

# **DSD**

Diseño de filtros digitales

© Roberto Ripio, 2012-2019

## Introducción

DSD quiere decir (por ejemplo) Digital Speaker Design. Es un conjunto de scripts de GNU-octave para el cálculo de filtros FIR de ecualización y corte de altavoces en recintos multivía, y filtros FIR de corrección de sala. Contiene las funciones específicas para el diseño de filtros digitales, y los scripts de diseño que las utilizan.

GNU-octave es un entorno de programación interpretada para cálculos matemáticos que puede usarse interactivamente o por medio de scripts. DSD se ejecuta dentro de ese entorno, lanzando los scripts de cálculo con el nombre del archivo de proyecto como argumento.

## Instalación

DSD puede ejecutarse en los sistemas operativos habituales. Las instrucciones a continuación se detallan para el sistema operativo Ubuntu-Linux, pero los procedimientos generales son semejantes en otros sistemas, y su usuario podrá trasladarlos fácilmente teniendo en cuenta también que el modo de indicar los directorios y la localización de los archivos de usuario pueden variar.

Los pasos son:

- Instalar GNU-Octave, más los paquetes de funciones necesarios.
- Instalar git.
- Instalar DSD con git.
- Dar de alta el directorio de DSD en la ruta de búsqueda de Octave.
- Dar de alta el directorio de DSD en la ruta de búsqueda de ejecutables del sistema operativo.

## Octave

En Ubuntu-Linux: `'sudo apt install octave'`.

Es necesario que estén instalados los siguientes paquetes de Octave: control, general, io, signal y statistics. Dependiendo del sistema operativo esto se consigue de formas diferentes. En Ubuntu-Linux, por ejemplo, se ejecutaría la orden:

```
'sudo apt install octave-control octave-general octave-io  
octave-signal octave-statistics'
```

Es posible que la distribución de octave para algunos sistemas operativos ya tenga los paquetes necesarios instalados. Puede comprobarse ejecutando en la línea de órdenes de Octave la orden `pkg list`.

Para instalar en otros S.O. consultar la página web de Octave: <https://www.gnu.org/software/octave/#install>.

## Git

En Ubuntu-Linux: `'sudo apt install git'`.

Para instalar en otros S.O. consultar la página web de git: <https://git-scm.com/downloads>.

## DSD

Cambiar al directorio bajo el que se quiera instalar DSD, por ejemplo: `cd /home/[user]`, donde [user] se cambiará por el nombre de usuario real.

Clonar con git el repositorio de DSD: `git clone https://github.com/rripio/DSD`.

## Ruta de octave

Abrir Octave, y en su línea de órdenes, escribir la secuencia:

```
addpath(genpath('/home/[user]/DSD/src'))  
savepath()
```

Esto guardará permanentemente la ruta de carga de los scripts de DSD.

## Ruta del sistema operativo

Editar el fichero de usuario `/home/[user]/.bashrc` y añadir:

```
# Añade DSD al path  
PATH=$PATH:/home/[user]/DSD/src
```

Recordamos que [user] se cambiará por el nombre de usuario real.

## Confección de los filtros

El procedimiento es el siguiente:

- Elaboración de los archivos de respuesta en frecuencia de los diferentes altavoces de la caja acústica, en formato **.frd**.
- Elección de las frecuencias de corte y del tipo de crossover en cada corte.
- Escribir los parámetros del filtro en los correspondientes archivos de guión (extensión **.xof**), uno por cada vía.
- Ejecución del programa 'dsdxof', con el archivo de proyecto como parámetro, y opcionalmente añadiendo como parámetros la frecuencia de muestreo y la clase de filtro.
- Revisar los niveles de atenuación producidos por el filtro para tenerlos en cuenta en la estructura de filtrado y ganancia.

Estos pasos se detallan a continuación.

### ***Elaboración de los archivos de respuesta en frecuencia***

El filtrado se elabora partiendo de una cierta respuesta en frecuencia del altavoz (o de la vía, en caso de que esta tenga dos o más altavoces) en formato **.frd**, cuya elaboración queda a cargo del diseñador, con las herramientas para medición y exportación que considere oportunas.

