**Laboratorio 2: Señales y Sistemas**

Última modificación: 11/10/20

**Objetivos**

**Esta sesión de laboratorio busca:**

**a) Afianzar conceptos de frecuencia discreta, su relación con la frecuencia analógica y en qué consiste el aliasing.**

**b) Cuantización de señales, efectos en imágenes y sonido.**

**c) Sistemas lineales: resolución de la ecuación en diferencias, respuesta impulsiva, sistemas FIR e IIR, convolución.**

**Actividades**

**a) Señales discretas**

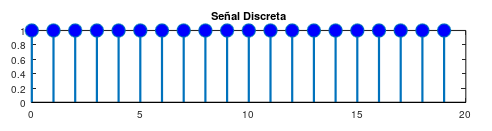
**• Representar señales discretas con diferentes valores de f.**

**• Ejemplos:**

**1. Dibuje con Octave las señales *x* [*n*]=cos(2π *f n*+θ) con f=0, f=0.1, f=0.25, f=0.5 c/m con n=[0,19] y con θ = 0 y 90º**

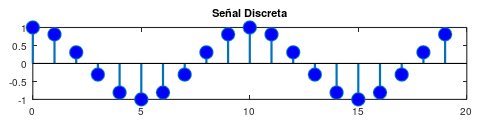
**Fichero: Lab2.m**

| %Inicializar el entorno en limpio clear all close all clc  % Declarar variables auxiliares N=20; n=0:N-1; %matriz de elementos f=0; fase=0;  %Generamos la señal x=cos(2\*pi\*f\*n+fase);  %Representamos stem(n,x,'linewidth',2,'markerfacecolor','b'); |
| --- |

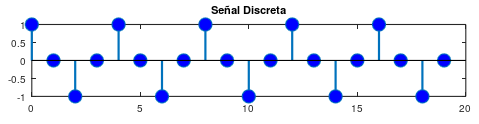


Al tener f=0 al señal es continua o constantes. No varía en el ‘tiempo’.

Si cambiamos la frecuencia f=0.1 y aplicamos el teorema del muestreo tendremos 1 ciclo cada 10 muestras, por lo que tendremos 2 ciclos de una función de coseno en las 20 muestras.



¿Si la frecuencia f=0.25 cuantos ciclos voy a tener?

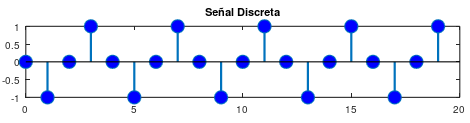


Como capturamos 4 muestras por ciclo, y tenemos 20 muestras obtenemos 5 ciclos.

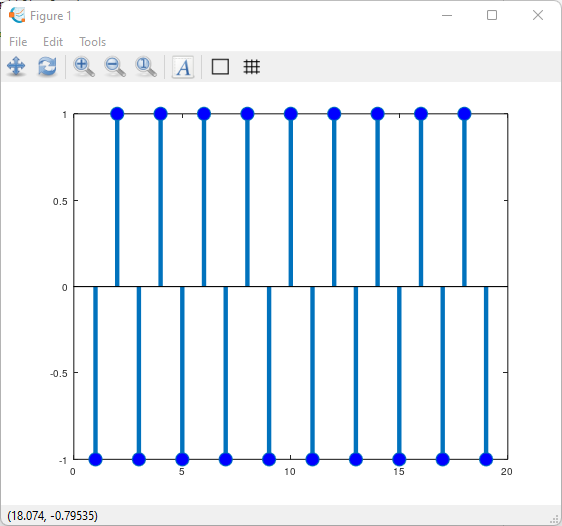
Ahora cambiamos el ángulo de la señal a 90º, recordarles que debemos pasar de grados a radianes en Octave:

* pi/2
* 90\*pi/180
* deg2rad (90)

**Esto nos va a afectar no en el número de muestras sino en lo atrasado o retrasado.** El cambio de la fase va a indicar dónde comienza el ciclo. Por ejemplo, si ponemos una fase negativa (-pi/2), la señal se nos va a desplazar hacia la izq (sentido contrario). La fase lo que provoca son desplazamientos de izq a derecha dependiendo del signo de la fase.

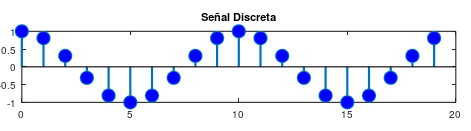


Para f=0.5 máximo valor de muestreo y ángulo 0. Esta es la máxima frecuencia representable en el mundo discreto. Cada ciclo se representa con dos muestras. Con menos que esto no podríamos representar la función.

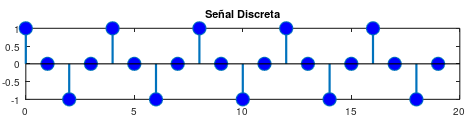


**2. Repita el ejemplo anterior con f= 0.9, f=0.75, f=1.1, f=2 y f=-0.1c/m. ¿Existen diferencias con las señales anteriores?**

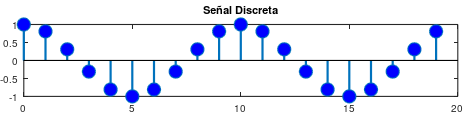
f=0.9



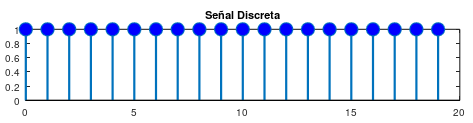
f=0.75



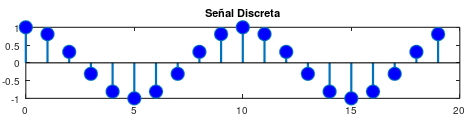
f=1.1



f=2

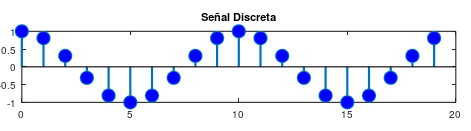
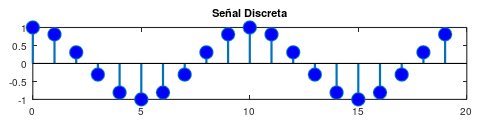


f=-0.1

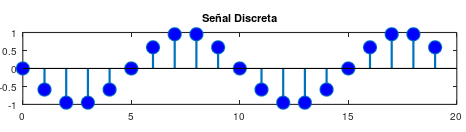
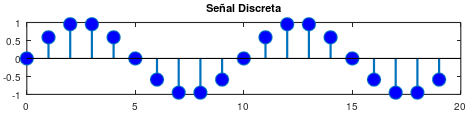


Lo que está ocurriendo es que al salirnos del rango de muestreo de entre 0 y 0.5 lo que obtenemos son sus equivalentes, debido a que la fase es 0

f=0.9 fase=0 f=0.1 fase=0

Pero que ocurre si cambiamos la fase a 90º observamos que ya no se cumple. Las señales no son idénticas.

f=0.9 fase=90º f=0.1 fase=90º  

Incremente en números enteros implementan señales que son idénticas

0.1= 1.1= 2.1= 3.1

-0.1 = 0.9 porque la distancia entre ellas es 1 número entero

**3. Calcule la transformada de Fourier de los ejemplos anteriores**

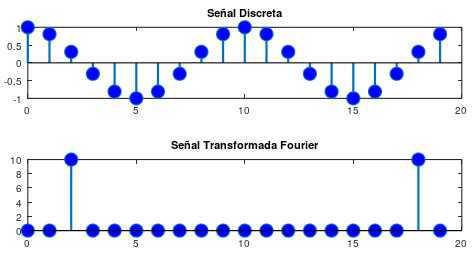
La función en Octave la transformada de Fourier es ttf(muestras)

