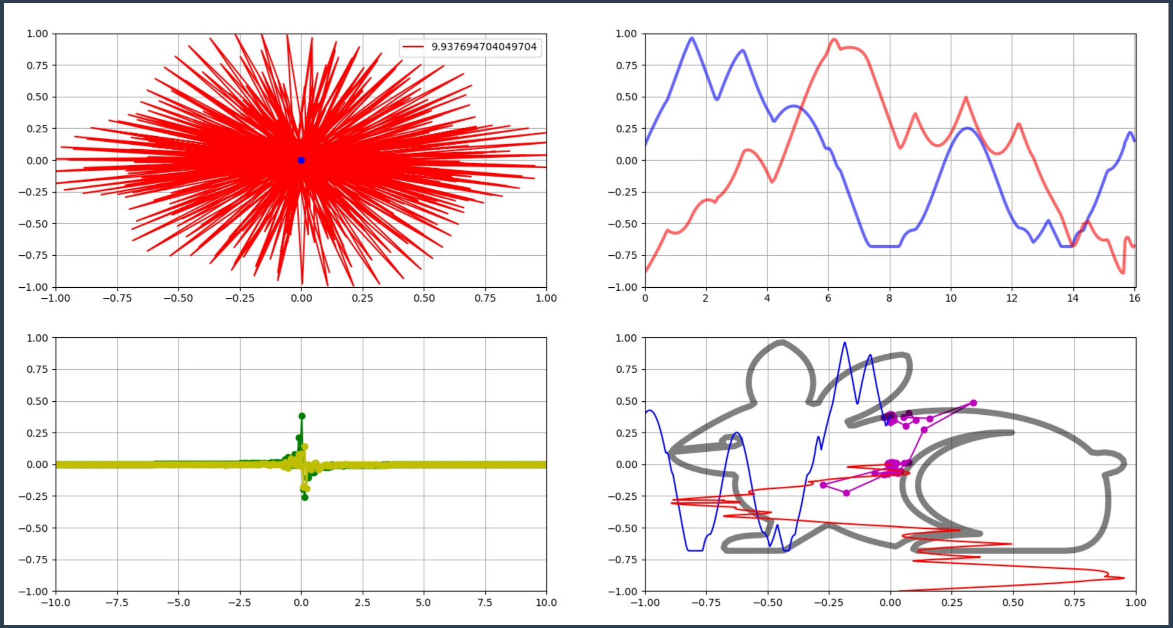
Procesamiento de señales. Fundamentos

Clase 4 – IDFT

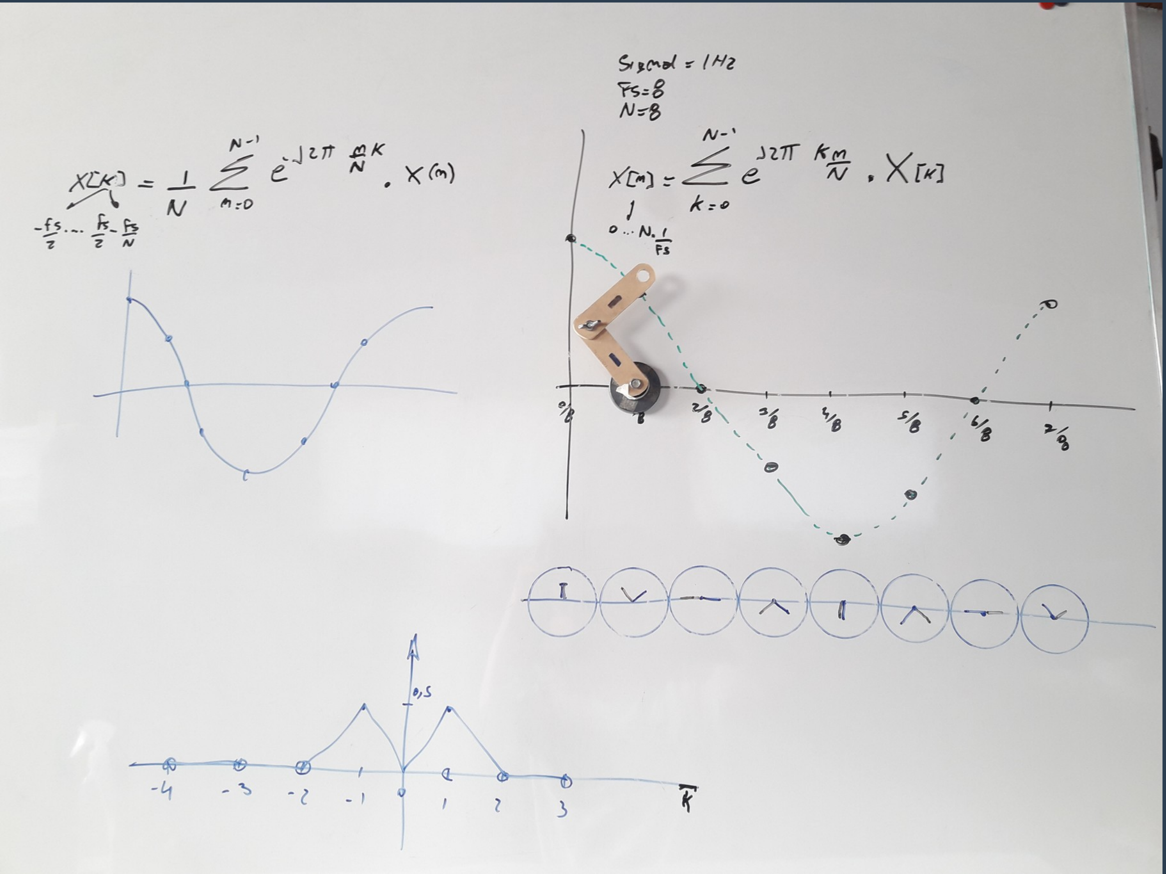
- I-DFT Transformada inversa discreta de Fourier.
- Maquina de I-Rei-Ruof
- DFT<>IDFT con Señales complejas
- IDFT con numpy
- Pares DFT<>IDFT relevantes
- IDFT con CMSIS-DSP



