PRÁCTICA 2

REALIZACIÓN DE SISTEMAS DE CONTROL DISCRETOS

1. Introducción

El objetivo de esta práctica es estudiar los efectos de precisión finita en los sistemas digitales mediante el uso de la herramienta MATLAB.

MATLAB dispone de herramientas que permiten crear y operar con datos en coma fija. Además de operaciones aritméticas y lógicas, se permiten operaciones como el escalado, el redondeo y tratar el *overflow*.

Todas estas funciones y herramientas se encuentran incluidas en el Toolbox de MATLAB "Fixed-Point Toolbox". En esta práctica se va a presentar algunas de estas funciones con el objetivo de trabajar de una forma práctica algunos de los conceptos vistos en la asignatura.

2. Conversión de modelos LTI

MATLAB dispone de funciones para realizar conversiones entre el modo continuo y el discreto con diferentes métodos de discretización.

La función que realiza la conversión del modo continuo al discreto es c2d y permite discretizar por los métodos de aproximación de la respuesta temporal: invariante al impulso y al escalón (Zero-Order Hold), Transformación Bilineal (Método de Tustin) sin y con pre-escalado en frecuencia, y el método de emparejamiento de polos y ceros:

```
>> sys_disc=c2d(sys_cont,Ts,'tipo_discretizacion',wc);
        % tipo_discretización:
        %       Por defecto o 'zoh': Invariante al escalón (ZOH)
        %       'imp': Invariante al impulso
        %       'tustin': Transformación bilineal
        %       'prewarp': Trans. Bilineal con pre-escalado (wc en rad/s)
        %       'matched': Emparejamiento de polos y ceros
```

Para obtener el modelo continuo a partir del discreto se utiliza la función d2c (con las mismas opciones).

Ejercicio 2.1

[Laboratorio] Discretiza el sistema H(s) del Ejercicio 1.1 de la Práctica 1 mediante el método invariante al escalón (ZOH), ya que es el indicado en el diagrama de bloques (DAC más sistema continuo H(s)).

Obtén y representa la respuesta en frecuencia del sistema.

Ejercicio 2.2

[Laboratorio] Discretiza la misma función del Ejercicio 2.1 mediante el método de Tustin.

Obtén y representa la respuesta en frecuencia del sistema y compáralo con los resultados obtenidos con los del ejercicio anterior.

Ejercicio 2.3

[Laboratorio] Para un sistema como el del Ejercicio 1.1 de la Práctica 1:

1. Discretiza la función de transferencia H(s) mediante el método de la transformación bilineal:

$$H(s) = k\frac{1}{s(s+1)}$$

2. Suponiendo el sistema totalmente discreto, obtén $G_D(z)\cdot H(z)$ (H(s) discretizada) y Y(z)/X(z) (función de transferencia en lazo cerrado), pero con:

$$G_D(z) = \frac{2.3798z - 1.9387}{z - 0.5589}$$

Estudia la función de transferencia en lazo cerrado para valores de k=1,2,3 y 4.

Compara las funciones de transferencia de $G_D(z)$, $G_D(z)\cdot H(z)$ y Y(z)/X(z).

Utiliza la aplicación LTI Viewer para ver y analizar los resultados.

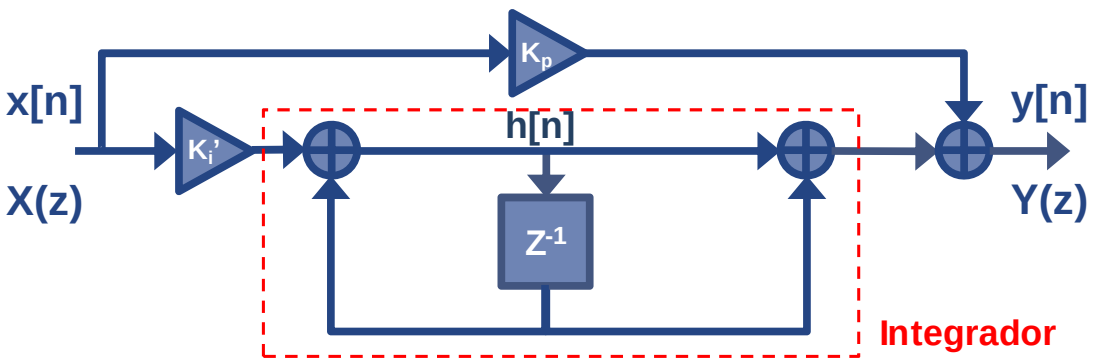
Visualiza también la respuesta al escalón del sistema.

