Sistemas y Circuitos
Lab 1: Señales
Curso académico 15/16

1. Objetivos de la práctica

En esta primera sesión de laboratorio el estudiante aprenderá a:

- Manejar los aspectos básicos de Matlab
- Representar secuencias finitas mediante vectores de Matlab
- Realizar operaciones sencillas con señales discretas empleando Matlab
- Calcular medias, potencias y energías parciales de secuencias.

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. Manejo de señales dicretas en MatLab

Matlab es una potente herramienta software matemática orientada al manejo de matrices. Las mayores ventajas de Matlab para el procesamiento de señales y datos en general son la facilidad de su lenguaje de programación (de bastante alto nivel), la cantidad de bibliotecas software (*toolboxes*) que implementan distintas técnicas y tecnologías de análisis y la facilidad de representación gráfica de los resultados de estos análsis.

El primer objetivo de esta práctica es aprender a **representar señales en Matlab**. Matlab únicamente entiende de vectores y matrices, siendo:

- **Vector**: Secuencia ordenada de números de longitud N. V = [3 8 -32 9 0 -4]
- **Matriz**: Tabla ordenada de números. Se puede ver como un conjunto de vectores de la misma longitud N.

Matlab en realidad está orientado a matrices. Un vector en Matlab es una matriz de dimesiones $N \times 1$ (vector columna) o $1 \times N$ (vector fila). Hay que tener un poco de cuidado con esto, si vamos a sumar 2 vectores ambos tienen que ser del mismo tipo. A continuación veremos cómo podemos **representar una señal unidimensional $y = f(t)$ en Matlab**. Las señales del mundo real como pueden ser la temperatura, presión, voltaje... son señales continuas que además pueden tomar valores en cualquier instante de tiempo, esto supone una serie de problemas que vemos a continuación.

1) Un ordenador no puede almacenar una señal contínua dado que esta tiene infinitos valores en cualquier intervalo de tiempo. Por ejemplo, si medimos la temperatura de una habitación de 12:00 a 13:00 deberíamos almacenar la temperatura de cada minuto, de cada segundo, de cada milisegundo, de cada millonésima de segundo y así hasta el infinitésimo. Como el ordenador tiene

una memoria finita para almecenar estos valores, lo se hace es fijar un intervalo de tiempo ts, por ejemplo segundos, tras el cual se muestrea la señal. En nuestro ejemplo, tendríamos 3600 muestras de la señal.

2) Un ordenador no puede almacenar infinitos instantes de la señal. Aunque sólo muestreásemos la temperatura cada segundo, si la muestreamos desde el principio de los tiempos hasta el final del mismo, tendríamos infinitas muestras que tampoco podríamos almacenar. Así pues en Matlab representamos señales reales muestreándolas en un intervalo finito.

La siguiente gráfica muestra la **discretización** (muestreo) de dos periodos de una señal seno. La función seno tiene un dominio desde menos infinito a infinito pero nosotros sólo muestrearemos 2 ciclos, de 0 a 4π. El intervalo de muestreo es tal que tomamos 30 muestras de dicha señal. Así pues estamos estamos obteniendo la señal discreta y[n] a partir de la señal continua y(t). Tener en cuenta que t es una variable continua que contiene los segundos donde toma valores nuestro seno mientras que n es discreta y contiene el instante discreto de la muestra. Para calcular en qué instante fue de verdad tomada una muestra basta con multiplicar n por el periodo de muestreo ts.

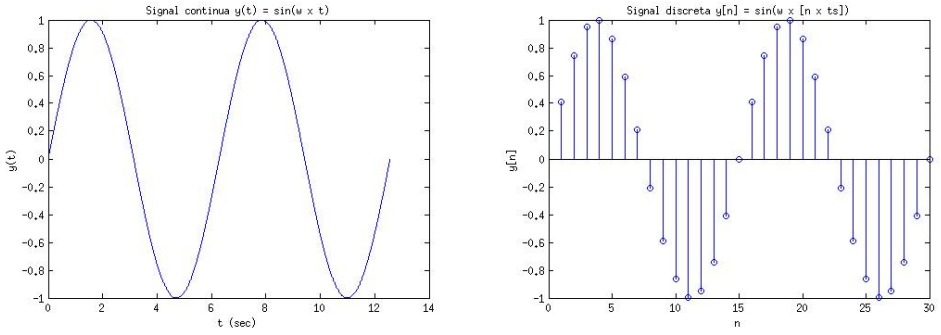

Figura 1: Discretización (muestreo).

