

ASN Filter Designer

FilterScript user's guide

April 2026

ASN21-DOC016, Rev. 12

For public release

Legal notices

All material presented in this document is protected by copyright under international copyright laws. Unless otherwise stated in this paragraph, no portion of this document may be distributed, stored in a retrieval system, or reproduced by any means, in any form without Advanced Solutions Nederland B.V. prior written consent, with the following exception: any person is hereby authorized to store documentation on a single computer for personal use only and to print copies for personal use provided that the documentation contains Advanced Solutions Nederland B.V. copyright notice.

No licenses, expressed or implied are granted with respect to any of the technology described in this document. Advanced Solutions Nederland B.V. retains all intellectual property rights (IPR) associated with the technology described within this document.

The information presented in this document is given in good faith and although every effort has been made to ensure that it is accurate, Advanced Solutions Nederland B.V. accepts no liability for any typographical errors.

In no event will Advanced Solutions Nederland B.V. be liable for any damages resulting from any defects or inaccuracies in this document, even if advised that such damages may occur.

Advanced Solutions Nederland B.V.

www.advsolned.com

support@advsolned.com

Copyright © 2026 Advanced Solutions Nederland B.V. All rights reserved.

Technical documentation feedback

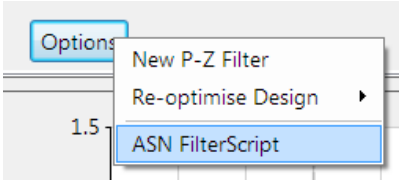
If you would like to make a suggestion or report an error in our documentation, please feel free to contact us. You are kindly requested to provide as much information as possible, including a full description of the error or suggestion, the page number and the document number/description. All suggestions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

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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 82 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

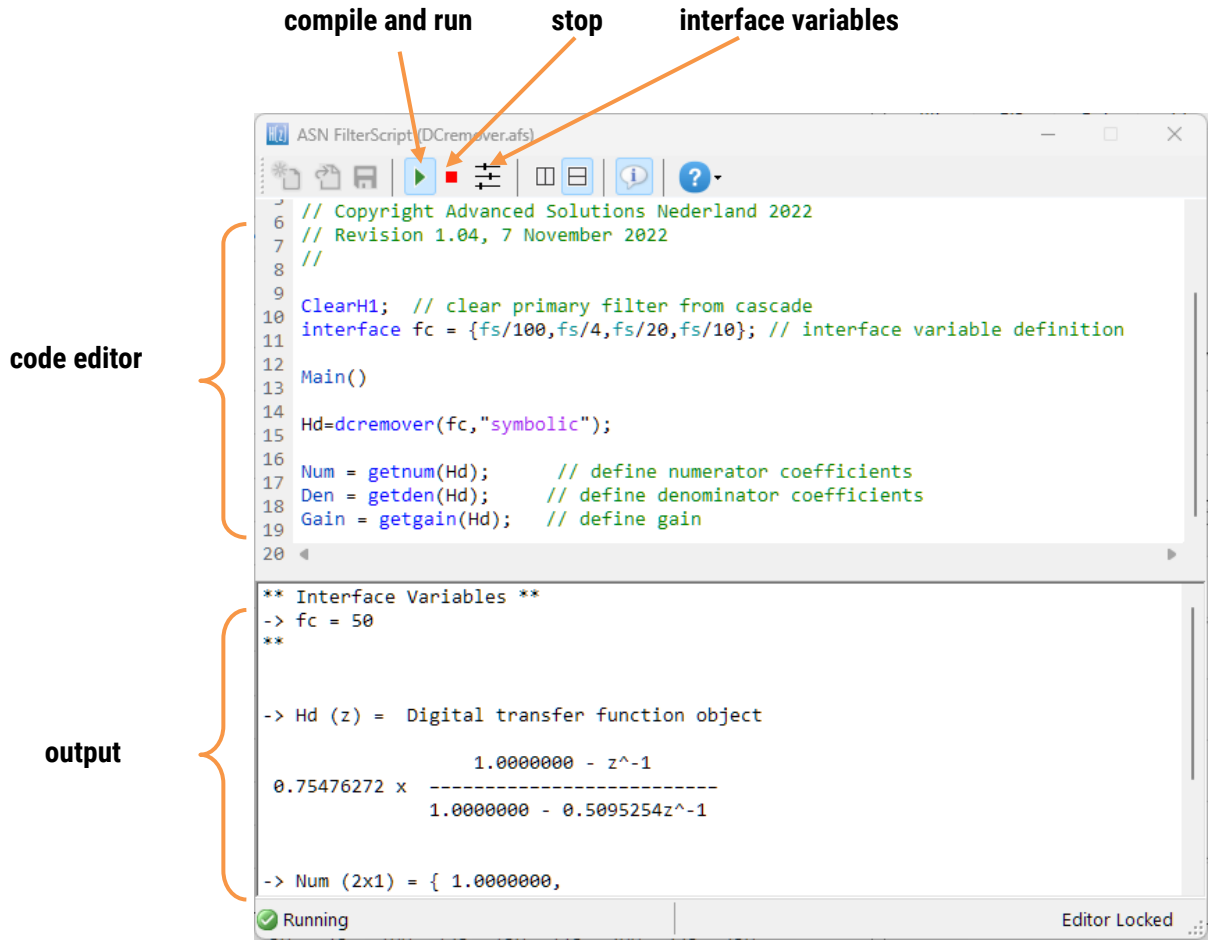


ASN FilterScript: learn the basics

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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:

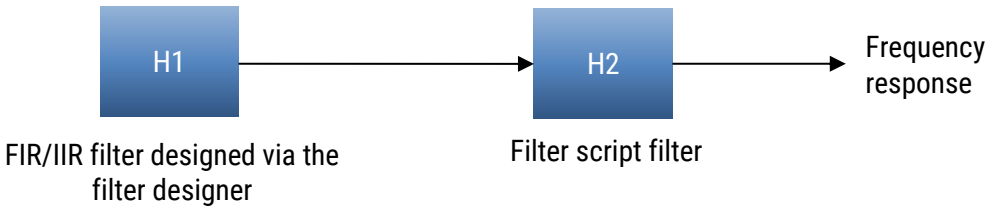
- *Initialisation*: contains all definitions of interface variables and any constant expressions. This section is run only once after compilation.
- *Main*: contains the bulk of the code, including the `Num`, `Den` and `Gain` expressions. Any expressions containing interface variables will be updated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