3. Programación de redes digitales

El proceso de implementación de una red digital implica una serie de pasos, entre los que está el diseño de la red, que implica la obtención de la función de transferencia, y la elección del método de programación de la red.

Una vez realizados estos pasos, y seleccionado el hardware, hay que implementarlo. Para ello, hay que descomponer la red en ecuaciones que puedan ser programadas por la herramienta de programación adecuada al hardware, o en caso de estar simulando, a MATLAB.

Veamos la siguiente red digital:



De este circuito podemos extraer las siguientes funciones:

$$h[n] = k_i' x[n] + h[n-1]$$
$$y[n] = k_p x[n] + h[n] + h[n-1]$$

Estas ecuaciones se han extraído directamente del circuito. Para programar estas ecuaciones en MATLAB, tendremos además que suponer unas condiciones iniciales para nuestro circuito, en este caso vamos a suponer que h[0] y valores anteriores son iguales a 0. También, y dado que vamos a analizar nuestro circuito, vamos a estudiar la salida para una entrada escalón de un número determinado de muestras.

Hay que tener en cuenta, que tanto en MATLAB como para un DSP, las operaciones deben ejecutarse de forma secuencial, por lo que el orden de ejecución de las diferentes sentencias extraídas a partir del circuito es importante.

En MATLAB tendríamos, por ejemplo, la siguiente cadena de sentencias:

```
% Entrada escalón por 0.5, y 40 muestras
x=0.5*ones(40,1);

% Declaración de constantes
ki=0.036075;
kp=229;

% Declaración de vectores para las señales 'y' y 'h'
y=zeros(size(x));
h=zeros(size(x));                % h[n] y h[n-1]

% Inicialización
h(1)=ki*x(1);
y(1)=kp*x(1) + h(1);

% Evaluación de la salida para 40 muestras
for k=2:length(x)
        h(k)=ki*x(k) + h(k-1);
        y(k)=kp*x(k) + h(k) + h(k-1);
end
```

Simplemente hay que tener en cuenta que en MATLAB el primer índice es '1', por lo que valores de la respuesta al impulso anteriores a h(1) son cero, tal y como habíamos supuesto más arriba, y se puede apreciar en la parte de inicialización del código.

A la hora de estudiar los resultados hay que tener en cuenta que MATLAB utiliza distintos formatos de datos, por ejemplo, el tipo 'long' es un formato en coma fija de 15 dígitos. Por ejemplo, el número π se representaría por, 3.14159265358979.

Ejercicio 2.4

[Trabajo previo] Diseña un filtro paso-bajo de Butterworth de orden 2 con frecuencia de corte de 2 KHz y frecuencia de muestreo de 12 KHz.

Obtén el filtro de dos maneras distintas, utilizando la herramienta FDAtool y funciones para el diseño de filtros a través de la ventana de comandos (ver Anexo).

[Laboratorio] Obtén la red digital necesaria para implementar el filtro en su forma directa II y transpuesta.

Ejercicio 2.5

[Laboratorio] Obtén las ecuaciones para la forma directa II.

Ejercicio 2.6

[Laboratorio] Obtén la respuesta del filtro para una entrada escalón y 50 muestras. Los valores iniciales del sistema son cero.

4. Efectos del redondeo de los coeficientes

En MATLAB se utilizan formatos de datos del tipo ANSI C para representar los datos. Los rangos de estos tipos de datos no son iguales que los rangos de los números representados en Complemento a 2 o con un formato de coma flotante de 32 bit, aunque este último se puede aproximar bastante. Por ejemplo, con el tipo 'long', el rango de representación es [-2147483647, 2147483647], que incluso en inferior al rango de representación de coma flotante de 32 bit.

