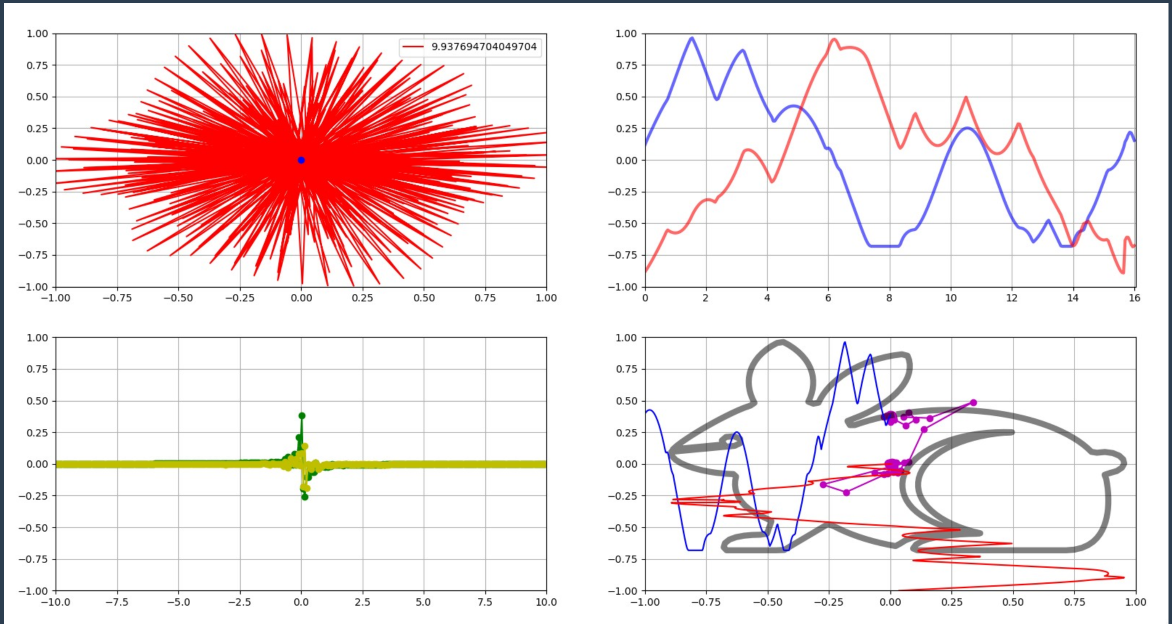
Procesamiento de señales. Fundamentos

Clase 4 – IDFT

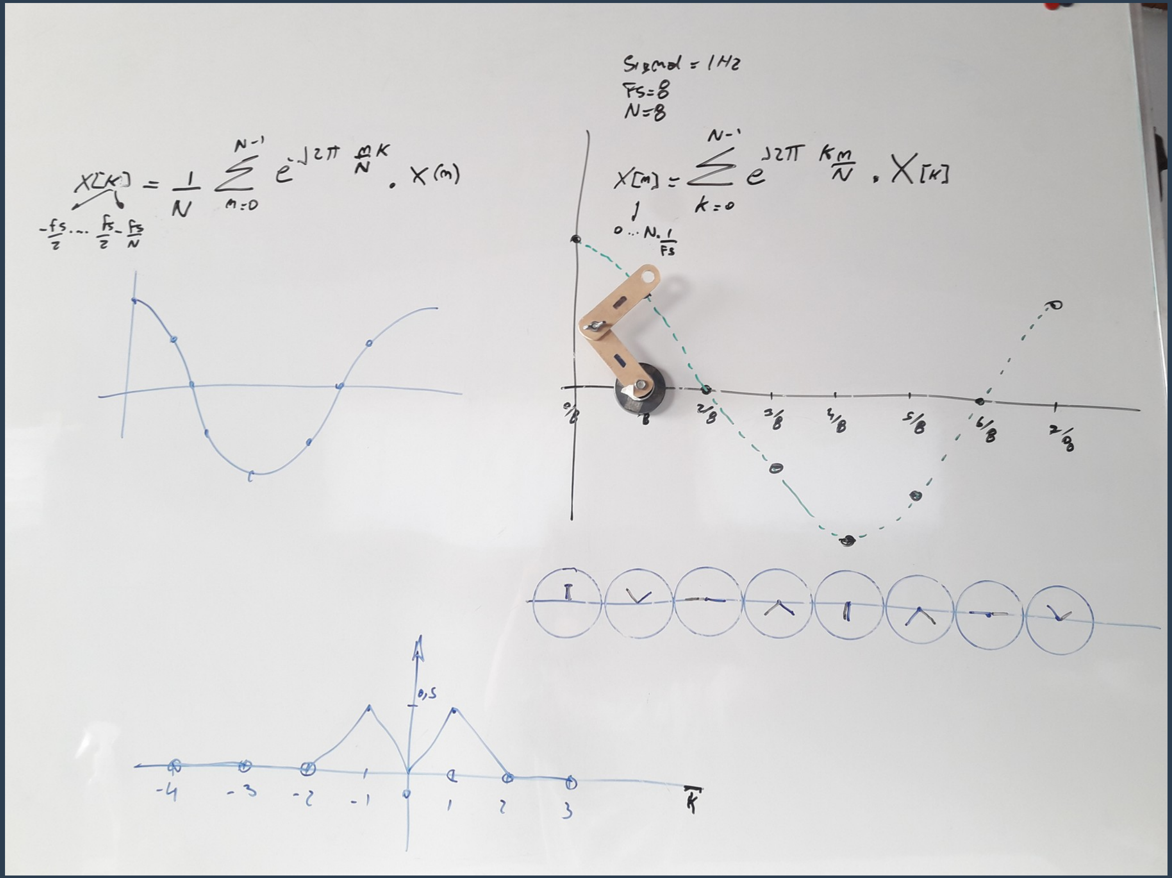
- I-DFT Anti transformada discreta de Fourier.
- Maquina de I-Rei-Ruof
- DFT<>IDFT con Señales complejas
- IDFT con numpy
- Pares DFT<>IDFT relevantes
- IDFT con CMSIS-DSP
- Correlación
- Correlación con CMSIS-DSP
- Preparación TP Final



