PRÁCTICA 1: MUESTREO Y RECONSTRUCCIÓN.

Resumen. En esta práctica se estudiarán las etapas para convertir una señal analógica en digital y a la inversa (conversión D/A y A/D). Veremos cada uno de los problemas que nos encontraremos en estos procesos y las diferentes aproximaciones a seguir para solucionarlos. Hay que destacar la importancia que tiene el Teorema de Muestreo.

En primer lugar hay que tener cuidado con una cuestión fundamental: MATLAB trabaja con datos discretos; puede parecer un poco ilógico usar esta herramienta para explicar una práctica de muestreo (¡¡los datos ya están muestreados!!). La solución adoptada es considerar intervalos de tiempo muy pequeños frente al periodo de muestreo de tal forma que tendremos una señal "quasi-continua". El siguiente programa determina una señal continua de frecuencia dada y su versión muestreada:

```
t = 0:0.0001:1;
f=input('frecuencia de la señal ');
xa = cos(2*pi*f*t);
subplot(2,1,1)
plot(t,xa);grid
xlabel('Tiempo, ms');
ylabel('Amplitud');
title('Señal continua x_{a}(t)');
axis([0 1 -1.1 1.1])
subplot(2,1,2);
T = 0.1;
n = 0:T:1;
xs = cos(2*pi*f*n);
k = 0:length(n)-1;
stem(k,xs);grid;
xlabel('Índice de muestreo');
ylabel('Amplitud');
title('Señal discreta x[n]');
axis([0 (length(n)-1) -1.1 1.1])
```

- Como puedes comprobar el programa es bastante incómodo a la hora de cambiar los parámetros, haz las debidas modificaciones para que se puedan variar la frecuencia de la señal y el periodo de muestreo.

- Una vez que se tiene la forma de generar señales continuas y discretas se puede evaluar el Teorema de Muestreo; para ello varía la frecuencia de la señal continua y el periodo de muestro, ¿qué ocurre cuando se cumple la igualdad entre la frecuencia de la señal y la mitad de la frecuencia de muestreo?.

- Tomando como base el programa anterior desarrolla un fichero que te permita representar, en la misma gráfica, dos señales continuas (diferentes frecuencias) y sus correspondientes muestreadas. Haz que las frecuencias cumplan la relación $f_1 = f_0 + k \cdot f_m$ siendo k un número entero positivo y f_m la frecuencia de muestreo, ¿qué ocurre?.

- Otra forma de ver el Teorema de Muestreo es observar el dominio de la frecuencia. Determinaremos el espectro de una sinusoide (aparece representado como una línea) aumentando su frecuencia. Veremos que conforme aumenta ésta, aumenta la distancia al origen de la línea (¡lógico estamos aumentando la frecuencia!) hasta que.....Ejecuta el siguiente programa y observa lo que ocurre.

```
clear
clc
clf
f=5;
fm=50;
M=moviein(20);

for s=1:20,
f=f+5;
x=cos(2*pi*(f/fm)*(0:99));
g=abs(fft(x));
t=linspace(0,fm/2,51);
plot(t,g(1:51));
axis([0 fm/2 0 51])
xlabel('frecuencia ');
ylabel('Magnitud espectral ');
M(:,s)=getframe;
end
movie(M,2,0.5)
```

  Si al ejecutarlo ocurre demasiado rápido o consideras que lo has visto pocas veces modifica los parámetros de la instrucción *movie* que aparece al final.

- El siguiente ejemplo es igual que el anterior pero "un poco más elegante": usaremos el espectrograma de una señal chirp. Un espectrograma se puede definir como la representación temporal de la transformada de Fourier. Cuando se estudie la transformada de Fourier se verá que la condición para determinarla es que las características de la señal no varíen con el tiempo. Cuando esto no ocurre se utiliza el espectrograma para analizar frecuencialmente la señal. Esta situación la tenemos con la señal chirp. La expresión de esta señal es:

$$c(t) = A \cdot \cos\left(\pi \cdot \mu \cdot t^2 + 2 \cdot \pi \cdot f \cdot t + \psi\right) = A \cdot \cos\left(\Omega(t)\right)$$

