UNIVERSITAT POLITÈCNICA DE CATALUNYA

ESCOLA UNIVERSITARIA POLITÈCNICA DE VILANOVA I LA GELTRÚ

FILTRES ELECTRONICS ANALOGICS I DIGITALS (FEAD)

Eines software per al disseny de filtres

Tutorial bàsic de processament de senyal i filtres digitals amb Matlab

GRUPS: K45-S45 CURS: 2006/2007 -2

José Antonio Soria Pérez
Miguel Ángel Ruiz

UNIVERSITAT POLITÈCNICA DE CATALUNYA

1. Introducción

Hasta ahora, hemos visto que existen diversos programas diseñados específicamente para el diseño de filtros analógicos. Sin embargo, existen otros programas matemáticos que tienen un carácter más general, los cuales, disponen de librerías y funciones específicas para el diseño de filtros. Uno de ellos es *Matlab*.

Matlab es un entorno interactivo de trabajo desarrollado por la compañía *The Math Works, Inc.* que permite tratar problemas de ingeniera y científicos complejos que precisan de cálculos numéricos para su resolución. En este caso, la notación utilizada es bastante similar a la que utiliza una persona cuando trabaja sobre el papel. Además, gracias a una extensa colección de funciones (*Toolboxes*) que sirven para resolver problemas en diferentes ámbitos de la ingeniería, entre ellos: el control de procesos, el procesado de señal, la optimización, las redes neuronales y el tratamiento de imagen, entre otros.

El presente documento sirve como introducción para todos aquellos que no estén familiarizados el uso de esta herramienta, la cual, será utilizada para el diseño y desarrollo de *filtros digitales*, uno de los objetivos de la asignatura.

2. Comandos básicos y asignación de variables

MATLAB es el acrónimo utilizado para designar MATrix LABoratory. Esto se debe a que en sus orígenes esta herramienta fue creada con el objetivo de facilitar la realización de cálculos numéricos complejos mediante el uso de matrices. De aquí viene que la estructura básica utilizada para dichos cálculos sea la **matriz**. Una matriz es un conjunto de números reales formado por N filas y M columnas. Los números *escalar*es y los *vectores* son casos particulares de las matrices. Las dimensiones son (1x1) y (1xN) respectivamente. Además MATLAB utiliza un lenguaje de programación orientado al uso de funciones, es decir, que trabaja mediante la evaluación de procesos matemáticos definidos en base a valores o parámetros (constantes) y devuelve los resultados en variables de salida.

Asignación de variables y operaciones

Al iniciar el programa *Matlab*, en primer lugar nos aparece una *línea de comandos* donde hay un cursor que está a la espera de que se introduzcan entradas:

```
»
```

Para asignar valores numéricos a variables el formato que hay que utilizar es <variable> = <valor>. Por ejemplo, si se quiere realizar la asignación x=2 y z=0.5 hay que introducir:

```
» x=2

x =
    2

» z=0.5;
»
```

Como se puede observar, cuando no se incluye el signo ";" al final del comando *Matlab* indica justo a continuación la asignación que se acaba de realizar, mientras que cuando se incluye dicho signo aparece únicamente el cursor. El signo ";" por tanto, se utiliza para indicar a *Matlab* que únicamente se quiere realizar la asignación sin que aparezca por pantalla la operación realizada, y puede utilizarse tanto en la asignación como en el cálculo de funciones matemáticas

Los números complejos se introducen de manera similar como se suelen expresar en la práctica. En este sentido, se puede hacer referencia a la *variable compleja* tanto con la letra "**i**" como con la letra "**j**":

```
» nc=3+2i
nc =
  3.0000 + 2.0000i

» nc2=2+0.5j
nc2 =
  2.0000 + 0.5000i
```

A medida que se van sucediendo las asignaciones y los cálculos numéricos, *Matlab* va guardando toda esta información en memoria. Para saber que variables se encuentran en memoria hay que utilizar el comando ***who*:**

» **who**
Your variables are:
nc nc2 x z

Para realizar operaciones con las variables o simplemente con números, se utilizan los operadores habituales: **'+'** suma, **'-'** resta, **'/'** cociente, **'*'** producto, **'^'** potenciación. Además de otras funciones disponibles como sin, log, exp, etc...

» **4*3.5**
ans =
14

» **x/y**
ans =
4

» **op=nc^2**
op =
5.0000 +12.0000i

» **sin(pi/2)**
ans =
1

Como se puede observar, cuando la operación de cálculo no realiza ninguna asignación aparece una nueva variable (**ans**). *Matlab* también tiene almacenadas otras variables con valores predefinidos por defecto. Este es el caso, por ejemplo, de la variable '**pi'** (3.1416…) o el número **'e'**. Los valores de las funciones trigonométricas han de estar expresados en 'radianes' para realizar el cálculo correctamente.

Vectores y matrices

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & -4 \\ -3 & -2 & -1 \end{pmatrix} \qquad (1.1)$$

Los valores de las matrices hay que introducirlos entre corchetes. Cada elemento en una misma fila va separado por espacios (" ") y cada fila se separará con el signo";". Por ejemplo para introducir la matriz indicada por la expresión 1.1 habría que introducir el comando:

» **A=[1 2 3;4 5 -4;-3 -2 -1]**

A =

```
    1     2     3
    4     5    -4
   -3    -2    -1
```

En el caso de los vectores (1xN), como, por ejemplo, V=(1,2,3) se tiene que introducir:

» **V=[1 2 3]**
V =
```
    1     2     3
```

El signo "," también se puede utilizar de manera indiferente para separar los elementos de una misma fila:

» **VV=[9,8,7]**
VV =
```
    9     8     7
```

También es posible generar vectores con números equi-espaciados. Así, si se quiere generar valores entre **0** y **5** separados en incrementos de **0.5** habría que introducir:

» **g=0:0.5:5**
g =
```
  Columns 1 through 7

         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000

  Columns 8 through 11

    3.5000    4.0000    4.5000    5.0000
```

Este método es bastante práctico cuando se quiere generar señales con una referencia temporal. Por otro lado, no necesariamente hay que escribir números para introducir datos en el ordenador, sino que también se puede indicar expresiones y variables que estén contenidas en la memoria de *Matlab*.

» **vc=[**exp(1);**x;**sin(pi/2)**]**
vc =
```
    2.7183
    2.0000
    1.0000
```

Llegados a este punto, si ahora se introduce el comando *whos* aparece el listado de variables. Otra información importante que muestra este comando es la dimensión de las variables (*size*), y el número de elementos que contiene, además de la capacidad de memoria:

» **whos**

Name	Size	Elements	Bytes	Density	Complex
A	3 by 3	9	72	Full	No
V	1 by 3	3	24	Full	No
VV	1 by 3	3	24	Full	No
ans	1 by 1	1	8	Full	No
g	1 by 11	11	88	Full	No
vc	3 by 1	3	24	Full	No
x	1 by 1	1	8	Full	No
z	1 by 1	1	16	Full	Yes

Grand total is 32 elements using 264 bytes

En las operaciones con matrices, es importante que las dimensiones de los operandos sean coherentes según establecen las reglas matemáticas del cálculo de matrices. Así, la multiplicación de A*V (3x3*1x3) no es posible:

» A*V
??? Error using ==> *mtimes
Inner matrix dimensions must agree.

En cambio, si es posible multiplicar V*A ya que cumple las reglas (1x3 * 3x3 = 1x3):

» V*A
ans =
0 6 -8

Una operación muy útil, al mismo tiempo que utilizada en el cálculo de matrices es la *transpuesta*. Esta operación intercambia las filas de una matriz por sus columnas. Para ello, simplemente hay que incluir el signo “ ’ ” a continuación de la variable

» A'

ans =

1 4 -3
2 5 -2
3 -4 -1

De esta manera, si a un vector de una fila (1xN) se le realiza la transpuesta se obtiene una columna.

» **V'**
ans =
1
2
3

Para hacer cálculos de matrices punto a punto hay algunas diferencias dependiendo del tipo de operación que se quiera realizar. Por ejemplo, en el caso de la *suma* y la *resta* el cálculo puede realizarse de la manera normal:

» **[**5 3**]+[**2 6**]**
ans =
7 9

Sin embargo, en otras operaciones como el producto, el cociente y la potenciación (* , / , ^) es necesario introducir un punto antes del operador **(.* , ./ , .^)**. Como se puede comprobar a continuación, si se intenta multiplicar dos vectores con dimensiones (1x2) sin poner el punto delante del signo de operación, *Matlab* devuelve un error indicando que las dimensiones de las matrices no se corresponden, mientras que en el segundo caso devuelve un vector (1x2) que contiene el resultado de la multiplicación elemento a elemento.

» **[**5 3**]*[**2 6**]**
??? Error using ==> *
Inner matrix dimensions must agree.

» **[**5 3**].*[**2 6**]**
ans =
10 18

También existen funciones adicionales que permiten generar algunas matrices especiales. De este modo:

- **zeros (n,m)** genera una matriz de dimensiones n x m que contiene 'ceros'
- **ones (n,m)** genera una matriz de dimensiones n x m que contiene 'unos'
- **rand (n,m)** genera una matriz de dimensiones n x m que contiene números reales aleatorios entre 0 i 1.
- **eye(n)** genera una *matriz identidad* n x n

Para extraer datos de una o varias filas/columnas de la matriz hay que utiliza la notación ':'. Así, con la matriz anterior (A) la notación es: Nombre_de _variable(n,:) para extraer la fila; y Nombre_de_variable(:,m) para la columna.

» **A**
A =
1 2 3
4 5 -4
-3 -2 -1

» **A(**2,:**)**
ans =
4 5 -4

» **A(**:,3**)**
ans =
3
-4
-1

En *Matlab*, existen diversas maneras para leer diferente información de las matrices. Por ejemplo, para leer un único elemento de la matriz hay que indicar la fila y la columna del componente a extraer: Nombre_de_variable(n,m). También es posible extraer datos parciales de una fila o columna, en este caso hay que indicar el rango en un vector como se indica. También es posible leer las dimensiones de una matriz gracias al comando '**length**'. Por último, si se desconocen las dimensiones de una matriz, a veces es bastante útil el comando '**end'**, el cual, puede utilizarse tanto para filas como columnas.

» A(2,3**)**
ans =
-4

» A(2,[2:3])
ans =
5 -4

» length A(1,:)
ans =
3

» A(2,[1**:end])**
ans =
4 5 -4

En *Matlab* existen otros operadores y comandos básicos que, por brevedad, no se han comentado en esta sección. Todos ellos, se pueden consultar a través del menú de ayuda, o bien, utilizando el comando **'help'** directamente.

» help elmat;
» help elfun;
» help ops;
….

Entradas y salidas

Matlab también dispone de comandos especiales para mostrar datos por pantalla y también para solicitar información a los usuarios durante la ejecución de procesos. Para mostrar un texto en pantalla hay que utilizar la función ***disp****(X)*.

» disp ('Texto a mostrar');
Texto a mostrar

Como detalle, destacar que no aparece visualizada la variable '*ans'*. Esto se debe a que, a diferencia de los cálculos numéricos, en este caso no se realiza ningún tipo de asignación

Para visualizar los valores numéricos de las variables, el texto de la misma no ha de ir entre comillas " ' ". Por consiguiente al realizar la siguiente acción, se muestra por pantalla los valores de la matriz A.

» disp (A);
```
    1     2     3
    4     5    -4
   -3    -2    -1
```

Si la información a mostrar contiene tanto texto como valores numéricos, antes es necesario convertir los valores numéricos a texto. Para ello hay que utilizar la función ***num2str****(X)*. Una manera posible de hacerlo puede ser la siguiente:

```
» y=4;
» a=['y=' num2str(y)];
» disp(a);
y=4
```

En la ayuda de *Matlab* (**help** strfun), se puede encontrar diversas funciones relacionadas con el tratamiento de cadenas de texto.

