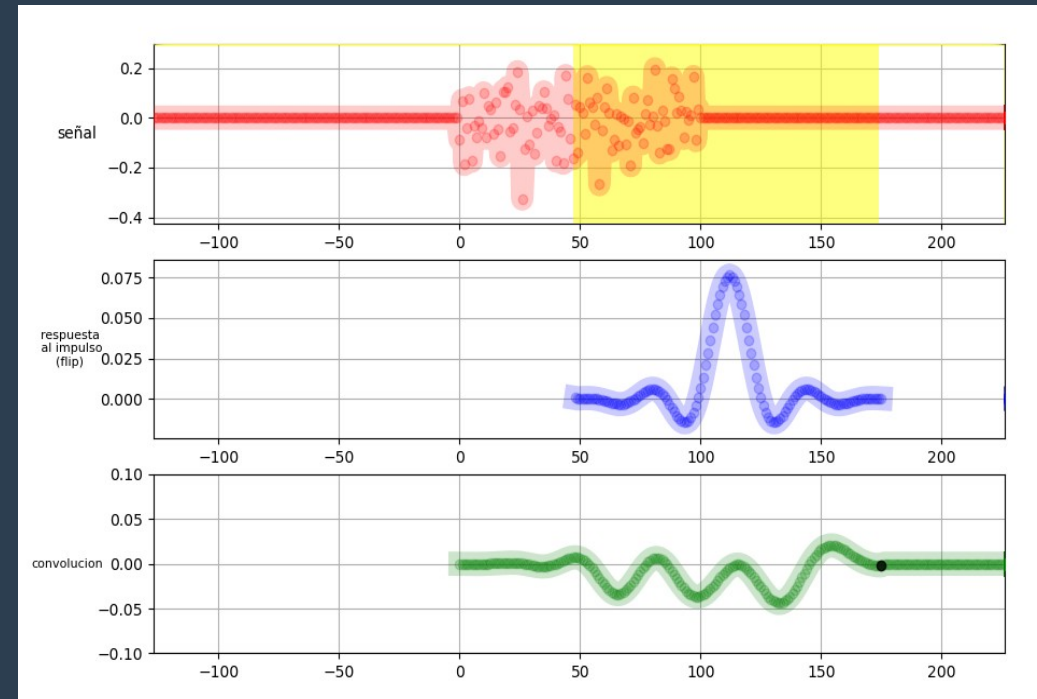


Procesamiento de señales. Fundamentos

Clase 5 – Respuesta al impulso | Convolución

- Respuesta al impulso
 - Repaso sistemas LTI
 - Descomposición en deltas
 - Repaso al impulso unitario
- Convolución
 - Input side
 - Output side
 - Multiplicación
 - Polinomios
- Teorema de la convolución
- Propiedades de la convolución
- Convolución con numpy
- Convolución con CMSIS-DSP



Impulso $\delta[n]$

Impulso en la panza de Homero

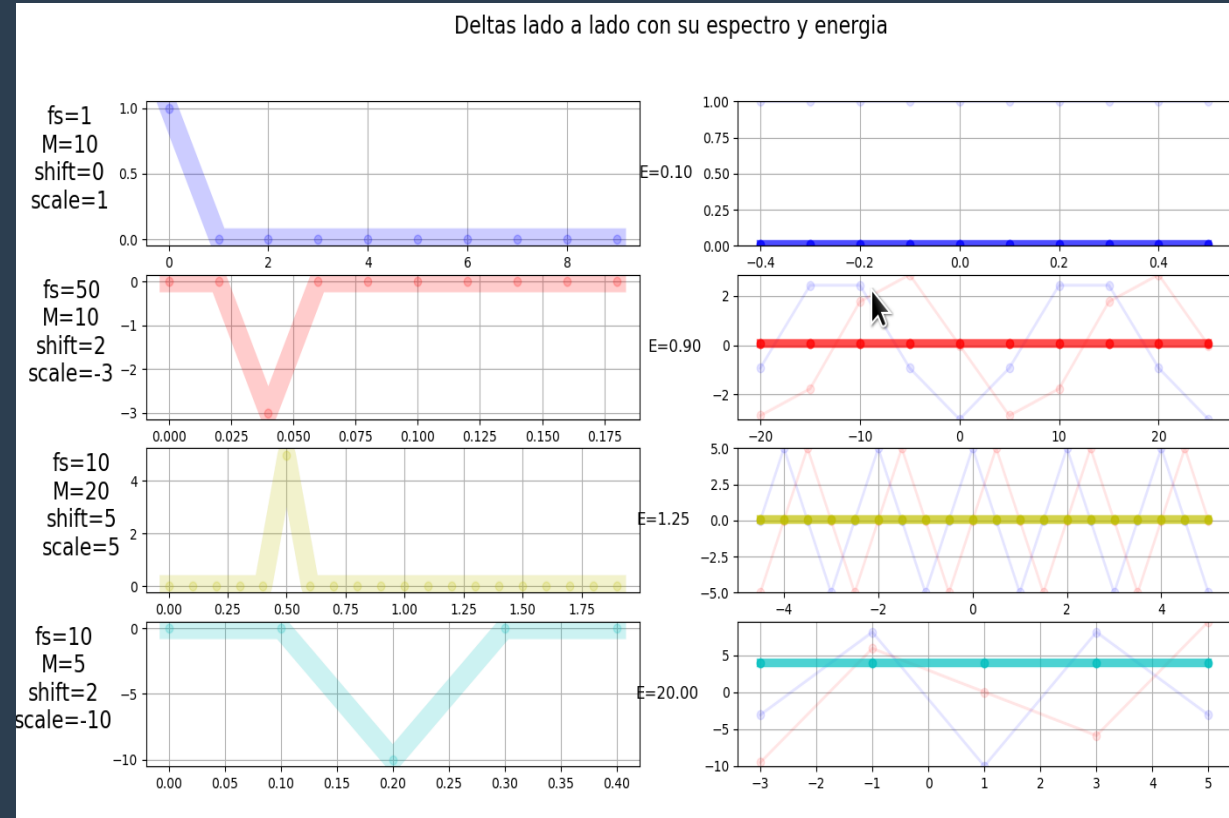
- El impulso es una señal de cualquier forma pero de una duración ordenes de magnitud menor a la duración de la salida del sistema como consecuencia de este impulso.
- En un sistema LTI, si el impulso se demora, es mas fuerte o mas débil la respuesta es **siempre** la misma demorada y escalada los mismos factores que el propio impulso.
- En el video se puede ver como el golpe del medico es de una duración mucho menor que la consecuencia, y la medición del tiempo de esta salida parece ser un indicador del estado de Homero.