**Fichero: Lab2.m**

| %Preparamos el entorno clear all close all clc  N=20; n=0:N-1; %matriz de elementos f=0.1; fase=0; % Debe estar en radianes grados \* pi / 180 o la funcion deg2rad(90)  %Generamos la señal x=cos(2\*pi\*f\*n+fase);  %Representamos subplot(211); stem(n,x,'linewidth',1,'markerfacecolor','b');  %Calculamos al transformada de fourier Xk=fft(x);  %Calculamos el modulo k=n; modXk=abs(Xk); subplot(212); stem(k,modXk,'linewidth',1,'markerfacecolor','b'); |
| --- |

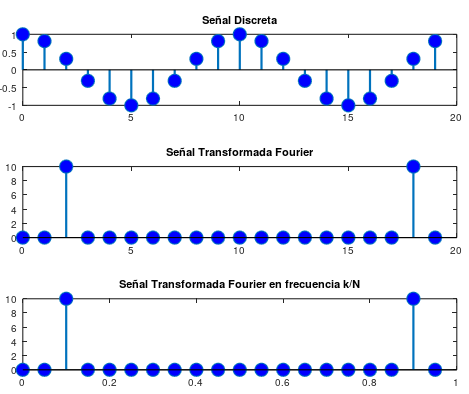
En la función **subplot** -> primer número indica el número de filas (2 porque tenemos 1 gráfica en cada fila), el segundo el número de columnas (1 porque tenemos una debajo de la otra) y el tercero es el número del gráfico.

Sí, observamos la gráfica superpuesta de la frecuencia discreta y de su transformada de Fourier para f=0.1 y fase=0



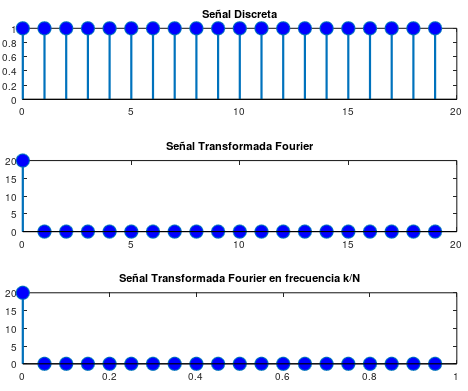
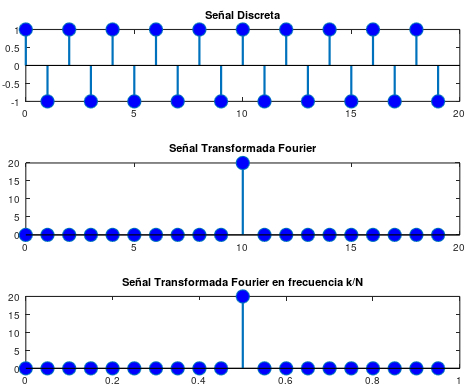
Para k=2 tenemos ->k=f\*N ->f=2/20=0.1 (es la frecuencia asociada a nuestra señal)

Por lo general, la transformada de una señal de coseno va a generar 1 o 2 espigas y esto es debido a que en la señal obtenemos un valor para el sentido antihorario y otro para el sentido horario que representan la misma señal como vimos para todas las señales cuya distancia sea 1 número entero.

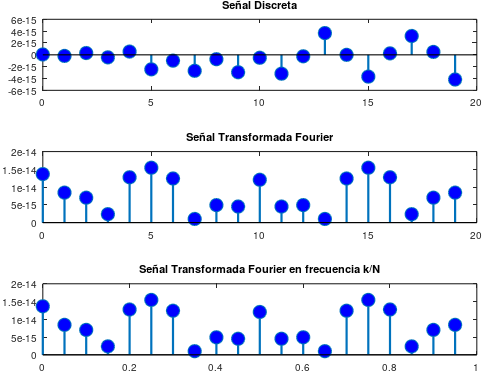
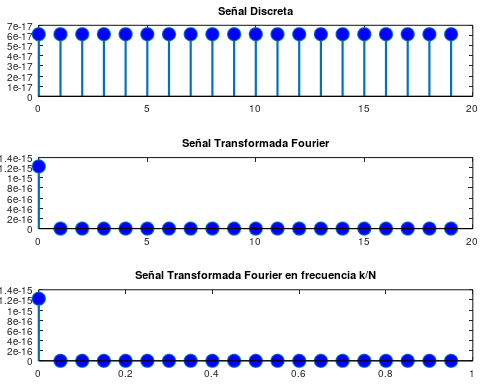


Si lo comparamos con la señal transformada, pero con base en su frecuencia, observamos más claramente como esta señal para 0.1 tiene su homólogo en 0.9. Sin embargo, como para nosotros, la menor tasa es f=0 que es una señal continua y la mayor f=0.5, solo tendremos en cuentas las situadas a la izquierda en las gráficas

f=0 fase=0 f=0.5 fase=0

f=0 fase=90 f=0.5 fase=90



Tenemos que mirar siempre hasta 0.5

Para una señal continua (f=0), tenemos como transformada de Fourier una gráfica delta.

Con f=0.5 que es la frecuencia máxima nos aparece una única espiga en medio (k=0.5). Para el resto de frecuencias nos aparecerán dos espigas, y nos quedamos con la de la izquierda (previa a 0.5)

**• Relación de frecuencias discretas y analógicas. Teorema de muestreo**

**Ejemplos:**

**1. Crea una señal senoidal para audio con una frecuencia de 500Hz durante 1.5s. Cambie la frecuencia a 1kHz. Escuche el audio sabiendo que la Fs=8kHz. Repita para 7kHz. Tenemos que llevar a cabo este experimento con los siguientes datos:**

**F = 500 Hz, 1kHz, 7KHz Fs = 8KHZ**

Para los dos primeros casos de la frecuencia, se cumple el teorema de muestreo en el que la frecuencia tiene que ser menor que la mitad de la frecuencia de muestreo (F<FS/2).

Para el caso de 7 kHz no. Cuando pasamos al dominio de f discreto, podemos observar que 7/8 (f=F/Fs) está dentro del rango de 0.5 y 1, por lo que tiene que poder mapearse con un valor dentro del rango [0, 0.5] el alias de esta señal sería 1/8 (el mismo que el de 1KHz). Las señales discretas distintas que podemos representar en el mundo discreto está en el rango [0, 0.5].La máxima frecuencia a la que podríamos implementar con Fs (8khz) 4khz. La frecuencia de muestreo de 8kHz significa que vamos a generar 8000 muestras en un segundo. Como nos indican que vamos a escuchar la señal durante 1.5s, tenemos que hacer una regla de tres:

Tiempo = 1.5 segundos Fs=8kHz

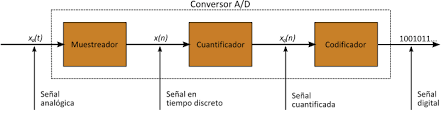
8000 muestras —> 1 s N muestras —> 1.5 s => N=Fs\*T;

