FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Procesamiento de señales, fundamentos

Maestría en sistemas embebidos
Universidad de Buenos Aires
MSE 5Co2020

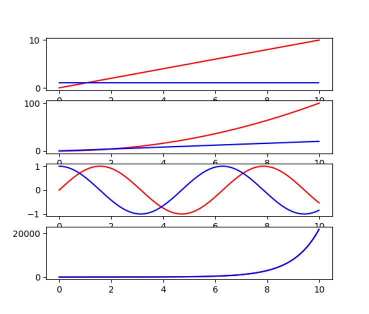
Clase 3 - Euler | Fourier - DFT

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

2.7182818284590450907955982984276488423347473144

- $f(t) = t$
- $f(t) = t^2$
- $f(t) = sin(t)$

- $f'(t) = 1$
- $f'(t) = 2 * t$
- $f'(t) = cos(t)$



La derivada es igual a la funcion

$$f(t) = e^t \implies f'(t) = e^t$$
$$f(t) = e^{kt} \implies f'(t) = ke^{kt}$$

$e^{j2\pi ft}$

Euler

Pero que pasa con e^{jt}?

La derivada es igual a la funcion

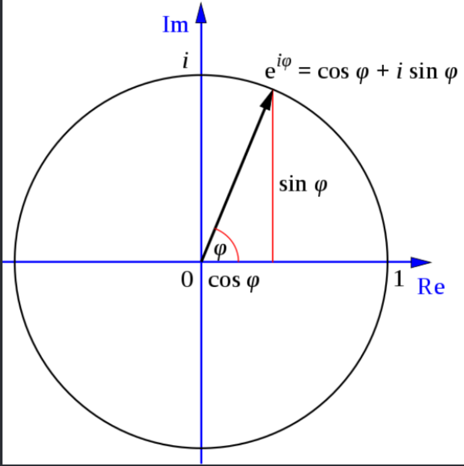
$$f(t) = e^{jt} \implies f'(t) = je^{jt}$$

$$e^{jt} = \cos(t) + j\sin(t)$$

$$e^{j\pi} = -1$$

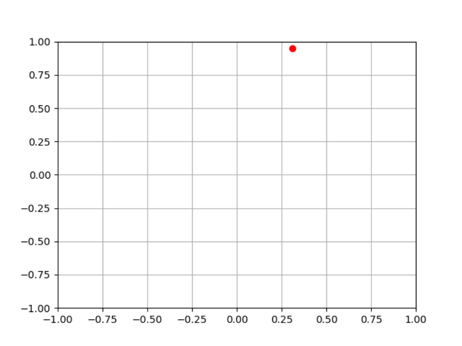
$$e^{\frac{j\pi}{2}} = j$$

$$e^{\frac{j3\pi}{2}} = -j$$



$e^{j2\pi ft}$ *animado*

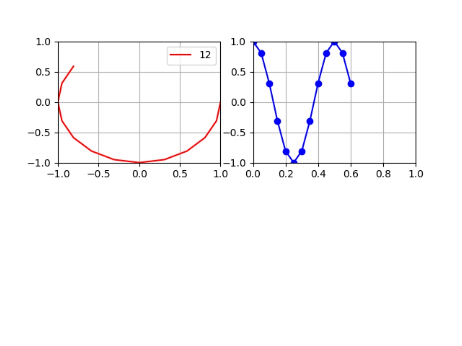
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
fig         = plt.figure()
fs          = 20
N           = 20
circleAxe   = fig.add_subplot(1,1,1)
circleLn,   = plt.plot([],[],’ro’)
circleAxe.grid(True)
circleAxe.set_xlim(-1,1)
circleAxe.set_ylim(-1,1)
circleFrec = 1
def circle(c,f,n):
    return c*np.exp(-1j*2*np.pi*f*n*1/fs)
def init():
    return circleLn,
def update(n):
    circleLn.set_data(np.real(circle(1,circleFrec,n)),
                      np.imag(circle(1,circleFrec,n)))
    return circleLn,
ani=FuncAnimation(fig,update,N,init,interval=100 ,blit=False,repeat=True
     )
plt.show()
```



$e^{j2\pi ft}$ y $\sin(t)$ *animados independientemente*

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure
#------------------------------------
fig        = plt.figure()
fs         = 20
N          = 20
#------------------------------------
circleAxe  = fig.add_subplot(2,2,1)
circleLn,  = plt.plot([],[],'r-')
circleAxe.grid(True)
circleAxe.set_xlim(-1,1)
circleAxe.set_ylim(-1,1)
circleLn.set_label(0)
circleLg=circleAxe.legend()
circleFrec = 1
circleData = []
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 = 2
```

