Análisis y Síntesis de Circuitos con MATLAB®

1. Introducción

Este tutorial ha sido desarrollado a partir del libro de Edward Kamen y Bonnie Heck, *Fundamentals of Signals and Systems Using Matlab*, publicado por Prentice Hall. Aquí pretendemos cubrir los comandos básicos de MATLAB necesarios para introducirnos en el análisis y síntesis de circuitos. Para más información sobre los comandos tendremos que acudir a los manuales oficiales de MATLAB.

MATLAB es, conceptualmente, un lenguaje de programación de alto nivel. Contiene amplias librerías de funciones matemáticas que nos permitirán operar con matrices y obtener representaciones gráficas de datos. Iniciamos MATLAB mediante el correspondiente icono en el escritorio de Windows-98 o mediante el item correspondiente en el menú de inicio. Acto seguido se abrirá una ventana que nos permitirá ir introduciendo datos y comandos de manera interactiva. Detrás de cada comando debemos pulsar la tecla de retorno ('return', 'enter' o '↵') para que este sea ejecutado. Como veremos más adelante podemos leer los datos de un fichero, o incluso leer una secuencia de comandos. Para salir de MATLAB teclearemos '`exit`'.

2. MATLAB Básico

2.1 Definición de variables

Asignamos valores numéricos a las variables simplemente tecleando las expresiones correspondientes:

```
a = 1+2
```

lo cual resulta:

```
a =
     3
```

Si colocamos `;` al final de la expresión, el resultado se almacena en `a` pero no aparece en pantalla. Por ejemplo teclear `a = 1+2;`.

MATLAB utiliza los siguientes operadores aritméticos:

`+`	suma
`-`	resta
`*`	multiplicación
`/`	división
`^`	potencia
`'`	transposición

Una variable puede ser asignada mediante una fórmula que emplee operadores aritméticos, números o variables previamente definidas. Por ejemplo, como `a` estaba definida de antemano, la siguiente expresión es válida:

```
b = 2*a;
```

Para visualizar o recordar el valor de una variable que ha sido previamente asignada basta con teclearla de nuevo. Si tecleamos:

```
b
```

obtenemos:

```
b =
     6
```

Si la expresión no cabe en una línea de la pantalla, podemos utilizar una elipsis, esto es, tres o mas puntos suspensivos:

```
c = 1+2+3+...
5+6+7;
```

Existen algunas variables predefinidas en MATLAB, por ejemplo:

`i`	sqrt(–1)
`j`	sqrt(–1)
`pi`	3.1416...

o sea que, si introducimos:

```
y= 2*(1+4*j)
```

tendremos

```
y=
       2.0000 + 8.0000i
```

Existe también una serie de funciones predefinidas que pueden ser empleadas para asignar valores a nuevas variables, por ejemplo:

`abs`	valor absoluto de un número real o módulo de un número complejo
`angle`	fase de un número complejo en radianes
`cos`	función coseno, con el argumento en radianes
`sin`	función seno, con el argumento en radianes
`exp`	función exponencial

Por ejemplo, con la `y` definida anteriormente:

```
c = abs(y)
```

resulta en

```
c =
     8.2462
```

```
c = angle(y)
```

resulta en

```
c =
     1.3258
```

y con `a=3` como definimos antes:

```
c = cos(a)
```

resulta en

```
c =
          -0.9900

     c = exp(a)
```

resulta en

```
c =
          20.0855
```

Nótese que `exp` puede usarse con números complejos. Por ejemplo, para el `y = 2+8i` definido anteriormente:

```
     c = exp(y)
```

resulta en

```
c =
          -1.0751 + 7.3104i
```

2.2 Definición de matrices

MATLAB está basado en el álgebra de vectores y matrices, incluso los escalares son considerados matrices de 1×1 elementos. Así que las operaciones entre vectores y matrices son tan simples como las operaciones de cálculo comunes que ya hemos revisado.