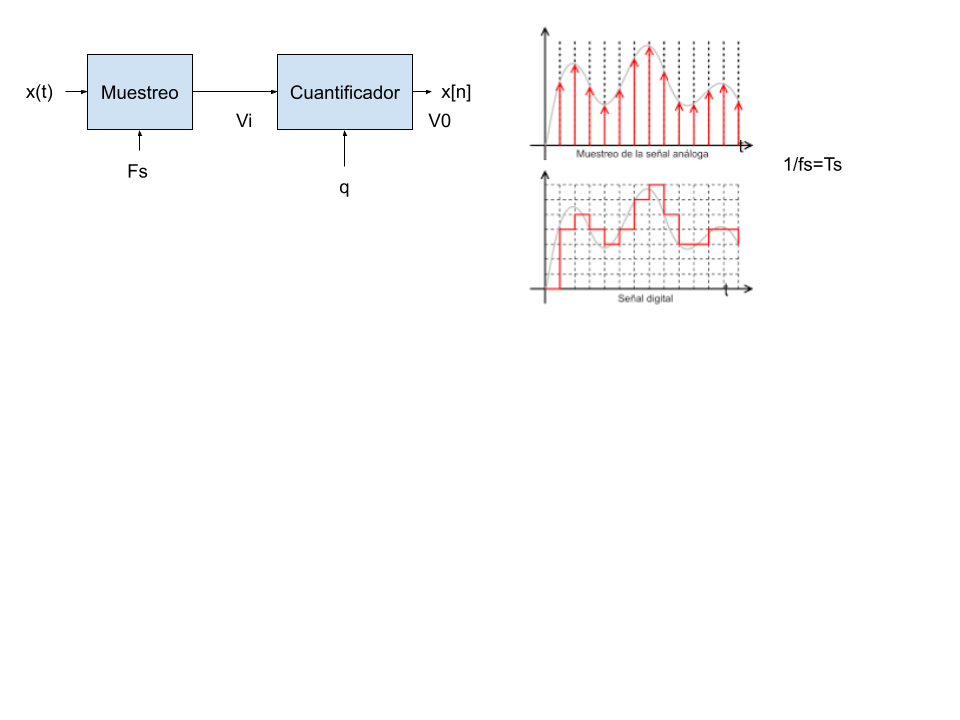
**Fichero: Ejemplo1.m**

| %Preparamos el entorno clear all close all clc  T=1.5; %Tiempo en segundos Fs=8e3;%Frecuencia muestreo F=1e3; %Frecuencia N=Fs\*T;%Muestras que es igual a la frecuencia de muestreo por el tiempo f=F/Fs; n=0:N-1;%Matriz de elementos x=cos(2\*pi\*f\*n); % Generamos el audio % help audiowrite audiowrite (FILENAME, Y, FS) audiowrite ('SonidoEj1.wav', x, Fs); %system('play SonidoEj1.wav')  %Otra opción es reproducirlo directamente player = audioplayer (x, Fs); play (player); |
| --- |

**b) Cuantización**

• Diseñe una función que permita la cuantización de datos de entrada-salida normalizados entre [0,1].



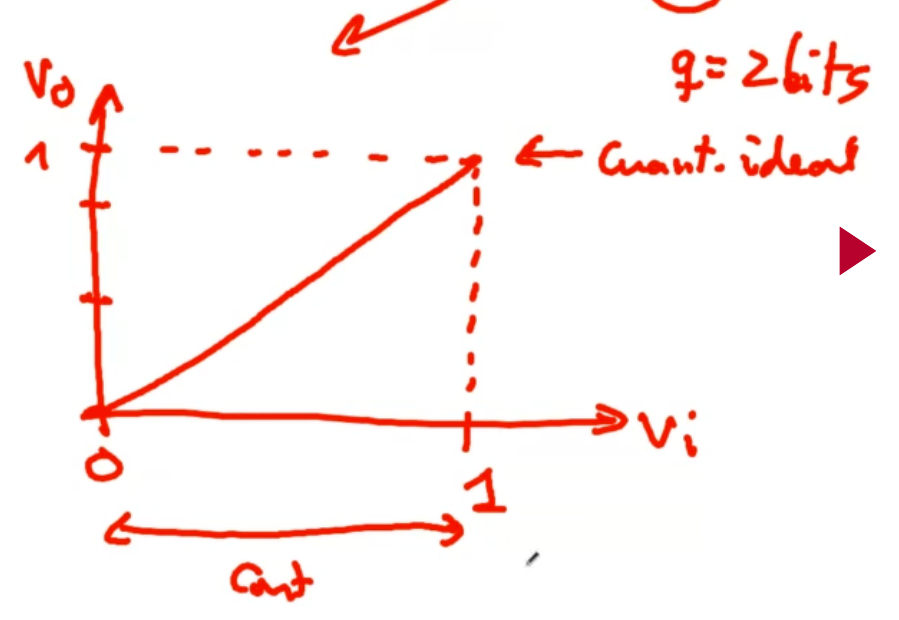


A partir de la señal continua, tenemos una serie de muestras en instantes concretos. La altura de esas muestras es el valor analógico, pero este valor analógico, cuando se guarda en la memoria de un dispositivo digital, también tiene que hacerlo de forma **finita**. Tenemos un número de bits determinado para guardar la magnitud de los números. Este número de bits, se representa como q y es lo que vamos a utilizar para representar la magnitud de la señal. Entonces nuestra señal es discreta para n, pero también para q.

Tenemos un voltaje de entrada, y otro a la salida dependiente de q. Este cuantizador tiene un error asociado.

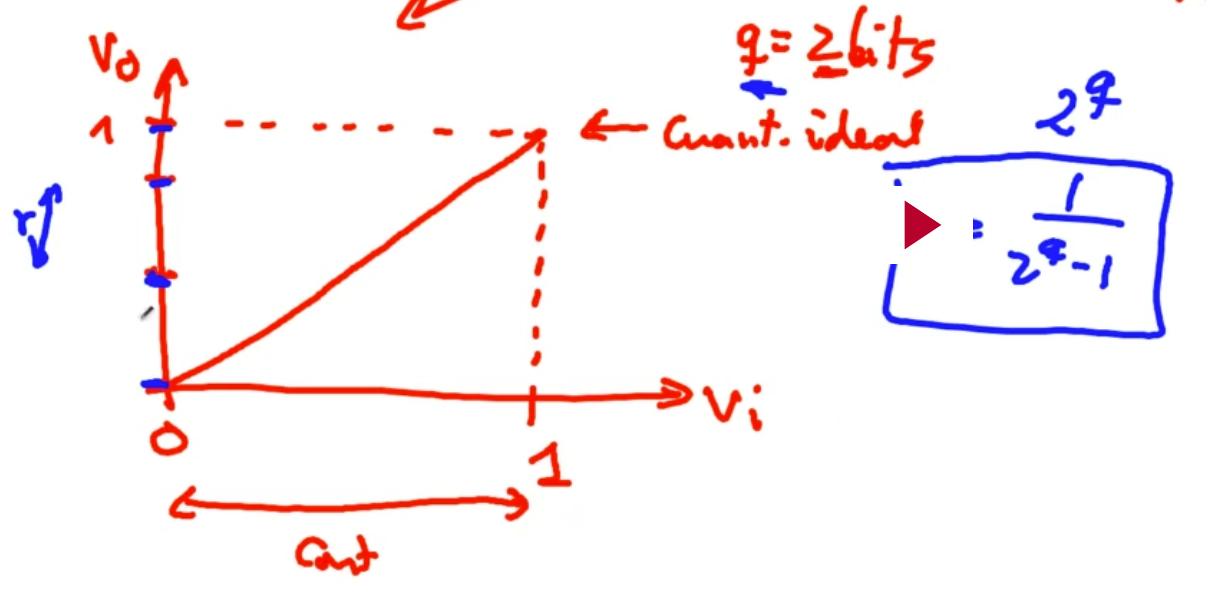
El rango de salida está normalizado entre 0 y 1. La tensión de entrada puede ser cualquier valor es continua. Si por ejemplo, el valor de q fueran 2 bits, solo podríamos guardar a la salida 4 valores posibles.

