

# 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



2020-05-21

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



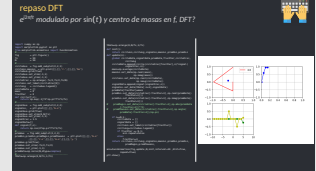
# repaso DFT

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



2020-05-21

repaso DFT



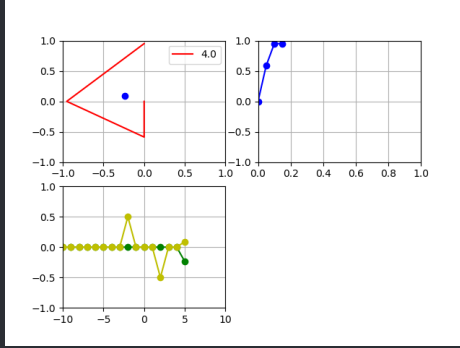
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#-----
fig = plt.figure()
fs = 50
N = 50
#-----
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)
signalFrec = 1.5
signalData = []
def signal(f,n):
    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)
```

```
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)

    if n==N-1:
        circleData = []
        signalData = []
        circleLn.set_label(circleFrec[frecIter])
        circleLg=circleAxe.legend()
        if frecIter == N-1:
            ani.repeat=False
        else:
            frecIter+=1
    return circleLn,circleLg,signalLn,massLn,promRLn,promILn,
        promMagLn,promPhaseLn,

ani=FuncAnimation(fig,update,N,init,interval=10 ,blit=True,
    repeat=True)
plt.show()
```



- 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 circulo y viceversa
- Por lo tanto lo que tenemos son relojos girando

# Repaso DFT

Analisis, Transformada discreta de Fourier

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

2020-05-21

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

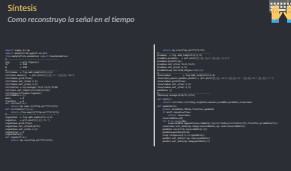
## Como reconstruyo la señal en el tiempo



2020-05-21

Procesamiento de señales, fundamentos

### Síntesis



```
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, penRLn, penILn, 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, 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))
    penILn.set_data(np.imag(penData), t)
```

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

# Síntesis

## Como reconstruyo la señal en el tiempo



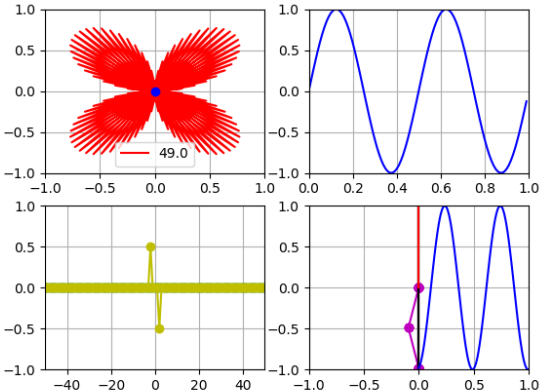
2020-05-21

### Síntesis



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

```
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,
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:
        frecIter=0
        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=20 ,blit=True,repeat=True)
plt.show()
```



# Síntesis, Transformada inversa discreta de Fourier

IDFT

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

2020-05-21

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?



2020-05-21

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

#-----
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)
penData= []
#-----
tData=np.arange(0,N/fs,1/fs)
def init():
    return circleLn,circleLg,signalRLn,signalILn,massLn,promRLn,promILn,inversaLn,penLn,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))
    penILn.set_data(np.imag(penData),t)
```

# IDFT

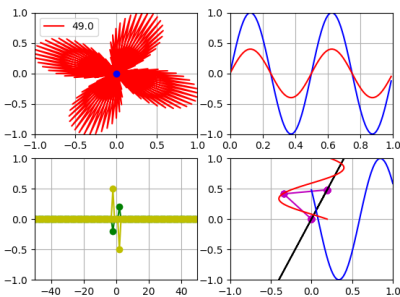
## Señales complejas como entrada?



```
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,

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()

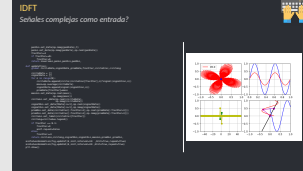
    if frecIter == N-1:
        frecIter=0
        aniT.repeat=False
    else:
        frecIter+=1
    return circleLn,circleLg,signalRLn,signalILn,massLn,promRLn,promILn,
aniT=FuncAnimation(fig,updateT,N,init,interval=10 ,blit=True,repeat=True)
aniF=FuncAnimation(fig,updateF,N,init,interval=30 ,blit=True,repeat=True)
plt.show()
```