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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 to 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 2.11 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};
```

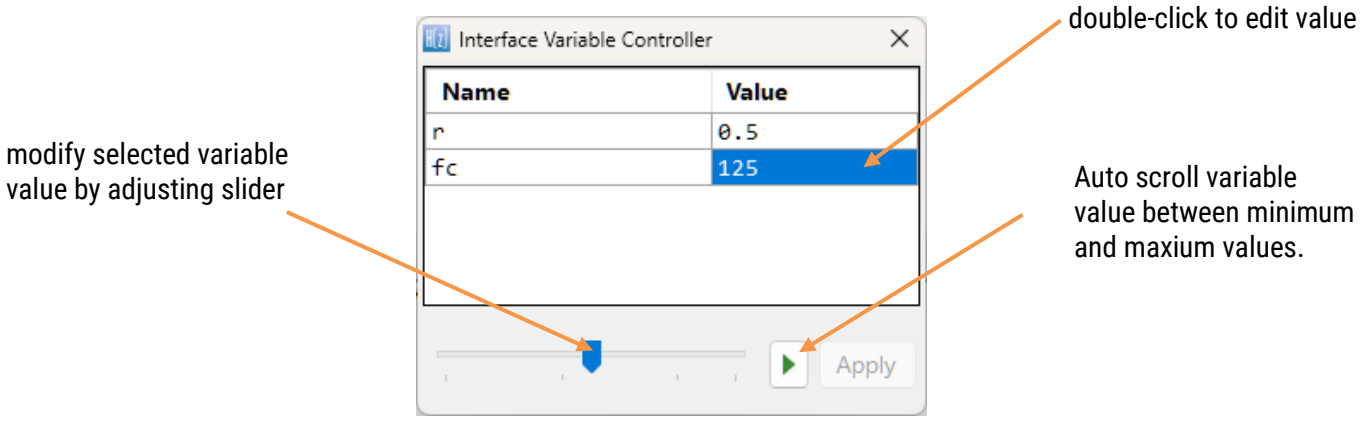
where, all entries must be real scalars values. Vectors and complex values will not compile.

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.

1.2.2. Programming the H1 filter via ASN FilterScript

Although FilterScript is primarily aimed at the H2 filter, designers may also program the H1 filter via the **H1Num**, **H1Den** and **H1Gain** commands. This provides a great deal of flexibility, as two independent filters can be customised and cascaded. See the biomedical example for more details.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2. The Scripting language

The scripting language supports over 78 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. 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.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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 -∞
fft	`y = fft(x)`	Two sided Fast Fourier transform.
ifft	`y = ifft(x)`	Inverse Fourier transform.
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 $0 \leq freq \leq \pm fs/2$ $roots = (z - r_1 e^{-i\theta_1}), (z - r_2 e^{-i\theta_2}), \ldots$ 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`

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.4. Vector functions

Function	Syntax	Description
augmentpoly	y = augmentpoly(a,R)	Creates a single polynomial from R identical polynomials. Where, $Y(z) = [A(z)]^R$ for $1 \leq R \leq 10$. You may use this function for implementing Kolmogorov–Zurbenko (KZ) or equivalent filters, as discussed in section 2.7.12. The function is also useful for improving the sideband attenuation characteristics of a general FIR filter of any type, e.g. lowpass, highpass, bandpass etc. Finally, the function will automatically analyse the generated coefficients and remove any leading or trailing zeros.
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 2.6
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
winfunc	y = winfunc(L,WinType, Beta)	Returns L weighting coefficients for the Window method specified by WinType. The following window functions are supported: rectangular, blackman, blackmanharris, hamming, hanning, flattop, kaiser and Chebyshev. Beta is only required for Chebyshev and kaiser and should be 0 for all other functions.
zeros	y = zeros(N)	Vector of 0s of length N

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.5. Analog and digital filter objects

Many of the design methods make use of so-called 'filter objects', which are simply a struct of the filter's numerator, denominator and gain parameters and type, i.e. analog or digital. Using objects provides an easy way of augmenting multiple filters, computing gains and assigning a reference spectrum. The latter is particularly useful for comparing the results of two filters, and seeing the difference between an original analog spectrum and the resulting digital filter spectrum when transformed using the Bilinear or match z-transform method.

The following commands should be used when working with filter objects.

2.5.1. Analog

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.
`refspectrum`	`y=refspectrum(Ha)`	Sets the reference magnitude spectrum to Ha.

2.5.2. Digital

Function	Syntax	Description
`augment`	`Hd=augment(Hd1,Hd2)`	Augment or merge two digital filter objects, and return the merged objects as a single object in `Ha`.
`computegain`	`G=computegain(Hd,Fo)`	Get the gain, `G` of the digital filter (Hd) at frequency point, `Fo`. Where, `Fo` is specified in Hz.
`getnum`	`Num=getnum(Hd)`	Get the numerator coefficients of digital filter object, Hd.
`getden`	`Den=getden(Hd)`	Get the denominator coefficients of digital filter object, Hd.
`getgain`	`G=getgain(Hd)`	Get the gain of digital filter object, Hd.
`refspectrum`	`y=refspectrum(Hd)`	Sets the reference magnitude spectrum to Hd.

The `refspectrum()` function is used to set the H2 design markers to the magnitude spectrum of the given filter object. FilterScript will acknowledge this by returning a `reference spectrum...OK` message via the variable `y`. The return variable `y` has no further significance, and can be reused for other operations. The filter object parameters will be evaluated based on the current frequency axis, such that it will always match the displayed filter's magnitude spectrum.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6. IIR filter design

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

2.6.1. ellip

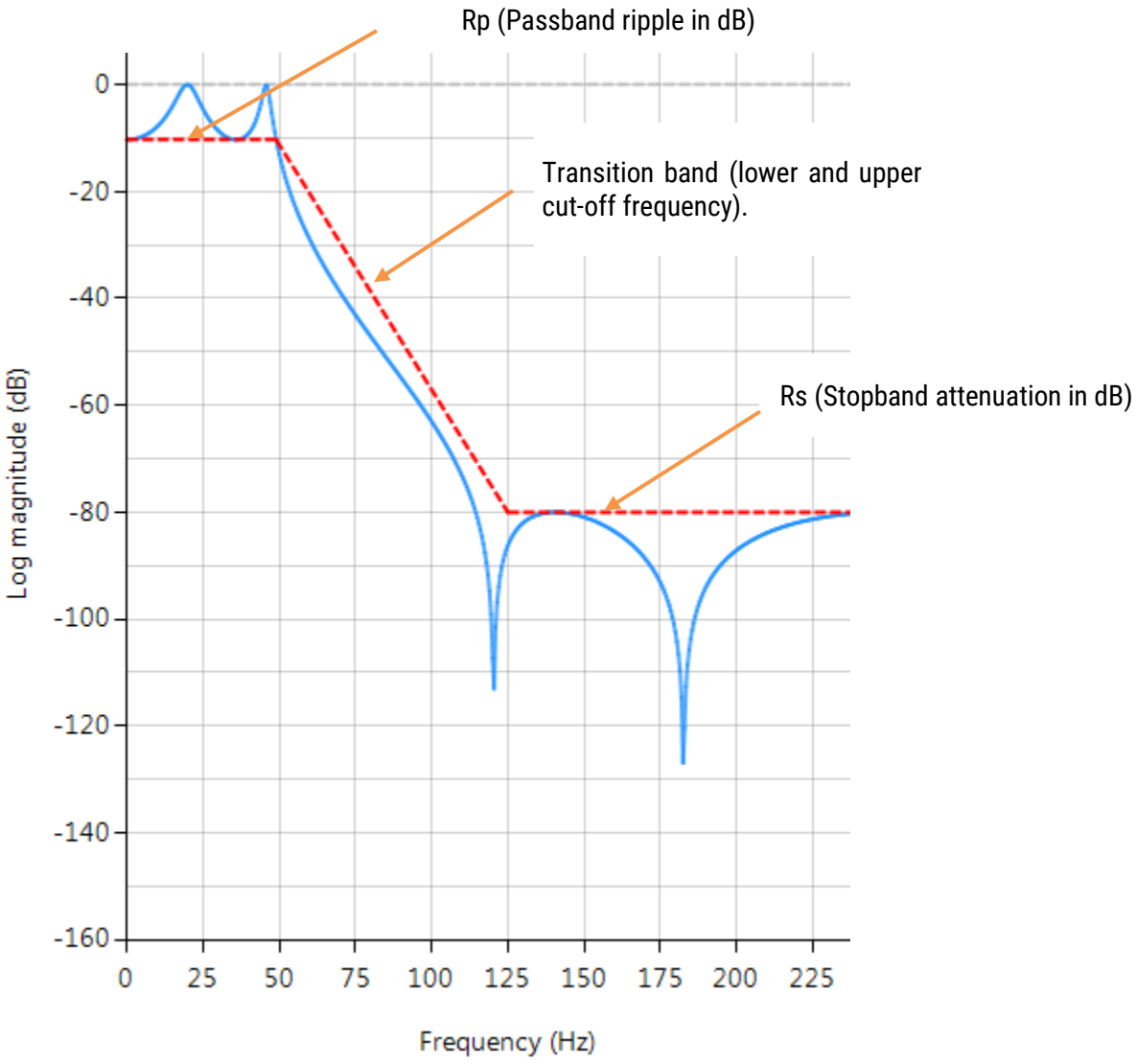
Syntax

```
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.



Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

`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 as such requires 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 the 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=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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.2. butter

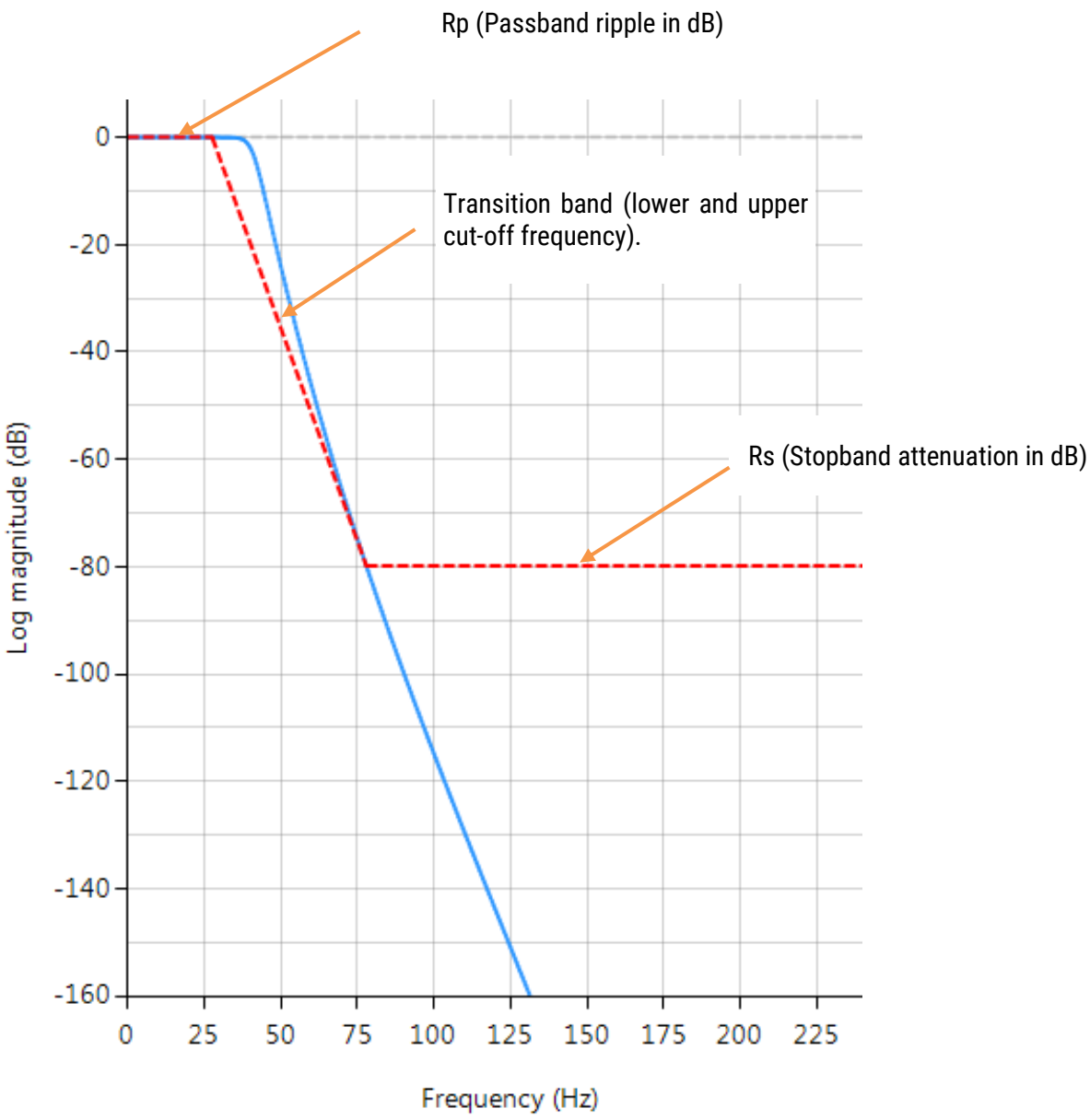
Syntax

```
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.



Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

`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 as such requires 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 the 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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.3. cheby1

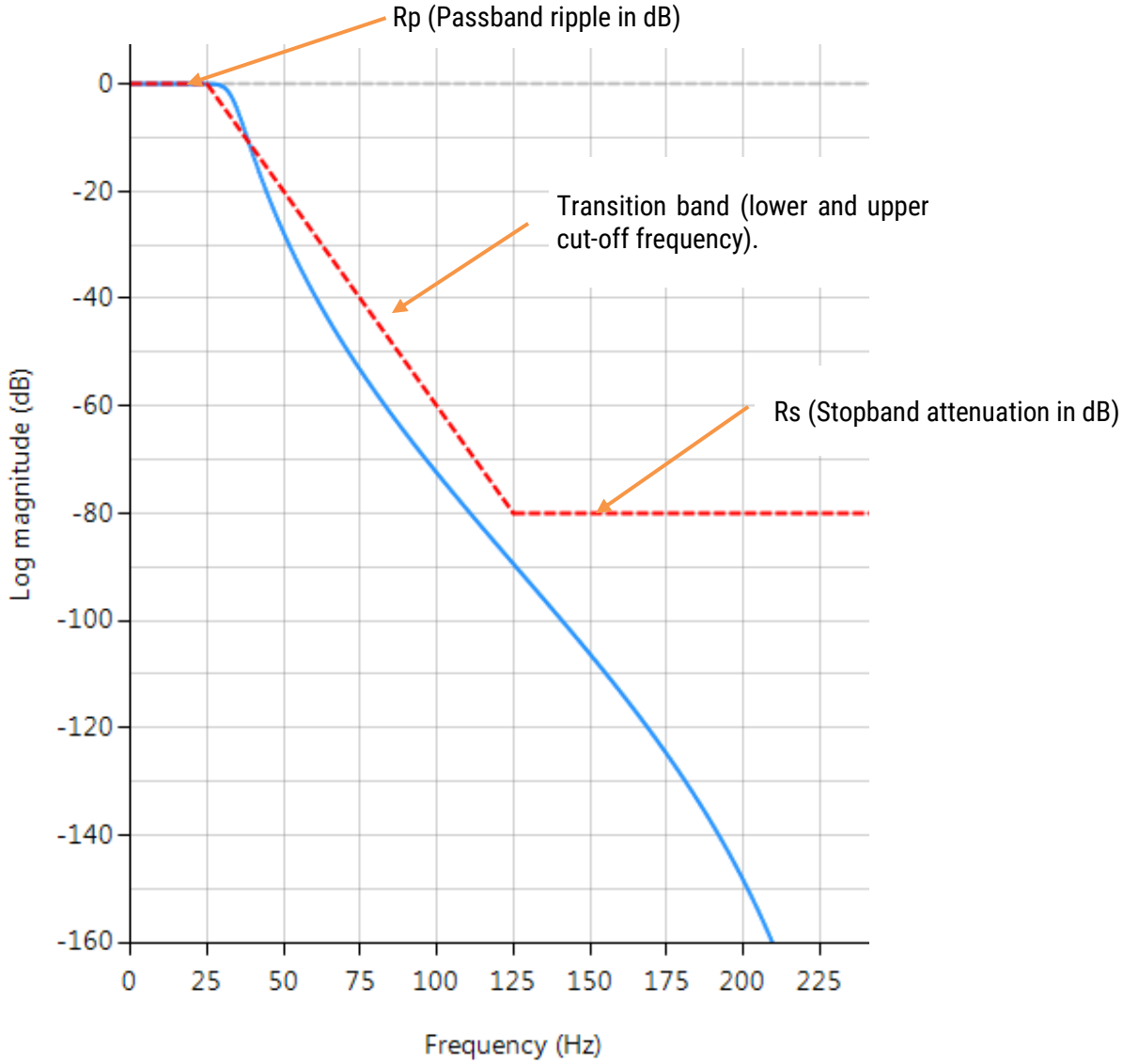
Syntax