Para cuantificar y así representar los datos (coeficientes, por ejemplo) podemos utilizar la función fi. Con esta función se pueden crear tipos de datos cuyos rangos de representación son adecuados para representar números en Complemento a 2, además de indicar el número de bits que utilizamos para dicha representación.

```
>> v_cuant=fi(v,s,w,f,'propiedades')
        % v_cuant: Valor cuantificado
        % v: Valor a cuantificar
        % s: Signo (1), sin signo (0)
        % w: Número de bits
        % f: Tamaño de la parte decimal
```

Algunas propiedades interesantes son:

- SumMode y ProductoMode:
    - FullPrecision: El tamaño del producto es la suma de los tamaños de palabras de los dos operandos. Para la suma, el tamaño crece un bit en la parte más significativa para acomodar un posible bit de acarreo (operaciones sin signo).
    - KeepMSB: En este modo se especifica el tamaño del producto y de la suma. Este parámetro permite modelar el comportamiento de la mayoría de DSPs donde el resultado del producto y de la suma son almacenados en dos registros

del tamaño de los operandos, y dónde el diseñador escoge transferir sólo la parte alta (bits más significativos).

- ProductWordLength: Tamaño de palabra para los productos (de 2 a 128).
- SumWordLength: Tamaño de palabra para las sumas (de 2 a 128).
- OverflowMode: Tipo de tratamiento del overflow (Saturate, Wrap).
- RoundMode: Tipo de redondeo (ceil, convergent, fix, floor, round).
- CastBeforeSum: Si es verdadero, los operandos se cuantifican antes al formato especificado para la suma, de forma que no hay que hacer desplazamientos para alinear la 'coma'. Si es falso, los operandos se suman y luego son cuantificados.

Ejemplo:

```
>> pi_cuant = fi(pi,0,16,7, 'RoundMode', 'floor', 'OverflowMode', 'wrap')

>> pi

ans = 3.14159265358979

>>pi_cuant=fi(pi,0,16,7,'RoundMode','floor','OverflowMode','saturate',
                'ProductWordLength',32,'SumWordLength',16,
                'SumMode','KeepMSB')

pi_cuant = 3.14062500000000

    DataTypeMode: Fixed-point: binary point scaling
    Signed: false
    WordLength: 16
    FractionLength: 7
    RoundMode: floor
    OverflowMode: saturate
    ProductMode: KeepMSB
    ProductWordLength: 32
    SumMode: KeepMSB
    SumWordLength: 16
    CastBeforeSum: true
```

El mismo valor "pi" sin cuantificar, en coma flotante de 32 bit es 3.1415927, con una precisión del al menos 6 decimales, mientras que con un formato Q de 32 bit, se obtiene un valor de 3.1416015625, con una precisión de 3 decimales. Una manera sencilla de obtener los valores en coma flotante, es utilizar la siguiente página web:

http://babbage.cs.qc.edu/IEEE-754/Decimal.html

El valor decimal de un dato en formato en coma flotante de 32 bits se puede obtener en Matlab mediante el uso de la función single.

Siguiendo con las variables cuantificadas con la función fi, los objetos creados al aplicar esta función no se pueden utilizar en cualquier función de MATLAB. Si se quieren utilizar sólo los datos cuantificados:

```
>> v_datac=v_cuant.data;
```

Algunas funciones muy útiles a la hora de trabajar con este tipo de objetos son hex y bin, que permiten obtener el valor del dato cuantificado en hexadecimal o en binario respectivamente. Para el ejemplo anterior:

```
>> pi_cuanth=hex(pi_cuant)
   pi_cuanth = 0192
```

Estas funciones son muy útiles a la hora de trasladar los resultados de las simulaciones en MATLAB a la implementación con DSPs.

Otras funciones interesantes son: realmax, realmin.

