Sistemas y Circuitos
Lab 2: Sistemas
Curso académico 15/16

1. Objetivos de la práctica

En esta segunda sesión de laboratorio el estudiante trabajará sobre

- Implementar sistemas en tiempo discreto como funciones Matlab,
- Implementar la convolución discreta como una función Matlab,
- Simular un escenario de eco acústico empleando sistemas lineales e invariantes en el tiempo

2. Reglamentación y puntuación de las sesiones de laboratorios

- Los estudiantes deberán completar las actividades propuestas durante la sesión de laboratorio.
- Las actividades podrán realizarse por parejas.
- La evaluación de esta práctica se realizará al final de la sesión mediante una serie de preguntas que engloben los aspectos aprendidos en la práctica.
- Esta evaluación supondrá el 2,5 % del total de la nota de la asignatura.
- La asistencia a esta práctica no es obligatoria. Los alumnos que no asistan obtendrán una calificación de 0 en la misma.

3. Implementación de sistemas discretos descritos mediante ecuaciones en MatLab

Como vimos en la práctica anterior, Matlab trabaja con Vectores y Matrices para realizar operaciones matemáticas. Vimos también que podemos representar señales discretas mediante 2 vectores:

- **Vector n**: Es el dominio de la señal. El intervalo de interés sobre el que vamos a trabajar. Es una secuencia de números discretos, como por ejemplo n = [-7 -6 -5 ⋯ 0 1 2 ... 30]
- **Vector y**: El valor que toma la señal para cada uno de los valores de "n". Como por ejemplo y = [0 0.4 0.74 ⋯]

Los sistemas son dispositivos físicos o software que transforman una señal de entrada en una señal de salida. Un sistema discreto transforma una señal de entrada $x[n]$ en una señal de salida $y[n]$.

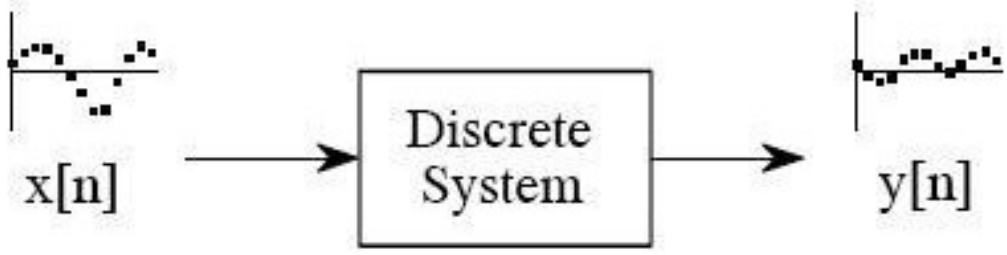

Figura 1: Sistema discreto.

El concepto de sistema es similar al de una función clásica. Una función $f(x)$ clásica toma una variable x y la transforma en otro valor $y = f(x)$. Un sistema $S()$ se puede ver como una función a la que le pasas una señal $x[n]$ y te devuelve otra señal $y[n] = S(x[n])$ aplicando diferentes operaciones matemátimas que pueden depender de los diferentes valores de $x[n]$ y de la variable independiente n.

Estas operaciones pueden ser cualquier cosa, desde logaritmos, retardos, eliminación, multiplicación por otras señales... no hay limites. Así pues un sistema viene definido por las operaciones que realiza sobre su señal de entrada $x[n]$.

En Matlab definiremos los sistemas como funciones de Matlab que reciben la señal de entrada $x[n]$ como argumento y devuelven la salida del sistema $y[n]$ ante dicha señal realizando operaciones sobre $x[n]$. Una posible definición de este sistema sería `[y, yn] = sistema1(x, xn)`. Como ejemplo absurdo de sistema a implementar tenemos el siguiente:

$$y[n] = 0.2x[n]^2 - 0.3x[n-1]^2 + \text{sen}(n/10\pi)x[n] - 0.4x[n-2]$$

Por suerte tenemos nuestra pequeña librería de funciones y podemos implementar el sistema de forma fácil. Podemos realizar los 4 sumandos de forma independiente y después sumar las señales con la función `suma()` de la práctica anterior.

```
function [ yo, no] = sistema1( x,xn )
    %% Primer sumando
    y1 = 0.2 * x.*x;
    n1 = xn;

    %% Segundo sumando
    y2 = - 0.3 * x.*x;
    n2 = xn;
    [y2, n2] = desplaza(y2, n2,-1);

    %% Tercer Sumando
    y3 = sin(xn/10*pi);
    n3 = xn;

    %% Cuarto Sumando
    y4 = - 0.4 * x;
    n4 = xn;
    [y4, n4] = desplaza(y4, n4,-2);

    %% Lo sumamos todo
    [yo, no] = suma(y1,n1,y2,n2);
    [yo, no] = suma(yo,no,y3,n3);
    [yo, no] = suma(yo,no,y4,n4);
end
```