```
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.



Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

`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 as such requires 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 the 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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.4. cheby2

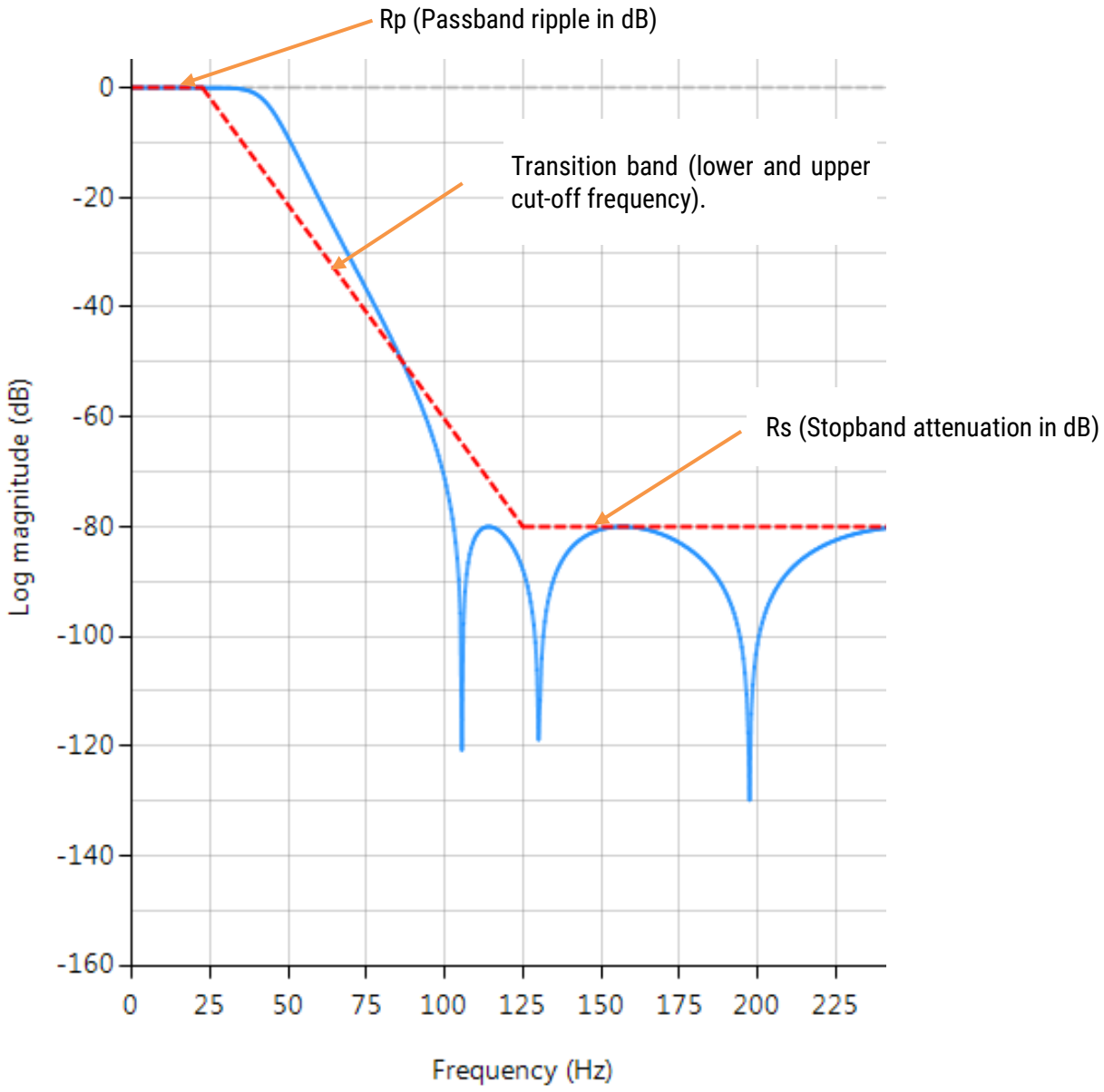
Syntax

```
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.



Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

`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 as such requires 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 the 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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.5. bessel

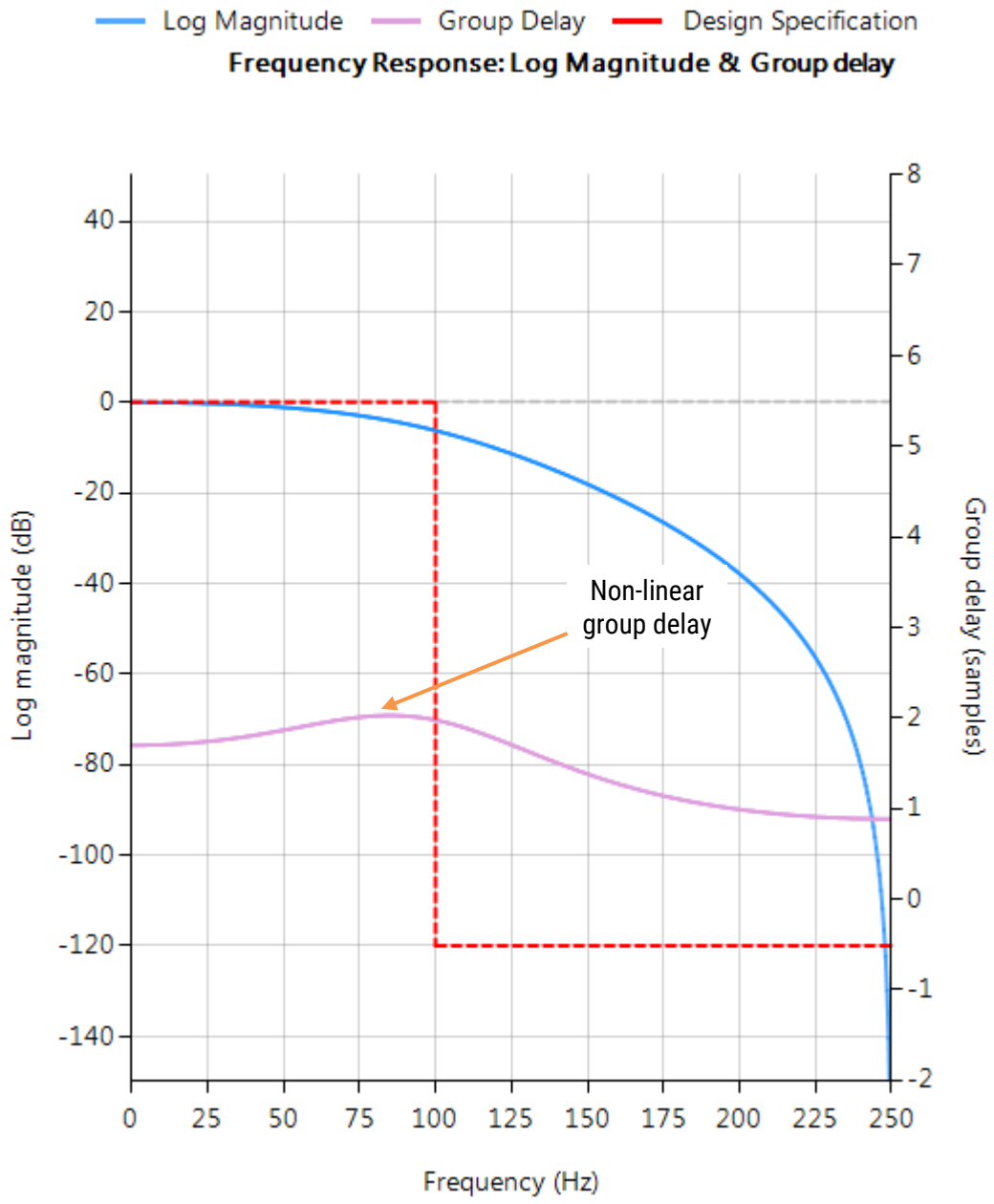
Syntax

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

Description

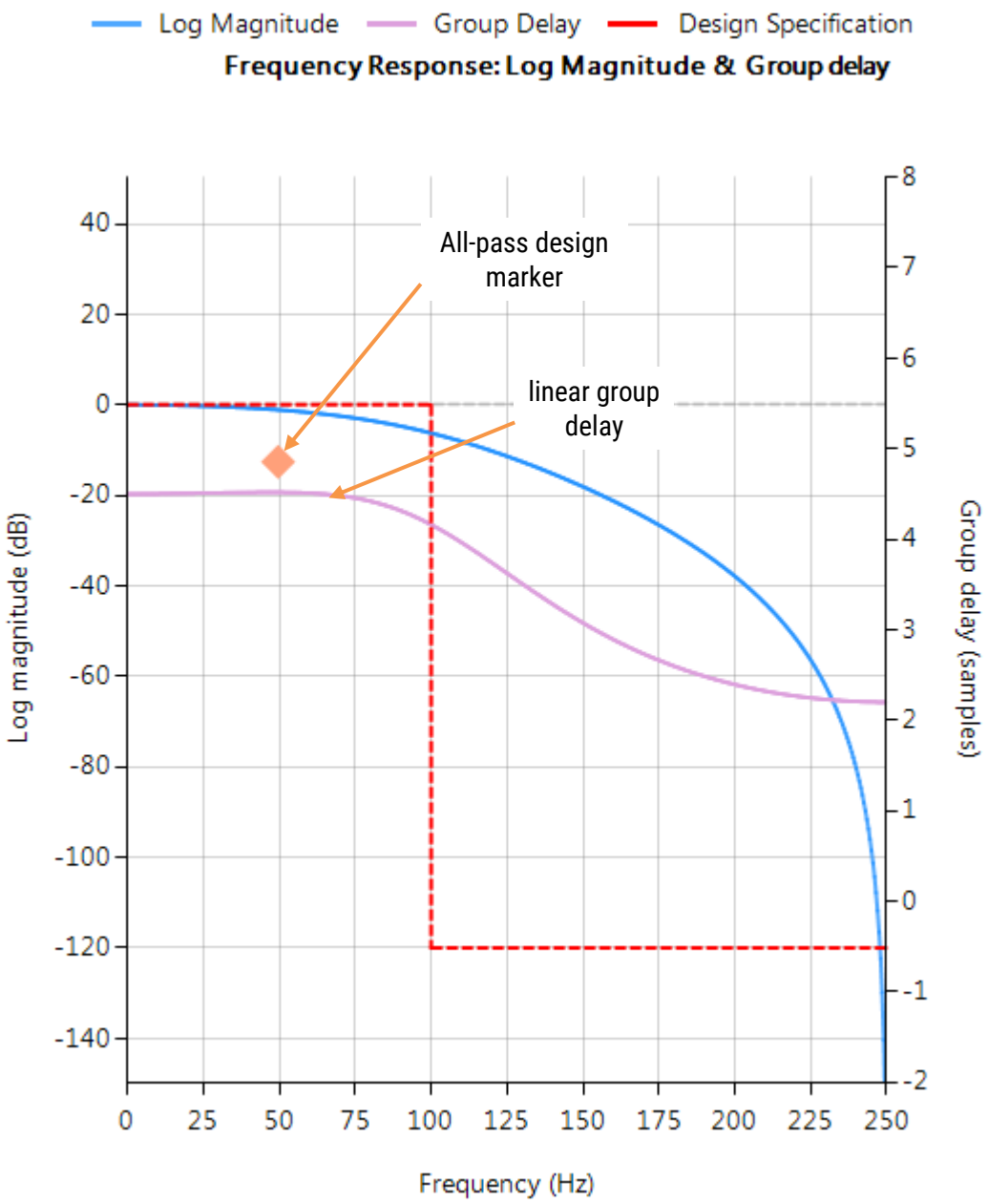
IIR Bessel filter design. Analog Bessel filters have a constant group delay in the passband, which is very desirable for a variety of measurement applications. The method implemented within FilterScript uses the Bilinear transform which modifies the standard analog Bessel characteristic, and as a consequence 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.
- Does not match amplitude cut-off specifications



Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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



This method does not match amplitude cut-off specifications (i.e. -3dB points), but attempts to equalise the group delay in the passband instead.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

`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 the 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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.6. 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, $BW \leq \frac{F_o}{2}$

`DFormat`: allows you to specify the display format of the 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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.7. 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 the 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
```



