Señales y Sistemas

Práctica 4

Conversión A/D y D/A

ÍNDICE

1. CONVERSIÓN ANALÓGICA-DIGITAL

En primer lugar, veamos cada una de las limitaciones que nos encontramos en la conversión A/D.

Antes de empezar 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*". Esto ya se tuvo en cuenta en prácticas anteriores, cuando se utilizó Matlab para representar señales en tiempo continuo.

Para diferenciar entre señales continuas y discretas, emplearemos distintos modos de representación según las señales sean de un tipo u otro. Para representar señales continuas emplearemos la función `plot`, y para representar señales discretas emplearemos la función `stem`. Para comprobar cómo representa las secuencias esta última función, ejecuta las siguientes órdenes en Matlab:

```
>> n=0:19;
>> x=cos(0.125*pi*n);
>> stem(n,x)
```

1.1. Estudio del muestreo en el dominio del tiempo

A continuación, crea la siguiente función de extensión M que proporciona dos periodos de una sinusoide continua de frecuencia dada y su versión muestreada:

```
function [xa,xs]=muestreosinu(f0,Ts)
% f0 es la frecuencia de la sinusoide continua en Hz
% Ts es el periodo de muestreo en segundos
% xa contiene los valores de la señal continua
% xs contiene los valores de la señal discreta (muestras)
T0 = 1/f0;
t = [0:2*T0/10000:2*T0];
xa = cos(2*pi*f0*t);
subplot(2,1,1)
plot(t, xa)
grid
xlabel('Tiempo continuo (s)','Fontsize',8)
ylabel('Amplitud','Fontsize',8)
title('Señal continua x_a(t)')
axis([0 2*T0 -1.5 1.5]);
nTs = [0:Ts:2*T0];
xs = cos(2*pi*f0*nTs);
n = 0:length(nTs)-1;
subplot(2,1,2)
stem(n, xs)
grid
textox=['Tiempo discreto (n). T_s=',num2str(Ts*1000),' ms'];
xlabel(textox,'Fontsize',8)
ylabel('Amplitud','Fontsize',8)
title('Señal discreta x[n]=x_a(nT_s)')
axis([0 2*T0/Ts -1.5 1.5])
```

Vamos a utilizar ahora la función para evaluar el Teorema de Muestreo de Nyquist:

- Considera un periodo de muestreo de valor `Ts = 0.0001`. Varía la frecuencia de la señal sinusoidal continua y escoge:
    a. Un caso donde se cumpla el Teorema de Muestreo. Observa el resultado.
    b. Un caso donde no se cumpla, es decir, donde se esté produciendo *aliasing*. Observa el resultado.
- ¿Cuántas muestras por ciclo se toman cuando la relación entre periodo de muestreo y frecuencia de la señal es la correspondiente al límite para asegurar la reconstrucción según el Teorema de Muestreo de Nyquist?

1.2. Cuantificación

Una vez que se ha visto el muestreo, pasemos a la siguiente etapa en la conversión A/D: la cuantificación. Comprobarás lo que ocurre cuando el margen dinámico del conversor no se adecúa a la excursión de la señal.

Para obtener la secuencia cuantificada, $x_q[n]$, a partir de la secuencia de muestras originales, $x[n]$, vamos a usar un cuantificador uniforme con la siguiente función característica:

$$x_q[n] = Q(x[n]) = \begin{cases} \left(E\left[\frac{|x[n]|}{\Delta}\right] + \frac{1}{2}\right) \cdot \Delta \cdot \text{sign}(x[n]), & |x[n]| < x_{máx} \\ \frac{L-1}{2} \cdot \Delta \cdot \text{sign}(x[n]), & |x[n]| \geq x_{máx} \end{cases}$$

donde $E[\,]$ es la función parte entera (**`fix()`** en Matlab), Δ es el tamaño del escalón de cuantificación, $\text{sign}()$ es la función signo y L es el número de niveles de cuantificación.

- Programa en Matlab una <u>función</u> `cuantifica.m` que cuantifique de modo uniforme de acuerdo con la función $x_q[n] = Q(x[n])$ especificada. Las <u>variables de entrada</u> han de ser un vector `x` con los valores de la secuencia de muestras originales, el valor `xmax` y el número de bits de cuantificación `b`. La <u>variable de salida</u> ha de ser un vector `xq` con la secuencia de muestras cuantificadas. La función tiene que <u>representar</u> la secuencia original, la secuencia cuantificada y el error de cuantificación. Las secuencias han de estar todas en una misma ventana, pero sin superponerse en los mismos ejes.

- Genera en Matlab un vector con $N = 200$ valores de la siguiente secuencia de muestras originales,
$$x[n] = \text{sen}(2\pi f_0 n),$$
con $f_0 = {}^{1}/_{50}$.
- Utiliza la función `cuantifica` para cuantificar la señal $x[n]$ empleando $L = 8$ niveles de cuantificación y un valor $x_{máx} = \text{máx}\{x[n]\} = 1$ (caso óptimo).
- Comprueba qué ocurre cuando el margen dinámico del conversor es menor que la excursión de la señal.
- Comprueba qué ocurre cuando el margen dinámico del conversor es mucho mayor que la excursión de la señal.

Hasta aquí hemos cubierto las dos primeras partes de una conversión A/D: muestreo y cuantificación. Pasemos ahora a estudiar el proceso inverso: la reconstrucción de la señal analógica a partir de la secuencia de muestras.