En los códigos adjuntos a esta práctica dispone de dicha función y de un fichero llamado `main_sistema.m` que crea una señal $x[n]$ y obtiene la salida del sistema $y[n]$ ante dicha señal.
Apoyándose en el código anterior y en lo aprendido en la práctica 1, codifique una función `sistema2.m` que implemente el sistema descrito mediante la ecuación:

$$y[n] = 0.5x[4-n] + 0.7x[n-5] - 0.4\cos(2\pi x[2-n])$$

Para ello tenga en cuenta lo siguiente:

- La salida se compone sumando 3 señales intermedias
- Cada una de estas señales intermedias conlleva una operación sobre la variable independiente. Tenga esto en cuenta a la hora de determinar los instantes n en los que la salida está definida.

4. Nuestra amiga delta $\delta[n]$

Cualquier señal discreta puede ser expresada como un conjunto de deltas discretas $\delta[n]$ desplazadas y multiplicadas por una constante. En tiempo discreto una delta $\delta[n]$ es simplemente una señal que toma valor 1 cuando su argumento $[n]$ es igual a 0.

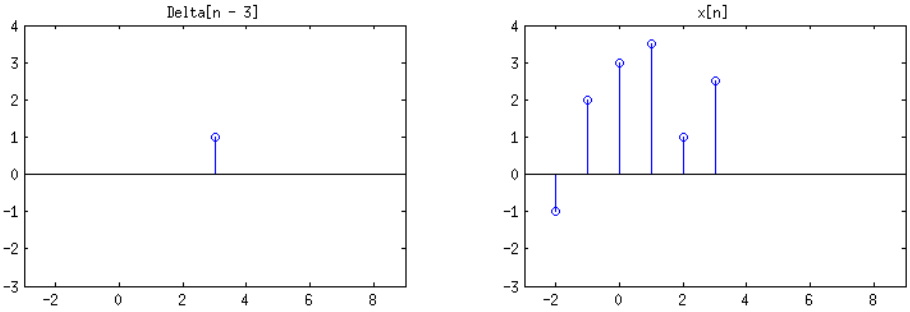

Figura 2: Delta desplazada y señal x[n].

Podemos expresar cualquier señal discreta como un conjunto de deltas desplazadas y multiplicadas por los valores de la señal x[n] en cada instante:

$$x[n] = \sum_{m=-\infty}^{\infty} c_m \delta[n-m]$$

Por ejemplo la señal x[n] de la imagen anterior tiene la ecuación:

$$x[n] = -\delta[n+2] + 2\delta[n+1] + 3\delta[n] + 3.5\delta[n-1] + \delta[n-2] + 2.5\delta[n-3]$$

Expresar una señal $x[n]$ como una combinación de deltas $\delta[n]$ nos da las herramientas matemáticas necesarias para poder calcular fácilmente la señal de salida $y[n]$ de un sistema ante cualquier señal de entrada.

5. Sistemas LTI

Los sistemas LTI (Linear Time Invariant) son aquellos sistemas cuyas operanciones sobre la señal de entrada $x[n]$ son lineales y además son siempre las mismas a lo largo del tiempo.

Matemáticamente estas propiedades se expresan de la siguiente manera:

1. **Linealidad**: Si $x_1[n] \rightarrow y_1[n]$ y $x_2[n] \rightarrow y_2[n]$. Entonces la salida de una señal $x_z[n]$ que sea combinación lineal de ambas, es decir: $x_z[n] = k_1 x_1[n] + k_2 x_2[n]$ Es: $x_z[n] \rightarrow y_z[n] = k_1 y_1[n] + k_2 y_2[n]$

2. **Invarianza en el tiempo**: Para que un sistema sea invariante en el tiempo, las operaciones que realiza no deben depender de la variable independiente $[n]$, matemáticamente esto equivale a: Si $x_1[n] \rightarrow y_1[n]$ entonces $x_1[n-k] \rightarrow y_1[n-k]$