● Ver video: [homero_panza.mp4](#)

La función delta $\delta[n]$ (impulso unitario)

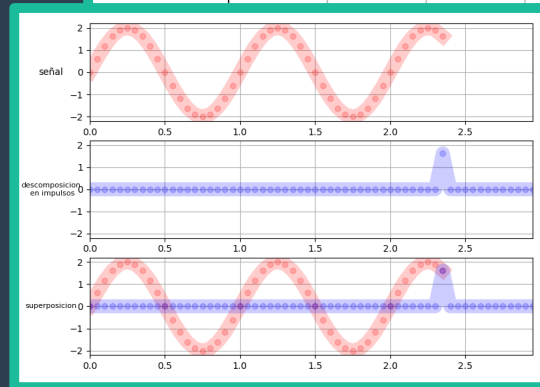
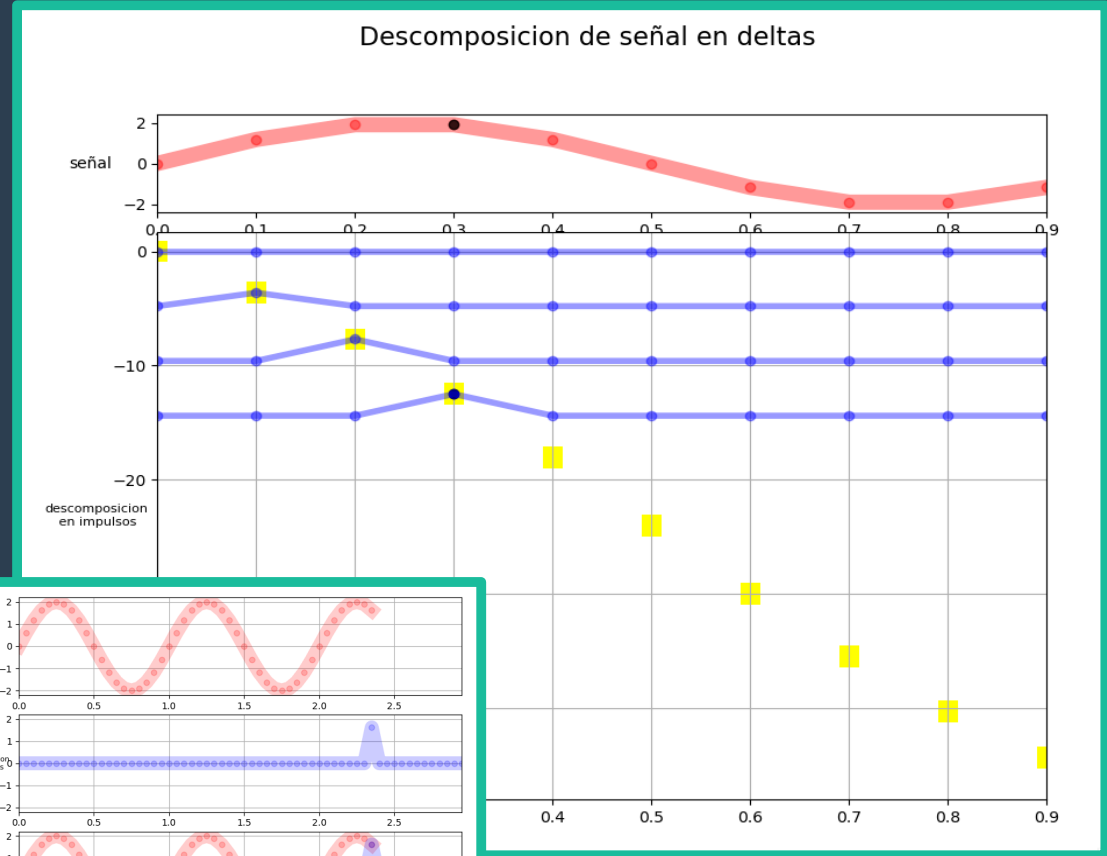
- $\delta[n]$
 - 1 para $n=0$
 - 0 para $n \neq 0$
- $A * \delta[n-x]$
 - A para $n=x$
 - 0 para $n \neq x$
- El impulso normalizado tiene área 1
- El contenido espectral de una delta es una constante. Tiene todas las frecuencias con igual amplitud (aunque con diferente fase)



• Ver código: [delta.py](#)

Descomposición en deltas

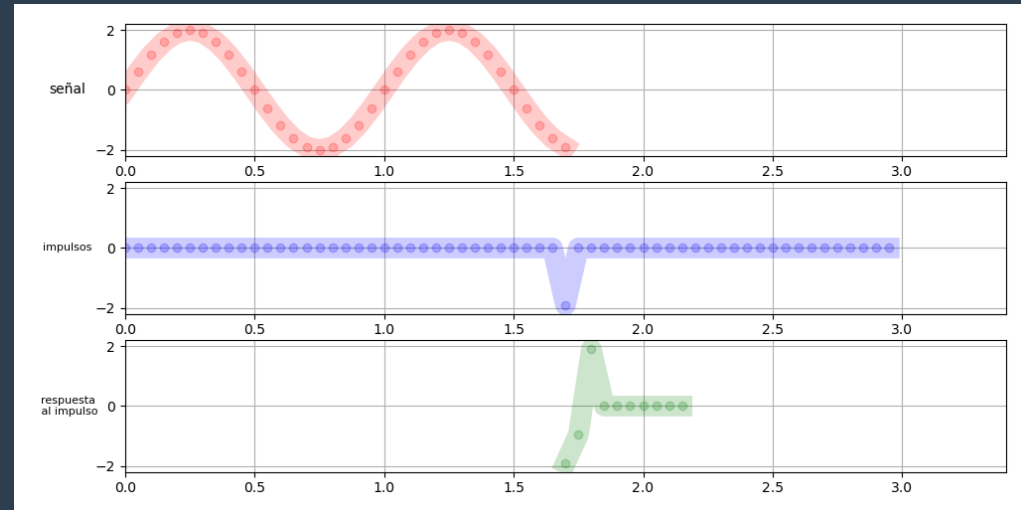
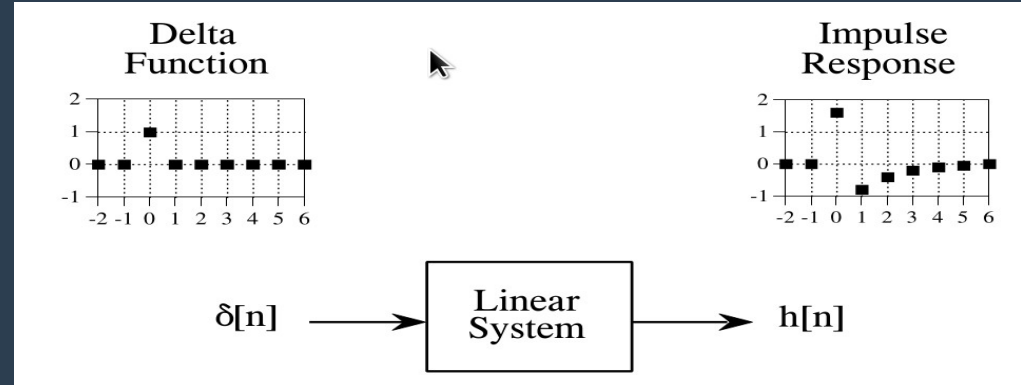
- Porque es importante la función delta?
- Porque se puede descomponer cualquier señal en sumatoria de deltas escaladas y shifteadas
- Si el sistema es LTI, podríamos calcular la respuesta a impulsos escalados y shifteados y superponer las respuestas de cada uno
- Así la respuesta de un sistema al impulso es **todo** lo que hace falta para conocer la respuesta del sistema a cualquier otra señal



