Procesamiento de señales. Fundamentos

Clase 7 – Repaso, TP final y tips & tricks

- Arreglos de números con python
- Zona útil de convolución
- Antialias digital + downsampling
- Automatic gain control
- Centro de masas en espectro
- Deconvolucion
- Afinador de guitarra con CIAA

Arreglo de números en numpy

- Atención con la definición de un rango de números float en numpy.
- Por ejemplo:

```
7 import numpy as np
6 #-------------------------------
5 fs             = 17
4 N              = 12
3 arange2 = np.arange(0,N/fs,1/fs)
```

- Esta definición que debería tener 12 valores, termina con 13 !!

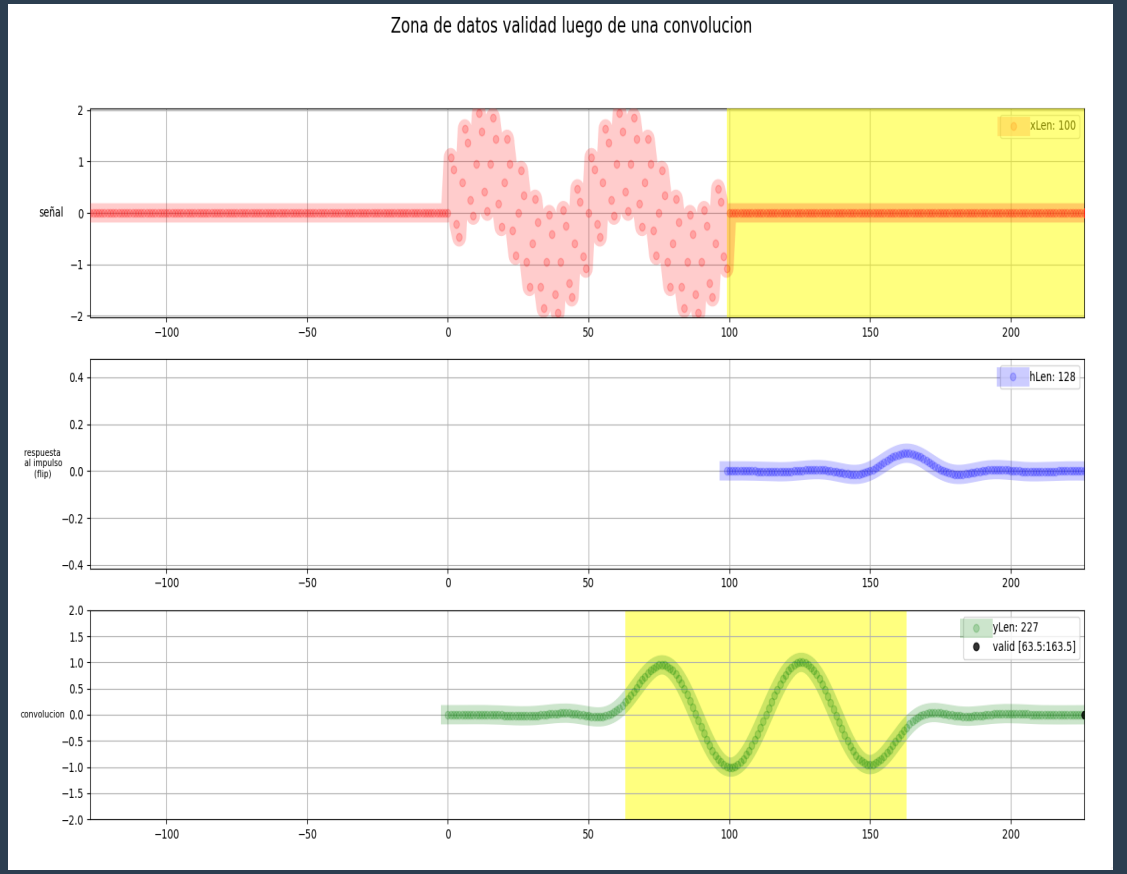
```
5 arange2: [0.         0.05882353 0.11764706 0.17647059 0.23529412 0.29411765
4  0.35294118 0.41176471 0.47058824 0.52941176 0.58823529 0.64705882
3  0.70588235]
2 len: 13
```