Se hace así para tener la máxima flexibilidad para promediar, o suavizar, o lo que fuere que entiende el diseñador que representa mejor el comportamiento del altavoz que va a filtrar y ecualizar.

Es importante tener en cuenta que si los niveles relativos entre los diversos altavoces se respetan en estas curvas, los niveles de atenuación que se obtengan con la herramienta DSD también podrán usarse para nivelar las vías en el programa de convolución que ejecute los filtros.

Debe tenerse en cuenta que el nombre de archivo empleado será el que se use para componer, junto con la clase de filtro, el nombre de los filtros de salida.

### ***Elección de las frecuencias de corte y del tipo de crossover***

Hay dos tipos principales de filtro: Filtros de fase lineal (linear phase, LP) y filtros de fase mínima (minimum phase, MP). Dentro de los de fase mínima podemos optar entre Butterworth de cualquier orden, y Linkwitz-Riley de cualquier orden par.

Debe tenerse en cuenta que los filtros de fase lineal imparten a la señal un retardo que es intrínseco al filtro (ie. no depende del hardware usado), por lo que no deben mezclarse filtros LP y MP en diferentes cortes de una misma caja acústica.

En los filtros MP es posible indicar varias frecuencias de corte pasaaaltos, a fin de añadir a una vía los retardos de las vías inferiores. Añadiendo los mismo pasaaaltos con los que se cortan estas reproducimos el efecto de un filtro activo en cascada.

### ***Confección de los archivos de proyecto***

La carpeta **example** del paquete contiene un archivo ejemplo **\_ejemplo.xof** que se puede copiar (este u otro ya elaborado) a una carpeta de trabajo, que será preferentemente aquella en que se tengan los archivos de medición, y editarlo a conveniencia.

Se han de cambiar el nombre a estos archivos por otros significativos, y se han de editar para asignarles los parámetros adecuados.

En cuanto a los nombres, el autor suele llamarlos por ejemplo **t.xof**. Es decir, con un prefijo que aluda a la vía (**t**, **m**, **w** por tweeter, mid, woofer), pero nada de esto es preceptivo, salvo la extensión del archivo (**.xof**).

## ***Opciones por defecto***

En el directorio de DSD está el archivo **dsdxof.ini**, con las opciones por defecto. Los archivos de guión específicos tienen precedencia sobre estas opciones. Un modo conveniente de operar es hacer archivos de guión que contengan solo los parámetros específicos que varíen respecto a las opciones por defecto.

Los parámetros de línea de órdenes tienen a su vez precedencia sobre lo anterior, de modo que pueden omitirse de los guiones la frecuencia de muestreo y la clase de filtro, a fin de emplear un solo guión para diversos filtros con los mismos cortes.

## ***Parámetros***

Se detalla a continuación el significado de los distintos parámetros, agrupados por el área de operación, que se expresa en el prefijo del nombre de cada parámetro:

### **GS, General Settings**

#### ***GSLExp***

Es la potencia de dos que expresa la longitud final del filtro. Es sabido que la operación FFT (transformada rápida de Fourier) requiere operandos cuya longitud sea una potencia de dos. Valores habituales serán *15* o *16*, que resultan en filtros de  $2^{15}$  o  $2^{16}$  coeficientes, respectivamente.

#### ***GSFs***

Es la frecuencia de muestreo para la que se diseña el filtro. Puede omitirse si se da como segundo parámetro al llamar al programa.

#### ***GSLExpCalc***

Es la potencia de dos que expresa la longitud de filtro usada en la fase de cálculo. En general será mayor que ***GSLExp*** para obtener mayor precisión.

### **FS File Settings**

#### ***FSInputFile***

Nombre del archivo **.frd** de la medición de la vía que corresponda.

#### ***FSNormFile***

Archivo con los valores de normalización de niveles, para la posterior aplicación de ganancias por vía. Por defecto **niveles.txt**.

### ***FSOutPcmFilter***

Valor booleano (*true* or *false*) que indica si se generan filtros en formato raw **.pcm**. El formato de los filtros es ‘float 32’.

