# FilterIIR

**FilterIIR** [***flags***] [***waveName,***…]

The FilterIIR operation applies to each *waveName* either the automatically-designed IIR filter coefficients or the IIR filter coefficients in *coefsWaveName*. Multiple filter designs are combined into a composite filter. The filter can be optionally placed into the first *waveName* or just used to filter the data in *waveName*.

The automatically-designed filter coefficients are bilinear transforms of the Butterworth analog prototype with an optional variable-width notch filter.

To design more advanced IIR filters, see **Designing the IIR Coefficients**.

### Parameters

*waveName* may be multidimensional, but only the one dimension selected by /DIM is filtered (for two-dimensional filtering, see **MatrixFilter**).

*waveName* may be omitted for the purpose of checking the format of *coefsWaveName*. If the format is detectably incorrect an error code will be returned in V_flag. Use /Z to prevent command execution from stopping.

### Flags

/CASC
: Specifies that *coefsWaveName* contains cascaded bi-quad filter coefficients. The cascade implementation is more stable and numerically accurate for high-order IIR filtering than Direct Form 1 filtering. See **Cascade Details**.

/COEF [=*coefsWaveName*]
: Replaces the first output *waveName* by the filter coefficients instead of the filtered results or, when *coefsWaveName* is specified, replaces the output wave(s) by the result of filtering *waveName* with the IIR coefficients in *coefsWaveName*.

  *coefsWaveName* must not be one of the destination *waveName*s. It must be single- or double-precision numeric and two-dimensional.

  When used with /CASC, *coefsWaveName* must have 6 columns, containing real-valued coefficients for a product of ratios of second-order polynomials (cascaded bi-quad sections).

  If /ZP is specified, it must be complex, otherwise it must be real.

  See **Details** for the format of the coefficients in *coefsWaveName*.

/DIM=*d*
: Specifies the wave dimension to filter.

  *d*=-1: Treats entire wave as 1D (default).
  *d*=0: Operates along rows.
  *d*=1: Operates along columns.
  *d*=2: Operates along layers.
  *d*=3: Operates along chunks.

  Use /DIM=0 to apply the filter to each individual column (each one a channel, say left and right) in a multidimensional *waveName* where each row comprises all of the sound samples at a particular time.

/ENDV=*ev*
: Values before the beginning of each filtered wave are replaced with *ev*. If you omit /ENDV they are replaced with zeros. To prevent filter startup artifacts set *ev* to the first value or a localized mean value at the start of the wave to be filtered.

  /ENDV was added in Igor Pro 9.00.

/HI=*fHigh*
: Creates a high-pass Butterworth filter with the -3dB corner at *fHigh*. The order of the filter is controlled by the /ORD flag.

  *fHigh* is a filter design frequency measured in fractions of the sampling frequency, and may not exceed 0.5 (the normalized Nyquist frequency).

| | |
|---|---|
| /LO=*fLow* | Creates a low-pass Butterworth filter with the -3dB corner at *fLow*. The /ORD flag controls the order of the filter.<br>*fLow* is a filter design frequency measured in fractions of the sampling frequency, and may not exceed 0.5 (the normalized Nyquist frequency).<br>Create bandpass and bandreject filters by specifying both /HI and /LO. For a bandpass filter, set *fLow* > *fHigh*, and for a band reject filter, set *fLow* < *fHigh*. |
| /N={*fNotch*, *notchQ*} | Creates a notch filter with the center frequency at *fNotch* and a -3dB width of *fNotch*/*notchQ*.<br>*fNotch* is a filter design frequency measured in fractions of the sampling frequency, and may not exceed 0.5 (the normalized Nyquist frequency).<br>*notchQ* is a number greater than 1, typically 10 to 100. Large values produce a filter that "rings" a lot. |
| /ORD=*order* | Sets the order of the Butterworth filter(s) created by /HI and /LO. The default is 2 (second order), and the maximum is 100. |
| /Z=*z* | Prevents procedure execution from aborting when an error occurs. Use /Z=1 to handle this case in your procedures using **GetRTError**(1) rather than having execution abort. /Z=0 is the same as no /Z at all. |
| /ZP | Specifies that *coefsWaveName* contains complex z-domain zeros (in column 0) and poles (in column 1) or, if *coefsWaveName* is not specified, that the first output *waveName* is to be replaced by filter coefficients in the zero-pole format. See **Zeros and Poles Details**. |