A continuación, veremos como se puede pedir desde la pantalla al usuario como puede introducir datos. Para ello hay que utilizar la función ***input***. Cuando se ejecuta esta función, el programa se detiene y espera a que el usuario introduzca un dato a través del teclado y pulse el retorno de carro. En el caso de que este comando vaya acompañado de una asignación, la variable utilizada coge el valor introducido.

```
» n=input('Introduce numero : ')
Introduce numero : 7
n =
    7
```

Si se desea introducir texto, hay que añadir el parámetro 's' a esta función.:

```
» as=input ('Asignatura? ','s')
Asignatura? FEAD
as =
FEAD
```

Para concluir esta sección, veremos dos funciones bastante útiles que sirven para la temporización. En primer lugar, la instrucción ***pause*** detiene la ejecución del programa y espera hasta que se pulse alguna de las teclas. A esta función se le puede pasar un parámetro adicional que indica el tiempo de espera. Así, si se introduce el comando ***pause****(n)*, donde 'n' és un entero, la ejecución de programa se detiene durante *n* segundos.

```
» pause
» pause(2)
» (Después de 2 segundos)
```

Por último, las instrucciones ***tic*** y ***toc*** sirven para temporizar eventos. En primer lugar, hay que ejecutar el comando 'tic' para empezar que se inicie la contabilización del tiempo. Después de ejecutar las instrucciones que se deseen, al ejecutar el comando 'toc' la temporización se detiene y se muestra por pantalla el tiempo transcurrido.

```
» tic
» a=10;
» toc
Elapsed time is 4.445 seconds
»
```

Instrucciones de programación

Matlab incluye estructuras de programación que son muy similares a las que utilizan la mayoría de lenguajes de programación, como por ejemplo: C, C++, Pascal, etc. Los programas de Matlab (también denominados 'scrips') pueden implementarse con cualquier editor de textos y han de grabarse siempre con la extensión "***.m**". Repasaremos en esta sección algunas de las más básicas.

La sentencia **IF** permite ejecutar un grupo de instrucciones en función del cumplimiento de la *condición lógica* indicada. La estructura a utilizar es la siguiente:

```
IF condición
        Instrucciones;
ELSEIF condición
        Instrucciones;
ELSEIF condición
        Instrucciones;
.....
ELSE
        Instrucciones;
END
```

En el siguiente ejemplo, muestra por pantalla si un determinado número, introducido previamente por el usuario, es positivo o negativo

```
n=input('numero? ');
if n<0
        disp ('numero negativo');
elseif n==0
        disp ('cero');
else
        disp ('numero positivo');
end
```

Las condiciones lógicas disponibles con el entorno de *Matlab* son: **==, <, >, <=, >=,** o **~=.**

La estructura **FOR** permite repetir una serie de instrucciones o comandos un número finito de veces. La forma general es la siguiente:

FOR variable = expresión
    Instrucciones;
END

El siguiente ejemplo utiliza esta estructura para generar una matriz que en la primera fila contiene los números del 1 al 5, y sus cuadrados en la segunda.

```
for i=1:5
   v(1,i)=i;
   v(2,i)=i^2;
end
```

Si las instrucciones anteriores se guardan en un archivo, por ejemplo: *prfor.m*, el resultado que se obtiene al ejecutarlo es el siguiente:

```
» prfor
» v
v =
   1   2   3   4   5
   1   4   9  16  25
```

La estructura **WHILE** es similar a la estructura FOR con la diferencia de que las instrucciones se repiten *mientras* se cumpla una condición lógica. La forma general es la siguiente:

WHILE condicion
    Instrucciones;
END

En el siguiente ejemplo, la variable a se inicializa a cero. Esta variable se incrementa una unidad dentro del bucle. Esta acción se repite hasta que deja de cumplirse la condición (a<4). Por tanto, cuando se sale del bucle el valor de **a** es 4

```
a=0;
while a<4
   a=a+1;
end
```

Funciones

Muchas de las instrucciones de *Matlab* son en la práctica funciones a las que se les pasa unos parámetros, realizan un tratamiento y devuelven unos resultados. Este es el caso, por ejemplo, de la función ***size()*** que se utiliza para obtener el tamaño de la matriz. El formato que hay que utilizar es: [M,N] = SIZE(X), donde X es la matriz de entrada y la función devuelve en M el número de filas y en N el número de columnas (ver siguiente figura).

```
» v
v =
    1    2    3    4    5
    1    4    9   16   25
» [f,c]=size(v)
f =
    2
c =
    5
```

Matlab permite crear nuestras propias funciones. Para ello, además de crear un fichero (*.m) con las instrucciones que componen la función, hay que especificar en la primera línea que el fichero corresponde a una función. Para que estas funciones puedan ejecutarse, ha de cumplirse que el nombre de la función coincida con el nombre del fichero (*.m) creado, y que la entrada de parámetros junto a la asignación que se realice sea compatible con el formato especificado en dicha línea, tal y como se indica a continuación:

function salidas = **nombre_función** (entradas)

Así, si se quisiera crear una función que, por ejemplo, determine la media de los valores que hay en una matriz habría que proceder del siguiente modo. En primer lugar hay que crear un archivo de programa que tendrá el nombre *media.m*. Seguidamente, hay que indicar en la primera línea que el archivo corresponde a una función donde '*x*' contendrá la matriz introducida '*m*' será la variable designada para guardar el resultado (Fig. 36). Si se necesita crear variables adicionales dentro de la función, éstas tienen un carácter local. Esto significa que sólo pueden utilizarse en esta función. Sólo las variables que han sido declaradas como *globales* son las que permanecen en el área de trabajo y, por tanto, pueden utilizarse independientemente del punto de ejecución donde se encuentre el programa. Para ello, o se define como tal, (***global*** *nombre_variable*) o se introduce en la línea de comandos (»), o bien se asigna dentro de una función a una variable de salida que esté definida en el área de trabajo.

```
function m=media(x)

%Calculo de la media de los valores

[f,c]=size(x);
suma=0;
for it=1:f*c
      suma=suma+x(it);
end
m=suma/(f*c);

% Fin de la función
```

```
» a=[1 2 3;4 5 6]
a =
    1    2    3
```

```
    4     5     6
» med=media(a)
med =
   3.5000
```

Representación de gráficos con Matlab

Matlab también proporciona utilidades y herramientas para la representación gráfica tanto en 2D como 3D. La instrucción ***plot*()** es el comando principal que hay para este fin y puede utilizarse tanto para representar vectores como matrices en 2D.

De este modo, si 'X' y 'Y' son dos vectores, la sentencia ***plot***(X,Y) representa el vector Y (ordenadas) respecto el vector 'X' (abscisas). En el caso de haber indicar una única variable ***plot***(Y) la representación se realiza con respecto al *índice* de muestras de 'Y'. En este caso, si 'Y' está formado por números complejos, la acción anterior es equivalente a ***plot***(***real***(Y),***imag***(Y)). En el resto de casos donde se usa este comando, la parte imaginaria se ignora. El ejemplo de la figura 2.1 muestra como representar una senoide. Como se puede comprobar, existen comandos adicionales que sirven para mejorar el aspecto de los gráficos como: rejillas (***grid***), etiquetas en los ejes (***xlabel***, ***ylabel***), títulos (***title***), etc.

```
» a=0:2*pi/360:2*pi;
» b=sin(a);
» plot(a,b);
» grid
» xlabel('rad/seg')
» ylabel('sinus')
```

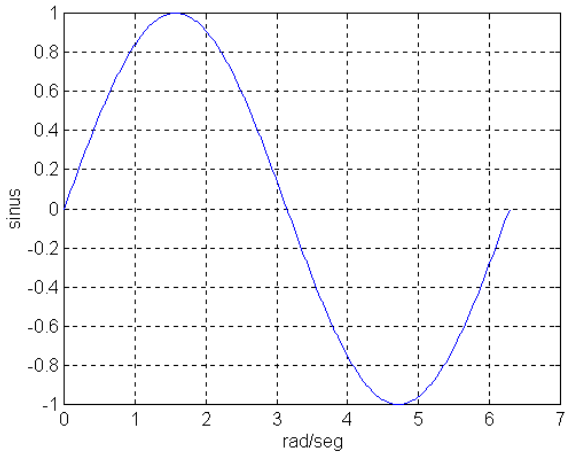


Figura 2.1.- Representación de una senoide con *Matlab*

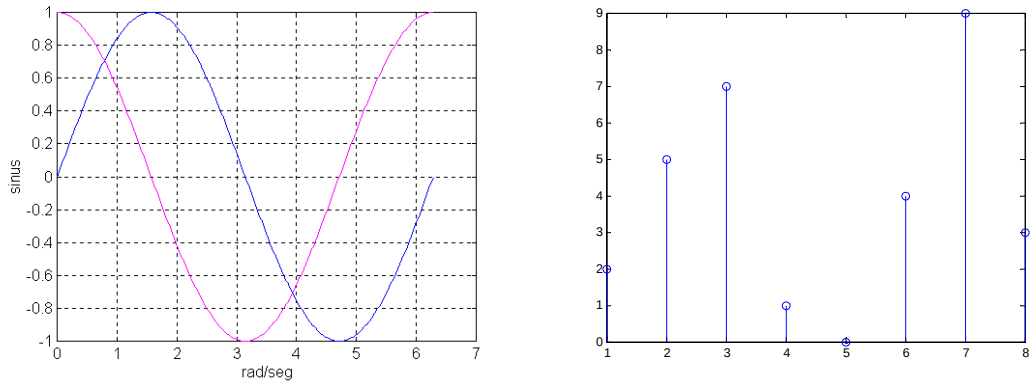
También es posible añadir un gráfico a la figura anterior. Para ello, hay que utilizar el comando ***hold on***. Además, la función ***plot*** también permite introducir parámetros adicionales para configurar tanto el color como el trazado de las líneas a utilizar. De este modo, el parámetro 'g-' de la figura 2.2, especifica que el nuevo gráfico a incluir se representará con una línea sólida de color '*cyan*'. Otra instrucción que puede ser de

utilidad en la representación de señales y sistemas discretos corresponde al comando ***stem***.

Más cuestiones sobre la representación gráfica se pueden encontrar en la sección de ayuda (***help*** plotxy, ***help*** graphics, ...)

» c=cos(a**);**
» hold on;
» plot(a,c,'c-'**);**

» a=[2 5 7 1 0 4 9 3**];**
» stem(a**)**



a) *b)*

Figura 2.- Uso de los comandos *hold on* y *stem. a)* Comandos a utilizar en el área de trabajo. *b)* Representación gráfica con el comando *hold on*. *c)* Representación con el comando *stem*

3 Análisis de sistemas en tiempo continuo

Después de haber conocido el funcionamiento básico de *Matlab*, a continuación aprenderemos a trabajar con las herramientas que sirven para el análisis de sistemas en tiempo continuo. Estas utilidades se encuentran en las librerías que vienen junto al programa y que se denominan *toolboxes*, algunas desarrolladas por terceras partes. Concretamente, las 'toolboxes' más adecuadas para este fin son las de control (*Control Toolbox*). Éstas, facilitan enormemente el manejo de las expresiones matemáticas que caracterizan las funciones de transferencia en tiempo continuo y, por tanto, también su análisis. Veamos como se utilizan.