IDFT – Transformada inversa de Fourier

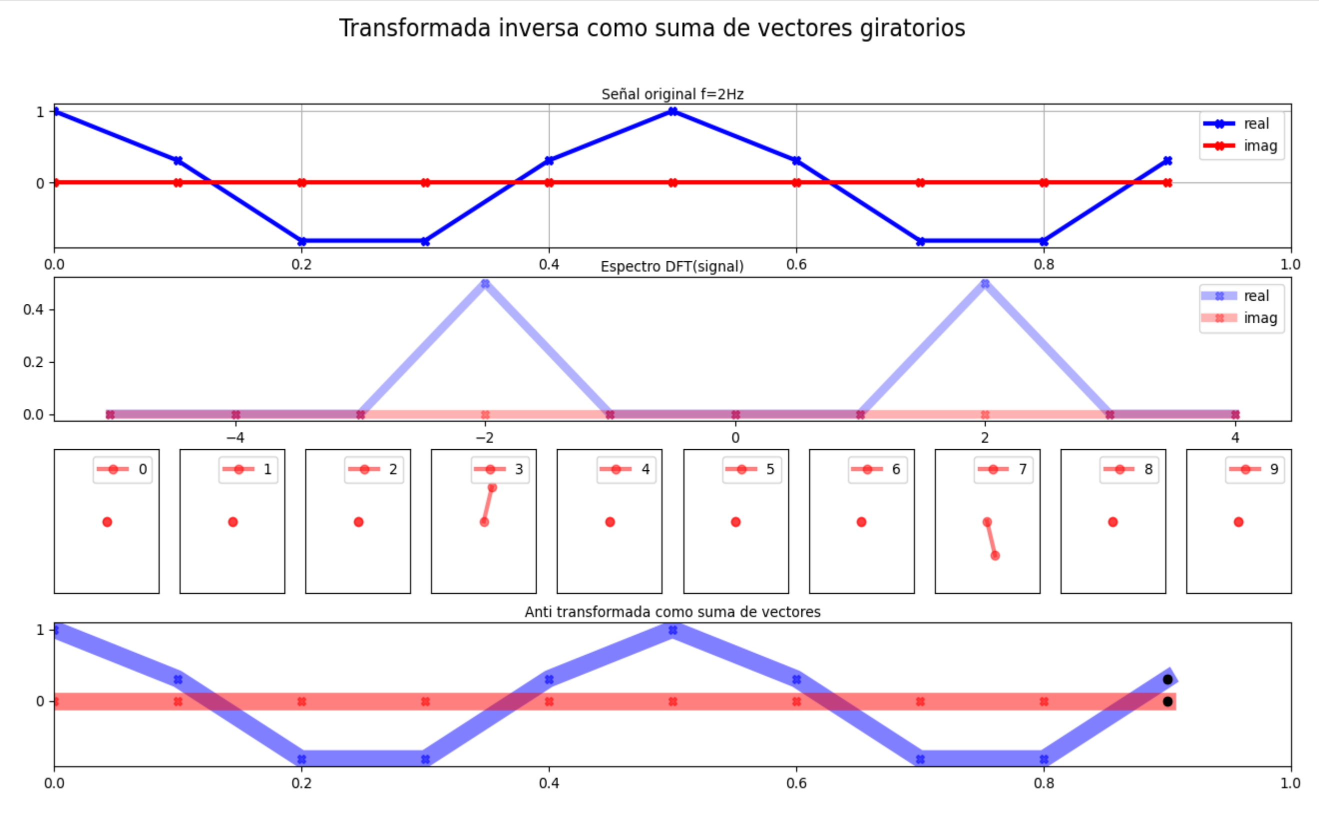
La maquina de I-Rei-Ruof

- 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=2Hz
- 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_output_side.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
- Segun el análisis que se hizo en DFT el resultado son n números complejos.