- Ver códigos:
 - `descomposicion.py`
 - `recomposicion.py`

La respuesta al impulso unitario

- La respuesta al impulso unitario de un sistema es una nueva señal que permite caracterizar completamente al sistema
- Permite además calcular la respuesta del sistema a **cualquier** otra señal de cualquier forma y amplitud
- Es una característica intrínseca de cada sistema
- Un sistema solo puede tener una y solo una respuesta al impulso unitario
- Notar que a partir del último punto de la señal $x[N-1]$, la respuesta al impulso se extiende el largo de la misma menos 1
- Largo resultante = $N + \text{length}(h) - 1$

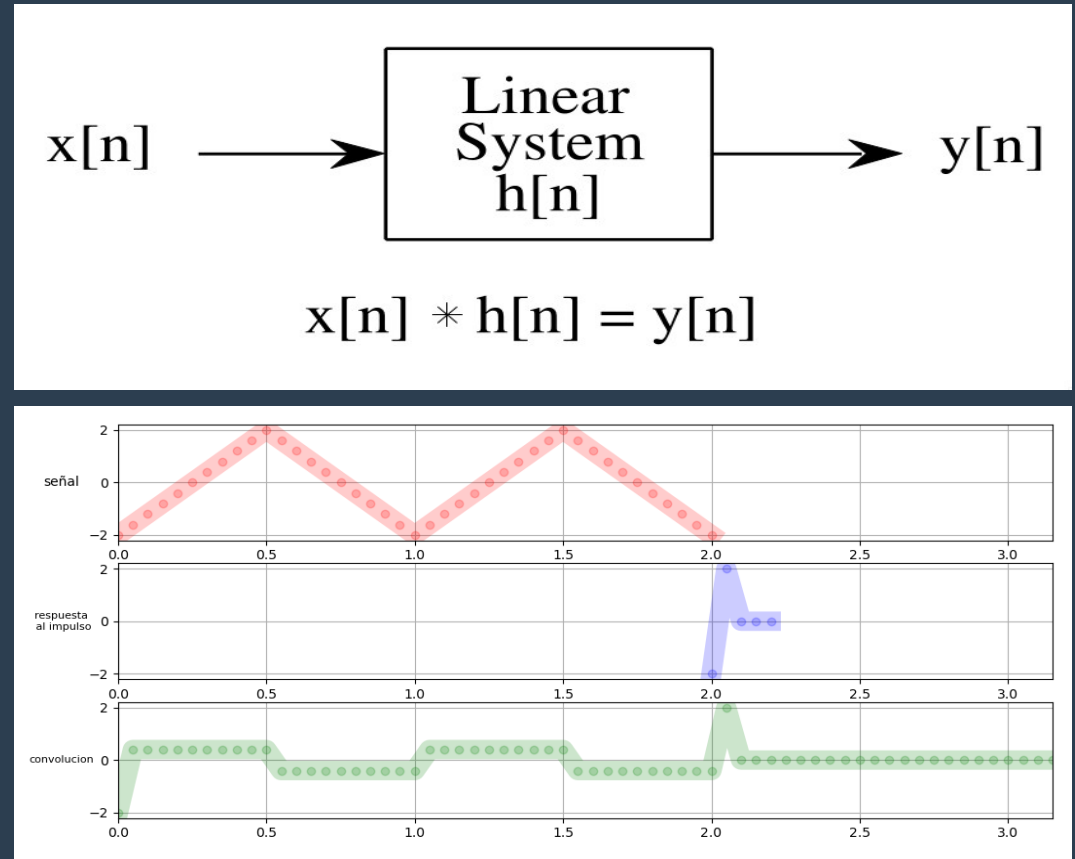


Ver código: [respuesta_impulso.py](#)

Convolución

Convolución

- Es una operación matemática, se simboliza con *
- Relaciona la señal de entrada, la de salida y la respuesta al impulso
- Como el sistema es LTI, puedo aplicar el principio de superposición y sumar la respuesta al impulso unitario de la descomposición en impulsos escalados y shifteados de la señal.
- Se obtiene la respuesta del sistema cuando la entrada es dicha señal.
- Dicha superposición **es la convolución entre la señal y la respuesta al impulso**