2020-05-21

## Procesamiento de señales, fundamentos

IDFT

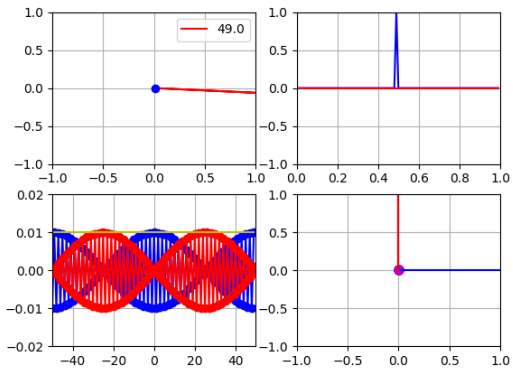




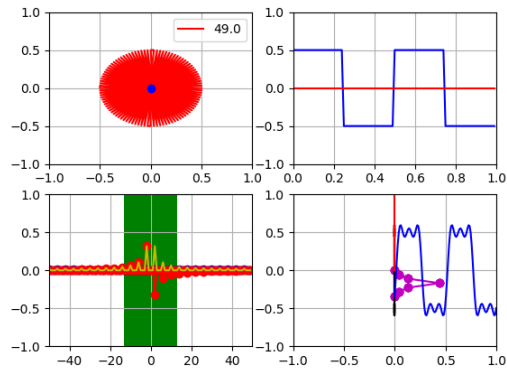
# DFT<>IDFT

## Transformadas relevantes

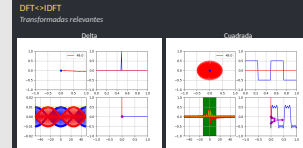
Delta



Cuadrada



DFT<>IDFT

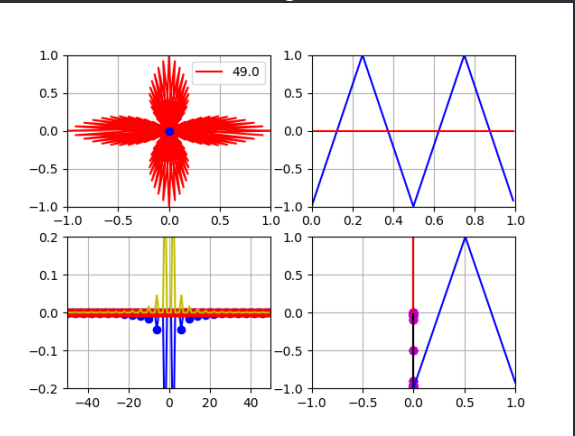


- lanzar euler9
- jugar un poco con la cuadrada y ver como se va formando con los diferentes armonicos
- explicar la idea de compresion, mp3, etc cortando ancho de banda
- destacar lo de los armonicos 1f, 3f, 5f para la cuadrada
- probar la delata y mostrar como tiene todas las frecuencias

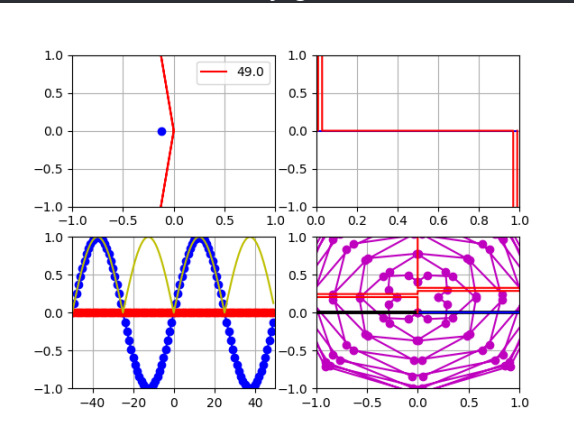
# DFT<>IDFT

Transformadas relevantes

Triangular



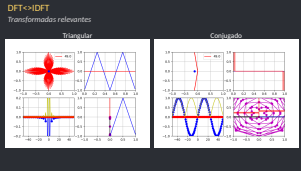
Conjugado



2020-05-21

Procesamiento de señales, fundamentos

DFT<>IDFT



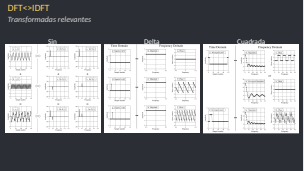
- jugar con el conjugado y mostrar como sale la senoide
- hablar del tema de dualidad