Como se ha visto en las sesiones teóricas de la asignatura, las funciones de transferencia de los filtros analógicos tienen la forma:

$$\frac{as^n + bs^{n-1} + cs^{n-2} + ... + d}{es^m + fs^{m-1} + gs^{m-2} + ... + h} \tag{1.2}$$

Para que *Matlab* las pueda identificar, tanto el numerador como el denominador se han de tratar como vectores de tamaño *n* y *m* respectivamente, tal y como se especifica a continuación:

Numerador = [a b c ..... d]
Denominador = [e f g ....h]

De este modo, la función de transferencia caracterizada por la expresión 1.3 se puede introducir como se indica:

$$\frac{s^2}{s + 5s + 6} \tag{1.3}$$

$\underline{s^2}$

» num=[1 0 0**]**

$\underline{s^2}\underline{+5s+6}$

» **den=[**1 5 6**]**

Las funciones de transferencia también pueden asignarse a una variable estructurada como tal. Para ello se utiliza la función ***tf()***. En este caso, las variables adquieren una estructura específica denominada (*tf object*) que en *Matlab* equivale a una función de transferencia. Su notación es como sigue:

» **H=tf(num,den)**
Transfer function :

```
      s^2
-------------
s^2 + 5 s + 6
```

Una vez definida la función, existen múltiples instrucciones para su tratamiento. Una de ellas es la función ***bode()*** que sirve para calcular el comportamiento en frecuencia tanto gráficamente como numéricamente. Cuando esta función se asigna a una variable únicamente realiza el cálculo numérico y guarda el resultado en las variables correspondientes (MAG.- Magnitud, PHASE.- Fase, W.- Frecuencia angular).

[Mag,Phase,W] = ***bode(***num,den***)***

En cambio, cuando no se especifica variable alguna, *Matlab* representa gráficamente la respuesta en frecuencia (Fig. 3.1). Las unidades utilizadas en el gráfico son el 'decibelio' (dB) para la magnitud y los 'grados' para la fase.

» ***bode(***num,den***)***

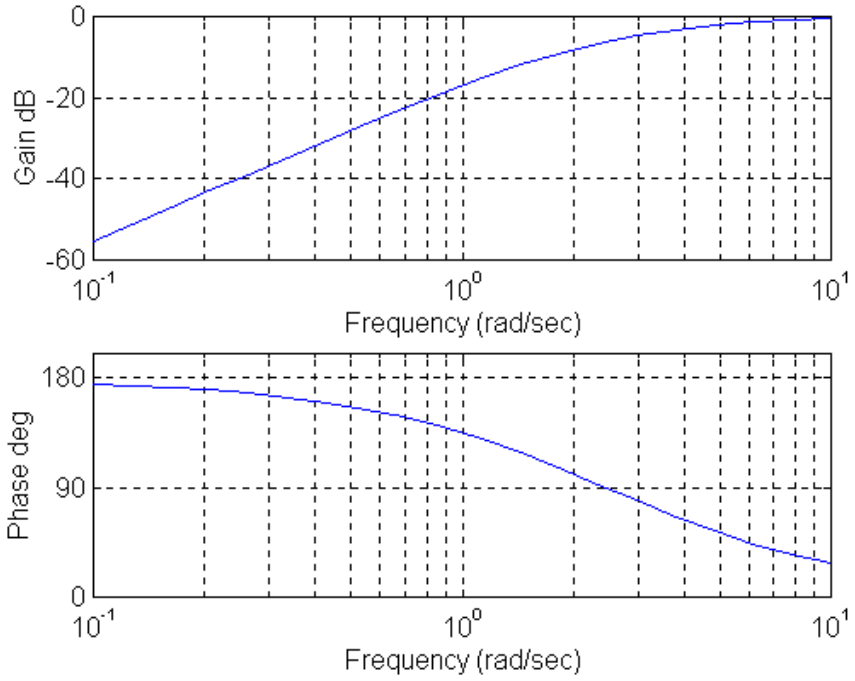


Figura 3.1.- Representación de la respuesta en frecuencia de una función de transferencia con Matlab

Para determinar las raíces de un polinomio se puede utilizar la función ***roots()***. Así, si se considera el ejemplo del denominador de la función de transferencia anterior (s^2+5s+6) la notación a utilizar es:

```
» de=[1 5 6]
» roots(de)
ans =
   -3
```

Estos indica que la solución del polinomio anterior es: –3 y –2, con lo que $s^2+5s+6 = (s+3)(s+2)$. Análogamente, también es posible obtener un polinomio a partir de las raíces. Esto se consigue con la función ***poly(·)***:

```
» raices=[-3 -2];
» poly(raices)
ans =
    1     5     6
```

Esta función que acabamos de ver tiene connotaciones similares a otra función disponible en *Matlab* conocida como la *convolución*. Aunque dicha función tiene su importancia en el análisis de la respuesta impulsional en sistemas discretos, también sirve para calcular el producto de polinomios en sistemas de tiempo continuo. La notación a utilizar en este caso es ***conv(·,·)***:

```
» p1=[1 3];
» p2=[1 2];
» conv(p1,p2)
ans =
    1     5     6
```

De este modo, si las dos secuencias anteriores corresponden a los polos de un sistema determinado esta se puede expresar como: $(s+3)(s+2)= s^2+5s+6$. Cuando se trabaja con sistemas más grandes (más de una función de transferencia) es bastante útil utilizar funciones de estructuras como, por ejemplo: ***series****(num1,den1,num2,den2),* ***parallel****(num1,den1,num2,den2)* o ***feedback****(num1,den1,num2,den2)*. La primera se utiliza para conectar estructuras que están conectadas en serie. De este modo, los polinomios especificados por la función de transferencia (*num1/den1*) se multiplican con los que hay en la otra función de transferencia (*num2/den2*). Así para formar una estructura en serie con dos funciones de transferencia de la expresión 1.4 habría que introducir los siguientes comandos:

$$\left(\frac{s+1}{s^2+2s+1}\right)\left(\frac{s^2}{s^2+4s+2}\right) \qquad (1.4)$$

```
» n1=[1 1];
» d1=[1 2 1];
» printsys(n1,d1);
num/den =
      s + 1
  -------------
  s^2 + 2 s + 1

» n2=[1 0 0];
» d2=[1 4 2];
» printsys(n2,d2);
```

```
num/den =
        s^2
  ------------
  s^2 + 4 s + 2
```

```
» [n12,d12]=series(n1,d1,n2,d2);
» printsys(n12,d12);
num/den =
           s^3 +  s^2
  ------------------------------
  s^4 + 6 s^3 + 11 s^2 + 8 s + 2
```

La funciones ***parallel*** y ***feedback*** funcionan de manera similar con la única diferencia que la primera realiza una suma de funciones de transferencia y la segunda aplica una realimentación negativa (El bloque formado por *num2/den2*, en este caso, forma parte del lazo de realimentación). Por otro lado, si se desea ver el diagrama de polos y zeros hay que utilizar la función ***zplane****(·,·)*. Aquí es necesario pasarle como parámetros las raíces de los polinomios (Fig. 3.2)

```
» num=[1 2];
» den=[1 2 2];
» printsys(num,den);
 num/den =
        s + 2
  ------------
  s^2 + 2 s + 2
```

```
» zplane(roots(num),roots(den));
```

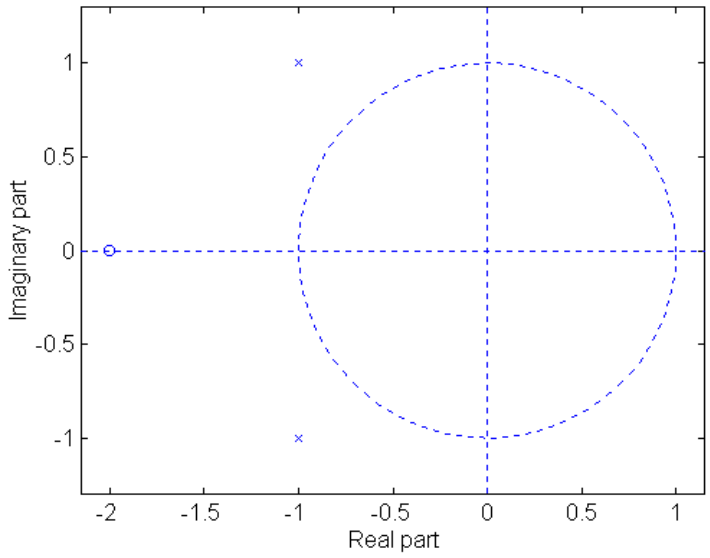


Figura 3.2- Diagrama polo-zero de la función detransferncia $(s+2)/(s^2+2s+2)$ con la función *zplane()*

Las cruces indican los polos sobre la figura, mientras que los ceros se indican mediante círculos pequeños de trazo continuo, tal y como se deduce a partir de las raíces:

» ceros=roots(num**)**
ceros =
    -2
» polos=roots(den**)**
polos =
  -1.0000 + 1.0000i
  -1.0000 - 1.0000i

Otra función útil es la que se ejecuta con el comando ***damp(·)***, el cual, se utiliza para determinar parámetros de un polinomio característico. En este caso, únicamente se ha de introducir el polinomio en forma de vector, y la función devuelve sus raíces, el factor de amortiguamiento ξ y la/s frecuencia/s de corte natural/es del sistema ω_o. Por ejemplo, si se buscan estos parámetros para la función de segundo orden $s^2+20s+400$ que es del tipo $s^2+2\xi\omega_o s+\omega_o^2$, se obtiene que $\xi=0.5$, $\omega_o=20$ y las raices: $-10\pm17.3205j$

» damp([1 20 400])
  Eigenvalue   Damping   Freq. (rad/sec)
 -10.0000 +17.3205i 0.5000   20.0000
 -10.0000 -17.3205i 0.5000   20.0000

4.- Creación y representación de señales discretas

La *toolbox* de procesado de señal de Matlab (*Signal Processing Toolbox*) consiste en toda una serie de funciones pensadas para el diseño, cálculo y análisis de sistemas discretos. Dicha librería comprende un amplio abanico de funciones de generación y procesado de señales, así como el diseño e implementación de sistemas digitales. Por tanto, es una buena herramienta para el desarrollo de filtros digitales. Esta librería, además proporciona dos tipos de herramientas: funciones que se ejecutan por línea de comandos e interfaces gráficas. Las funciones que se pueden utilizar en el área de trabajo se encuentran disponibles bajo las siguientes categorías:

- Análisis, diseño e implementación de filtros discretos (digitales)
- Análisis, diseño e implementación de filtros continuos (analógicos)
- Transformación de señales y sistemas lineales
- Análisis espectral
- Procesado estadístico
- Modelado parametrizado
- Predicción lineal
- Generación de señal
- Etc.

Las herramientas gráficas están disponibles para:

- Diseño y análisis de filtros
- Representación de señales, análisis espectral y de filtrado

Todos los comandos están implementados en archivos cuyo formato tiene la extensión '*.m'. Los parámetros de entrada básicos con los que estas funciones trabajan pertenecen a señales y sistemas. El filtro digital más básico que soportan dichas funciones es de la clase LTI (*Linear Time Invariant*), y más concretamente, los filtros FIR e IIR.

Las funciones, además, se complementan con las herramientas interactivas. Entre ellas, para el análisis de filtros digitales destacan:

- ***fdatool*** *(Filter Design and Análisis Tool).-* Es un entorno gráfico que ayuda a abordar aspectos de diseños de filtros.
- ***fvtool*** *(Filter Visualization Tool).-* Es un entorno gráfico que sirve para representar y visualizar la respuesta de filtros en general

Debido a la importancia de ellas en el diseño de filtros digitales, la herramienta *fdatool* se explicarán extensamente más adelante.

- ***Conceptos básicos sobre señales***

- ***Representación de señales***

Como acabamos de comprobar, *Matlab* utiliza el *array* como principal estructura para representar la información a procesar. En el caso de las señales, dichos datos, vienen caracterizados por vectores (1-*n*) o (*n*-1), donde *n* es el número de muestras de la señal. Una manera posible de introducir una secuencia en *Matlab* consiste en ponerlos como si fuera una lista de elementos. Así, al ejecutar la siguiente sentencia en la línea de comandos se crea un vector fila de cinco elementos

» **x** = [4 5 7 -9 1];

La transpuesta (x = x') convierte la fila en una columna, haciendo que la variable tenga el siguiente aspecto:

```
x =
     4
     3
     7
    -9
     1
```

Esta última distribución de datos es más adecuada para la representación de señales simples porque, a posteriori, facilita la extensión de la misma variable (x) a señales multi-canal simplemente añadiendo más columnas.

