ASN Filter Designer

FilterScript user's guide

July 2019
ASN17-DOC002, Rev. 4

For public release

Legal notices

Advanced Solutions Nederland B.V.

www.advsolned.com

support@advsolned.com

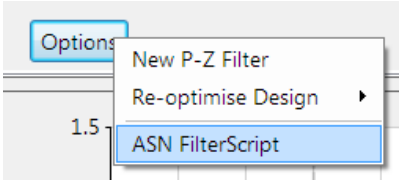
Technical documentation feedback

If you would like to make a suggestion or report an error in our documentation, please feel free to contact us. You are kindly requested to provide as much information as possible, including a full description of the error or suggestion, the page number and the document number/description. All suggestions or errors may be sent to: documentation@advsolned.com

Summary

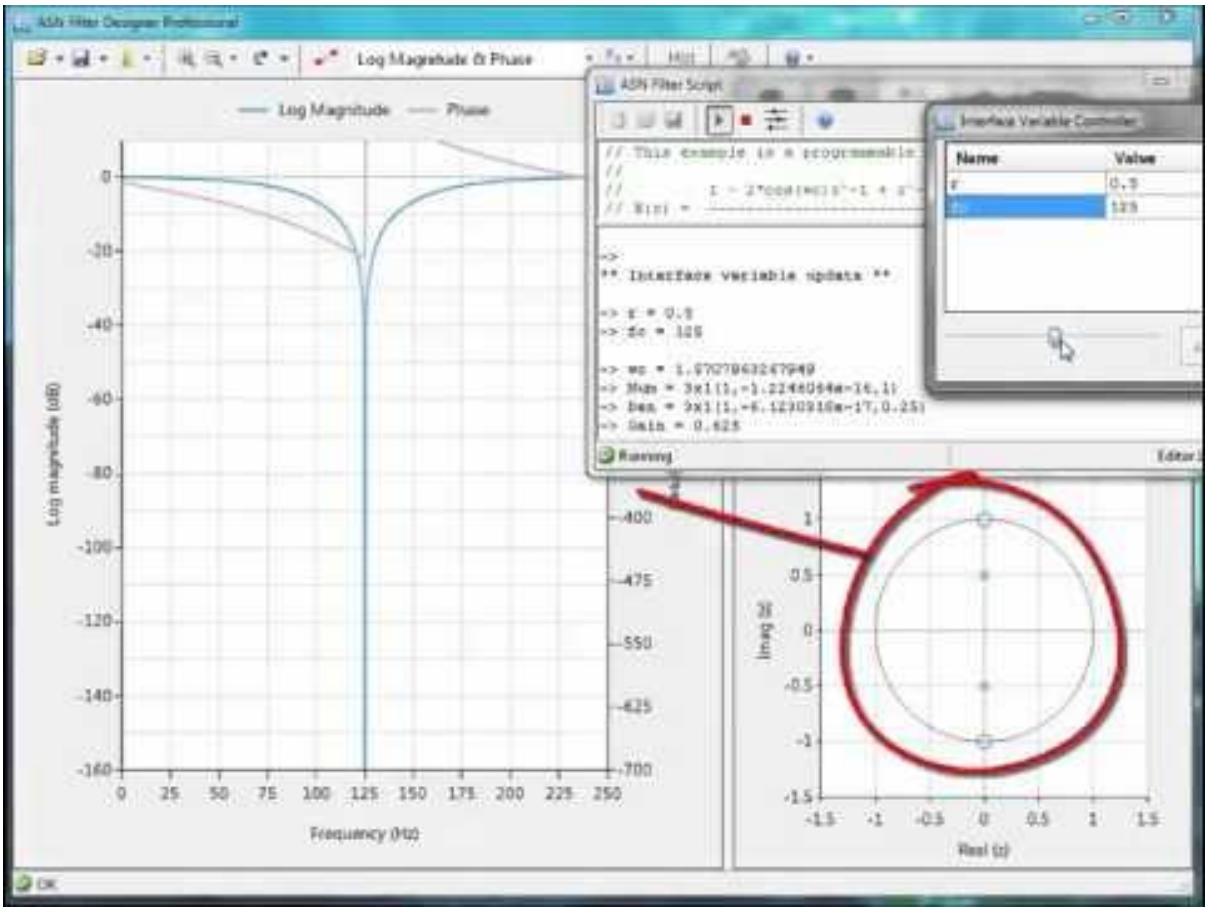
This user's guide is intended to provide users of the ASN Filter Designer with a concise overview of the symbolic math scripting language IDE.

The symbolic filter script session may be started from the main toolbar by  or selecting `ASN FilterScript` in the **options** menu in the P-Z editor.



The scripting language supports over 65 scientific commands and
provides designers with a familiar and powerful programming language, while at the same time allowing them to implement complex symbolic mathematical expressions for their filtering applications. The scripting language offers the unique and powerful ability to modify parameters on the fly with the so called interface variables, allowing for real-time updates of the resulting frequency response. This has the advantage of allowing the designer to see how the coefficients of the symbolic transfer function expression affect the frequency response and the filter's time domain dynamic performance.

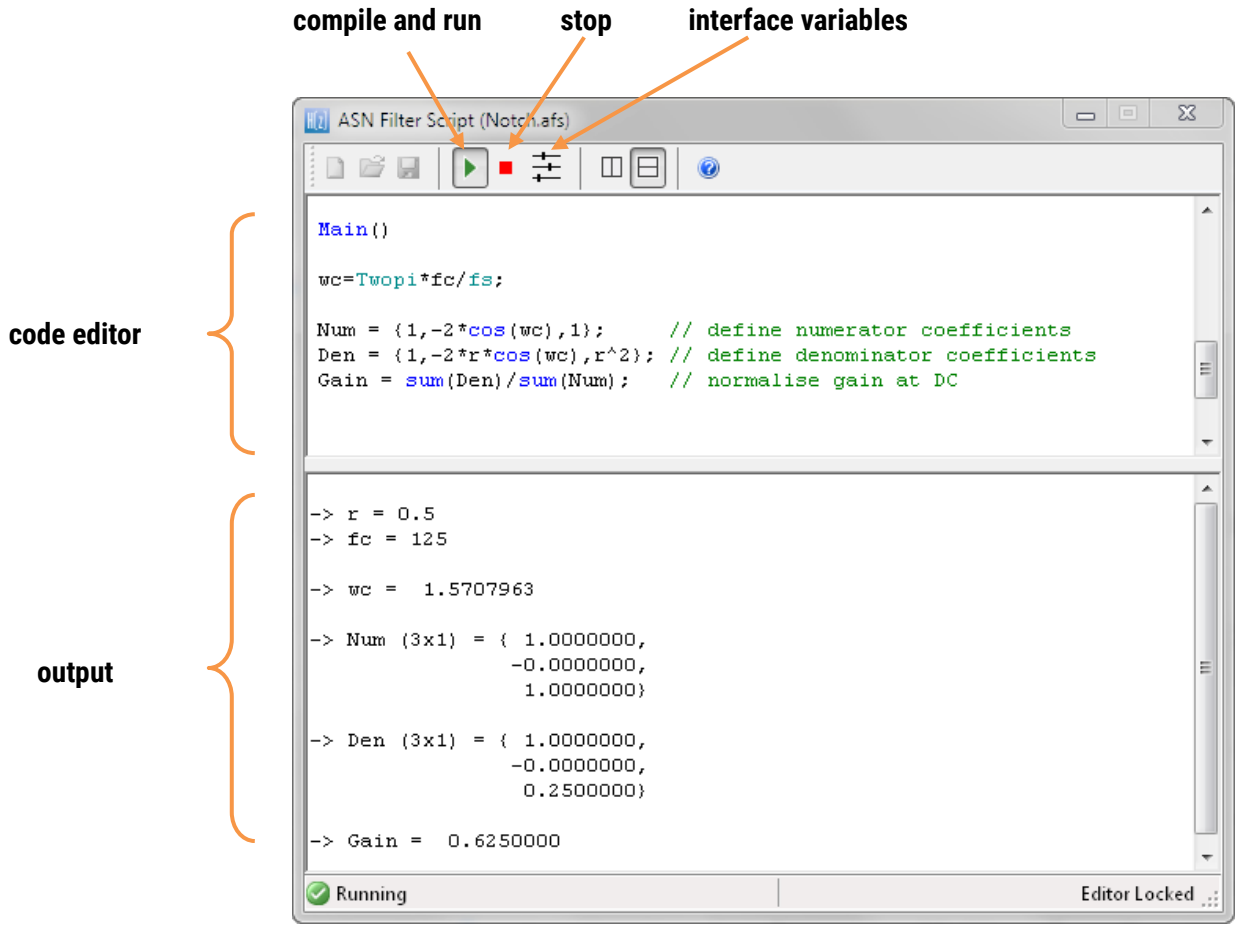
Video tutorial to get you started



ASN Filter script in a nutshell

1. Filter script IDE

The filter script IDE (integrated development environment) provides you with all of the necessary features in order to design and evaluate your symbolic filter concept. The IDE output is coupled to the filter designer GUI, providing a fully interactive method of customising your filter transfer function on the fly.



As seen, the IDE is split up into a code editor and an output window. The IDE differs from other scripting IDEs in that all executed code appears in the output window, and there is no provision for entering and evaluating expressions in the output window directly.

As with all standard code editors, right clicking in the editor produces a standard options menu for copying, pasting, cutting and adding/removing comments respectively.

An IntelliSense is provided in order to help with the syntax of commands.

1.1. Code structure

The primary purpose of the symbolic filter script is to obtain values for the following three inputs:

- `Num`: the numerator coefficients
- `Den`: the denominator coefficients
- `Gain`: the filter gain

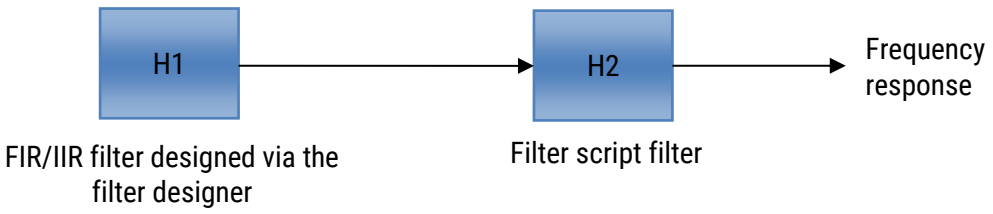
In order to provide a flexible means of modifying parameters on the fly (see section 1.2), the code is spilt up into two sections:

- *Initialisation*: contains all definitions of interface variables and any constant expressions. This section is run only once after compilation.
- *Main*: contains the bulk of the code, including the `Num`, `Den` and `Gain` expressions. Any expressions containing interface variables will be updated when modified via the interface variable GUI (see section 1.2.1).

The basic code structure is summarised below:

```
ClearH1;
interface variables
constant  expressions

Main()
{
        Num = {};
        Den = {};
        Gain = 1;
}
```

The `ClearH1` expression allows you to remove the H1 filter (primary) from the filter cascade and just use the H2 (secondary) filter. The relationship of the H1 and H2 filters is shown below:



As seen, the main FIR/IIR filter designed via the filter designer GUI is assigned to the *primary filter, H1*. All poles and zeros defined via the filter script are added to a *secondary filter block, H2*. The H2 filter block implements the filter as a single section (i.e. no biquads) IIR, which eases the implementation, but also the advantage of assigning poles to an FIR primary filter. In the case of no poles, the H2 filter becomes an FIR filter.

It should be noted that a direct form (single section) implementation may become problematic (due to numerical stability issues) for higher IIR filter orders, especially when poles are near to the unit circle.

The Heq (all-pass filter) and H3 (post filter) filters must be disabled via the main UI if they are not required.

1.2. Interface variables

Central to the interactivity of the tool are the so called *interface variables*. An interface variable is simply stated: *a scalar input variable that can be used modify a symbolic expression without having to re-compile the code*.

As discussed in section 1.1, interface variables must be defined in the initialisation section of the code, and may contain constants (such as, `fs` and `pi` - see section 2.10 for the complete list) and simple mathematical operators, such as multiply `*` and `/` divide. Where, adding functions to an interface variable is not supported.