Así pues, podemos representar una señal unidimensional en Matlab mediante una secuencia finita de valores tomados a intervalos regulares. La señal será representada 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, en el ejemplo anterior n = [0 1 2 ... 30]
- **Vector y**: El valor que toma la señal para cada uno de los valores de "n". En el ejemplo anterior y = [0 0.4 0.74 $\cdots$]

Las señales con las que trabajamos en Matlab por tanto serán señales discretas de longitud finita. Estas señales discretas como hemos visto pueden representar señales continuas que han sido muestreadas en un intervalo finito de tiempo, también pueden ser simplemente señales discretas originales. Si una señal no tiene definido su valor en un instante dado, supondremos que es 0.
El siguiente trozo de código crea 2 señales cortas definidas en distintos intervalos de tiempo

```
y1 = [-2 1 5]; % Values of signal 1
n1 = [3 4 5];  % Domain of signal 1

y2 = [2 3 3 1];   % Values of signal 2
n2 = [8 9 10 11]; % Domain of signal 2
```

Usando Matlab podemos realizar cualquier operación con dichas señales pero tendremos que ser conscientes de que para Matlab no son más que vectores, tendremos que ser nosotros los que le demos la lógica a las operaciones.

3.1. Suma de señales discretas

Si queremos **sumar un par de señales y1 e y2** como las del ejemplo anterior, no podemos simplemente escribir ys = y1 + y2. Primero porque ambos vectores no tienen la misma longitud

y Matlab solo permite sumar vectores de igual longitud, hace la suma de cada par de números de los vectores. Aunque tuvieran el mismo tamaño, sumarlas de esta manera también estaría mal ya que estaríamos sumando unos valores con otros sin tener en cuenta el dominio. Así pues nos tenemos que hacer una **función que nos sume dos señales**, teniendo en cuenta los dominios de las mismas. La función deberá:

- Alinear los dominios de n1 y n2 creando un dominio conjunto que los aúne.
- Añadir ceros a la izquierda y derecha de y1 y y2 para después sumar las señales. Entonces ya podremos realizar el deseado ys = y1 + y2.

A continuación se muestra una posible implementación de esta función.

```
function [ yo, no ] = suma(y1,n1,y2,n2)
    n_min = min(min(n1),min(n2));   % Get the limits of the domain
    n_max = max(max(n1),max(n2));
    no = [n_min:n_max];             % Create the domain

    % Add the proper number of 0s to the left and right of the first signal
    n0_iz = min(n1) - min(no);
    n0_de = max(no) - max(n1);
    y1 = [zeros(1,n0_iz), y1, zeros(1,n0_de)];

    % Add the proper number of 0s to the left and right of the second signal
    n0_iz = min(n2) - min(no);
    n0_de = max(no) - max(n2);
    y2 = [zeros(1,n0_iz), y2, zeros(1,n0_de)];

    yo = y1 + y2; % Add signal

end
```

En Matlab las funciones se guardan en ficheros .m con el mismo nombre que la función. Para utilizarlos simplemente escribimos **salidas = nombre_funcion(entradas)**. Tener en cuenta que para utilizar una función que hayamos creado nosotros, su fichero tiene que estar en la misma carpeta donde está el fichero desde el que la llamamos.

A continuación se muestra un código que utiliza la función suma que hemos creado para sumar dichas las señales y nos las dibuja mediante la función stem de Matlab. El resto de funciones xlim, ylim y title son para darle un poco de formato a las gráficas, definiendo los límites de los eje xy a mostrar y el título de la gráfica.

```
[ys, ns] = suma(y1,n1,y2,n2);  % Sum of signals

figure();  % Create the main window
subplot(2,2,1); %
stem(n1,y1);
xlim([-3 15])
ylim([-3 6])
title('1st signal')

subplot(2,2,2)
stem(n2,y2);
xlim([-3 15])
ylim([-3 6])
title('2nd signal')

subplot(2,2,3)
stem(ns,xs);
xlim([-3 15])
ylim([-3 6])
title('Suma')
```

La siguiente gráfica muestra ambas señales y su suma. Además muestra dicha suma desplazada 4 posiciones en el tiempo,es decir ys[n+4], parte que deberá realizar el alumno más adelante.