- ***Generando formas de onda***

Existen diversas maneras de generar formas de onda con las funciones de *Matlab*. La gran mayoría precisan de una base temporal (normalmente unitaria). Por ejemplo, si se desea generar datos con una frecuencia de muestreo de 1000Hz (1000muestras/seg), el vector de tiempo ($t=nT$) se puede crear haciendo:

t = (0:1/1000:1)';

En este caso, *t* queda formado por un vector columna que contiene 1001 elementos que comprenden el rango [0,1]. A partir de este vector es más fácil crear señales muestreadas. Por ejemplo, una señal que consiste de dos sinusoides: una a 50Hz i la otra a 120Hz con el doble de amplitud respecto a la primera (Fig. 4.1)

» **y = sin**(2*pi*50*t)+ 2***sin**(2*pi*120*t);
» **plot(y,t);**

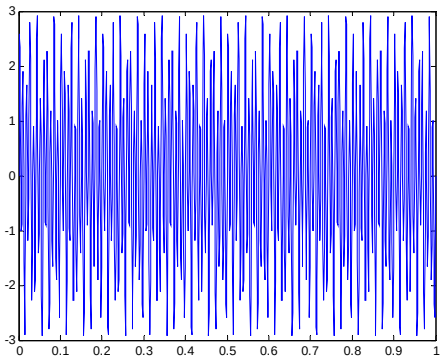


Figura 4.1.- Representación de la función: sin(2*pi*50*t)+ 2*sin(2*pi*120*t)

Esta nueva variable (*y*) también tiene una longitud de 1001 muestras. También se puede añadir *ruido blanco* a señales previamente creadas y dibujar, por ejemplo, las primeras 50 muestras de la señal resultante:

randn('state',0);
yn=y+0.5***randn(size(**t**));**
plot(t(1:50),**yn**(1:50));

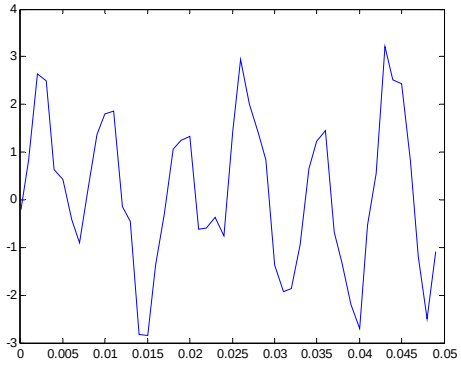


Figura 4.2.- Representación de las primeras 50 muestras correspondientes a la función: sin(2*pi*50*t)+ 2*sin(2*pi*120*t) con ruido blanco.

En general, debido a que *Matlab* utiliza un lenguaje de programación es posible generar infinidad de señales. Aquí tenéis una muestra de algunas de las más comunes:

t = (0:0.001:1)';
y =[1; zeros(99,1)]; % Respuesta impulsional unitaria de 100 muestras
y = **ones**(100,1); % Escalón unitario
y = **t;** %Rampa unitaria
y = **t**.^2; % Función quadrática
y = **square**(4*t); % Onda cuadrada (Similar a la función *sin*)

Como se puede observar, las tres últimas utilizan la base temporal para generar la señal correspondiente.

- ***Gestión de datos en el área de trabajo***

En la sección anterior, hemos visto que se pueden entrar los datos manualmente desde el teclado, o bien utilizando funciones contenidas en las librerías de *Matlab* (**sin**, **cos**, etc). Sin embargo, según la aplicación a veces es muy útil poder importar datos desde el exterior. Dependiendo del formato utilizado en los datos, esto se puede realizar de diferentes maneras:

- Cargando datos en formato ASCII, o bien '*.mat' a partir del comando **load** (**help** load)
- A partir de la lectura de ficheros *.txt con funciones de bajo nivel **fopen**, **fread**, **fscanf** (**help** fopen, fread, fscanf)
- Desarrollando un fichero '*.mex' para la lectura de datos

También hay disponibles recursos que trabajan en lenguaje de más alto nivel (Fortram o C, por ejemplo) que pueden utilizarse para convertir datos al formato '*.mat'. Consultar la documentación **Matlab External Interfaces**, para saber más detalles al respecto.

Análogamente, también existen técnicas parecidas para exportar datos generados en Matlab a otros programas (Consultar el documento **Importing and Exporting Data** para obtener más información al respecto)

Actividad de laboratorio (Matlab)

La actividad que realizareis a continuación cumple un doble objetivo. Por un lado, os ha de servir para familiarizaros con el método de trabajo de Matlab que se ha de utilizar en los algoritmos de cálculo matemático. Estos métodos que se acaban de explicar en los apartados anteriores os han de servir, por otro lado, para llevar a la práctica los aspectos sobre procesado digital que se han explicado en las sesiones teóricas, en particular: las señales y sistemas: la que se realizará justo a continuación (Actividad 4) y los aspectos relacionados con los sistemas y filtros digitales (Actividades 5 y 6 respectivamente) que se realizarán más adelante

La resolución de estas actividades se tendrán que entregar con el correspondiente informe (junto con los archivos correspondientes) para poder realizar la evaluación de las prácticas de laboratorio de filtros digitales.

Actividad nº 4

- **Trabajo previo:** Contestad las preguntas que se plantean en el ejercicio 1.10 de la colección de problemas y cuyo enunciado es el siguiente:

Por un enlace de comunicaciones digitales se transmiten palabras codificadas en binario que representan muestras de la siguiente señal:

$$x_a(t) = 3\cdot\cos(600\cdot\pi\cdot t) + 2\cdot\cos(1800\cdot\pi\cdot t)$$

El enlace trabaja a 10000 bits/s y cada muestra se cuantifica con 1024 niveles de tensión diferentes. Se pide:

a) Frecuencia de muestreo máxima que se puede utilizar para digitalizar el señal que no produciría ambigüedad en una hipotética situación en la que se quisiera recuperar el señal original.

b) Tasa de >Nyquist de $x_a(t)$

c) Componentes frecuenciales de la señal en tiempo discreto.

d) Resolución (Δ) y SQNR en dB's ($SQNR = 10\cdot\log_{10}(P_x/P_q)$)

- **Desarrollo con Matlab:**

P1) Implemente la versión digital de la señal $x_a(t) \equiv x_a(nT)$ y guárdela en una variable. Para ello utilizad **200 muestras** para representar los primeros **10mseg** de la señal. ¿Cuál es la frecuencia de muestreo F_s utilizada en la digitalización de dicha señal?

P2) Repita el apartado anterior para las siguientes frecuencias de muestreo: F_s = **10KHz**, F_s = **5KHz**, F_s = **2KHz**, F_s = **1KHz**. Comentar los resultados obtenidos en cada caso. ¿Se puede decir que el muestreo utilizado por el enlace es correcto para poder reconstruir la señal original?

Nota: Existe una manera de evaluar los resultados en cada caso que os puede ayudar a ver mejor la naturaleza del problema. Ésta consiste en obtener el espectro de cada una de estas señales a partir de la *transformada de Fourier* (consultar el *help* de Matlab para obtener más información al respecto)

P3) Determine la convolución *y(n) = x(n)*h(n)*, de las señales siguientes:

a) ***x(n)***={1,1,1,1,0} ***h(n)***={6,5,4,3,2,1,0,0,0}
b) ***x(n)***={1,2,4} ***h(n)***={1,1,1,1}
c) ***x(n)***={1,2,-1} ***h(n)=x(n)***
d) ***x(n)***={0,0,1,1,1,1} ***h(n)***={1,-2,3}

Nota: Para realizar este apartado se recomienda implementar una función en Matlab que a partir de las entradas: *x(n), h(n)* y los índices de cada una de ellas, devuelva el resultado en un vector de salida con su correspondiente índice inicial. Comparar el resultado con la función ***conv()*** que incluye Matlab

P4) Realizar el ejercicio **1.14** de la colección de problemas. Para ello hacer servir la herramienta Matlab para contestar las cuestiones planteadas

<u>Actividad opcional</u>:

Adicionalmente, y con carácter voluntario, se propone analizar el efecto del error de cuantificación que suele producirse en el proceso de conversión de las señales analógicas. Para ello, se propone realizar el ejercicio **1.15** de la colección de problemas que aborda esta problemática ayudándose de la herramienta Matlab

(Continuación cuaderno de prácticas de FEAD)

5.- Análisis de sistemas discretos

En esta sección, vamos a explicar como utilizar las funciones para trabajar con sistemas discretos, entre las cuales, se encuentran los filtros digitales. También aprenderemos a utilizar las funciones de las librerías para analizar las características de los filtros: respuesta impulsional, magnitud, fase, etc.

Convolución vs filtrado

Como se ha explicado en sesiones teóricas, la base matemática del filtraje de señales es la convolución. En *Matlab* este cálculo se lleva a cabo con la función ***conv***() que realiza el cálculo a partir de dos vectores estándar.

» **conv(**[1 1 1],[1 1 1]**)**
ans=
1 2 3 2 1

La convolución relaciona la salida *y(n)* con la entrada *x(n)* a partir de la respuesta impulsional del filtro *h(n)* (ver expresión 5.1). Si la respuesta impulsional del filtro y la señal de entrada son de longitud finita (filtro FIR), éste se puede implementar con la función ***conv***. Para ello, simplemente hay que crear dos vectores que contengan la señal *x(n)* y la respuesta impulsional *h(n)*, respectivamente y calcular la convolución de ambos

$$y(n) = h(n) * x(n) = \sum_{k=-\infty}^{\infty} h(n-k)\cdot x(k) \qquad (5.1)$$

» **x = randn(**5,1**])**; % vector aleatorio de longitud 5
» **h = [**1 1 1 1/4**];** % respuesta impulsional del filtro
» **y = conv(**h,x**);**

La función FILTER

Sin embargo, este no es el caso de la mayoría de filtros que se implementan en la práctica, que son los IIR. La función ***filter*** soluciona este problema ya que con ella, es posible determinar la respuesta de cualquier filtro. Dicha función, implementa la estructura directa II transpuesta (Fig. 5.1) puesto que es la forma canónica que tiene el menor número de retardos.