An interface variable is defined as vector expression:

```
interface name = {minimum, maximum, step_size, default_value};
```

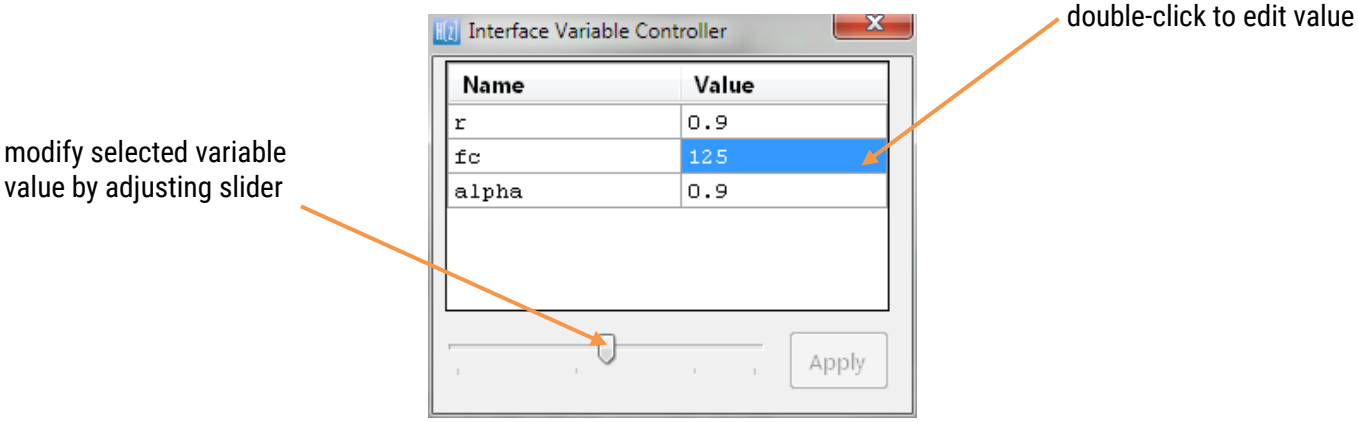
where, all entries must be real scalars values. Vectors and complex values will not compile.

Examples

`interface alpha = {-1,1,0.1,0.3};`	sets the variable `alpha` to `0.3`, and bounds the range to `±1`, in steps of `0.1`.
`interface fc = {-fs/2,fs/2,1,fs/10};`	sets the variable `fc` to `fs/10`, and bounds the range to `±fs/2`, in steps of `1`.

1.2.1. User interface

All interface variables are modified via the interface variable GUI, by clicking on 



As seen, a list of valid interface variables is presented together with their current values. Where, the list is automatically updated at compilation time in order to ensure that it matches the user code.

2. The Scripting language

The scripting language supports over 65 scientific commands and provides designers with a familiar and powerful programming language for designing filters with the most demanding specifications.

2.1. Trigonometrical functions

Function	Syntax	Description
angle	y = angle(x)	Compute the inverse tangent (four quadrant)
cos	y = cos(x)	Compute the cosine
cosh	y = cosh(x)	Compute the hyperbolic cosine
sin	y = sin(x)	Compute the sine
sinh	y = sinh(x)	Compute the hyperbolic sine
tan	y = tan(x)	Compute the tangent
tanh	y = tanh(x)	Compute the hyperbolic tangent

2.2. Vector functions

Function	Syntax	Description
cols	y = cols(x)	Gets number of columns in the vector x
conv	y = conv(a,b)	Computes the linear convolution of input vectors a and b. Where, the length of y is equal to length(a) + length(b)-1
diff	y = diff(x)	Gets the difference between adjacent values of the vector x
eldef	A(3,0) = eldef(5)	Sets the vector element value to the given value
length	y = length(x)	Gets the vector length
max	y = max(x)	Gets the maximum of the vector x
mean	y = mean(x)	Gets the mean of the vector x
min	y = min(x)	Gets the minimum of the vector x
ones	y = ones(N)	Vector of 1s of length N
poly	y = poly(x)	Convert roots to polynomial
reverse	y = reverse(x)	Flip vector elements up-to-down
roots	y = roots(x)	Get the roots of polynomial x
rows	y = rows(x)	Gets the number of rows in vector x
series	y = series(min,step,max)	Creates a data series - see section 2.5
sortup	y = sortup(x)	Sort vector in ascending order: smallest first, largest last
sortdown	y = sortdown(x)	Sort vector in descending order: largest first, smallest last
stddev	y = stddev(x)	Gets the standard deviation of x
sum	y = sum(x)	Gets the sum of vector x
transpose	y = transpose(x)	Transpose vector x
zeros	y = zeros(N)	Vector of 0s of length N

2.3. General functions

Function	Syntax	Description
abs	`y = abs(x)`	Compute the absolute value(s).
ceil	`y = ceil(x)`	Round up to infinity.
conj	`y = conj(x)`	Compute the complex conjugate.
exp	`y = exp(x)`	Compute the exponential of the argument, i.e. e^x.
pow2	`y = pow2(x)`	Compute the element to the power of 2, i.e 2^x
pow10	`y = pow10(x)`	Compute the element to the power of 10, i.e 10^x
flip	`y = flip(x)`	Flip the real and imaginary components of `x`
floor	`y = floor(x)`	Round down to -∞
imag	`y = imag(x)`	Get the imaginary component of `x`
ln	`y = ln(x)`	Natural log
log10	`y = log10(x)`	Log base 10
log2	`y = log2(x)`	Log base 2
logn	`y = logn(N,x)`	Log of `x` to base `N`
newpz	`y = newpz(mag,freq)`	Define a root: $0 \leq mag \leq 5$ and $0 \leq freq \leq \pm fs/2$ $roots = \left(z - r_1 e^{-i\theta_1}\right), \left(z - r_2 e^{-i\theta_2}\right), \ldots$ Where, $\theta_x = \frac{2\pi f}{f_s}$
real	`y = real(x)`	Get the real component of `x`
round	`y = round(x)`	Round `x`
sqr	`y = sqr(x)`	Square of `x`
sqrt	`y = sqrt(x)`	Square root of `x`

2.4. Math operators

Operator	Example syntax	Description
+	`a+b`	Addition.
-	`a-b`	Subtraction.
*	`A*B`	Multiplication.
/	`A/B`	Division.
.*	`A.*B`	Element-by-element vector multiplication.
./	`A./B`	Element-by-element vector division.
.^	`a.^N`	Element-by-element vector to the power.
^	`a^N`	Vector to the power.
!	`N!`	Factorial.

2.5. IIR filter design

A summary of all classic IIR filter design methods that are supported is given in this section.

2.5.1. ellip

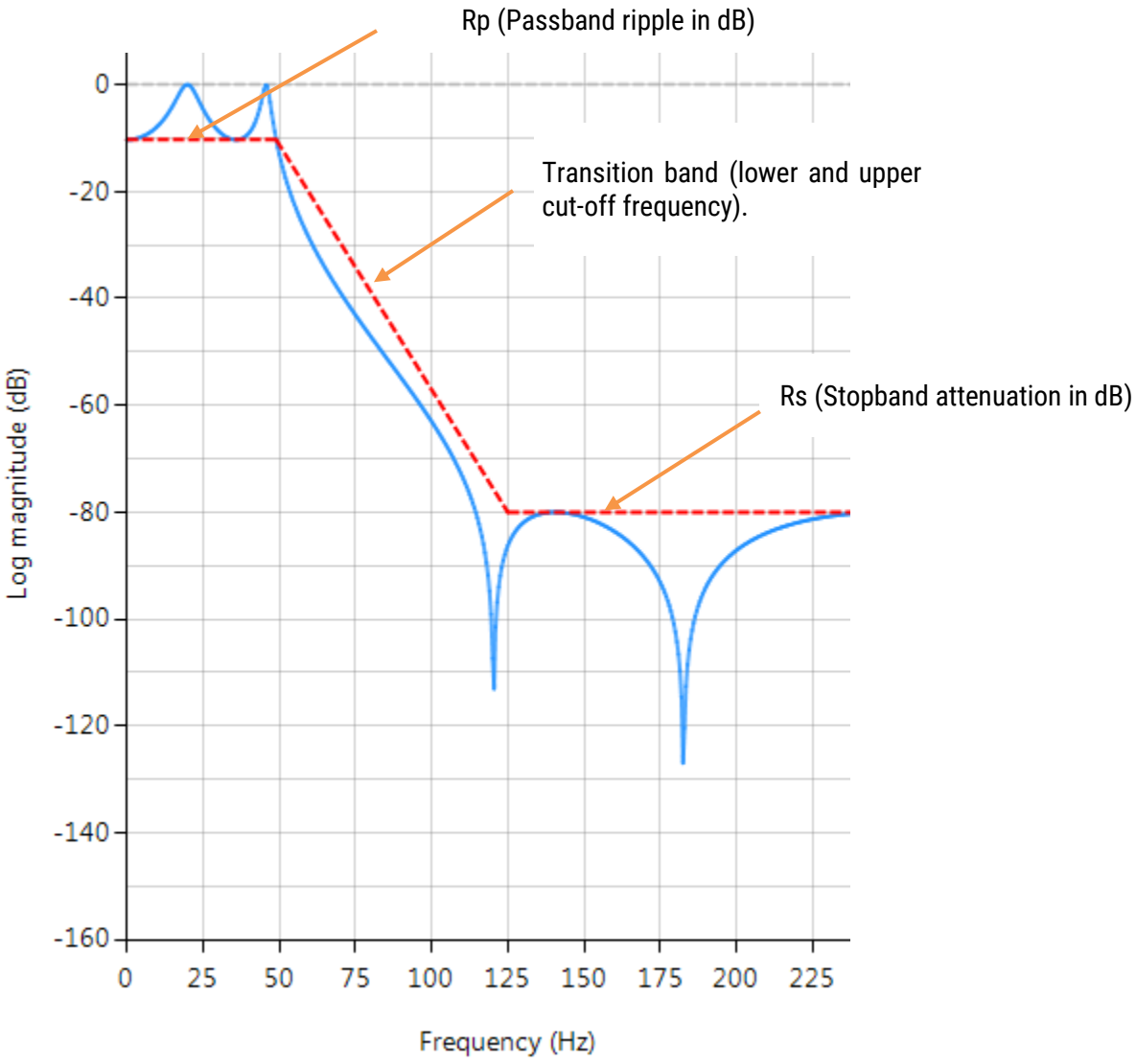
Syntax

```
Hd = ellip (Order, Frequencies, Rp, Rs, Type, DFormat)
```

Description

Classic IIR Elliptic filter design.

- Equiripple in both the passband and stopband.
- Fastest roll-off.
- Lowest order filter of all supported prototypes.



`Hd = ellip (Order, Frequencies, Rp, Rs, Type, DFormat)`

`Order`: may be specified up to `20` (professional) and up to `10` (educational) edition. Setting the `Order` to `0`, enables the automatic order determination algorithm.

`Frequencies`: lowpass and highpass filters have one transition band, and in as such require two frequencies (i.e. lower and upper cut-off frequencies of the transition band). For bandpass and bandstop filters, four frequencies are required (i.e. two transition bands). All frequencies must be ascending in order and < Nyquist (see the example below).

`Rp`: Passband ripple in dB.
`Rs`: Stopband attenuation in dB.

`Type`: The elliptic method facilitates the design of `lowpass`, `highpass`, `bandpass` and `bandstop` filters respectively.

`Hd`: the elliptic method designs an IIR Elliptic filter based on the entered specifications and places the transfer function (i.e. numerator, denominator, gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)`, `getden(Hd)` and `getgain(Hd)` will extract the numerator, denominator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Examples

```
ClearH1;  // clear primary filter from cascade
ShowH2DesignMarkers;   // show DM on chart

Main()

Rp=1;
Rs=80;
F={50,120};
Hd=ellip(0,F,Rp,Rs,"lowpass","symbolic");

F={50,80,100,120};
Hd=ellip(0,F,Rp,Rs,"bandpass","symbolic");


Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

2.5.2. butter

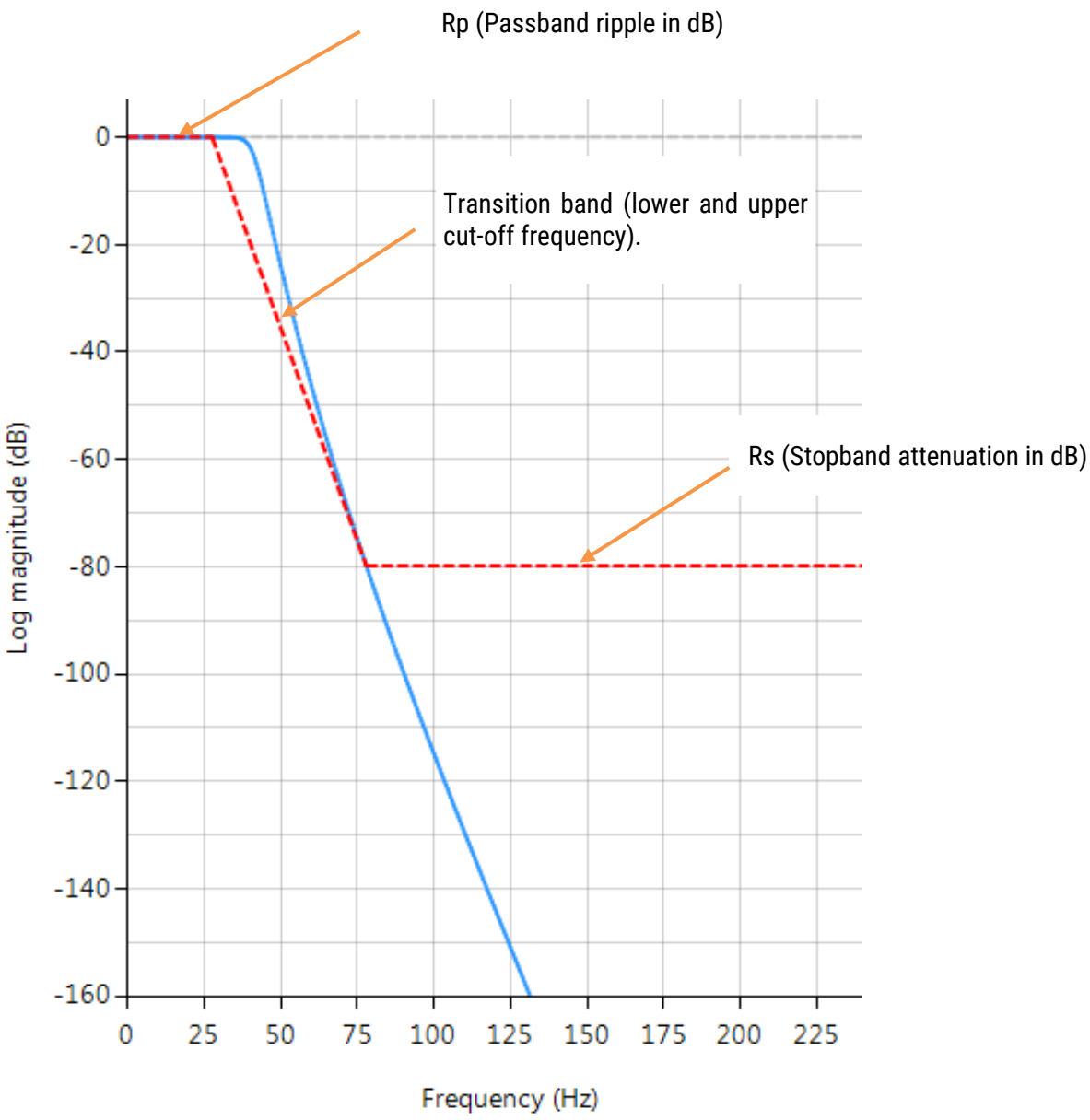
Syntax