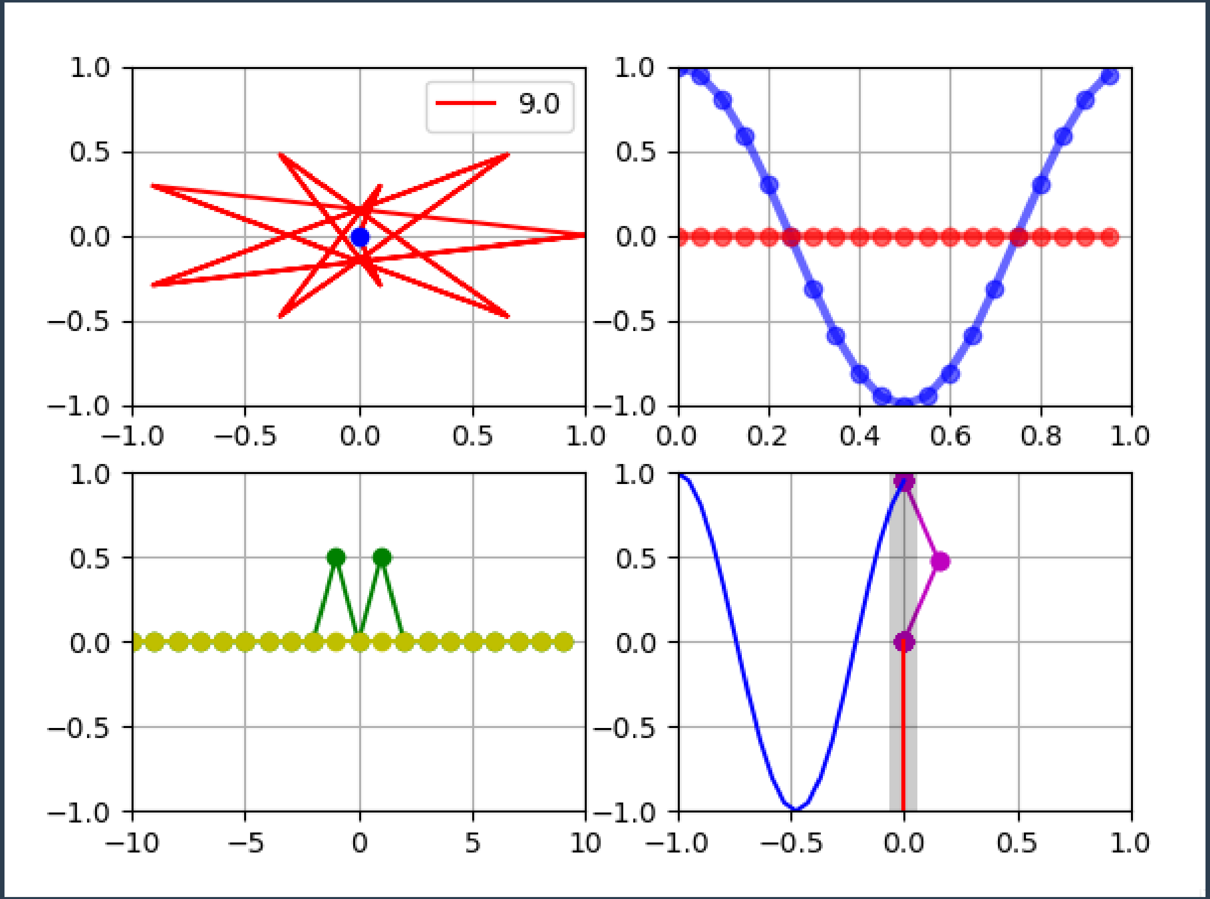
$$x[n] = \sum_{k=0}^{N-1} X[k]\,e^{j2\pi kn/N}$$

$$x[n] = \sum_{k=0}^{N-1} Re X[k]\left(\cos(2\pi kn/N) + j\sin(2\pi kn/N)\right)$$
$$- \sum_{k=0}^{N-1} Im X[k]\left(\sin(2\pi kn/N) - j\cos(2\pi kn/N)\right)$$

- Ver códigos: maquina_reiruof.py

IDFT Real animado en Python

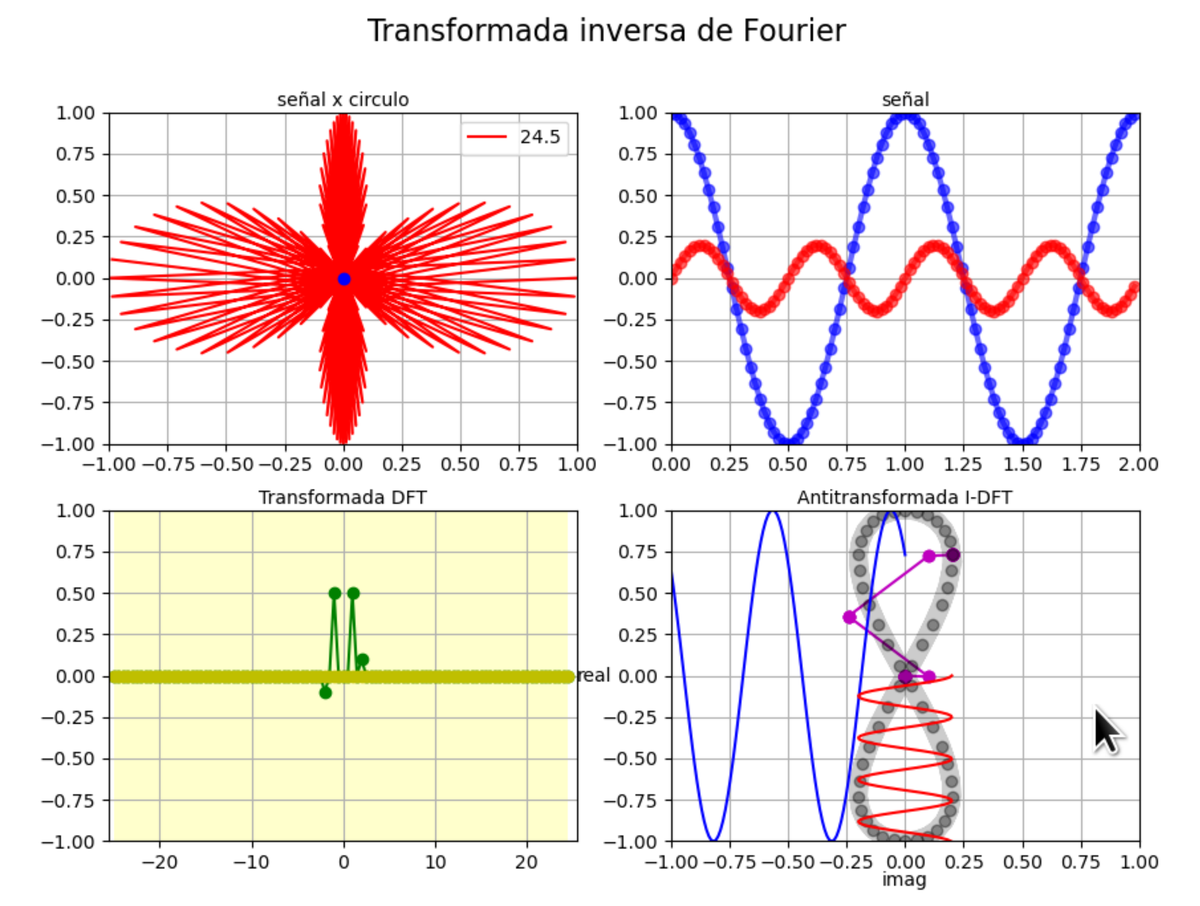
- Como reconstruyo la señal en tiempo desde la DFT??
- Cada bin en la DFT es un número complejo. Un vector.
- Si se suman vectorialmente todos los vectores se obtiene la señal en tiempo.
- El resultado es real ?... Seguro? En todos los casos?