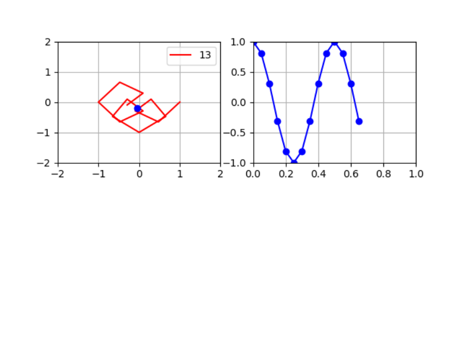
```python
signalFrec = 2
signalData=[]
def signal(f,n):
    return np.cos(2*np.pi*f*n*1/fs)
#--------------------------------------
tData=np.arange(0,N/fs,1/fs)
def init():
    return circleLn,circleLg,signalLn,
def update(n):
    global circleData,signalData
    circleData.append(circle(circleFrec,n))
    circleLn.set_data(np.real(circleData),
                      np.imag(circleData))
    signalData.append(signal(signalFrec,n))
    signalLn.set_data(tData[:n+1],signalData)

    if n==N-1:
        circleData=[]
        signalData=[]
    circleLn.set_label(n)
    circleLg=circleAxe.legend()
    return circleLn,circleLg,signalLn,

ani=FuncAnimation(fig,update,N,init,interval=500 ,
        blit=True,repeat=True)
plt.get_current_fig_manager().window.showMaximized
        ()
b=buttonOnFigure(fig,ani)
plt.show()
```



$e^{j2\pi ft}$ modulado por $\sin(t)$ y centro de masas

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure
#----------------------------------------
fig        = plt.figure()
fs         = 20
N          = 20
#----------------------------------------
circleAxe  = fig.add_subplot(2,2,1)
circleLn,  = plt.plot([],[],'r-')
circleAxe.grid(True)
circleAxe.set_xlim(-1,1)
circleAxe.set_ylim(-1,1)
circleLn.set_label(0)
circleLg=circleAxe.legend()
circleFrec = 1
circleData = []
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 = 2
signalData=[]
def signal(f,n):
```

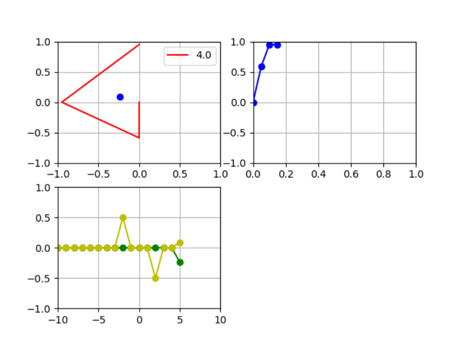
```python
signalData=[]
def signal(f,n):
    return np.cos(2*np.pi*f*n*1/fs)
#---------------------------------------
tData=np.arange(0,N/fs,1/fs)

def init():
    return circleLn,circleLg,signalLn,massLn
def update(n):
    global circleData,signalData
    circleData.append(circle(circleFrec,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)

    if n==N-1:
        circleData = []
        signalData = []
    circleLn.set_label(n)
    circleLg=circleAxe.legend()
    return circleLn,circleLg,signalLn,massLn

ani=FuncAnimation(fig,update,N,init,interval=10 ,
        blit=True,repeat=True)
plt.get_current_fig_manager().window.showMaximized
        ()
b=buttonOnFigure(fig,ani)
plt.show()
```



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

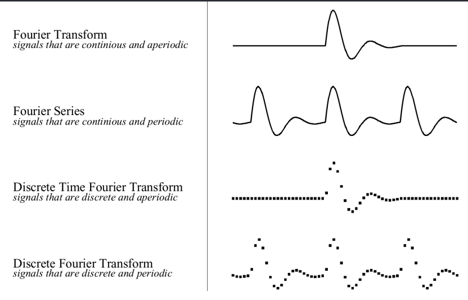
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure
#-------------------------------------
fig        = plt.figure()
fs         = 50
N          = 50
#----------CONJUGADO---------------------
#conjugado=np.zeros(100,dtype=complex)
#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)
signalFrec = 2
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)
```

```python
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=circleAxe.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,

ani=FuncAnimation(fig,update,N,init,interval=10 ,blit=True,repeat=True)
plt.get_current_fig_manager().window.showMaximized()
b=buttonOnFigure(fig,ani)
plt.show()
```



