f,n):
    return np.sin(2*np.pi*f*n*1/fs)+0.4j*np.sin(2*np.pi*f*1.5*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)
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,frecIter,penData
    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))
    penData.insert(0,inversaData[-1])
    penData=penData[0:N]
    t=np.linspace(0,1,len(penData))
    penRLn.set_data(t,np.real(penData))
```

IDFT

Señales complejas como entrada?

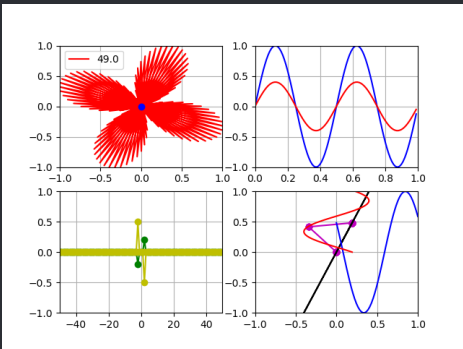


```
penRLn.set_data(t,np.real(penData))
penILn.set_data(np.imag(penData),t)
penLn.set_data(np.imag(penData),np.real(penData))
frecIter+=1
if frecIter==N:
    frecIter=0
return inversaLn,penLn,penILn,penRLn,circleLn,signalRLn,signalILn,promRLn,promILn,

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:
        frecIter=0
        aniT.repeat=False
    else:
        frecIter+=1
    return circleLn,circleLg,signalRLn,signalILn,massLn,promRLn,promILn,