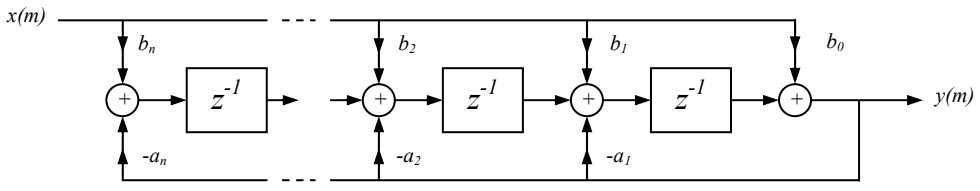


Figura 5.1.- Forma canónica del filtro implementado con la estructura directa II transpuesta

Para determinar la respuesta de salida en la muestra *m*, el filtro calcula las siguientes ecuaciones en diferencias:

$$y(m) = b(0)\cdot x(m) + z_1(m-1)$$
$$z_1(m) = b(1)\cdot x(m) + z_2(m-1) - a(1)\cdot y(m)$$
$$\ldots.$$
$$z_{n-2}(m) = b(n-1)\cdot x(n) + z_{n-1}(m-1) - a(n-1)\cdot y(m)$$
$$z_{n-1}(m) = b(n)\cdot x(m) - a(n)\cdot y(m)$$

La notación que hay que utilizar es la siguiente:

***y* = *filter*(**b,a,x**);**

donde *x* corresponde al vector con la señal de entrada e *y* es la salida del filtro caracterizado por los vectores con los coeficientes *a* y *b*. Dicha notación corresponde a la forma más básica, la cual, inicializa todos los retardos a cero. No obstante, se puede inicializar estos retardos utilizando un cuarto parámetro de entrada o, también, mirar el estado final de los mismos utilizando otro parámetro de salida:

» [***y,zf***] = ***filter*(**b,a,x, zi**);**

Poder acceder a las condiciones iniciales y finales de los retardos es muy útil cuando se necesita filtrar la entrada por trozos. Esto ocurre cuando las limitaciones de memoria son bastante severas. Supongamos que se tienen datos en dos vectores de 5000 muestras cada uno y que se quieren procesar:

» **x1 = randn(**5000,1**);**
» **x2 = randn(**5000,1**);**

En este sentido, la secuencia x_1 puede pertenecer a los diez primeros minutos de datos y x_2 a los diez minutos adicionales. La secuencia completa seria x=[x1 ; x2]. Si no se dispone de suficiente memoria para guardar la secuencia completa, el filtrado se puede realizar de manera separada utilizando las condiciones finales de x_1 como condiciones iniciales de x_2:

» **[y1,zf] = filter(**b,a,,x1**);**
» **y2 = filter(**b,a,x2,zf**);**

También es posible especificar condiciones iniciales particulares mediante la función **filtic** (Consultar la ayuda de *Matlab* para más información)

Respuesta frecuencial

La función ***freqz()*** utiliza un método basado en la transformada rápida de Fourier (FFT) para calcular la respuesta en frecuencia de un filtro discreto. La notación básica es:

[*h,w*] = *freqz*(b,a,p**);**

Dicha función, retorna la respuesta en frecuencia compleja del filtro digital, $H(e^{jw})$, especificado por la expresión 5.2. En general, ***freqz*** utiliza las valores de los vectores *a* y *b* como coeficientes y devuelve la respuesta frecuencial compleja en el vector *h* para un total de *p* puntos. El vector *w* contiene el valor de la frecuencia en rad/seg. para todos los puntos de *h*.

$$H\left(e^{jw}\right)=\frac{b(1)+b(1)e^{-jw}+\ldots+b(n+1)e^{-jw(n)}}{a(1)+a(2)e^{-jw}+\ldots+a(m+1)e^{-jw(n)}} \tag{5.2}$$

freqz() también acepta otros parámetros como pueden ser la frecuencia de muestreo o, incluso, un vector de puntos arbitrario. El ejemplo que indicamos a continuación calcula la respuesta frecuencial para 256 puntos del filtro de Chebyshev (tipo I) de orden 12, especificando una frecuencia de muestreo f_s de 1000Hz

» **[b,a] = cheby1(**12,0.5,100/500**);**
» **[h,f] = freqz(**b,a,256,1000**);**

En este caso, debido a que la expresión incluye la frecuencia de muestreo, para evitar *aliasing* la función retorna un vector *f* que contiene la respuesta frecuencial para 256 puntos entre 0 y $f_s/2$.

Cuando no se asignan variables de salida a esta función, por defecto, *Matlab* entiende que lo que se desea es visualizar la magnitud i la fase de la respuesta frecuencial. Por ejemplo, la Fig. 5.2 representa la respuesta de un filtro paso-bajo de Butterworth con una frecuencia de corte de 400Hz con una frecuencia de muestreo de 2000Hz.

» **[b,a] = butter(**9,0400/1000**);**
» **freqz(**b,a,256,2000**);**

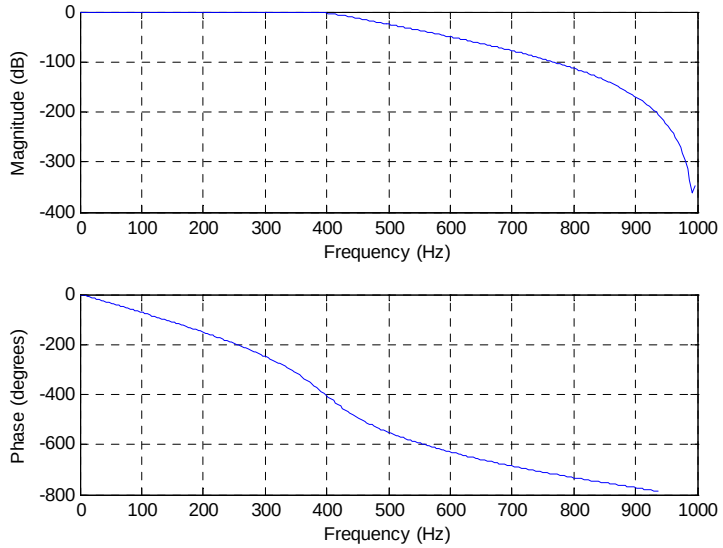


Figura 5.2.- Respuesta frecuencial del filtro de Butterworth de orden 9

De manera alternativa, la magnitud y fase también se puede visualizar en la herramienta interactiva **fvtool**, la cual, contiene herramientas adicionales para el análisis de filtros. Para ello se ha de ejecutar el siguiente comanda y 'clickar' en el botón  :

» **fvtool(**b,a**);**

Transformada de Fourier Discreta

La transformada discreta de Fourier (DFT.- *Discrete Fourier Transform*) constituye uno de los métodos principales que se utiliza de manera general en el ámbito de procesado digital de la señal. La FFT (*Fast Fourier Transform*) sienta las bases para un cálculo rápido de la DFT y está incluida como parte del proceso en alguna de las funciones de diseño de filtros que acabamos de ver.

Matlab tiene los comandos **fft** e **ifft** para el cálculo de la transformada de Fourier y su inversa, respectivamente. Esta última se utiliza en la reconstrucción de señales y el filtrado. En general, dada una secuencia de entrada *x* - en un caso, o la transformada *X*, - en el otro; las funciones que implementan estos cálculos son:

$$X(k+1)=\sum_{n=0}^{N-1}x(n+1)\cdot e^{-j\left(\frac{2\pi kn}{N}\right)} \rightarrow \text{DFT} \qquad (5.3)$$

$$x(n+1)=\sum_{k=0}^{N-1}X(k+1)\cdot e^{j\left(\frac{2\pi kn}{N}\right)} \rightarrow \text{IDFT} \qquad (5.4)$$

Como se puede apreciar, las variables de salida empiezan por el índice 1, en vez de 0 debido a la notación utilizada por *Matlab*. De este modo, si *x* es un vector, **fft** calcula la DFT de dicho vector, en cambio, si *x* es una matriz **fft** calcula la DFT de cada columna.

Veamos un ejemplo. Crearemos una señal que contiene dos componentes sisusoidales de 15 y 40Hz, respectivamente. El muestreo utilizado en las mismas es de 100 muestras/seg, y se representa para 10 segundos de tiempo.

» **t = (**0:1/100:10-1/100**);**
» **x = sin(**2*pi*15*t**)+sin(**2*pi*40*t**);**

Al calcular la transformada de esta señal, la magnitud y fase se pueden determinar a partir de los siguientes comandos:

» **y = fft(**x**);** % Cálculo de DFT
» **m = abs(**y**); p = unwrap(angle(**y**))**; % Magnitud y fase

Para dibujar la magnitud y la fase (Fig. 5) se tendría que hacer lo siguiente:

» **f = (**0 : **length(**y-1)*99**)/length(y);** % vector que especifica el eje de la frecuencia *f*
» **plot(**f,m**), title(**'Magnitud'**);**
» **set(**gca,'XTick',[15 40 60 85]**);**
» **figure; plot(**f,p*180/pi**); title(**'Phase'**);**
» **set(**gca,'XTick',[15 40 60 85]**);**

También se puede introducir un parámetro adicional *n* para especificar el número de puntos de la variable de salida (y = fft(x,n);). Esto es aconsejable cuando se desea aumentar la resolución del espectro y, por características de las señales, éstas poseen componentes frecuenciales bastante próximas entre ellas. En este caso, la **fft** añade ceros a la secuencia de entrada *x*, o bien realiza un truncamiento de la misma si el número de muestras de ésta es inferior al parámetro especificado.

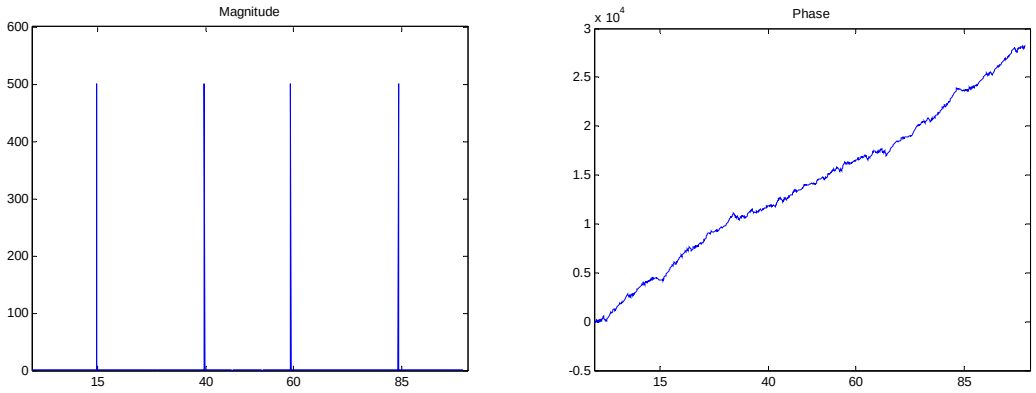


Figura 5.3.- Cálculo de la DFT a partir de la función *fft()* de *Matlab*

La transformada inversa **ifft** también precisa de un vector de entrada para poder realizar el cálculo. La diferencia, en este caso es que dicho vector contiene una secuencia de números complejos ya que contienen la información de la magnitud y la fase correspondiente. En el procesado de señal, la transformada inversa se utiliza en la reconstrucción de señales. Esta característica es lo que se puede contemplar si se ejecutan los siguientes comandos:

» **t = (**0:1/255:1**);**
» **x = sin(**2*pi*120*t**);**
» **y = real(ifft(fft(**x**));**

Actividad nº 5. Implementación de sistemas discretos

P1) Represente el espectro de las señales de los apartados P1 y P2 de la actividad 4. Indique la amplitud de cada componente frecuencial y obtenga el espectro de la fase.

P2) Considere el sistema discreto descrito por la ecuación en diferencias

$$y(n) = -a_1 \cdot y(n\text{-}1) - a_2 \cdot y(n\text{-}2) + b_0 \cdot x(n)$$

donde: $a_1 = -0.8$, $a_2 = -0.64$ y $b_0 = 0.866$

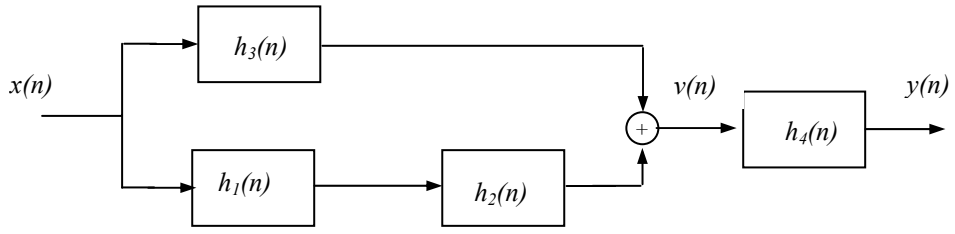
a) Represente el diagrama de bloques correspondiente al sistema descrito
b) Desarrolle un programa en Matlab que calcule implemente el sistema. Represente la respuesta impulsional del sistema para las 50 primeras muestras.

   Nota: Se aconseja utilizar la función ***filter(·)***
c) Represente la respuesta del sistema a la función escalón
d) Con los datos obtenidos, comente las características del sistema ¿Es causal? ¿Es lineal? ¿Se trata de un sistema estable? ¿Cómo es su respuesta impulsional?