IDFT – Transformada inversa de Fourier

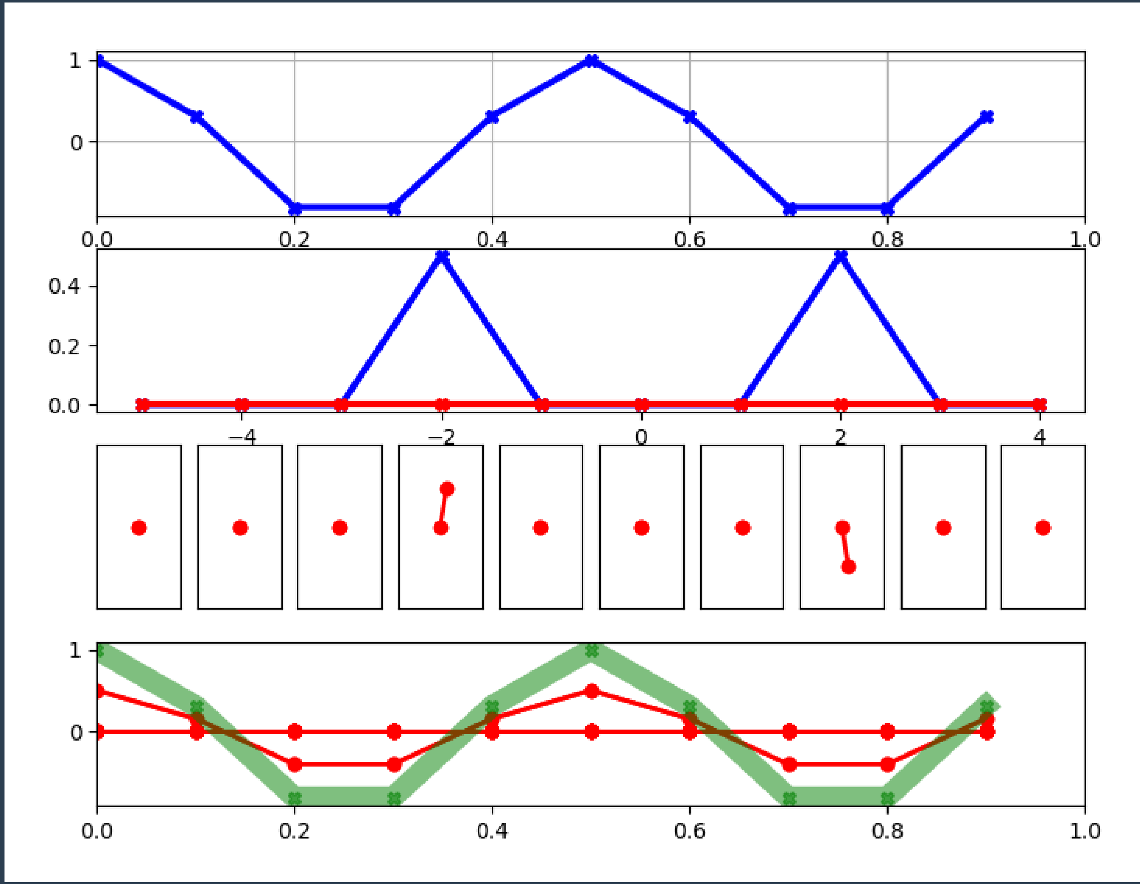
La maquina de I-Rei-Ruof

- Fs=8
- N=8
- Signal=1Hz
- Cada punto de la DFT se puede pensar como un circulo girando a una frecuencia k de radio **F**(k) y evaluado cada n/N segs => e^(2*pi*k*n/N)
- La función en el instante n/Fs será la suma **vectorial** de todos los círculos evaluados en n/N



I-Rei-Ruof con Python

- Fs=8
- N=8
- Signal=1Hz
- Cada punto de la DFT se puede pensar como un circulo girando a una frecuencia k de radio **F**(k) y evaluado cada n/N segs => e^(2*pi*k*n/N)
