DSD

Diseño de filtros digitales

Instalación de Octave

Bajar el instalador de windows de:

http://sourceforge.net/projects/octave/files/Octave_Windows%20-%20MinGW/Octave%203.4.3%20for%20Windows%20MinGW%20Installer/

Al ejecutarlo se sugire instalarlo en *C:\Octave\Octave3.4.3_gcc4.5.2*. Conviene cambiarlo a *C:\Octave*, por un defecto del instalador.

Se debe bajar también el archivo de paquetes, *Octave3.4.3_gcc4.5.2_pkgs_20111026.7z* tal como se detalla en las instrucciones de instalación de la URL de arriba.

Se instala con Octave una versión adaptada del entorno *cygwin*, por lo que este no debe estar instalado. Si no se sabe de qué hablo, puede asegurarse que *cygwin* no está instalado.

Estas indicaciones son válidas en el momento en que se han escrito, pero variarán con sucesivas actualizaciones de Octave. Deben consultarse las instrucciones de instalación de la versión de Octave que se instale.

Configuración

Editor de texto

El *readme* indica un procedimiento para establecer notepad++ como **editor por defecto**:

> *In order to install Notepad++ (recommended) as an editor:*
>
> *a. download the installation package from http://notepad-plus-plus.org/*
>
> *b. install, edit <your octave dir>\share\octave\site\m\startup\octaverc*
>
> *c. uncomment the line which set octave default editor: EDITOR('C:\\Program Files\\Notepad++\\notepad++.exe');*

En mi caso la ruta es: `C:\Program Files (x86)\Notepad++\notepad++.exe`

Unidades de red

Si se trabaja con archivos en algún disco compartido en red, es conveniente dar de alta una unidad de red de windows en lugar de utilizar la ruta de red (rutas UNC).

De lo contrario los archivos gráficos no se generarán.

Achivo de inicio

Por defecto el directorio *home*, donde el usuario tiene sus archivos personales, y donde comienza el programa, está en `C:\Users\[usuario]`. Para cambiar a ese directorio desde el *prompt*: `cd ~`

En este directorio `HOME`, o ~, se encuentra el archivo `.octaverc`, que se ejecuta al inicio como un script de *octave*, es decir, ejecuta órdenes del programa, no de *cygwin* o del sistema operativo. Entre estas órdenes es importante la asignación de la ruta de búsqueda de funciones. Por ejemplo:

```
addpath('C:\Users\Roberto\octave\dsd');
```

Aunque no la usaremos así directamente.

Eliminar advertencias

La actualización 3.4.3 introdujo un aviso bastante molesto, que se produce al evaluar condiciones *if* en las que hay operadores lógicos, relativo a *short circuit operators*.

Lo eliminamos añadiendo a *.octaverc*:

```
warning off Octave:possible-matlab-short-circuit-operator;
```

Volver a gnuplot

La máquina de gráficos por defecto es ahora *kkkkk*, que da problemas al generar los archivos gráficos. Es preferible por ahora seguir usando el viejo *gnuplot*. Para ello, añadimos a *.octaverc*:

```
graphics_toolkit ('gnuplot');
```

Pager

Para que la salida en pantalla se pagine con *less* , añadimos a *.octaverc*:

```
more on;
```

Instalación de paquetes

Paquete DSD

DSD quiere decir (por ejemplo) Digital Signal Design. Contiene las funciones específicas para el diseño de filtros digitales, y los scripts de diseño que las utilizan.

Debe copiarse a un directorio, por ejemplo:

C:\Users\[usuario]\Octave

Para guardar permanentemente la ruta de búsqueda, ejecutar la secuencia:

```
addpath(genpath('C:\Users\Roberto\octave'))
savepath()
```

En esta órden vemos algunos signos que debemos conocer. En primer lugar, se trata de una función cuyo argumento, el nombre del directorio a añadir, va entre paréntesis. Dicho argumento es un dato tipo *string*, una cadena de texto, y debe ir entre comillas, dobles o simples. Si se usan comillas simples no se expandirán las secuencias de escape.

Otros paquetes

Aún no se utilizan, y por tanto son opcionales, pero por su interés se reseñan a continuación.

HUTear

Bajar http://www.acoustics.hut.fi/software/HUTear/HUTear.zip y http://www.acoustics.hut.fi/software/HUTear/winmex.zip.

Descomprimir el primero en (p.ej.) *C:\Users\[usuario]\Octave*. Se creará un directorio HUTear2, y dentro de este un HUTear. En este último descomprimir los archivos mex precompilados de winmex.zip.

Ejecutar las órdenes de asignación de rutas de búsqueda indicadas más arriba.

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.
- 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 dsd, con el archivo de guión 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 dsd 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

En el momento de redactar este manual hay dos tipos principales de filtro: Filtros de fase lineal (LP) y filtros de fase mínima (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 pasaaltos, a fin de añadir a una vía los retardos de las vías inferiores, cortando adicionalmente con el mismo pasaaltos que estas, reproduciendo el efecto de un filtro activo en cascada.