Transformada de Fourier

Diferentes tipos segun la señal



Time Duration		
Finite	Infinite	
Discrete FT (DFT) $X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\omega_k n}$ $k = 0, 1, \ldots, N-1$	Discrete Time FT (DTFT) $X(\omega) = \sum_{n=-\infty}^{+\infty} x(n) e^{-j\omega n}$ $\omega \in [-\pi, +\pi)$	discr. time n
Fourier Series (FS) $X(k) = \frac{1}{P} \int_0^P x(t) e^{-j\omega_k t} dt$ $k = -\infty, \ldots, +\infty$	Fourier Transform (FT) $X(\omega) = \int_{-\infty}^{+\infty} x(t) e^{-j\omega t} dt$ $\omega \in (-\infty, +\infty)$	cont. time t
discrete freq. k	continuous freq. ω	

Analisi, Transformada discreta de Fourier

DFT

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

DFT

Densidad de potencia espectral

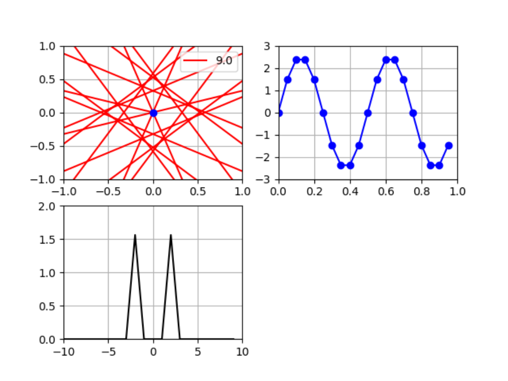
$$P_{sin} = \frac{A^2}{2}$$

$$P_{sin} = \frac{2,5^2}{2}$$

$$P_{sin} = 3,125W$$

$$P = 1,56 + 1,56$$

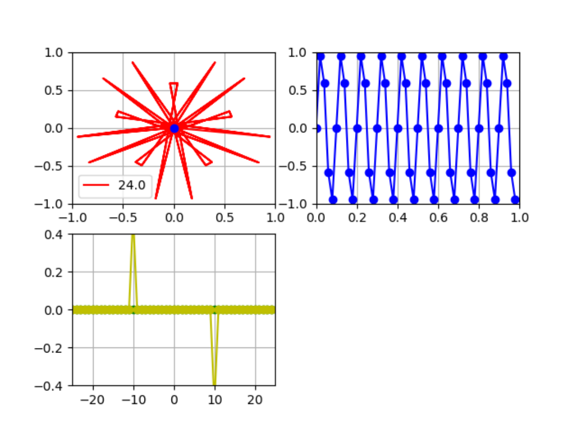
$$P = 3,125W$$



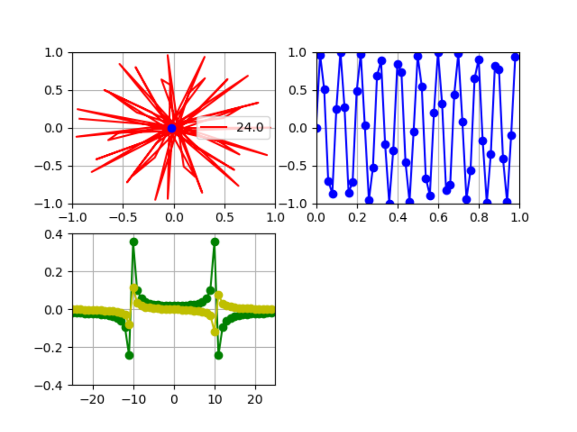
DFT

Fuga espectral (Spectral leakage)