### ***FSOutWavFilter***

Valor booleano (*true* or *false*) que indica si se generan filtros en formato **.wav**.

### ***FSOutWavDepth***

Profundidad de bits de los filtros en formato **.wav**.

## **CF Crossover Filter**

Debe tenerse en cuenta que, si bien los woofers o los tweeters no tienen tradicionalmente corte en el extremo de la banda, el filtrado digital permite o aconseja poner cortes en esos extremos, a criterio del diseñador, para algunas estrategias de ecualización de fase, o para mejorar los filtros de reconstrucción de los convertidores (apodizing filters).

### ***CFClass***

Clase de filtro (*lp* o *mp*). Puede omitirse si se da como tercer parámetro al llamar al programa.

### ***CFLengthFactor***

Para filtros *lp*, es el factor de longitud del impulso generado. Este factor multiplica al parámetro ***GSFs/CFLowF*** o ***GSFs/CFHighF*** (frecuencia de muestreo / frecuencia de corte), y permite generar filtros de igual pendiente a ambos extremos de la banda con la mínima longitud posible en cada extremo, y por tanto con mínimo pre-ringing.

En caso de que este factor sea demasiado alto pueden llegar a generarse filtros de menor pendiente en el lado izquierdo, si la longitud total del filtro no permite obtener la misma pendiente.

### ***CFLowType***

Para filtros *mp*, tipo del filtro pasaaltos (que filtrará el extremo izquierdo de la vía —low—). Puede ser *B* (Butterworth) o *LR* (LinkwitzRiley). No tiene efecto si ***CFClass*** es *lp*. Debe darse entre llaves, al tratarse de un “cell array”.

Puede darse como “cell array” de varios valores si se van a dar varios cortes pasaaltos para simular un filtro en cascada o para aplicar a un tweeter el retardo que provoca en el woofer su pasaaltos natural.

### ***CFLowOrder***

Es el orden del filtro pasaaltos *mp*. No tiene efecto si ***CFClass*** es *lp*.

Puede darse como vector de varios valores si se van a dar varios cortes pasaaltos para simular un filtro en cascada.

### ***CFLowF***

Frecuencia de corte correspondiente. Un valor 0 indica que no hay corte en este lado (p.ej. woofers).

Puede darse como vector de varios valores si se van a dar varios cortes pasaaltos para simular un filtro en cascada. Cuando el archivo de guión se use para filtros de fase lineal se usará solo el primer valor del vector como frecuencia de corte.

### ***CFLowAsMP***

En caso de que se implemente un pasaaltos butterworth en el woofer, indica si debe seguirse la misma curva de corte en los filtros de fase lineal, de modo que la curva de respuesta sea idéntica para *mp* y *lp*. Debe darse un valor booleano.

### ***CFHighType***

Para filtros *mp*, tipo del filtro pasabajos (que filtrará el extremo derecho de la vía —high—). Puede ser *Butterworth* o *LinkwitzRiley*. No tiene efecto si *CFClass* es *lp*.

### ***CFHighOrder***

Es el orden del filtro pasabajos *mp*. No tiene efecto si *CFClass* es *lp*.

### ***CFHighF***

Un valor 0 indica que no hay corte en este lado (p.ej. tweeters).

## **TW Transition Window**

La estrategia general de ecualización y filtrado conjunto que lleva a cabo el programa puede implicar ecualizaciones con ganancias muy elevadas si se intenta ecualizar fuera de la banda pasante. Para poner límites a esta ecualización está el siguiente grupo de parámetros.

### ***TWFlatInterval***

Es el rango, expresado en octavas, que se va a ecualizar plano antes de filtrarse, más allá de la frecuencia de corte. P.ej., un valor 2 en este parámetro para un pasabajos a 3000 hz implica que previamente al corte se ecualiza la curva hasta 12000 hz, dos octavas más allá del corte.

### ***TWTransitionInterval***

Pasado el umbral que define *TWFlatInterval*, la ecualización se va eliminando progresivamente durante este otro intervalo en octavas.