- La función en el instante n/Fs será la suma **vectorial** de todos los círculos evaluados en n/N



- Ver códigos: maquina_i_reiruof.py

Transformada Inversa Discreta de Fourier

- IDFT
- Notar la similitud con la ecuación de la DFT !
- $$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]\, e^{-j2\pi kn/N}$$
- Solo cambia el signo del exponente y el escalado
- Notar que ahora el resultado x[n] podría ser complejo.

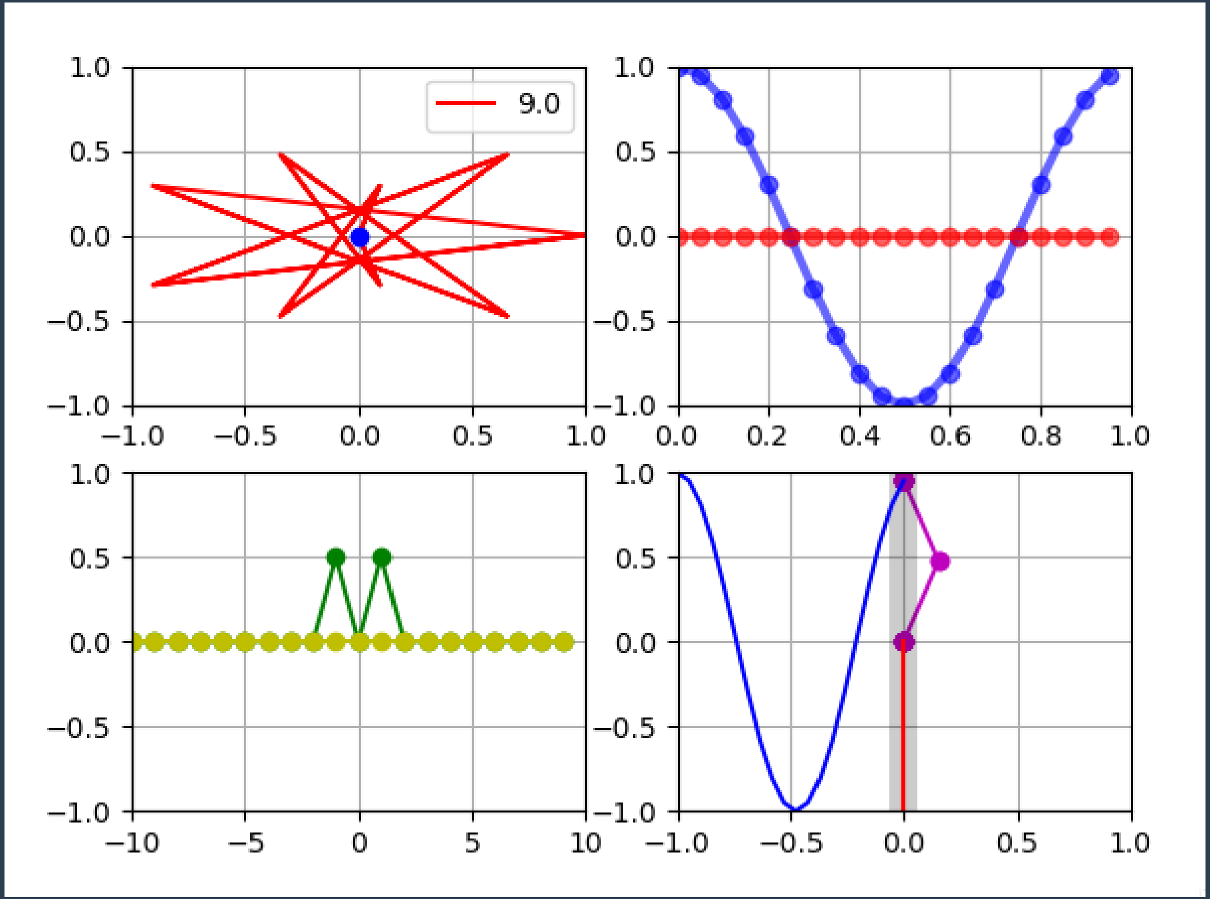
$$x[n] = \sum_{k=0}^{N-1} X[k]\, e^{j2\pi kn/N}$$