Idealmente, el cuantizador de salida debería poder general el mismo voltaje que el de la entrada:

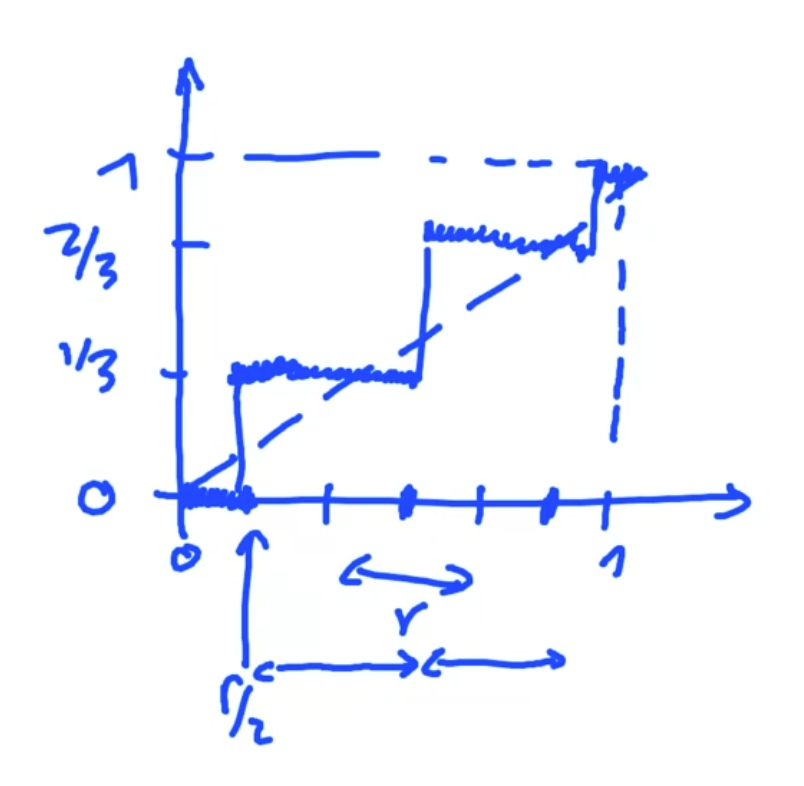


Sin embargo, vemos que el cuantizador no puede generar esta respuesta porque tenemos un número de bits limitado (en este ejemplo, solo tenemos disponibles 4 posibles valores de salida). El número de valores de salida viene determinada por la expresión **2^q** (potencias de 2).

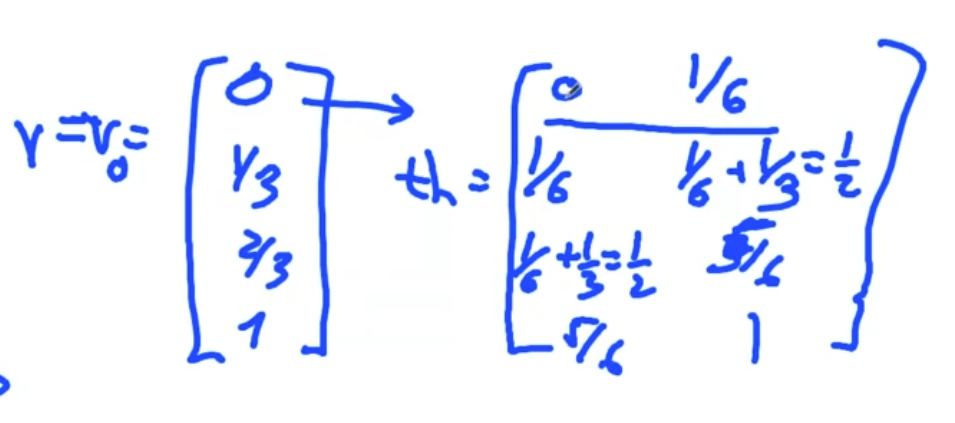
el intervalo entre los posibles valores de la salida viene definido por r con la siguiente fórmula: r=1/(2^q-1). como tenemos 4 valroes en este ejemplo, tenemos 3 intervalos.



Lo que hacemos es un diagrama como en escalera:



De esta forma, la salida quedaría:



• Ejemplos:

**Función cuantización octave:**

**Fichero: cuantizacion.m**

| %Funcion de cuantizacion function y= cuantizacion(in,p)  %in es la entrada  %p es el numero de bitset  %Asignamos la entrada a una variable  y=in;  %Intervalo de cuantización  r=(1/(2^p-1));  %Calculo de umbrales  th=r/2:r:1;  %Matriz  mt=[[0;th'] [th';1]]  v=0:r:1  %Cuantizacion  for n=1:length(v)  pos=find(in>=mt(n,1) & in<=mt(n,2));  y(pos)=v(n);  end endfunction |
| --- |

Para la función cuantizacion(0.1,2) obtenemos el valor 0 ya que 0.1 está dentro del primer intervalo y le corresponde ese valor

**1. Aplique la cuantización para diferentes niveles en las siguientes imágenes: beach.jpg hojas.jpg**

Cada pixel de la imagen es un número entero de 8 bit sin signo. Representa: 0 representa negro absoluto hasta 255 un blanco y tenemos entre medias la escala de grises.

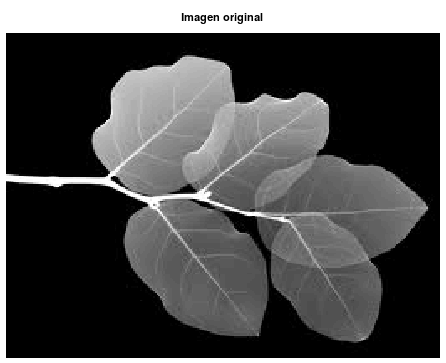
A esta imagen le vamos a pasar el cuantificador entre 0 y 1. Por defecto considera octave los números de double, pero si nos fijamos en la matriz donde hemos cargado la imagen, es que está en formato unit8 -> tenemos que pasarlo a double -> imd = double(im)/255 para normalizarlo;

**Imagen cuantización octave:**

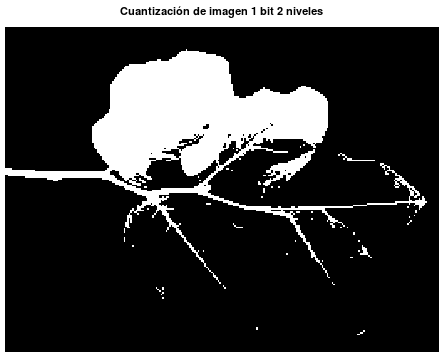
**Fichero: CuantizacionImagen.m**

| %Preparamos el entorno clear all close all clc  %Leemos la imagen %imagen=imread('beach.jpg'); imagen=imread('hojas.jpg'); %Tamaño de la imagen size(imagen) %Filas 888 Columnas 1334  %Al tratarse de una imagen en escala de grises %la matriz solo contiene numeros positivos enteros %entre 0 y 255 %imagen % 91 83 66 47 29 18 14 14 % 96 80 57 35 21 16 17 19 % 88 70 45 25 17 20 27 32 % 54 45 35 30 31 32 29 25  %Mostramos la imagen %imshow(imagen)  %Para aplicar la cuantización debemos %convertir los valoresd e la matriz de %de la imagen que son enteros sin signos a doble %ya que vamos a dividir en 255 %de esta forma hemos normalizado entre 0 y 1 imagenNormalizada=double(imagen)/255; %cuantizamos a nivel de 1 bit %obtendremos una imagen binaria imagenCuantizada1bit=cuantizacion(imagenNormalizada,1); imagenCuantizada2bit=cuantizacion(imagenNormalizada,2); imagenCuantizada3bit=cuantizacion(imagenNormalizada,3); %Para regenerar la imagen debemos volver %a convertir los valores double a int imagenCasting1bit=uint8(imagenCuantizada1bit\*255); imagenCasting2bit=uint8(imagenCuantizada2bit\*255); imagenCasting3bit=uint8(imagenCuantizada3bit\*255); %imshow(imagenCasting) %Mostrar imagenes %subplot(numerotablas,celdasTabla,posicion) figure('name','Procesamiento de imagenes.','NumberTitle','off'); imshow(imagen); title ("Imagen original");  figure('name','Procesamiento de imagenes.','NumberTitle','off'); imshow(imagenCasting1bit); title ("Cuantización de imagen 1 bit 2 niveles");  figure('name','Procesamiento de imagenes.','NumberTitle','off'); imshow(imagenCasting2bit); title ("Cuantización de imagen 2 bits 4 niveles");  figure('name','Procesamiento de imagenes.','NumberTitle','off'); imshow(imagenCasting3bit); title ("Cuantización de imagen 3 bits 8 niveles"); |
| --- |