Ejercicio 2.7

[Laboratorio] Cuantifica los coeficientes de los filtros del ejercicio 2.4. Utiliza el formato de datos adecuado (representación de 16 bits).

[Laboratorio] Visualiza la magnitud y la fase del filtro, el diagrama de polos y ceros, y compáralos con los obtenidos en el ejercicio 2.4.

5. Análisis del ruido de cuantificación

En la sección anterior sólo se ha estudiado el efecto de la cuantificación de los coeficientes. Sin embargo, otro análisis importante es el que resulta de estudiar el ruido que introducen los redondeos tras los productos.

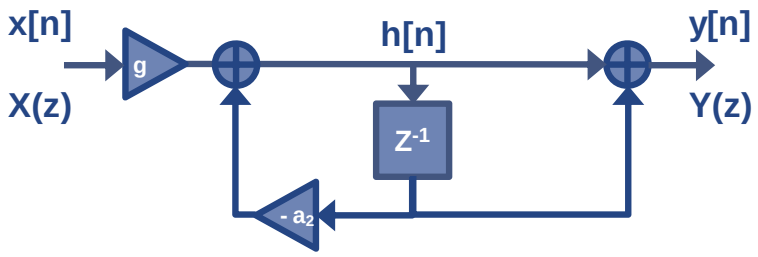
Cuando se realizan operaciones sabemos que:

- Los sumadores no introducen ruido debido a que la suma de dos valores cuantificados da otro valor cuantificado.
- En los sumadores se puede producir *overflow*, pero podemos evitarlo escalando bien la señal.

5.1. Escalado del sistema

Como ya se ha visto, es importante conocer la amplitud máxima de la señal de entrada para que no haya *overflow* en ningún nodo.

Veamos un ejemplo para un filtro IIR como el de los ejercicios de la sección anterior:



La función de transferencia a la salida es:

$$H(z)=\frac{Y(z)}{X(z)}=0.36602783203125\frac{1+z^{-1}}{1-0.2679443359375\,z^{-1}}$$

Los coeficientes ya están cuantificados en formato Q_{14}.

La amplitud máxima de la señal de entrada debe cumplir la siguiente ecuación:

$$|x[n]|<x_{max} \quad \Rightarrow \quad x_{max}<\frac{y_{max}}{\sum_k |h[k]|} \quad \text{para } |y[n]|<y_{max} \tag{2.1}$$

Suponiendo que la salida está codificada en Q_{14}, debe ser menor que 2, por lo que tenemos que $y_{max}=2$.

A partir de la función residue podemos descomponer la función y obtener la respuesta al impulso analíticamente a partir de tablas:

$$H(z)=\frac{Y(z)}{X(z)}=-1.36605922551253+\frac{1.73208705754378}{1-0.26794433593750\,z^{-1}}$$

$$h[n]=-1.36605922551253\delta[n]+1.73208705754378\cdot 0.26794433593750^{n}u[n]$$

La respuesta al impulso también podemos obtenerla en MATLAB mediante la función filter, tal y como vimos en la Práctica 1. Este procedimiento es más rápido y sencillo.

Ahora podemos calcular la amplitud máxima de la entrada para, por ejemplo, 50 muestras, aplicando la ecuación (2.1):

$$|x[n]|<2$$

Otra consideración, menos restrictiva, es considerar que la energía total en cada nodo es menor que la energía total de la secuencia de entrada. Para asegurar esto, se puede escalar la variable de entrada por un factor 's' según las fórmulas siguientes:

$$|x[n]|<s \quad \Rightarrow \quad s^2<\frac{y_{max}^{\,2}}{\sum_k |h[k]|^2} \quad \text{para } |y[n]|<y_{max} \tag{2.2}$$

A partir de esta ecuación se obtiene que:

$$|x[n]| < 2.57$$

Hay que puntualizar, que la ecuación (2.1) proporciona un método más conservador que el visto en la ecuación (2.2).