- Ver código: idft.py

DFT<>IDFT de Señales complejas

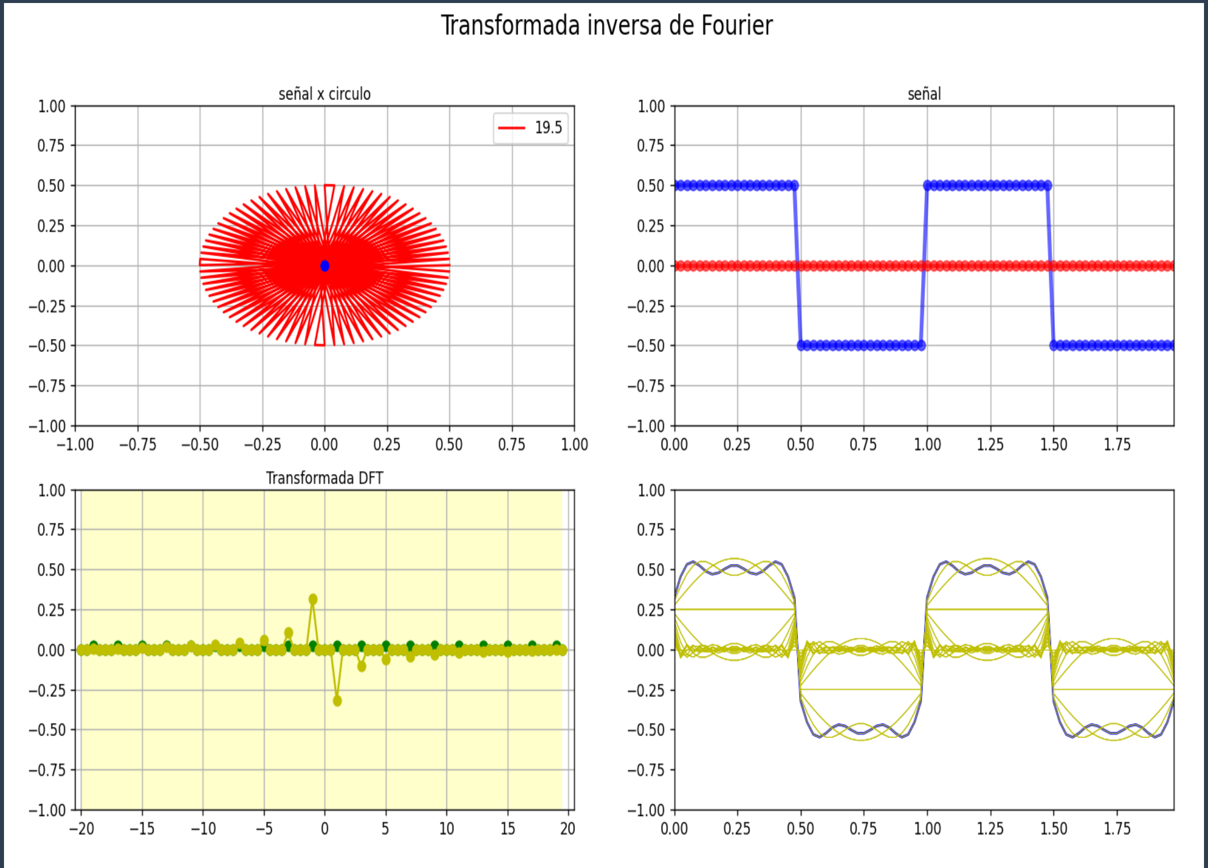
- Y si la entrada es compleja??
- También se puede hacer la DFT.
- El resultado sigue siendo un arreglo de N números complejos.
- La interpretación de los datos de entrada es arbitraria
- Se pueden desacoplar al reconstruir
- Habilita la posibilidad de utilizar una sola operación de FFT para 2 señales ‘reales’ independientes!
  - Sampleo presion y temperatura y mando la FFT de ambas



- Ver código: idft.py

IDFT como sumatoria de senos y cosenos

- Ejemplo en donde se reconstruye la señal en el tiempo como la sumatoria de senos y cosenos de las frecuencias indicadas por la IDFT