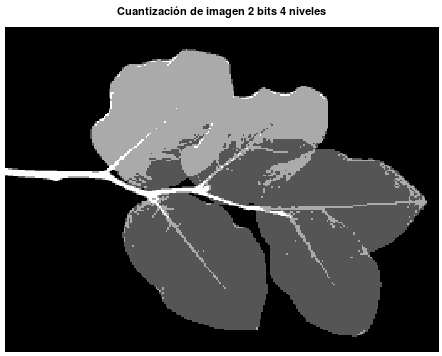
**Original**

****

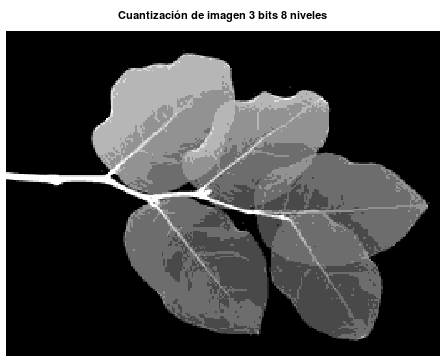
**Para 1 bit**

****

**Para 2 bits:**

****

**Para 3 bits:**

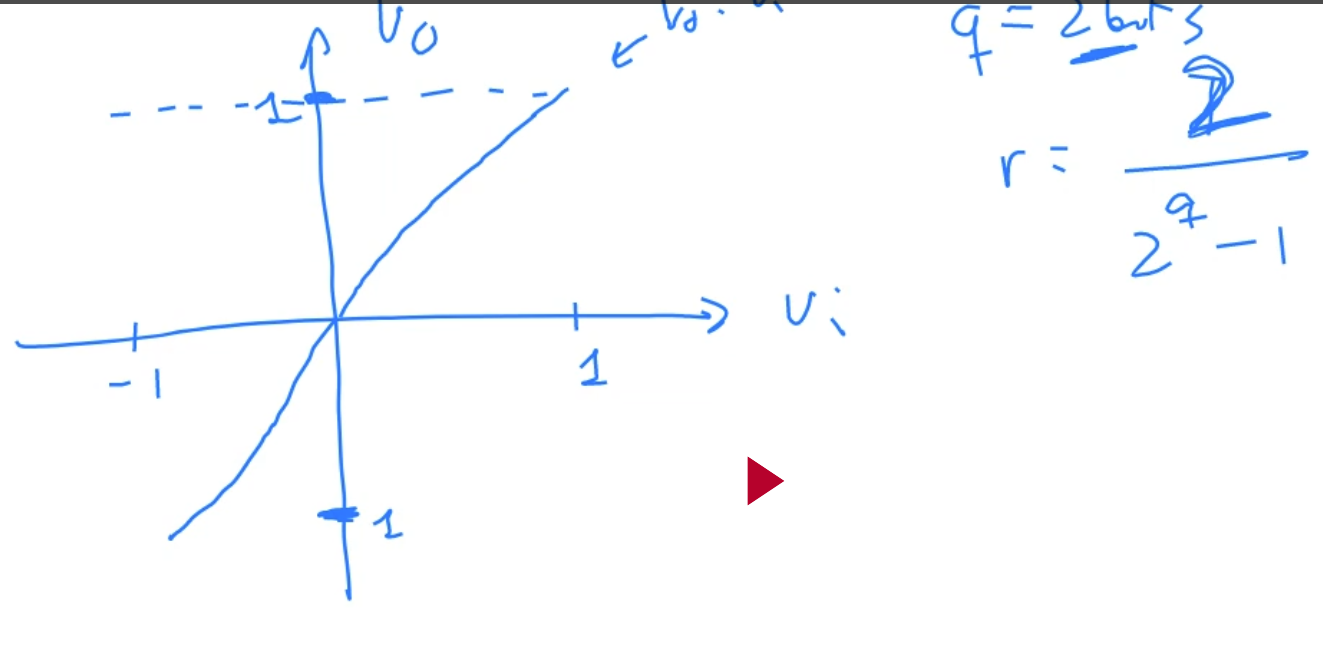
****

**Si q=3, tendríamos 8 niveles de grises; para 4 16 niveles de grises**

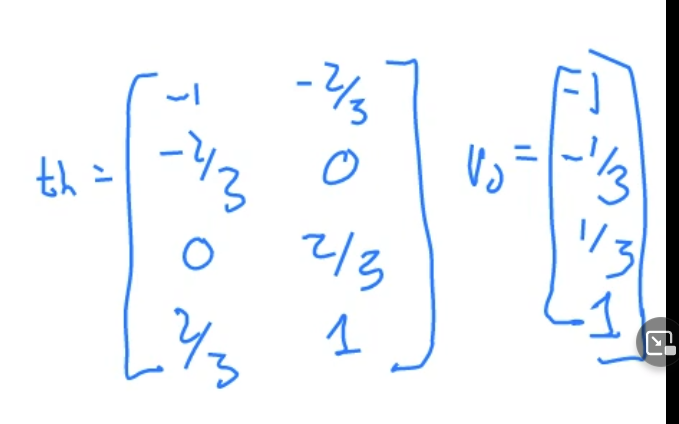
**(RECUERDA POTENCIA DE 2)**

**• Obtenga una función de cuantización para datos de entrada-salida normalizados entre [-1,1].**

**Ahora, al tener el rango de [-1,1]:**



r=2/(2^q-1) Para q=2:



**Fichero: cuantizacionbipolar.m**

| %Funcion de cuantizacion function y= cuantizacionbipolar(in,p)  %in es la entrada  %p es el numero de bitset  %Asignamos la entrada a una variable  y=in;  %Intervalo de cuantización es a 2  r=(2/(2^p-1));  %Calculo de umbrales debe empezar en -1  th=-1+r/2:r:1;  %Matriz  mt=[[-1;th'] [th';1]]  v=-1:r:1  %Cuantizacion  for n=1:length(v)  pos=find(in>=mt(n,1) & in<=mt(n,2));  y(pos)=v(n);  end endfunction |
| --- |

**Ejemplos**

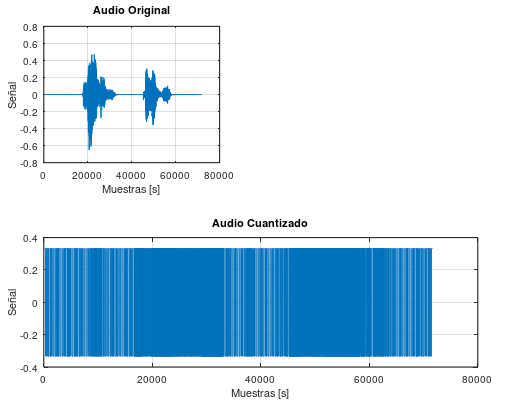
**1. Aplique la cuantización a una señal de audio y observe el ruido de cuantización para valores bajos y altos de cuantización**

**Fichero: CuantizacionAudio.m**

| % clear all close all clc  %Fichero='Pitido.wav'; Fichero='BuenosDias.wav'; nb=2 ; %Numero de bits  %La funcion audioread, genera una matriz de datos %entre sus valores nos da la señal y su Fs [senal Fs]= audioread(Fichero);  %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); subplot(221) plot(senal); grid on; xlabel('Muestras [s]'); ylabel('Señal'); title ("Audio Original");  %Reproducimos audio original player = audioplayer (senal, Fs); play (player); pause(5) %Procesamos el audio senalq=cuantizacionbipolar(senal,nb);  %Representamos la onda cuantizacion subplot(212) plot(senalq); grid on; xlabel('Muestras [s]'); ylabel('Señal'); title ("Audio Cuantizado");  %Guardamos el audio audiowrite ('Holaq.wav', senalq, Fs);  %Reproducimos audio cuantizado player = audioplayer (senalq, Fs); play (player); |
| --- |

Para minimizar el ruido y minimizar el error. Para ello, hay que aumentar el número de bits para la cuantización.