Use the zero-phase filtering function in the main UI to set the IIR filter's phase to zero.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.8. arbmagphase

Syntax

```
Hd = arbmagphase(Nb,Na,mag,phase,freq,Type,DFormat)
```

Description

Find the poles and zeros of an unknown filter from its complex frequency response specification, $H(k)$. Where, $mag = 20log_{10}(|H(k)|)$, $phase = \frac{180}{\pi} arg\big(H(k)\big)$ and freq = 0 – Nyquist in Hz - data may have gaps, and is not required to have equally spaced incremental frequency steps between 0 and Nyquist. Use the `fsunits` keyword to scale a normalised frequency specification into Hz.

`Nb`: The number of zeros (`0-20`) to be fitted to the data. If you want to create all-zero model, set `Na=0`
`Na`: The number of poles (`0-20`) to be fitted to the data. If you want to create all-pole model, set `Nb=0`
`Mag, phase` and `freq`: the complex frequency response data in *dB, degrees and Hz*.
`Type: real` produces a real filter, whereas `complex` produces a complex filter.

`DFormat`: allows you to specify the display format of the 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.

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

interface Na = {0,10,1,5}; // interface variable definition
interface Nb = {0,20,1,6}; // interface variable definition

H=importdata("Datafiles\mysteryfilter.txt"); // load specification (Fs=500Hz)

freq=H(:,0); // extract frequency specification (Hz)
mag=H(:,1);  // extract magnitude specification (dB)
phase=H(:,2); // extract phase specification (degrees)

Main()

// design a "real" filter from the specification
Hd=arbmagphase(Nb,Na,mag,phase,freq,"real","symbolic");

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


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.9. Peaking

Syntax

```
Hd = peaking(Fo,BW,K,DFormat)
```

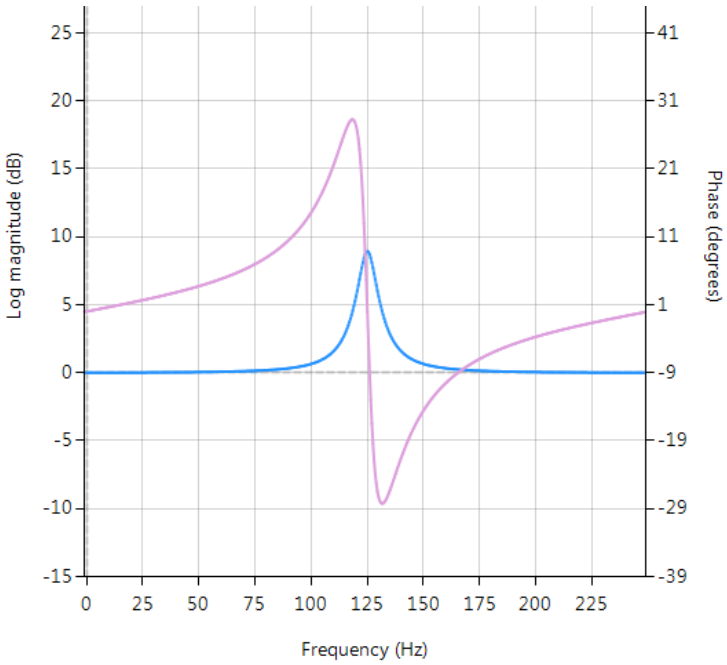
Description

Designs a 2nd order audio peaking/Bell filter. All frequencies are in Hz.