Los vectores pueden definirse se dos formas. Por una lado, podemos definirlos explícitamente, introduciendo los valores de los elementos:

```
v = [1 3 5 7];
```

este comando crea un vector de dimensiones 1×4 con los elementos 1, 3, 5 y 7. Podemos usar comas para separar los elementos. Además, podemos añadir elementos al vector, teclear:

```
v(5) = 8;
```

resulta en el vector `v = [1 3 5 7 8]`. Los vectores definidos con anterioridad pueden servirnos para definir nuevos vectores, por ejemplo:

```
a = [9 10];
b = [v a];
```

origina el vector `b = [1 3 5 7 8 9 10]`.

El otro método se utiliza para definir vectores con elementos equi-espaciados:

```
t = 0:.1:10;
```

origina un vector de dimensiones 1×101 con los elementos 0, .1, .2, .3,...,10. Nótese que en la definición de `t`, el número que aparece en medio define el incremento de un elemento al siguiente. Si sólo tenemos dos números, el incremento por defecto es 1. Así:

```
k = 0:10;
```

da lugar a un vector de dimensiones 1×11 con los elementos 0, 1, 2, ..., 10.

Las matrices se definen introduciendo los elementos fila a fila. Por ejemplo:

```
M = [1 2 4; 3 6 8];
```

origina la matriz

$$M = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 6 & 8 \end{bmatrix}$$

Hay varias matrices especiales que pueden ser definidas mediante:

matriz nula:	`M = [];`
matriz de $n \times m$ ceros:	`M = zeros(n,m);`
matriz de $n \times m$ unos:	`M = ones(n,m);`
matriz identidad $n \times n$:	`M = eye(n);`

Podemos asignar valores a un elemento de una matriz en concreto, por ejemplo:

```
M(1,2) = 5;
```

coloca un 5 en la primera fila, segunda columna.

Las operaciones y funciones antes definidas para escalares pueden usarse también con vectores y matrices. Por ejemplo, si introducimos:

```
a = [1 2 3];
b = [4 5 6];
c = a + b
```

obtenemos:

```
c =
     5     7     9
```

Las funciones se aplican elemento a elemento:

```
t = 0:10;
x = cos(2*t);
```

origina un vector `x` cuyos elementos valen $\cos 2t$ para $t = 0, 1, 2, \ldots, 10$.

A veces necesitamos que las operaciones se realicen elemento a elemento. Para ello precedemos el operador correspondiente de un punto ".". Por ejemplo, para obtener un vector `x` que contenga como elementos los valores de $x(t) = t \cdot \cos t$ para unos instantes de tiempo determinados, no podemos multiplicar simplemente el vector `t` por el vector `cos(t)`. Lo que hacemos es:

```
t = 0:10;
x = t.*cos(t);
```

2.3 Ficheros M (*M-files*)

Los ficheros M (*M-files*) son macros de comandos de MATLAB almacenadas como ficheros de texto con extensión "`.m`", o sea `nombre_de_fichero.m`. Un M-file puede ser una función con variables de entrada y salida o simplemente una lista de comandos (un *batch* o *script* de comandos de MATLAB). Para usar M-files en un PC, MATLAB requiere que dicho M-file se encuentre en el directorio de trabajo (teclear `pwd` para saber cual es o `cd` para cambiar de directorio de trabajo), o bien en un directorio que debe estar especificado en la lista de *paths* de MATLAB. Por ejemplo, si tenemos los M-files que vamos a utilizar en un directorio llamado `D:\matlab\mfiles` entonces, para acceder a estos ficheros tendremos que hacer `cd D:\matlab\mfiles` desde la ventana de comandos de MATLAB o necesitaremos añadir dicho directorio a la lista de *paths* de MATLAB. Para añadirlo de forma permanente editaremos el fichero `D:\matlab\matlabrc.m` mientras que añadirlo de forma temporal se hace tecleando `path(path,'D:\matlab\mfiles')` desde la ventana de comandos.