$$x[n] = \sum_{k=0}^{N-1} ReX[k]\Big(\cos(2\pi kn/N) + j\sin(2\pi kn/N)\Big) - \sum_{k=0}^{N-1} ImX[k]\Big(\sin(2\pi kn/N) - j\cos(2\pi kn/N)\Big)$$

- Ver códigos: maquina_reiruof.py

IDFT Real animado en Python

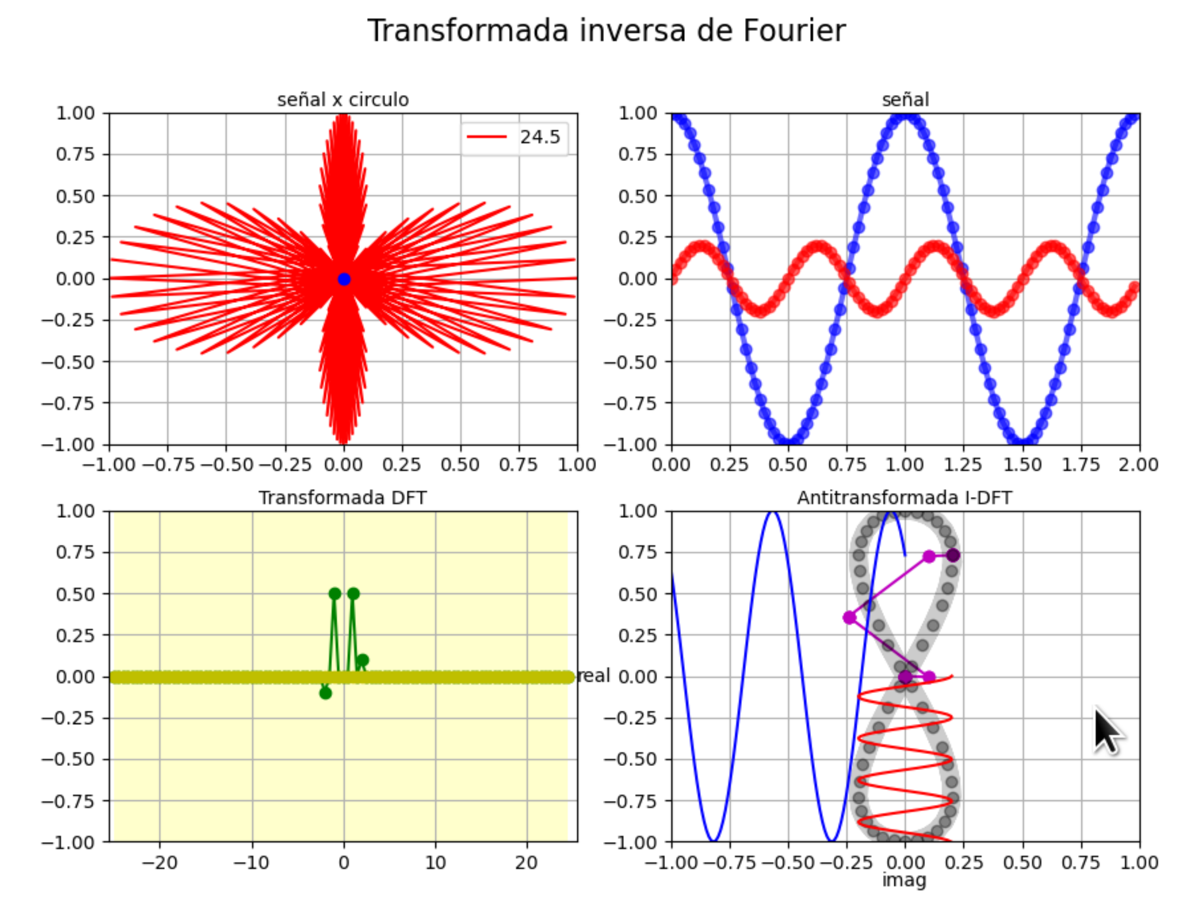
- Como reconstruyo la señal en tiempo desde la DFT??
- Cada bin en la DFT es un número complejo. Un vector.
- Si se suman vectorialmente todos los vectores se obtiene la señal en tiempo.
- El resultado es real... Seguro? En todos los casos?