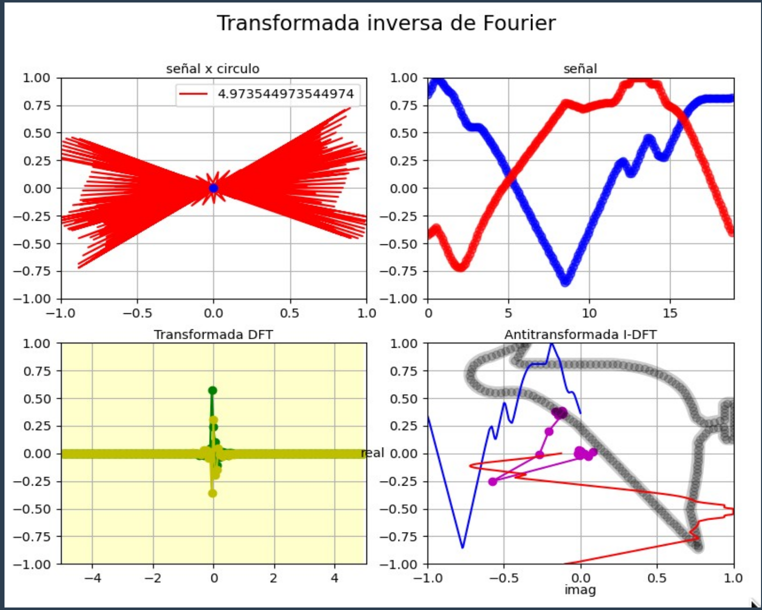


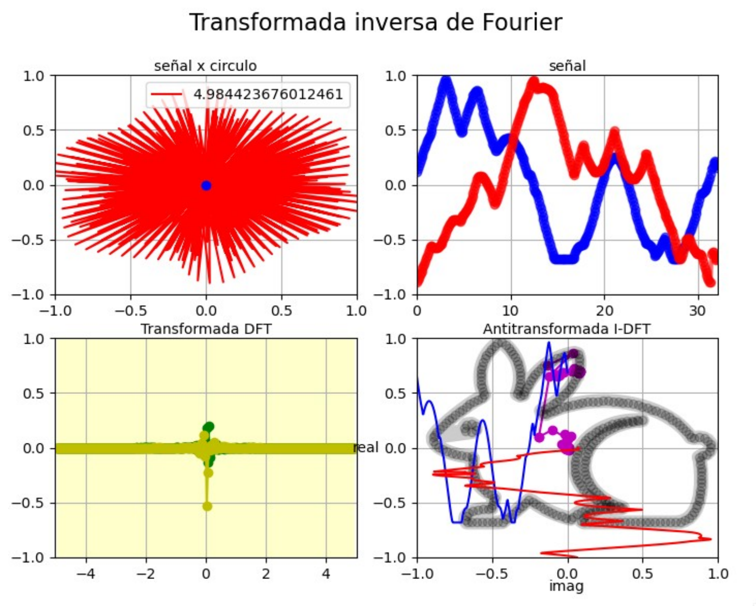
- Ver código: sumatoria.py

Señales arbitrarias complejas

- Y si la entrada es es el trazo de un conejo o una paloma??
- Si el objetivo es conocer de que se trata la figura, es necesario todo el espectro en F para reconstruir el conejo?
- En el espectro del conejo se puede ver que la mayor parte del espectro no tiene mucha informacion util => puedo recortar y ‘comprimir’

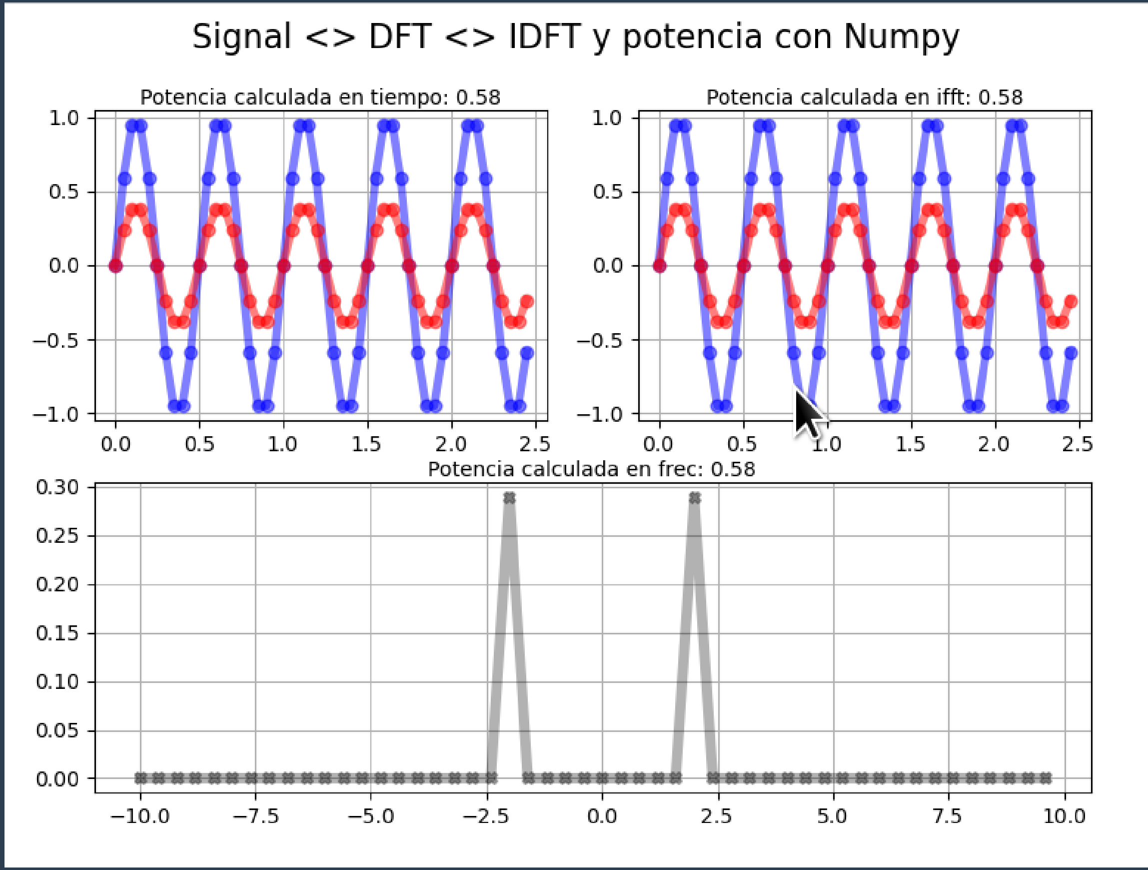
- Ver código: idft.py





I-DFT con numpy

- Modulo np.fft
- Función np.fft.idft
- Investigar el código para ver el calculo de potencia en tiempo y frecuencia
- Tener en cuenta la normalización

