TPNº2- AntiTransformada Discreta de Fourier

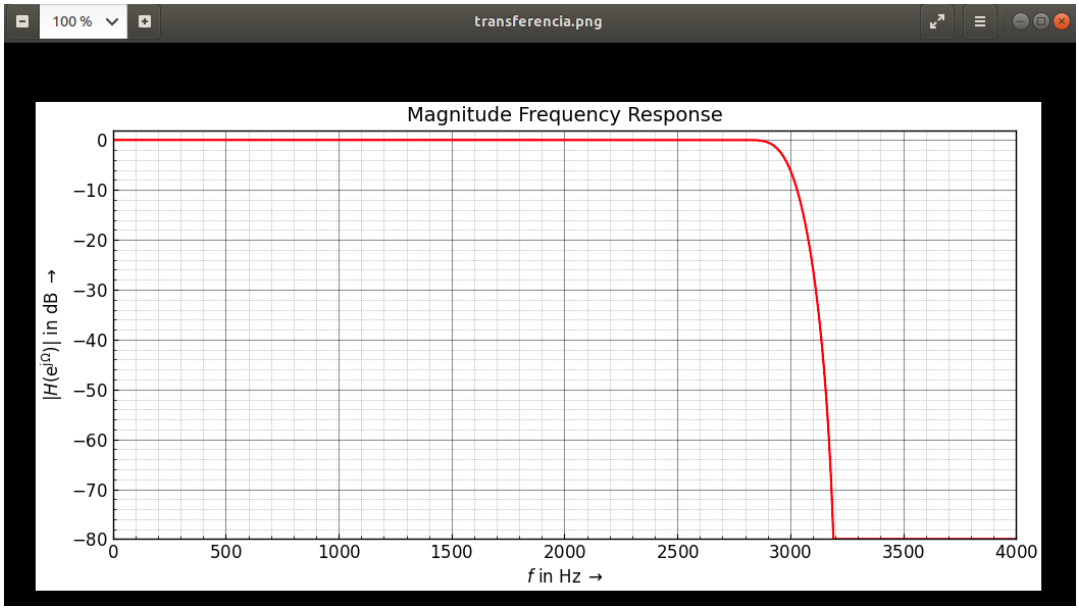
Autor: Pablo D. Folino
Link del repositorio: https://github.com/PabloFolino/MSE_PSF_TP2.git

Enunciado:
Dado el segmento de audio almacenado en el archivo clases/tp2/chapu_noise.npy con Fs=8000, mono de 16b y contaminado con ruido de alta frecuencia: 1) Diseñe un filtro que mitigue el efecto del ruido y permita percibir mejor la señal de interés 2) Filtre con la CIAA utilizando alguna de las técnicas vistas 3) Grafique el espectro antes y después del filtro. 4) Reproduzca el audio antes y después del filtro 5) Pegue el link a un .zip comentando los resultados y los criterios utilizados, la plantilla del filtro con capturas de la herramienta de diseño y un video mostrando la CIAA/HW en acción y la reproducción de audio antes y después del filtrado.

Se realiza un programa(**noise_folino_v4.py**) en donde se lee el archivo de referencia, se gráfica la señal de entrada su espectro, tiene la posibilidad de enviar la misma por la placa de sonido y brinda algunas características. Para visualizar los datos que provienen de la EDU-CIAA, se usa otro programa llamado **visualize_v3.py.** Por otro lado hay un tercer programa que se graba en la EDU-CIAA(**psf.c**). Tanto el primer programa como el último programa leen un archivo que modeliza el filtro h(t) con el programa **pyFDA**.

La señal de entrada posee un ruido de alta frecuencia, pero de frecuencia variable, con lo cual se optó en primer medida en hacer una filtra pasa bajos, con una frecuencia de corte de 2,5KHz(primero), y luego con una **fc**=3KHz. Con la primer opción se tuvo mejores resultados.