```
Hd = butter (Order, Frequencies, Rp, Rs, Type, DFormat)
```

Description

Classic IIR Butterworth filter design.

- Smooth monotonic response (no ripple).
- Slowest roll-off for equivalent order.
- Highest order of all supported prototypes.



`Hd = butter (Order, Frequencies, Rp, Rs, Type, DFormat)`

`Order`: may be specified up to `20` (professional) and up to `10` (educational) edition. Setting the `Order` to `0`, enables the automatic order determination algorithm.

`Frequencies`: lowpass and highpass filters have one transition band, and in as such require two frequencies (i.e. lower and upper cut-off frequencies of the transition band). For bandpass and bandstop filters, four frequencies are required (i.e. two transition bands). All frequencies must be ascending in order and < Nyquist (see the example below).

`Rp`: Passband ripple in dB. This is somewhat of a misnomer, as the Butterworth filter has a maximally flat passband. A good default value is 0.001dB, but increasing this value will affect the position of the filter's lower cut-off frequency.

`Rs`: Stopband attenuation in dB. This is somewhat of a misnomer, as the Butterworth filter has a maximally flat stopband, which means that the stopband attenuation (assuming the correct filter order is specified) will be ≥ stopband specification.

`Type`: The Butterworth method facilitates the design of `lowpass`, `highpass`, `bandpass` and `bandstop` filters respectively.

`Hd`: the Butterworth method designs an IIR Butterworth filter based on the entered specifications and places the transfer function (i.e. numerator, denominator, gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)`, `getden(Hd)` and `getgain(Hd)` will extract the numerator, denominator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Examples

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;   // show DM on chart

Main()

Rp=0.001;
Rs=80;
F={50,120};
Hd=butter(0,F,Rp,Rs,"lowpass","symbolic");

F={50,80,100,120};
Hd=butter(0,F,Rp,Rs,"bandpass","symbolic");


Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

2.5.3. cheby1

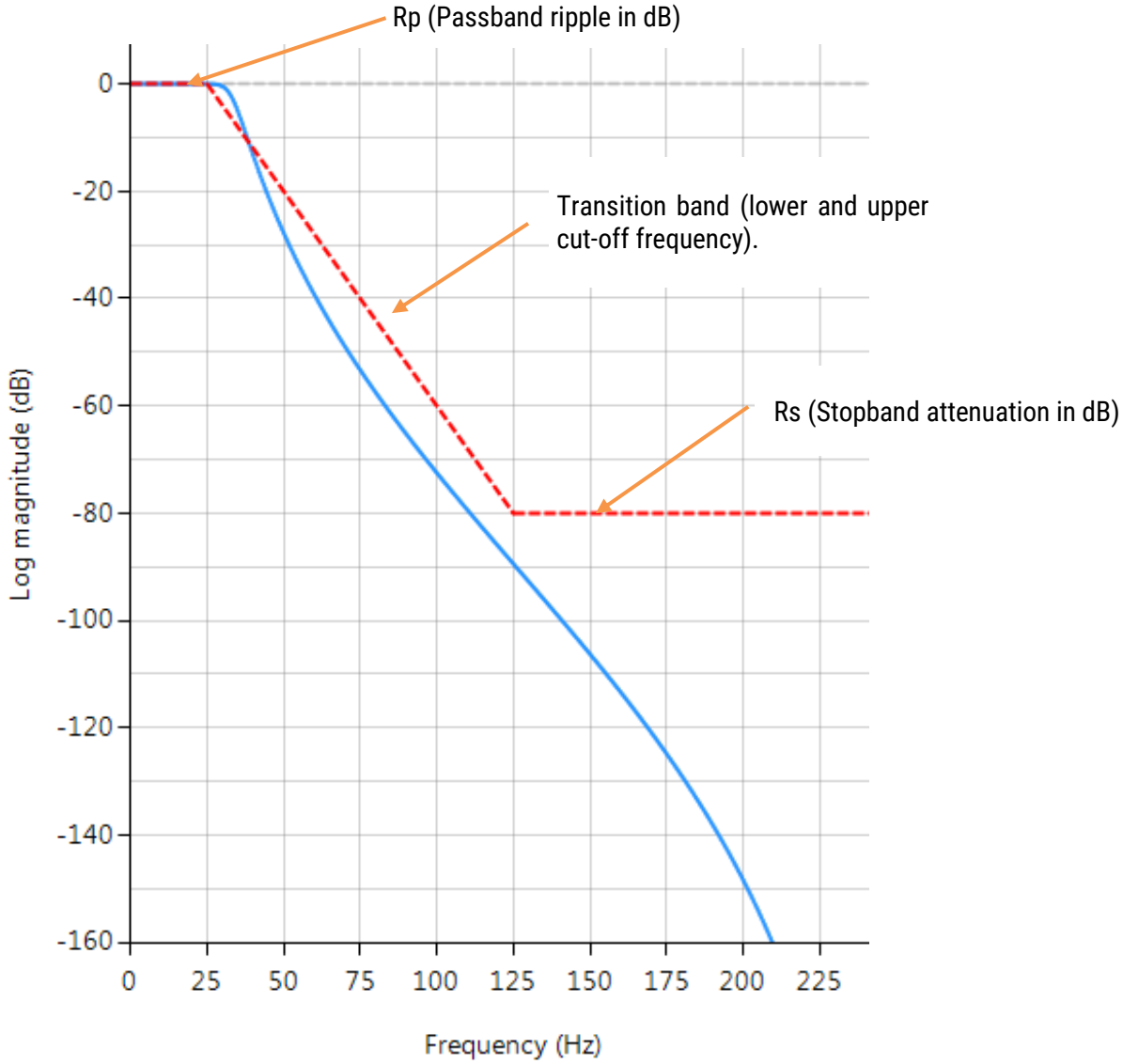
Syntax

```
Hd = cheby1 (Order, Frequencies, Rp, Rs, Type, DFormat)
```

Description

Classic IIR Chebyshev Type I filter design.

- Maximally flat stopband.
- Faster roll off (passband to stopband transition) than Butterworth.



```
Hd = cheby1 (Order, Frequencies, Rp, Rs, Type, DFormat)
```

`Order`: may be specified up to `20` (professional) and up to `10` (educational) edition. Setting the `Order` to `0`, enables the automatic order determination algorithm.

`Frequencies`: lowpass and highpass filters have one transition band, and in as such require two frequencies (i.e. lower and upper cut-off frequencies of the transition band). For bandpass and bandstop filters, four frequencies are required (i.e. two transition bands). All frequencies must be ascending in order and < Nyquist (see the example below).

`Rp`: Passband ripple in dB.
`Rs`: Stopband attenuation in dB. This is somewhat of a misnomer, as the Chebyshev Type I filter has a maximally flat stopband, which means that the stopband attenuation (assuming the correct filter order is specified) will be ≥ stopband specification.

`Type`: The Chebyshev Type I method facilitates the design of `lowpass`, `highpass`, `bandpass` and `bandstop` filters respectively.

`Hd`: the Chebyshev Type I method designs an IIR Chebyshev Type I filter based on the entered specifications and places the transfer function (i.e. numerator, denominator, gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)`, `getden(Hd)` and `getgain(Hd)` will extract the numerator, denominator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Examples

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;   // show DM on chart

Main()

Rp=1.4;
Rs=80;
F={50,120};
Hd=cheby1(0,F,Rp,Rs,"lowpass","symbolic");

F={50,80,100,120};
Hd=cheby1(0,F,Rp,Rs,"bandpass","symbolic");


Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

2.5.4. cheby2

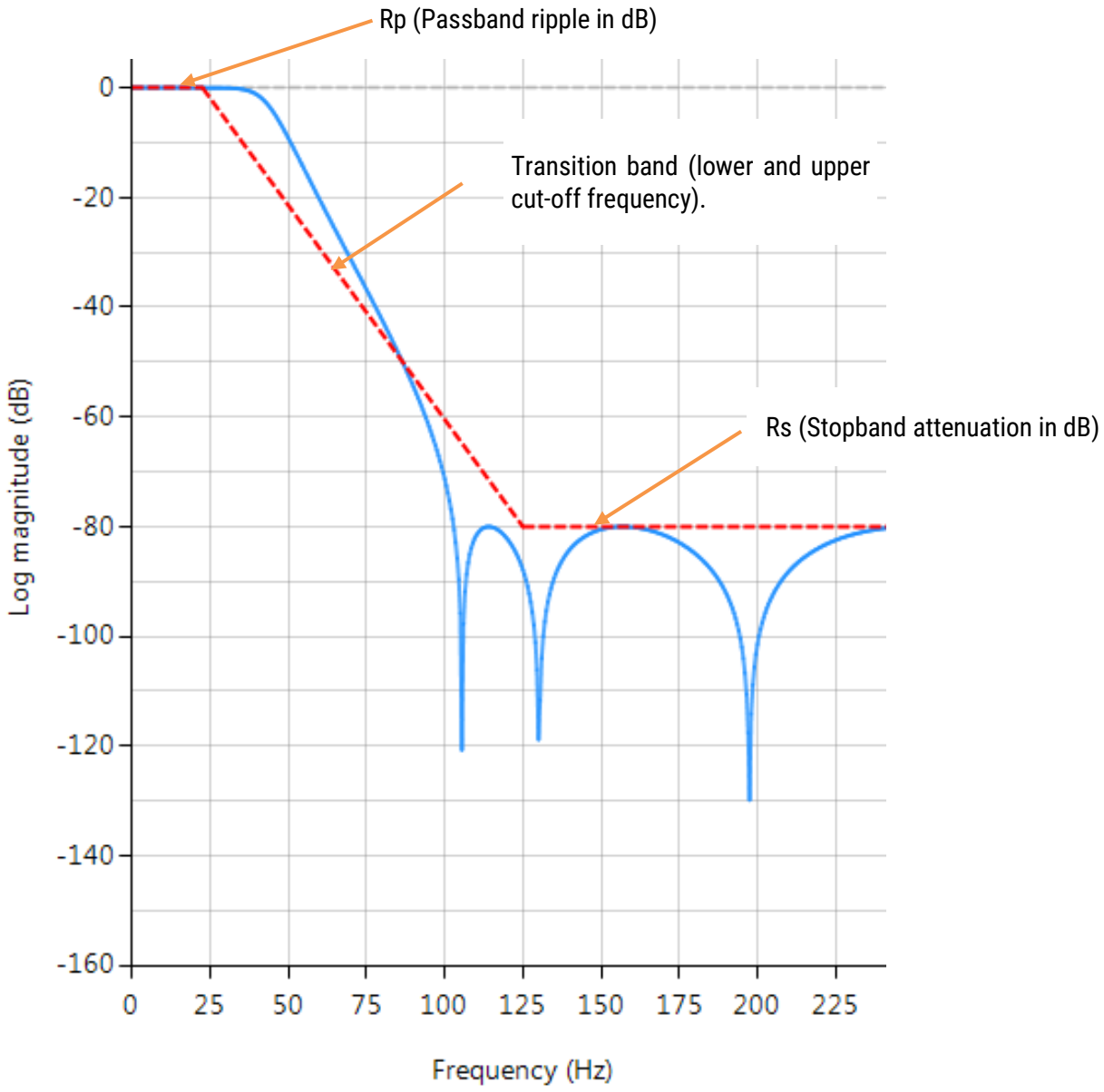
Syntax

```
Hd = cheby2 (Order, Frequencies, Rp, Rs, Type, DFormat)
```

Description

Classic IIR Chebyshev Type II filter design.

- Maximally flat passband.
- Slower roll off (passband to stopband transition) than Chebyshev Type I.