Como ejemplo, crear un fichero en el directorio de trabajo de nombre `yplusx.m` que contenga los siguientes comandos:

```
function z = yplusx(y,x)
z = y + x;
```

Si ahora nos vamos a la ventana de comandos y tecleamos:

```
x = 2;
y = 3;
z = yplusx(y,x)
```

tendremos a la salida:

```
z =
      5
```

Para obtener ficheros M más eficientes, tendremos que escribirlos tratando de hacer uso de las operaciones en forma matricial. Aunque disponemos de bucles y de elementos sintácticos para establecer comparaciones, estos son computacionalmente ineficientes (MATLAB es un lenguaje interpretado, no compilado, al menos en un principio) por lo que si podemos evitar usarlos tendremos un menor tiempo de ejecución. Un ejemplo del uso del comando `for` sería:

```
for k=1:10,
   x(k) = cos(k);
end
```

Esto da lugar a un vector de 1×10 elementos conteniendo el coseno de los números enteros positivos del 1 al 10. Esta operación puede realizarse más eficientemente así:

```
k = 1:10;
x = cos(k);
```

donde utilizamos una función de un vector en lugar de un bucle.

Para establecer comparaciones (sentencias condicionales) se emplea el comando `if`. Por ejemplo:

```
if(a <= 2),
  b = 1;
elseif(a >=4)
  b = 2;
else
  b = 3;
end
```

los comparadores permitidos son `>=` (mayor o igual que), `<=` (menor o igual que), `<` (mayor que), `>` (menor que), `==` (igual que) y `~=` (distinto de).

Es posible también solicitar información del usuario desde un programa o función en un *M-file*, mediante el comando `input`. Si hacemos:

```
T = input('Input the value of T: ')
```

aparecerá en la ventana de comandos el mensaje:

```
Input the value of T: ?
```

a lo que deberemos responder con un valor apropiado para que el programa continue su ejecución.

2.4 Análisis de Fourier

MATLAB está equipado con una serie de funciones especiales que nos van a permitir realizar un análisis de Fourier de funciones definidas por un conjunto de valores discretos. Por ejemplo, el comando `fft` nos permite obtener la transformada rápida de Fourier (*Fast Fourier Transform*) de una secuencia de números x[n] definida por el vector `x`. Por ejemplo:

```
X = fft(x);
```

Si queremos que sea más eficiente en el cálculo de la FFT, la longitud del vector `x` deberá ser una potencia de 2, o sea 64, 128, 256, 512, 1024, 2048, etc. Podemos rellenar de ceros el vector `x` para que tenga la longitud apropiada. Esto se consigue automaticamente haciendo:

```
X = fft(x,N);
```

donde `N` es el exponente de 2. Mientras más largo sea `x`, más fina será la escala para la FFT. Debido a un fenómeno de plegamiento del espectro, sólo la primera mitad de los puntos obtenidos serán de utilidad. El comando `ifft` sirve para obtener la transformada inversa:

```
x = ifft(X);
```

2.5 Información general

MATLAB detecta las mayúsculas y minúsculas como diferentes, de modo que "`a`" y "`A`" serán dos variables distintas.

Las líneas de comentario en los programas deben estar precedidas de "`%`"

Mediante el comando `help` podemos obtener ayuda on-line. Si tecleamos `help` aparecerá todo un menú de temas sobre los que existe la ayuda y si tecleamos `help` seguido del nombre de una función o de un *M-file* recibiremos ayuda específica para dicha función.

El número de dígitos con que MATLAB representa los números en pantalla no está relacionado con la precisión con que estos han sido calculados. Para cambiar el formato de pantalla teclearemos '`format short e`' si queremos notación científica con 5 cifras significativas, '`format long e`' para notación científica con 15 cifras significativas y '`format bank`' para tener sólo dos dígitos decimales.

Los comandos `who` y `whos` nos dan los nombres de las variables definidas actualmente en el espacio de trabajo (*workspace*).

El comando `length(x)` nos da la longitud del vector `x` y `size(x)` las dimensiones de la matriz `x`.

2.6 Salvar y recuperar datos desde un fichero

Es muy probable que cuando usemos MATLAB estemos interesados en guardar los vectores y matrices que hemos creado. Para hacer esto sólo tenemos que hacer:

```
save nombre_del_fichero
```

y para recuperar dichos datos en otra sesión

```
load nombre_del_fichero
```

En general no vamos a salvar todos los vectores y matrices generados sino que guardaremos sólo los que nos interesen mediante:

```
save variables,de,interes nombre_del_fichero
```

Para más información hacer `help save`.