Los sistemas LTI pueden ser completamente descritos por su respuesta al impulso $h[n]$. Esta respuesta al impulso es la señal de salida $y_\delta[n]$ que genera el sistema ante una delta $\delta[n]$ de entrada. Utilizando las propiedades de linealidad e invarianza en el tiempo, dada una señal de entrada al sistema $x[n]$ expresada como un sumatorio de deltas desplazadas, podemos expresar la salida del sistema como:

$$y[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m] = \sum_{m=-\infty}^{\infty} c_m h[n-m]$$

Así pues la salida del sistema LTI se obtiene convolucionando la señal de entrada $x[n]$ con la señal respuesta al impulso del sistema $h[n]$. La salida del sistema puede ser vista como el sumatorio de respuestas al impulso para cada delta de la señal $x[n]$.

En los códigos proporcionados, el fichero `main_convol.m` realiza la convolución de una señal $x[n]$ y un sistema $h[n]$ generando la Figura 3.

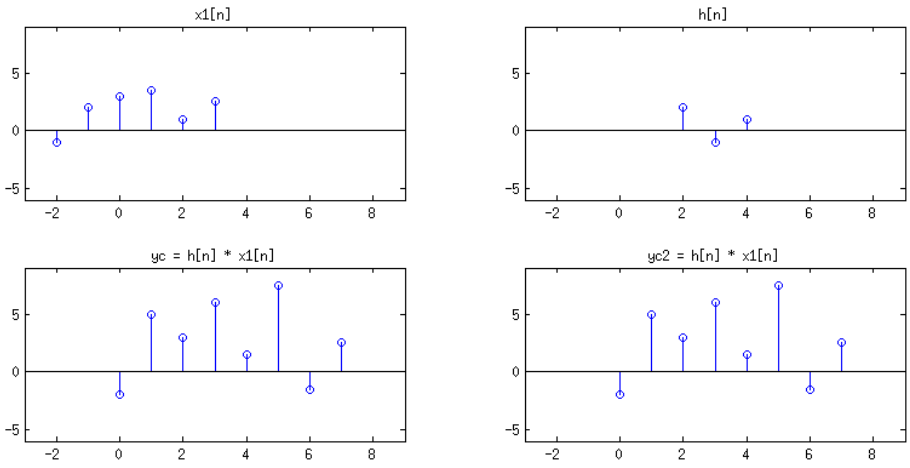

Figura 3: Convolución

Por tanto hemos visto que podemos calcular la salida del sistema $y[n]$ como la suma de respuestas al impulso $h[n]$ multiplizadas y desplazadas por las deltas de $x[n]$. Además, la operación de convolución posee la propiedad asociativa por lo que es lo mismo: $y[n] = x[n] * h[n]$ que $y[n] = h[n] * x[n]$. Es decir, también podemos calcular la salida $y[n]$ como la suma de la señal de entrada $x[n]$ multiplizada y desplazada por las deltas de $h[n]$.

La siguiente gráfica muestra las diferentes valores $x[n-m]h[m]$ de la convolución anterior.

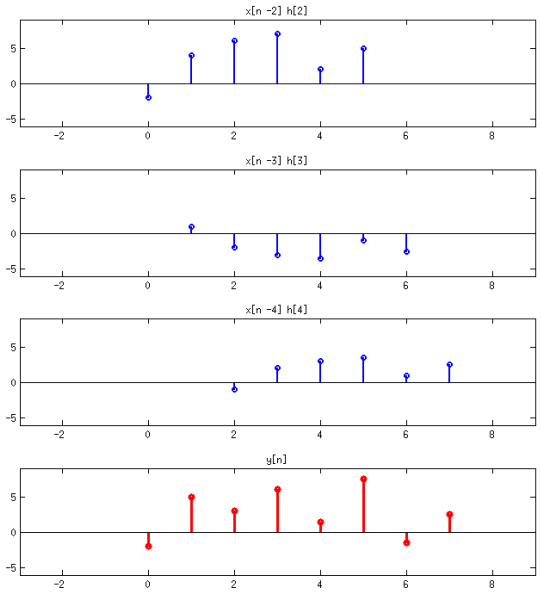

Figura 4: Señales intermedias de la convolución.

6. Implementación de la convolución en tiempo discreto para secuencias de duración finita

Como hemos visto, los sistemas LTI perfectamente definidos mediante la respuesta al impulso $h[n]$. Conocida $h[n]$, la salida para cualquier entrada puede calcularse mediante la operación de convolución:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k].$$

Matlab incorpora una implementación de la función convolución `conv()` que convoluciona dos vectores de valores. Esta función no tiene en cuenta los dominios de ambas, únicamente sus valores.