10Hz



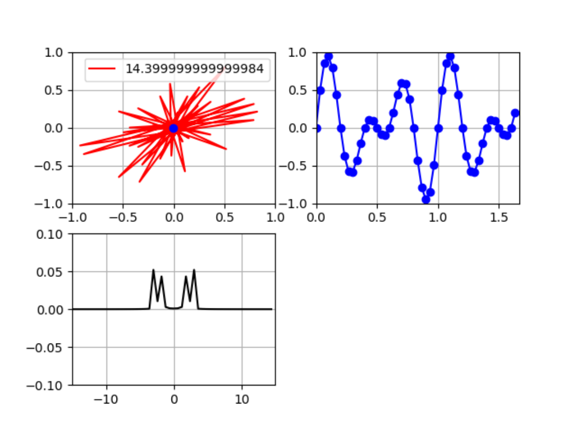
10.4hz



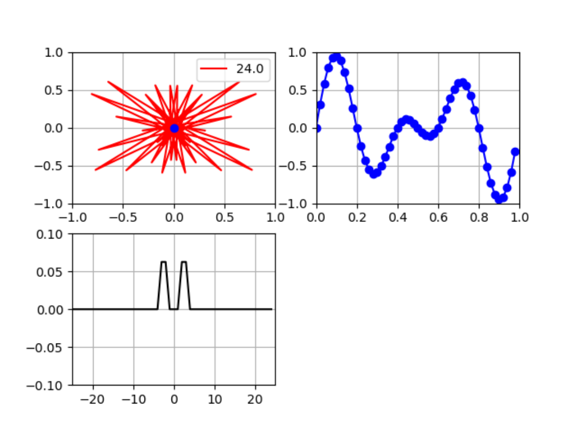
DFT

Resolucion espectral

f1=2 f2=3 fs = 30 N = 50



f1=2 f2=3 fs = 50 N = 50



DFT

DFT Zero padding

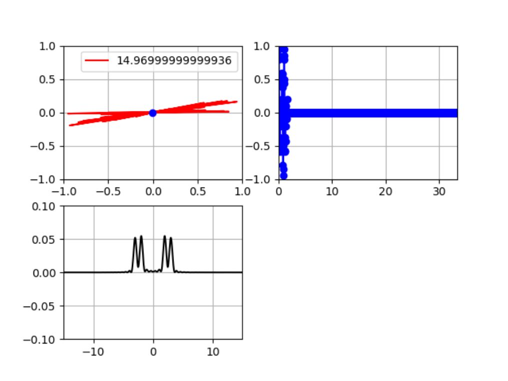
$f1 = 2$

$f2 = 3$

$fs = 10$

$N = 50$

$Z = 950$

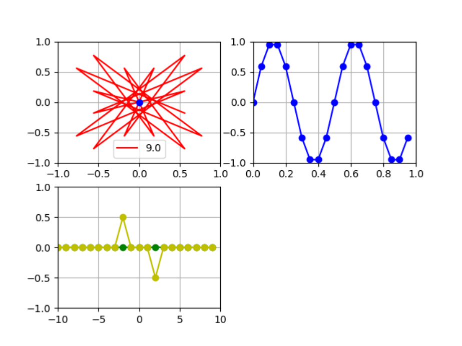


DFT

Acelerada

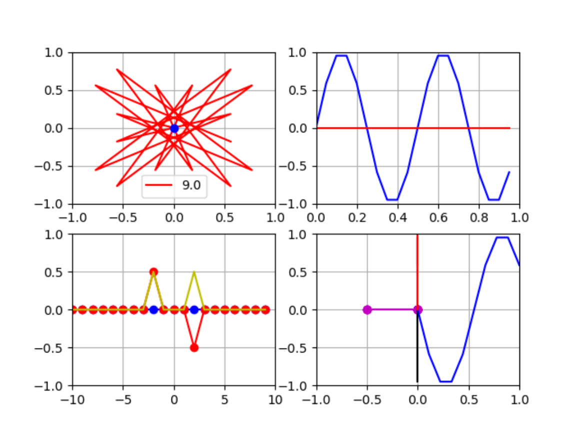
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from buttons import buttonOnFigure
#------------------------------------
fig        = plt.figure()
fs         = 20
N          = 20
#------------------------------------
circleAxe  = fig.add_subplot(2,2,1)
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 = 2
signalData=[]
def signal(f,n):
    return np.sin(2*np.pi*f*n*1/fs)
#    if n<50:
#        return 0.5*np.sin(2*np.pi*f*n*1/fs)+0.5*np.sin(2*np.pi*f*1.5*n*1/fs)
#    return 0
#------------------------------------
promAxe  = fig.add_subplot(2,2,3)
promRLn,promILn,promMagLn,promPhaseLn  = plt.plot([],[],'b-o',[],[],'r-o',[],[],'k-'
             ,[],[],'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(nn):
    global circleData,signalData,promData,frecIter,circleFrec,circleLg
```