### ***TWLimitLowF1***

El mismo límite definido con *TWTransitionInterval* se define aquí como límite absoluto de frecuencia, que tiene precedencia sobre el resultado de aplicar *TWTransitionInterval*. Si no hay corte en el extremo izquierdo se aplica directamente este límite.

### ***TWLimitLowF2***

El mismo límite definido con *TWFlatInterval* se define aquí como límite absoluto de frecuencia, que tiene precedencia sobre el resultado de aplicar *TWFlatInterval*. Si no hay corte en el extremo izquierdo se aplica directamente este límite.

### ***TWLimitHighF1***

El mismo límite definido con ***TWFlatInterval*** se define aquí como límite absoluto de frecuencia, que tiene precedencia sobre el resultado de aplicar ***TWFlatInterval***. Si no hay corte en el extremo derecho se aplica directamente este límite.

### ***TWLimitHighF2***

El mismo límite definido con ***TWTransitionInterval*** se define aquí como límite absoluto de frecuencia, que tiene precedencia sobre el resultado de aplicar ***TWTransitionInterval***. Si no hay corte en el extremo derecho se aplica directamente este límite. Puede ser conveniente que sea inferior a la mayor frecuencia del archivo de datos **.frd**, en el caso de que se generen filtros para una frecuencia de muestreo superior a la usada al hacer la medida.

## **PS Plot Settings**

Estos valores definen algunos aspectos de los gráficos de salida.

### ***PSFLow***

Extremo izquierdo de frecuencia.

### ***PSFHigh***

Extremo derecho de frecuencia.

### ***PSVTop***

Valor máximo de presión sonora en los gráficos (dB).

### ***PSVStep***

Escalones de presión sonora en los gráficos (dB).

### ***PSVRange***

Rango total de presiones sonoras de salida (dB).

### ***PSOutRaster***

Valor booleano (*true* or *false*) que indica si se generan gráficos en formato raster.

### ***PSOutRasterExt***

Extensión del archivo de gráficos raster.

### ***PSOutVector***

Valor booleano (*true* or *false*) que indica si se generan gráficos en formato vectorial.

### ***PSOutVectorExt***

Extensión del archivo de gráficos vectoriales.



## Ejecución del programa DSD

En una terminal se cambiará a la carpeta de trabajo ('cd [carpeta de trabajo]'), o bien se dará el parámetro del archivo de guión con su ruta completa, sin extensión.

Al ejecutar 'dsdxof <archivo\_de\_guión>' se generará el filtro **.pcm** y los gráficos explicativos de la vía correspondiente.

Si el archivo de guión no incluye **GSFs** (frecuencia de muestreo) y **CFClass** (tipo de filtro, *mp* o *lp*) deben darse como parámetros adicionales. La frecuencia de muestreo se da como su valor numérico:

```
'dsdxof <archivo_de_guión> [fs <valor_numérico>] [class <mp|lp>]'
```

Por ejemplo, una posible llamada al programa podría ser:

```
'dsdxof t fs 44100 class lp'
```

Que calcularía un filtro según lo indicado en el archivo **t.xof** del directorio actual, con filtros para una **GSFs** de *44100*, de fase lineal.

Aunque **GSFs** y **CFClass** se especifiquen en el archivo de guión pueden darse como parámetros, que tendrán precedencia sobre el guión. Los parámetros no son posicionales, de modo que puede darse uno solo de ellos, o en distinto orden, pero deben ir siempre a continuación del nombre del archivo de guión, que sí es posicional.

Es correcto 'dsdxof t fs 44100' y 'dsdxof t' pero no 'dsdxof class lp t'.

El nombre de los filtros generados será:

**xo-<clase\_de\_filtro>-<nombre\_del\_frd\_de\_la\_medición>.pcm**

dentro de un subdirectorio (respecto a aquel en que esté el archivo de guión) cuyo nombre es la frecuencia de muestreo.