- Ver código: idft.py

DFT<>IDFT de Señales complejas

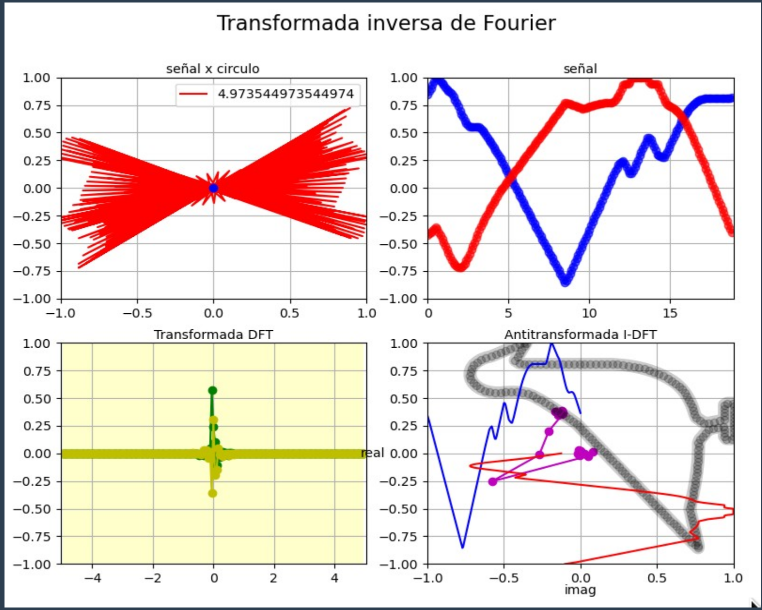
- Y si la entrada es compleja??
- También se puede hacer la DFT.
- El resultado sigue siendo un arreglo de N números complejos.
- La interpretación de los datos de entrada es arbitraria
- Se pueden desacoplar al reconstruir