```python
    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]))
    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=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=100 ,blit=True,repeat=True)
plt.get_current_fig_manager().window.showMaximized()
b=buttonOnFigure(fig,ani)
plt.show()
```

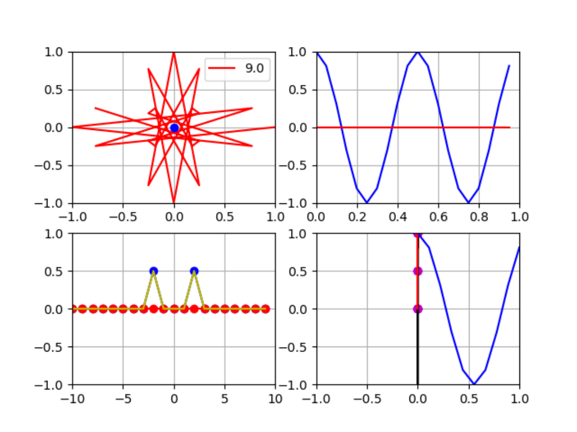


DFT para señales reales RDFT

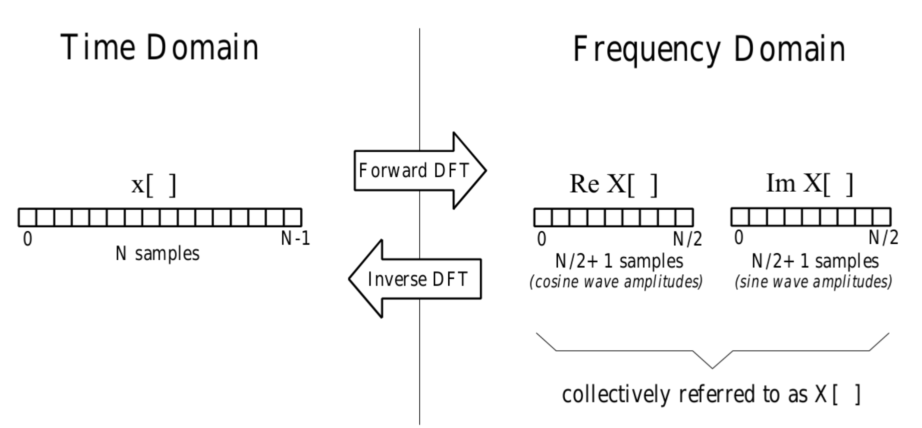
Sin



Cos



DFT para señales reales RDFT



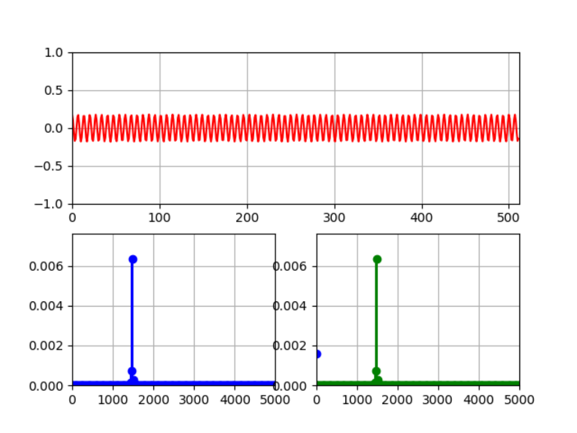
RDFT

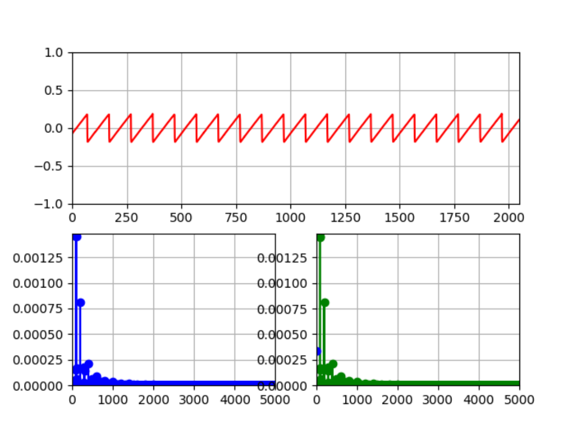
Análisis en la CIAA

```c
#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
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
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) {
      cyclesCounterReset();                                                       // inicializa el
          conteo de ciclos de reloj
      uartWriteByteArray ( UART_USB ,(uint8_t* )&adc[sample]    ,sizeof(adc[0]) );   // envia el sample
          ANTERIOR
      uartWriteByteArray ( UART_USB ,(uint8_t* )&fftOut[sample] ,sizeof(fftOut[0])); // envia la fft del
          sample ANTERIO
      //TODO hay que mandar fftLength/2 "+1" y solo estoy mandando fftLength/2. revisar
      adc[sample]   =(((int16_t )adcRead(CH1)-512)>>(10-BITS))<<(6+10-BITS);         // PISA el sample que
           se acaba de mandar con una nueva muestra
      fftIn[sample] = adc[sample];                                                 // copia del adc
          porque la fft corrompe el arreglo de entrada
      if ( ++sample==fftLength ) {                                                 // si es el ultimo
         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
                  estructira 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(LEDB);
            if((fftLength<<=1)>MAX_FFT_LENGTH)
               fftLength=32;
            while(gpioRead(TEC1)==0)
               ;
         }
      }
      while(cyclesCounterRead()< 20400) //clk de 204000000 => 10k samples x seg.
         ;
   }
}
```

RDFT

Análisis en la CIAA





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