gconv

The procedure gconv computes the discrete convolution of an input array with a Gaussian kernel.

Syntax

GCONV,x,y,fwhm,nx,ny[,ppr=ppr][,nsigma=nsigma][,inter=inter] [,original=original]

Return Value

GCONV returns the smoothed spectrum, and a (possibly resampled) array of abscissae.

Arguments

x - (float array) input array of abscissae (e.g. wavelength)

y - (float array) input array (ordinates, e.g. flux array)

fwhm - (float) FWHM of the Gaussian kernel (pixels)

nx - (float array) output array of abscissae (e.g. wavelength)

ny - (float array) output array (ordinates, e.g. flux array)

Keywords

- ppr - (float) number of pixels per resolution element
- nsigma - (float) how far the convolution goes in terms of the kernel sigma (default is 3)
- inter - if set, the x-axis is interpolated to force a constant step; if the step is already constant, 'inter' will have no effect
- original if set, the original sampling is kept for the output arrays, unless 'inter' is also set and an interpolation is performed, in which case, the output x-axis (array nx) will just have the same number of points as the input x, but resampled to have a constant step.

Discussion

The code performs a straight (i.e. in direct space) discrete convolution of an input array (y) with a Gaussian kernel

$$ny_i = (y \star G)_i = \sum_{j=-m}^{m} y_{i-j} G_j, \tag{1}$$

where the kernel G is sampled symmetrically around the origin with 2×m+1 points. Because G is symmetric, we can also write

$$ny_i = \sum_{j=-m}^{m} y_{i+j} G_j, \tag{2}$$

which is what the code actually implements.

By default this routine automatically adjustes the sampling, keeping the number of pixels per resolution element to be 3 – this can be changed by varying the keyword 'ppr'. When the width of the Gaussian kernel is large, this gives a substantial speed up compared to the case of keeping all the points in the arrays (frequencies)[1].

Note that if the kernel is not well sampled (FWHM< 2) the results will be inaccurate.

Near the edges the discrete convolution becomes ill-defined, as we run out of data. There are several workarounds for this issue, such as considering the array represents one cycle of a periodic function, but here we just trim the edges, so that the first data in the output arrays are sufficiently far from the edges of the input arrays (>nsigma×sigma or about |FWHM*1.27| pixels with the default nsigma=3).

References

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. 1986, Numerical Recipes, Cambridge: Cambridge University Press

Examples

1. Smooth an input spectrum with a Gaussian of FWHM=2.0 AA assuming the spectrum is given on a uniform (linearly sampled) wavelength scale with a step of 0.1 AA.

[1] Thanks to Lars Koesterke for this idea!

```
IDL>  fwhm=2.0/0.1
IDL>  gconv,w,f,fwhm,w2,f2
```

2. Smooth the same spectrum with a Gaussian of FWHM=50. km/s

```
IDL>  step=(max(alog(w))-min(alog(w)))/n_elements(w)
IDL>  fwhm=50./299792.458/step
IDL>  gconv,alog(w),f,fwhm,w2,f2,/inter
```

Because of the keyword inter, the spectrum will be internally interpolated to a step in ln (w) of step=(max(alog(w))-min(alog(w)))/n_elements(w) Then the 50 km/s (or V/c = 0.000166782) is equivalent to 50./299792.458/step pixels.

Version History

C. Allende Prieto, Univ. of Texas, coded in March 2006