- La manera correcta es:

```
9 import numpy as np
8 #-------------------------------
7 fs             = 17
6 N              = 12
5 arange1 = np.arange(0,N,1)/fs
```

Zona útil luego de una convolución

- Luego de realizar la convolución, la primera y ultima zona de la salida no son utilizables porque el sistema esta en fase de estabilización.
- Si la h[n] es simétrica, se puede ver que el largo de esa zona es justo la mitad del largo de h[n]
- Entonces la zona útil sera y[(M-1) / 2 : (M-1) / 2 +N].
- Se puede ver que el largo útil de y[n] sera también N.
- En la imagen se ve una entrada con 100 puntos con dos senoides de 2 y 20hz, esta ultima es filtrada y la salida describe dos ciclos justo con 100 puntos

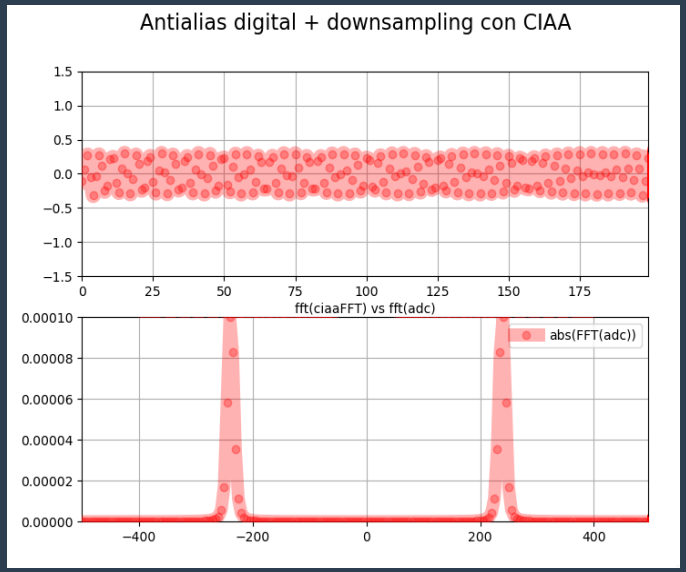


- Ver código: convolución_util.py

Antialias digital + downsampling

- Para reducir el orden del FAA analógico es posible oversamplear y luego filtrar en digital
- Se samplea a una frecuencia donde la atenuación del FAA es razonable
- Luego se filtra en digital con todos los datos
- Se descartan el inicio y el final de la salida filtrada para quedarse con la zona útil
- Luego se hace un downsampling y se obtiene la señal deseada lista para seguir el procesado

```
if(sample<header.N)
   uartWriteByteArray ( UART_USB ,(uint8_t* )&adc[sample] ,sizeof(adc[0]) );
adc[sample]          = (((int16_t )adcRead(CH1)-512)>>(10-BITS))<<(6+10-BITS); // PISA

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

   //---------------filtrado antialias en digitial-------------------------
   arm_conv_fast_q15  ( adc,header.N*overSample,h,h_LENGTH,y);

   //---------------downsampling-------------------------
   for(int i=0;i<header.N;i++){
      adc[i]=y[i*overSample+garbageOffset];
   }

   uartWriteByteArray ( UART_USB ,(uint8_t*)&header ,sizeof(struct header_struct ));
   header.id++;
   sample = 0;
   adcRead(CH1); //why?? hay algun efecto minimo en el 1er sample.. puede ser por el b
}
gpioToggle ( LED1 );                                          // este led blinkea a

//---------------over sampling-------------------------
while(cyclesCounterRead()< EDU_CIAA_NXP_CLOCK_SPEED/(header.fs*overSample)) // el clk
   ;
```