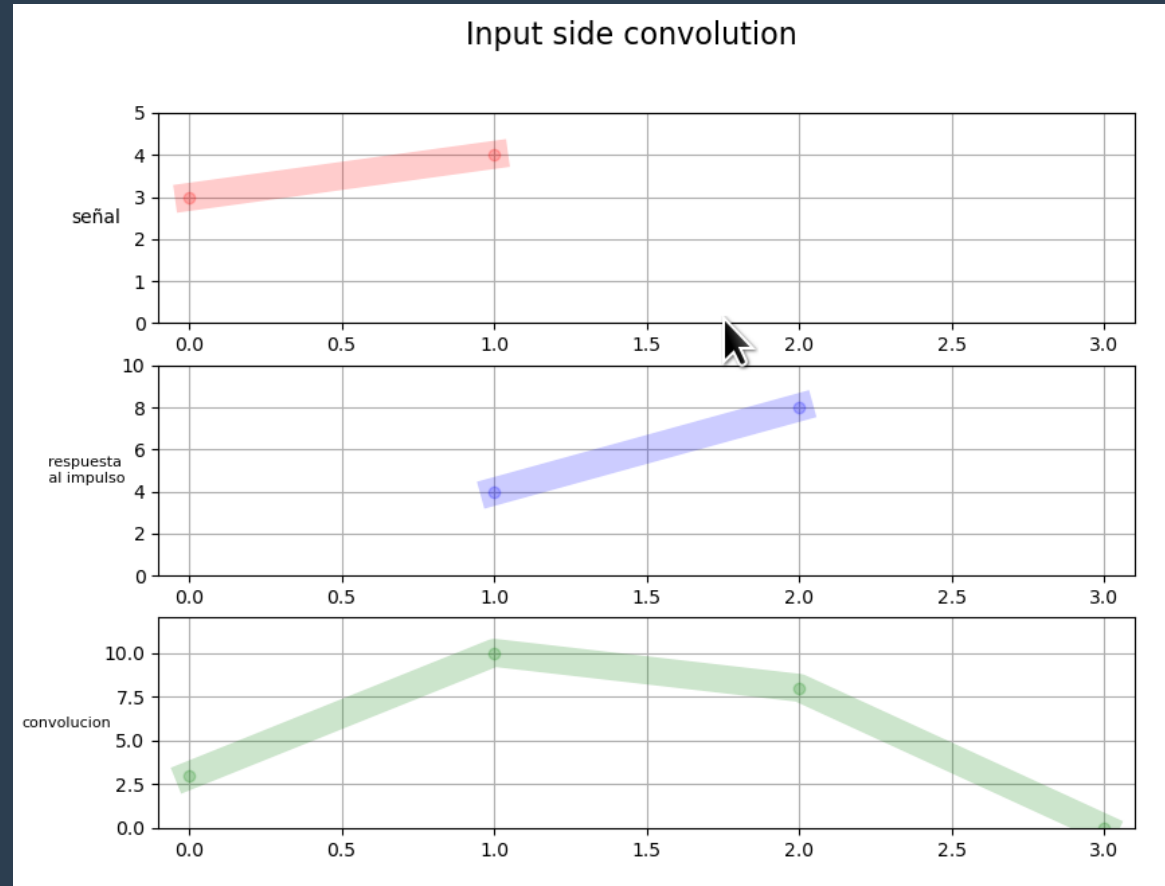


● Ver códigos: [convolucion.py](#)

Convolución – Input side

Convolución – Input side

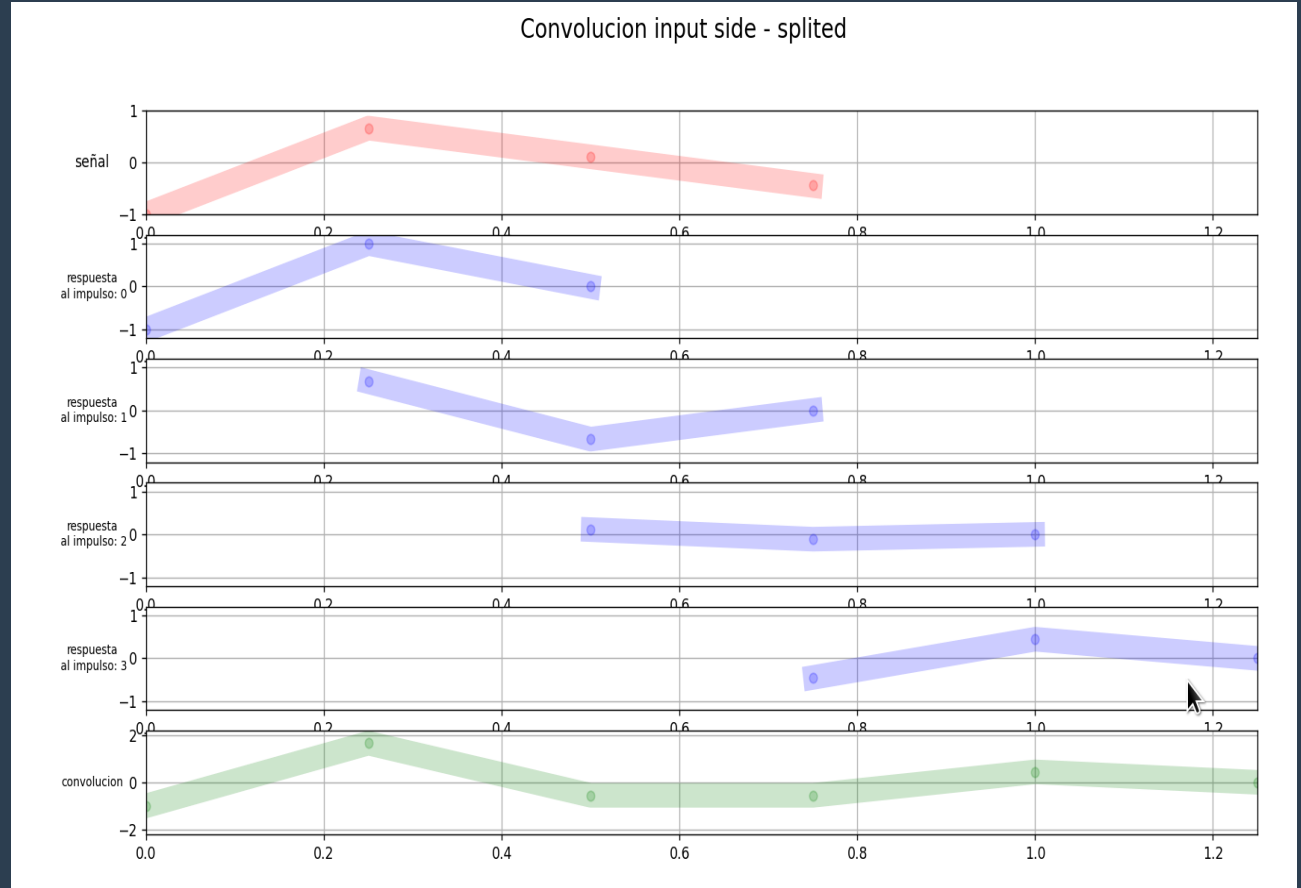
- Para cada punto de entrada $x[n]$, tomo la $h[n]$ y acumulo en $y[n:n+M-1]$
- Cada punto de la entrada $x[n]$ modifica varios puntos de la salida
- $y[n]=x[n]*h[n]$