```
Hd = cheby2 (Order, Frequencies, Rp, Rs, Type, DFormat)
```

`Order`: may be specified up to `20` (professional) and up to `10` (educational) edition. Setting the `Order` to `0`, enables the automatic order determination algorithm.

`Frequencies`: lowpass and highpass filters have one transition band, and in as such require two frequencies (i.e. lower and upper cut-off frequencies of the transition band). For bandpass and bandstop filters, four frequencies are required (i.e. two transition bands). All frequencies must be ascending in order and < Nyquist (see the example below).

`Rp`: Passband ripple in dB. This is somewhat of a misnomer, as the Chebyshev Type II filter has a maximally flat passband. A good default value is 0.001dB, but increasing this value will affect the position of the filter's lower cut-off frequency.

`Rs`: Stopband attenuation in dB.

`Type`: The Chebyshev Type II method facilitates the design of `lowpass`, `highpass`, `bandpass` and `bandstop` filters respectively.

`Hd`: the `cheby2` method designs an IIR Chebyshev Type II filter based on the entered specifications and places the transfer function (i.e. numerator, denominator, gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)`, `getden(Hd)` and `getgain(Hd)` will extract the numerator, denominator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Examples

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;   // show DM on chart

Main()

Rp=1;
Rs=80;
F={50,120};
Hd=cheby2(0,F,Rp,Rs,"lowpass","symbolic");

F={50,80,100,120};
Hd=cheby2(0,F,Rp,Rs,"bandpass","symbolic");


Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

2.5.5. bessel

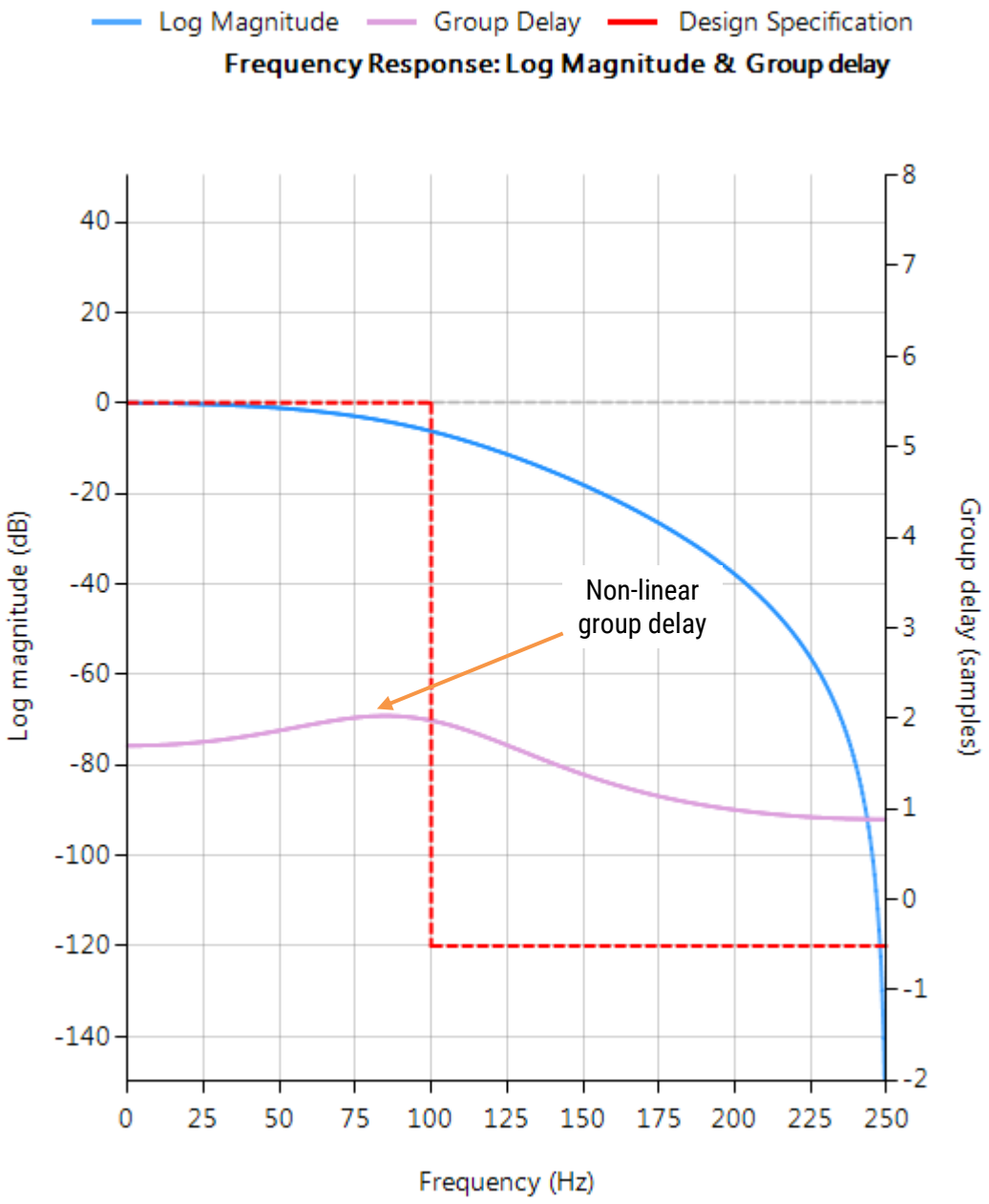
Syntax

```
Hd = bessel (Order, Frequencies, Type, DFormat)
```

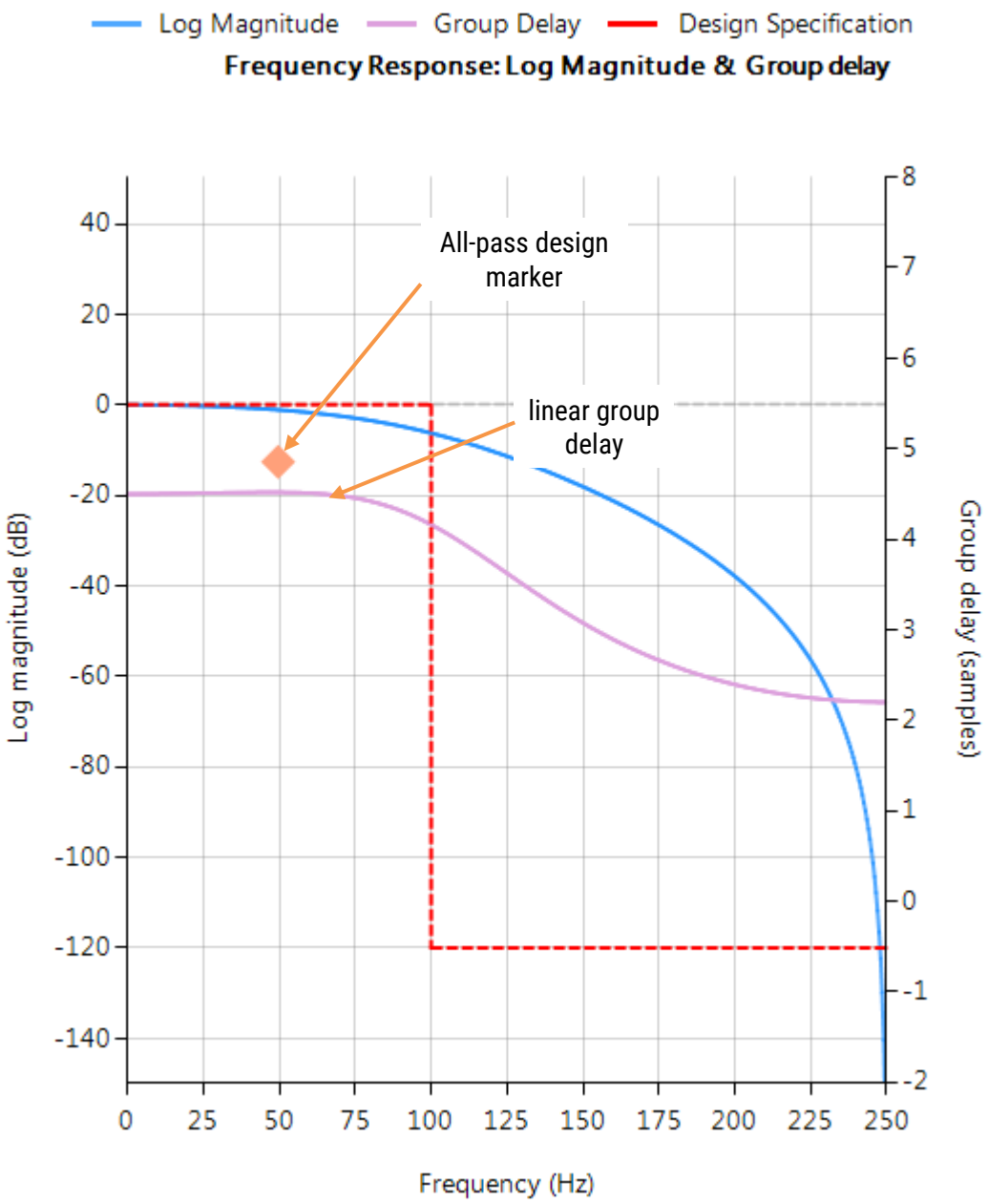
Description

IIR Bessel filter design. Analog Bessel filters have a constant group delay in the passband, which is very desirable for a variety of measurement applications. The method implemented within FilterScript uses the Bilinear transform which modifies the standard analog Bessel characteristic, and as a consequence it does not preserve the constant group delay characteristic in the passband. You may equalise the group delay by using an all-pass filter, either in FilterScript or in the main tool with the all-pass filter designer.

- Near constant group delay in the passband.
- Slower roll-off than other prototypes.



Cascading a single all-pass filter (designed with the all-pass filter designer) with the Bessel filter, linearizes the group delay in the passband – see below.



`Hd = bessel (Order, Frequencies, Type, DFormat)`

`Order`: may be specified up to `20` (professional) and up to `10` (educational) edition.

`Frequencies`: lowpass and highpass are specified via one cut-off frequency, whereas bandpass and bandstop filters require two frequencies (i.e. lower and upper cut-off). All frequencies must be ascending in order and < Nyquist (see the example below).

`Type`: The Bessel method facilitates the design of `lowpass`, `highpass`, `bandpass` and `bandstop` filters respectively.

`Hd`: the Bessel method designs an IIR Bessel filter based on the entered specifications and places the transfer function (i.e. numerator, denominator, gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)`, `getden(Hd)` and `getgain(Hd)` will extract the numerator, denominator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Examples

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;   // show DM on chart

Main()

F={75};
Order=5;
Hd=bessel(Order,F,"lowpass","symbolic");

F={50,100};
Hd=bessel(Order,F,"bandpass","symbolic");


Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

2.5.6. notch

Syntax

```
Hd = notch (Fo, BW, DFormat)
```

Description

IIR notch filter design, defined as:

$$H(z) = \frac{1 - 2\cos w_o z^{-1} + z^{-2}}{1 - 2r\cos w_o z^{-1} + r^2 z^{-2}}$$

where, $w_o = \frac{2\pi f_o}{fs}$ controls the centre frequency, f_o of the notch, and $r = 1 - \frac{\pi BW}{fs}$ controls the bandwidth (-3dB point) of the notch.

`Fo`: centre frequency of the notch.

`BW`: Bandwidth (-3dB point) of the notch. Where, $BW \leq \frac{F_o}{4}$

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Example

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;   // show DM on chart

interface BW={0.1,10,.1,1};

Main()

F=50;
Hd=notch(F,BW,"symbolic");

Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

2.5.7. dcremover

Syntax

```
Hd = dcremover (Fc, DFormat)
```

Description

Implements a first order IIR highpass filter (DC component remover), defined as:

$$H(z) = \frac{1}{(w+1)}\left[\frac{1-z^{-1}}{1+\frac{w-1}{w+1}z^{-1}}\right]$$

`Fc`: -3dB cut-off frequency, given by: $\mathrm{w} = \tan\left(\frac{\pi F_c}{fs}\right)$

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Example

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;   // show DM on chart

Main()

F=5;
Hd=dcremover(Fc,"symbolic");

Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

2.6. FIR design methods

A summary of all FIR filter design methods that are supported is given in this section.

2.6.1. movaver

Syntax