2. CONVERSIÓN DIGITAL-ANALÓGICA

2.1. Reconstrucción ideal

Una vez que se tiene la señal muestreada hay que reconstruirla para lo que se utiliza la función sinc de longitud infinita. Lógicamente no podemos usar algo infinito, por lo que nos contentaremos con una versión truncada de esta función. Recordemos que la reconstrucción a aplicar (fórmula de interpolación ideal) es la siguiente:

$$xr(t) = \sum_{n=-\infty}^{+\infty} x[n] \cdot \text{sinc}\left(\frac{1}{T_s}(t - nT_s)\right)$$

El siguiente programa realiza la reconstrucción de una sinusoide, en un intervalo de 1 segundo, empleando funciones sinc:

```
Ts = 0.1;
f0 = 9;
ta = [0:0.0001:1];
xa = cos(2*pi*f0*ta);
nTs = (0:Ts:1).';
xd = cos(2*pi*f0*nTs);
xr = zeros(size(ta));
for m=1:length(nTs)
    xr = xr + xd(m).*sinc((ta-nTs(m))/Ts);
end
plot(ta,xa,'.-g',nTs,xd,'or',ta,xr,'b')
grid
legend('Original','Muestras','Reconstruida')
xlabel('Tiempo (s)')
```

```
ylabel('Amplitud')
axis([0 1 -1.9 1.9])
```

- Modifica el valor de `Ts` con respecto a `f0` en la primera línea para observar la reconstrucción de la señal sinusoidal muestreada correctamente e incorrectamente.

Observa que una de las limitaciones del reconstructor ideal es que, de acuerdo con la expresión de interpolación, es necesario conocer TODAS LAS MUESTRAS para ir determinando la señal continua. Dicho de otro modo, para reconstruir el intervalo de señal entre, por ejemplo, sólo dos muestras, es necesario conocer el valor de todas las muestras.

Observa además que para una señal sinusoidal, que es de **duración infinita**, se necesitan idealmente un NÚMERO INFINITO DE MUESTRAS. En el programa se emplea una cantidad finita, se ha programado en realidad la siguiente ecuación:

$$\widehat{xr}(t) = \sum_{n=0}^{N-1} x[n] \cdot \mathrm{sinc}\left(\frac{1}{T_s}(t - nT_s)\right) \neq xr(t)$$

Fíjate que la diferencia entre la señal continua original $x(t) \equiv xr(t)$ y la señal $\widehat{xr}(t)$ es más significativa en los extremos del intervalo de tiempo considerado.

¿Qué se puede hacer entonces cuando se trabaja en tiempo real y no se conocen a priori todas las muestras de la señal? Veamos una solución sencilla en el siguiente apartado.

2.2. Reconstrucción práctica

El siguiente programa simula el funcionamiento de un sistema de interpolación muy sencillo, se trata de un reconstructor de primer orden con retardo. Razona cómo funciona este sistema a partir del código de simulación siguiente:

```
function []=rec(fs, f)
Ts=1/fs;
tc=0.001*Ts;
xc=cos(2*pi*f*(0:tc:1));
xd=cos(2*pi*f*(0:Ts:1));
stem(0:Ts:1,xd)
hold on
xrec=[0 xd];
plot(0:tc:1,xc,'g',Ts:Ts:1,xrec(2:end-1),'k'), grid
hold off
```

- Prueba para empezar con este ejemplo:

  ```
  >> rec(25,5)
  ```

- Fíjate cómo mejora la calidad de la reconstrucción de este sistema conforme se aumenta fs.
- ¿Qué condición se debe imponer en el muestreo para poder utilizar este reconstructor tan sencillo en la conversión D/A?

2.3. Alteración de la frecuencia de muestreo en reconstrucción

Considera 8 señales sinusoidales reales continuas de amplitud 1 y fase inicial 0 radianes, cuyas frecuencias son:

$$f_{01} = 550 \text{ Hz},$$
$$f_{02} = f_{01} \cdot (9/8) \text{ Hz},$$
$$f_{03} = f_{02} \cdot (10/9) \text{ Hz},$$
$$f_{04} = f_{03} \cdot (16/15) \text{ Hz},$$
$$f_{05} = f_{04} \cdot (9/8) \text{ Hz},$$
$$f_{06} = f_{05} \cdot (10/9) \text{ Hz},$$
$$f_{07} = f_{06} \cdot (9/8) \text{ Hz},$$
$$f_{08} = f_{07} \cdot (16/15) \text{ Hz}.$$

- Crea un fichero tipo M para calcular $N = 5000$ muestras de cada sinusoide suponiendo que se emplea una frecuencia de muestreo $f_S = 11025$ Hz.
- Concatena las ocho sinusoides en un único vector x, en orden creciente de frecuencia.
- Escucha la secuencia generada, con 4 y 16 bits por muestra. Para ello utiliza la función `sound` de Matlab (o bien guarda la secuencia en un fichero WAV con `wavwrite`, y reprodúcelo):

```
>> sound(x,11025,4)
>> sound(x,11025,16)
```

- ¿Notas alguna diferencia de calidad entre la secuencia cuantificada con 4 bits y la cuantificada con 16 bits? ¿Cuál es la razón?
- Por último vamos a observar qué ocurre si modificamos la frecuencia de muestreo (inversa del tiempo entre muestras) en reconstrucción. Para ello, escucha de nuevo la secuencia a 16 bits, pero empleando de forma sucesiva las frecuencias 13, 15 y 18 KHz. ¿Cuál es el efecto observado? ¿A qué se debe?
- ¿Qué sucede cuando se emplea una frecuencia en reconstrucción más baja que la frecuencia de muestreo nominal? Compruébalo escuchando la secuencia con 16 bits y una frecuencia de 8 KHz.