3. Análisis de sistemas en tiempo continuo

3.1 Representación de la función del sistema

En MATLAB podemos definir funciones de transferencia almacenando los coeficientes del numerador y del denominador en sendos vectores. Supongamos la función de transferencia siguiente:

$$H(s) = \frac{N(s)}{D(s)} = \frac{a_m s^m + a_{m-1} s^{m-1} + \ldots + a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} + \ldots + b_1 s + b_0}$$

para esta función definimos los vectores `num` y `den` (por ejemplo) como $[a_m, a_{m-1}, \ldots$ $\ldots, a_1, a_0]$ y $[b_n, b_{n-1}, \ldots, b_1, b_0]$, respectivamente. O sea, para la función:

$$H(s) = \frac{2s + 3}{s^3 + 4s^2 + 5}$$

definiremos:

```
num = [2 3];
den = [1 4 0 5];
```

Nótese que incluso los coeficientes que son cero deben incluirse en el vector que define al polinomio en s. A continuación podemos definir una variable del espacio de trabajo `sys1` que será la función de transferencia del sistema, mediante el comando `tf`. Tecleando:

```
sys1=tf(num,den)
```

obtendremos:

```
Transfer function:
      2 s + 3
-----------------
3 s^3 + 4 s^2 + 5
```

Por otro lado, una función de transferencia pude definirse a partir de sus polos, sus ceros y una constante multiplicativa (ganancia para $s \to \infty$):

$$H(s) = \frac{k(s-z_1)(s-z_2)\ldots(s-z_m)}{(s-p_1)(s-p_2)\ldots(s-p_n)}$$

si comparamos esta expresión con la anterior vemos que $k = a_m / b_n$. De manera análoga a como hacíamos antes, podemos definir los vectores `z` y `p` como $[z_1, z_2, \ldots, z_m]$ y $[p_1, p_2, \ldots, p_n]$ respectivamente y por otro lado la constante `k`. Así:

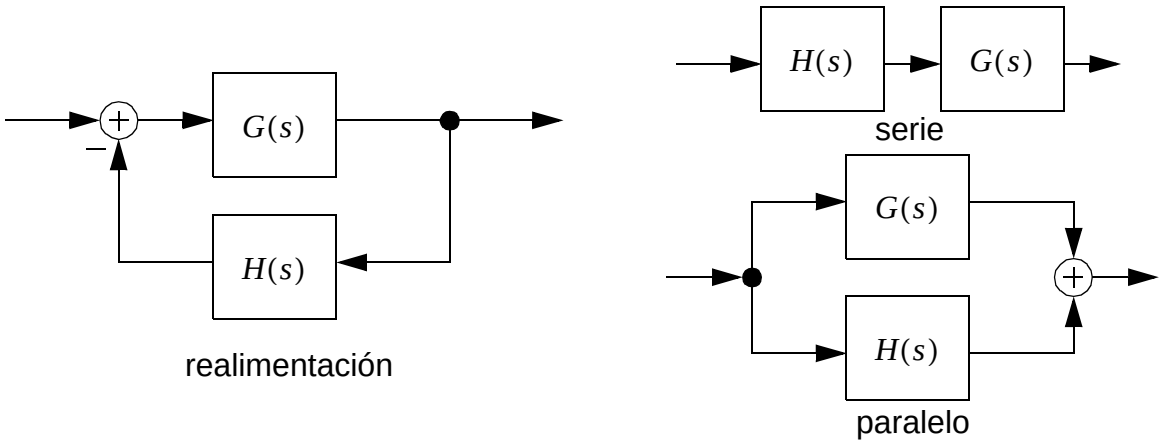
```
z = [1 2];
p = [0 3 5];
k = 3;
```

Ahora definimos la función de sistema `sys2` mediante el comando `zpk`. Tecleando:

```
sys2=zpk(z,p,k)
```

obtendremos:

```
Zero/pole/gain:
```




```
3 (s-1) (s-2)
-------------
s (s-3) (s-5)
```

