Resample

Resample [***flags***] ***waveName*** [**, *waveName***]…

The Resample operation resamples *waveName* by interpolating or up-sampling (set by /UP=*upSample*), lowpass filtering, and decimating or down-sampling (set by /DOWN=*downSample*).

Lowpass filtering is specified with /N and /WINF or with /COEF=*coefsWaveName*.

The sampling frequency (1/DimDelta) of a resampled output wave *waveName* is changed by the ratio of *upSample*/*downSample*. For example, if *upSample*=4 and *downSample*=3, then the final sampling rate is 4/3 of the original value.

Straight interpolation can be accomplished by setting *upSample* to the interpolation factor and *downSample*=1, in which case the sample rate is multiplied by *upSample*. Deltax(*waveName*) will be proportionally smaller. Linear interpolation is accomplished by setting *numReconstructionSamples*=3 and *windowKind*=None.

For decimation only, set *upSample*=1 and *downSample* to the decimation factor. The sample rate is divided by *downSample*, and deltax(*waveName*) will be proportionally larger.

Use **RatioFromNumber** to choose appropriate values for *upSample* and *downSample*, or use /SAME=*sWaveName* or /RATE=*sampRate*. See **Resampling Rates Example** for details.

When using /COEF=*coefsWaveName*, the filter coefficients should implement a low-pass filter appropriate for the *upSample* and *downSample* values or aliasing (filtering errors) will result. See **Advanced Externally-Supplied Low Pass Filter Example** for details.

Resampling Rates Flags

The *upSample* and *downSample* values define how much interpolation and decimation to perform. They can be set directly with /UP and /DOWN or indirectly with /SAME or /RATE

/DOWN=*downSample*

Down-samples or decimates the filtered result by this integer factor after up-sampling and lowpass filtering. The default is 1 (no down-sampling).

For example, /DOWN=3 places only every third value in the output wave.

Down-sampling divides the sampling rate of the filtered data by a factor of *downSample*. The DimDelta(*waveName*, *dim*) value is multiplied by the same factor.

/RATE=*sampRate*

Converts the output *waveName* to the specified sampling rate frequency (normally Hz).

The necessary *upSample* and *downSample* values for each *waveName* are computed internally as if you had executed:

```
RatioFromNumber (deltax(waveName)*sampRate)
upSample = V_numerator
downSample = V_denominator
```

/RATE returns V_numerator and V_denominator set to these automatically-determined values for the last *waveName*.

/SAME=*sWaveName*

Converts the output *waveName* to the same sampling rate as *sWaveName*, 1/DimDelta(*sWaveName*, *dim*). The necessary *upSample* and *downSample* values are computed internally as if you had executed:

```
Variable dd = DimDelta(waveName,dim)
RatioFromNumber dd/DimDelta(sWaveName,dim)
upSample = V_numerator
downSample = V_denominator
```

/SAME returns V_numerator and V_denominator set to these automatically determined values for the last *waveName*.

/UP=*upSample*

Up-samples or interpolates the input by this integer factor. The default is 1 (no up-sampling).

For example, /UP=4 inserts three extra points between each input point (producing 4 times as many values) before the lowpass filtering and down-sampling occurs.

Up-sampling multiplies the sampling rate of the input data by a factor of *upSample*, though no additional signal information is created. The DimDelta(*waveName*, *dim*) value is divided by the same factor.

Internal Sinc Reconstruction Filter Flags

If /COEF=*coefsWaveName* is not specified, Resample computes a windowed sinc filter from /N, /DOWN, /UP, and /WINF flag values.

If /COEF=*coefsWaveName* is specified, then *coefsWaveName* supplies the filter, and /N and /WINF are ignored. See **Externally-Supplied Low Pass Filter Flags**.

/COEF

Replaces the first *waveName* with coefficients generated by *downSample*, *upSample*, *numReconstructionSamples*, and *windowKind*, a windowed sinc impulse response.

When resampling multiple *waveName*s with different filters (because /RATE or /SAME were specified and the multiple *waveName*s had different sampling rates), the filter used to resample the last *waveName* is returned.

/N=*numReconstructionSamples*

Specifies the number of input values used to created the up-sampled values (default is 21).

The value of *numReconstructionSamples* must be odd.

The size of the computed filter is (*numReconstructionSamples*-1) * *upSample* + 1.

Bigger is better: 15 is usually on the low side for yielding reasonably accurate results, and although 101 will nearly always give very good results, it will be slow.