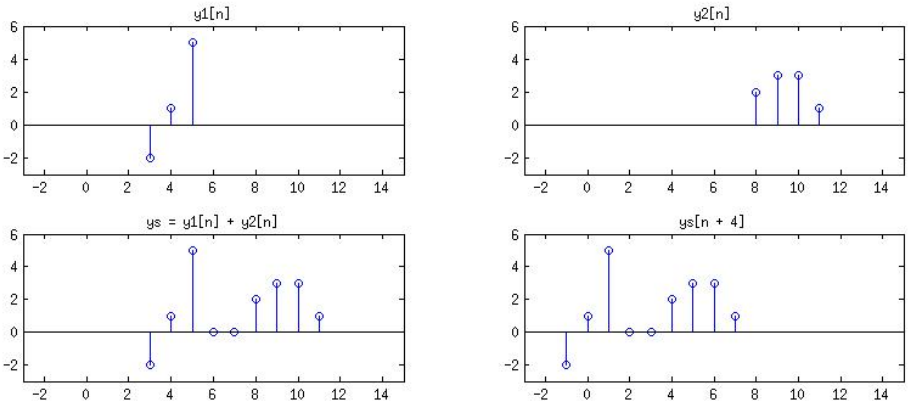

Figura 2: Representación de las señales y su suma.

3.2. Operando con senos

En el ejemplo anterior definimos 2 señales discretas y después operamos con ellas. En este ejemplo muestrearemos 2 senos con diferentes frecuencias y operaremos con ellos. En este caso no nos importa los instantes de tiempo t reales en los que muestreamos los senos, nos da igual el periodo de muestreo ts, sólo queremos tener en N = 10000 muestras:

- **Señal 1**: 4 ciclos muestreados de una señal seno.
- **Señal 2**: 2 ciclos muestreados de otra señal seno.

Obviamente la primera señal tiene el doble de frecuencia w que la segunda pero repito que el valor real de los instantes de muestreo no nos interesa, sólo queremos tener en N = 10000 muestras 4 ciclos de una señal y 2 de la otra.

Cuando muestreamos la señal, el vector n como siempre, será un conjunto de números enteros, el valor temporal real de la señal lo podemos obtener multiplicando n por el valor del intervalo de muestreo ts.
La señal seno es una señal continua $y(t) = \sin(w \cdot t)$ que toma valores desde $-\infty$ hasta ∞, para operar por ella por tanto habrá que muestrearla en un periodo finito. Además es una señal periódica de periodo 2π por lo que cada vez que su fase recorra 2π habrá pasado un ciclo, en otras palabras $sin(w \cdot t) = sin(w \cdot t + 2\pi)$. Las funciones seno muestradas tendrán la forma.

$$y[n] = \begin{cases} \sin(w \cdot n) & 1 \leq n \leq 10000 \\ 0 & \text{resto} \end{cases}$$

Nuestro objetivo será coger 4 ciclos de un seno con N = 10000 muestras. Para ello el argumento del seno debe haber recorrido 2π cuatro veces por lo que tenemos que $w = 4 \cdot 2\pi/N$. De esta forma el argumento del seno $w \cdot n$ va entre 0 y 8π. El cálculo de la frecuencia w de la segunda señal se calcula de forma homóloga.