Podemos asimismo encontrar los polos y ceros de una función de transferencia, a partir de los polinomios en s que constituyen el numerador y el denominador, mediante el comando `tf2zp`. Así si hacemos:

```
[z1,p1,k1] = tf2zp(num,den)
```

obtenemos:

```
z1 =
      -1.5000

p1 =
      -1.8307
      0.2487 + 0.9212i
      0.2487 - 0.9212i

k1 =
      0.6667
```

También podemos obtener facilmente los polinomios del numerador y denominador a partir de la descripción en ceros, polos y ganancia mediante `zp2tf`. Si hacemos:

```
[num2,den2] = zp2tf(z,p,k)
```

obtenemos:

```
num2 =
          0     3    -9     6

den2 =
          1    -8    15     0
```

En la figura de arriba podemos ver un esquema genérico de realimentación y también la conexión en serie y en paralelo de dos funciones de transferencia. Podemos reducir el sistema realimentado a una sola función de transferencia $G_{cl}(s)$, que viene dada por:

$$G_{cl}(s) = \frac{G(s)}{1 + G(s)H(s)}$$

Con MATLAB, esta puede obtenerse fácilmente a partir del comando `feedback`. Consideremos que `numG` y `denG` definen a la función $G(s)$, y que `numH` y `denH` definen a $H(s)$. El numerador y el denominador de la función $G_{cl}(s)$ se obtiene haciendo:

```
[numcl,dencl] = feedback(numG,denG,numH,denH);
```

o bien, mediante las funciones de sistema `sysH` y `sysG`, obtenemos `syscl` mediante:

```
syscl = feedback(sysH,sysG);
```

Para reducir la asociación en serie a una sola función de transferencia $G_s(s) = G(s)H(s)$ utilizamos el comando `series`:

```
[numS,denS] = series(numG,denG,numH,denH);
```

o bien:

```
sysS = series(sysH,sysG);
```

Para reducir la asociación en serie a una sola función de transferencia $G_p(s) = G(s) + H(s)$ utilizamos el comando `parallel`:

```
[numP,denP] = parallel(numG,denG,numH,denH);
```

o bien:

```
sysP = parallel(sysH,sysG);
```

3.2 Respuesta en el tiempo

El método analítico para determinar la respuesta temporal de un sistema es calcular la tranformada inversa de Laplace de la transformada de la salida $Y(s)$. MATLAB contribuye a esta operación computando la expansión parcial en fracciones simples de $Y(s)$ mediante el comando `residue`. Por ejemplo, si el numerador y el denominador de $Y(s)$ están almacenados en los vectores `num` y *den*, al hacer:

```
[r,p,k] = residue(num,den)
```

obtenemos los valores de los residuos en el vector `r`, correspondientes a los polos que hay en el vector `p`. `k` es el término independiente de la expansión, coincide con el `k` anterior. Si usamos esta función al revés:

```
[num,den] = residue(r,p,k)
```

obtenemos los polinomios `num` y `den` a partir de la expansión en fracciones.Una vez que hemos calculado la expanión en fracciones de $Y(s)$ es fácil obtener una expresión analítica para $y(t)$ a mano.

Aparte de este método analítico, MATLAB dispone de un método numérico para evaluar la respuesta en el tiempo de un sistema para una entrada determinada. Por ejemplo, siendo ahora `num` y `den` el numerador y el denominador de una función de transferencia $H(s)$, también definida por `sys=tf(num,den)`, podemos visualizar la respuesta a un escalón mediante el comando `step`:

```
step(num,den)
```

o bien

```
step(sys)
```

o la respuesta impulsiva mediante

```
impulse(num,den)
```

o bien

```
impulse(sys)
```

Para obtener la respuesta a una entrada arbitraria emplearemos el comando `lsim`. En primer lugar necesitamos un vector `t` que contenga los instantes en que queremos evaluar la respuesta. Esto lo hacemos con:

```
t = a:b:c;
```

donde `a` es el tiempo de inicio de la simulación, `c` es el final y `b` es el paso de tiempo. Si no queremos ver escalones en la representación gráfica de la respuesta es conveniente tomar al menos 300 puntos. A continuación definimos la entrada `x` como una función del tiempo (p. ej. una rampa sería `x=t;`). La visualización de la salida se realiza mediante:

```
lsim(num,den,x,t);
```