Use /COEF to output the impulse response, and the FFT to display the frequency response of the interpolator:

```
Make/O coefs
Variable numReconstructionSamples= 51, upSample= 5
Resample/COEF/N=(numReconstructionSamples)/UP=(upSample) coefs
Variable evenNum= 2*floor((numpnts(coefs)+1)/2)
FFT/OUT=3/PAD={evenNum}/DEST=coefs_FFT coefs
Display coefs_FFT
```

Bigger is also slower: the filtering is computed in the time-domain, and execution time is linearly related to

upSample/*downSample* * *numReconstructionSamples*.

/WINF=*windowKind*

Applies the window, *windowKind*, to the computed filter coefficients. If /WINF is omitted, the Hanning window is used. For no coefficient windowing, use /WINF=None, though this is discouraged unless you want linear interpolation, in which case it should be paired with /N=3.

Windows alter the frequency response of the filter in obvious and subtle ways, enhancing the stop-band rejection or steepening the transition region between passed and rejected frequencies. They matter less when *numReconstructionSamples* is large.

Choices for *windowKind* are:

Bartlett, Blackman367, Blackman361, Blackman492, Blackman474, Cos1, Cos2, Cos3, Cos4, Hamming, Hanning, KaiserBessel20, KaiserBessel25, KaiserBessel30, Parzen, Poisson2, Poisson3, Poisson4, Riemann and None.

See **FFT** for window equations and details.

Externally-Supplied Low Pass Filter Flags

/COEF =*coefsWaveName*

Identifies the wave, *coefsWaveName*, containing filter coefficients that implement a low-pass filter with a cutoff frequency of the lesser of 0.5/*upSample* and 0.5/*downSample*, where 0.5 corresponds to the Nyquist frequency of the up-sampled data.

For example, if *upSample*=2, then the filter must contain the classic "half-band" filter, which stops the higher half of the frequencies and passes the lower half. If *upSample*=10, then the filter must pass only the lowest 1/10th of the frequencies.

For *downSample* > *upSample*, the low-pass filter's cutoff frequency must be 0.5/*downSample*. This prevents the decimation from introducing aliasing to the resampled data.

To avoid shifting the output with respect to the input, *coefsWaveName* must have an odd length with the "center" coefficient in the middle of the wave.

The length of *coefsWaveName* must be 1+*upSample***n*, where *n* is any even integer.

Note: Instead of using /N=*numReconstructionSamples* with /COEF=*coefsWaveName*, *numReconstructionSamples* is computed from *upSample* and the number of points in *coefsWaveName*:

numReconstructionSamples=1+(numpnts(*coefsWaveName*)-1)/*upSample*.

Coefficients are usually symmetrical about the middle point, but this is not enforced.

coefsWaveName must not be a destination *waveName*.

coefsWaveName must be single- or double-precision numeric and one-dimensional.

Data Range Flags

/DIM=*d* Specifies the wave dimension to resample.

For *d* =0, 1, …, resampling is along rows, columns, etc.

The default is /DIM=0, which resamples each individual column (each one a channel, say left and right) in a multidimensional *waveName* where each row comprises all sound samples at a particular time.

To resample in multiple dimensions, execute the command once for each dimension. For example, use /DIM=0 followed by another command with /DIM=1 to resample a two-dimensional wave in each direction.

E=*endEffect* Determines how to handle the ends of the resampled wave(s) (w) when fabricating missing neighbor values.

endEffect=0:	Bounce method. Uses w[*i*] in place of the missing w[-*i*] and w[*n*-*i*] in place of the missing w[*n*+*i*].
endEffect=1:	Wrap method. Uses w[*n*-*i*] in place of the missing w[-*i*] and vice versa.
endEffect=2:	Zero method (default). Uses 0 for any missing value.
endEffect=3:	Repeat method. Uses w[0] in place of the missing w[-*i*] and w[*n*] in place of the missing w[*n*+*i*].

Parameters

waveName can be a wave with any number of dimensions. Only one dimension is resampled. Use multiple Resample calls to resample across multiple dimensions.

Without /DIM, resampling is done along the row (first) dimension for each column. That is, each column is resampled as if it were a separate one-dimensional wave. This allows multichannel audio to be resampled to another frequency.

If /DIM=1, then resampling proceeds across all the columns of each row.

If /COEF is specified without *coefsWaveName*, then the first *waveName* is overwritten by the filter coefficients instead of being resampled.