  Vemos que en esta expresión la frecuencia de la señal varía con el tiempo. Podemos definir la frecuencia instantánea como:

$$f_{inst} = \frac{1}{2 \cdot \pi} \cdot \frac{d\Omega(t)}{dt} = \mu \cdot t + f$$

  Vemos que en este tipo de señal la frecuencia cambia con el tiempo linealmente. El siguiente programa implementa esta función; los parámetros a pasar son la frecuencia inicial, frecuencia final, frecuencia de muestreo y tiempo de adquisición. Muestra la señal continua (más clarificadora) y el espectrograma. Interpreta los resultados obtenidos según se tenga o no solapamiento (*aliasing*).

```
close all
clc

tfinal=input('Duración de la adquisición ');
f1=input('Frecuencia inicial ');
f2=input('Frecuencia final ');
fm=input('Frecuencia de muestreo ');

f=f1;
mu=(f2-f1)/tfinal;

TM=1/fm;
tcont=0.01*TM;
t = 0:tcont:tfinal;
xa=cos(pi*mu*(t.^2)+2*pi*f*t);
plot(t,xa);grid
xlabel('Tiempo');
ylabel('Amplitud');
title('Señal continua x_{a}(t)');
axis([0 1 -1.1 1.1])
zoom
n = 0:TM:tfinal;
xs = cos(pi*mu*(n.^2)+2*pi*f*n);
figure
specgram(xs)
title('Espectrograma de la señal muestreada ')
```

- Una vez que se ha visto el muestreo pasemos a la siguiente etapa en la conversión A/D: la cuantización. La siguiente función implementa un cuantizador bipolar en el que se pueda escoger el valor máximo de dicho conversor así como el número de bits. Comprueba el funcionamiento de dicho cuantizador observando el aspecto de una sinusoide cuando se cuantiza con diferente número de bits; comprueba lo que ocurre cuando el rango dinámico del conversor es menor que la amplitud de la señal.

```
function xc=cuant_1(N,v,x);

close all
numniv=(2^N)-1;
res=v/(2^(N-1));
xx=x+v;
tem=round(xx/res);
[a,b]=find(tem>numniv);
tem(b)=(numniv)*ones(size(b));
[aa,bb]=find(tem<0);
tem(bb)=zeros(size(bb));
xc=tem*res-v;
plot(x);
hold on
plot(xc,'.-r')
```

- El siguiente programa comprueba la regla que afirma que la relación señal-ruido de cuantización sigue la regla de 6B siendo B el número de bits del conversor. Para comprobarla se ha utilizado una señal aleatoria que sigue una distribución normal de media cero y varianza (energía) unidad. Considera un conversor bipolar de rango máximo 5 voltios. Al final se muestra el ajuste por mínimos cuadrados a una recta. El primer valor es la pendiente de dicha recta.

```
clear
v=input('Rango del conversor bipolar ');
close all

for N=2:16,

x=randn(1,5000);
numniv=(2^N)-1;
res=v/(2^(N-1));
xx=x+v;
tem=round(xx/res);
[a,b]=find(tem>numniv);
tem(b)=(numniv)*ones(size(b));
[aa,bb]=find(tem<0);
tem(bb)=zeros(size(bb));
xc=tem*res-v;
er=x-xc;
ener=cov(er);
relac(N-1)=10*log10(1/ener);

end

plot(relac)
grid
zoom
[a,b]=polyfit(2:16,relac,1)
```

¿Qué ocurre si cuantizamos otro tipo de señal?, ¿se sigue utilizando esta regla?. ¿Qué ocurre si utilizamos otro valor máximo del cuantizador?.

Hasta aquí hemos cubierto la primera parte de una conversión A/D "normal"; no hemos implementado ni modificaciones de la frecuencia de muestreo (interpolación y decimación) y hemos considerado una cuantización uniforme (existen otros esquemas de paso de cuantización no uniforme). No nos preocuparemos de la codificación pues se estudia en otras asignaturas. Pasemos ahora a la reconstrucción.....