P3)

a) Cree un programa en Matlab correspondiente al siguiente sistema descrito que determine la respuesta a cualquier señal de entrada. El sistema viene descrito por las siguientes respuestas impulsionales y diagrama de bloques:

$$h_1(n) = \{1, \tfrac{1}{2}, \tfrac{1}{4}, 1/8, 1/16, 1/32\}$$
$$h_2(n) = \{1, 1, 1, 1, 1\}$$
$$h_3(n) = \tfrac{1}{4} \cdot x(n) + \tfrac{1}{2} \cdot x(n\text{-}1) + \tfrac{1}{4} \cdot x(n\text{-}2)$$
$$h_4(n) = 0.9 \cdot y(n\text{-}1) - 0.81 \cdot y(n\text{-}2) + v(n) + v(n\text{-}1)$$



b) Comente los resultados y las características del sistema creado

Continuación del cuaderno de prácticas de FEAD

6 Diseño e implementación de filtros digitales

El diseño de filtros es un proceso que básicamente consiste en determinar el orden y los coeficientes del filtro de manera que cumplan unas especificaciones determinadas. Sin embargo, también hay que tener en cuenta su implementación, la cual consiste en escoger y determinar una estructura específica para estos coeficientes. En suma, el filtro no se implementa hasta que no se consigue una estructura específica en la que se puedan introducir señales y, por tanto, filtrar datos. El presente apartado describe el diseño de filtros y su implementación con *Matlab*

Requisitos y especificaciones

Normalmente, el diseño de un filtro persigue la alteración de una señal en base a unas especificaciones de frecuencia. Por ejemplo, un requisito podría ser el de eliminar ruido por encima de 30Hz en una señal que ha sido muestreada a 100Hz. En este caso, una posible opción sería implementar un filtro paso bajo con una frecuencia de corte de 30Hz (notad que la banda completa del espectro en este caso vendría delimitada por el muestreo, en este caso, 50Hz). También existen otras especificaciones más rigurosas, como por ejemplo: el *rizado* de la banda de paso, *atenuación* de la banda eliminada o, incluso la banda de *transición*. Otras especificaciones más rigurosas pueden solicitar conseguir los objetivos con el mínimo orden de filtro posible, forma de la respuesta o, incluso pueden llegar a requerir un filtro FIR.

Los métodos de diseño varían básicamente dependiendo del comportamiento que se espera conseguir de ellos. Para especificaciones bastante relajadas, como el ejemplo que viene a continuación, es suficiente con un filtro Butterworth IIR. En este caso, para diseñar un filtro paso-bajo de orden 5 a la frecuencia de corte indicada y aplicarlo a la señal de entrada bastaría con hacer:

```
» [b,a]=butter(5,30/50);
» Hd = dfilt.df2t(b,a);
» y = filter(hd,x);
```

Como se puede apreciar, el segundo parámetro de la función ***butter()*** especifica la frecuencia de corte normalizada (en el dominio digital) que en este caso es la mitad de la frecuencia de muestreo (100Hz). Este es un requisito fundamental en todas las funciones de diseño de filtros discretos de *Matlab*, es decir, trabajan con frecuencias normalizadas. Gracias a esta característica no es necesario incluir el tiempo de muestreo como parámetro adicional. Por convención, todas las librerías en Matlab trabajan bajo la consideración de que la frecuencia unitaria es la frecuencia de Nyquist (definida como la mitad de la frecuencia de muestreo, ya que se supone que es la máxima con la que se puede trabajar). Por tanto, la frecuencia normalizada siempre está incluida en el intervalo [0,1]. Así, para un sistema discreto que presenta un muestreo de 1000Hz, una frecuencia de corte de 300Hz equivale a 300/500 = 0.6. Para convertir una frecuencia normalizada a una frecuencia angular en el círculo unitario, simplemente hay que multiplicar su valor por π, mientras que para convertir a Hz hay que multiplicar por la mitad de la frecuencia de muestreo.

Por otro lado, especificaciones como el rizado de la banda pasante (R_p, en dB's) la banda de atenuación (R_s) o la banda de transición (W_s- W_p en Hz) son válidas con el diseño de filtros de Butterworth, Chebyshev (tipos I y II) y los filtros elípticos. Estas funciones también estiman el mínimo orden del filtro necesario para cumplir con estas especificaciones.

Especificaciones más concretas como la fase lineal o la forma de su respuesta utilizan estructuras FIR más complejas y no se estudiarán en este curso.

Diseño de filtros IIR

La principal ventaja de los filtros IIR sobre los filtros FIR es que, generalmente, cumplen con las especificaciones sin necesidad de tener que utilizar un filtro de orden elevado. Sin embargo, la no linealidad de su fase hace que no sean muy utilizados en aplicaciones de tiempo real. *Matlab* incluye funciones para crear este tipo de filtros. Existen dos grupos de funciones que se pueden utilizar en el diseño de filtros:

Métodos de diseño directo.- Son funciones que devuelven los coeficientes del filtro en base a unos parámetros de entrada (especificaciones del filtro).

- **besself**, **butter**, **cheby1**, **cheby2**, **ellip**: Funciones básicas de diseño de filtros IIR de Bessel, Butterworth, Chebyshev (I y II) y elíptico, respectivamente. Sirven tanto para el dominio analógico como el digital (exceptuando el filtro Besself que únicamente está disponible en el dominio analógico). La notación que utilizan es la siguiente:

  Bessel → [b,a] = **besself**(n, Wn, *opciones*)
  Butterworth → [b,a] = **butter**(n, Wn, *opciones*)
  Chebyshev I → [b,a] = **cheby1**(n, Rp, Wn, *opciones*)
  Chebyshev II → [b,a] = **cheby2**(n, Rs, Wn, *opciones*)
  Elíptico → [b,a] = **ellip**(n, Rp, Rs, Wn, *opciones*)

  Todas estas funciones retornan por defecto un filtro paso-bajo. Los parámetros de entrada que no están en cursiva son los mínimos que se han de introducir para que el diseño del filtro sea implementado por *Matlab*. El parámetro '*n*' corresponde al orden del filtro mientras que *Wn* tiene que estar normalizada a la frecuencia de Nyquist (0< Wn < 1). Para indicar que el filtro es paso-alto hay que incluir la cadena de caracteres ('high') como opción adicional Si el filtro es paso-banda o banda-eliminada, Wn es un vector que contiene dos elementos que especifican los limites de la banda en cuestión. En este último caso, además, hay que incluir la cadena 'stop'. Rp y Rs han de expresarse en dB's He aquí unos ejemplos:

  » **[b,a] = butter(**5,0.4**);**
  » **[b,a] = cheby1(**4,1, [0.4 0.7]**);**
  » **[b,a] = cheby2(**6,60, 0.8, 'high'**);**
  » **[b,a] = ellip(**3,1, 60,[0.4 0.7],'stop'**);**

En todas las funciones de filtros se puede añadir como acción el parámetro 's' para especificar que los parámetros de entrada corresponden al *dominio analógico*. En este caso, las frecuencias Wp, Ws, ... se especifican en rad/seg.

» **[b,a] = butter(**5,pi/2,'s'**);**

Las funciones de diseño retornan los coeficientes del filtro en uno u otro formato dependiendo de cómo se haga la asignación en las variables de salida:

» **[b,a] = butter(**n,Wn**)** **%** Coeficientes de la función de transferencia
» **[z,p,k] = butter(**n,Wn**);** % Zero-polo-ganancia
» **[A,B,C,D] = butter(**5,0.4**);** % Modelo representado en el espacio de estado

- **buttord**, **cheb1ord**, **cheb2ord**, **ellipord**: Funciones para estimar el orden mínimo necesario para la implementación de un filtro digital en base a las especificaciones de la banda de paso especificada por las frecuencias de corte y el rizado: Wp frecuencia de de la banda de paso), Ws (frecuencia de la banda de atenuación), Rp (rizado en la banda de paso) y Rs (Rizado en la banda de atenuación)

  Butterworth → [n,Wn] = **butterord**(Wp, Ws, Rp, Rs)
  Chebyshev I → [n,Wn] = **cheby1ord**(Wp, Ws, Rp, Rs)
  Chebyshev II → [n,Wn] = **cheby2ord**(Wp, Ws, Rp, Rs)
  Elíptico → [n,Wn] = **ellipord**(Wp, Ws, Rp, Rs)

  El tipo de filtro queda determinado por el valor de Wp y Ws:

  Paso bajo: Ws>Wp
  Paso-alto: Wp>Ws
  Paso banda: Wp =[Wp1 Wp2], Ws=[Ws1 Ws2] donde: Ws1<Wp1<Wp2<Ws2
  Banda-eliminada:
  Wp=[Wp1 Wp2], Ws=[Ws1 Ws2] donde: Wp1<Ws1<Ws2<Wp2

  Wn corresponde a la frecuencia natural del filtro.

Diseño basado en el método de la conversión del filtro analógico a digital.- MatlabbSon funciones de soporte al diseño, que se basan en la ***conversión*** de filtros analógicos paso-bajo normalizados a su equivalente en digital. Este método consiste en tres pasos principales: 1) Obtención de la función base (un filtro paso bajo con frecuencia de corte Wo =1); 2) Transformación en frecuencia para obtener el filtro deseado; 3) Discretización. A continuación se indican las funciones que hay en *Matlab* disponibles para cada uno de estos pasos:

1º Obtención del prototipo analógico paso-bajo normalizado.

- **besselap**, **buttap**, **cheb1ap**, **cheb2ap**, **ellipap**: Estas funciones sirven para generar lo filtros paso-bajo normalizados (Wo = 1) correspondientes a las configuraciones básicas: Bessel, Butterworth, Chebyshev (I y II) y Elíptico.

Bessel → [z,p,k] = **besselap**(n)
Butterworth → [z,p,k] = **buttap**(n)
Chebyshev I → [z,p,k] = **cheby1ap**(n, Rp)
Chebyshev II → [z,p,k] = **cheby2ap**(n, Rs)
Elíptico → [z,p,k] = **ellipap**(n, Rp, Rs)

En todas estas funciones, las variables de salida '*p*' '*k*' y '*z*' contienen polos, ganancia y una matriz vacía (Butterworth es un sistema todo polos), respectivamente. Por otro lado, 'n', *'Rp'* y *'Rs'* son parámetros de entrada que sirven para indicar el orden del filtro, el rizado de la banda de paso y la banda de rechazo, respectivamente. Por ejemplo, el prototipo paso bajo Butterworth de orden 2 tiene la siguiente expresión normalizada:

```
» [z,p,k]=buttap(2)
z =
[]
p =
 -0.7071 + 0.7071i
 -0.7071 - 0.7071i
k =
 1
» den=poly(p)
den =
 1.0000    1.4142    1.0000
» printsys(k,den)
 num/den =
         1
 -----------------
 s^2 + 1.414 s + 1
```

2º Transformación del filtro analógico a la configuración de la banda requerida

- **lp2bp**, **lp2bs**, **lp2hp**, **lp2lp**: Estas funciones sirven para aplicar transformaciones en frecuencia y obtener otros tipos de filtros. Con ellas, se pueden trabajar tanto a partir de su función de transferencia como con su representación en espacio de estado. A continuación se indican ambos casos para cada transformación

lp2lp.- Paso-bajo → Paso-bajo.

» [numt,dent] = **lp2lp(**num,den,Wo**)**
» [At,Bt, Ct, Dt] = **lp2lp(**A,B, C, D, Wo**)**

$$s' = \frac{s}{\omega_0} \tag{6.1}$$

lp2hp.- Paso-bajo → Paso-alto.

» [numt,dent] = **lp2hp(**num,den,Wo**)**

» [At,Bt, Ct, Dt] = **lp2hp(**A,B, C, D, Wo**)**

$$s' = \frac{\omega_0}{s} \tag{6.2}$$

lp2bp.- Paso-bajo → Paso-banda.

» [numt,dent] = **lp2bp(**num,den,Wo, Bw**)**
» [At,Bt, Ct, Dt] = **lp2bp(**A,B, C, D, Wo**)**

$$s' = \frac{\omega_0}{B_W} \frac{(s/\omega_0)^2 + 1}{s/\omega_0} \tag{6.3}$$

lp2bs.- Paso-bajo → Banda-eliminada.