Details

The filtering convolution is performed in the time-domain. That is, the FFT is not employed to filter the data. For this reason the coefficients length (/N or the length of *coefsWaveName*) should be small in comparison to the resampled waves.

Resample should not be used on data that contains NaNs; use **Smooth**/M=NaN to replace NaN values with a local median, or use **MatrixOp** zapNaNs to remove them.

Resample assumes that the middle point of *coefsWaveName* corresponds to the delay=0 point. The "middle" point number = trunc((numpnts(*coefsWaveName*)-1)/2). *coefsWaveName* usually contains the two-sided impulse response of a filter, an odd number of points, and implements a low-pass filter whose cutoff frequency is the lesser of 0.5/*upSample* and 0.5/*downSample* (0.5 corresponds to the Nyquist frequency = 1/2 sampling frequency).

When /COEF creates a coefficients wave it sets the wave's X scaling deltax property to the deltax of the input wave multiplied by the /DOWN factor and divided by the /UP factor. It also sets the x0 property so that the zero-phase (center) coefficient is located at x=0.

Simple Examples

```
// Interpolation by factor of 4, default filter
Resample/UP=4 data

// Interpolation by factor of 7, linear interpolation
Resample/UP=7/N=3/WINF=None data

// Decimation by factor of 3, default filter
Resample/DOWN=3 data

// Match sampling rates, default filter
Resample/SAME=dataAtDesiredRate dataAtWrongRate1, dataAtWrongRate2,...

// Resample waves to 10 KHz sampling rate
Resample/RATE=10e3 dataAtWrongRate1, dataAtWrongRate2,...

// Interpolate an image by a factor of 2
Resample/UP=2 image              // default is /DIM=0, resample rows
Resample/UP=2/DIM=1 image        // resample across columns
```

Note: Interpolating by a factor of two does not produce an image with twice as many rows and columns. The new number of rows = (original rows-1)**upSample* +1, and a similar computation applies to columns.

Resampling Rates Example

Suppose we have an audio wave sampled at 44,100 Hz and we wish to resample it to a higher 192,000 Hz frequency.

We can use /RATE= 192000 and let Resample determine the correct values (provided *waveName* has its X scaling set properly to reflect sampling at 44100 Hz), but let's compute *upSample* and *downSample* ourselves.

Because the sampling rate = 1/deltax(*wave*), we can recast the /SAME formula to RatioFromNumber (*desiredSamplingRate*/*currentSamplingRate*):

```
•RatioFromNumber/V (192000 / 44100)
  V_numerator= 640; V_denominator= 147;
  ratio= 4.3537414965986;
  V_difference= 0;
```

Then *upSample*=640 and *downSample*=147.

The 44100 Hz input data will be interpolated by 640 to 28,224,000 Hz.

The result is low-pass filtered with a "cutoff frequency" of 1/640th of the interpolated Nyquist frequency = (28224000/2)/640 = 22,050 Hz, the same as the input signal's original Nyquist frequency.

The result will be decimated by 147 to 192,000 Hz, which is the desired output sampling frequency.

Note: If *downSample* had been greater than *upSample*, then the low-pass filter's cutoff frequency would have been 1/*downSample*th of the interpolated Nyquist frequency = (28224000/2)/*downSample*. This prevents the decimation from introducing aliasing to the resampled data.

```
Resample/UP=640/DOWN=147 sound          // convert 44.1 KHz to 192 KHz
```

Advanced Externally-Supplied Low Pass Filter Example

You can generate an appropriate filter by executing commands like these:

```
// Compute a filter for after the input is upsampled
// to restore the frequency content to the original range.
Variable fc = min(0.5 / upSample, 0.5 * upSample / downSample)
// Transition width, small widths need big n
Variable tw= fc/10
// Set end of pass band
Variable f1= fc-tw/2
// Set start of stop band
Variable f2= fc+tw/2
// Use bigger values of n to make the filter smoother
Variable nReconstruct= 31
Variable n= (nReconstruct-1)*upSample+1        // odd = no phase shift
// Create a wave to hold the coefficients; it gets resized to n
Make/O/N=0 coefsWaveName
FilterFIR/COEF/LO={f1,f2,n} coefsWaveName
```

However, FilterFIR does not create windowed sinc lowpass filters that have the endearing property that the original input values are unaltered in the filtered output, though only if *upSample* > *downSample*. This is called a "Nyquist filter" or "Kth-band filter" in the literature.