```
Hd = movaver(Order,DFormat)
```

Description

Moving average FIR filter design. The moving average (MA) filter is probably one of the most widely used FIR filters due to its conceptual simplicity and ease of implementation. However, despite its simplicity, the moving average filter is optimal for reducing random noise while retaining a sharp step response. Where a simple rule of thumb states that the amount of noise reduction is equal to the *square-root of the number of points in the average*. For example, an MA of length 9 will result in a factor 3 noise reduction.

`Order`: may be specified up to `499` (professional) and up to `128` (educational) edition.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

```
ClearH1;  // clear primary filter from cascade

Main()

Hd=movaver(8,"symbolic");

Num = getnum(Hd);    // define numerator coefficients
Den = {1};           // define denominator coefficients
Gain = getgain(Hd); // define gain
```

2.6.2. fircombparams

Syntax

```
P = fircombparams(fc)
```

Description

FIR comb filter parameter design. Returns the required sampling frequency and filter order needed to satify the specification, where:

```
P(0,0) = required order
P(0,1) = required Fs
```

and `fc` is the required notch frequency in Hz. The results may be used with `fircombfilter` function, as shown below.

2.6.3. fircombfilter

Syntax

```
Hd = fircombfilter(Order, Alpha, DFormat)
```

Description

FIR comb filter design.

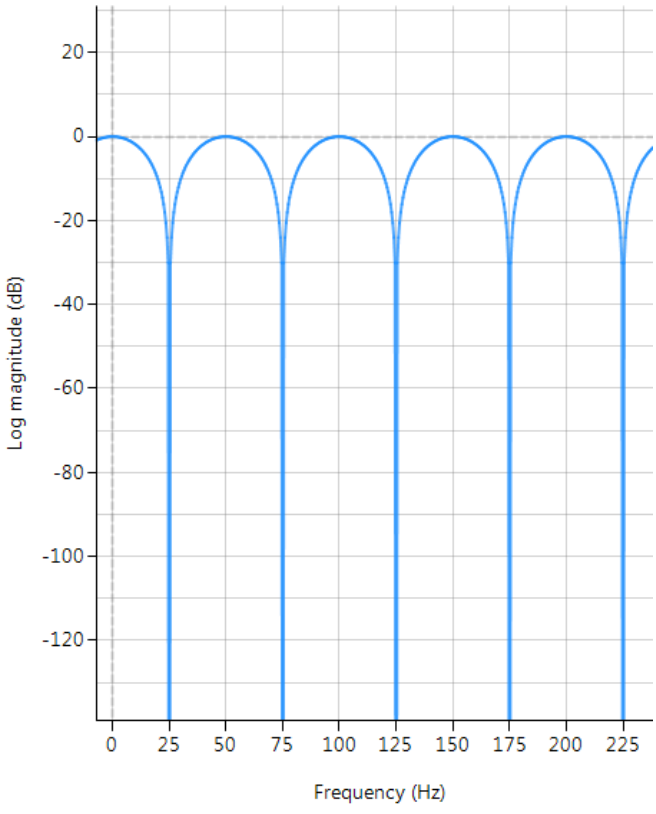
`Order`: may be specified up to `499` (professional) and up to `128` (educational) edition.

`Alpha`: The frequency response of a comb filter consists of a series of regularly-spaced troughs, giving the appearance of a comb. Where the spacing of each trough appears at either odd or even harmonics of the desired fundamental frequency. Thus, an FIR comb filter can be described by the following transfer function:

$$H(z) = 1 + \alpha z^{-L}$$

where, α is used to set the Q (bandwidth) of the notch and may be either positive or negative depending on what type of frequency response is required. In order to elaborate on this, negative values of α have their first trough at DC and their second trough at the fundamental frequency. Clearly this type of comb filter can be used to remove any DC components from a measured waveform if so required. All subsequent troughs appear at even harmonics up to and including the Nyquist frequency.

Positive values of α on the other hand, only have troughs at the fundamental and odd harmonic frequencies, and as such cannot be used to remove any DC components.



`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Example

```
ClearH1;  // clear primary filter from cascade
interface fc = {fs/8,fs/2,fs/20,fs/8};
interface alpha = {-1,0.99,0.01,0.7};

Main()

P=fircombparams(fc); //P(0,0) = order, P(0,1) = required Fs
Hd=fircomb(P(0,0),alpha,"symbolic");

Num=getnum(Hd);
Den=getden(Hd);
Gain=getgain(Hd);
```

2.6.4. firwin

Syntax

```
Hd = firwin(Order,Frequencies,Window,Type,DFormat)
```

Description

FIR filter design based on the Window method.

`Order`: may be specified up to `499` (professional) and up to `128` (educational) edition.

`Frequencies`: lowpass and highpass are specified via one cut-off frequency, whereas bandpass, bandstop and hilbert filters require two frequencies (i.e. lower and upper cut-off). All frequencies must be ascending in order and < Nyquist (see the example below).

`Window`: The `firwin` method supports the following window functions: `rectangular`, `blackman`, `blackmanharris`, `hamming`, `hanning`, `flattop` and `Chebyshev`.

`Type`: The `firwin` method facilitates the design of `lowpass`, `highpass`, `bandpass`, `bandstop and hilbert` filters respectively.

`Hd`: the `firwin` method designs an FIR window filter based on the entered specifications and places the transfer function (i.e. numerator and gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)` and `getgain(Hd)` will extract the numerator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Examples

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;

interface L = {10,400,10,50};

Main()

F={40,90};
Hd=firwin(L,F,"hamming","bandstop","numeric"); // Bandpass with a Hamming window
Hd=firwin(L,F,"hanning","hilbert","numeric");  // Hilbert with a Hanning Window

Num=getnum(Hd);
Den={1};
Gain=getgain(Hd);
```

2.6.5. firarb

Syntax

```
Hd = firarb(Order,Amplitude,Frequencies,Window,DFormat)
```

Description

Designs an FIR window based filter with an arbitrary magnitude response.

`Order`: may be specified up to `499` (professional) and up to `128` (educational) edition.

`Amplitude`: a vector of the magnitude specification in dB.

`Frequencies`: a vector of the frequency specification. The first element must be 0 and the last element equal to Nyquist – see below.

`Window`: The `firarb` method supports the following window functions: `rectangular`, `blackman`, `blackmanharris`, `hamming`, `hanning`, `flattop` and `Chebyshev`.

`Hd`: the `firarb` method designs an FIR filter based on the entered specifications and places the transfer function (i.e. numerator and gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)` and `getgain(Hd)` will extract the numerator and gain coefficients respectively – see below.

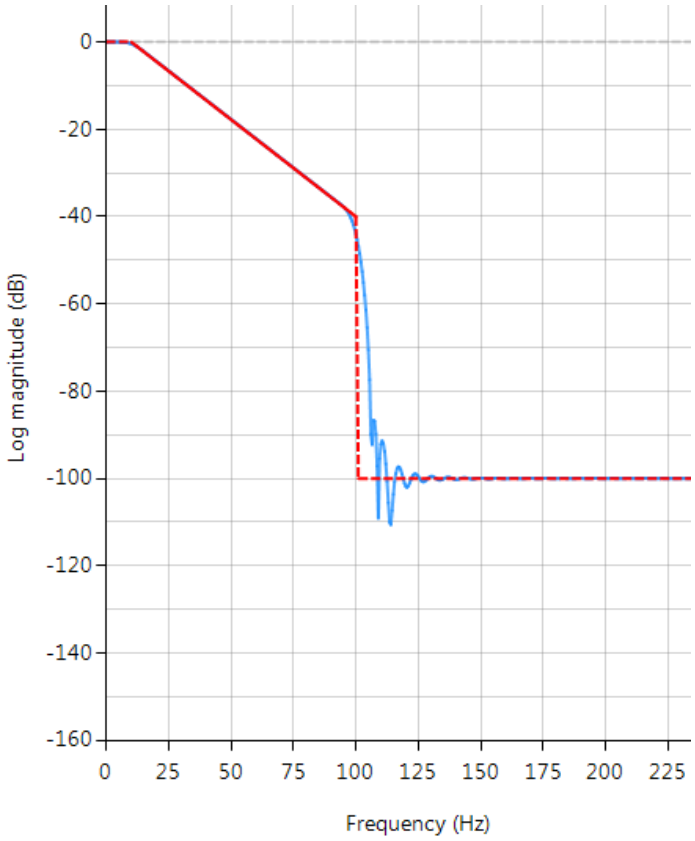
`DFormat`: allows you to specify the display format of resulting digital filter object.

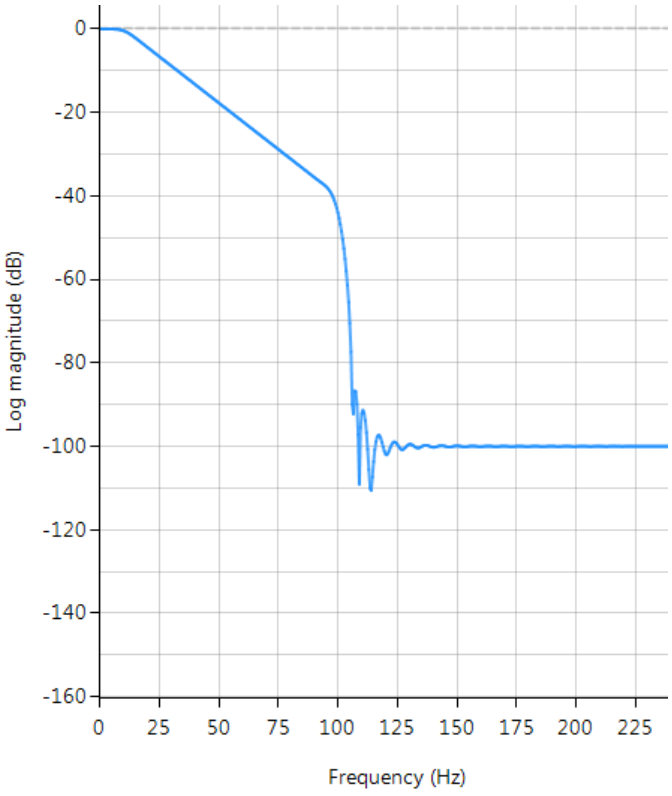
`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`Void`	Create a filter object, but do not display output.

Example

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;

interface L = {10,400,10,50};

Main()

A=series(0,-1,-40);
F=series(10,90/(length(A)-1),100);
A={0,A,-100,-100};  //  specify arb response
F={0,F,101,fs/2};   //

Hd=firarb(L,A,F,"hanning","numeric");

Num=getnum(Hd);
Den={1};
Gain=getgain(Hd);
```

Running the script shown pre-leaf, we obtain the following plots (with and without the design markers). Where, it can be seen that the design specifications have been met.





2.6.6. firkaiser

Syntax

```
Hd = firkaiser(Frequencies,Rs,Type,DFormat)
```

Description

Designs an FIR filter based on the Kaiser window method.

The method automatically determines the required filter order (`499` (professional) and up to `128` (educational) edition) and returns coefficients in `Hd`.

`Frequencies`: lowpass and highpass are specified via one cut-off frequency, whereas bandpass, bandstop and hilbert filters require two frequencies (i.e. lower and upper cut-off). All frequencies must be ascending in order and < Nyquist (see the example below).

`Type`: The `firkaiser` method facilitates the design of `lowpass`,`highpass`,`bandpass`, `bandstop`, `hilbert3` (Type 3 Hilbert), `hilbert4` (Type 4 Hilbert), `integrator3`, `integrator4`, `differentiator3` and `differentiator4` filters respectively.

`Hd`: the `firkaiser` method designs an FIR window filter based on the entered specifications and places the transfer function (i.e. numerator and gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)` and `getgain(Hd)` will extract the numerator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Example

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;

interface F = {10,80,2,40}; // frequency spec (2 bands for bandstop)
interface TW ={10,40,2,20}; // Band transition width

Main()

Freq={F,F+TW,100+TW,125+TW};   // frequency specification
Hd=firkaiser(Freq,60,"bandstop","numeric");

Num=getnum(Hd);
Den={1};
Gain=getgain(Hd);
```

2.6.7. firgauss

Syntax

```
Hd = firgauss(L,Gain,Alpha,DFormat)
```

Description

Designs an FIR Gaussian lowpass filter. Gaussian filters are good pulse shaping filters, and as such are typically used in communication systems, as they have no overshoot and fast transitions. The function returns a Gaussian window of length `L` with a standard deviation, σ

$$\sigma = \frac{L-1}{2\alpha}$$

As seen, the width of the window (standard deviation) is inversely related to `alpha(`α`)`, i.e. a smaller value of α produces a tighter transition frequency band in the frequency domain and vice versa. A good default value is 2.5.