También podemos almacenar las muestras de la salida en otra variable. Por ejemplo, consideremos la función de trasferencia

$$H(s) = \frac{2}{s+2}$$

obtenemos la respuesta al escalón mediante:

```
num = 2; den = [1 2];
t = 0:3/300:3;    % for a time constant of 1/2
y = step(num,den,t);
plot(t,y)
```

Para la respuesta impulsiva, basta con sustituir `step` por `impulse`. Para la respuesta a una entrada arbitraria:

```
y = lsim(num,den,x,t);
plot(t,y)
```

3.3 Respuesta en frecuencia

Supongamos que la función de transferencia $H(s)$ es decrita por `num` y `den` como hemos visto en los apartados anteriores. MATLAB permite calcular la respuesta en frecuencia $H(j\omega)$ mediante el comando `freqs`. Para ello definimos un vector de frecuencias `w = a:b:c`, en el que `a` es la frecuencia de inicio del barrido, *c* es la final y `b` el incremento, de manera que el comando:

```
H = freqs(num,den,w)
```

devuelve un vector complejo que corresponde al valor de $H(j\omega)$ para las frecuencias contenidas en `w`.

Para dibujar el diagrama de Bode de una funcion de transferencia tan sólo tenemos que teclear:

```
bode(num,den)
```

Si queremos adaptar el diagrama a nuestras necesidades —diferentes rangos de magnitud o fase y frecuencia, etc.— comenzaremos por definir un vector `w` que contenga el rango de frecuencias para el cual el diagrama será calculado. Como `w` debería ser definido en una escala logarítmica usaremos el comando `logspace`. Por ejemplo, para un diagrama de Bode que vaya desde los 10^{-1}Hz hasta los 10^{2}Hz definiremos `w` de esta forma:

```
w = logspace(-1,2);
```

La magnitud y la fase puden guardarse en sendos vectores:

```
[mag,phase] = bode(num,den,w);
```

Para visualizar la magnitud en decibelios, convertimos `mag` mediante:

```
magdb = 20*log10(mag);
```

y a continuación realizamos un diagrama semilogarítmico (lineal en el eje y y logarítmico en el eje x) de la magnitud:

```
semilogx(w,magdb)
```

y de la fase:

```
semilogx(w,phase)
```

El diagrama de la fase puede tener saltos de $\pm 2\pi$, lo cual no resulta nada conveniente, de modo que lo mejor es que utilicemos el comando `unwrap` antes de visualizar:

```
semilogx(w,unwrap(phase))
```

3.4 Diseño de filtros analógicos

Existen en MATLAB una serie de comandos que permiten calcular filtros analógicos con facilidad. Por un lado, podemos determinar el orden de un filtro que cumple determinadas especificaciones:

Para la aproximación de Butterworth:

```
[N, Wn] = buttord(Wp,Ws,Ap,As,'s')
```

Para la aproximación de Chebyshev:

```
[N, Wn] = cheby1ord(Wp,Ws,Ap,As,'s')
```

Para la aproximación de Chebyshev inversa:

```
[N, Wn] = cheby2ord(Wp,Ws,Ap,As,'s')
```

Para la aproximación elíptica:

```
[N, Wn] = ellipord(Wp,Ws,Ap,As,'s')
```

aquí `Wp` y `Ws` son las frecuencias de borde de la(s) banda(s) pasante(s) y de rechazo, `Ap` y `As` son la atenuación máxima permitida en la(s) banda(s) pasante(s) y la atenuación mínima exigida en la(s) banda(s) de rechazo. Es necesario incluir `'s'` al final para que MATLAB entienda que se trata de un filtro analógico. `N` representa el orden mientras que `Wn` se define como la frecuencia de corte, o sea, la frecuencia de 3dB.