- `K` is amplitude of peak
- `BW` is bandwidth when `K=0`, and `0 < Fo < Fs/2`

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:

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

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.6.9.1. 3-band equaliser example

The following script implements a 3-band equaliser, comprised of a highpass filter and two peaking filters.

```
/// Description: This script implements a simple 3-band audio equaliser
// Author: Advanced Solutions Nederland BV
// Date: Monday, 29 June 2020
//
// Tested with fs=44.1kHz
//
ClearH1;  // clear primary filter from cascade

//band 1
interface f={10,1000,10,500}; // highpass filter

// band 2
interface BW = {1,2000,10,1500}; // filter bandwidth
interface fc = {0, fs/2,fs/100,fs/8}; // peak/notch centre frequency
interface K = {0,3,0.1,2}; // gain/sign

//band 3
interface BW2 = {1,2000,10,1500}; // filter bandwidth
interface fc2 = {0, fs/2,fs/100,fs/4}; // peak/notch centre frequency
interface K2 = {0,3,0.1,0.25}; // gain/sign


Main()

// design filters
hpfband={1,f};
Hdhpf=butter(2,hpfband,3,80,"highpass","symbolic");

Hd1=peaking(fc,BW,K,"symbolic");
Hd2=peaking(fc2,BW2,K2,"symbolic");

// merge filters
Hd=augment(Hdhpf,Hd1,"void");
Hd=augment(Hd,Hd2,"symbolic");

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


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7. FIR design methods

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

2.7.1. 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 the 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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.2. fircombparams

Syntax

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

Description

FIR comb filter parameter design. Returns the required sampling frequency and filter order needed to satisfy 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.

2.7.3. fircombfilter

Syntax

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

Description

FIR comb filter design.

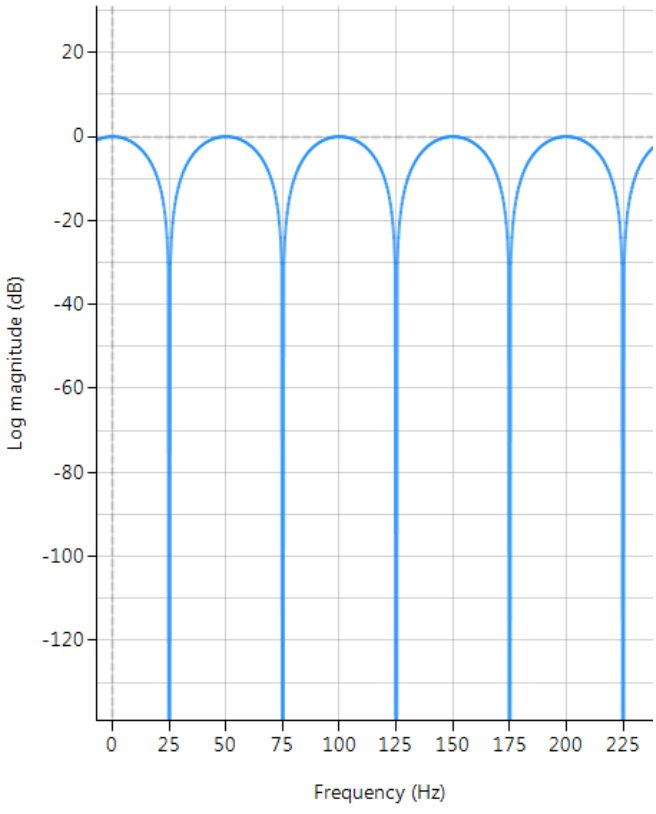
`Order`: may be specified up to `499` (professional) and up to `128` (educational) edition.

`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.



Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

`DFormat`: allows you to specify the display format of the 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);
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.4. 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`, `kaiser`, `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 the 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);
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.5. 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`, `kaiser` 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 the 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()

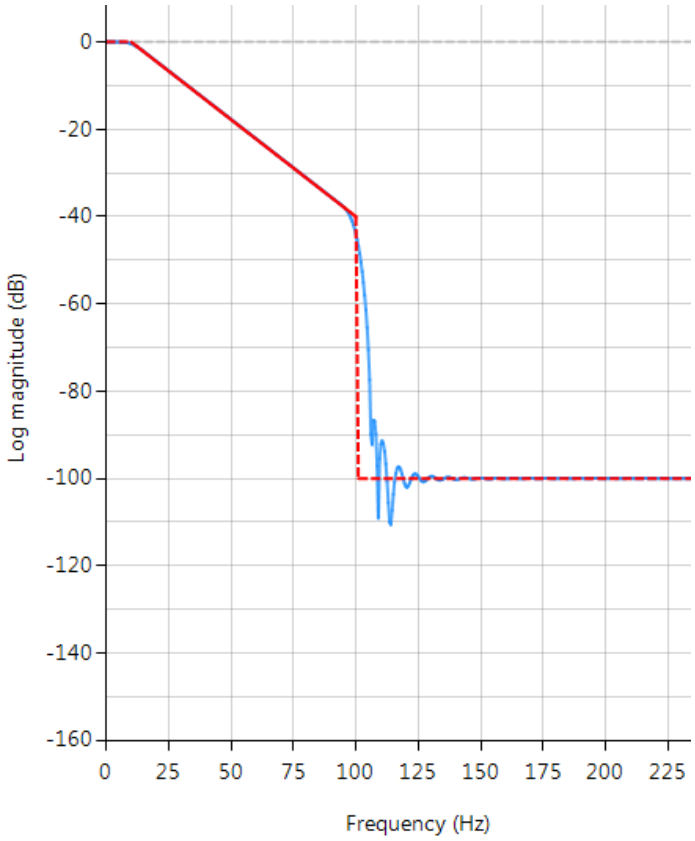
A=series(0,-1,-40);
F=series(10,90/(length(A)-1),100);
A={0,A,-100,-100};  //  specify arb response
F={0,F,101,fs/2};   //

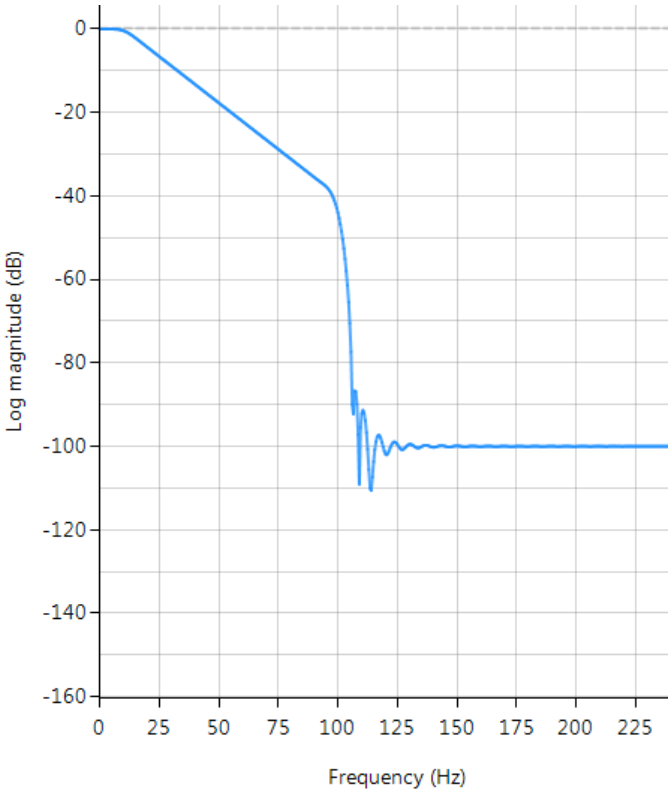
Hd=firarb(L,A,F,"hanning","numeric");

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


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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.





Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.6. 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 in Hz, in ascending 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 the 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;

// Fs=500Hz
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);
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.7. 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 the 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);
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.8. firminphase

Syntax

```
Hd = firminphase(Hdlp, DFormat)
```

Description

Designs an FIR filter with minimum phase.

Minimum phase filters (also referred to as **zero-latency filters**) overcome the inherent N/2 latency (group delay) in a linear phase FIR filter, by moving any zeros outside of the unit circle to their conjugate reciprocal locations inside the unit circle. The result of this 'zero flipping operation' is that the magnitude spectrum will be identical to the original filter, and the phase will be nonlinear, but most importantly the latency will be reduced from N/2 to something much smaller (although non-constant), making it suitable for real-time control applications where IIR filters are typically employed.

For applications where phase is less important, this may sound ideal, but the difficulty arises in the numerical accuracy of the root-finding algorithm when dealing with large polynomials. Therefore, orders of about 80 should be considered a maximum when using this approach.

`Hd`: the `firminphase` method designs an FIR minimum phase by transforming a linear phase FIR digital filter object, `Hdlp`. The resulting filter is then placed in 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 the 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

```
/// FIR minimum phase filter design.

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() // main loop

Freq={F+TW};   // frequency specification
Hd=firwin(50,Freq,"hanning","highpass","numeric");  // design linear phase FIR
Hd=firminphase(Hd,"numeric"); // transform FIR to minimum phase (zero latency)

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


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.9. fir2lp2notch

Syntax

```
Hd = firlp2notch(Hd,DFormat)
```

Description

Designs an FIR notch filter from a lowpass filter by computing the difference between the prototype lowpass filter and its amplitude complementary, i.e. H_{BPF} (z) $= 2H_{LPF}(z) - 1$

The resulting bandpass filter has linear phase (odd length), but a 180-degree phase shift at the notch frequency. This means that all frequencies > notch frequency will be shifted by 180 degrees. For biomedical applications, this excess phase shift is undesirable and may be overcome by cascading two identical filters, such that the net phase shift is 360 degrees. The `augument()` function should be used for achieving this.

`Hd`: the `firlp2notch` 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 the 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

```
interface F = {10,400,2,44}; // frequency spec (2 bands for bandstop)
interface TW ={1,40,2,12}; // Band transition width
interface N = {20,300,1,36};


Main() // main loop

Freq={F,F+TW};   // frequency specification

Hd=firwin(N,Freq,"hanning","lowpass","numeric");

Hd=firlp2notch(Hd,"void");
Hd=augment(Hd,Hd,"void"); // compsensate for 180 phase shift by cascading a second
filter

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


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.10. ciccomp

Syntax

```
Hd = ciccomp(L,N,R,D,Fc,DFormat)
```

Description

CIC (Cascaded Integrator-Comb) filters are commonly used in sigma-delta ADCs for efficient decimation. While CIC filters are simple and hardware-efficient, they introduce significant passband droop due to their sinc-shaped frequency response. This droop must be corrected for high-accuracy applications, especially in biomedical signal processing such as ECG, where a flat passband is crucial.

A CIC filter is characterized by three parameters: `N` (Number of stages), `R` (Decimation rate) and `D` (Differential delay).

To compensate for the CIC droop, a FIR filter can be designed that approximates the inverse response of the CIC. ASN FilterScript's `ciccomp()` function automates this process, designing a linear-phase FIR filter of order `L` that restores a flat passband. The `Fc` parameter is used to set the compensation filter's cut-off frequency.

`DFormat`: allows you to specify the display format of the 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 = {32,64,2,32};   // filter order
interface Fc ={50,80,1,60};

Main() // main loop

// design CIC compensation filter N=3, R=128, D=1
Hdcic=ciccomp(L,3,128,1,Fc,"numeric");

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


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.11. blwtracker

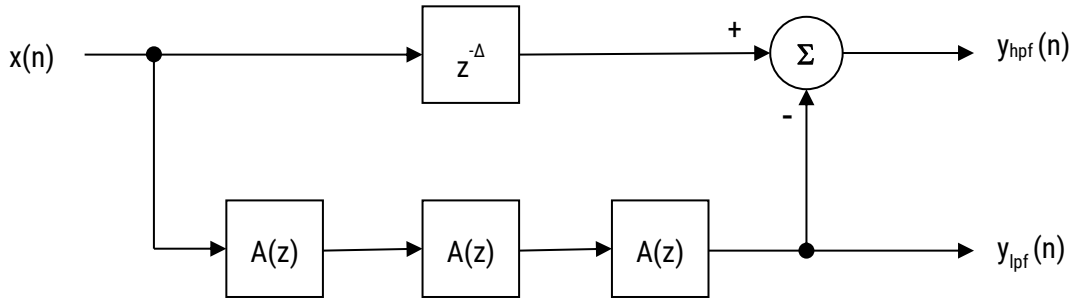
Syntax