El filtro diseñado con 3KHz es el siguiente:



El programa **noise_folino_v3.py** posee un menú en donde se maneja la transmisión y la transforma y anti transformada en la PC.
En un pcio. se leen los archivos y se verifican distintas informaciones:

```
204  #============================================================
205  # Inicio del programa principal
206  #============================================================
207  global fData_rx,señal_ciaa
208  # Se leen los archivo
209  tData=np.load("../Informe/Archivos de enunciados/chapu_noise.npy")[::1]
210  hData=(genfromtxt("./xx/filtro_pasabajos.txt",skip_header=1)).astype(np.int16)
211
212
213  os.system("clear")
214
215  # Se calcula las longitudes
216  N=len(tData)
217  M=len(hData)
218  size_vector=M+N-1
219
220  # Se verifica que se leyeron los N elementos
221  print("\t Se pudieron leer los N={} elementos del archivo".format(N))
222  print("\t El formato de los elementos es={}".format(type(tData[0])))
223  print("\t Se pudieron leer los M={} elementos del filtro".format(M))
224  print("\t El formato de los elementos es={}".format(type(hData[0])))
225  print("\t La longitud de los vectores a antitransformar es={}".format(size_vector))
226
```

El menú que muestra el programa es el siguiente:

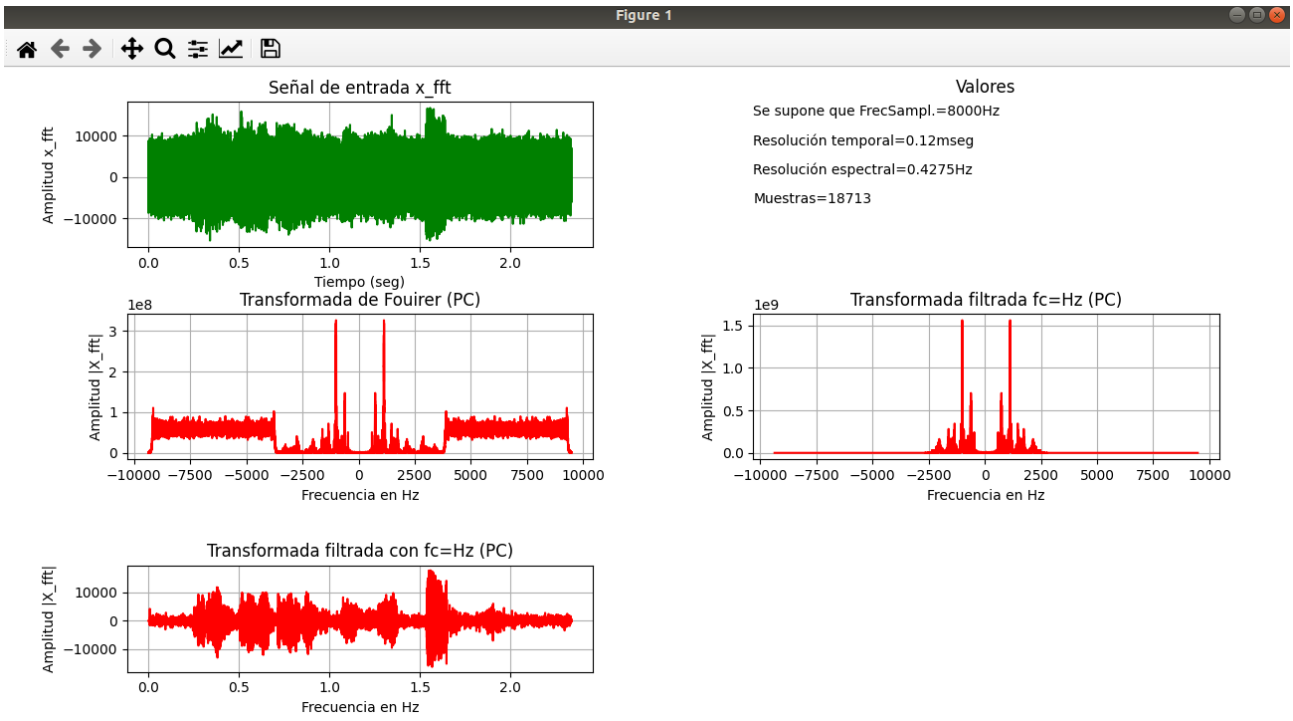
```
Programa para Tx señales a la EDU-CIAA por placa de sonido:

[1] Escuchar señal original(TP2)
[2] Escuchar señal filtrada(PC)(TP2)
[3] Visualizar gráfico(PC)
[4] Rx EDU-CIAA
[5] Escuchar Señal dela EDU-CIAA
[6] Borrar buffer de Rx

[7] TBD
[8] Seteo de frecuencia de la señal de entrada, número de muestras,
    frecuencia de sampling.
[9] Salir

Elija una opción:
```