# DFT<>IDFT

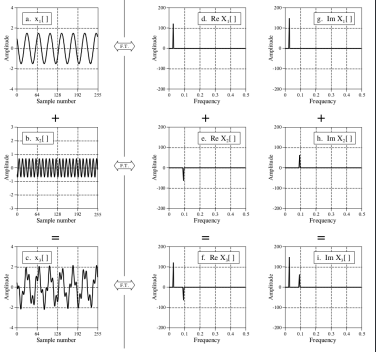
Transformadas relevantes

2020-05-21

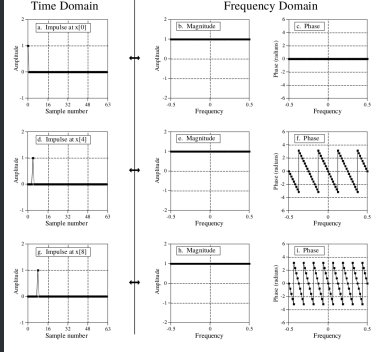
DFT<>IDFT



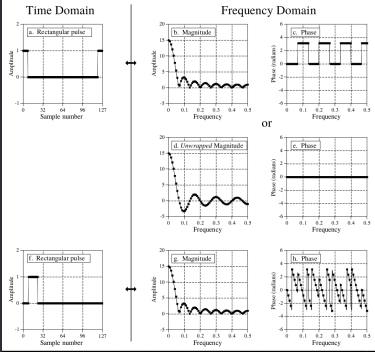
Sin



Delta



Cuadrada



# IDFT

## Un conejo como entrada?



2020-05-21

## Procesamiento de señales, fundamentos

IDFT

```
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")[:,10]
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)
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)
        for f in harmonicRange:
```



# IDFT

## Un conejo como entrada?

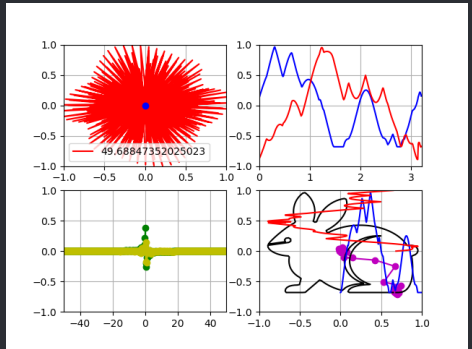


```
for f in harmonicRange:
    inversaData.append(inversaData[-1]+circleInv(circleFrec[f],frecIter,promData[f]))
inversaLn.set_data(np.imag(inversaData),np.real(inversaData))
penData.insert(0,inversaData[-1])
penData=penData[0:N]
t=np.linspace(0,1,len(penData))
penRLn.set_data(t,np.real(penData))
penILn.set_data(np.imag(penData),t)
penLn.set_data(np.imag(penData),np.real(penData))
promHarmomicLn = 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,promHarmomicLn

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
        stopAt=False
    else:
        frecIter+=1
    return circleLn,circleLg,signalRLn,signalILn,massLn,promRLn,promILn,
```