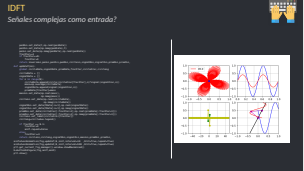
aniT=FuncAnimation(fig,updateT,N,init,interval=10 ,blit=True,repeat=True)
aniF=FuncAnimation(fig,updateF,N,init,interval=500 ,blit=True,repeat=True)
plt.get_current_fig_manager().window.showMaximized()
b=buttonOnFigure(fig,aniT,aniF)
plt.show()
```



2020-05-22

Procesamiento de señales, fundamentos

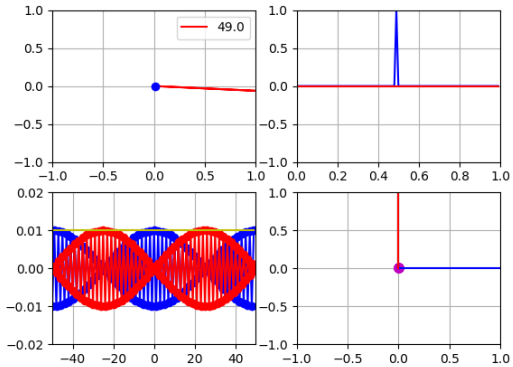
IDFT



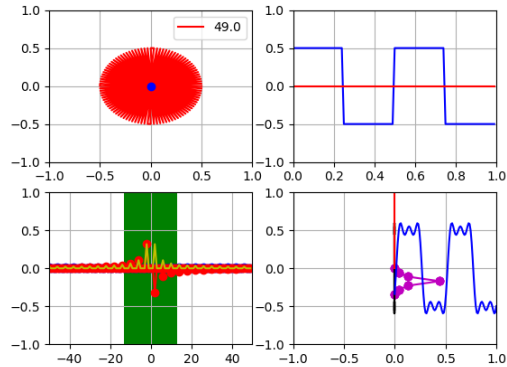
DFT<>IDFT

Transformadas relevantes

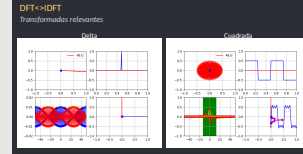
Delta



Cuadrada



DFT<>IDFT

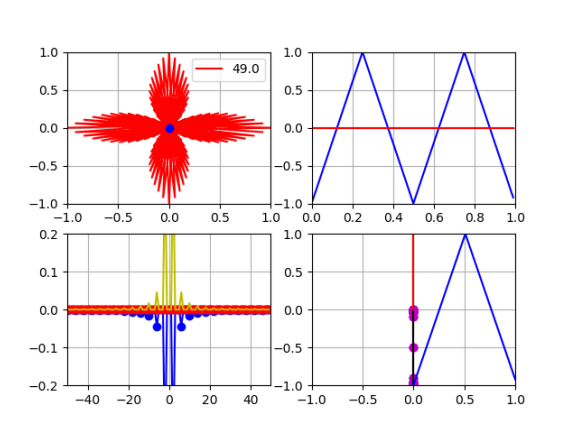


- lanzar euler9
- jugar un poco con la cuadrada y ver como se va formando con los diferentes armónicos
- explicar la idea de compresion, mp3, etc cortando ancho de banda
- destacar lo de los armónicos 1f, 3f, 5f para la cuadrada
- probar la delata y mostrar como tiene todas las frecuencias

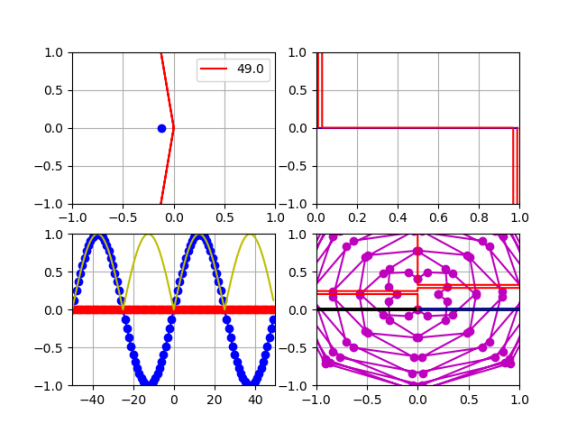
DFT<>IDFT

Transformadas relevantes

Triangular



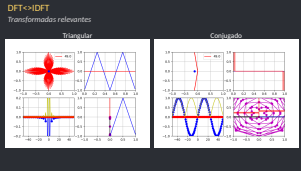
Conjugado



2020-05-22

Procesamiento de señales, fundamentos

DFT<>IDFT



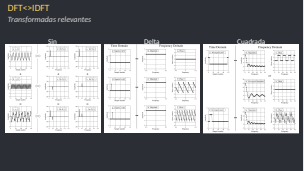
- lanzar el euler4 con los conjugados y ver la dualidad
- jugar con el conjugado y mostrar como sale la senoide
- hablar del tema de dualidad

DFT<>IDFT

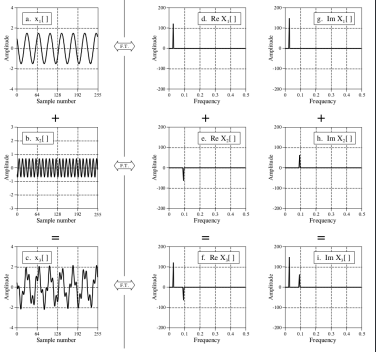
Transformadas relevantes

2020-05-22

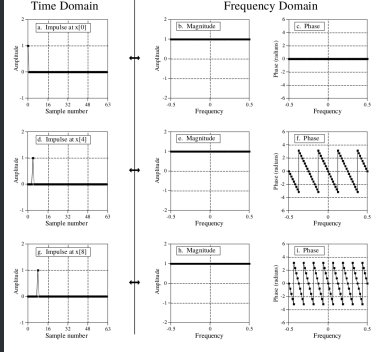
DFT<>IDFT



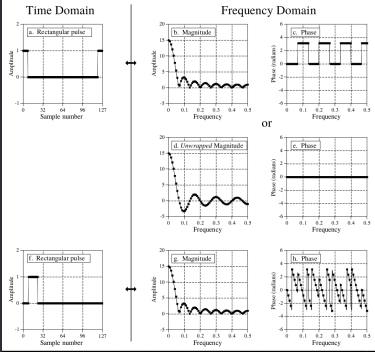
Sin



Delta



Cuadrada



IDFT

Un conejo como entrada?



2020-05-22

Procesamiento de señales, fundamentos

IDFT

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure

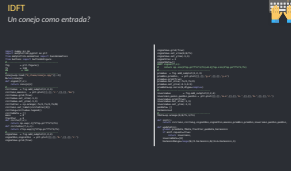
#-----
fig = plt.figure()
fs = 100
#N = 100
#-----
conejo=np.load("4_clase/conejo.npy")[:,1]
N=len(conejo)
def signal(f,n):
    return conejo[n]

#-----
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.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)
#-----
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(-0.1,0.5)
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)
penData= []
harmonics=2
#-----
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,frecIter,penData,harmonics
    if aniT.repeat==True:
        return inversaLn,
        inversaData=[0]
        harmonicRange=range(N//2-harmonics,N//2+1+harmonics,1)
```