## Cálculos múltiples desde un script de octave

Al poder especificar como parámetros la frecuencia de muestreo y el tipo de filtro (*lp* o *mp*) es posible hacer un script para calcular de una sola vez filtros *lp* y *mp* a varias **GSFs**, para todas las vías del altavoz, con un solo archivo por vía en que se especifiquen los parámetros básicos (cortes, etc.) que se repiten.

Para ello los valores que varían en la secuencia de cálculo deseada se dan como vectores, en el caso de valores numéricos, o como cell arrays, en el caso de valores string. Por ejemplo:

```
% datos de usuario
inputfiles = {'t','m'};
fs = [44100,48000];
filtertypes = {'lp','mp'};
% fin datos

for i2 = 1:length(fs)
    unlink([num2str(fs(i2)) "/niveles.txt"]);
    for i1 = 1:length(inputfiles)
        for i3 = 1:length(filtertypes)
            status = system(["dsdxof " inputfiles{i1}
                             " " num2str(fs(i2))
                             " " filtertypes{i3}]);
        end
    end
end
```

end

En este caso se calcularían los filtros correspondientes a tweeter y medios (archivos **t.xof** y **m.xof**) a las frecuencias de muestreo **GSFs** de *44100* y *48000*, de los tipos *lp* y *mp*, para un total de ocho filtros **.pcm** de salida.

# Funciones del paquete DSD

## **audioaxe**

Configura una gráfica de respuesta en frecuencia con un formato convencional y obtiene sus ejes.

```
audioaxe(magtop, magstep, flow, fhigh, plotitle)
```

```
F          = Vector de frecuencias.
dBmag      = Vector de magnitudes en dB.
magtop     = Máximo de la magnitud (dB).
magrange   = Rango visible de la magnitud (dB).
magstep    = Escalones de dB a efectos de rejilla y rotulación.
flow       = Límite inferior de frecuencias (Hz).
fhigh      = Límite superior de frecuencias (Hz).
plotitle   = Título de la gráfica.
```

## **audioplot**

Dibuja una gráfica de respuesta en frecuencia con un formato convencional.

```
audioplot(F, dBmag, magtop, magstep, flow, fhigh, plotitle)
```

```
F          = Vector de frecuencias.
dBmag      = Vector de magnitudes en dB.
magtop     = Máximo de la magnitud (dB).
magrange   = Rango visible de la magnitud (dB).
magstep    = Escalones de dB a efectos de rejilla y rotulación.
flow       = Límite inferior de frecuencias (Hz).
fhigh      = Límite superior de frecuencias (Hz).
plotitle   = Título de la gráfica.
```

## **biqshelving**

Obtiene los coeficientes del filtro IIR asociado a un filtro shelving tal como se define en [www.linkwitzlab.com](http://www.linkwitzlab.com). La pendiente se limita a la ausencia de overshoot, con un máximo de 6 dB/oct. Las ganancias en la banda pasante son siempre positivas.

```
[b,a] = biqshelving(fs, f1, f2, type)
```

```
[b,a]      = Coeficientes del filtro IIR.
fs         = Frecuencia de muestreo.
f1         = Frecuencia de inicio de la pendiente.
f2         = Frecuencia final de la pendiente.
type       = Valor de cadena entre: lowShelf o highShelf.
```

## **biquad**

Obtiene los coeficientes del filtro IIR asociado a un biquad. Las ganancias en la banda pasante son siempre positivas.

```
[b,a] = biquad(Fs,f0,Q,type,dBgain)
```

```
[b,a]      = Coeficientes del filtro IIR.
fs         = Frecuencia de muestreo.
f0         = Frecuencia central del filtro.
Q          = Definido en "DSP EQ cookbook". En "peakingEQ" el ancho de
             banda es entre puntos de ganancia mitad.
type       = Valor de cadena entre:
             LPF, HPF, notch, peakingEQ, lowShelf o highShelf.
```

dBgain = Solo para peakingEQ, lowShelf o highShelf.

### **buttwindow**

Genera una ventana estándar de promediado de potencia con filtrado butterworth de 6º orden.

```
x = buttwindow (m,ppo,ppoSm)
```

x = Ventana.  
m = Longitud del espectro logarítmico a promediar.  
ppo = Fracción de octava del intervalo de frecuencias.  
ppoSm = Fracción de octava del suavizado.