Para audio -> q=16 Para imagen -> q=8

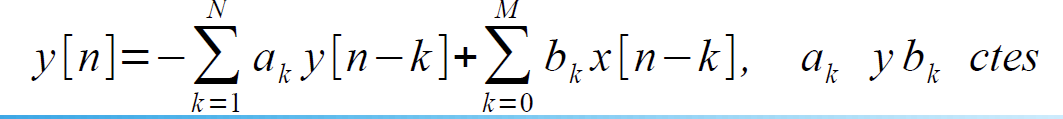


**c) Sistemas lineales**

**• Resolución numérica de sistemas discretos, lineales e invariantes en el tiempo**

Los sistemas LTI (lineales e invariantes con el tiempo) están caracterizados por ecuaciones en diferencias con coeficientes constantes.

y[n]=y[n-1]+0.5x[n-1]



En octave tenemos una función que se llama filter que nos permite resolver analíticamente este sistema de ecuaciones en diferencias. Entonces, si conocemos los coeficientes constantes, octave nos puede calcular la salida para cualquier entrada..

FILTER tiene como entrada:

* b ->coeficiente que multiplica a X[n]
* a-> coeficiente que multiplica a y[n]
* x -> entrada del sistema

y=filter(b,a,x);

Procedimiento para actuar:

1. Sacar las matrices b y a
   1. Tenemos que pasar al primer miembro todas las **y** y dejar en el segundo miembro las **x.**

y[n]-y[n-1]=0.5x[n-1]

* 1. Vemos los coeficientes A y B (números que multiplican a **y** y **x** respectivamente)

a=[1 -1] b=[0 0 0.5]

1. Introducimos en octave filter(b,a,x)
   1. Le ponemos a x tantos 0 como queramos representar. La longitud de la salida x es igual a la longitud de la entrada y.

x=[X1,x2,X3,Xn]

Podemos mostrar la salida con Stem

**Ejemplos con filter() para x[n] = {1,2,-1} , L=4 y c=0.8:**

**1. y[n] = x[n] + c x[n-L]**

MODELO DE ECO Simple. Escuchamos (nosotros somos y) el valor de entrada (x) y posteriormente la volvemos a oir (x(n-4)) con cierta atenuación.

|  | y | x |
| --- | --- | --- |
| L=0 | 1 | 1 |
| L=1 | 0 | 0 |
| L=2 | 0 | 0 |
| L=3 | 0 | 0 |
| L=4 | 0 | 0.8 |

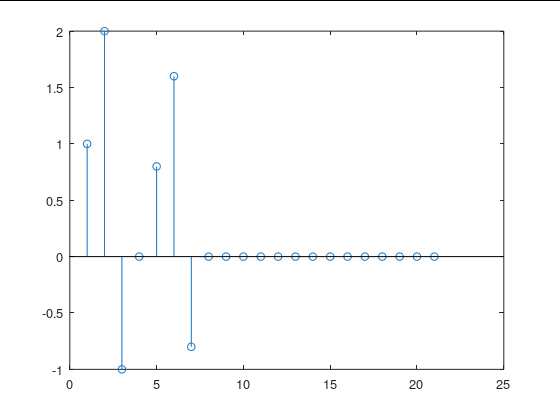
A=[ 1 0 0 0 0]

B=[1 0 0 0 0.8]

X=[1 2 -1 0 0]

y=filter(A, B, X)

Para mostrarlo -> stem(y)



**Es FIR**

**Fichero: EcuacionesLineales.m**

| %Preparamos el entorno clear all close all clc  ##Ejemplos con filter() para x[n] = {1,2,-1} , L=4 y c=0.8: ##1. y[n] = x[n] + c x[n-L]  x=[1 2 -1]; x=[x zeros(1,10)]; c=0.8;%Factor atenuación a=1;%representa las y de la ecuacion de diferencia b=[1 0 0 0 c];%represetna las x de la ecuacion de diferencia y=filter(b,a,x)  %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); stem(y); title ("Señal Eco"); |
| --- |

**2. y[n] = c y[n-L] + x[n]**

Modelo de reverberación (variante del eco). Lo que reintroducimos es la propia salida

**OJO -> pasar todas las y al primer miembro**

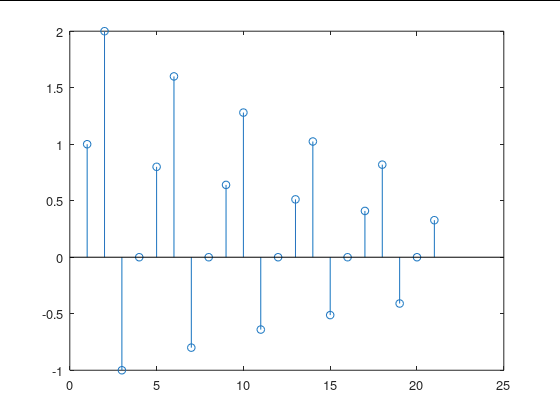
y[n] - c y[n-L]= x[n]

|  | y | x |
| --- | --- | --- |
| L=0 | 1 | 1 |
| L=1 | 0 | 0 |
| L=2 | 0 | 0 |
| L=3 | 0 | 0 |
| L=4 | -0.8 | 0 |

A=[1 0 0 0 -0.8]

B= [ 1 0 0 0 0]

X=[1 2 -1 0 0]

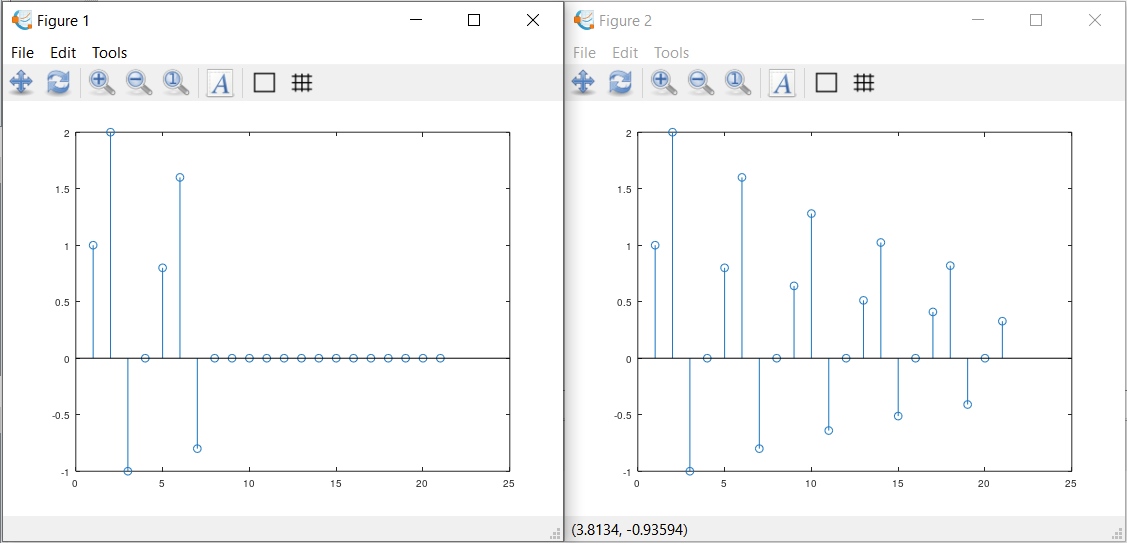


Es **IIR**

**Fichero: EcuacionesLineales.m**

| ##2. y[n] = c y[n-L] + x[n] x=[1 2 -1]; L=10; x=[x zeros(1,10+L)]; c=0.8;%Factor atenuación  a=1;%representa las y de la ecuacion de diferencia b=zeros(1,L+1); b(1)=1;%represetna las x de la ecuacion de diferencia b(L)=c; y=filter(b,a,x)  %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); stem(y); title ("Señal Rever"); |
| --- |