- Ver carpeta: clase7/ciaa/psf1

Automatic gain control

- Cuando lo que se busca es separación en frecuencia, puede ser útil escalar los samples para aprovechar al máximo el rango dinámico del sistema de números elegido
- En este sentido se propone un sencillo algoritmo para utilizalo luego del sampleo para llevar al máximo la excursión de salida
- Esto mantiene acotado el error en los cálculos, pero podría también amplificar el ruido. Utilizar con discreción

```
17 void agc(int16_t* adc,uint16_t len)
16 {
15    int max=0;
14    int i,gainFactor;
13    for(i=0;i<len;i++) {
12       int abs=adc[i]>0?adc[i]:-adc[i];
11       if(abs>max)
10          max=abs;
 9    }
 8    gainFactor=0x8000/max;
 7    for(i=0;i<len;i++) {
 6       adc[i]*=gainFactor;
 5    }
 4    return;
 3 }
```

```
---------------calculo maximo de la
c(adc,header.N*overSample);
```

- Ver carpeta: clase7/ciaa/psf2

Out of memory en la CIAA

- En la medida que los algoritmos se hacen mas complejos, se hace necesario contar con suficiente RAM para almacenar los datos
- La primera sugerencia es usar en cuanto se pueda el stack y reutilizar
- También verificar las funciones de la CMSIS-DSP para elegir aquellas en las cuales las salidas se sobrescriben en el mismo vector (siempre que sea posible)
- En el caso de la CIAA tener en cuanta que en el linker script las zonas de RAM están segmentadas como se ve en la figura
- Se pueden reservar espacio para ciertos vectores en dichas zonas según sea el caso para aprovechar los recursos disponibles

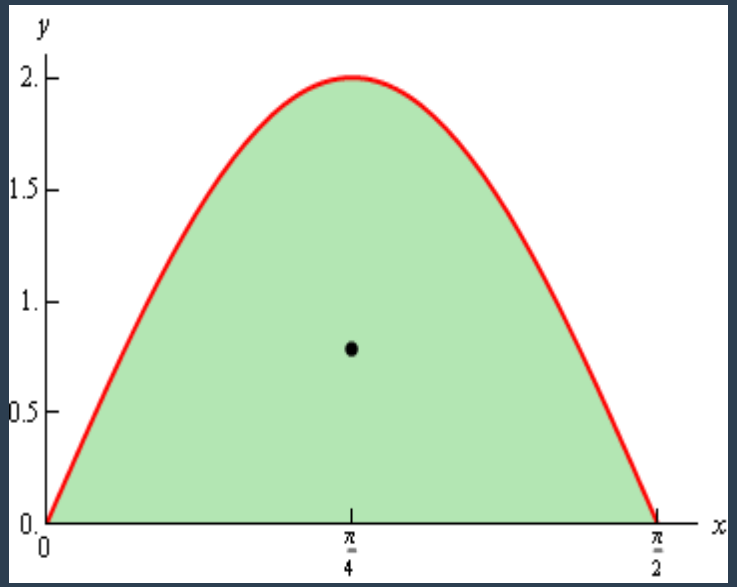
```
LD ../clases/7_clase/ciaa//psf3/out/psf3.elf...
/usr/lib/gcc/arm-none-eabi/11.2.0/../../../../arm-none-eabi/bin/ld: ../clases/7_clase/ci
aa//psf3/out/psf3.elf section `.bss' will not fit in region `RamLoc32'
/usr/lib/gcc/arm-none-eabi/11.2.0/../../../../arm-none-eabi/bin/ld: region `RamLoc32' ov
erflowed by 94588 bytes
```

```
MEMORY
{
  /* Define each memory region */
  MFlashA512 (rx) : ORIGIN = 0x1a000000, LENGTH = 0x80000 /* 512K bytes */
  MFlashB512 (rx) : ORIGIN = 0x1b000000, LENGTH = 0x80000 /* 512K bytes */
  RamLoc32 (rwx) : ORIGIN = 0x10000000, LENGTH = 0x8000 /* 32K bytes */
  RamLoc40 (rwx) : ORIGIN = 0x10080000, LENGTH = 0xa000 /* 40K bytes */
  RamAHB32 (rwx) : ORIGIN = 0x20000000, LENGTH = 0x8000 /* 32K bytes */
  RamAHB16 (rwx) : ORIGIN = 0x20008000, LENGTH = 0x4000 /* 16K bytes */
  RamAHB_ETB16 (rwx) : ORIGIN = 0x2000c000, LENGTH = 0x4000 /* 16K bytes */
}
```