```
Hd = blwtracker(Fc,R,Type)
```

Description

Designs a lowpass or highpass linear phase optimised FIR filter using a Kolmogorov–Zurbenko (KZ) filtering cascade (see section 2.7.12 for more information). These filters are ideal for designing a variety of low frequency filters, such as biomedical BLW removal highpass filters for ECG and predictive maintenance applications, and lowpass DC tracking filters for loadcell applications.

A KZ cascade can be designed by cascading R identical MA (moving average) filters of length L into a single polynomial. A 3-section (R=3) Kolmogorov–Zurbenko cascade is shown below:



As seen, a highpass filter is realised by delaying the input, x(n) by the KZ cascade group delay, Δ and then subtracting the delayed input from the lowpass filtered output. An important point to realise is that the KZ cascade is extremely computationally efficient, i.e. each MA filter only requires one addition and one subtraction, rather than multiple MAC operations typically required with a traditional FIR filter.

`R` is the number of sections (3 recommended), and the cut-off frequency is given by `Fc`.

`Type`: selects the mode of operation

`Removal`	Highpass filtering
`Tracking`	Lowpass filtering

NB. Calling `blwtracker()` will automatically update the `Hblw` filtering block, and calling the optional `ClearHblw` keyword will disable the `Hblw` filtering block.

```
interface fc = {0.2,1,.1,0.67}; //hpf cut-off
interface R = {3,5,1,3}; // number of KZ sections

Main()      // main loop

Hblw=blwtracker(fc,R,"removal"); // design highpass using a KZ cascade

// don't use the H2 filter
Num=1;
Den=1;
Gain=1;
```

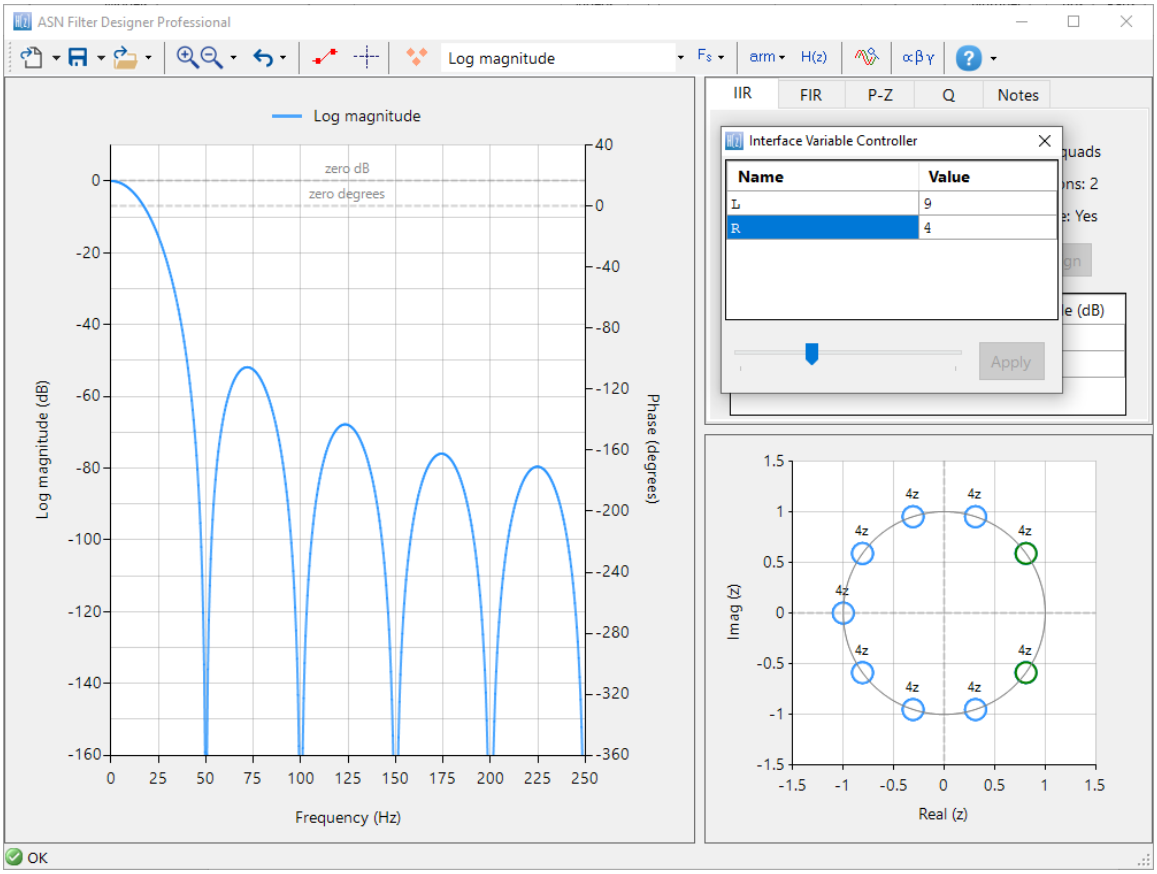

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.12. More about Kolmogorov–Zurbenko (KZ) filters

As discussed in section 2.7.1, the MA filter has poor stopband attenuation characteristics, which makes it a poor low pass filter. However, a simple method of improving the stopband characteristics is via a so called Kolmogorov–Zurbenko (KZ) filter, that in essence cascades R identical MA filters of length L into a single polynomial.

$$H(z) = [A(z)]^R = \left[\frac{1 + z^{-1} + z^{-2} + z^{-3} \ldots\ldots + z^{-(L-1)}}{L}\right]^R$$

As seen below (see here for a video), the result of the cascade significantly improves the stopband characteristics (even for low values of R), while retaining the highly desirable sharp step response of the MA.



FilterScript provides developers with a alternative method of designing KZ filters via the `augmentpoly()` function.

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

interface L = {1,100,2,77};   // model length (order = length - 1)
interface R = {1,10,1,6};    // Number of filters

Main()

Hd=movaver(L,"numeric"); // design moving average
Num = getnum(Hd); // moving average filter coefficients
Den = 1;
Gain = getgain(Hd);      // normalise gain at DC

// Kolmogorov-Zurbenko filter coefficients
Num=augmentpoly(Num*Gain,R);
Gain=1;
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.12.1. Biomedical filtering application

This example demonstrates correct usage of the `H1Num`, `H1Den` and `H1Gain` keywords, allowing designers to program the H1 filter directly.

The `firlp2notch` function can be used in conjunction with `dcremover` function in order to build an ECG biomedical filtering cascade with linear phase around the notch cut-off frequency. Notice that the two filters are split up into the H1 and H2 filters respectively, and may therefore be optimised independently to each other.

```
/// Biomedical HPF + FIR notch filter
// This notch filter removes 50/60Hz power line interference while
// maintaining linear phase around the notch cut-off frequency. The DC offset
// and low frequencies are removed via the H1 IIR HPF.
//
// Copyright Advanced Solutions Nederland 2021
// Revision: 1.00


interface F = {10,400,2,44}; // frequency spec (2 bands for bandstop)
interface TW = {1,40,2,12};  // Band transition width
interface R = {10,80,5,40};  // Band transition width
interface N = {20,300,1,36};


Main() // main loop

Freq={F,F+TW};   // frequency specification

//Hd=firkaiser(Freq*fsunits,R,"lowpass","numeric");
Hd=firwin(N,Freq,"hanning","lowpass","numeric");

Hd=firlp2notch(Hd,"void");
Hd=augment(Hd,Hd,"void"); // compensate for 180 phase shift by cascading a second
filter

Hd1=dcremover(2,"void");

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

H1Num=getnum(Hd1);
H1Den=getden(Hd1);
H1Gain=getgain(Hd1);
```

Designers are also referred to the following application note that discusses the design and implementation of the Pan-Tompkins filter cascade for QRS complex detection in ECG signals.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.13. 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). In general, Savitzky-Golay filters should be used for data requiring peak enhancement, such as biomedical ECG and PPG applications.

`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 the 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);
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.14. savgolaydiff

Syntax

```
Hd = savgolaydiff(Order, Polyfit, Derv, DFormat)
```

Description

Design an FIR Savitzky-Golay robust differentiation filter of length, `Order+1`, Derivative, `Derv` and polynomial fit, `Polyfit`. These types of filters are good for feature extraction (e.g. biomedical data), as they combine differentiation with lowpass filtering.

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

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

`Derv: 1 < Derv < PolyFit`

`Hd`: the `savgolaydiff` method designs an FIR Savitzky-Golay robust differentiation 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 the 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 M = {2, 50,2,24};
interface P = {2, 10,1,4};
interface D = {1, 10,1,2};

Main() // main loop

Hd=savgolaydiff(M,P,D,"numeric");  // Design Savitzky-Golay Differentiator filter

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


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.7.15. 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 the 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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.8. Analog → digital filter design: Laplace transforms

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

2.8.1. analogtf

Syntax

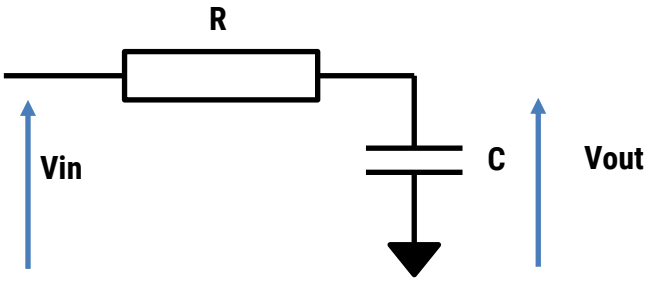
```
Ha=analogtf(ANum, ADen, AGain, DFormat)
```

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 transformation or Matched-z method methods. See the example script `AnalogRCfilter.afs` for more information.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.8.2. bilinear

Syntax

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

Description

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

The bilinear transformation, 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 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. If no pre-warping is required, set `Fp=0`.

`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 the 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.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

Example

```
ClearH1;  // clear primary filter from cascade
ShowH2DM;   // show analog magnitude spectrum on chart

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 44 for a description of the analog RC filter model used in this example.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.8.3. mztrans

Syntax

`Hd=mztrans(Ha,DFormat)`

Description

Convert an analog filter object to its digital equivalent using the Matched-z method (S→Z transformation).