IDFT

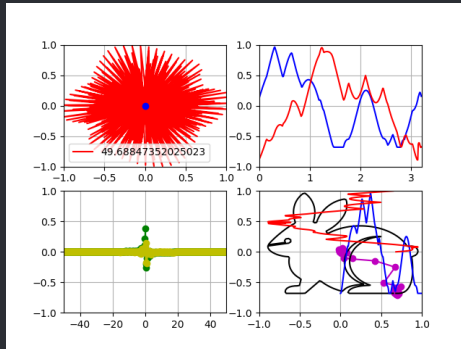
Un conejo como entrada?



```
harmonicRange=range(N//2-harmonics,N//2+1+harmonics,1)
for f in harmonicRange:
    inversaData.append(inversaData[-1]+circleInv(circleFrec[f],frecIter,promData[f]))
inversaLn.set_data(np.imag(inversaData),np.real(inversaData))
penData.insert(0,inversaData[-1])
penData=penData[0:N]
t=np.linspace(0,1,len(penData))
penRLn.set_data(t,np.real(penData))
penILn.set_data(np.imag(penData),t)
penLn.set_data(np.imag(penData),np.real(penData))
promHarmonicLn = promAxe.fill_between([circleFrec[harmonicRange[0]],circleFrec[harmonicRange
[-1]]],1,-1,facecolor="green",alpha=0.1)
frecIter+=1
print(harmonics,N)
if frecIter==N:
    frecIter=0
    harmonics+=1
    if harmonics>=N//2:
        harmonics=1
return inversaLn,penLn,penILn,penRLn,signalRLn,signalILn,promRLn,promILn,promHarmonicLn

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:
        ani1.repeat=False
    else:
        frecIter+=1
return circleLn,signalRLn,signalILn,promRLn,promILn,promHarmonicLn
```

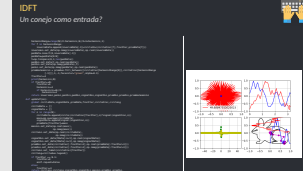


2020-05-22

Procesamiento de señales, fundamentos

IDFT

- lanzar euler8.py y hablar de que todo se puede transformar
- volver a mostara la idea de compresion limitando la idft en ancho de banda



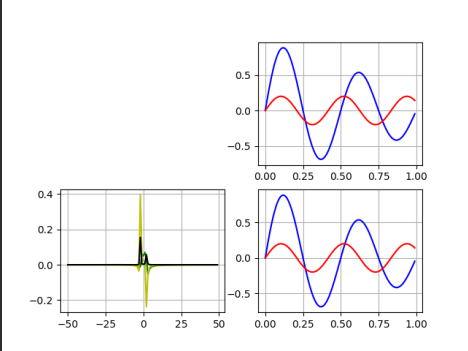
FFT-IFFT

Transformadas usando FFT en Python



```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#-----
fig = plt.figure()
fs = 100
N = 100
frecIter = 0
signalFrec = 2
#-----
tData = np.arange(0,N/fs,1/fs)
nData = np.arange(0,N,1)
circleFrec = np.arange(-fs/2,fs/2,fs/N)
#-----SIGNAL-----
signalData = np.exp(-nData/fs)*np.sin(2*np.pi*signalFrec*nData*1/fs)+0.2j*np.sin(2*np.pi*1.2*
    signalFrec*nData*1/fs)
signalAxe = fig.add_subplot(2,2,2)
signalRLn,signalILn,= plt.plot(tData,np.real(signalData),'b-',tData,np.imag(signalData),'r-')
signalAxe.grid(True)
#-----FFT IFFT-----
fftData = np.fft.fft(signalData)
ifftData = np.fft.ifft(fftData)
fftData = np.concatenate((fftData[N//2:N],fftData[0:N//2]))/N
#-----FFT-----
fftAxe = fig.add_subplot(2,2,3)
fftRLn,fftILn,fftAbsLn = plt.plot(circleFrec,np.real(fftData),'g-',circleFrec,np.imag(
    fftData),'y-',circleFrec,np.abs(fftData)**2,'k-')
fftAxe.grid(True)
#-----IFFT-----
ifftAxe = fig.add_subplot(2,2,4)
penRLn,penILn = plt.plot(tData,np.real(ifftData),'b-',tData,np.imag(ifftData),'r-')
ifftAxe.grid(True)
#-----
plt.show()
```