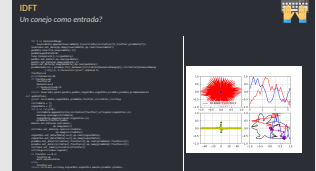


2020-05-21

## Procesamiento de señales, fundamentos

IDFT

- volver a mostara la idea de compresion limitando la idft en ancho de banda



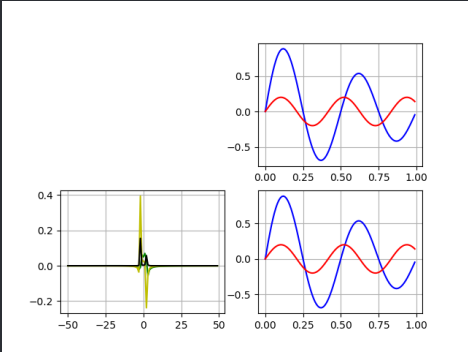
# FFT-IFFT

## Transformadas usando FFT en Python



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

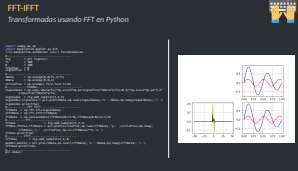
#-----
fig = plt.figure()
fs = 100
N = 100
frecIter = 0
signalFrec = 2
#-----
tData = np.arange(0,N/fs,1/fs)
nData = np.arange(0,N,1)
circleFrec = np.arange(-fs/2,fs/2,fs/N)
#-----SIGNAL-----
signalData = np.exp(-nData/fs)*np.sin(2*np.pi*signalFrec*nData*1/fs)+0.2j*np.sin(2*np.pi*1.2*
    signalFrec*nData*1/fs)
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()
```



2020-05-21

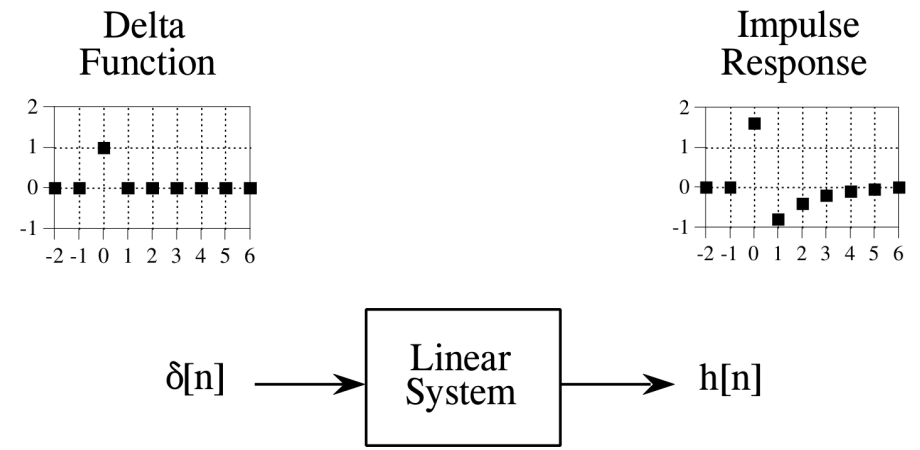
## Procesamiento de señales, fundamentos

FFT-IFFT



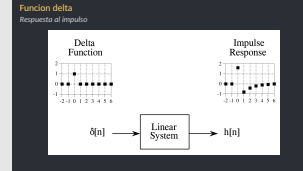
# Funcion delta

Respuesta al impulso



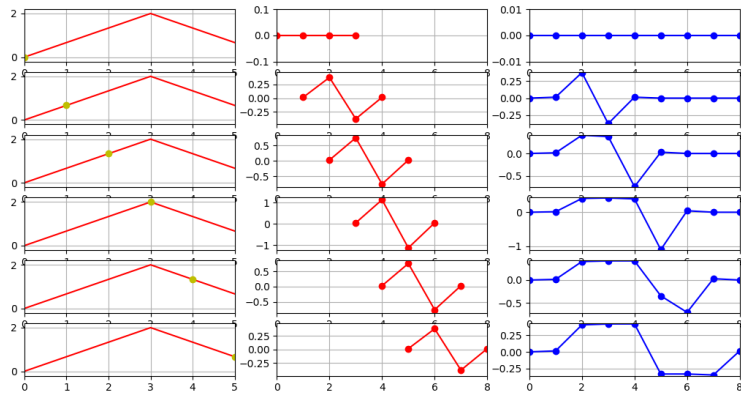
2020-05-21

- Procesamiento de señales, fundamentos
  - Respuesta al impulso
    - Funcion delta



# Convolution

## Descomposicion Delta



2020-05-21

## Procesamiento de señales, fundamentos

— Respuesta al impulso

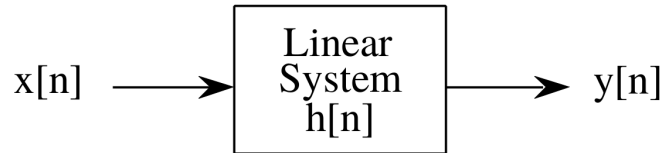
- Convolution





# Funcion delta

## Respuesta al impulso



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

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

2020-05-21

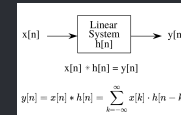
## Procesamiento de señales, fundamentos

└ Respuesta al impulso

└ Funcion delta

## Funcion delta

Respuesta al impulso

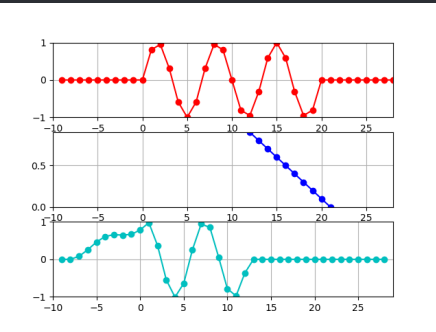


# Convolucion

## 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
#-----
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])
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()
```