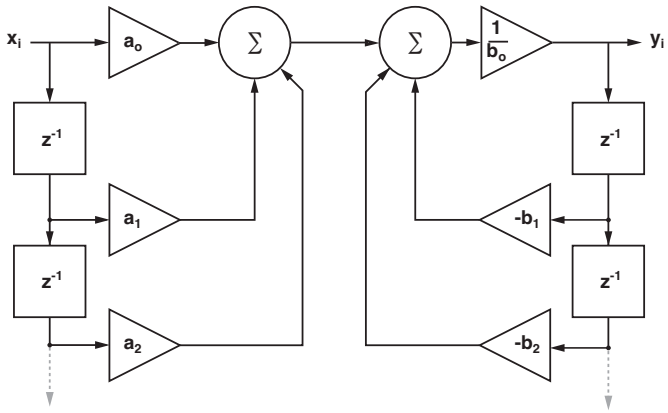
**Details**

FilterIIR sets V_flag to 0 on success or to an error code if an error occurred. Command execution stops if an error occurs unless the /Z flag is set. Omit /Z and call **GetRTError** and **GetRTErrMessage** under similar circumstances to see what the error code means.

**Direct Form 1 Details**

Unless /CASC or /ZP are specified, the coefficients in *coefsWaveName* describe a ratio of two polynomials of the Z transform:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots}{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots}$$




**Direct Form I Implementation**

$$y_i = \frac{a_0 x_i + a_1 x_{i-1} + a_2 x_{i-2} + \ldots - b_1 y_{i-1} - b_2 y_{i-2} + \ldots}{b_0}$$

where *x* is the input wave *waveName* and *y* is the output wave (either *waveName* again or *destWaveName*).

FilterIIR computes the filtered result using the Direct Form I implementation of *H*(*z*).

The rational polynomial numerator ($a_i$) coefficients in are column 0 and denominator ($b_i$) coefficients in column 1 of *coefsWaveName*.

The coefficients in row 0 are the nondelayed coefficients $a_0$ (in column 0) and $b_0$ (in column 1).

The coefficients in row 1 are the $z^{-1}$ coefficients, $a_1$ and $b_1$.

The coefficients in row n are the $z^{-n}$ coefficients, $a_n$ and $b_n$.

The number of coefficients for the numerator can differ from the number of coefficients for the denominator. In this case, specify 0 for unused coefficients.

**Note**: If all the coefficients of the denominator are 0 ($b_i$= 0 except $b_0$ = 1), then the filter is actually a causal FIR filter (Finite Impulse Response filter with delay of $n$-1). In this sense, FilterIIR implements a superset of the FilterFIR operation.

**Alternate Direct Form 1 Notation**

The designation of $a_i$, etc. as the numerator is at odds with many textbooks such as *Digital Signal Processing*, which uses $b$ for the numerator coefficients of the rational function, $a$ for the denominator coefficients with an implicit $a_0$ = 1, in addition to reversing the signs of the remaining denominator coefficients so that they can write $H(z)$ as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 - \sum_{i=1}^{n} a_i z^{-i}}.$$

Coefficients derived using this notation need their denominator coefficients sign-reversed before putting them into rows 1 through n of column 1 (the second column), and the "missing" nondelayed denominator coefficient of 1.0 placed in row 0, column 1.

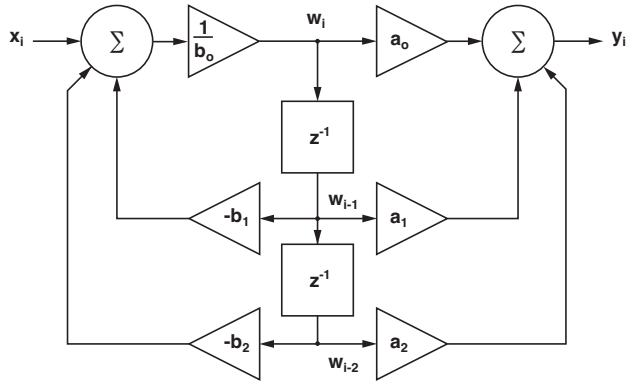
**Cascade Details**

When using /CASC, coefficients in *coefsWaveName* describe the product of one or more ratios of two quadratic polynomials of the Z transform:

$$H(z) = \frac{Y(z)}{X(z)} = \prod_{k=1}^{K} \frac{a_{0_k} + a_{1_k} z^{-1} + a_{2_k} z^{-2}}{b_{0_k} + b_{1_k} z^{-1} + b_{2_k} z^{-2}}.$$

Each product term implements a "cascaded bi-quad section", and $H(z)$ can be realized by feeding the output of one section to the next one.