● Ver códigos: `conv_input_side.py`

Convolución – Input side – Paso a paso

- Se puede ver en el ejemplo la respuesta al impulso $h[n]$ escalada según la señal y shifteada muestra a muestra
- La salida, en verde, será la suma algebraica de todas las muestras de $h[n]$ que solapan en el eje vertical
- $y[n] = x[n] * h[n]$
- No es posible definir matemáticamente la salida en función de la entrada con esta técnica

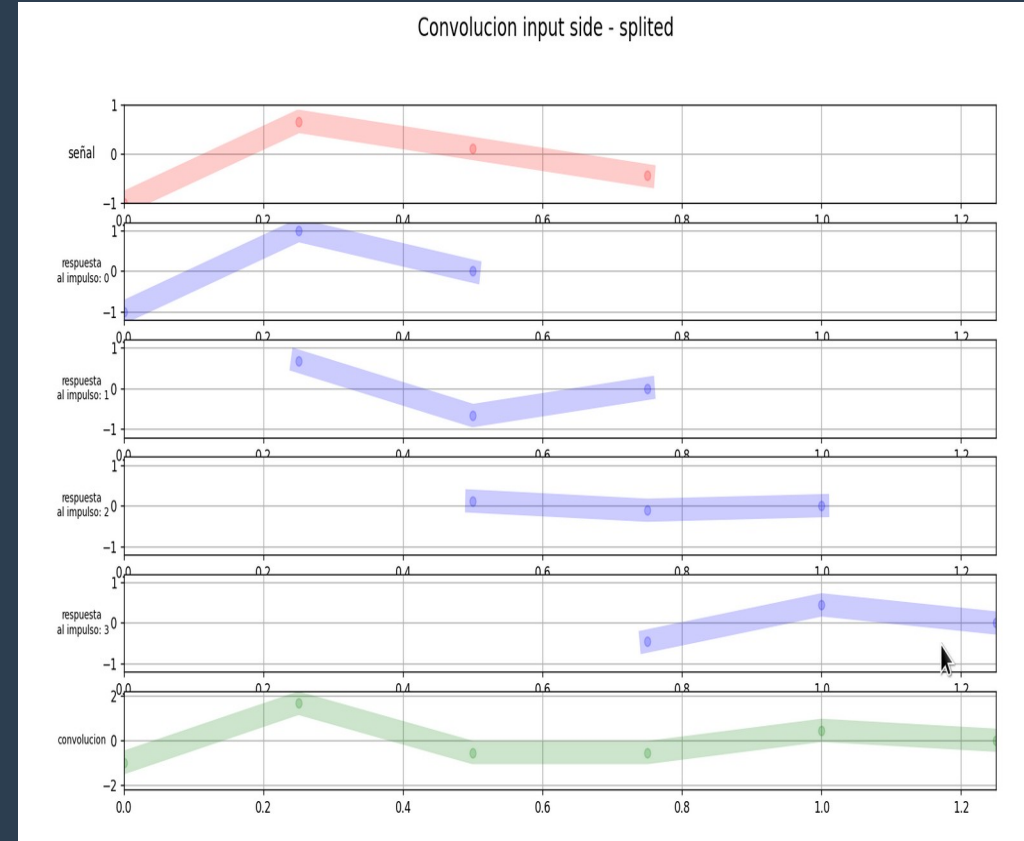


Ver códigos: [conv_input_side_splited.py](#)

Convolución – Output side

Convolución – Input side → output side

- $y[0]=x[0]*h[0]$
- $y[1]=x[0]*h[1] + x[1]*h[0]$
- $y[2]=x[0]*h[2] + x[1]*h[1] + x[2]*h[0]$
- $y[3]=x[1]*h[2] + x[2]*h[1] + x[3]*h[0]$
- $y[4]=x[2]*h[2] + x[3]*h[1]$
- $y[5]=x[3]*h[2]$
- Se evalúan cada punto de la salida en función de la entrada $x[n]$ y $h[n]$
- Notar que la asimetría en los índices de $h[n]$.
- Mientras se evalúa x en índices crecientes, se evalúa h en índices decrecientes
- Esto no permite calcular la convolución utilizando multiplicación punto a punto.
- Pero si se espeja h en el eje vertical, los índices se alinean y se podría utilizar la multiplicación punto a punto



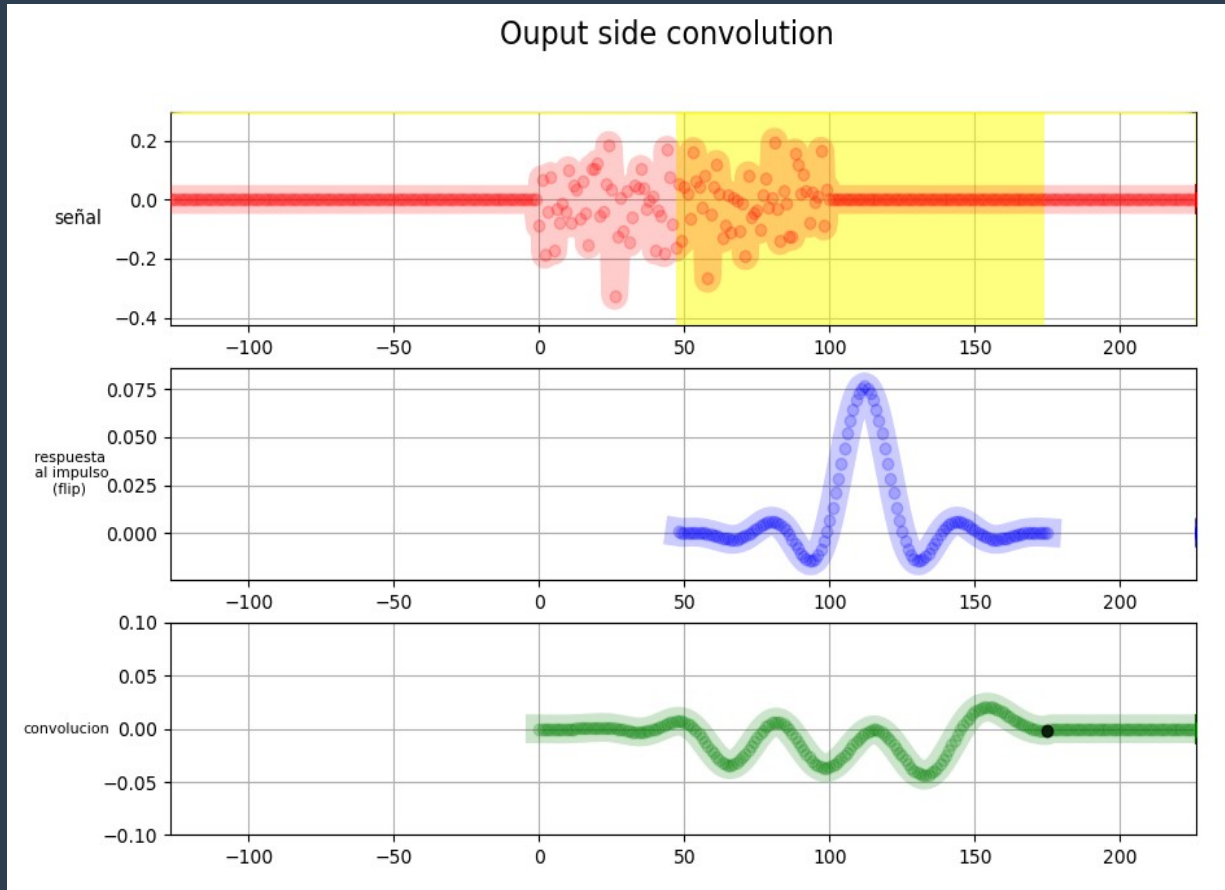
• Ver códigos: `conv_input_side_splited.py`