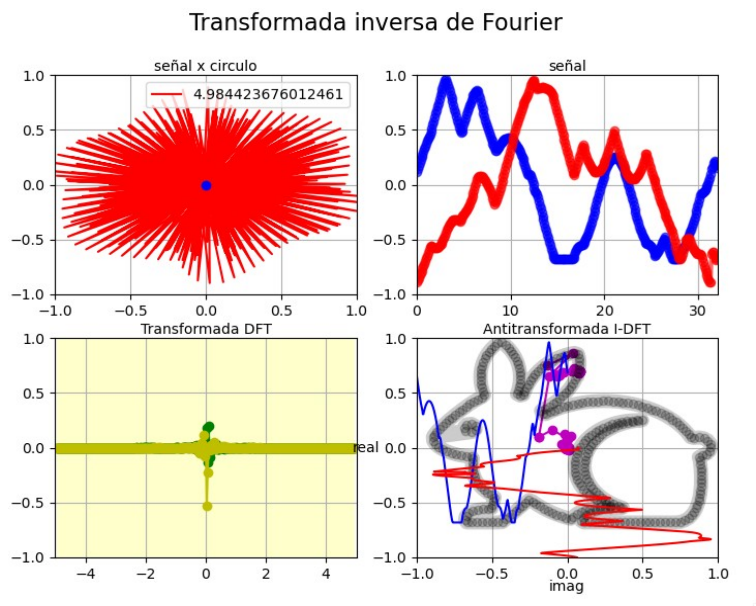


- Ver código: idft.py

Señales arbitrarias complejas

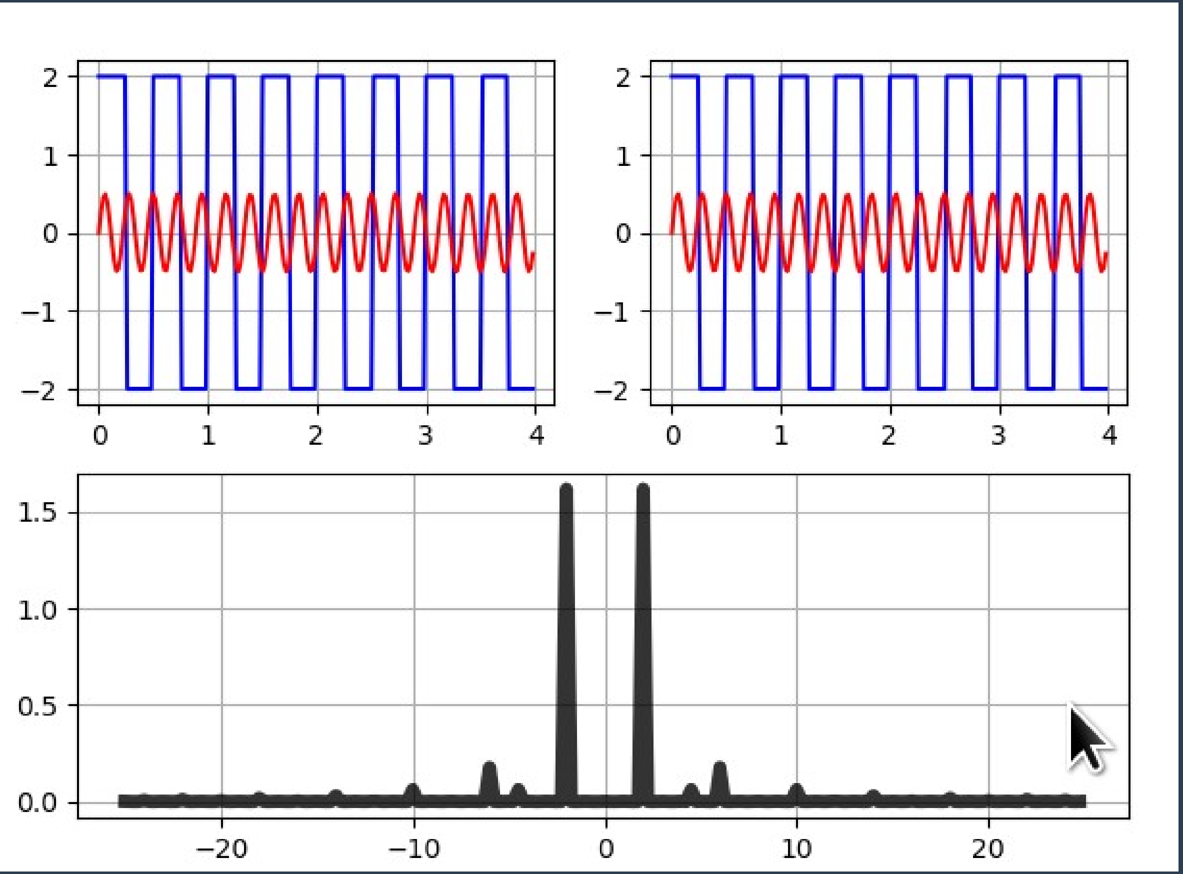
- Y si la entrada es es el trazo de un conejo o una paloma??
- La interpretación de los datos de entrada es arbitraria
- Si el objetivo es conocer de que se trata la figura, es necesario todo el espectro en F para reconstruir el conejo?
- Ver código: idft.py