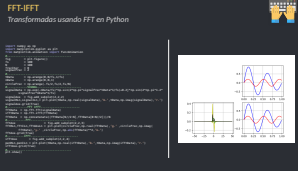


2020-05-22

Procesamiento de señales, fundamentos

FFT-IFFT

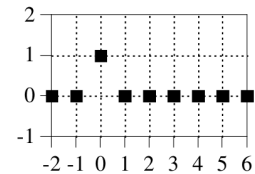
- lanzar fft1
- verificar los mismos resultados que la maquina animada



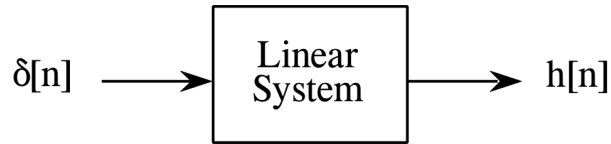
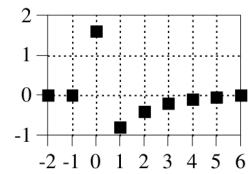
Función delta

Respuesta al impulso

Delta Function



Impulse Response



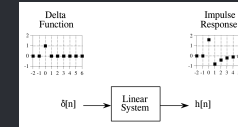
2020-05-22

Procesamiento de señales, fundamentos

↳ Respuesta al impulso

↳ Función delta

Función delta
Respuesta al impulso



- comentar que cada sistema tiene una única $h(n)$
- golpe de Homero en la panza
- si algo cambia en el sistema, $h(n)$ cambia si o si
- recordar propiedad de shift y linealidad en las 2 filminas siguientes

Todo esta en la respuesta al impulso



2020-05-22

Procesamiento de señales, fundamentos

└ Respuesta al impulso

└ Todo esta en la respuesta al impulso

Todo esta en la respuesta al impulso

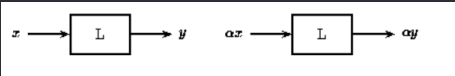


- todo esta en la $h(n)$

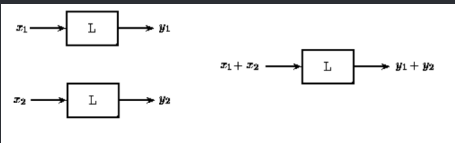
Linealidad

Un sistema es lineal cuando su salida depende linealmente de la entrada. Satisface el principio de superposición.

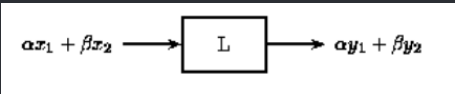
escalado



adición



superposición



$$y(t) = e^{x(t)}$$
$$y(t) = \frac{1}{2}x(t)$$

2020-05-22

Procesamiento de señales, fundamentos

Respuesta al impulso

Repaso Sistemas

Repaso Sistemas

Linealidad

Un sistema es lineal cuando su salida depende linealmente de la entrada. Satisface el principio de superposición.

escalado

adición

superposición

$y(t) = e^{x(t)}$
 $y(t) = \frac{1}{2}x(t)$

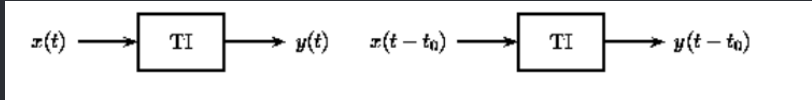
- recordar las propiedades fundamentales de LTI