`Hd`: the `firgauss` method designs an FIR Gaussian lowpass filter based on the entered specifications and places the transfer function (i.e. numerator and gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)` and `getgain(Hd)` will extract the numerator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Example

```
ClearH1;  // clear primary filter from cascade
interface L = {2,100,1,14};          // filter length
interface alpha = {0.5,10,0.1,2.5};  // standard deviation = (L - 1)/(2*alpha)

Main()

Hd=firgauss(L,1,alpha,"symbolic");

Num=getnum(Hd);
Den={1};
Gain=getgain(Hd);
```

2.6.8. savgolay

Syntax

```
Hd = savgolay(Order, Polyfit, DFormat)
```

Description

Design an FIR Savitzky-Golay lowpass smoothing filter. Savitzky-Golay (polynomial) smoothing filters or least-squares smoothing filters are generalizations of the FIR average filter that can better preserve the high-frequency content of the desired signal, at the expense of not removing as much noise as an FIR average (see `movaver` for more information). The particular formulation of Savitzky-Golay filters preserves various moment orders better than other smoothing methods, which tend to preserve peak widths and heights better than Savitzky-Golay.

`Order`: may be specified up to `499` (professional) and up to `128` (educational) edition.

`Polyfit`: Polynomial fit, which must be < `Order+1`

`Hd`: the `savgolay` method designs an FIR Savitzky-Golay lowpass smoothing filter based on the entered specifications and places the transfer function (i.e. numerator and gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)` and `getgain(Hd)` will extract the numerator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

Examples

```
ClearH1;  // clear primary filter from cascade

interface L = {2, 50,2,24};
interface P = {2, 10,1,4};

Main()

Hd=savgolay(L,P,"numeric");  // Design Savitzky-Golay lowpass filter
Num=getnum(Hd);
Den={1};
Gain=getgain(Hd);
```

2.6.9. cplxfreqshift

Syntax

`Hds=cplxfreqshift(Hd,Fo,DFormat)`

Description

Apply a Complex frequency shift transformation to the digital transfer function object (Hd) centred at frequency point, `Fo`. Where, `Fo` is specified in same base unit as the sampling frequency, `Fs`, i.e. Hz, kHz etc. The function returns a new (frequency shifted) digital transfer object in `Hds`.

This transform may be used for designing a complex bandpass filter, whereby a real lowpass filter's frequency response is shifted up or down the spectrum in order to produce a complex bandpass filter. Complex bandpass filters are useful for communication applications and signal property analysis, as they provide a simple way of obtaining the instantaneous frequency, phase and amplitude of sinusoid.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`void`	Create a filter object, but do not display output.

```
ClearH1;  // clear primary filter from cascade

interface f = {1,200,1,2};  // define cut-off frequency
interface fo = {0,200,10,10}; // define centre frequency of bandpass
interface Rs = {10,100,5,60};  // define stopband attenuation
interface BW = {1,100,1,5};   // define bandwidth

Main()

fc={f,f+BW}; // define a transition band
Rp=0.001;    // define passband ripple in dB
Hd=butter(5,fc,Rp,Rs,"lowpass","void");  // 5th order Type II Chebyshev lowpass
Hd=cplxfreqshift(Hd,fo,"symbolic");      // shift lowpass filter poles and zeros,
                                          // and make a bandpass

Num=getnum(Hd);   // get numerator coefficients
Den=getden(Hd);   // get denominator
Gain=getgain(Hd); // get gain
```

2.7. Analog → digital filter design: Laplace transforms

Laplace analog transfer functions may be entered and converted into their digital equivalents via the following commands:

2.7.1. analogtf

Syntax

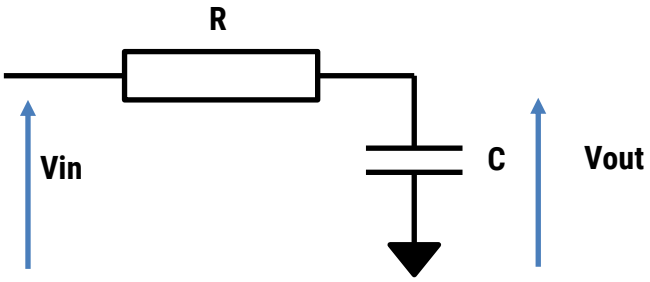
```
Ha=analogtf(ANum, ADen, AGain, DFormat)
```

Description

Define an analog filter object. Where, the `ANum` and `ADen` vectors are the Laplace transfer function coefficients in descending order.

Example

A first order analog lowpass filter may be designed by an RC network:



$$\frac{Vout(s)}{Vin(s)} = \frac{w_c}{s + w_c}; \quad w_c = \frac{1}{RC}$$

Choosing R=100kΩ and C=100nF, gives cut-off at 100rad/s or ≈ 15.9Hz. The analog transfer function may be simply implemented in FilterScript (output shown on the right) as:

```
Main()

wc=100;  // 1/(R*C);
ANum={0,1};
ADen={1,wc};
AGain=wc;
Ha=analogtf(ANum,ADen,AGain,"symbolic");
```

```
-> wc =  100.00000

-> ANum (2x1) = { 0.0000000,
                  1.0000000}

-> ADen (2x1) = { 1.0000000,
                  100.00000}

-> AGain =  100.00000

-> Ha (s) =  Analog transfer function object

                      1.0000000
  100.00000 x ---------------------
                  s + 100.00000
```

This analog filter object may be now transformed into a digital filter via the Bilinear or Match Z-transform methods. See the example script `AnalogRCfilter.afs` for more information.

2.7.2. bilinear

Syntax

```
Hd=bilinear(Ha,Fp,DFormat)
```

Description

Convert an analog filter object to its digital equivalent using the Bilinear transform (S→Z transformation).

The Bilinear z-transform (BZT), simply converts an analog transfer function, H(s) into a discrete transfer function, H(z) by replacing all s terms with the following:

$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}$$

where, T is the discrete system's sampling period. However, substituting $s = j\Omega$ and $z = e^{jwT}$ into the BZT equation and simplifying, notice that there is actually a non-linear relationship between the analog, Ω and discrete, w frequencies. This relationship is shown below, and is due to the nonlinearity of the arctangent function.

$$w = 2\,tan^{-1}\left(\frac{\Omega T}{2}\right)$$

Analysing the equation, it can be seen that the equally spaced analog frequencies in the range $-\infty<\Omega<\infty$ are nonlinearly compressed in the frequency range $-\pi<w<\pi$ in the discrete domain. This relationship is referred to as frequency warping, and may be compensated for by pre-warping the analog frequencies by:

$$\Omega_c = \frac{2}{T}\,tan\left(\frac{\Omega_d\,T}{2}\right)$$

where, Ω_c is the compensated or pre-warped analog frequency, and Ω_d is the desired analog frequency.

Pre-warping and scaling: In order to better match the analog transfer function, you may specify a single pre-warping frequency (matching frequency) via the `Fp` parameter. A further internal gain scaling operation is automatically performed in order to match the analog gain specification.

`Hd`: the `bilinear` method designs an IIR filter based on the analog filter object, `Ha` and places the transfer function (i.e. numerator and gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)`, `getden(Hd)` and `getgain(Hd)` will extract the numerator, denominator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`Void`	Create a filter object, but do not display output.

Example

```
ClearH1;  // clear primary filter from cascade
interface wc={20,200,10,100}; // wc=1/(R*C);

Main()

// define analog RC filter
ANum={0,1};
ADen={1,wc};
AGain=wc;
Ha=analogtf(ANum,ADen,AGain,"symbolic");

Hd=bilinear(Ha,wc/Twopi,"symbolic");

Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

See page 32 for a description of the analog RC filter model used in this example.

2.7.3. mztrans

Syntax

`Hd=mztrans(Ha,DFormat)`

Description

Convert an analog filter object to its digital equivalent using the Match Z-transform (S→Z transformation).

The matched-z transformation **converts the poles and zeros from the analog transfer function directly into poles and zeros in the z-plane**. The transformation is described below, where T is the sampling rate.

$$\frac{\prod_{k=1}^{q}(s+b_k)}{\prod_{k=1}^{p}(s+a_k)} \rightarrow \frac{\prod_{k=1}^{q}(1-e^{-b_k T}z^{-1})}{\prod_{k=1}^{p}(1-e^{-a_k T}z^{-1})}$$

`Hd`: the `mztrans` method designs an IIR filter based on the analog filter object, `Ha` and places the transfer function (i.e. numerator and gain) into a digital filter object, `Hd`. The digital filter object can then be combined with other methods if so required. For a digital filter object, `Hd`, calling `getnum(Hd)`, `getden(Hd)` and `getgain(Hd)` will extract the numerator, denominator and gain coefficients respectively – see below.

`DFormat`: allows you to specify the display format of resulting digital filter object.

`symbolic`	Display a symbolic representation of the filter object. If the order > 10, the `symbolic` display option will be overridden and set to `numeric`.
`numeric`	Display a matrix representation of the filter object.
`Void`	Create a filter object, but do not display output.

Example

```
ClearH1;  // clear primary filter from cascade
interface wc={20,200,10,100}; // wc=1/(R*C);

Main()

// define analog RC filter
ANum={0,1};
ADen={1,wc};
AGain=wc;
Ha=analogtf(ANum,ADen,AGain,"symbolic");

Hd=mztrans(Ha,"symbolic");

Num = getnum(Hd); // define numerator coefficients
Den = getden(Hd); // define denominator coefficients
Gain = getgain(Hd); // define gain
```

See page 32 for a description of the analog RC filter model used in this example.

2.7.4. Miscellaneous analog filter design functions

Function	Syntax	Description
augment	Ha=augment(Ha1,Ha2)	Augment or merge two analog filter objects, and return the merged objects as a single object in Ha.
computegain	G=computegain(Ha,Fo)	Get the gain, G of the analog filter (Ha) at frequency point, Fo. Where, Fo is specified in Hz.
getnum	Num = getnum(Ha)	Get the numerator coefficients of analog filter object, Ha.
getden	Den = getden(Ha)	Get the denominator coefficients of analog filter object, Ha.
getgain	G = getgain(Ha)	Get the gain of analog filter object, Ha.

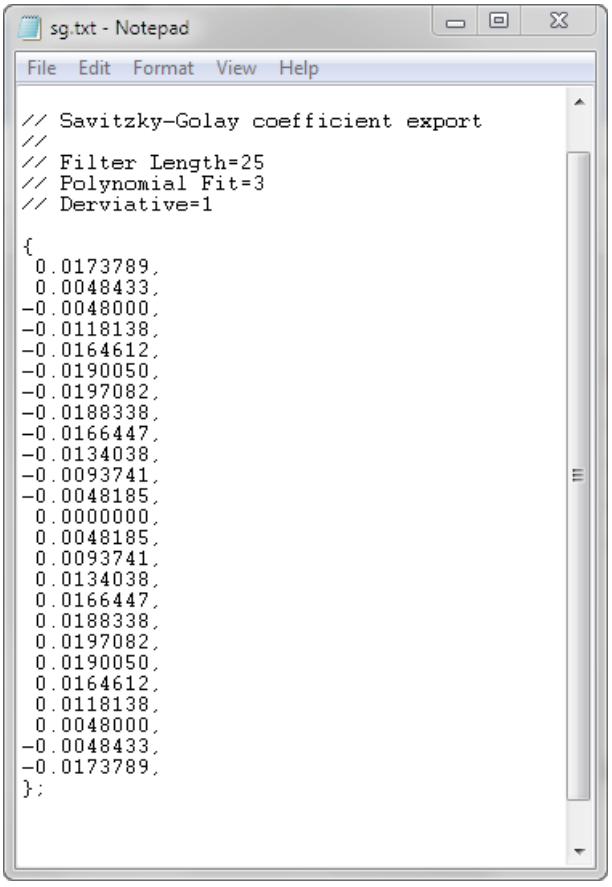
2.8. Loading an external coefficient datafile

You may import a vector of data/coefficients into FilterScript via the `importdata` function:

```
h=importdata(Filename)
```

Where, `Filename` must be the full pathname and filename entered in quotes, e.g.