I-DFT con numpy

- Modulo np.fft
- Función np.fft.idft
- Tener en cuenta la normalización

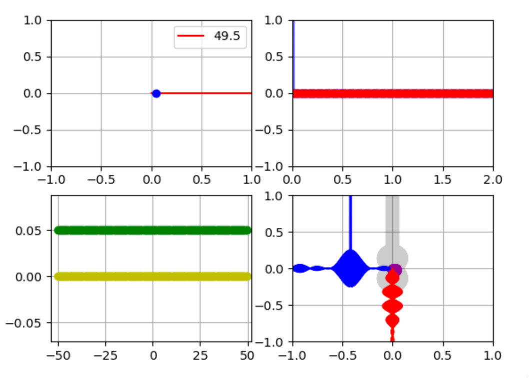


- Ver código: dft_idft_numpy.py

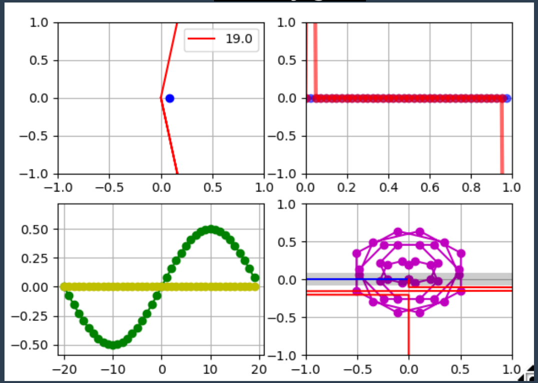
Pares DFT<>IDFT relevantes

- Hay formas de onda cuyas transformadas tienen características particulares.
- Cuadrada, delta y deltas conjugadas son algunas de ellas.
- Utilizar el codigo de ejemplo para familiarizarse con las propiedades y probar modificaciones