- Ver file: ciaa/libs/lpc_open/lib/mem.ld

Interpolador para buscar el máximo

- Cuando se busca una frecuencia en particular con la FFT, los puntos validos estarán dados por la resolución espectral, esto es fs/N.
- Sin embargo podría mejorarse sutilmente el resultado si se pudiera interpolar los bines vecinos con tal de encontrar un punto intermedio que represente mejor la posición del máximo
- Si bien se podría conseguir un resultado similar con una FFT mas larga no siempre es posible
- Una manera simple es calcular el centro de masas en la zona de interés. Esta técnica permite obtener una mejora aproximación inter-bin a costa de un calculo muy simple



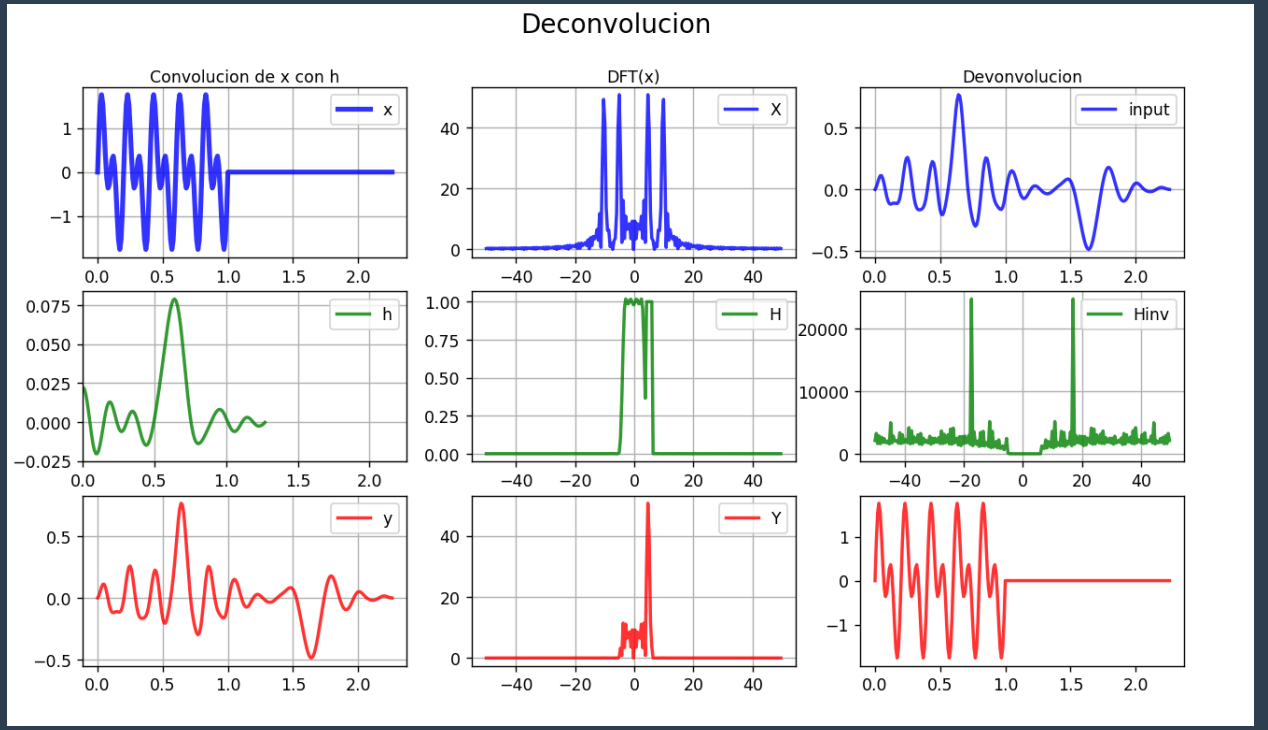
$$X_{com} = \frac{\sum x_i m_i}{\sum m_i}$$

- Ver código: 7_class/ciaa/psf3.py

Deconvolucion

Deconvolucion

- Dado un sistema en donde aparece una convolucion indeseada, es posible deconvolucionar la salida de ese sistema y recuperar la señal original.
- Siempre y cuando se cuente con el h del sistema
- Para lograrlo se utiliza el teorema de la convolucion y se hace la multiplicacion de la salida del sitema por 1/H, (donde H es la dft de h)