Por las propiedades de la convolución, la señal $y[n] = x[n] * h[n]$ tiene las siguientes características:

1. El número de muestras de $y[n]$ es igual al número de muestras de $x[n]$ más el número de muestras de $h[n]$ menos 1.
2. El dominio de $y[n]$ empieza en aquel instante $[n]$ donde empieza la señal $x[n]$ más el retardo mínimo que introduce el sistema $h[n]$.

Sabiendo esto, el siguiente código calcula la convolución de 2 señales, tanto en valor como en dominio.

```
x1 = [-1 2 3 3.5 1 2.5];   % Values of signal 1
x1n = [-2 -1 0 1 2 3]; % Domain of signal 1

h = [2 -1 1];
hn = [2 3 4];

%% Convolution using matlab functions
yc = conv(x1,h);
nc = 1:length(yc);
nc = nc + hn(1) + x1n(1) - 1;
```

En vez de utilizar la función de convolución proporcionada por matlab vamos a crear nuestra própia función de convolución utilizando la lógica de la Figura 4. Para ello la función deberá ir generando réplicas de la señal $x[n-m]$ desplazadas y multiplicadas por $h[m]$. Después simplemente las sumamos con nuestra función `suma()` que ya se encargará de los dominios. El código de esta función es:

```
function [ yo, no] = Myconv( x,xn, h, hn )
    yo = h(1).*x;  % The output of the system to the first value of x[n]
    no = xn + hn(1); % The domain is retarded as much as the system hn
    if (length(h) > 1)  % If the signal has more than one value
        for i = 2:length(h)
            xaux = x.*h(i);
            naux = xn + hn(1);
            [xaux, naux] = desplaza(xaux, naux, -i +1);

            [yo, no] = suma(yo, no, xaux, naux);
        end
    end
end
```

Esta función va generando señales $x[n-m]h[m]$ y las va sumando. La primera de ellas la calcula fuera del bucle para ir después sumándole el resto dentro del bucle. Esta función es utilizada en `main_convol.m` y se puede ver que genera el mismo resultado que la función de Matlab.

7. Miniproyecto 1: Simulación de eco acústico

Con lo aprendido en esta sesión podemos simular el eco acústico producido por la reflexión de un sonido. El eco puede verse como una réplica retardada y atenuada del sonido original. La señal que el emisor del sonido escucha ante la presencia de eco es:

- El propio sonido que produce, lo llamaremos $x[n]$
- El eco que recibe, lo llamaremos e[n]. Este eco no es más que una versión retardada y atenuada del sonido original por lo que $e[n] = Kx[n-m]$

Este miniproyecto consiste en simular el eco producido por un edificio que se encuentra 250 metros del emisor. Para ello se modelará el edificio como un sistema LTI que ante una señal $x[n]$ devuelve lo que escucharía el emisor. La respuesta al impulso del sistema tiene 2 deltas, una para el sonido original y otra para el eco.

El retardo y atenuación que sufre el eco se puede calcular utilizando los siguientes parámetros:

- La velocidad del sonido es de 340 m/s.
- La amplitud del sonido se reduce a la mitad cada 250 metros.
- Los audios están grabados a una frecuencia de muestreo de $f_s = 11025\ muestras/sec$

El alumno deberá implemetar una función llamada edificio.m que ante una señal de entrada $x[n]$ devuelva la señal que escucha el emisor en presencia del eco producido por el edificio. Respecto al eje de tiempos, asuma que el primer instante de la señal $x[n]$ es $n = 0$.

8. Miniproyecto 2: Función de convolución mejorada

En Matlab los bucles `for` consumen mucho tiempo, cuantas menos iteraciones tenga el bucle, mejor. Nuestra función de convolución va creando réplicas desplazadas y multiplicadas de $x[n]$, es decir, va generando las señales $x[n-m]h[m]$ y sumándolas.

Gracias a la propiedad asociativa de la convolución podríamos realizar la convolución creando réplicas desplazadas y multiplicadas de $h[n]$, es decir, señales $h[n-m]x[m]$ e ir sumándolas.

El objetivo de este miniproyecto es que la función `Myconv.m` mire primero cuál de los 2 métodos requiere menos iteraciones del bucle y después calcula la convolución de una forma u otra de tal forma que se minimicen las iteraciones.