El procesamiento de la señal se realiza en la PC, para tenerlo como referencia se la señal obtenida en la EDU_CIAA, a continuación se muestra parte del código principal:

```
228  # Datos extendidos
229  tData_ext=np.zeros(size_vector)
230  hData_ext=np.zeros(size_vector)
231  for i in range(0,N,1):
232      tData_ext[i]=tData[i]
233  for i in range(0,M,1):
234      hData_ext[i]=hData[i]
235
236
237  # Genero el vector de estados temporales
238  n_Data      = np.arange(0,N,1) #arranco con numeros enteros para evitar errores de float
239  t_Data      = n_Data*Ts
240
241  # Transformada
242  f_Data=np.arange(-N/2,N/2+M-1)
243  fftData  = np.fft.fft(tData_ext)
244  ffthData = np.fft.fft(hData_ext)
245  YData=fftData*ffthData/10000                # Multiplico en frecuencia
246
247  ifftData = np.fft.ifft(YData)               # Antitransformada
248  fftData  = np.fft.fftshift(fftData)
249  YData    = np.fft.fftshift(YData)
250
```

Importante: de ***pyFDA***se exportó el filtro en vez de **float**, en **enteros signados**, con lo cual hay un cambio de escala(es algo que me dí cuenta a último momento).

Se grafica en la PC los datos procesados y la señal de entrada:

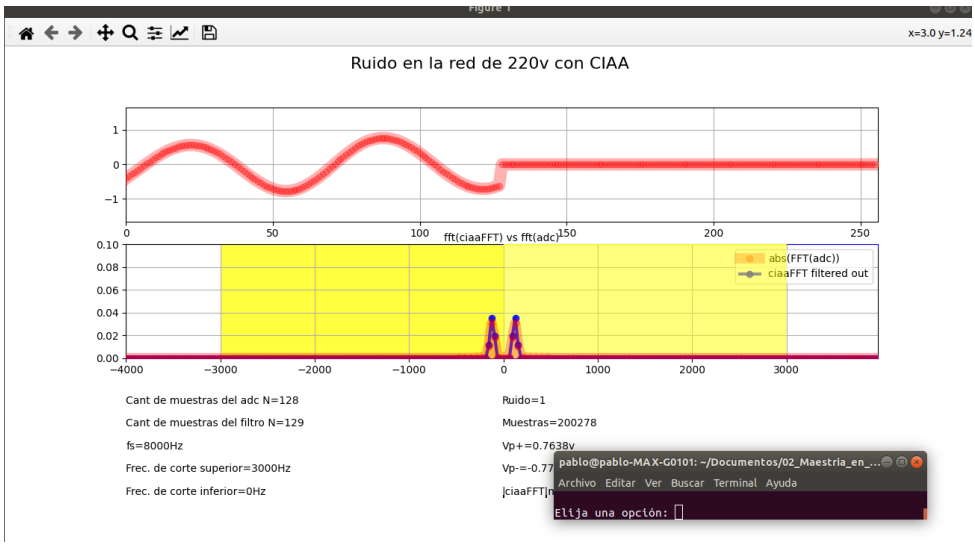


En la EDU-CIAA se programa con el siguiente código, se observa que se envía la frecuencia de corte inferior y superior del filtro(en este caso como es un pasa bajos la fci=0Hz)