El siguiente código crea ambas señales seno muestreadas. Notar que dado que asumimos que el dominio de ambas señales es el mismo, ya no hace falta tenerlo en cuenta para operar con ellas. Esta vez, para dibujar las señales utilizamos la función plot() de Matlab que nos las dibuja como si fueran señales contínuas uniendo los puntos.

```
N = 10000;   % Number of samples
P1 = 4;      % Number of periods s1
P2 = 2;      % Number of periods s2

n = 1:N;   % Dominio sobre el que muestreamos
```

```
s1 = sin((2*pi*P1/N)*n);
s2 = sin((2*pi*P2/N)*n);

y1 = s1 + s2;   % Sum of signals
y2 = s1 .* s2;  % Product

figure();  % Create the main window
subplot(2,2,1); %
plot(n,s1);
title('1st signal')
subplot(2,2,2)
plot(n,s2);
title('2nd signal')
subplot(2,2,3)
plot(n,y1);
title('Suma')
subplot(2,2,4)
plot(n,y2);
title('Producto')
```

Se puede observar a partir del código que sin(x) es la función de Matlab que te calcula el seno del valor x que le pases como argumento. Si le pasas un vector, como estamos haciendo nosotros, te devolverá otro vector con el seno de todos valores de x.También podemos notar que para multiplicar dos vectores valor a valor se utiliza el operador .*

La siguiente gráfica muestra ambas señales seno, su suma y su producto.

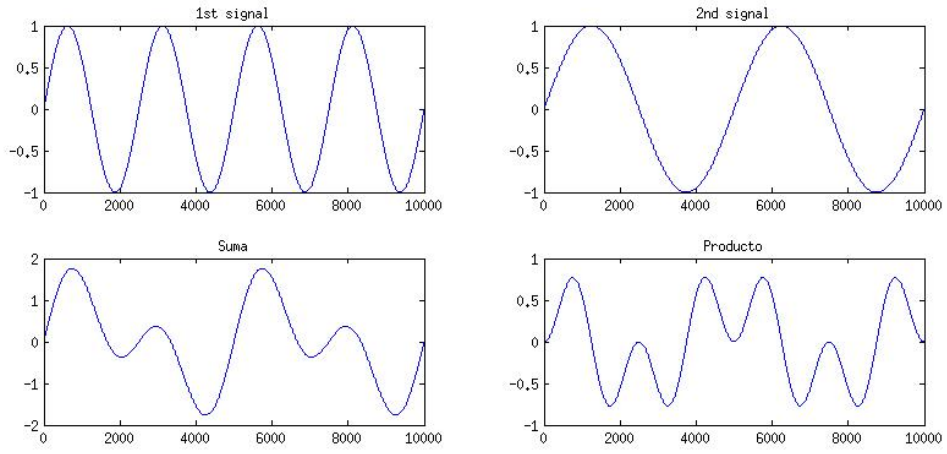

Figura 3: Representación de las señales y su suma.

Con estos ejemplo concluimos la introducción a la manipulación de señales en Matlab. A continuación se mostrarán una serie de operaciones con señales a realizar y se pedirá al alumno que implemente funciones genéricas para su uso, al igual que la funcion suma() dada. El objetivo de esto es crear una librería de funciones para poder más tarde crear sistemas complejos de una manera fácil.

4. Transformaciones de variable independiente en tiempo discreto

4.1. Desplazamiento temporal

Recordemos que en el ordenador sólo tenemos almacenado el intervalo de interés de la secuencia, y que en Matlab estamos representando cada secuencia con dos vectores, uno que almacena los instantes de tiempo y otro que almacena los valores de la señal. Para hacer un desplazamiento temporal de la misma, sólo tenemos que **modificar el vector que almacena el eje temporal**, sumando o restando el valor del desplazamiento.

El siguiente código crea una señal x[n] y le realiza un desplazamiento de x[n -10], después representa ambas señales. El alumno deberá implementar una función **[yo, no] = desplaza (x,n, d)** que desplace la señal dada siendo d el desplazamiento.

```
x = [0:0.1:1 ones(1,5)]; % construimos la secuencia x
nx = 1:16;               % eje temporal para x
figure()          % la representamos graficamente
subplot(1,2,1)
stem(nx,x);
axis([-1 30 -0.2 1.2]);
title('x[n]')

ny = nx + 10; % la retardamos 10 instantes de tiempo
y = x; % la forma de la secuencia se queda como estaba
subplot(1,2,2)
stem(ny,y);
axis([-1 30 -0.2 1.2]);
title('x[n - 10]')
```

La gráfica que genera al código anterior es:

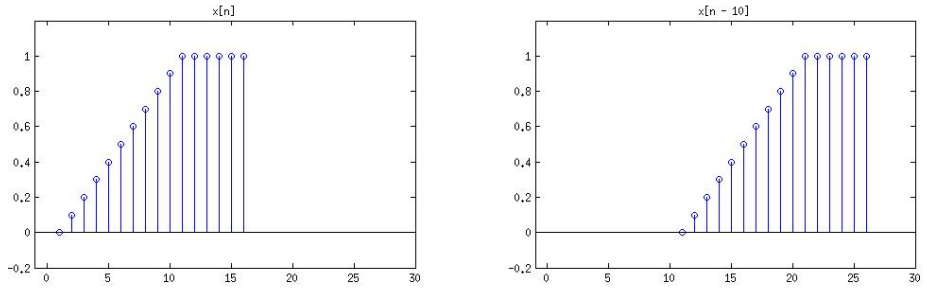

Figura 4: Desplazamiento.

4.2. Abatimiento

Abatir una secuencia $x[n]$ significa construir una nueva secuencia $y[n] = x[-n]$. Para realizar esta operación en MatLab hay pensar de la misma manera que se haría con papel y lápiz, debemos tener hacer un espejo de los valores de la señal $y[n]$ y además hacer un espejo del dominio n e invertirlo.

El siguiente código crea una señal x[n] y le realiza un abatimiento de x[-n], después representa ambas señales. El alumno deberá implementar una función **[yo, no] = abate(x,n)** que abata la señal dada.

```
x = [0:0.1:1 ones(1,5)]; % construimos la secuencia x
nx = 1:16;               % eje temporal para x
figure()          % la representamos graficamente
subplot(1,2,1)
stem(nx,x);
axis([-20 20 -0.2 1.2]);
title('x[n]')

y = x(end:-1:1); % abatimos los valores de la sennal
ny = -nx(end:-1:1); % tambien hay que abatir el vector de tiempos!!

subplot(1,2,2)
stem(ny,y);
axis([-20 20 -0.2 1.2]);
title('y[n]=x[-n]');  % titulo
```

La gráfica que genera al código anterior es:

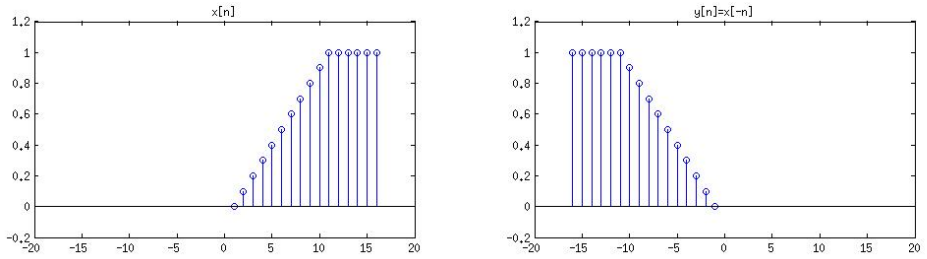

Figura 5: Abatimiento.

4.3. Diezmado

Diezmar una secuencia consiste en construir otra secuencia a partir de desechar muestras de la secuencia original. Un diezmado de grado M, $y[n] = x[n \cdot M]$ toma sólo uno de cada M valores de la señal original.

El siguiente código crea una señal x[n] y le realiza un abatimiento de $y[n] = x[3n]$, después representa ambas señales. El alumno deberá implementar una función **[yo, no] = diezma(x,n, M)** que diezme la señal dada con un factor M.

```
x = [0:0.1:1 ones(1,5)]; % construimos la secuencia x
nx = 1:16;               % eje temporal para x
figure()          % la representamos graficamente
subplot(1,2,1)
stem(nx,x);
axis([-1 20 -0.2 1.2]);
title('x[n]')

xaux = [x x x];  % Triplicamos la señal en longitud
Ny = floor(length(x)/3);  % Cogemos una de cada 3 muestras
y = [];
for k=1:Ny
  y = [y xaux(k*3)];
end
ny = 1:Ny;  % eje temporal para y

subplot(1,2,2) % representamos graficamente
stem(ny,y);
axis([-1 20 -0.2 1.2]);
title('y[n]=x[3n]');  % titulo
```