» [numt,dent] = **lp2bs(**num,den,Wo**)**
» [At,Bt, Ct, Dt] = **lp2bs(**A,B, C, D, Wo**)**

$$s' = \frac{B_W}{\omega_0} \frac{s/\omega_0}{(s/\omega_0)^2 + 1} \tag{6.4}$$

Para los filtros paso-banda y banda-eliminada, $\omega_0 = (\omega_1 \cdot \omega_2)^{1/2}$, y $B_W = \omega_2 - \omega_1$, donde ω_1 y ω_2 corresponden al limite inferior y superior respectivamente

3º Discretización del filtro

- **bilinear**, **impinvar**: Estas dos funciones sirven para transformar filtros analógicos a filtros digitales. El primero se basa en la transformación bilineal y es el que más se utiliza, mientras que el segundo aplica un muestreo del filtro analógico (varianza impulsional). La notación en cada caso es la siguiente:

  Varianza impulsional

  » **[numd, dend] = impinvar(**num, den, fs**)**

  Transformación bilineal

  » **[numd, dend] = bilinear(**num, den, fs, fp**)**
  » **[zd, pd, kd] = bilinear(**z, p, k, fs, fp**)**
  » **[Ad, Bd, Cd, Dd] = bilinear(**A, B, C, D, fs, fp**)**

  En todos los casos, el parámetro ***fs*** indica la frecuencia de muestreo del filtro. En el caso de la transformación bilineal, además de poder trabajar con diferentes representaciones de la función de transferencia en tiempo continuo, ***fp*** es un valor que se utiliza para indicar la frecuencia en tiempo continuo ($\Omega = 2 \cdot \pi \cdot fp$) que se ha de corresponder con la frecuencia en tiempo discreto ($\omega = 2 \cdot \pi \cdot fp/fs$).

Ejemplo de diseño de un filtro digital a partir de filtros analógicos

En esta sección, vamos a mostrar una manera posible de utilizar las funciones que se acaban de explicar para diseñar un filtro digital a partir de especificaciones analógicas. El filtro a diseñar en un filtro paso-banda de Chebyshev (I) de orden 10 con un máximo de 3dB de rizado en la banda de paso y frecuencias de corte $\Omega_1 = \pi/5$ y $\Omega_2 = \pi$.

Primero se crea la función base:

» [z,p,k] = cheb1ap(5,3**);**

Las salidas *z, p, k* contienen los ceros, los polos y la ganancia de un filtro paso bajo con una frecuencia de corte (Ω_o) de 1 rad/seg. A continuación hay que utilizar la función ***lp2bp*** para transformar el filtro paso bajo a otro filtro analógico paso-banda con las frecuencias de corte especificadas, pero antes, habrá que convertir el filtro normalizado en el formato de *espacio de estados* para que dicha función lo pueda aceptar. Para ello, se utiliza la función ***zp2ss***:

» [A,B,C,D] = zp2ss(z,p,k**);**

Ahora, se calcula el ancho de banda Bw y la frecuencia de corte central del filtro Wo, y se procede a llamar a la función ***lp2bp***:

» u1 = 0.1*2*pi; **u2** = 0.5*2*pi;
» Bw = u2-u1;
» Wo = sqrt(u1*u2);
[At,Bt,Ct,Dt] = lp2bp(A,B,C,D,Wo,Bw**)**;

Y para concluir la obtención del filtro analógico, se obtiene la función de transferencia (en Laplace) y se dibuja su respuesta frecuencial (Fig.6.1)

» [b,a] = ss2tf(At,Bt,Ct,Dt**)**; % Convierte de espacio de estado a función en Laplace
» w = linspace(0.01,1,500**)***2*pi;
» h = freqs(b,a,w**)**;
» semilogy(w/2/pi,**abs(**h**))**, **grid**
» xlabel('Frequency (Hz)'**)**;

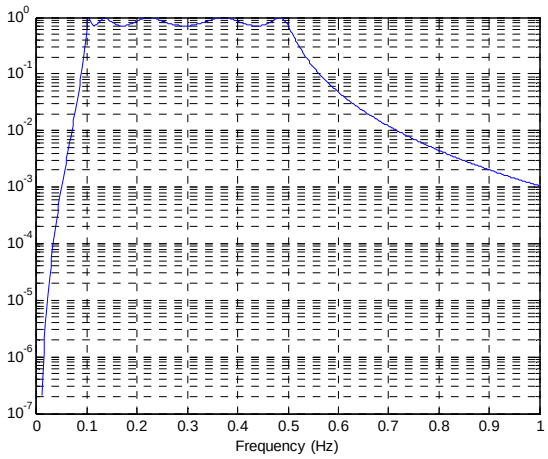


Figura 6.1.- Respuesta frecuencial del filtro de Chebyshev paso-banda en el dominio analógico. R_p= 3dB, Ω_1 = 0.1Hz y , Ω_2 = 0.5Hz

El tercer y último paso consiste en la transformación del filtro analógico en el dominio digital. En el caso de la transformación con la *varianza impulsional*, para una frecuencia de muestreo de 2Hz se ejecuta la función **impinvar** y se representa el filtro discreto mediante la herramienta **fvtool**:

» [bz,az] = impinvar(b,a,2**);**
» fvtool(bz,az**);**

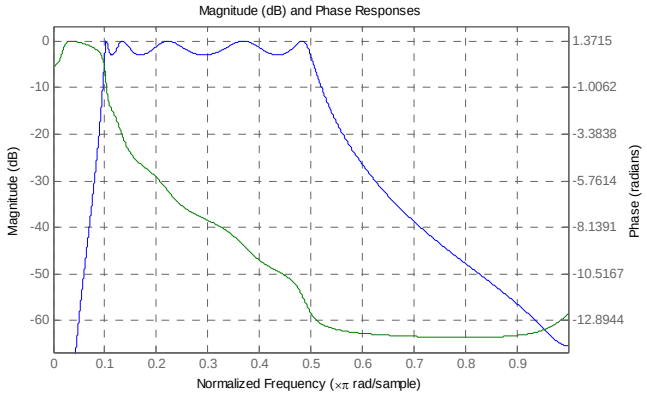


Figura 6.2.- Respuesta frecuencial del filtro digital después de realizar la conversión mediante el método de la varianza impulsional. Después de la conversión, este método mantiene las frecuencias de corte del filtro paso-banda: 0.1Hz i 0.5Hz, respectivamente.

En el mismo caso, con *transformación bilineal* hay que ejecutar:

» [Ad, Bd, Cd, Dd] = bilineal(A, B, C, D, 2, 0.1**);**

La respuesta frecuencial mediante este método se obtiene como sigue:

» [bz, az] = ss2tf(Ad, Bd, Cd, Dd**);** % conversión de espacio de estados a función de transferencia
» fvtool(bz,az**);**

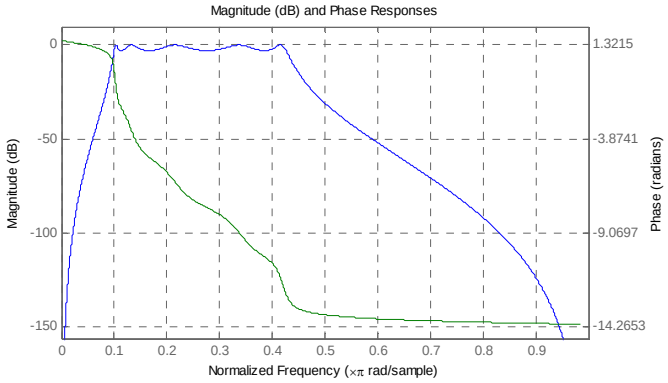


Figura 6.3.- Respuesta frecuencial del filtro digital después de realizar la conversión mediante el método de la transformación bilineal.

Como se puede observar en la figura 8, la frecuencia de corte inferior es la que se espera (0.1Hz) ya que en la función se ha especificado que esta frecuencia analógica se corresponda con su equivalente en digital. Sin embargo, la frecuencia de corte superior (0.5Hz) es ligeramente inferior. Esto refleja la naturaleza no lineal que tiene dicho método de conversión. Una manera de contrarrestar este efecto consiste en crear filtros analógicos pre-modulados en la banda de paso. De este modo, al aplicar la conversión, ambas frecuencias quedan emparejadas. Para ello habría que ejecutar las siguientes líneas:

» **fs** = 2; % Frecuencia de muestreo
» **u1** = 2*fs***tan(**0.1*(2*pi/fs)/2**)**; % Límite inferior
» **u2** = 2*fs***tan(**0.5***(**2*pi/fs)/2**)**; % Límite superior
» **Bw** = u2 - u1; % Ancho de la banda pasante
» **Wo** = **sqrt(**u1*u2**)**; % Frecuencia central
» **[At,Bt,Ct,Dt]** = **lp2bp(**A,B,C,D,Wo,Bw**)**; % Transformación a paso-banda del filtro base, pero con las nuevas frecuencias de corte.
» **[Ad,Bd,Cd,Dd]** = **bilinear(**At,Bt,Ct,Dt,fs**)**;
» **[bz,az]** = **ss2tf(**Ad,Bd,Cd,Dd**);**
» **fvtool(**bz,az**)**

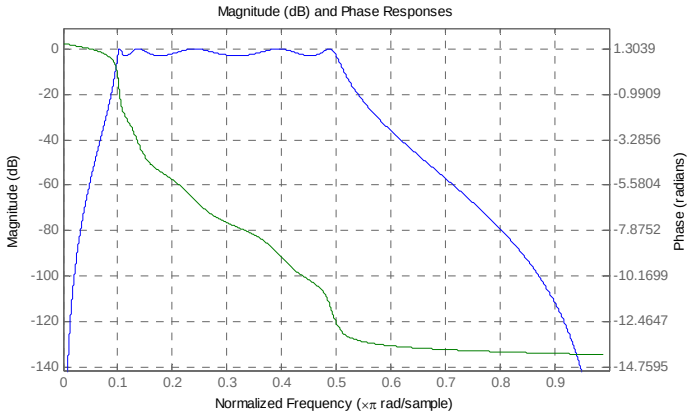


Figura 6.4.- Respuesta frecuencial del filtro digital obtenido a partir de la transformación lineal alterando las frecuencias de corte de filtro analógico

7 La herramienta FDATool de Matlab

La herramienta *FDATool* es un entorno interactivo de Matlab que permite diseñar, crear y analizar filtros analógicos y digitales de forma sencilla y rápida. Para poder utilizar esta herramienta, hay que escribir el comando correspondiente en el entorno de trabajo de Matlab. Tras esta acción, se abre la herramienta con el panel de diseño de la figura 10:

» fdatool;