Repaso - Sistemas

Invariantes en el tiempo

Invariantes en el tiempo

Un sistema es invariante en el tiempo cuando la salida para una determinada entrada es la misma sin importar el tiempo en el cual se aplica la entrada



$$y(t) = x(t) * \cos(t)$$
$$y(t) = \cos(x(t))$$

2020-05-22

- Procesamiento de señales, fundamentos
 - Respuesta al impulso
 - Repaso - Sistemas

Repaso - Sistemas

Invariantes en el tiempo

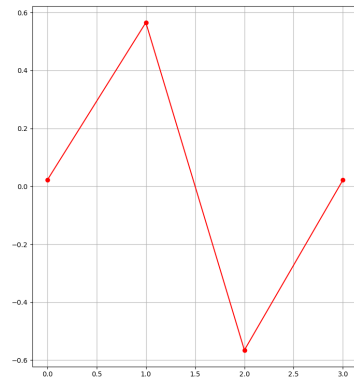
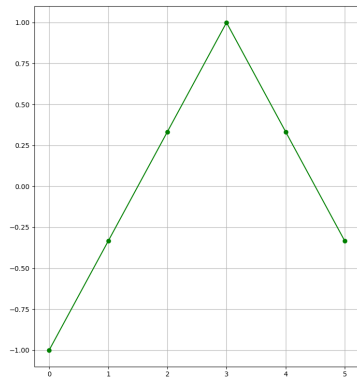
Invariantes en el tiempo

Un sistema es invariante en el tiempo cuando la salida para una determinada entrada es la misma sin importar el tiempo en el cual se aplica la entrada

$$y(t) = x(t) * \cos(t)$$
$$y(t) = \cos(x(t))$$

Convolución

Señal vs $h(n)$



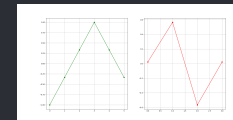
2020-05-22

Procesamiento de señales, fundamentos

└ Respuesta al impulso

└ Convolución

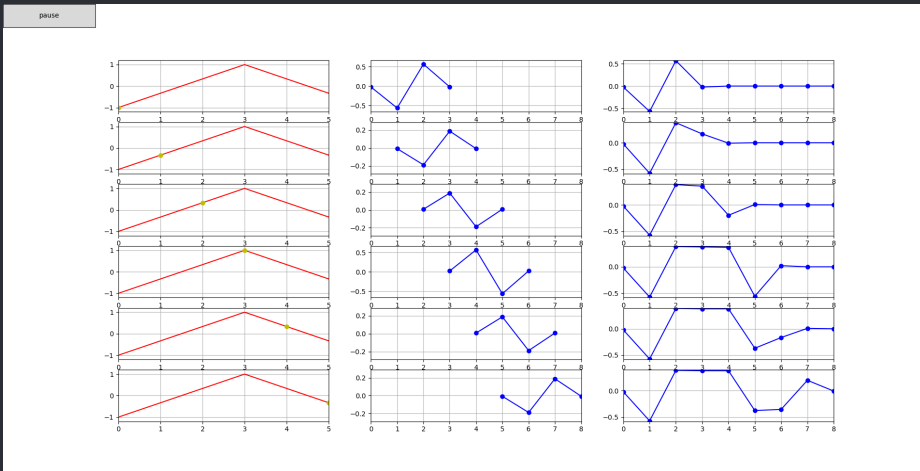
Convolución
Señal vs $h(n)$



- analizar senial y h
- vamos a hacer las cuentas a mano..