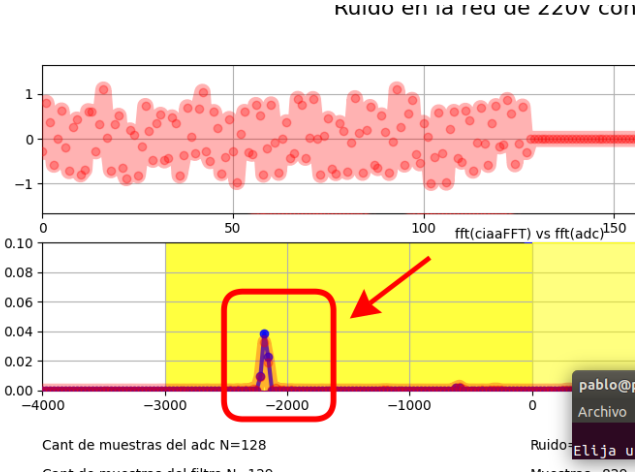
```
#define CUTFREC2    0       // Frecuencia de corte inferior del filtro
#define FS          8000

#define NIVEL_SENIAL 1      // Umbral de ruido

struct header_struct {
   char     pre[8];
   uint32_t id;
   uint16_t N;
   uint16_t fs ;
   uint16_t cutFrec ;
   uint16_t cutFrec2 ;
   uint16_t senial ;
   uint16_t M ;
   char     pos[4];
} __attribute__ ((packed)); //importante para que no paddee

struct header_struct header={"*header*",0,128,FS,CUTFREC,CUTFREC2,ON,h_LENGTH,"end*"};
```

```
while(1) {
   cyclesCounterReset();

   adc[sample]         = (((int16_t )adcRead(CH1)-512)>>(10-BITS))<<(6+10-BITS); // PISA el sample que se acaba de mandar con una nueva m
   fftInOut[sample*2]   = adc[sample];                                           // copia del adc porque la fft corrompe el arreglo de en
   fftInOut[sample*2+1] = 0;                                                     // parte imaginaria cero

   if(header.senial==true){
      header.senial=false;
      gpioToggle ( LED2 );
   }

   if ( ++sample>=header.N ) {
      gpioToggle ( LEDR );                        // este led blinkea a fs/N

//------------TRANSFORMADA-----------------
      init_cfft_instance ( &CS,(header.N+h_LENGTH-1)); //512. 256 esto tienen que ser power of 2
      arm_cfft_q15       ( &CS ,fftInOut ,0 ,1      ) ;

//------------MAGNITUD-----------------
      arm_cmplx_mag_squared_q15 ( fftInOut ,fftAbs ,(header.N+h_LENGTH-1 ));

//------------FILTRADO MULTIPLICANDO ESPECTROS-----------------
      arm_mult_q15 ( fftAbs,HAbs ,fftAbs ,(header.N+h_LENGTH-1));

      for (uint16_t i=0; i<(header.N+h_LENGTH-1);i++ ) {
          if(NIVEL_SENIAL<fftAbs[i]) {
              header.senial=true;
          }
      }


      header.id++;
      uartWriteByteArray ( UART_USB ,(uint8_t*)&header ,sizeof(struct header_struct ));

      for (sample=0; sample<(header.N+h_LENGTH-1);sample++ ) {
         uartWriteByteArray ( UART_USB ,(uint8_t* )&adc[sample]      ,sizeof(adc[0]) );     // envia el sample ANTERIOR
         uartWriteByteArray ( UART_USB ,(uint8_t* )&fftAbs[sample*1] ,sizeof(fftInOut[0])); // envia la fft del sample ANTERIO
      }

      sample = 0;
      adcRead(CH1); //why?? hay algun efecto minimo en el ler sample.. puede ser por el blinkeo de los leds o algo que me corre 10 puntos

   }
   // gpioToggle ( LED1 );                                  // este led blinkea a fs/2
   while(cyclesCounterRead()< EDU_CIAA_NXP_CLOCK_SPEED/header.fs) // el clk de la CIAA es 204000000
      ;
}
```

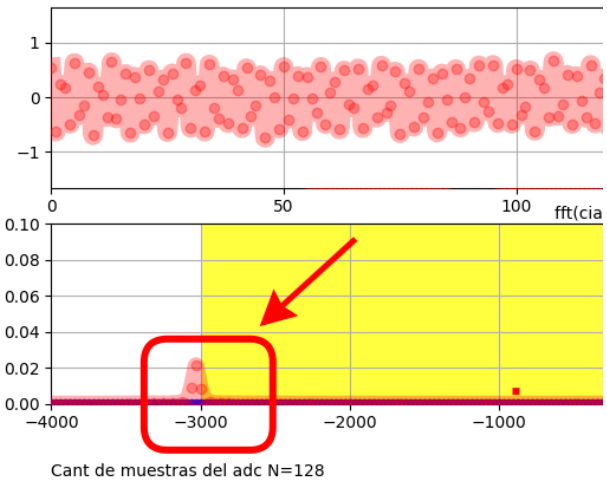
Para visualizar se usa el programa **visualize_v3.py** el cual se muestra a continuación:



Se observa en la franja amarilla los límites de frecuencia del pasa bajos. Cuando la frecuencia se encuentra dentro del pasabajos, la señal que proviene de la EDU-CIAA filtrada (trazo azul), se observa en la gráfica conjuntamente con la señal no filtrada (trazo rojo).



Cuando la señal sale del pasabajos solamente queda la señal no filtrada(trazo rojo).



Se adjunta el sonido de la señal con y sin el filtro.

- Señal sin filtro
- Señal con filtro(3Khz)

Con un filtro de 2,5Khz se mejora notablemente el audio, aunque queda una señal de ruido de bajo nivel. Para mejorar esto se podría promediar, y luego amplificar.