5.2. Relación señal a ruido

Para calcular la relación señal a ruido, SNR, tenemos que estudiar dónde va a haber fuentes de ruido debido al redondeo. En este caso es sencillo, ya que a la salida de cada rama de coeficientes tendremos una fuente de ruido (en este caso sólo en la rama de los coeficientes a_i). Esta fuente de ruido está a la entrada y se corresponde con la expresión:

$$\sigma_{er}^2 = \sum \sigma_i^2 = \sigma_g^2 + \sigma_{a_2}^2 = 2\frac{Q^2}{12} = 2\cdot\frac{2^{-2n}}{12}\left(V_{max} - V_{min}\right)^2 = 2\cdot\frac{2^{-2n}}{12}(4)^2 = \frac{2^{-2n+3}}{3} \qquad \begin{matrix} V_{max} \approx 2 \\ V_{min} = -2 \end{matrix} \text{ en } Q_{14}$$

La operación de multiplicación se realiza en formato Q_{14} por ser el formato en el que se han codificado los coeficientes. En el caso del ejemplo, sólo hay dos multiplicadores, por lo que sólo hay dos fuentes de ruido en la rama de entrada (rama de los polos) y ninguna en la rama de salida (rama de los ceros). También hay que tener en cuenta que el sistema que hay entre la entrada de las fuentes de ruido y la salida es:

$$H_e(z) = \frac{Y(z)}{X_e(z)} = \frac{1 + z^{-1}}{1 - 0.2679443359375\, z^{-1}}$$

Por otro lado, hay que tener en cuenta el ruido de cuantificación de la entrada, para ello vamos a suponer que la señal de entrada puede tomar valores de -2 a 2 V, valores que están:

$$\sigma_{ex}^2 = \frac{2^{-2n}}{12}\left(V_{max} - V_{min}\right)^2 = \frac{2^{-2n}}{12}(4)^2 = \frac{2^{-2(n-1)}}{3} \qquad \begin{matrix} V_{max} = 2\,V \\ V_{min} = -2\,V \end{matrix}$$

Por esta razón, la fuente de ruido a la salida es:

$$\sigma_{y\_ruido}^2 = \sigma_{ex}^2 \cdot \sum_{n=-\infty}^{\infty} |h[n]|^2 + \sigma_{er}^2 \cdot \sum_{n=-\infty}^{\infty} |h_e[n]|^2 = 1.92\cdot 10^{-9}$$

Suponiendo que la entrada es ruido blanco y está entre los márgenes de amplitud máxima que hemos calculado con anterioridad (sección 5.1), podemos obtener la potencia de la señal a la salida:

$$\sigma_x^2 = \frac{1}{12}\left(V_{max} - V_{min}\right)^2 = 1.333$$

$$\sigma_y^2 = \sigma_x^2 \cdot |H(j\omega)|^2 = \sigma_x^2 \cdot \sum_{n=-\infty}^{\infty} |h[n]|^2 = 0.488$$

A partir de la potencia de la señal de entrada y de la potencia de la señal de salida debido al ruido, se puede obtener la relación señal a ruido:

$$SNR(dB) = 10\log\left(\frac{\sigma_y^2}{\sigma_{y\_ruido}^2}\right) = 84.04$$

De estos cálculos podemos concluir que a mayor potencia de señal de entrada mayor SNR.

Ejercicio 2.8

[Laboratorio] Calcula la SNR del filtro del ejercicio 2.4. Utiliza el formato de datos adecuado (representación de 16 bits).

Ejercicio 2.9

[Laboratorio] Calcula la respuesta al escalón del sistema, luego calcula la misma respuesta para el sistema cuantificado (cuantificación de la entrada y de los coeficientes). Utiliza el formato de datos adecuado (representación de 16 bits).

[Laboratorio] Visualiza el resultado de ambas respuestas y compáralas.

[Laboratorio] Prueba con varios tipos de codificación, utiliza el mismo para todas las cuantificaciones que realices.

6. Bibliografía

[1] “Fixed-Point Toolbox 2 User’s Guide”, MATLAB

[2] “Signal Processing Toolbox 6 User’s Guide”, MATLAB

7. Anexo

MATLAB dispone de una herramienta para el diseño y análisis de filtros (FDATool). Esta herramienta permite diseñar filtros FIR e IIR introduciendo las especificaciones, importando filtros ya diseñados, e incluso añadiendo y/o moviendo los polos y los ceros.