### **centerimp**

Aumenta la longitud de un impulso centrándolo. El impulso original debe tener longitud impar.

```
imp = centerimp(imporig,m)
```

imp = Coeficientes del filtro FIR.  
imporig = Impulso a centrar. Debe ser de longitud impar.  
m = Longitud final del impulso.

### **crossButterworth**

Obtiene el filtro FIR de un filtro Butterworth de orden n.

```
imp = crossButterworth(Fs,m,nl,fl,nh,fh)
```

imp = Coeficientes del filtro FIR.  
Fs = Frecuencia de muestreo.  
m = Número de muestras.  
nl = Orden del filtro pasaaltos.  
fl = Frecuencia de corte inferior (pasaaltos). 0 para pasabajos.  
nh = Orden del filtro pasabajos.  
fh = Frecuencia de corte superior (pasabajos). 0 para pasaaltos.

### **crossLinear**

Obtiene el filtro FIR windowed sinc de fase lineal y alta pendiente, con ventana Blackman-Harris. Genera un filtro de longitud efectiva igual a un cierto número de ciclos de señal (a la frecuencia de corte), centrado en un vector de ceros de longitud m. m siempre es par, y el filtro como tal siempre es impar, de modo que el centrado tiene un desplazamiento de una muestra hacia el pasado.

```
imp = crossLinear(Fs,m,nc,fl,fh)
```

imp = Coeficientes del filtro FIR.  
Fs = Frecuencia de muestreo.  
m = Número de muestras.  
nc = Número de ciclos del impulso.  
fl = Frecuencia de corte inferior (pasaaltos). 0 para pasabajos.  
fh = Frecuencia de corte superior (pasabajos). 0 para pasaaltos.

### **crossLinkwitzRiley**

Obtiene el filtro FIR de un filtro Linkwitz-Riley de orden nl, nl par.

```
imp = crossLinkwitzRiley(Fs,m,nl,fl,nh,fh)
```

imp = Coeficientes del filtro FIR.  
Fs = Frecuencia de muestreo.  
m = Número de muestras.

nl	= Orden del filtro pasaaltos.
fl	= Frecuencia de corte inferior (pasaaltos). 0 para pasabajos.
nh	= Orden del filtro pasabajos.
fh	= Frecuencia de corte superior (pasabajos). 0 para pasaaltos.

### **crossLRmag**

Obtiene la magnitud de los filtros Linkwitz-Riley pasabajos y pasaaltos con pendiente dada sobre un semiespectro. El espaciado de frecuencias es arbitrario.

```
[magL, magH] = crossLRmag(F, fc, slope)
```

magL	= Vector columna con la magnitud del pasabajos.
magH	= Vector columna con la magnitud del pasaaltos.
F	= Vector columna con las frecuencias del semiespectro.
fc	= Frecuencia de corte.
slope	= Pendiente en dB/oct.

### **dB2mag, dB2pow**

Pasa un vector de decibelios a magnitud o potencia.

```
b = dB2mag(a)
```

b	= Magnitud.
a	= Decibelios.

```
b = dB2pow(a)
```

b	= Potencia.
a	= Decibelios.

### **delta**

Obtiene un impulso de longitud *m* con valor *1* en su primera muestra.

```
imp = delta(m)
```

imp	= Coeficientes del filtro FIR.
m	= Número de muestras.

### **deltacentered**

Obtiene un impulso de longitud *m* con valor *1* en su muestra central.

```
imp = deltacentered(m)
```

imp	= Coeficientes del filtro FIR.
m	= Número de muestras. Debe ser impar.

### **frjoin**

Une dos respuestas en magnitud sobre el semiespectro, mezclándolas en un intervalo de índices dado.

```
ssp = frjoin(ssp1, ssp2, k1, k2)
```

ssp	= Vector columna con la magnitud de la mezcla.
ssp1	= Vector columna con la respuesta a mezclar por la izquierda.
ssp2	= Vector columna con la respuesta a mezclar por la derecha.
k1	= Primer índice del intervalo.
k2	= Segundo índice del intervalo.