Convolución – Output side

- Para cada punto de la salida $y[n]$, tomo la sumatoria de la multiplicación de los puntos que afectan a dicho $y[n]$
- Cada punto de la entrada $x[n]$ modifica varios puntos de la salida desde n a $n+M-1$
- Como resultado la convolución formal se define y se calcula como la sumatoria de la multiplicación de la señal con la respuesta al impulso invertida en tiempo:

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

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



• Ver códigos: `conv_output_side.py`

Convolución – Como multiplicación

Convolución – Como multiplicación

- Recordar la técnica de multiplicación aprendida en primaria
- Considerar que
 - $h[n] = [1, 2]$
 - $x[n] = [3, 4]$
- Suponer que tenemos infinitos símbolos, no solo 10 para representar el resultado, con lo que no hace falta acarreo
- Entran 2 números de 2 cifras y sale 1 de 3. ($2 + 2 - 1 = 3$)
- $y[n] = [3, 10, 8]$

	1	2	
	3	4	
	<hr/>		
	4	8	
3	6	0	
	<hr/>		
3	10	8	

Convolución – Multiplicación polinomial

Convolución – Producto de polinomios

- Considerar cada elemento de $h[n]$ y $x[n]$ como coeficientes de un polinomio.
- Multiplicar los 2 polinomios respetando exponentes
- Entren 2 vectores de 2 elementos y sale 1 de 3
- Notar la complejidad de la operación, los productos cruzados.
- Se puede ver que el grado del algoritmo computacional seria de $O(2)$

$$\begin{aligned}(1x10^1 + 2x10^0) * (3x10^1 + 4x10^0) &= \\ (3x10^2 + 4x10^1 + 6x10^1 + 8x10^0) &= \\ (3x10^2 + 10x10^1 + 8x10^0) &\end{aligned}$$

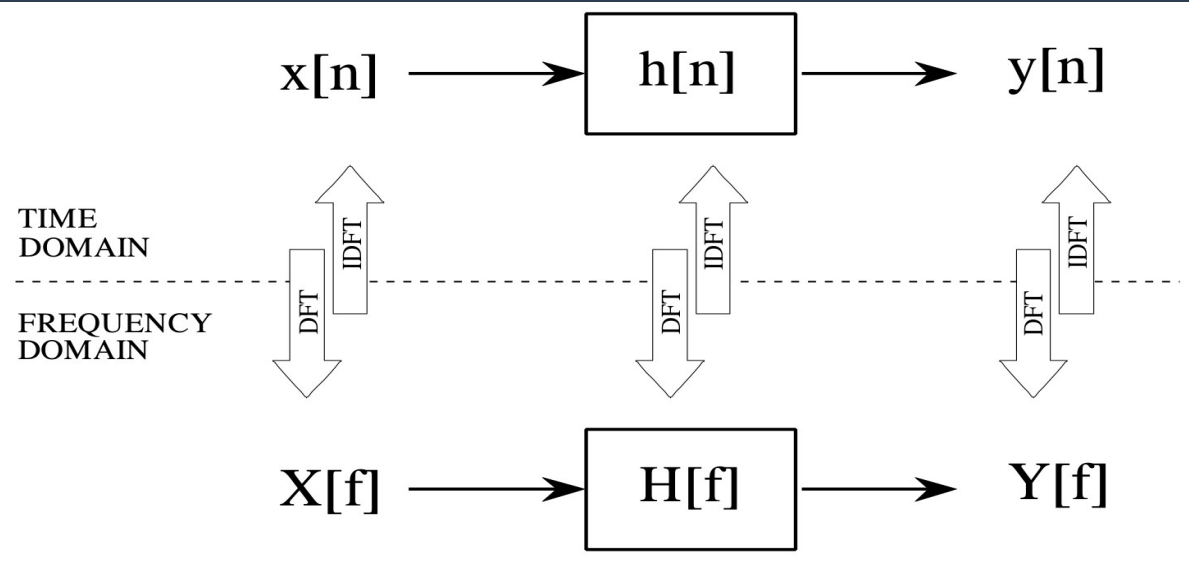
Teorema de convolución

Teorema de convolución

- La convolución de dos señales es equivalente a la antitransformada del producto de sus transformadas.
- La multiplicación de dos señales es equivalente a la antitransformada de la convolución de sus transformadas.
- Permite convertir la convolución en tiempo en una simple multiplicación en frecuencia.
- A partir de cierto número de N (~64) realizar la convolución de esta manera (FFT/IDFT) es computacionalmente más eficiente (en términos generales).
- Hay que prestar especial cuidado de que la cantidad de puntos de f y g sean menor que la cantidad de puntos utilizados para calcular la transformada.
- En caso contrario se rellena con ceros f y/o g para completar N .