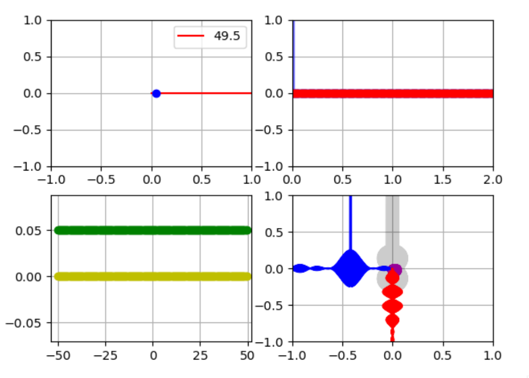


- Ver código: utils/dft_idft_numpy.py

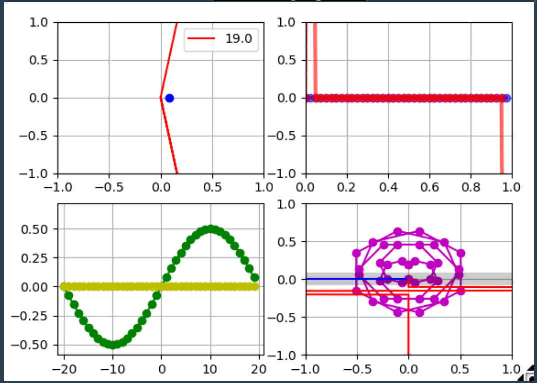
Pares DFT<>IDFT relevantes

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

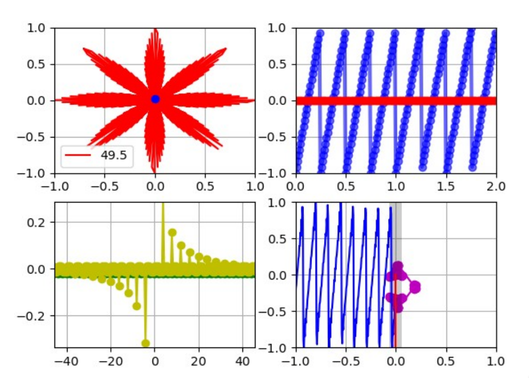
delta



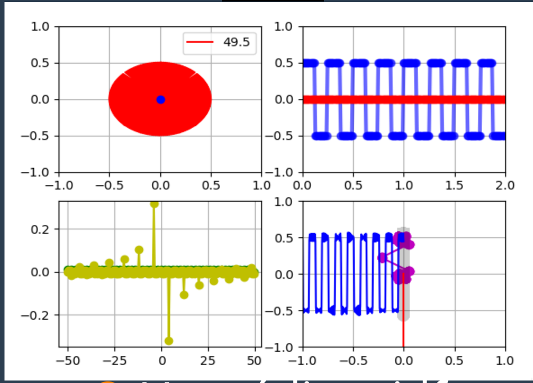
Par conjugado



Triangular



Cuadrada

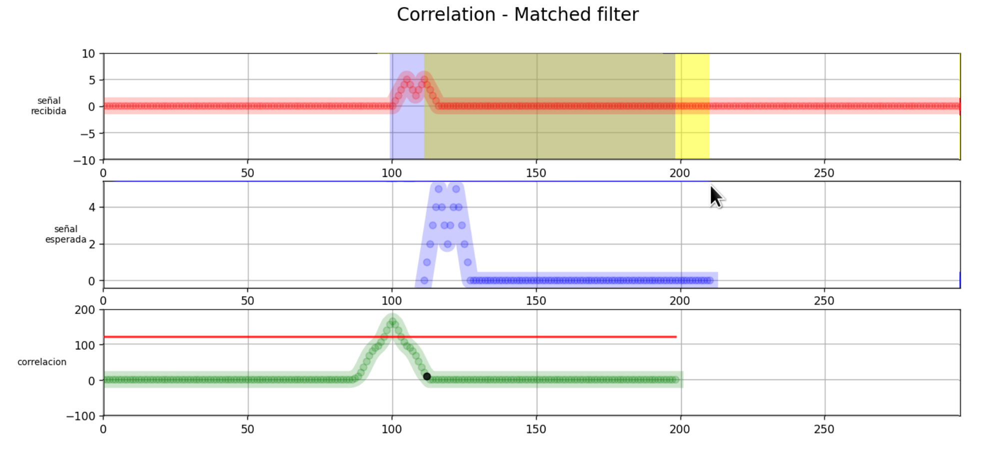


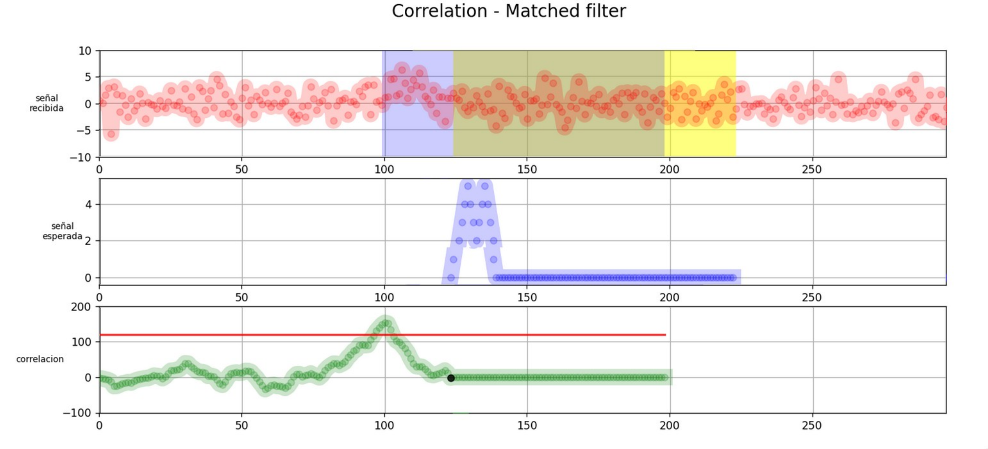
- Ver código: idft.py

Correlación

Correlación – Matched Filter

- Dada una señal que contiene a otra conocida, la correlación entre ambas es el metodo optimo para encontrarla esta ultima.
- Se suma el producto de la señal recibida y la que se espera encontrar para cada punto de la señal recibida.
- La señal conocida debera tener una longitud menor o igual que la recibida
- La DFT 'es' la correlación entre las funciones canónicas senos y cosenos.
- La DFT 'busca' de manera optima si la señal en analisis contiene senos y cosenos de diferentes frecuencias