```
h=importdata("c:\Temp\sg.txt");
```



All data must be single lined and followed by a comma. Complex data may be imported by using the `i` or `j` keyword.

The vector definition must be preceded with a `{` brace and closed with a `};`

A maximum of 200 values may be imported.

Comments may be placed anywhere, and must be preceded with the `//` keyword.

2.9. General syntax and data manipulation

Function	Description
General	All variables may contain upper and lower case characters, and numbers. e.g. `Num1`, `myGain`, `alpha15`
Interface Variables	The `interface` keyword must be used to define all interface variables.
Matrices	A generalised matrix is defined as `A(rows,columns)`. Although *matrix assignment is not supported*, certain vector operations may result in a matrix result, e.g. the vector multiplication: `A=a*transpose(a)`. All data indexes run from `0...N`, you may access a matrix element at row `R` and column `M` as: `y=A(R,M)`. However, you may also access a range of values using the `:` keyword, e.g. `Y=A(3:5,1:2)` which produces a new matrix `Y`. For modifying a value of a matrix/vector, use the `eldef` function, e.g. `a(2,1)=eldef(5)`
Vector assignment	By default, a vector is defined as an array with multiple rows and one column. It may contain expressions, variables and constants and must be enclosed in braces `{ }` with comma delimitation. Example: `b = {1,0,3.4,0,1};` Example: `A = {1,-2*cos(TwoPi*fc/fs),1};`

Vector manipulation	In order to accommodate transposed vectors, all vectors are defined as a generalised matrix, i.e. `A(rows,columns)`. By default, a vector of length `N` is defined as `A(N,1)`, whereas a transposed vector is defined as `A(1,N)`. As all data indexes run from `0...N`, you may access vector element `M` as: `y=A(M,0)`. However, you may also access a range of values using the `:` keyword, e.g. `y=A(3:5,0)`. For modifying a value of a vector, use the `eldef` function.
	
	Example
	
	`a = {1,0,3.4,0,1};` // assign five elements to vector `a`
	`a(2,0)=eldef(5);` // set element three to 5
	`y=a(0,0);` // get element zero and assign it to `y`

Data series	A real valued data series can be created with the following syntax:
	
	`y = series(min,step,max)`
	
	where, `step` represents the step size between `min` (minimum) and `max` (maximum).
	
	Example
	
	`a=series(-12,1,1.2);`

User comments	All user comments must be preceded with the `//` keyword. Where, the `/* */` syntax is not supported.

2.10. System variables and reserved constants

There are several system variables and constants which can be used in every script and expression.

Variable	Description
`fs`	The `fs` variable specifies the system sampling frequency in Hz, i.e. 50MHz is given as $50e^6$
`fsunits`	Returns the sampling frequency units, e.g. 500kHz would return `1e3` for kHz
`Ts`	The `Ts` variable specifies the system sampling period `Ts=1/fs`
`pi`	`3.14159265358979`
`Twopi`	`6.28318530717959`
`i`	Complex number token, $\sqrt{-1}$

2.11. Mandatory keywords

The following keywords **must be present** in every script.

Variable	Description
`Main()`	`Main()` is used to seperate the initialisation code from the "main" code - see section 1.1 for more information.
`Den`	`Den` specifies the denominator filter coefficients. This must be a vector.
`Num`	`Num` specifies the denominator filter coefficients. This must be a vector.
`Gain`	`Gain` specifies the filter gain. This must be real.

2.12. Optional Keywords

Variable	Description
`ClearH1`	The `ClearH1` keyword will delete the H1 filter from the cascade.
`ShowDM`	Show the design markers used for the design.

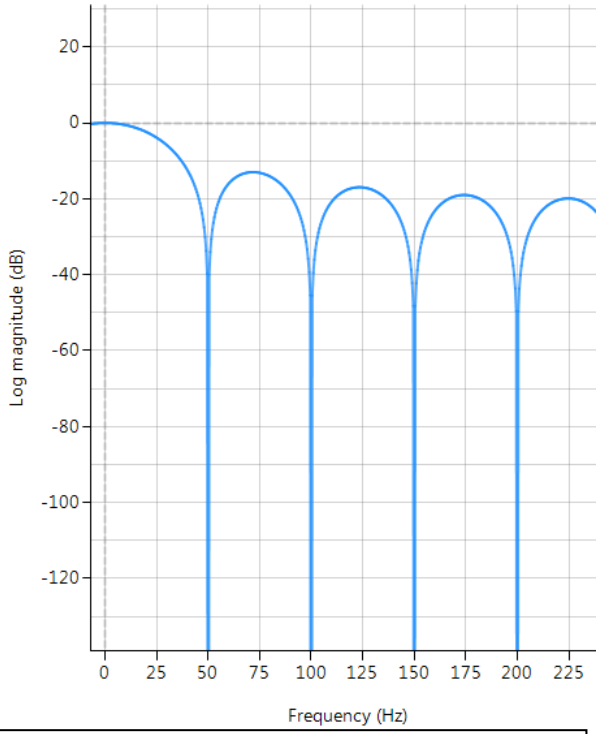
3. General example scripts

The following section is a collection of example scripts bundled with the software. All script files are `.afs` files which can be found in the `Scripts\Examples` directory. Please see our detailed online resource portal for more examples.

3.1. Moving average filter (movingaverage.afs)

The moving average (MA) filter is probably one of the most widely used FIR filters due to its conceptual simplicity and ease of implementation. However, despite its simplicity, the moving average filter is optimal for reducing random noise while retaining a sharp step response. Where a simple rule of thumb states that the amount of noise reduction is equal to the *square-root of the number of points in the average*. For example, an MA of length 9 will result in a factor 3 noise reduction.

Reference: Understanding Digital Signal Processing, Chapter 5, R. G. Lyons



The following script implements an adjustable length moving average filter. The interface variable `L` is used to set the filter length between `1` and `100`.

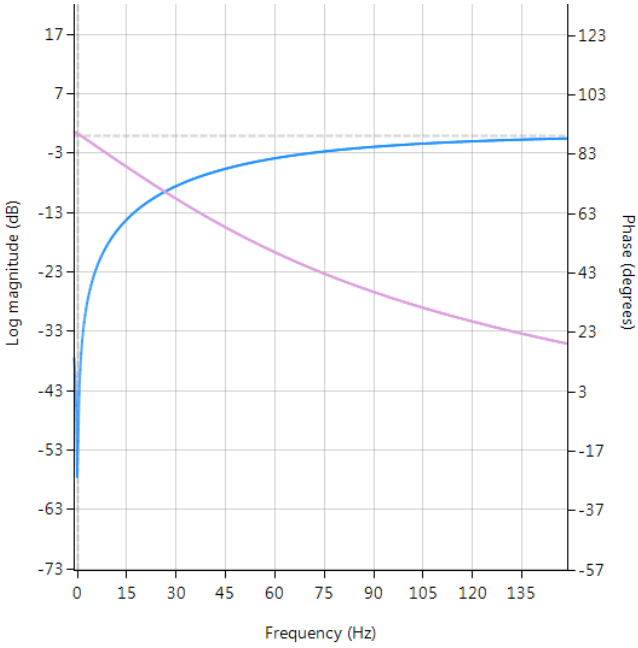
```
ClearH1;  // clear primary filter from cascade

interface L = {1,100,1,10};   // model length (order = length - 1)

Main()
Num = {ones(L)};        // moving average filter coefficients
Den = {1};
Gain = 1/L;
```

3.2. HPF (BilinearHPF.afs)

It is sometimes useful to transform an analogue filter into its digital/discrete equivalent. Although there are several transformation methods, the Bilinear z-transform (BZT) is a very popular method and is therefore used for this example. Central to the BZT concept is the S-Z transformation which maps an analogue transfer function, $H(s)$ into its digital equivalent $H(z)$:

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

where, *T* is the discrete system's sampling period. However, substituting $s = e^{j\Omega}$ and $z = e^{jw}$ into the above equation and simplifying, we see that there is actually a non-linear relationship between the analogue, Ω and discrete, *w* frequencies. This relationship is shown below and is due to the nonlinearity of the arctangent function.



$$w = 2tan^{-1}\left(\frac{\Omega T}{2}\right)$$

Design example

A first order Laplace highpass transfer function is given by:

$$H(s) = \frac{s}{s+w}; \mathrm{w} = \tan\left(\frac{\pi f}{fs}\right)$$

Applying the BZT to $H(s)$, we obtain:

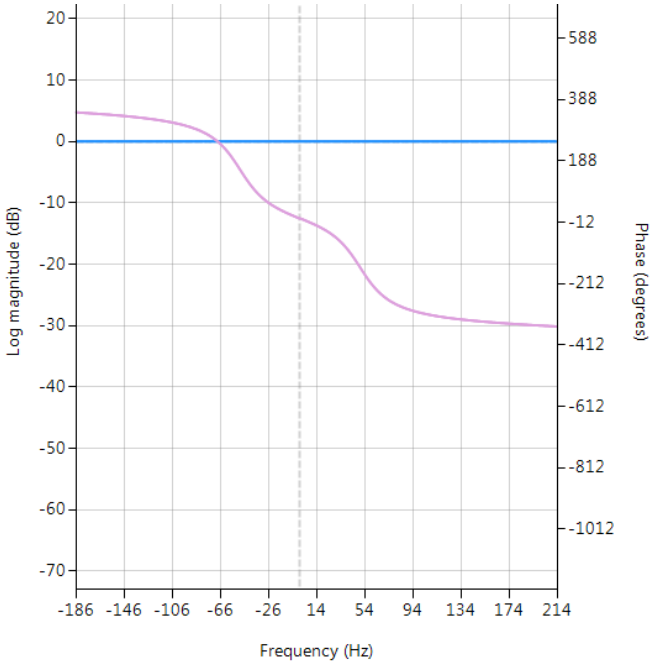
$$H(z) = \frac{1}{(w+1)}\left[\frac{1-z^{-1}}{1+\frac{w-1}{w+1}z^{-1}}\right]$$

The implementation of $H(z)$ is given below, where the cut-off frequency (-3dB point) is adjustable between $0 \leq f \leq fs/2$

```
ClearH1;  // clear primary filter from cascade
interface f = {0,fs/2,1,10}; // interface variable definition

Main()
w=tan(f*pi/fs);

Num = {1,-1}; // define numerator coefficients
Den = {1,(w-1)/(w+1)}; // define denominator coefficients
Gain = 1/(w+1); // define gain
```

3.3. Second order all-pass filter (SecondOrderAllPass.afs)

All-pass filters provide a simple way of altering/improving the phase response of an IIR without affecting its magnitude response. As such, they are commonly referred to as phase equalisers and have found particular use in digital audio applications.



A second order all-pass filter is defined as:

$$H(z) = \frac{r^2 - 2rcos\left(\frac{2\pi fc}{fs}\right)z^{-1} + z^{-2}}{1 - 2rcos\left(\frac{2\pi fc}{fs}\right)z^{-1} + r^2z^{-2}}$$

Notice how the numerator and denominator coefficients are arranged as mirror image (mirror-image pair) of one another.

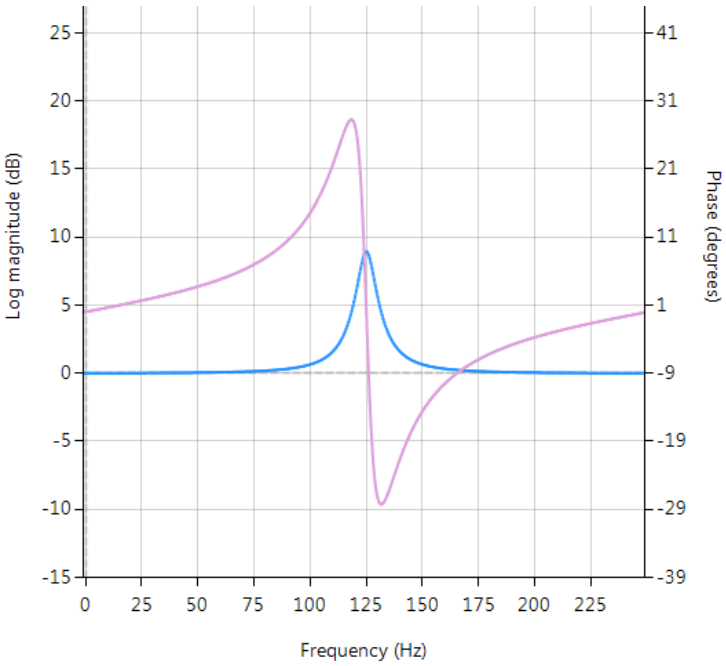
Reference: The digital All-pass Filter: A versatile signal processing building block, Regalia, Mitra et al., Proceedings IEEE, vol 76, January 1988.

The following script implements the symbolic transfer function with two interface variables `radius` and `fc`.