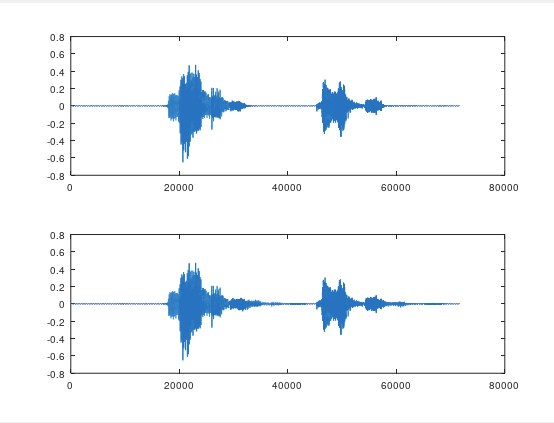
**Comparación de un FIR e IIR**

****

**3. Aplicación a eco y reverberación de 'BuenosDias.wav' y para un td= 1.5s**

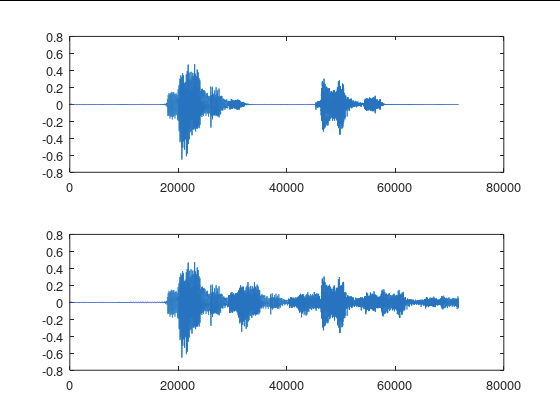
**Fichero: ELAudioEco.m**

| ##3.Aplicación a eco de 'BuenosDias.wav' y para un td= 1.5s Fichero='BuenosDias.wav'; %La funcion audioread, genera una matriz de datos %entre sus valores nos da la señal y su Fs [x Fs]= audioread(Fichero); td=0.5; ##Necesitamos la Fs para poder calcular L L=Fs\*td;; c=0.6;%Factor atenuación a=1;%representa las y de la ecuacion de diferencia b=zeros(1,L+1); b(1)=1;%represetna las x de la ecuacion de diferencia b(L)=c; y=filter(b,a,x) %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); subplot(211) plot(x) title ("Señal Audio"); subplot(212) plot(y);%Mejor para grandes cantidades de datos title ("Señal Eco Audio");  %Reproducimos audio cuantizado player = audioplayer (x, Fs); play (player); pause(6); player = audioplayer (y, Fs); play (player); |
| --- |



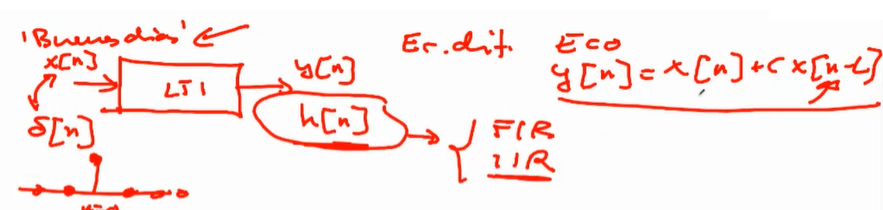
**Fichero: ELAudioRever.m**

| ##3.Aplicación reverberación de 'BuenosDias.wav' y para un td= 1.5s Fichero='BuenosDias.wav'; %La funcion audioread, genera una matriz de datos %entre sus valores nos da la señal y su Fs [x Fs]= audioread(Fichero); td=0.5; ##Necesitamos la Fs para poder calcular L L=Fs\*td;; c=0.6;%Factor atenuación b=1;%representa las y de la ecuacion de diferencia a=zeros(1,L+1); a(1)=1;%represetna las x de la ecuacion de diferencia a(L)=-c; y=filter(b,a,x) %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); subplot(211) plot(x) title ("Señal Audio"); subplot(212) plot(y);%Mejor para grandes cantidades de datos title ("Señal Rever Audio");  %Reproducimos audio cuantizado player = audioplayer (x, Fs); play (player); pause(6); player = audioplayer (y, Fs); play (player); |
| --- |



**• Respuesta impulsiva h[n] de los sistemas anteriores**

La entrada para estos sistemas es una señal lineal, genera lo que es una salida denominada respuesta impulsiva.



Con base en esto, los sistemas se pueden clasificar como:

* FIR : Respuesta impulsiva es finita.
  + ECO es FIR, ya que vemos fácilmente que la función no es recursiva
* IIR: Respuesta impulsiva es infinita.
  + REVER es IIR

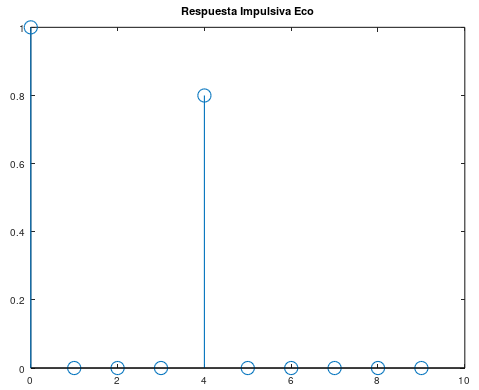
La respuesta impulsiva es muy importante, porque muchas veces imposible encontrar un modelo de la acústica del entorno.

h[n] -> siempre es [1 0 0 0 ..] es un delta.

Obtener la repuesta impulsiva del eco y del rever.

**Fichero: RespuestaImpulsivaEco.m**

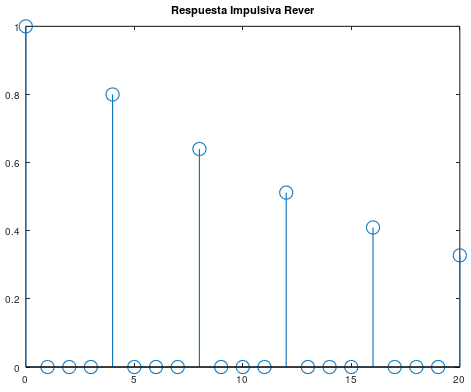
| %Preparamos el entorno clear all close all clc %y[n] = x[n] + c x[n-L] L=4; c=0.8;%Factor atenuación a=1;%representa las y de la ecuacion de diferencia b=zeros(5,1); b(1)=1;%represetna las x de la ecuacion de diferencia b(L+1)=c; h=filter(b,a,delta(0:9)) n=0:9;  %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); stem(n,h) title ("Señal Audio"); |
| --- |



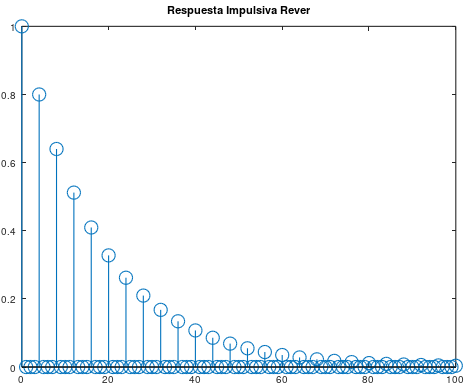
Aunque incrementemos las muestras, veremos que no aumentan las señales, por lo que es FIR.