- Ver código: correlacion.py

Correlación vs Fourier

- Notan algún parecido con la transformada de Fourier ???
- Notan algún parecido con la anti-transformada de Fourier ???
- Las transformadas de Fourier correlaciona una función circular con la señal.
- Notar que al avanzar k, el e^j… devuelve la siguiente posición del circulo que termina siendo un número (complejo), al igual que x [i – j]

$$y[i] = \sum_{j=0}^{M-1} h[j]\, x[i-j]$$

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n]\, e^{-j 2\pi kn/N}$$

$$x[n] = \sum_{k=0}^{N-1} X[k]\, e^{j 2\pi kn/N}$$

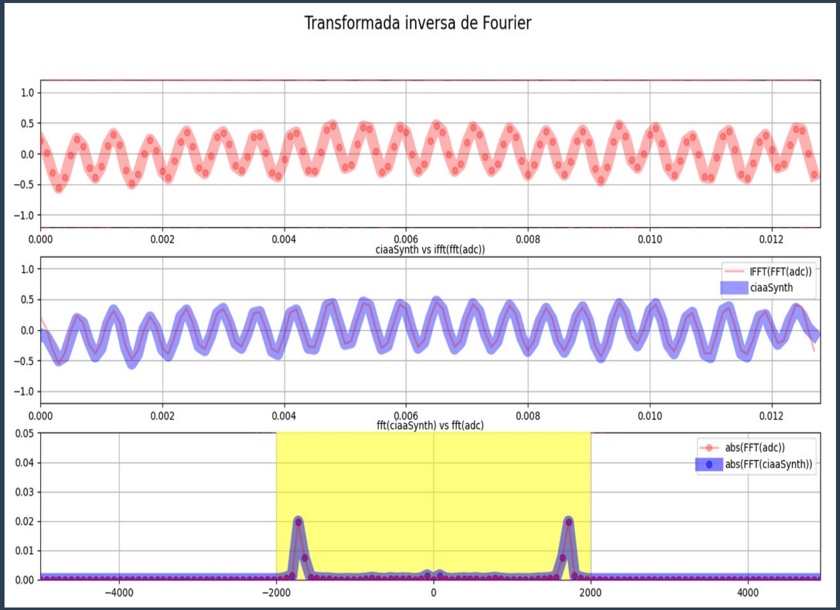
IDFT con la CMSIS-DSP

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

```c
//------------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
```



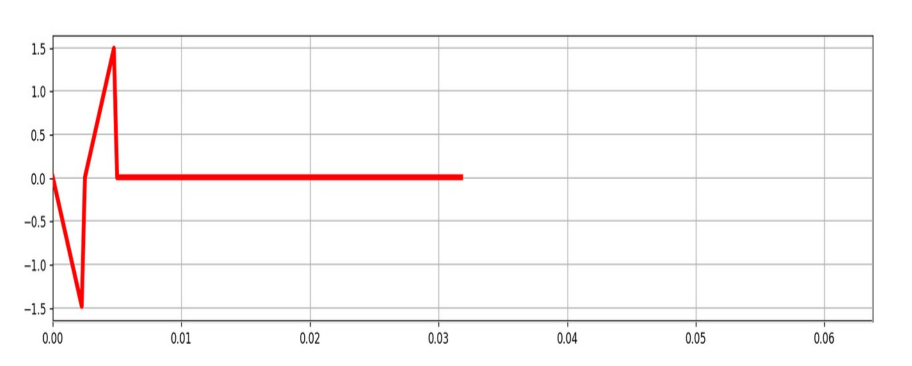
- Ver carpeta clase4/psf1

Correlación con la CMSIS-DSP

Correlación en CIAA – Señal esperada

- Genero una señal en la ciaa que espero recibir
- Sampleo y correlaciono la señal deseada con las muestras
- Busco el maximo en el vector de salida
- NOTA: Hay que poner a cero el vector de salida antes de calcular (leer la documentacion)

```
clearFloatBuf ( corrOut,2*header.N );
arm_correlate_f32 ( signal,header.N,adc ,header.N,corrOut   );
// trigger(2);
```



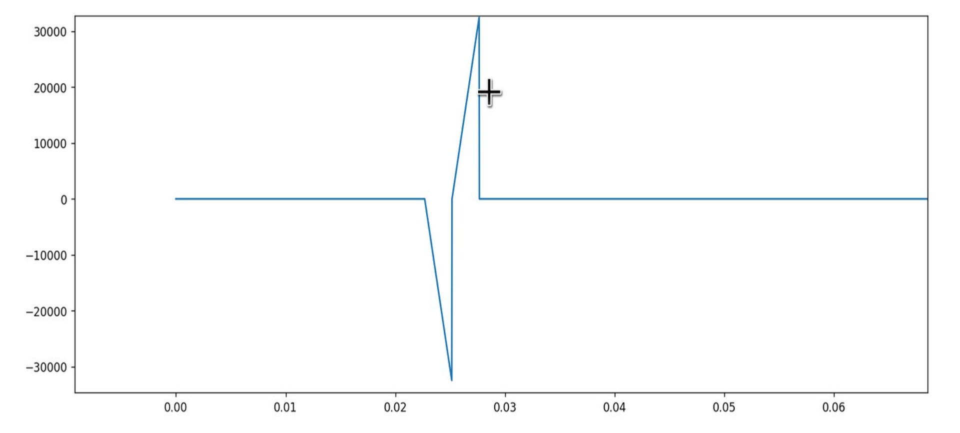

```
for(i=0;i<10;i++) {
   signal[i]=-i/10.0;
}
for(;i<20;i++) {
   signal[i]=(i-10)/10.0;
}
```