If *upSample* > *downSample*, you can enforce the Nyquist criterion by "zeroizing" the designed filter by setting every *upSample*th value to 0 except the center one.

```
// coefsWaveName length must be 1+upSample*n, where n is any even integer
Function Zeroize(w, upSample)
    Wave w                          // coefsWaveName
    Variable upSample               // upSample value

    Variable n= DimSize(w,0)
    Variable centerP= floor((n-1)/2)        // if n=101, centerP= 50
    Variable i
    for (i=0; i<n; i+=upSample)
        if( i != centerP )
            w[i] = 0
        endif
    endfor
End
```

Resample zeroizes the internally-generated low pass filter when *upSample* > *downSample*.

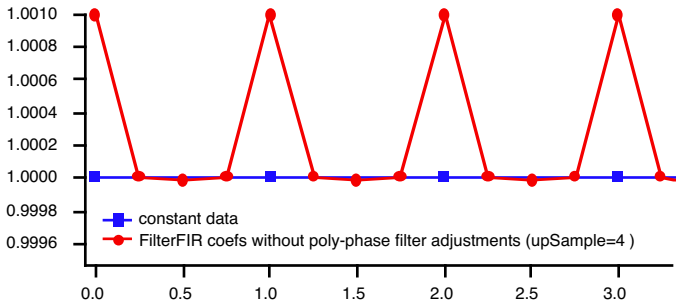
Additionally, the FilterFIR command generates a low-pass filter whose gain needs to be multiplied by *upsample*:

coefsWaveName ***= upSample***

When designing an externally supplied filter, you should also consider the filter's "polyphase" nature; *coefsWaveName* is actually a set of *upSample* interleaved filters, each with its own response. It makes sense to adjust these filters to produce consistent responses. If you don't, the results will contain ringing with a period of *upSample*/*downSample*. This is most apparent when *downSample* is 1.

Using the filter we've designed so far with *upSample*=4, here's the output of a constant-input wave:

```
Make/O constantData= 1
Resample/COEF=coefsWaveName/UP=4 constantData
```

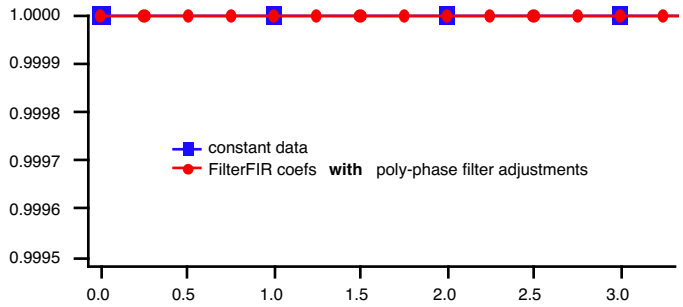


The graph shows that the filter response at 0 Hz for the first of 4 filters is 1.0010, the second and fourth filter's responses are very close to 1.0, and the third filter's response at 0 Hz is a little less than 1.0.

These variations can be eliminated by normalizing the sum of each polyphase filter to 1.0:

```
Function PolyphaseNormalize(w, upSample)
    Wave w                              // coefsWaveName
    Variable upSample                   // upSample value

    Variable n= DimSize(w,0)
    Variable filt
    // for each filter (0..upSample-1)
    for (filt=0; filt<upSample; filt+=1)
        Variable total=0
        Variable pt
        // compute total for this filter
        for (pt=filt; pt<n; pt+=upSample)
            total += w[pt]
        endfor
        // divide by total to normalize total to 1
        for (pt=filt; pt<n; pt+=upSample)
            w[pt] /= total
        endfor
    endfor
End
```



Now the filter is ready to be used to filter data:

```
Resample/COEF=coefsWaveName/UP=(upSample)/DOWN=(downSample) dataWave
```

You can see that designing an externally-supplied lowpass filter is much more complicated than using the internal sinc reconstruction filter, which does all this zeroizing, scaling, and polyphase normalization for you.

References

Mintzer, F., On half-band, third-band, and Nth band FIR filters and their design, *IEEE Trans. on Acoust., Speech, Signal Process., ASSP-30*, 734-738, 1982.

See Also

RatioFromNumber, **FilterFIR**, **interp**, **Interp2D**, **Interpolate2**, **ImageInterpolate**, **Loess**, **Smooth**, **Multidimensional Decimation**