### **frjoinlog**

Une dos respuestas en magnitud sobre el semiespectro, mezclándolas en un intervalo de índices dado. Asume que las respuestas a unir están en una escala de frecuencias logarítmica.

```
ssp = frjoinlog(ssp1,ssp2,k1,k2)
```

```
ssp      = Vector columna con la magnitud de la mezcla.
ssp1     = Vector columna con la respuesta a mezclar por la izquierda.
ssp2     = Vector columna con la respuesta a mezclar por la derecha.
k1       = Primer índice del intervalo.
k2       = Segundo índice del intervalo.
```

### **gainpcm**

Aplica una cierta ganancia en dB a un archivo **.pcm** y lo guarda.

```
gainpcm (filename, gaindB)
```

```
filename = Nombres del archivo pcm.
gaindB   = Ganancia a aplicar (dB).
```

### **HouseCurve**

Obtiene los valores de la ecualización House Curve sobre un vector de frecuencias **F**.

```
[mag, pha] = HouseCurve (F, f_corner, house_atten, fs)
```

```
mag      = Vector de magnitudes (dB).
pha      = Vector de fases (deg).
F        = Vector de frecuencias.
f_corner = Frecuencia en la que empieza a bajar la curva.
house_atten = Atenuación a 20kHz.
fs       = Frecuencia de muestreo.
```

### **lininterp**

Obtiene la magnitud en decibelios sobre el semiespectro a partir de un archivo **.frd**.

```
maglin = lininterp (F,mag,m,fs)
```

```
maglin   = Magnitud interpolada.
mag      = Magnitud a interpolar.
F        = Vector de frecuencias.
m        = Longitud del espectro completo (debe ser par).
fs       = Frecuencia de muestreo.
```

### **loadfrd**

Carga datos de respuesta en frecuencia desde un archivo **.frd**.

Se ignoran las líneas con texto no numérico.

```
[freq, magdB, pha] = loadfrd (filename)
```

```
freq     = Vector de frecuencias.
magdB    = Magnitud en dB.
pha      = Fase.
filename = nombre completo del archivo .frd.
```

### **loadpcm**

(Tomada de DRC-fir) Lee archivos **.pcm**.

```
pcm = loadpcm(fname)

pcm      = Vector columna del impulso.
fname    = Nombre del archivo .pcm.
```

### ***loadpir***

Lee archivos **.pir** de ARTA.

```
pcm = loadpir(fname)

pcm      = Vector columna del impulso.
fname    = Nombre del archivo .pir.
```

### ***logfreq***

Genera un vector de frecuencias espaciado logaritmicamente, entre el bin menor no nulo del fft y ***fs/2***.

```
logf = logfreq(m, fs, ppo)

m      = Longitud del fft original.
fs     = Frecuencia de muestreo.
ppo    = Fracción de octava del intervalo de frecuencias.
```

### ***mag2dB, pow2dB***

Pasa un vector de magnitud o potencia a decibelios .

```
b = mag2dB(a)

b      = Decibelios.
a      = Magnitud.

b = pow2dB(a)

b      = Decibelios.
a      = Potencia.
```

### ***minexcphsp***

Obtiene el espectro de fase mínima y el pasatodo con el exceso de fase a partir de un espectro completo.

```
[minph, excph] = minexcphsp(sp)

minph    = Espectro completo de fase mínima con la misma magnitud de
           espectro que imp.
excph    = Espectro completo pasatodo de exceso de fase.
sp       = Espectro completo. Longitud par.
```

### ***minphsp***

Obtiene el espectro de fase mínima a partir de un espectro completo.

```
minph = minphsp(sp)

minph    = Espectro completo de fase mínima con la misma magnitud que
           sp.
sp       = Espectro completo. Longitud par.
```

## **RoomGain**

Obtiene los valores de la ecualización Room Gain sobre un vector de frecuencias  $F$ .

```
[mag, pha] = RoomGain (F, gain_dBS, fs)
```

mag	= Vector de magnitudes (dB).
pha	= Vector de fases (deg).
F	= Vector de frecuencias.
gain_dBS	= Ganancia total a DC sobre la respuesta plana.
fs	= Frecuencia de muestreo.