The matched-z method **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 the 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 analog magnitude spectrum on chart

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 44 for a description of the analog RC filter model used in this example.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.8.4. 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.
`refspectrum`	`y=refspectrum(Ha)`	Sets the reference magnitude spectrum to Ha.

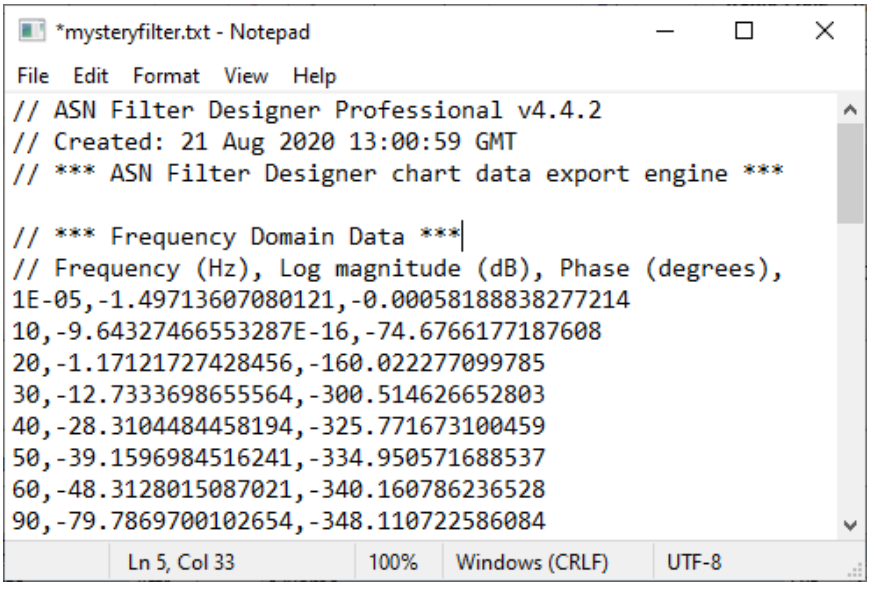
2.9. Loading an external coefficient datafile

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\mysteryfilter.txt");
```



- All data is read on a line by line basis. This allows for multiple entries per line that are delimitated by a comma.
- Complex data may be imported by using the `i` or `j` keyword.
- A maximum of 512 values may be imported.
- Comments may be placed anywhere, and must be preceded with the `//` keyword.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

Example (TFestimate.afs)

The following example loads an external datafile with three columns pertaining to an unknown filter's frequency response data, i.e. Frequency, magnitude and phase. The data are placed into a variable `H` for manipulation.

```
/// Example of finding the poles and zeros of an unknown filter
/// from its frequency response specification
//
// Rev: 1.01
// Advanced Solutions Nederland BV, 2022
//

ClearH1;  // clear primary filter from cascade
ShowH2DM;

interface Na = {0,10,1,3}; // interface variable definition
interface Nb = {0,20,1,4}; // interface variable definition

H=importdata("Datafiles\mysteryfilter.txt"); // load specification (Fs=500Hz)

freq=H(:,0); // extract frequency specification
mag=H(:,1);  // extract magnitude specification
phase=H(:,2); // extract phase specification

Main()

// design a 'real' filter from the specification based on 'Na' and 'Nb' estimates
Hd=arbmagphase(Nb,Na,mag,phase,freq,"real","symbolic");

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

Use the interface variable controller UI to modify `Na` and `Nb` in order to estimate mystery filter's poles and zeros.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.10. 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-1`, 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, you may assign the new value directly, e.g. `a(2,1)=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-1`, 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, you assign a double or complex value to the element.
	
	Example
	
	`a = {1,0,3.4,0,1};` // assign five elements to vector `a`
	`a(2,0)=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. Also, the `///` keyword will make the comment text bold.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

2.11. 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`
`eps`	`1e-12`
`i`	Complex number token, $\sqrt{-1}$

2.12. Mandatory keywords

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

Variable	Description
`Main()`	`Main()` is used to separate the initialisation code from the "main" code - see section 1.1 for more information.
`Den`	specifies the H2 denominator filter coefficients. This must be a vector or a scalar.
`Num`	specifies the H2 denominator filter coefficients. This must be a vector or a scalar.
`Gain`	specifies the H2 filter gain. This must be real.

2.13. Optional Keywords

Variable	Description
`ClearH1`	The `ClearH1` keyword will delete the H1 filter from the cascade.
`ClearHblw`	The `ClearHblw` keyword will delete the Hblw filter from the cascade.
`ShowH2DM`	Show the design markers or reference spectrum used for the design.
`SkipSC`	Skip pole stability check. This is useful when working on oscillators and integrators.
`H1Den`	specifies the H1 denominator filter coefficients. This must be a vector or a scalar.
`H1Num`	specifies the H1 denominator filter coefficients. This must be a vector or a scalar.
`H1Gain`	specifies the H1 filter gain. This must be real.

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

3. General example scripts

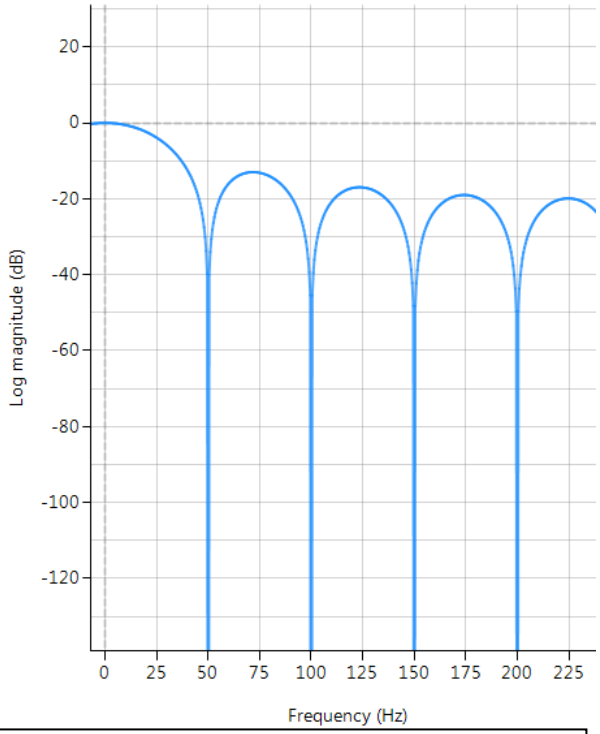
The following section is a collection of example scripts bundled with the software. All script files are `.afs` files which can be found in the `Scripts\Examples` directory. Please see our detailed online resource portal for more examples.

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()

Hd=movaver(L,"numeric"); // design moving average
Num = getnum(Hd);    // moving average filter coefficients
Den = 1;
Gain = getgain(Hd);      // normalise gain at DC
```

A more detailed article is available on the ASN blog.

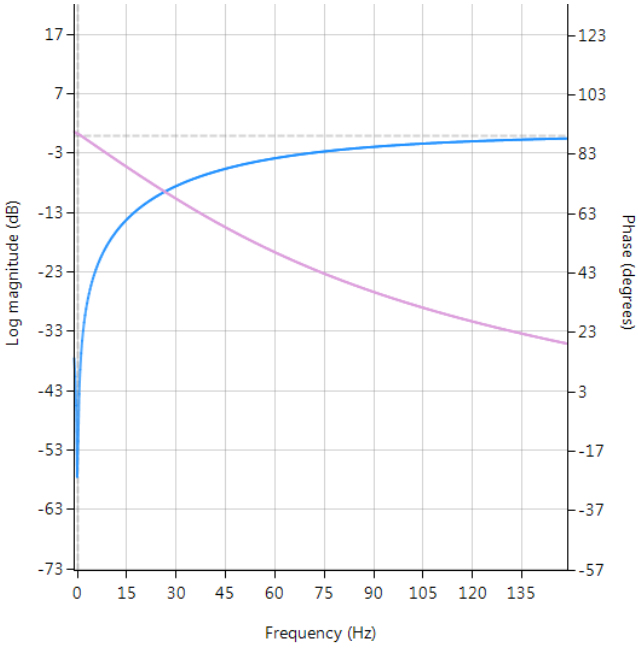
Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

3.2. HPF (BilinearHPF.afs)

It is sometimes useful to transform an analog filter into its digital/discrete equivalent. Although there are several transformation methods, the bilinear transformation is a very popular method and is therefore used for this example. Central to the bilinear transformation concept is the S-Z transformation which maps an analog 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 bilinear transformation 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);

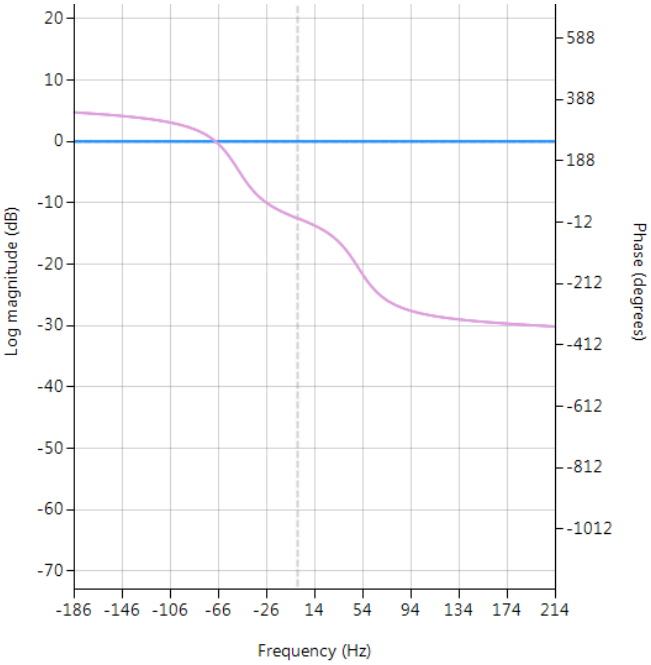
Num = {1,-1}; // define numerator coefficients
Den = {1,(w-1)/(w+1)}; // define denominator coefficients
Gain = 1/(w+1); // define gain
```

This method is implemented in the function dcremover().