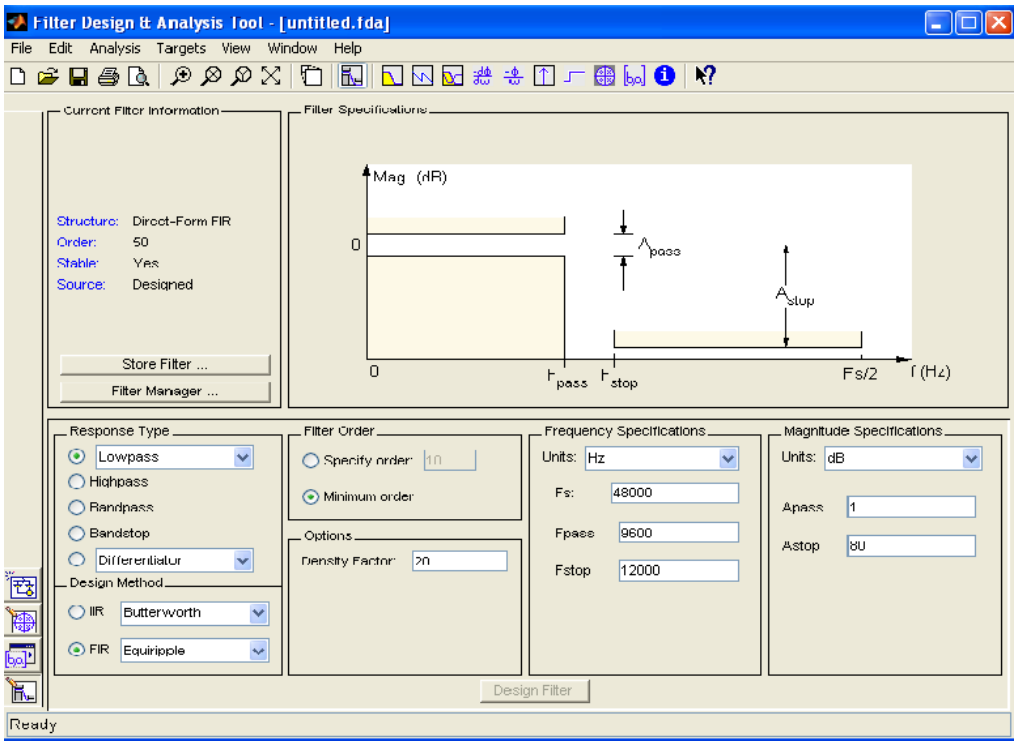


Figura 7.1.- Pantalla principal de la herramienta de diseño de filtros *FDATool*

De manera resumida, existe una zona central donde se representa la respuesta frecuencial del filtro escogido, con sus diferentes parámetros, y otra zona en la parte inferior donde se configuran las diferentes especificaciones. Los pasos más básicos que hay que realizar son: 1) Configurar las especificaciones de diseño y su correspondiente implementación; 2) Analizar el sistema obtenido para validar el diseño; 3) Exportar datos

- **Diseño e implementación**

Para realizar un diseño, básicamente hay que configurar dichos parámetros en los diferentes menús y 'clickar' después el botón ***Design Filter***. Podéis ver un ejemplo de diseño, configurando las especificaciones que se indican en la figura 11. Cuando se aplica el diseño, el programa determina los coeficientes de la función de transferencia, los cuales, quedan guardados en memoria. También notaréis que después de haber implementado el filtro el botón *Design Filter* queda deshabilitado hasta que no se realiza alguna modificación adicional

- **Validación del diseño**

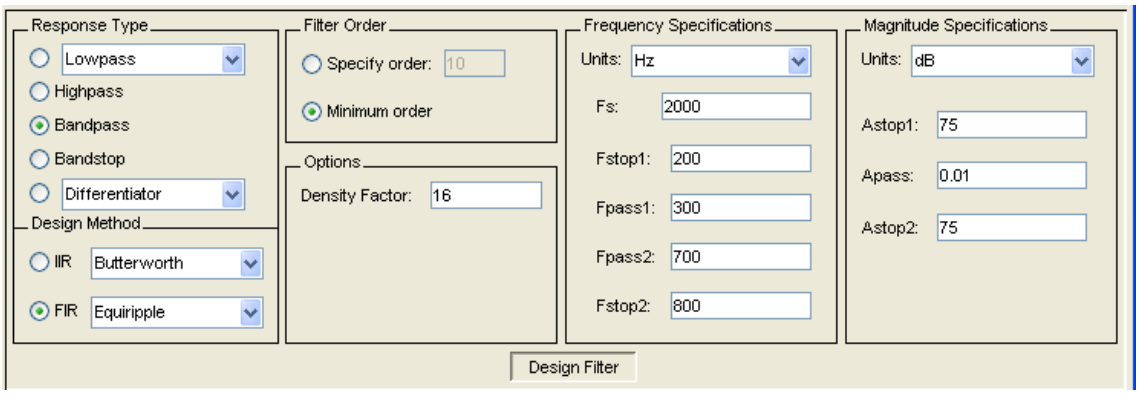


Figura 11.- Ejemplo de configuración de un filtro paso banda.

Después de diseñar el filtro, las características de su respuesta (Magnitud, fase, respuesta impulsional, grupo de retardo, coeficientes, etc) se pueden analizar en ventanas separadas (con **FVTool**), o bién dentro del propio entorno en el menú: **Analysis**. Por ejemplo, para ver la magnitud y la respuesta en frecuencia hay que hacer **Analysis > Magnitude Response** o apretar al botón correspondiente. Además se pueden superponer las especificaciones anteriormente indicadas al filtro obtenido (Fig. 12)

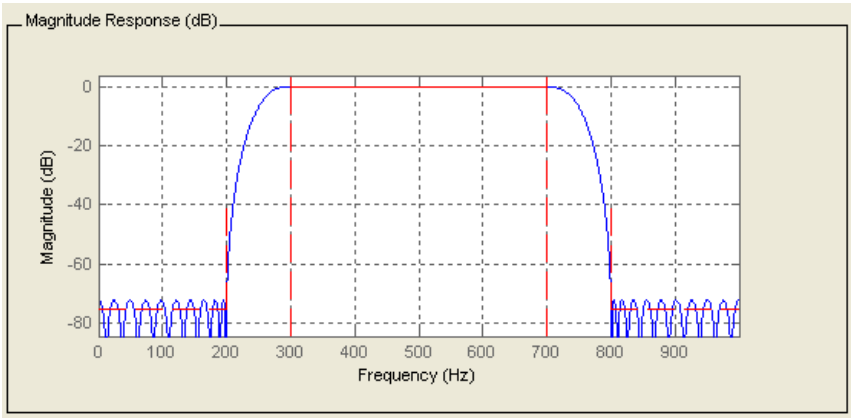


Figura 12.- Respuesta frecuencial del filtro paso bajo diseñado. Las especificaciones vienen marcadas en color rojo y trazo discontinuo.

- **Exportando el diseño**

Existen varias maneras de exportar el diseño a otros entornos dependiendo de las necesidades. Para exportar los datos al entorno de trabajo de *Matlab* hay que realizar los siguientes pasos:

1. Seleccionar: **File > Export** (Aparece el cuadro de diálogo de la figura 13)
2. Seleccionar *Workspace* en la lista de selección: **Export to**
3. **Seleccionar** *Coefitients* en el caso de querer exportar los coeficientes del filtro, o bien *Object* para guardar el filtro en un fichero, del menú: **Export As**

4. En este paso hay que asignar los valores obtenidos a las variables del entorno de trabajo de Matlab. Si en el apartado anterior se ha seleccionado la opción *Coefitients* existen tres posibilidades:

    i. **Filtros FIR**: Hay que asignar una variable con los coeficientes del numerador (*num*)
    ii. **Filtros IIR**: Hay que asignar las variables tanto para el numerador (num) como el denominador (den)
    iii. **Filtros IIR con estructuras de 2º orden**.- Hay que asignar dos variables (*SOS Matrix* y *Scale Value*)

    En el caso de seleccionar la opción *Object* simplemente hay que indicar el nombre que contiene el filtro digital

5. Por último, 'clickar' en el botón **Export** para generar las variables correspondientes.

Por último, comentar que dichas variables también se pueden guardar en ficheros con la extensión '*.mat' accediendo al menú **File > Generate M-File**

Actividad práctica de laboratorio nº 6

El trabajo a desarrollar en esta práctica consistirá en diseñar, una serie de filtros **analógicos** a partir de unas determinadas especificaciones con la herramienta *Matlab*. Utilizando las herramientas disponibles en Matlab, la última actividad práctica consistirá en diseñar e implementar el algoritmo de un filtro digital. Cada grupo tiene asignado su propio filtro cuyas especificaciones en el dominio analógico se pueden encontrar más abajo.

P1. **Diseño del filtro.** Diseñe el filtro que le halla sido asignado. En el diseño, no existen limitaciones respecto a la estructura de filtro a utilizar (Butterworth, Chebyshev (I o II), Cauer, etc). Básicamente hay que procurar que, en cualquier caso, se cumplen las especificaciones indicadas.

Nota: Se recomienda seguir el siguiente proceso:

1. Escoger una estructura que mejor se adapte a las especificaciones
2. Determinar el orden que se requiere para el filtro
3. Obtener el filtro paso-bajo normalizado
4. Transformar el filtro y obtener la función de transferencia en el dominio *analógico*

Nota: En caso de tener dudas considerables para el diseño, utilice la herramienta *FDATOOL*.

P2. **Conversión a digital.** Transforme la función de transferencia en el dominio analógico obtenida en el anterior apartado al dominio digital mediante la *transformación bilineal*, sólo en el caso de no haber utilizado la herramienta *FDATOOL*. Justifique el tiempo de muestreo T a escoger a través de las especificaciones del filtro

P3. **Realización del filtro.** Implemente el programa correspondiente al filtro digital obtenido. Para ello, cree una función donde a partir de los coeficientes del filtro (a y b), junto con una señal de entrada $x(n)$ calcule la salida $y(n)$

Represente el diagrama de bloques del filtro obtenido en la forma directa II transpuesta

P4. **Test y verificación.** Cree una señal de entrada digital $x(n)$ con tres componentes sinusoidales:

$$x(n) = A_1 \cdot sen\,(\omega_1 \cdot t) + A_2 \cdot sen\,(\omega_2 \cdot t) + A_3 \cdot sen\,(\omega_3 \cdot t)$$

con $\omega_m = 2 \cdot \pi \cdot F_m / T$, donde F_m corresponde a la frecuencia de la componente m de la señal y T es el mismo periodo de muestreo ($T=1/F_s$) utilizado en la discretización del filtro. La frecuencia de las componentes sinusoidales que se han de ser tal que una de ellas se sitúe dentro de la banda de paso y el resto fuera de ella.

Represente la salida del sistema *y(n)* con la función creada, o bien utilice la función ***filter()*** que dispone Matlab para el mismo propósito. Comente los resultados

Nota: Para dar soporte a los comentarios realizados, se recomienda realizar la misma verificación con la función de transferencia obtenida en el dominio analógico

Especificaciones de los filtros:

G1) Tipo de filtro: *Banda-eliminada*

- Frecuencia de corte inferior: F_{l1} = **200Hz**
- Frecuencia de corte superior F_{l2} = **400Hz.**
- Frecuencia de corte inferior: F_{h1} = **600Hz**
- Frecuencia de corte superior F_{h2} = **1KHz.**
- Rizado de la banda de paso R_p = **2dB.**
- Rizado de la banda de atenuación: R_s = **80dB**

G2) Tipo de filtro: *Paso-banda*

- Frecuencia de corte inferior: F_{l1} = **1.5KHz**
- Frecuencia de corte superior F_{l2} = **2.KHz.**
- Frecuencia de corte inferior: F_{h1} = **5.KHz**
- Frecuencia de corte superior F_{h2} = **6.KHz.**
- Rizado de la banda de paso R_p = **0.1dB.**
- Rizado de la banda de atenuación: R_s = **40dB**

G3) Tipo de filtro: *Paso-banda*

- Frecuencia de corte inferior: F_{l1} = **0.15Hz**
- Frecuencia de corte superior F_{l2} = **0.2Hz.**
- Frecuencia de corte inferior: F_{h1} = **0.9Hz**
- Frecuencia de corte superior F_{h2} = **1Hz.**
- Rizado de la banda de paso R_p = **2dB.**
- Rizado de la banda de atenuación: R_s = **70dB**

G4) Tipo de filtro: *Paso-bajo*

- Frecuencia de corte inferior: F_{u1} = **12Hz**
- Frecuencia de corte superior F_{u2} = **20Hz.**
- Rizado de la banda de paso: R_p = **1dB**

G5) Tipo de filtro: *Paso-alto*

- Frecuencia de corte inferior: F_{l1} = **20KHz**
- Frecuencia de corte superior F_{l2} = **22KHz.**
- Rizado de la banda de atenuación: R_p = **70dB**