## **savepcm**

(Tomada de DRC-fir) Escribe archivos **.pcm**.

```
savepcm(pcm, fname)
```

pcm	= Vector del impulso.
fname	= Nombre de archivo .pcm.

## **semiblackman**

Obtiene la mitad derecha de una ventana Blackman de longitud  $m$ .

```
w = semiblackman(m)
```

w	= Ventana.
m	= Número de muestras.

## **semisp**

Obtiene el espectro de las frecuencias positivas a partir de un espectro completo.

```
ssp = semisp(wsp)
```

ssp	= Semiespectro entre 0 y $m/2$ .
wsp	= Espectro completo entre 0 y $m-1$ ( $m$ par).

## **smooth**

Suaviza un semiespectro real con un ancho dado en fracción de octava.

```
xsss = smooth(xws,ppo)
```

xsss	= Vector columna con el semiespectro suavizado.
xws	= Vector columna de valores reales (magnitud o fase) del semiespectro.
ppo	= Fracción de octava del suavizado.

## **smoothpw**

Suaviza en potencia un semiespectro real con un ancho dado en fracción de octava.

```
xsss = smoothpw(xws,ppo)
```

xsss	= Vector columna con el semiespectro suavizado.
xws	= Vector columna de valores reales (magnitud o fase) del semiespectro.
ppo	= Fracción de octava del suavizado.



### ***smoothlog***

Suaviza un espectro logarítmico con un ancho dado en fracción de octava.

```
xs = smoothlog(x, ppo, ppoSm)
```

xs	=	Vector columna con el espectro logarítmico suavizado.
x	=	Vector columna de valores reales con el espectro logarítmico.
ppo	=	Fracción de octava del intervalo de frecuencias.
ppoSm	=	Fracción de octava del suavizado.

### ***smoothlogpw***

Suaviza en potencia un espectro logarítmico con un ancho dado en fracción de octava.

```
xs = smoothlogpw(x, ppo, ppoSm)
```

xs	=	Vector columna con el espectro logarítmico suavizado.
x	=	Vector columna de valores reales con el espectro logarítmico.
ppo	=	Fracción de octava del intervalo de frecuencias.
ppoSm	=	Fracción de octava del suavizado.

### ***trwcos***

Genera ventanas de transición complementarias en un intervalo dado, según una función “raised cosine” en escala de frecuencias logarítmica.

```
[trw1, trw2] = trwcos(n1, n2)
```

trw1	=	Ventana de transición de izquierda a derecha.
trw2	=	Ventana de transición de derecha a izquierda.
n1	=	Índice del extremo izquierdo de la ventana.
n2	=	Índice del extremo derecho de la ventana.

### ***trwcoslog***

Genera ventanas de transición complementarias en un intervalo dado, según una función “raised cosine”. Los datos de entrada han de estar en magnitudes logarítmicas, tanto en frecuencia como en valor.

```
[trw1, trw2] = trwcoslog(n1, n2)
```

trw1	=	Ventana de transición de izquierda a derecha.
trw2	=	Ventana de transición de derecha a izquierda.
n1	=	Índice del extremo izquierdo de la ventana.
n2	=	Índice del extremo derecho de la ventana.

### ***wholesplp***

Obtiene el espectro simétrico completo a partir del espectro de las frecuencias positivas.

```
wsp = wholesplp(ssp)
```

wsp	=	Espectro completo entre 0 y $m-1$ ( $m$ par).
ssp	=	Semiespectro entre 0 y $m/2$ .

### ***wholespmp***

Obtiene el espectro causal completo a partir del espectro de las frecuencias positivas.

```
wsp = wholespmp(ssp)
```

wsp       = Espectro completo entre 0 y  $m-1$  ( $m$  par).  
ssp       = Semiespectro entre 0 y  $m/2$ .