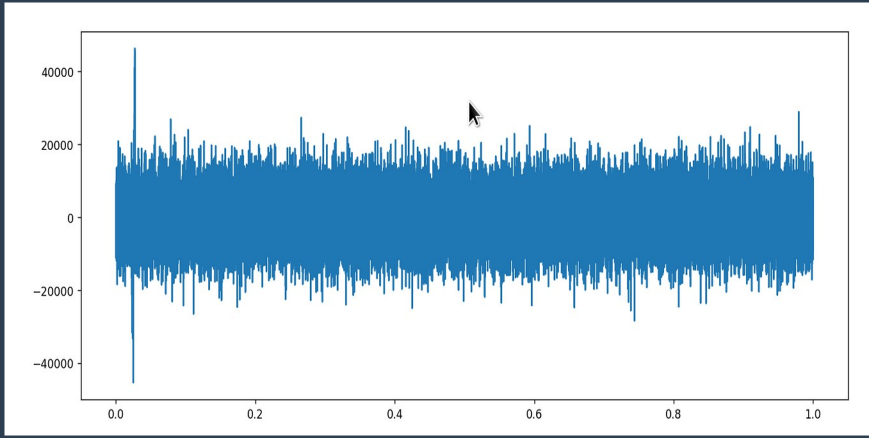
- Ver carpeta clase4/psf1

Correlación en CIAA – Señal enviada

- Genero una señal para transmitir con pulseaudio que contiene la señal eperada
- Agrego ruido en la transmision para probar la eficiencia del metodo
- Utilizo pulseaudio para enviar la trama

```python
note=np.zeros(len(t))
L=110
OFFSET=1000
for i in range(L):
    note[i+OFFSET]=-(2**15-1)*i/L
for i in range(L,L+L):
    note[i+OFFSET]=(2**15-1)*(i-L)/L
#note+=np.random.normal(0,((2**15)-1)/5,len(t))
```

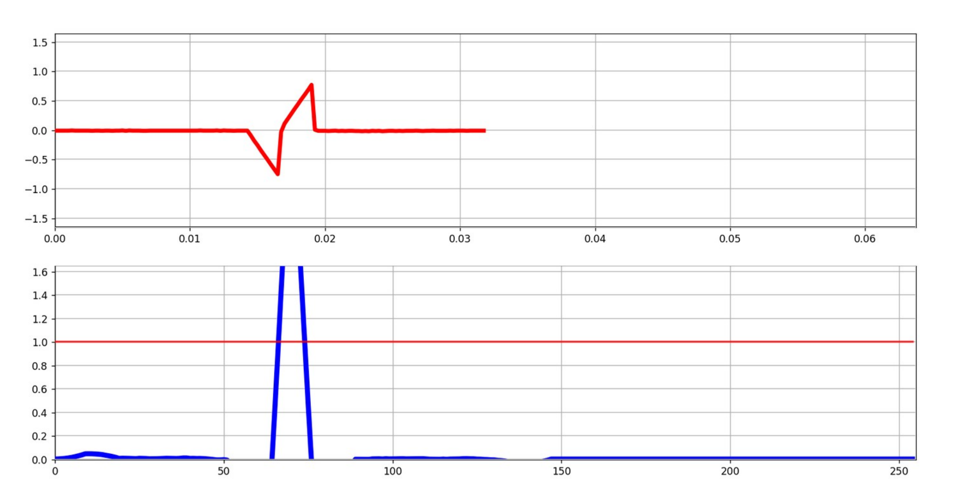


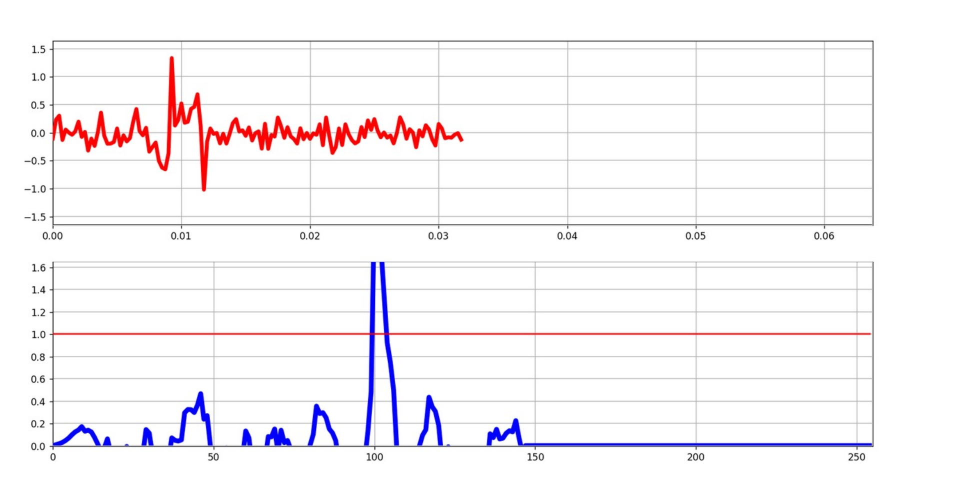


- Ver carpeta clase4/psf2/gen.py

Correlación en CIAA – Resultados

- Se puede ver el resultado de la correlacion con y sin el agregado de ruido
- Se puede ver a simple vista que la relacion señal a ruido de la señal correalacionada es mucho mayor que la original





- Ver carpeta clase4/psf2

Preparación TP Final

Preparacion TP final

- Indicar el tema del trabajo final
- Revisar los puntos que serán evaluados.
- Acotar el alcance a los tiempos disponibles para la materia
- Utilizar el espacio de consultas
- Link: https://forms.gle/ekNZUwKJQoqsmd6A9