Por otro lado, comandos como `buttap(n)` nos permiten obtener los polos y la constante k de un filtro prototipo paso de baja de Butterworth de orden n (por supuesto no tiene ceros por lo que `z=[]`) normalizado en frecuencia (es decir que el borde de la banda de paso está en $\omega_p = 1$. Por ejemplo:

```
[z,p,k] = buttap(5)
```

tiene como resultado:

```
z =
     []
p =
     -0.3090 + 0.9511i
     -0.3090 - 0.9511i
     -0.8090 + 0.5878i
     -0.8090 - 0.5878i
     -1.0000
k =
```

```
1
```

Para un filtro prototipo paso de baja de Chebyshev utilizaremos:

```
[z,p,k] = cheby1ap(n,Ap)
```

donde `n` es el orden del filtro y `Ap` es el rizado de la banda pasante en decibelios. Mientras que si queremos un prototipo paso de baja Chebyshev inverso emplearemos:

```
[z,p,k] = cheby2ap(n,As)
```

donde `n` es el orden del filtro y `As` es el rizado de la banda de rechazo en decibelios.

Finalmente si lo que buscamos es un filtro prototipo paso de baja elíptico tendremos que utilizar la función:

```
[z,p,k] = ellipap(n,Ap,As)
```

En todos los casos para obtener el numerador y denominador de la función de transferencia a partir de la salida de los comandos anteriores hacemos:

```
[num,den] = zp2tf(z,p,k)
```

También es posible, una vez que tenemos el prototipo paso de baja, realizar desnormalizaciones o transformaciones en frecuencia. Si nuestro filtro LPP tiene como numerador y denominador num y den respectivemente entonces podemos:

Pasar a otra frecuencia de corte `Wo` (desnormalizar):

```
[num2, den2] = lp2lp(num,den,Wo)
```

Pasarlo a un filtro paso de alta:

```
[num2, den2] = lp2hp(num,den,Wo)
```

Pasar a un filtro paso de banda con ancho de banda `Bw` centrado en `Wo`:

```
[num2, den2] = lp2bp(num,den,Bw, Wo)
```

Pasar a un filtro rechazo de banda con ancho de banda `Bw` centrado en `Wo`:

```
[num2, den2] = lp2bs(num,den,Wo)
```

4. Representación gráfica de datos y resultados

El comando más utilizado para la visualización de resultados es plot, que genera representaciones gráficas lineales de vectores y matrices. Por ejemplo:

```
plot(t,y)
```

representa los valores de `y` en el eje y frente a los de `t` en el eje x. Hay diferentes opciones para el tipo de línea, el color, etc. Haciendo `help plot` podremos ver cuales son. Por ejemplo, `plot(t,y,'--')` usa una línea discontínua, mientras que `plot(t,y,'*')` emplea asteríscos en todos los puntos y no los conecta. `plot(t,y,'g')` dibuja una línea verde sólida mientras que `plot(t,y,'g:')` pinta una línea verde de puntos.

Podemos también colocar dos gráficas juntas (en los mismos ejes) mediante `plot(t1,y1,t2,y2)`, que dibuja `y1` frente a `t1` e `y2` frente a `t2`.

Para colocar una etiqueta en los ejes o poner un título:

```
xlabel('time (sec)')
ylabel('step response')
```

```
title('My Plot')
```

y finalmente podemos añadir una cuadrícula para que se lea mejor con el comando `grid`.

En muchas ocasiones tendremos que adaptar los ejes a nuestras necesidades, esto se consigue mediante:

```
axis([xmin xmax ymin ymax]);
```

donde `xmin`, `xmax`, `ymin`, e `ymax` corresponden a los límites en ejes que queremos visualizar. Para retornar al autoescalado tecleamos `axis`.

Si queremos abrir más de una gráfica en una misma ventana utilizaremos el comando `subplot(m,n,p)` donde `p` indica cual de las $m \times n$ gráficas en las que se encuentra dividida la ventana que vamos a utilizar. Por ejemplo,

```
subplot(2,1,1),semilogx(w,magdb);
subplot(2,1,2),semilogx(w,phase);
```

representa un diagrama de Bode de la magnitud en el panel alto de la ventana y el de la fase en el panel inferior. Para volver a la pantalla completa hacemos `subplot(1,1,1)`.