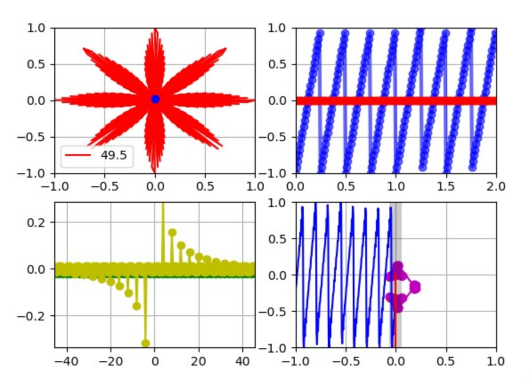
delta



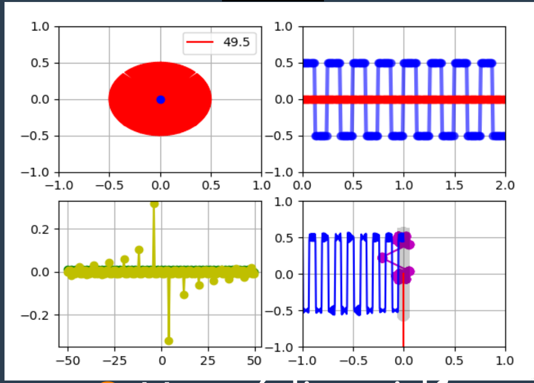
Par conjugado



Triangular



Cuadrada

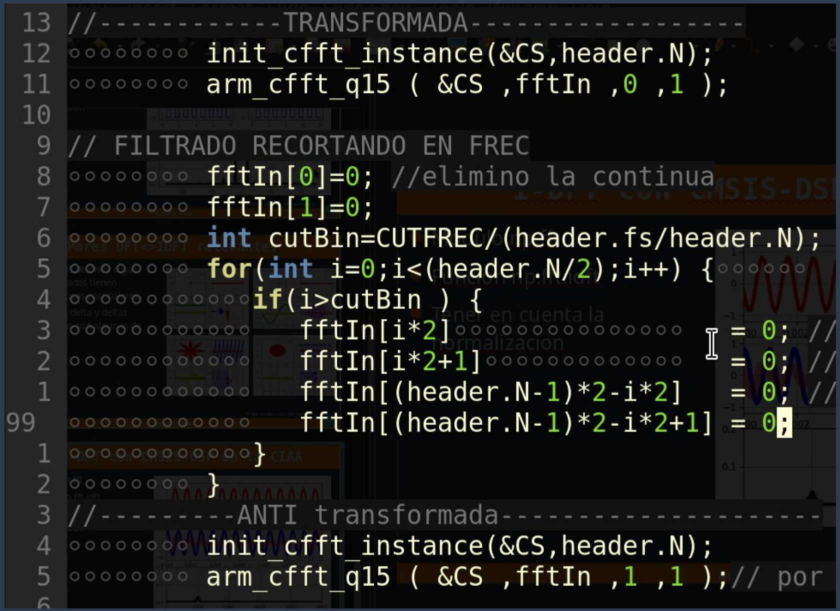


- Ver código: idft.py

IDFT con la CMSIS-DSP

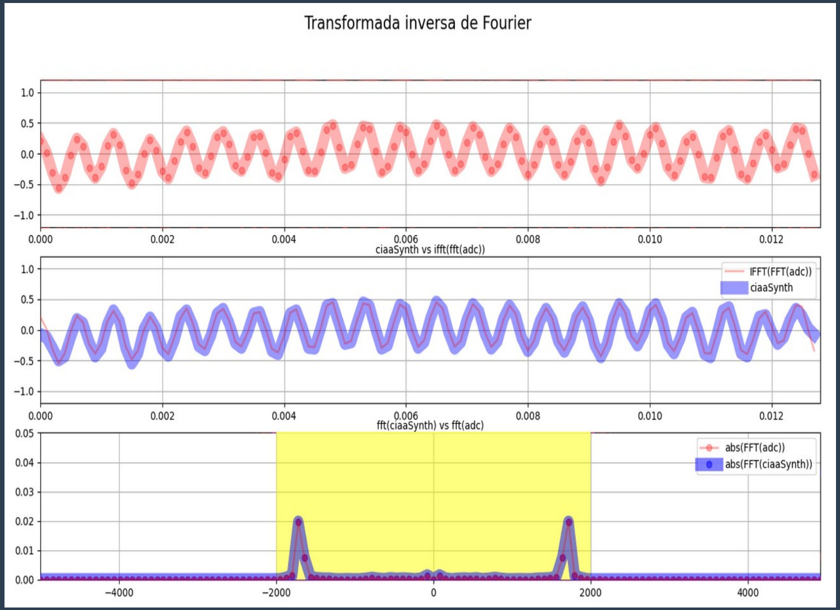
IDFT en la CIAA – Filtrado en F

- Calcula la fft usando cfft_q15
- Opcionalmente podría filtrar en frecuencia eliminando bines
- Calcula la i-fft y enviá solo la mitad de los datos porque el resto es complejo conjugado.
- Ojo que en función del N la salida de datos no es más q1.15 sino que se va corriendo, q2.14, q3.13, etc. La normalización se hace en Python



```
//------------TRANSFORMADA-----------------
        init_cfft_instance(&CS,header.N);
        arm_cfft_q15 ( &CS ,fftIn ,0 ,1 );

// FILTRADO RECORTANDO EN FREC
        fftIn[0]=0; //elimino la continua
        fftIn[1]=0;
        int cutBin=CUTFREC/(header.fs/header.N);
        for(int i=0;i<(header.N/2);i++) {
           if(i>cutBin ) {
              fftIn[i*2]                  = 0; //
              fftIn[i*2+1]                = 0; //
              fftIn[(header.N-1)*2-i*2]   = 0; //
              fftIn[(header.N-1)*2-i*2+1] = 0;
           }
        }
//--------ANTI transformada-------------------
        init_cfft_instance(&CS,header.N);
        arm_cfft_q15 ( &CS ,fftIn ,1 ,1 );// por
```



- Ver carpeta clase4/psf1