Convolución

Descomposición felta

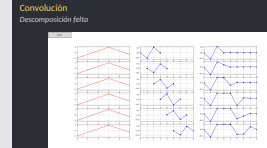


2020-05-22

Procesamiento de señales, fundamentos

↳ Respuesta al impulso

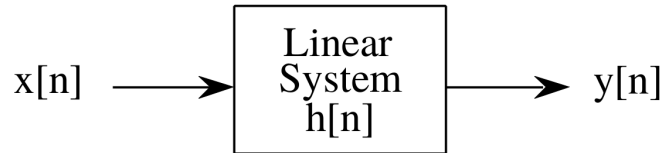
↳ Convolución



- lanzar conv3
- sumo delta a delta y voy acumulando..
- la salida es $N+M-1$

Funcion delta

Respuesta al impulso



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

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

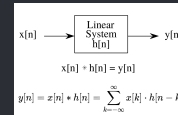
2020-05-22

Procesamiento de señales, fundamentos

└ Respuesta al impulso

└ Funcion delta

Funcion delta
Respuesta al impulso



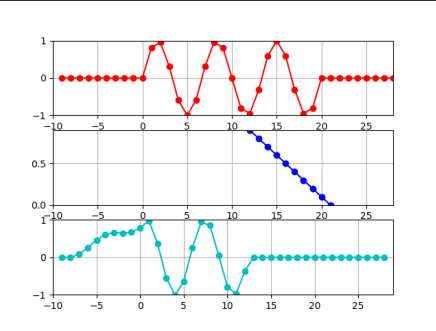
Convolución

Respuesta al impulso

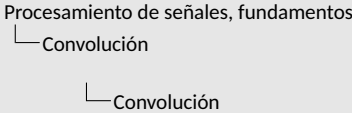
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure

#-----
fig = plt.figure()
fs = 20
N = 20
M=10
#-----
xFreq = 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(xFreq,tData[M-1:M-1+N])
xAxis = fig.add_subplot(3,1,1)
xLn,xHighLn = plt.plot(tData,xData,'r-o',[],[],'y-
o')
xAxis.grid(True)
xAxis.set_xlim(-M,M+N-2)
xAxis.set_ylim(np.min(xData),np.max(xData))
#-----
hData=[0.1*n for n in range(M)]
hAxis = fig.add_subplot(3,1,2)
```

```
hLn, = plt.plot([],[],'b-o')
hAxis.grid(True)
hAxis.set_xlim(-M,M+N-2)
hAxis.set_ylim(np.min(hData),np.max(hData))
#-----
yAxis = fig.add_subplot(3,1,3)
yLn, = plt.plot([],[],'c-o')
yAxis.grid(True)
yAxis.set_xlim(-M,M+N-1)
yAxis.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.get_current_fig_manager().window.showMaximized
()
b=buttonOnFigure(fig,ani)
plt.show()
```



2020-05-22



Convolution

Respuesta al impulso

```
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.get_current_fig_manager().window.showMaximized
()
b=buttonOnFigure(fig,ani)
plt.show()
```

Filtrado

Pasa bajos

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure

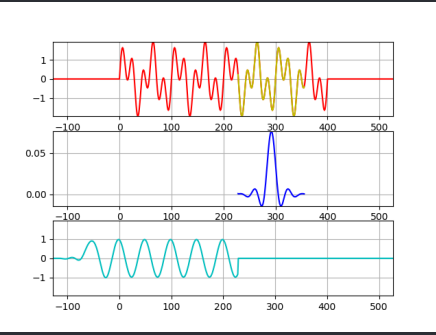
#-----
fig = plt.figure()
fs = 100
N = 400
fir=np.load("4_clase/low_pass.npy").astype(float)
M=len(fir)
#fir=np.load("diferenciador.npy").astype(float)
#M=len(fir)
#-----
def x(f,n):
    return np.sin(2*np.pi*2*n*1/fs)+\
           np.sin(2*np.pi*5*n*1/fs)

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])
hAxe = 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
()
b=buttonOnFigure(fig,ani)
plt.show()
```



2020-05-22

Procesamiento de señales, fundamentos

Filtrado

Filtrado

Filtrado

Pasa bajos

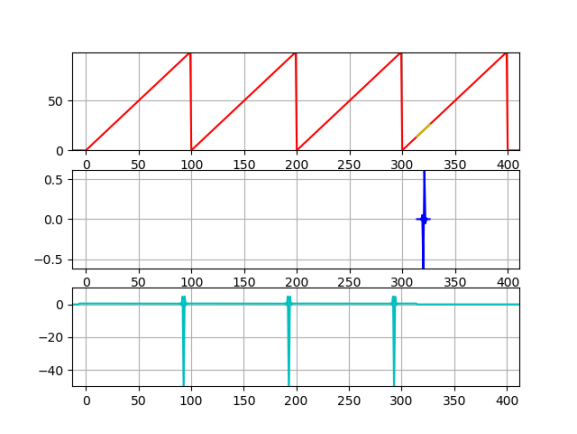
```
def init():
    global yData
    yData=np.zeros(N+2*(M-1))
    return hLn,xLn,xHighLn,yLn,

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
()
b=buttonOnFigure(fig,ani)
plt.show()
```