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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

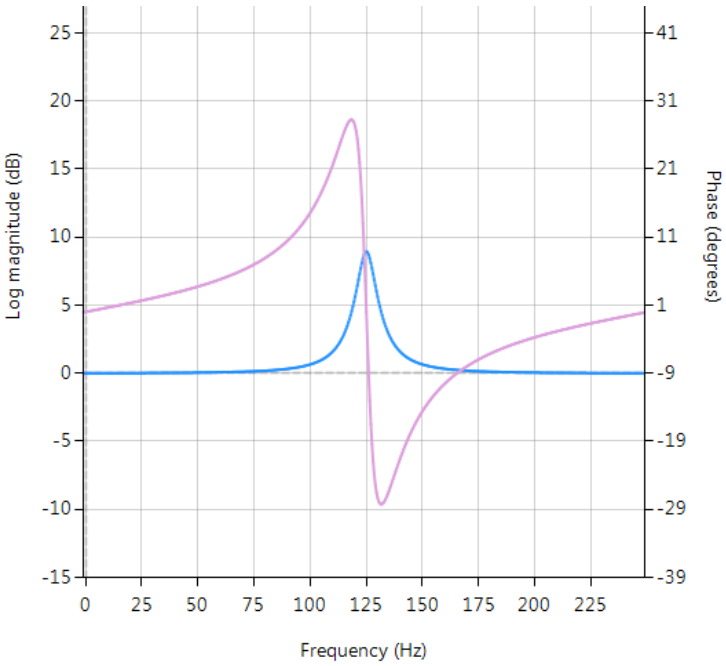
interface radius = {0,2,0.01,0.5};   // radius value
interface fc = {0,fs/2,1,fs/10};     // frequency value

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


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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:

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

See also the command peaking() for more information.

```
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;
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

3.5. AllpassNotch (AllpassNotch.afs)

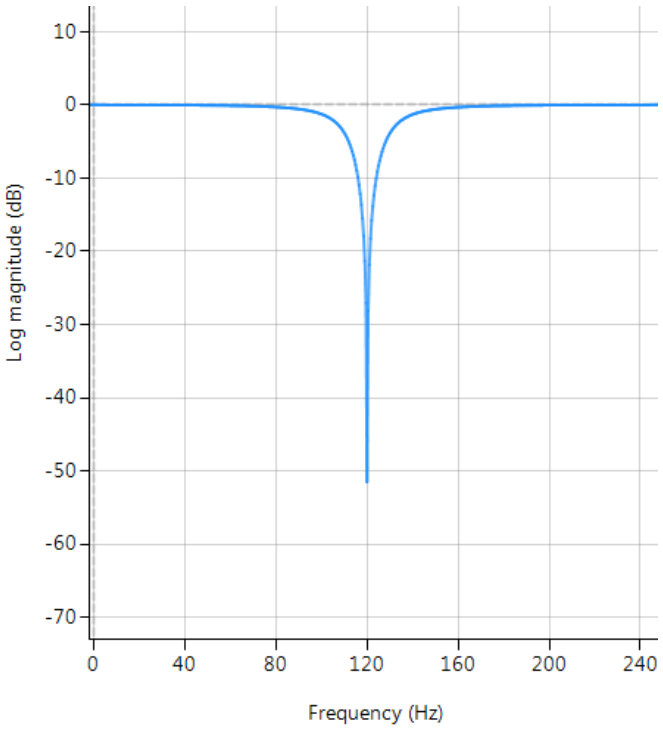
A notch filter can be constructed from an all-pass configuration (see section 3.3) by the following transfer function:

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

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()

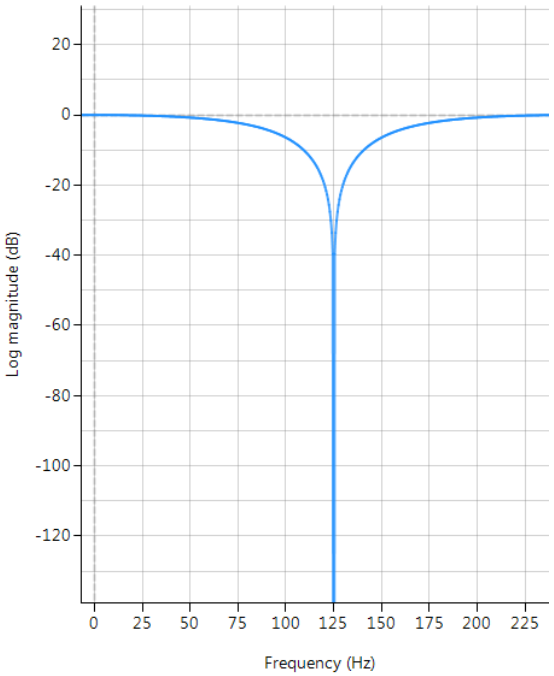
k1=-cos(2*pi*fc/fs);
k2=(1-tan(BW/2))/(1+tan(BW/2));

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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

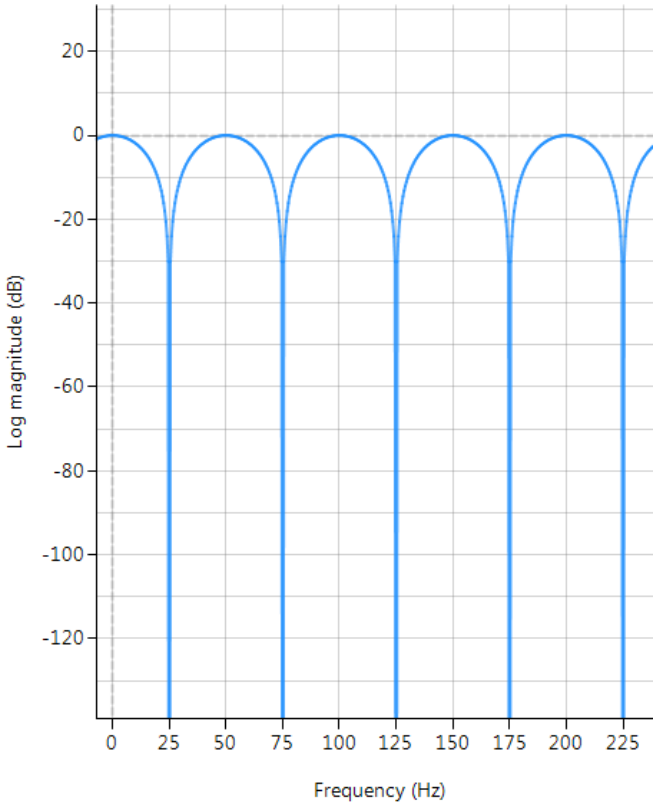
3.7. Comb (comb.afs)

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));
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

3.8. Fractional Farrow Delay

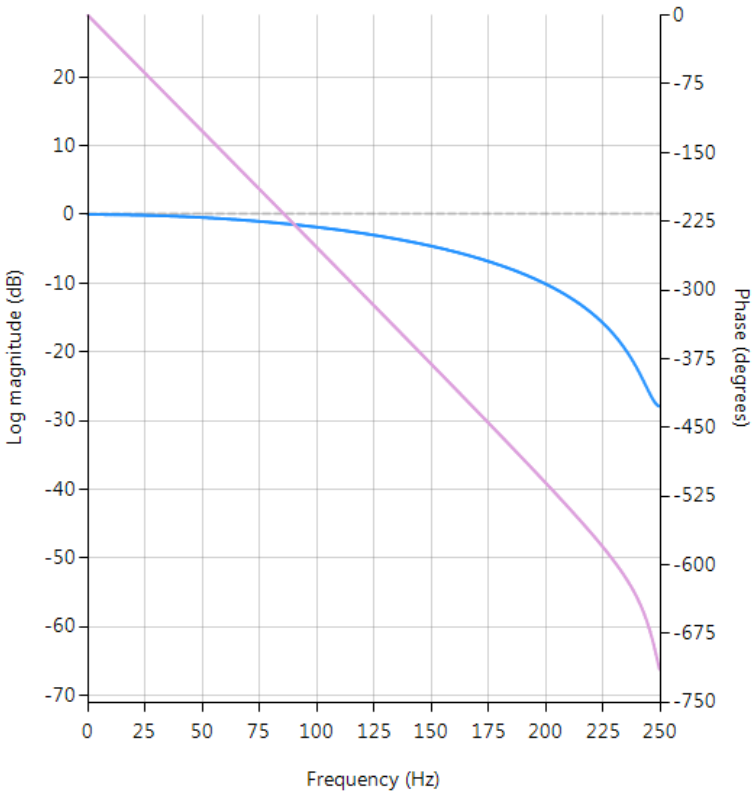
In signal processing, the need sometimes arises to nudge or fine-tune the sampling instants of a signal by a fraction of a sample. An FIR Farrow delay filter is typically employed to achieve this task, and may be combined with a traditional integer delay line in order to achieve a universal fractional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

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

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

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
```


Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.

Document Revision Status

Rev.	Description	Date
1	Document updated for v5 and released.	01/09/2021
2	updated for v5.1.0	27/10/2021
3	updated for v5.1.1	01/02/2022
4	updated for v5.1.2	27/04/2022
5	updated for v5.2.1	07/11/2022
6	updated for v5.2.2	31/03/2023
7	updated for v5.3.1	29/08/2023
8	updated for v5.3.2/3	25/01/2024
9	updated for v5.3.4	23/04/2024
10	updated for v5.3.5	07/01/2025
11	updated for v5.4.0	02/06/2025
12	updated for v5.4.2	14/04/2026

Copyright © 2026 Advanced Solutions Nederland BV. All rights reserved.