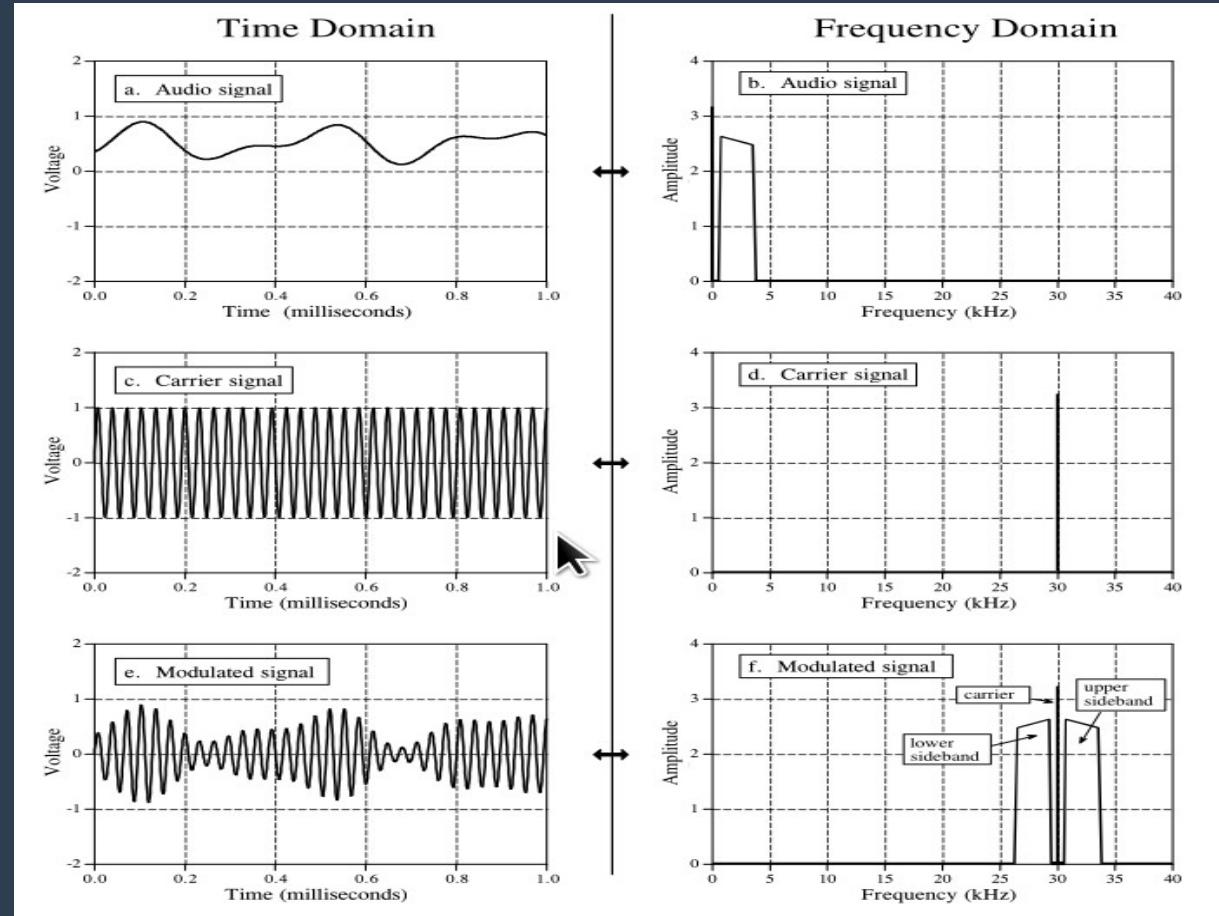
$$\{g_N * h\}[n] = \mathcal{F}^{-1}\{\mathcal{F}\{g\} \cdot \mathcal{F}\{h\}\}$$

$$\begin{aligned}\mathcal{F}\{f * g\} &= \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \\ \mathcal{F}\{\alpha \cdot g\} &= \mathcal{F}\{\alpha\} * \mathcal{F}\{g\}\end{aligned}$$



Teorema de convolución – Ejemplo AM

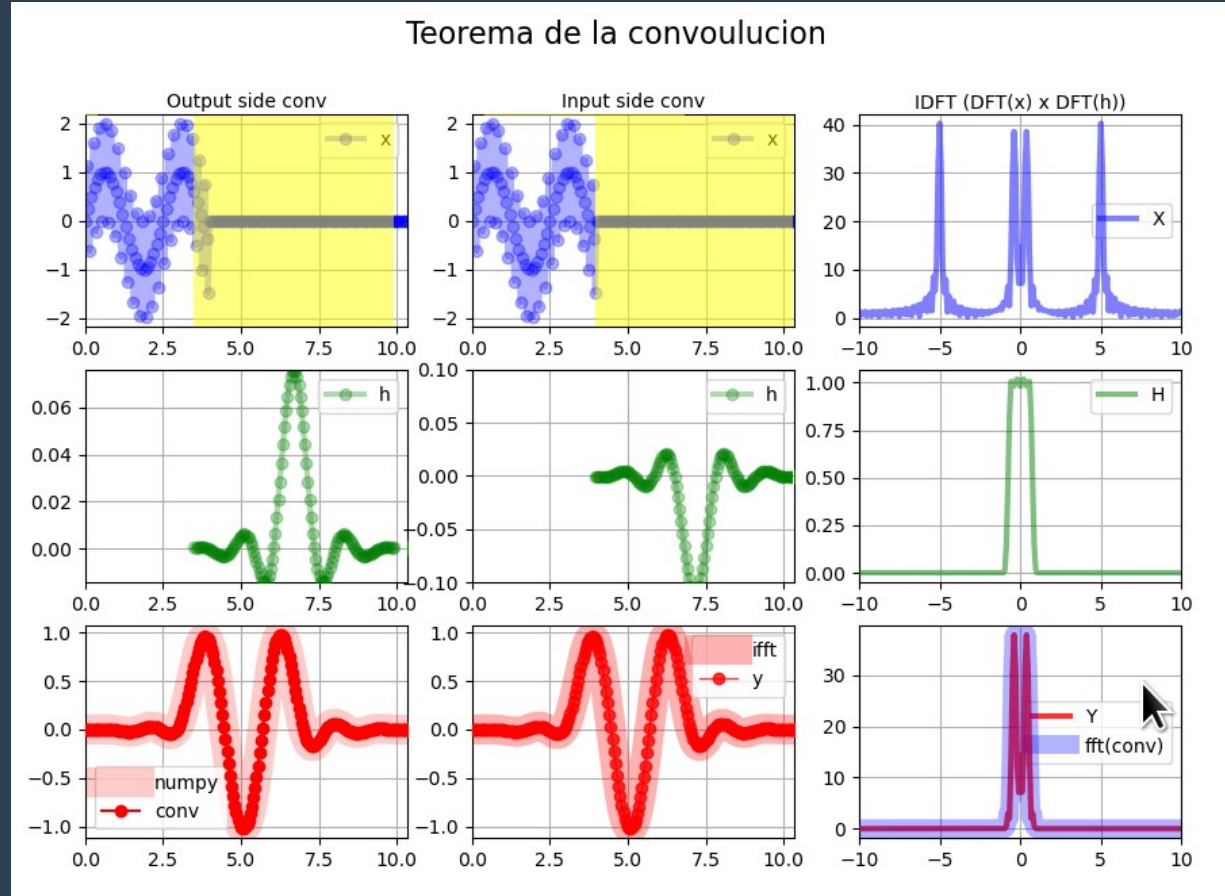
- Se puede ver en el caso de una modulación AM que implica multiplicar en t una portadora con la señal de interés
- Si se analiza en frecuencia se puede ver como la convolución en frecuencia revela las dos bandas, banda lateral inferior y superior (LSB y USB) a ambos lados de la portadora.