La gráfica que genera al código anterior es:

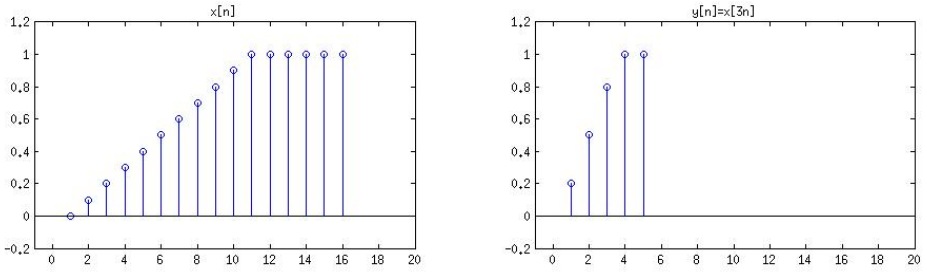

Figura 6: Diezmado.

4.4. Interpolado

Esta operación de interpolado de ceros, añade ceros entre las muestras originales de la señal, alargando la duración de la misma. Un interpolado de grado M, $y[n] = x[n/M]$ introduce M - 1 ceros entre cada par de valores de la señal.

El siguiente código crea una señal x[n] y le realiza un abatimiento de $y[n] = x[n/2]$, después representa ambas señales. El alumno deberá implementar una función **[yo, no] = interpola(x,n, M)** que interpole la señal dada con un factor M.

```
x = [0:0.1:1 ones(1,5)]; % construimos la secuencia x
nx = 1:16;               % eje temporal para x
figure()          % la representamos graficamente
subplot(1,2,1)
stem(nx,x);
axis([-1 20 -0.2 1.2]);
title('x[n]')

Ny = length(x)*2;
y = [];
for k=1:Ny   % interpolamos con ceros
   if rem(k,2) == 0
      y = [y x(k/2)];
   else
      y = [y 0];
   end
end
ny = 1:Ny;  % eje temporal para y

subplot(1,2,2) % representamos graficamente
stem(ny,y);
axis([-1 40 -0.2 1.2]);
title('y[n]=x[n/2]');  % titulo
```

La gráfica que genera al código anterior es:

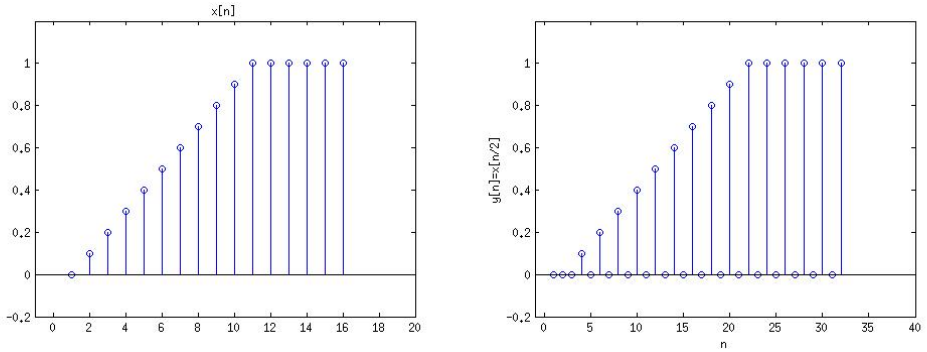

Figura 7: Intercalado.

5. Cálculo de Energía y Potencia

5.1. Energía

La energía de una señal discreta no es más que la suma de sus valores al cuadrado. El siguiente código calcula la energía de una secuencia almacenada en una variable Matlab `x`.

```
Energia = sum(abs(x).^2)
```

5.2. Potencia

En general, para secuencias de duración finita no estaremos interesados en calcular la potencia total, porque será cero. Para las secuencias de duración finita se suele considerar la **potencia promedio en el intervalo de interés** (aquel en que la secuencia es distinta de cero). En el caso de la secuencia del ejemplo anterior, esta potencia promedio en el intervalo de interés se calcularía en Matlab mediante

```
Potencia = mean(abs(x).^2)
```

En el caso de señales periódicas, en Matlab es suficiente con almacenar un periodo.