The cascade coefficients filter the data using a Direct Form II cascade implementation:




$$w_i = \frac{x_i - b_1 w_{i-1} - b_2 w_{i-2}}{b_0}$$

$$y_i = a_0 w_i + a_1 w_{i-1} + a_2 w_{i-2}$$

**Cascaded Bi-Quad Direct Form II Implementation**

The cascade implementation is more stable and numerically accurate for high-order IIR filtering than Direct Form I filtering. Cascade IIR filtering is recommended when the filter order exceeds 16 (a 16th-order Direct Form I filter has 17 numerator coefficients and 17 denominator coefficients).

*coefsWaveName* must be a six-column real-valued numeric wave. Each row describes one bi-quad section. The coefficients for the second term (or "section") of the product (*k*=2) are in the following row, etc.:

| *k* | Row | Col 0 | Col 1 | Col 2 | Col 3 | Col 4 | Col 5 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | $a_{0_1}$ | $a_{1_1}$ | $a_{2_1}$ | $b_{0_1}$ | $b_{1_1}$ | $b_{2_1}$ |
| 2 | 1 | $a_{0_2}$ | $a_{1_2}$ | $a_{2_2}$ | $b_{0_2}$ | $b_{1_2}$ | $b_{2_2}$ |
| | … | | | | | | |

The number of coefficients for the numerator (*a*'s) is allowed to differ from the number of coefficients for the denominator (*b*'s). In this case, specify 0 for unused coefficients.

For example, a third order filter (three poles and three zeros) cascade implementation is a single-order section combined with a second order section. The values for $a_{2_k}$, $b_{2_k}$ for that section (*k*) would be 0. Here the second section is specified as the first-order section:

| *k* | Row | Col 0 | Col 1 | Col 2 | Col 3 | Col 4 | Col 5 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | $a_{0_1}$ | $a_{1_1}$ | $a_{2_1}$ | $b_{0_1}$ | $b_{1_1}$ | $b_{2_1}$ |
| 2 | 1 | $a_{0_2}$ | $a_{1_2}$ | 0 | $b_{0_2}$ | $b_{1_2}$ | 0 |

**Alternate Cascade Notation**

In the DSP literature, the $b_{0_k}$ gain values are typically one and the *H*(*z*) expression contains an overall gain value, usually *K*. Here each product term (or "section") has a user-settable gain value. Computing the correct gain values to control overflow in integer implementations is the responsibility of the user. For floating implementations, you might as well set all $b_{0_k}$ values to one except, say, $b_{0_1}$, to control the overall gain.

**Zeros and Poles Details**

When using /ZP, coefficients in *coefsWaveName* contains complex zeros and poles in the (also complex) Z transform domain:

$$H(z)=\frac{Y(z)}{X(z)}=\frac{(z-z_0)(z-z_1)(z-z_2)\ldots}{(z-p_0)(z-p_1)(z-p_2)\ldots}$$

*coefsWaveName* must be a two-column complex wave with zero0, zero1,… zeroN in the first column of N+1 rows, and pole0, pole1,… poleN in the second column of those same rows:

| k | Row | Col 0 | Col 1 |
|---|---|---|---|
| 1 | 0 | (zero0Real, zero0Imag) | (pole0Real, pole0Imag) |
| 2 | 1 | (zero1Real, zero1Imag) | (pole1Real, pole1Imag) |
| 3 | 2 | (zero2Real, zero2Imag) | (pole2Real, pole2Imag) |
| | … | | |

If a zero or pole has a nonzero imaginary component, the conjugate zero or pole must be included in *coefsWaveName*. For example, if a zero is placed at (0.7, 0.5), the conjugate is (0.7, -0.5), and that value must also appear in column 0. These two zeros form what is known as a "conjugate pair". The conjugate values must match within the greater of 1.0e-6 or 1.0e-6 * |zeroOrPole|.