Confección de los archivos de guión

La carpeta 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 *Rrxof.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.

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".

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.

CFLowType

Para filtros MP, tipo del filtro pasabajos (que filtrará el extremo derecho de la vía). Puede ser "Butterworth" o "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.

CFLowOrder

Es el orden del filtro pasabajos 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. Si el archivo de guión se usa también 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 pasaaltos (que filtrará el extremo izquierdo de la vía). Puede ser "Butterworth" o "LinkwitzRiley". No tiene efecto si CFClass es LP.

CFHighOrder

Es el orden del filtro pasaaltos 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 crossover 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 crossover 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 crossover 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 crossover 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.

PSVStep

Escalones de presión sonora en los gráficos (dB).

PSVRange

Rango total de presiones sonoras de salida (dB).

Ejecución del programa DSD

Tras lanzar octave, se cambia desde su prompt a la carpeta de trabajo (*cd [carpeta de trabajo]*), o bien se da el parámetro del archivo de guión con su ruta completa, *sin extensión*.

Al ejecutar *RRxof [archivo_de_guión]* se generará el filtro pcm y los gráficos explicativos de la vía correspondiente.

Si el archivo de guión no incluye GSFs y CFClass deben darse como parámetros. Por ejemplo, una posible llamada al programa podría ser:

Rrxof t 44100 lp

Que calcularía un filtro según lo indicado en el archivo *t.xof* del directorio actual, con filtros para fs=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 son posicionales, de modo que no pueden omitirse parámetros intermedios, pero no tienen que especificarse todos.

Es correcto: “Rrxof t 44100” y “Rrxof t” pero no “Rrxof t lp”.

El nombre de los filtros generados será <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.

Funciones del paquete DSD

audioplot

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

```
audioplot(F, dBmag, bottom, top, step, plotitle)
          F         = Vector de frecuencias.
          dBmag     = Vector de magnitudes en dB.
          bottom    = Mínimo de la magnitud (dB).
          top       = Máximo de la magnitud (dB).
          step      = Escalones de dB a efectos de rejilla y rotulación.
          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. 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 uno 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 uno en su muestra central.

```
imp = deltacentered(m)
          imp       = Coeficientes del filtro FIR.
          m         = número de muestras. Debe ser impar.
```

frdinterp

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

```
magdB = frdinterp(filename,m,fs)
          magdB     = Magnitud en dB sobre el semiespectro.
          filename  = Nombre del archivo .frd.
          m         = Longitud del espectro completo (debe ser par).
          fs        = Frecuencia de muestreo.
```

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         = Frecuencias.
          m         = Longitud del espectro completo (debe ser par).
          fs        = Frecuencia de muestreo.
```

loadpcm

(Tomada de DRC-fir) Lee archivos pcm.

```
pcm = loadpcm(fname)
          pcm       = vector columna del impulso.
          fname     = Nombre del archivo .pcm.
```

loadpcms

Carga archivos impulso que verifican una máscara en una matriz de vectores columna.

```
[imps,n] = loadpcms(filemask,kinit,kend)
          imps      = matriz de vectores columna de los impulsos cargados.
          n         = número de impulsos.
          filemask  = máscara de nombres de archivo (string).
          kinit     = índice para el comienzo del recorte.
          kend      = índice para el comienzo del recorte.
```

loadpir

Lee archivos .pir de ARTA.

```
pcm = loadpir(fname)
          pcm       = vector columna del impulso.
          fname     = Nombre del archivo .pcm.
```

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 pasa-todo 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. *[revisar]*

```
[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.
```

Otras funciones útiles

wavread

(Original de octave) Lee archivos wav.

```
[Y, FS, BITS] = wavread (FILENAME)
          Y         = muestras de audio (cada canal en una columna).
          FS        = frecuencia de muestreo (Hz).
          BITS      = bits por muestra.
```

wavwrite

(Original de octave) Escribe archivos wav.

```
wavwrite (Y, FS, BITS, FILENAME)
          Write Y to the canonical RIFF/WAVE sound file FILENAME with sample
          rate FS and bits per sample BITS.  The default sample rate is 8000 Hz
          with 16-bits per sample.  Each column of the data represents a
          separate channel.
```

fftfilt

(Original de octave).

```
fftfilt (b, x, n)
          filters x with the FIR filter b using the FFT.
```

filter

(Original de octave).

```
y = filter (b, a, x)
          In terms of the z-transform, y is the result of passing the discrete-
          time signal x through a system characterized by the following rational
          system function:
                       M
                      SUM d(k+1) z^(-k)
                      k=0
            H(z) = ----------------------
                         N
                    1 + SUM c(k+1) z^(-k)
                        k=1

          where c = a/a(1) and d = b/a(1).
```