2020-05-21

### Procesamiento de señales, fundamentos

└─ Convolucion

└─ Convolucion

Convolucion

Respuesta al impulso

```
def convolucion(x,h):
    N=len(x)
    M=len(h)
    y=np.zeros(N+M-1)
    for i in range(N):
        for j in range(M):
            y[i+j]=y[i+j]+x[i]*h[j]
    return y
```

# Filtrado

## Pasa bajos

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

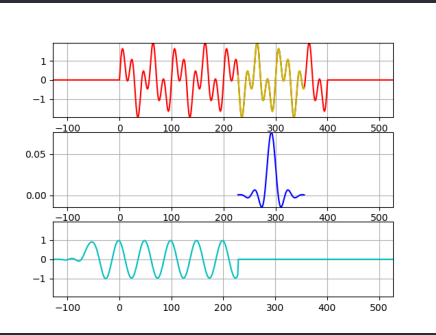
#-----
fig = plt.figure()
fs = 100
N = 400
fir=np.load("low_pass.npy").astype(float)
M=len(fir)
#fir=np.load("diferenciador.npy").astype(float)
#M=len(fir)
#-----
def x(f,n):
    return np.sin(2*np.pi*2*n*1/fs)+\
           np.sin(2*np.pi*5*n*1/fs)

xFreq = 3
tData=np.arange(-(M-1),N+(M-1),1)
xData=np.zeros(N+2*(M-1))
xData[M:M+N]=x(xFreq,tData[M:M+N])
hAxe = fig.add_subplot(3,1,1)
hLn,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,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
,bli=True,repeat=True)
plt.get_current_fig_manager().window.showMaximized
()
plt.show()
```

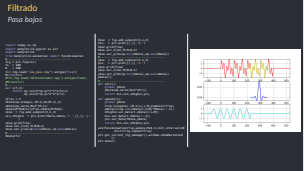


2020-05-21

## Procesamiento de señales, fundamentos

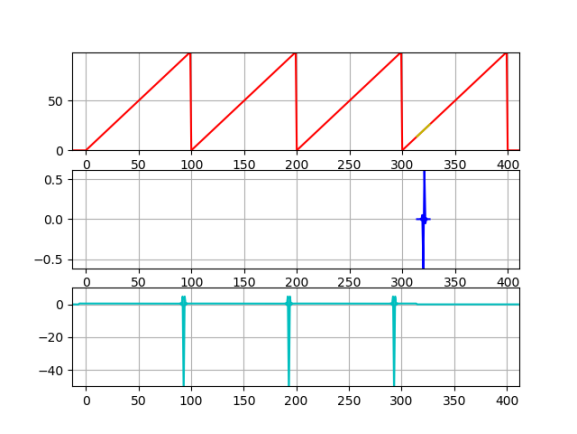
Filtrado

Filtrado



# Filtrado

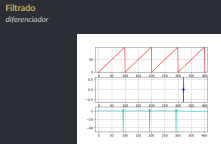
diferenciador



2020-05-21

Procesamiento de señales, fundamentos

- Filtrado
  - Filtrado



# Bibliografía

Libros, links y otro material

[1] ARM CMSIS DSP.  
link

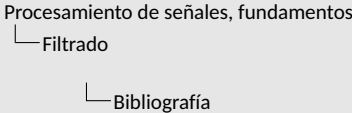
[2] Steven W. Smith. *The Scientist and Engineer’s Guide to Digital Signal Processing*. Second Edition, 1999.

[3] *Interactive Mathematics Site Info*.

[4] Grant Sanderson  
link

[5] *Interactive Mathematics Site Info*.  
link

2020-05-21



**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