Filtrado

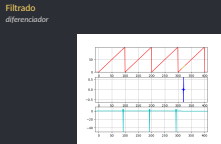
diferenciador



2020-05-22

Procesamiento de señales, fundamentos

- Filtrado
 - Filtrado



Convolución

Análisis en la CIAA



```
#include "sapi.h"
#include "arm_math.h"
#include "arm_const_structs.h"

#define MAX_FFT_LENGTH 2048 // maxima longitud para la fft y chunk de samples
#define BITS 10 // cantidad de bits usado para cuantizar
#define FIR_LENGTH 162

int16_t fftLength = 32; // longitud de la fft y samples variable
int16_t adc [ MAX_FFT_LENGTH ]; // guarda los samples
q15_t fftIn [ MAX_FFT_LENGTH ]; // guarda copia de samples en Q15 como in para la fft.La fft corrompe los datos de la entrada!
q15_t fftOut[ MAX_FFT_LENGTH*2 ]; // salida de la fft
q15_t fftMag[ MAX_FFT_LENGTH*2+1 ]; // magnitud de la FFT
//low_pass 1000hz 127
//q15_t fir[ FIR_LENGTH]={ 22, 21, 24, 19, 6, -14, -40, -66, -83, -86, -73, -43, -3, 36, 65, 73, 56, 16, -35, -84, -115, -114, -77, -13, 63, 128, 161, 147, 84, -13, -119, -200, -227, -186, -81, 62, 204, 299, 310, 224, 56, -152, -341, -445, -422, -259, 11, 320, 575, 687, 593, 287, -176, -683, -1083, -1220, -977, -305, 755, 2074, 3450, 4656, 5480, 5772, 5480, 4656, 3450, 2074, 755, -305, -977, -1220, -1083, -683, -176, 287, 593, 687, 575, 320, 11, -259, -422, -445, -341, -152, 56, 224, 310, 299, 204, 62, -81, -186, -227, -200, -119, -13, 84, 147, 161, 128, 63, -13, -77, -114, -115, -84, -35, 16, 56, 73, 65, 36, -3, -43, -73, -86, -83, -66, -40, -14, 6, 19, 24, 21, 22};
//bandpass 440hz 162
q15_t fir[ FIR_LENGTH]={ 1, 0, 0, -0, -2, -5, -9, -14, -19, -25, -29, -32, -33, -30, -24, -14, 0, 17, 37, 57, 76, 91, 101, 104, 98, 83, 60, 31, -1, -36, -68, -95, -113, -121, -117, -103, -81, -54, -26, -3, 11, 13, 2, -22, -58, -99, -140, -172, -186, -175, -135, -62, 41, 170, 315, 461, 593, 694, 748, 742, 667, 522, 312, 46, -254, -566, -864, -1118, -1303, -1396, -1383, -1260, -1028, -704, -310, 123, 561, 967, 1307, 1551, 1679, 1679, 1551, 1307, 967, 561, 123, -310, -704, -1028, -1260, -1383, -1396, -1303, -1118, -864, -566, -254, 46, 312, 522, 667, 742, 748, 694, 593, 461, 315, 170, 41, -62, -135, -175, -186, -172, -140, -99, -58, -22, 2, 13, 11, -3, -26, -54, -81, -103, -117, -121, -113, -95, -68, -36, -1, 31, 60, 83, 98, 104, 101, 91, 76, 57, 37, 17, 0, -14, -24, -30, -33, -32, -29, -25, -19, -14, -9, -5, -2, -0, 0, 0, 1};

q15_t firOut [ MAX_FFT_LENGTH+FIR_LENGTH+1 ];
uint32_t maxIndex = 0; // indexador de maxima energia por cada fft
q15_t maxValue = 0; // maximo valor de energia del bin por cada fft
arm_rfft_instance_q15 S;
uint16_t sample = 0; // contador para samples

int main ( void ) {
    boardConfig ( );
    uartConfig ( UART_USB, 460800 );
    adcConfig ( ADC_ENABLE );
    cyclesCounterInit ( EDU_CIAA_NXP_CLOCK_SPEED );
    while(1) {
        Ing. Pablo Slavkin
```

```
sample = 0; // arranca de nuevo
uartWriteByteArray ( UART_USB ,(uint8_t*) &maxValue ,2);
uartWriteByteArray ( UART_USB ,(uint8_t*) &maxIndex ,2);
uartWriteByteArray ( UART_USB ,"header" ,6 ); // manda el header
// que casualmente se llama "header" con lo que arranca una nueva trama
uartWriteByteArray ( UART_USB ,(uint8_t*) &fftLength ,sizeof(fftLength)); // manda el largo de la fft que es variable
arm_rfft_init_q15 ( &S ,fftLength ,0 ,1 ); // inicializa una estructura que usa la funcion fft para procesar los datos. Notar el /2 para el largo
arm_rfft_q15 ( &S ,fftIn ,fftOut ); // por fin.. ejecuta la rfft REAL fft
arm_cmplx_mag_squared_q15 ( fftOut ,fftMag ,fftLength/2+1 );
arm_max_q15 ( fftMag ,fftLength/2+1 ,&maxValue ,&maxIndex );
gpioToggle( LEDR);
if ( gpioRead(TEC1 )==0) {
    gpioToggle(LED8);
    if(((fftLength<=1)>MAX_FFT_LENGTH)
        fftLength=32;
        while(gpioRead(TEC1)==0)
            ;
    }
}
while(cyclesCounterRead()< 20400) //clk de 204000000 => 10k samples x seg.
{
    ;
}
```