freqz

(Original de octave).

```
[h, w] = freqz (b, a, w, Fs)
```

```
Return the complex frequency response h of the rational IIR filter
whose numerator and denominator coefficients are b and a,
respectively. Evaluate the response at the specific frequencies in the
vector w. Return frequencies in Hz instead of radians assuming a
sampling rate Fs.
```

impz

(Original de octave).

```
usage: [x, t] = impz(b [, a, n, fs])

          Generate impulse-respo,nse characteristics of the filter. The filter
          coefficients correspond to the the z-plane rational function with
          numerator b and denominator a.  If a is not specified, it defaults to
          1. If n is not specified, or specified as [], it will be chosen such
          that the signal has a chance to die down to -120dB, or to not explode
          beyond 120dB, or to show five periods if there is no significant
          damping. If no return arguments are requested, plot the results.
```

```
usage: zplane(b [, a]) or zplane(z [, p])

 Plot the poles and zeros.  If the arguments are row vectors then they
 represent filter coefficients (numerator polynomial b and denominator
 polynomial a), but if they are column vectors or matrices then they
 represent poles and zeros.

 This is a horrid interface, but I didn't choose it; better would be
 to accept b,a or z,p,g like other functions.  The saving grace is
 that poly(x) always returns a row vector and roots(x) always returns
 a column vector, so it is usually right.  You must only be careful
 when you are creating filters by hand.

 Note that due to the nature of the roots() function, poles and zeros
 may be displayed as occurring around a circle rather than at a single
 point.

 The transfer function is

        B(z)   b0 + b1 z^(-1) + b2 z^(-2) + ... + bM z^(-M)
 H(z) = ---- = ------------------------------------------
        A(z)   a0 + a1 z^(-1) + a2 z^(-2) + ... + aN z^(-N)

                b0           (z - z1) (z - z2) ... (z - zM)
              = -- z^(-M+N) ------------------------------
                a0           (z - p1) (z - p2) ... (z - pN)

 The denominator a defaults to 1, and the poles p defaults to [].
```

Function File: [*sos*, g] = **zp2sos** (*z*, *p*)

Convert filter poles and zeros to second-order sections.

INPUTS:

- *z* = column-vector containing the filter zeros
- *p* = column-vector containing the filter poles
- *g* = overall filter gain factor

RETURNED:

- *sos* = matrix of series second-order sections, one per row:
  sos = [*B1*.' *A1*.'; ...; *BN*.' *AN*.'], where
  B1.'==[b0 b1 b2] and *A1*.'==[1 a1 a2] for section 1, etc.
  b0 must be nonzero for each section.
  See `filter()` for documentation of the second-order direct-form filter coefficients *B*i and %*A*i, i=1:N.
- *Bscale* is an overall gain factor that effectively scales any one of the *B*i vectors.

EXAMPLE:

```
  [z,p,g] = tf2zp([1 0 0 0 0 1],[1 0 0 0 0 .9]);
  [sos,g] = zp2sos(z,p,g)

sos =
   1.0000    0.6180    1.0000    1.0000    0.6051    0.9587
   1.0000   -1.6180    1.0000    1.0000   -1.5843    0.9587
   1.0000    1.0000         0    1.0000    0.9791         0

g =
    1
```

rline = **dbstop** (*func, line, ...*)

Set a breakpoint in a function

`func`

String representing the function name. When already in debug mode this should be left out and only the line should be given.

`line`

Line number you would like the breakpoint to be set on. Multiple lines might be given as separate arguments or as a vector.

The rline returned is the real line that the breakpoint was set at.

dbcont

In debugging mode, quit debugging mode and continue execution.

dbquit

In debugging mode, quit debugging mode and return to the top level.

Modificaciones a DSD (ejem, es por una buena causa):

La causa es poder hacer scripts para calcular de un tirón filtros LP y MP a varias Fs, con un solo archivo en que se especifiquen los parámetros básicos (cortes, etc.) que se repiten siempre.

Por eso (y otras cosas) se han hecho las siguienetes mods:

1. Los archivos de guión no llevan la extensión .eqf, sino .xof, como se dice en el manual, y acorde con el nombre del script. La extensión es obligatoria porque...

2. Se da como parámetro de RRxof el nombre de archivo sin extensión (más cortito).

3. Opcionalmente de pueden dar, por este orden, fs y clase de filtro (lp o mp), y tienen precedencia sobre el guión (se pueden omitir en el guión, incluso). Por ejemplo: "RRxof t 44100 lp", o "RRxof t 44100", o "RRxof t" son órdenes válidas.

4. Desaparecen los parámetros FSOutPrefix y FSOutDir. El nombre de archivo y directorio se toman del archivo .frd de medición y se compone con la clase de filtro. De t.frd (lp) saldría lp-t.pcm.

5. Siempre se genera un subdirectorio cuyo nombre es la frecuencia de muestreo.

Y ya.

Es posible hacer scripts chulos, tipo esto:

```
% 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)
         RRxof(inputfiles{i1},num2str(fs(i2)),filtertypes{i3});
      end
   end
end
```