**Fichero: RespuestaImpulsivaRever.m**

| %Preparamos el entorno clear all close all clc %y[n] = c y[n-L] + x[n] L=4; c=0.8;%Factor atenuación  a=[1 0 0 0 -c]; b=1; n=0:20; h=filter(b,a,delta(n))  %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); stem(n,h); title ("Respuesta Impulsiva Rever"); |
| --- |

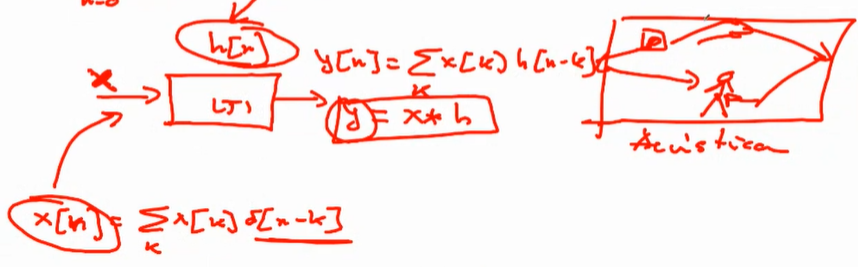


Al incrementar las muestras, veremos que aumentan las señales, por lo que es IIR.



**• Convolución 'conv()'**

Con la ayuda de la repuesta impulsiva para modelar por LTI obtenemos el método de la convolución

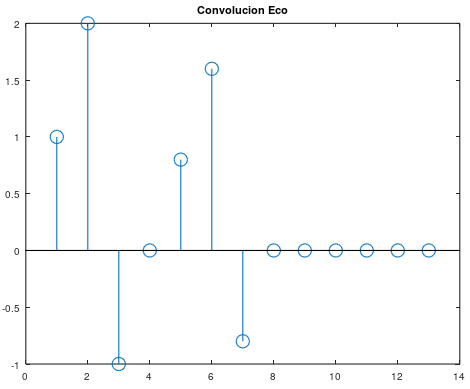


Para modelar la acusica empleamos una señal breve y potente, que se captara por un micrófono. Esta respuesta impulsiva nos ayudará a modelar el sonido.

Especial atención a la hora de realizar la convolución, para los sistemas FIR al ser finitos no tendremos problemas para la modelización, pero para los IIR que son infinitos deberos trabajar con muestras, esto limitará su precisión.

**Fichero: ConvolucionEco.m**

| %Preparamos el entorno clear all close all clc %y[n] = x[n] + c x[n-L] L=4; c=0.8;%Factor atenuación a=1;%representa las y de la ecuacion de diferencia b=zeros(5,1); b(1)=1;%represetna las x de la ecuacion de diferencia b(L+1)=c; h=filter(b,a,delta(0:9)) n=0:9;  %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); stem(n,h); title ("Respuesta Impulsiva Eco");  x=[1 2 -1 zeros(1,10)]; y=filter(b,a,x); yc=conv(h,x);  %Representamos la onda figure('name','Procesamiento de audio.','NumberTitle','off'); stem(y); title ("Convolucion Eco"); |
| --- |

****

Lo que hemos demostrado es que podemos resolver el sistema para encontrar el modelo que representa el audio o podemos mediante la respuesta impulsiva obtener el modelo, ambas formas son equivalentes.

**• Ejemplos:**

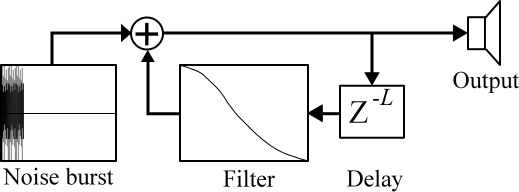
**1. Eco y reverberación**

**2. Acústica de una habitación**

| %Preparamos el entorno clear all close all clc  ## Convolución entre el sonido de disparo y el buenos dias ##Informacion de los audios infoDisparo=audioinfo('02SonidoMagnun500disparando.wav') infoBdias=audioinfo('BuenosDias.wav')  %Leemos audio disparo [Disparo FsDisparo]= audioread('02SonidoMagnun500disparando.wav'); %El audio esta en stereo quitamos un canal Disparo=Disparo(:,1);  %Generamos la grafica para el disparo figure('name','Procesamiento de audio.','NumberTitle','off'); subplot(221) plot(Disparo); title ("Audio Disparo");  %Leemos audio saludo [Bdias FsBdias]= audioread('BuenosDias.wav');  %Generamos la grafica para el saludo subplot(222) plot(Bdias); title ("Audio Buenos Dias");  %Generamos la convolucion entre el saludo y el disparo y=conv(Bdias,Disparo);  %Generamos la grafica para la convolucion subplot(212) plot(y); title ("Audio Convolucion Buenos Dias<>Disparo");  %Debemos normalizar el audio (desaturar) %y=y/max(y); y/=max(y);  %Generamos el fichero de audio de la convolucion audiowrite('acustica.wav',y,FsBdias);  %Reproducimos disparo playerDisparo = audioplayer (Disparo, FsDisparo); play(playerDisparo); %Obtenemos la duracion para que los audios no se solapen %introducimos una espera igual a la duración+1 pause(infoDisparo.Duration+1);  %Reproducimos el saludo playerBdias = audioplayer (Bdias, FsBdias); play(playerBdias);  %Reproducimos la convolucion %Como trabajamos con frecuencias diferentes %la función conv usa la Fs menor %eso lo podemos ver en su info infoAcustica=audioinfo('acustica.wav') [Acustica FsAcustica]= audioread('acustica.wav'); playerConvolucion = audioplayer (Acustica, FsAcustica); play(playerConvolucion); |
| --- |

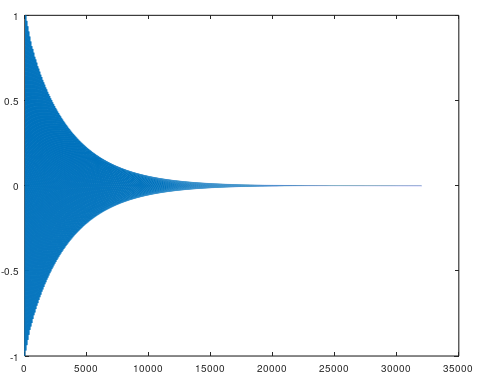
**• Estudio del algoritmo de Karplus-Strong**

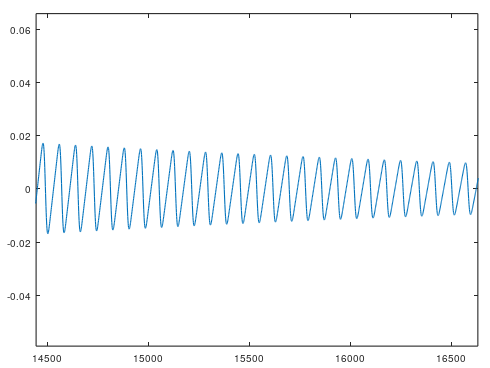
***y* [*n*]=*α*2*y* [*n*−*L*]+*α*2*y* [*n*−(*L*+1)]+*x* [*n*]**

****

Se trata de un algoritmo recursivo que asemeja un rever, pero que nos permite simular la pulsación de la cuerda de un instrumento de cuerda.

| %Preparamos el entorno clear all close all clc  %Dada la siguiente ECL %y[n]=(alfa/2) y[n-L] + (alfa/2) y[n-(L+1)]+x[n]  Fs=16000; %Frecuencia muestreo en Hz FD=200;%Frecuencia del tono a generar T=2;%Tiempo de la onda en segundos  L=floor(Fs/FD); alfa=0.98;  M=Fs\*T;%Muestras en total n=0:M-1; x=Rampa(n,L); %x=randn(M,1)\*(n'<L); y=zeros(M,1);  a=zeros(L+2,1); a(1)=1; b=1; a(L+1)=-alfa/2; a(L+2)=-alfa/2; y=filter(b,a,x); y/=max(y); plot(y);  audiowrite('Ks.wav',y,Fs); info=audioinfo('Ks.wav'); [Ks FsKs]= audioread('Ks.wav'); playerKs = audioplayer (Ks, FsKs); play(playerKs); |
| --- |

****

****