Use (0,0) for unused poles or zeros, as a zero or pole at *z*= (0,0) has no effect on the filter frequency response.

The /ZP format for the coefficients is internally converted into the Direct Form 1 implementation, or into the Cascade Direct Form 2 implementation if /CASC is specified. There is no option for returning these implementation-dependent coefficients in a wave.

### Designing the IIR Coefficients

Simple IIR filters can be used or created by specifying the /LO, /HI, /ORD, /N, /CASC, and /ZP flags. Use /COEF without *coefsWaveName* to put these simple IIR filter coefficients into the first *waveName*.

More advanced IIR filters (Bessel, Chebyshev) can be designed using the separate Igor Filter Design Laboratory (IFDL). IFDL is an Igor package that you use to design FIR (Finite Impulse Response) and IIR (Infinite Impulse Response) filters and to apply them to your data. The IIR design software creates IIR coefficients based on bilinear transforms of analog prototype filters such as Bessel, Butterworth, and Chebyshev.

Even without IFDL, you can create custom IIR filters by manually placing poles and zeros in the Z plane using the Pole and Zero Filter Design procedures. Copy the following line to your Procedure window and click the Compile button at the bottom of the procedure window:

```
#include <Pole And Zero Filter Design>
```

Then choose Pole and Zero Filter Design from the Analysis menu.

### Examples

```
// Make test sound from three sine waves
Variable/G fs= 44100                            // Sampling frequency
Variable/G seconds= 0.5                         // Duration
Variable/G n= 2*round(seconds*fs/2)
Make/O/W/N=(n) sound                            // 16-bit integer sound wave
SetScale/p x, 0, 1/fs, "s", sound
Variable/G f1= 200, f2= 1000, f3= 7000
Variable/G a1=100, a2=3000,a3=1500
sound= a1*sin(2*pi*f1*x)
sound += a2*sin(2*pi*f2*x)
sound += a3*sin(2*pi*f3*x)+gnoise(10)           // Add a noise floor

// Compute the sound's spectrum in dB
FFT/MAG/WINF=Hanning/DEST=soundMag sound
soundMag= 20*log(soundMag)
SetScale d, 0, 0, "dB", soundMag

// Apply a 5 kHz, 6th order low-pass filter to the sound wave
Duplicate/O sound, soundFiltered
FilterIIR/LO=(5000/fs)/ORD=6 soundFiltered       // Second order by default

// Compute the filtered sound's spectrum in dB
FFT/MAG/WINF=Hanning/DEST=soundFilteredMag soundFiltered
soundFilteredMag= 20*log(soundFilteredMag)
SetScale d, 0, 0, "dB", soundFilteredMag

// Compute the filter's frequency and phase by filtering an impulse
Make/O/D/N=2048 impulse= p==0                    // Impulse at t==0
SetScale/P x, 0, 1/fs, "s", impulse
Duplicate/O impulse, impulseFiltered
FilterIIR/LO=(5000/fs)/ORD=6 impulseFiltered
FFT/MAG/DEST=impulseMag impulseFiltered
impulseMag= 20*log(impulseMag)
SetScale d, 0, 0, "dB", impulseMag
FFT/OUT=5/DEST=impulsePhase impulseFiltered
impulsePhase *= 180/pi                           // Convert to degrees
SetScale d, 0, 0, "deg", impulsePhase
Unwrap 360, impulsePhase                         // Continuous phase

// Graph the frequency responses
Display/R/T impulseMag as "IIR Lowpass Example"
AppendToGraph/L=phase/T impulsePhase
AppendToGraph soundMag, soundFilteredMag
ModifyGraph axisEnab(left)={0,0.6}
ModifyGraph axisEnab(right)={0.65,1}
ModifyGraph axisEnab(phase)={0.65,1}
ModifyGraph freePos=0, lblPos=60, rgb(soundFilteredMag)=(0,0,65535)
ModifyGraph rgb(impulseMag)=(0,0,0), rgb(impulsePhase)=(0,65535,0)
ModifyGraph axRGB(phase)=(3,52428,1), tlblRGB(phase)=(3,52428,1)
Legend
```