- Una vez que se tiene la señal muestreada hay que reconstruirla para lo que se utiliza la función sinc(x) que es la función seno(x) dividida por el valor de x. Al avanzar más en el procesado de la señal (nosotros lo haremos) aparece de forma natural: la función sinc(x), de longitud infinita, genera un filtro paso-bajo ideal. Lógicamente

no podemos usar algo infinito por lo que nos contentaremos con una versión "truncada". Recordemos que la reconstrucción a aplicar es la siguiente:

$$x(t) = \sum_{n=-\infty}^{n=\infty} x(n) \cdot \frac{\sin\left(\pi \cdot \left(t - n \cdot T\right)/T\right)}{\pi \cdot \left(t - n \cdot T\right)/T}$$

El siguiente programa realiza esta reconstrucción:

```
clf;
T = 0.1;f = 9;
nn=0:0.0001:1;
xc=cos(2*pi*f*nn);
n = (0:T:1)';
xs = cos(2*pi*f*n);
t = linspace(0,1,1/0.0001)';
ya = sinc((1/T)*t(:,ones(size(n))) - (1/T)*n(:,ones(size(t)))')*xs;
plot(n,xs,'or',t,ya,'b',nn,xc,'.-b');grid;
legend('Muestras','Reconstruida','Original');
xlabel('Tiempo');ylabel('Amplitud');
axis([0  1  -1.9 1.9]);
```

Usa este programa para ver la reconstrucción de señales muestreadas correctamente e incorrectamente. Utiliza otras funciones (por ejemplo combina dos sinusoides).

Este es el reconstructor "ideal"; si os fijáis en la expresión que define la señal continua es necesario conocer TODAS las muestras para ir determinando la señal continua. ¿Qué ocurre cuando se trabaja en tiempo real?, en esta situación la solución pasa por usar los reconstructores de orden 0 y 1 (sin retardo y con él).
El siguiente programa simula un reconstructor de orden cero. Se puede escoger tanto la frecuencia de muestreo como la frecuencia de la señal a muestrear.

```
clear
close all
fm=input('frecuencia de muestreo ');
T=1/fm;
tc=0.01*T;
f=input('Frecuencia de la señal a muestrear ');
xm=cos(2*pi*f*(0:T:1));
xc=cos(2*pi*f*(0:tc:1));
l=length(0:T:1);
M=ones(100,1);
xtemp=M*xm;
xrec=reshape(xtemp(:,1:end-1),[1,length(0:tc:1)-1]);
plot(0:tc:1-tc,xc(1:end-1),0:T:1,xm,'ok',0:tc:1-tc,xrec,'r')
```

Se comprueba que el resultado no es para tirar cohetes por lo que utilizaremos el siguiente reconstructor que es el de orden 1. El siguiente programa implementa este elemento.

```
function rec_1(fm,f)

T=1/fm;
tc=0.01*T;
xc=cos(2*pi*f*(0:tc:1));
xd=cos(2*pi*f*(0:T:1));
L1=length(0:T:1);
tt=0:tc:T-tc;
L2=length(tt);
xrec=zeros(1,length(xc));

for n=2:L1-1,

pend=fm*(xd(n)-xd(n-1));
tt=(n-1)*T:tc:n*T-tc;
L=length(tt);
xrec((n-1)*L2+1:n*L2)=pend*(tt-(n-1)*T)+xd(n);

end

plot(0:tc:1,xc,'r',0:T:1,xd,'or',0:tc:1,xrec,'g')
```

Se puede observar que cualquier parecido con la realidad es coincidencia. El siguiente reconstructor es de primer orden también pero ahora con retardo.

```
function rec_2(fm,f)

T=1/fm;
tc=0.01*T;
xc=cos(2*pi*f*(0:tc:1));
xd=cos(2*pi*f*(0:T:1));
stem(0:T:1,xd);
hold on;
xrec=[0 xd];
plot(0:tc:1,xc,'g',T:T:1,xrec(2:end-1),'k')
grid
```

Fíjate cómo reconstruye este sistema conforme se aumenta la frecuencia de muestreo.