2020-05-22

Procesamiento de señales, fundamentos

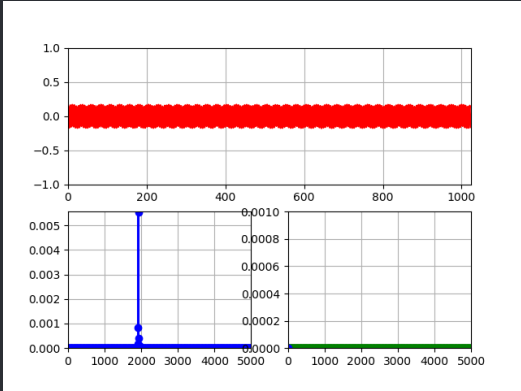
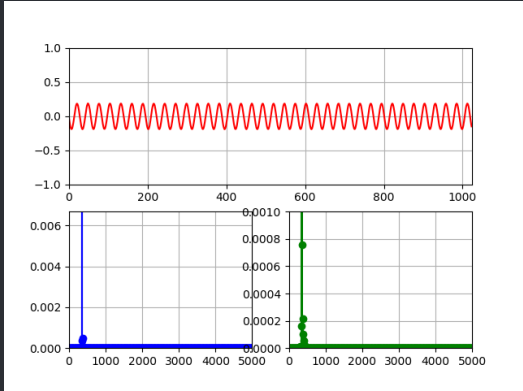
Filtrado

Convolución



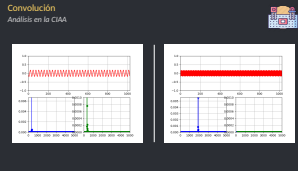
Convolución

Análisis en la CIAA



2020-05-22

- Procesamiento de señales, fundamentos
- Filtrado
 - Convolución



Bibliografía

Libros, links y otro material

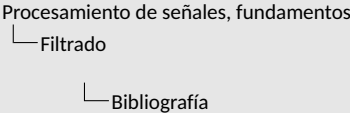
[1] ARM CMSIS DSP.
https://arm-software.github.io/CMSIS_5/DSP/html/index.html

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

[3] Grant Sanderson
<https://youtu.be/spUNpyF58BY>

[4] Interactive Mathematics Site Info.
<https://www.intmath.com/fourier-series/fourier-intro.php>

2020-05-22



Bibliografía
Libros, links y otro material

[1] ARM CMSIS DSP.
https://arm-software.github.io/CMSIS_5/DSP/html/index.html

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

[3] Grant Sanderson
<https://youtu.be/spUNpyF58BY>

[4] Interactive Mathematics Site Info.
<https://www.intmath.com/fourier-series/fourier-intro.php>