See Also

Text Encodings on page III-459, **Text Encoding Names and Codes** on page III-490

TextEncodingCode, **TextEncodingName**, **SetWaveTextEncoding**

ConvertGlobalStringTextEncoding, **String Variable Text Encoding Error Example** on page III-479

ConvexHull

`convexHull [`*`flags`*`]`*`xwave, ywave`*

`convexHull [`*`flags`*`] `*`tripletWave`*

The ConvexHull operation calculates the convex hull in either 2 or 3 dimensions. The dimensionality is deduced from the input wave(s). If the input consists of two 1D waves of the same length, the number of dimensions is assumed to be 2. If the input consists of a single triplet wave (a wave of 3 columns), then the number of dimensions is 3.

In 2D cases the operation calculates the convex hull and produces the result in a pair of x and y waves, W_XHull and W_YHull.

In 3D cases the operation calculates the convex hull and stores it in a triplet wave M_Hull that describes ordered facets of the convex hull.

ConvexHull returns an error if the input waves have fewer than 3 data points.

Flags

/C	(2D convex hull only) adds the first point to the end of the W_XHull and W_YHull waves so that the first and the last points are the same.
/E	(3D case only) if you use this flag the operation also creates a wave that lists the indices of the vertices which are not part of the convex hull, i.e., vertices which are interior to the hull. The output is in the wave W_HullExcluded.
/I	(3D convex hull only) use this flag to get the corresponding index of the vertex as the fourth column in the M_Hull wave.
/S	(3D convex hull only) use this flag if you want the resulting M_Hull to have NaN lines separating each triangle.
/T=*tolerance*	(3D case only) default tolerance for measuring if a point is inside or outside the convex hull is $1.0x10^{-20}$. You can use any other positive value.
/V	(3D case only) if you use this flag the operation also creates a wave containing the output in a list of vertex indices. The wave M_HullVertices contains a row per triangle where each entry on a row corresponds to the index of the input vertex.
/Z	No error reporting.

Examples

```
Make/O/N=33 xxx=gnoise(5),yyy=gnoise(7)
Convexhull/c xxx,yyy
Display W_Yhull vs W_Xhull
Appendtograph yyy vs xxx
ModifyGraph mode(yyy)=3,marker(yyy)=8,rgb(W_YHull)=(0,15872,65280)
```

See Also

Triangulate3D

Convolve

`Convolve [/A/C] `*`srcWaveName, destWaveName`*` [, `*`destWaveName`*`]`...

The Convolve operation convolves *srcWaveName* with each destination wave, putting the result of each convolution in the corresponding destination wave.

Convolve is not multidimensional aware. Some multidimensional convolutions are covered by the **MatrixConvolve**, **MatrixFilter**, and **MatrixOp** operations

Flags

/A	Acausal linear convolution.
/C	Circular convolution.

Details

Convolve performs linear convolution unless the /C or /A flag is used. See the diagrams in the examples below.

Depending on the type of convolution, the destination waves' lengths may increase. *srcWaveName* is not altered unless it also appears as a destination wave.

If *srcWaveName* is real-valued, each destination wave must be real-valued, and if *srcWaveName* is complex, each destination wave must be complex, too. Double and single precision waves may be freely intermixed; calculations are performed in the higher precision.

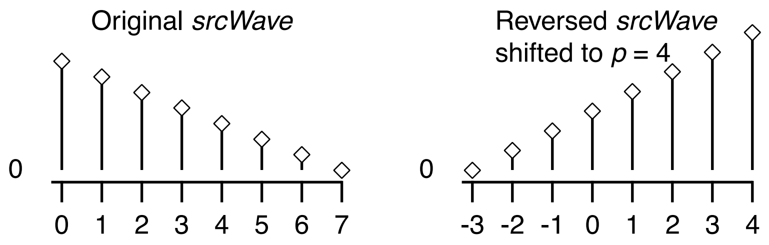
The linear convolution equation is:

$$destWaveOut[p] = \sum_{m=0}^{N-1} destWaveIn[m] \cdot srcWave[p-m]$$

where *N* is the number of points in the longer of *destWaveIn* and *srcWave*. For circular convolution, the index [*p -m*] is wrapped around when it exceeds the range of [0,numpnts(*srcWave*)-1]. For acausal convolution, when [*p -m*] exceeds the range a zero value is substituted for *srcWave* [*p -m*]. Similar operations are applied to *destWaveIn* [*m*].

Another way of looking at this equation is that, for all *p*, *destWaveOut*[*p*] equals the sum of the point-by-point products from 0 to *p* of the destination wave and an end-to-end reversed copy of the source wave that has been shifted to the right by *p*.

The following diagram shows the reversed/shifted *srcWave* that would be combined with *destWaveIn*. The points numbered 0 through 4 of the reversed *srcWave* would be multiplied with *destWaveIn*[0…4] and summed to produce *destWaveOut*[4]:



For linear and acausal convolution, the destination wave is first zero-padded by one less than the length of the source wave. This prevents the "wrap-around" effect that occurs in circular convolution. The zero-padded points are removed after acausal convolution, and retained after linear convolution. The X scaling of the waves is ignored.

The convolutions are performed by transforming the source and destination waves with the Fast Fourier Transform, multiplying them in the frequency domain, and then inverse-transforming them into the destination wave(s).

The convolution is performed in segments if the resulting wave has more than 256 points and the destination wave has twice as many points as the source wave. For acausal convolution, the length of the resulting wave is considered to be (numpnts(*srcWaveName*) +numpnts(*destWaveName*)-1) for this calculation.

Applications

The usual application of convolution is to compute the response of a linear system defined by its impulse response to an input signal. *srcWaveName* would contain the impulse response, and the destination wave would initially contain the input signal. After the Convolve operation has completed, the destination wave contains the output signal.

Use linear convolution when the source wave contains an impulse response (or filter coefficients) where the first point of *srcWave* corresponds to no delay (*t* = 0).