```
ClearH1;  // clear primary filter from cascade

interface radius = {0,2,0.01,0.5};    // radius value
interface fc = {0,fs/2,1,fs/10};     // frequency value

Main()
Num = {radius^2,-2*radius*cos(Twopi*fc/fs),1};  // mirror image pair
Den = reverse(Num);
Gain = 1;
```

3.4. Allpass Peaking/Bell filter (AllpassPeaking.afs)

A Bell or Peaking filter is a type of audio equalisation filter that boosts or attenuates the magnitude of a specified set of frequencies around a centre frequency in order to perform magnitude equalisation. As seen in the plot on the right-hand side, the filter gets its name from the shape of the its magnitude spectrum (blue line) which resembles a Bell curve.



A Bell filter can be constructed from an all-pass configuration (see section 3.3) by the following transfer function:

$$H(z) = \frac{(1+K) + A(z)(1-K)}{2}$$

where, $A(z)$ is the all-pass filter component:

$$H(z) = \frac{1}{2}\left[(1+K) + \underbrace{\frac{k_2 + k_1(1+k_2)z^{-1} + z^{-2}}{1 + k_1(1+k_2)z^{-1} + k_2 z^{-2}}}_{\text{all-pass filter}}(1-K)\right]$$

```
ClearH1;  // clear primary filter from cascade
interface BW = {0,2,0.1,0.5}; // filter bandwidth
interface fc = {0, fs/2,fs/100,fs/4}; // peak/notch centre frequency
interface K = {0,3,0.1,0.5}; // gain/sign

Main()

k1=-cos(2*pi*fc/fs);
k2=(1-tan(BW/2))/(1+tan(BW/2));

Pz = {1,k1*(1+k2),k2}; // define denominator coefficients
Qz = {k2,k1*(1+k2),1}; // define numerator coefficients
Num = (Pz*(1+K) + Qz*(1-K))/2;
Den = Pz;
Gain = 1;
```

3.5. AllpassNotch (AllpassNotch.afs)

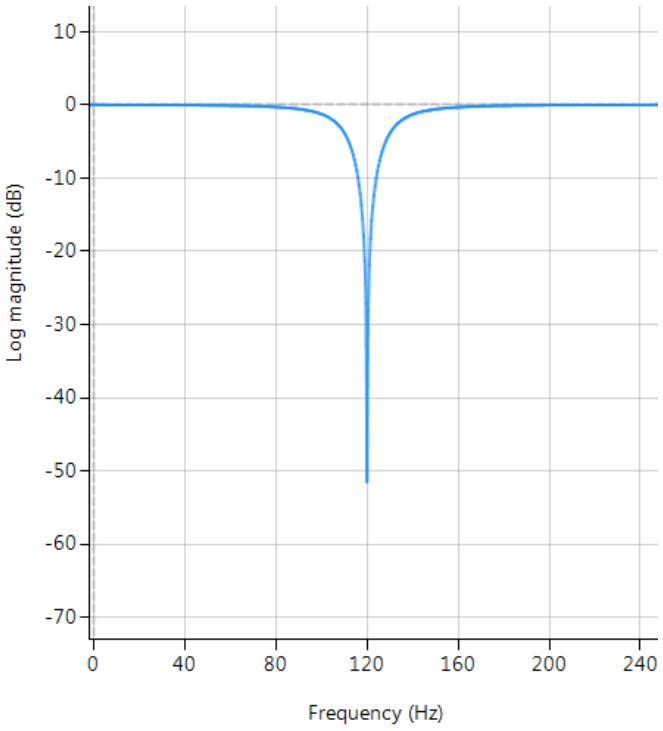
A notch filter can be constructed from an all-pass configuration (see section 3.3) by the following transfer function:

$$H(z) = \frac{1}{2}[1 + A(z)]$$

where, $A(z)$ is the all-pass filter component:

$$H(z) = \frac{1}{2}\left[1 + \underbrace{\frac{k_2 + k_1(1+k_2)z^{-1} + z^{-2}}{1 + k_1(1+k_2)z^{-1} + k_2 z^{-2}}}_{\text{all-pass filter}}\right]$$

$k_1 = -\cos\left(\frac{2\pi f}{fs}\right)$ controls the centre frequency of the notch, and $k_2 = \frac{1-\tan(BW/2)}{1+\tan(BW/2)}$ controls the bandwidth of the notch.

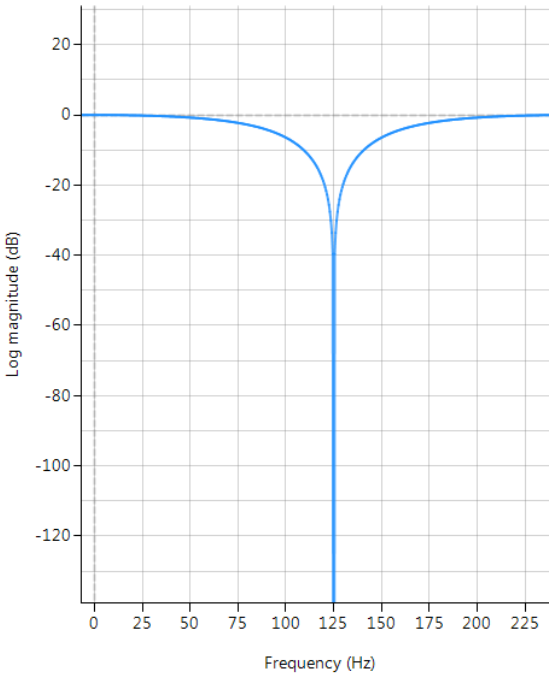


Reference: The digital All-pass Filter: A versatile signal processing building block, Regalia, Mitra et al., Proceedings IEEE, vol 76, January 1988.

```
ClearH1;  // clear primary filter from cascade
interface BW = {0,2,0.1,0.5}; // interface variable definition
interface fc = {0, fs/2,fs/100,fs/4};

Main()

k1=-cos(2*pi*fc/fs);
k2=(1-tan(BW/2))/(1+tan(BW/2));

Den = {1,k1*(1+k2),k2}; // define denominator coefficients
Num = {k2,k1*(1+k2),1}; // define numerator coefficients
Num = (Num+Den)/2;
Gain = Num(0,0)/Den(0,0);   // compsensate gain for normalisation
Num=Num/Num(0,0);   // normalise numerator
Den=Den/Den(0,0);   // normalise denominator
```

3.6. Notch (Notch.afs)

The primary purpose of a Notch filter is to attenuate (minimize) a specific frequency point in the spectrum, while leaving the rest of the spectrum unaffected. Notch filters are extensively used in audio and sensor signal processing applications in order to minimize the effects of 50/60Hz powerline interference on measured signals.



A notch filter may be defined as:

$$H(z) = \frac{1 - 2\cos w_c z^{-1} + z^{-2}}{1 - 2r\cos w_c z^{-1} + r^2 z^{-2}}$$

where, $w_c = \frac{2\pi f_c}{fs}$ controls the centre frequency, f_c of the notch, and r controls the bandwidth of the notch. The symbolic expressions are implemented as follows:

```
ClearH1;  // clear primary filter from cascade
interface r = {0,1,0.1,0.5};  // radius range
interface fc = {0, fs/2,fs/100,fs/4}; // centre frequency range

Main()

wc=Twopi*fc/fs;

Num = {1,-2*cos(wc),1};      // define numerator coefficients
Den = {1,-2*r*cos(wc),r^2}; // define denominator coefficients
Gain = sum(Den)/sum(Num);    // normalise gain at DC
```

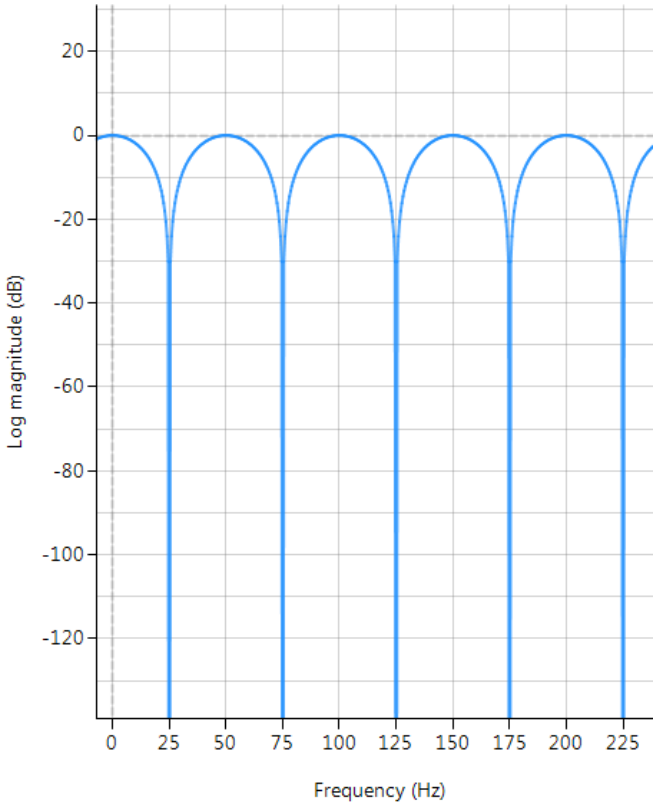
3.7. Comb (comb.afs)

The frequency response of a comb filter consists of a series of regularly-spaced troughs, giving the appearance of a comb. Where the spacing of each trough appears at either odd or even harmonics of the desired fundamental frequency. Thus, an FIR comb filter can be described by the following transfer function:

$$H(z) = 1 + \alpha z^{-L}$$

where, α is used to set the Q (bandwidth) of the notch and may be either positive or negative depending on what type of frequency response is required. In order to elaborate on this, negative values of α have their first trough at DC and their second trough at the fundamental frequency. Clearly this type of comb filter can be used to remove any DC components from a measured waveform if so required. All subsequent troughs appear at even harmonics up to and including the Nyquist frequency.

Positive values of α on the other hand, only have troughs at the fundamental and odd harmonic frequencies, and as such cannot be used to remove any DC components.




```
ClearH1;  // clear primary filter from cascade
interface L = {4,20,1,5}; // filter length
interface alpha = {-1,1,0.01,0.99};

Main()
Num = {1,zeros(L-1),alpha}; // numerator coefficients
Den = {1};
Gain = 1/sum(abs(Num));
```

3.8. Fractional Farrow Delay

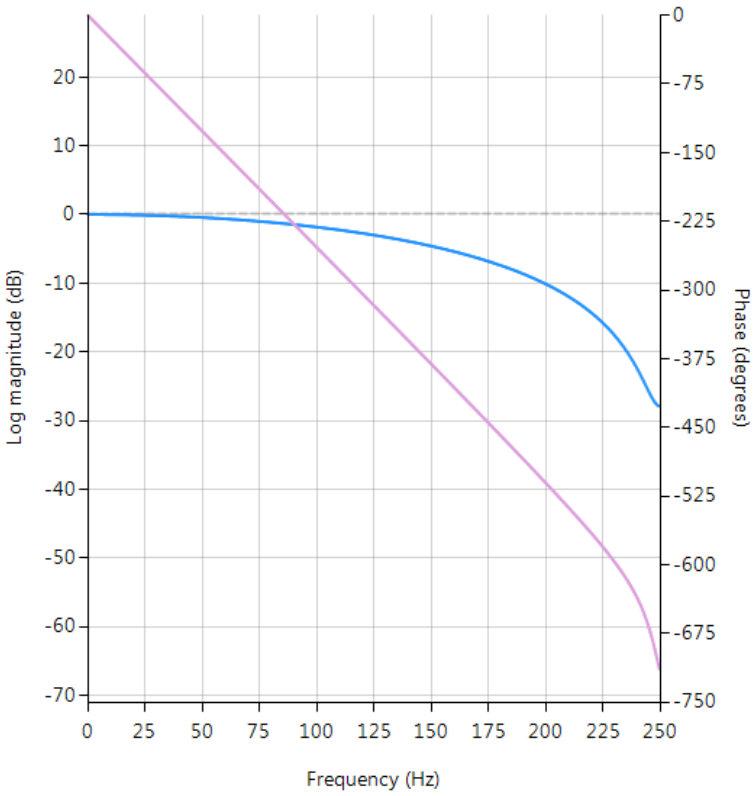
In signal processing, the need sometimes arises to nudge or fine-tune the sampling instants of a signal by a fraction of a sample. An FIR Farrow delay filter is typically employed to achieve this task, and may be combined with a traditional integer delay line in order to achieve a universal fractional length delay line.

A Fractional delay based on an FIR Farrow structure may be defined as:

$$H(z) = (1-\alpha) + \alpha z^{-1};\ \ 0 \leq \alpha \leq 1$$

Which produces a fractional linear delay of α between 0 and 1 samples. However, a more universal building block can be achieved by combining the Farrow delay structure with an integer delay, Δ

$$H(z) = (1-\alpha)z^{-\Delta} + \alpha z^{-(\Delta+1)}$$



The plot shown on the right shows the magnitude (blue) and phase (purple) spectra for $\Delta = 9, \alpha = 0.52$. As seen, the fractional delay element results in a non-flat magnitude spectrum at higher frequencies.

```
ClearH1;  // clear primary filter from cascade

interface alpha = {0,1,0.02,.5}; // fractional delay
interface D = {1,30,1,10};       // integer delay

Main()
Num = {zeros(D),1-alpha,alpha}; // numerator coefficients
Den = {1};                      // denominator coefficient
Gain = 1/sum(Num);              // normalise gain at DC
```

Document Revision Status

Rev.	Description	Date
1	Document updated for v4 and released.	26/09/2017
2	Updated document for v4.0.7 release.	24/02/2017
3	Updated document for v4.2 release.	08/01/2019
4	Updated document for v4.3 release.	10/07/2019