Convolución con Numpy

Convolución con numpy

- Modulo np.fft
- Función np.fft.dft
- Función np.fft.idft
- Función np.convolve
- En el ejemplo se puede validar el teorema de la convolución dado que se compara:
 - Output side conv
 - Input side con
 - IDFT(FFT() x FFT())
 - FFT(input side conv)

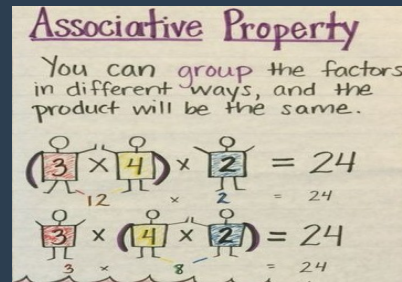
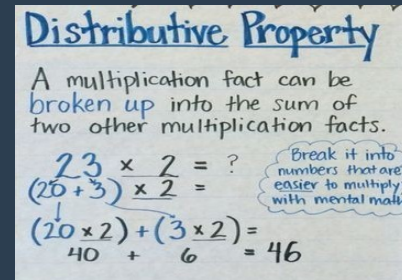
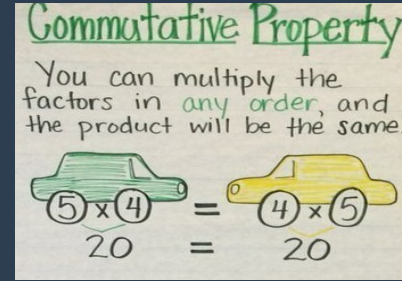


• Ver código: [teorema_convolucion.py](#)

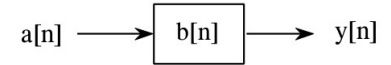
Convolución propiedades

Propiedades de la convolución

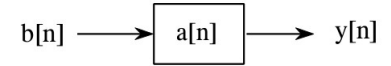
- Se destacan las propiedades matemáticas de la convolución
- Notar la similitud y correspondencia con las de la multiplicación extraídas de un manual de primaria
- Notar las consecuencias que tienen estas propiedades a la hora de interconectar sistemas y señales y como aprovecharlas a la hora de codificar los algoritmos



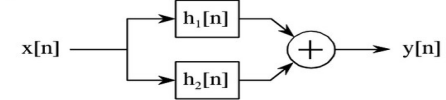
IF



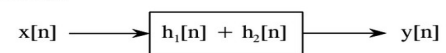
THEN



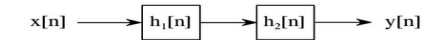
IF



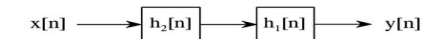
THEN



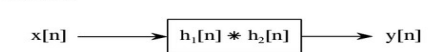
IF



THEN



ALSO

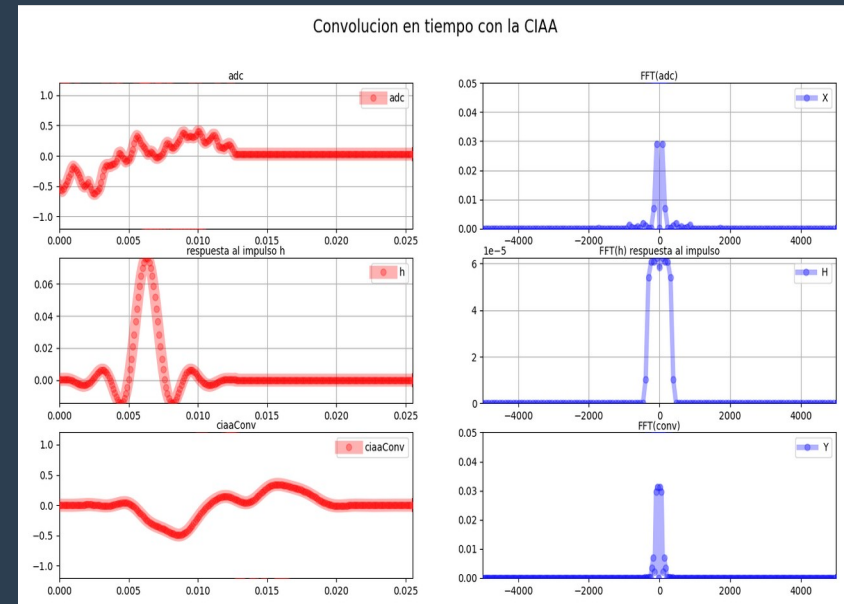


Convolución con la CMSIS-DSP

Convolución en la CIAA

- Calcula la convolución en tiempo usando `arm_conv_q15` o `arm_conv_fast_q15` (ver documentación)
- Se utiliza opcionalmente un conversor de `numpy` a `.h` para generar el `h` a partir de una lista de `python` que a su vez luego se obtendrá de la plantilla de un filtro.
- Recordar que la convolución de N con M genera $N+M-1$ datos. Reservar espacio en memoria para el arreglo resultante "y".
- Observar la zona útil de la convolución.

```
if ( ++sample==header.N ) {  
    gpioToggle ( LEDR );  
    sample = 0;  
  
    -----CONVOLUCION-----  
    arm_conv_q15      ( adc,header.N,h,h_LENGTH,y);  
    arm_conv_fast_q15 ( adc,header.N,h,h_LENGTH,y);  
  
    -----ENVIO DE TRAMA-----  
    header.id++;  
    uartWriteByteArray ( UART_USB ,(uint8_t*)&header ,sizeof  
    for (int i=0;i<(header.N+h_LENGTH-1);i++) {  
        uartWriteByteArray ( UART_USB ,(uint8_t*)(i<header.N  
        uartWriteByteArray ( UART_USB ,(uint8_t*)(i<h_LENGTH  
        uartWriteByteArray ( UART_USB ,(uint8_t*)(  
    }  
    adcRead(CH1); //why?? hay algun efecto minimo en el ler  
}
```



- Ver carpeta clase5/psf1