Esta herramienta se invoca con fdatool.

Por otro lado, MATLAB también permite el diseño de filtros mediante el uso de funciones a través de la ventana de comandos.

A continuación se muestran algunos tipos de filtros IIR y las funciones que realizan todas las operaciones necesarias para su diseño digitales: Obtención del prototipo paso-bajo, transformación al dominio digital y discretización del filtro:

Tipo de Filtros	Función	Opciones
Butterworth	[b,a]=butter(n,Wn,options) [z,p,k]=butter(n,Wn,options)	n: orden del filtro Wn: Frecuencia de corte normalizada Rp: Rizado en la banda de paso en dB Rs: Atenuación en la banda atenuada en dB
Chebyshev I	[b,a]=cheby1(n,Rp,Wn,options) [z,p,k]=cheby1(n,Rp,Wn,options)	
Chebyshev II	[b,a]=cheby2(n,Rs,Wn,options) [z,p,k]=cheby2(n,Rs,Wn,options)	
Elíptico	[b,a]=ellip(n,Rp,Rs,Wn,options) [z,p,k]=ellip(n,Rp,Rs,Wn,options)	

Por defecto, el filtro es paso-bajo digital, para obtener el resto de filtros:

- Paso-alto: Añadir 'high'.
- Paso-banda: Especificar Wn como un vector de dos elementos, con las frecuencias de las bandas de paso.
- Elimina-banda: Especificar Wn como un vector de dos elementos, con las frecuencias de las bandas de paso, y añadir 'stop'.

Estas funciones utilizan la función bilinear que convierte el filtro analógico en digital mediante la Transformación Bilineal con pre-escalado de frecuencia.

La función de transferencia obtenida es del tipo:

$$H(z)=\frac{Y(z)}{X(z)}=\frac{b(1)+b(2)z^{-1}+b(3)z^{-2}+\cdots+b(p+1)z^{-p}}{a(1)+a(2)z^{-1}+a(3)z^{-2}+\cdots+a(q+1)z^{-q}} \qquad (2.3)$$

Ejemplos:

```
>> [b,a]=butter(5,0.5)              % Filtro paso-bajo de Butterworth
>> [b,a]=cheby1(3,1,[0.4 0.6])      % Filtro paso-banda de Chebyshev I
>> [b,a]=cheby2(7,60,0.6,'high')    % Filtro paso-alto de Chebyshev II
>> [b,a]=ellip(3,1,50,[0.4 0.6],'stop') % Filtro elimina-banda elíptico
```

Hay que recordar que MATLAB normaliza las frecuencias digitales a la mitad de la frecuencia de muestreo ($f_s/2$). Por lo que la frecuencia de corte normalizada será la frecuencia de corte dividida por la mitad de la frecuencia de muestreo ($Wn=f_c/(f_s/2)$).

En general, es recomendable trabajar con la forma zero-pole-gain ([z,p,k]) para evitar errores de redondeo a la hora de obtener la función de transferencia ([b,a]). Además, esta forma puede transformarse fácilmente en secciones de segundo orden mediante la función zp2sos, tal y como se muestra a continuación:

```
>> [sos,g]=zp2sos(z,p,k)       % H(z)=g·H1(z)·…·HN(z)
```

Esta función devuelve una matriz con la función de transferencia dividida en secciones de segundo orden y la ganancia (g). La matriz es de la siguiente forma:

$$\mathrm{sos} = \begin{pmatrix} b(0)_1 & b(1)_1 & b(2)_1 & a(0)_1 & a(1)_1 & a(2)_1 \\ b(0)_2 & b(1)_2 & b(2)_2 & a(0)_2 & a(1)_2 & a(2)_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b(0)_N & b(1)_N & b(2)_N & a(0)_N & a(1)_N & a(2)_N \end{pmatrix}$$

Ver también: tf2sos, sos2zp, sos2tf.