6. Miniproyectos

Los conocimientos adquiridos en esta sesión nos permiten abordar una serie de miniproyectos de procesamiento de señal que impliquen la realización de operaciones sencillas con señales.
El enunciado de la práctica se acompaña de una serie de ficheros `.wav` que contienen fragmentos de la banda sonora de películas clásicas. Dichos fragmentos corresponden a música o a voz.

6.1. Miniproyecto 1: Mezcla de música con *fading*

Este proyecto consiste en conseguir que dos bandas sonoras suenen alternativamente usando transiciones suaves. Vamos a interpolar el tema principal de *El Gran Golpe* con el tema principal de *La Guerra de las Galaxias.*

1. **Carga de los ficheros de audio.** Mediante la función `wavread.m` se leen las formas de onda correspondientes a los clips. El siguiente ejemplo lee el track de starwars, guarda su longitud en Lsw y crea un dominio n para la misma. Con la función escucha() podremos oir los clips.

   ```
   starwars = wavread('./Audios/starwars.wav');  % audioread si no funciona.
   Lsw = length(starwars);   % Numero de muestras de starwars
   n = 1:Lsw;                % Creamos el dominio discreto de la señal
   escucha(starwars);   % Escuchamos la señal
   ```

   Análogamente cargue el clip de *La Guerra de las Galaxias* en una variable llamada `StarWars`.

2. **Alineamiento de longitudes.** Matlab emplea vectores para representar las señales. Las operaciones entre señales se realizan en Matlab mediante operaciones entre vectores. Para poder operar con dos o más vectores es necesario que todos tengan la misma longitud. El comando Matlab `length` nos devuelve la longitud de un vector. Empléelo para calcular la longitud de los clips de audio.

   Recorte el vector con la señal de audio más larga de las dos para que ambos vectores tengan la misma longitud. Asumiremos que ambos Audios empezaron a ser muestreados en el mismo instante de tiempo por lo que comparten el mismo dominio.

3. **Diseño del *fading*.** El *fading* se puede implementar multiplicando cada señal por una sinusoide del tipo $0.5 + 0.5 * cos(2\pi n/M)$. Esta sinuoside oscila entre 0 y 1, cuando tome valores próximos a 1 el audio se escuchará más o menos intacto. Para ello genere dicha señal utilizando las fórmulas vistas anteriormente. La siguiente gráfica muestra de forma visual el audio multiplicado por una sinusoide de 5 ciclos.

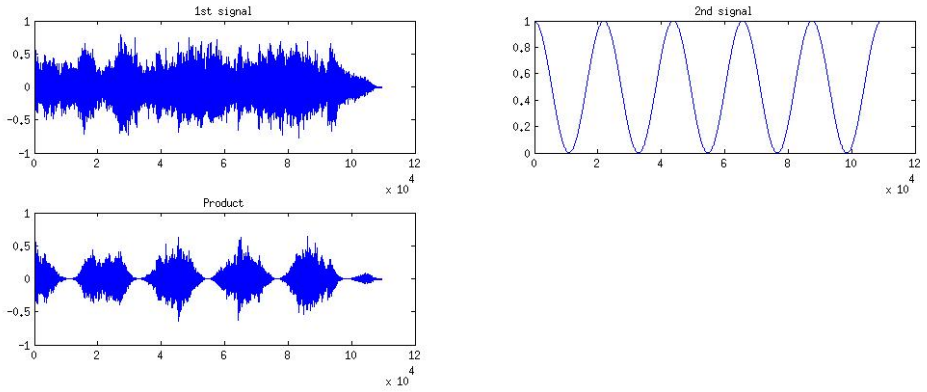

Figura 8: Fading.

4. **Construir la combinación que interpole los dos audios**. Primeramente se obtiene la señal `x2` multiplicando el segundo clip de audio (*El Gran Golpe*) por la señal de atenuación `s2`. Finalmente se suman las dos señales y se reproduce invocando la función Matlab `escucha` que se suministra con el enunciado de la práctica.

```
x = x1 + x2;
escucha(x);
```