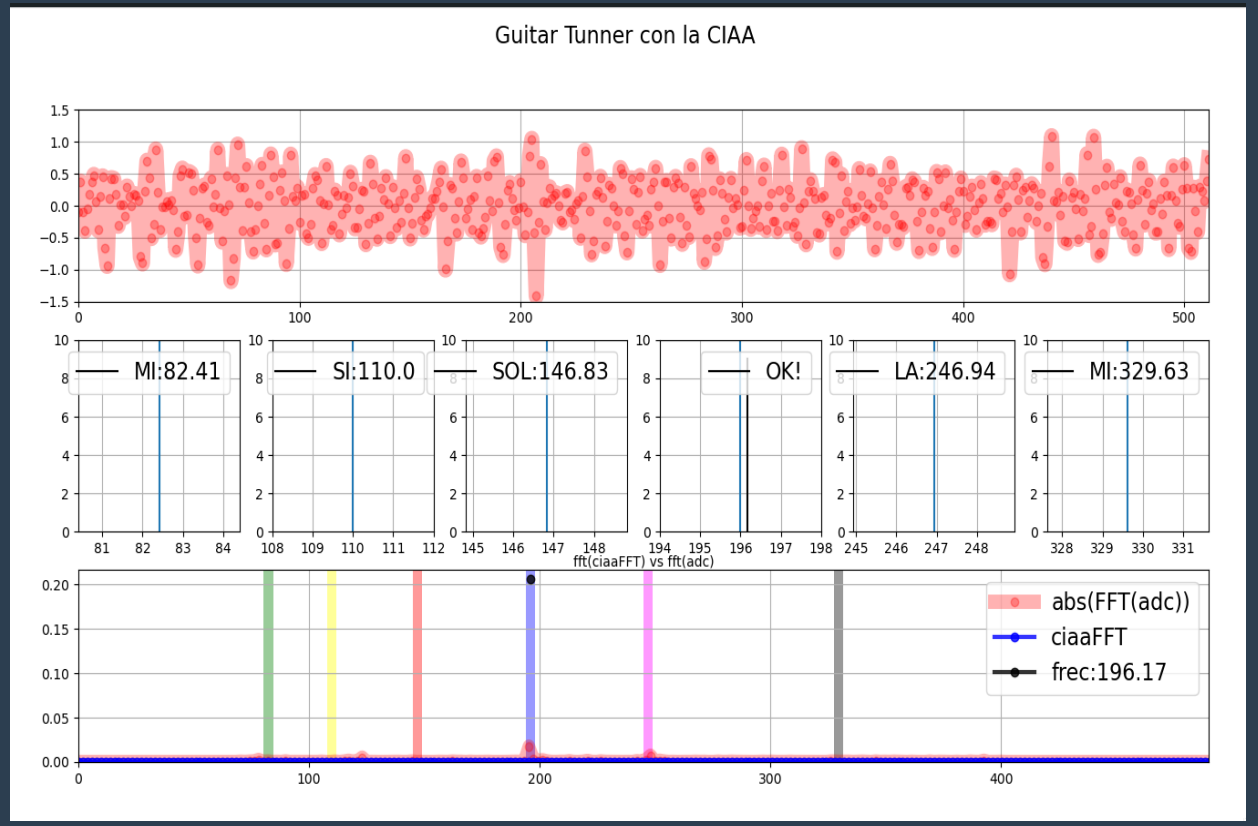


- Ver archivo: deconvolucion.py

Guitar Tunner con CIAA

Ejemplo de TP Final: Guitar Tunner con CIAA

- Se muestra en el video la presentación de un tp final como ejemplo o modelo de lo que se pretende.
- Duración de ~10mins
- Slides + presentación en vivo, o video
- Concentrar la idea en los temas que se vieron en la materia y relegar los temas muy específicos



- Ver carpetas:
  - clasese/ciaa_guitar_tunner
  - 7_clase/ciaa/psf3

Q&A TP Final