URL: http://cxc.harvard.edu/ciao3.4/aconvolve.html
Last modified: December 2006

AHELP for CIAO 3.4

aconvolve

Context: tools

Jump to: Description Examples Parameters CHANGES IN CIAO 3.3 CHANGES IN CIAO 3.0 Bugs See Also

Synopsis

Convolve an N−dimensional image with a kernel

Syntax

```
aconvolve  infile outfile kernelspec [writekernel] [kernelfile]
[writefft] [fftroot] [method] [edges] [const] [pad] [center]
[normkernel] [clobber] [verbose] [kernel]
```

Description

`aconvolve' convolves an N−dimensional image with a kernel (specified by the parameter 'kernelspec'). The kernel can be specified in one of several flexible ways: as either a file (image), a function, from the library, or specified on the command line. Other than memory restrictions of the machine, there are no limits to the size or dimensionality of either the kernel or image (i.e. the kernel could be larger than the image). One of the most common applications of this tool is smoothing of an image by convolving with a gaussian.

The convolution of a data set with a kernel yields a measure of "how much" the data set looks like the kernel at every location in the data set. Another way to think about the convolution operation is a filtering of the dataset by the kernel.

Two forms of convolution are available:

- spatial domain (sliding cell);
- frequency domain (FFT).

Frequency−domain convolution is the standard technique: take two functions, Fourier transform them (using the "FFTW" routine from MIT), take their product in Fourier space, and then back transforming the product to the spatial domain to produce the convolved result. When using method="fft", any size image can be used but the algorithm is faster when the data is "padded" in size to take advantage of certain algorithmic optimisations; see the pad parameter.

In spatial−domain convolution `aconvolve' can treat edge−located pixels in one of 5 ways:

- "wrap" the edges;
- "mirror" them;
- set them to a "constant";
- "renorm"−alize them;
- use the "nearest" neighbor value.

Example 1

```
aconvolve test_data.fits outfile.fits "lib:gaus(2,5,1,10,10)" method=fft
```

Convolve (smooth) the image in test_data.fits with a Gaussian that

- has 2 dimensions;
- is embedded in an array 5 sigma in size;
- is normalized to 1;
- has a sigma of 10 pixels along each axis.

The convolution is done by the method of FFTing the input arrays, multiplying, and taking the inverse. It will work with any size array but it is much faster if the size is a power of 2.

Example 2

```
aconvolve test_data.fits outfile.fits "file:/tmp/foo.fits" edges=wrap
pad=no
```

Convolve the image in test_data.fits with the image in the file /tmp/foo.fits wrapping the edges and storing the data in outfile.fits. The input data are not padded.

Example 3

```
aconvolve test_data.fits outfile.fits "lib:box(2,1,3,3):(1,1)" pad=yes
method=fft writefft=no
```

Convolve (smooth) the image test_data.fits with a box that is normalized to 1, has two dimensions, and is 3 units long in each direction. The center of the box is a (1,1). The convolution is done by the method of FFTing the input arrays, multiplying, and taking the inverse. The data are padded to the next power of 2**N before doing the convolution. It will work with any size array but it is much faster if the size is a power of 2.

Example 4

```
aconvolve test_data.fits outfile.fits
"txt:((1,1,1),(1,1,1),(1,1,1)):(1,1)" edges=constant const=0 pad=yes
```

Same as Example 3. The "box" is implemented from the command line as a text array. The edges are padded with a constant = 0.

Parameters

name	type	ftype	def	min	max	reqd	autoname
infile	file	input				yes	
outfile	file	output				yes	yes
kernelspec	string					yes	
writekernel	boolean		no				
kernelfile	file	output	./.				yes
writefft	boolean		no				
fftroot	file	output	./.				yes
method	string		slide				
edges	string		wrap				
const	real		0				
pad	boolean		no				
center	boolean		no				
normkernel	string		area				
clobber	boolean		no				
verbose	integer		0	0	5		
kernel	string		default				

Detailed Parameter Descriptions

Parameter=infile `(file required filetype=input)`

Input image file.

The input is an image and can have any number of dimensions. The input can have the following data types: "short" (BITPIX=16), "long" (BITPIX=32), "float" (BITPIX=−32), and "double" (BITPIX=−64).

The image can be a virtual image as defined by the datamodel. Thus one could specify a virtual image by using the "bin" syntax like my_file.fits[EVENTS][bin x=1:100:1, y=1:100:1] to specify a 2D image binned on X and Y columns in EVENTS extension of my_file.fits file.

Parameter=outfile `(file required filetype=output autoname=yes)`

Output image file.

The output file is an image and has the same dimensions as the input image. The output data type is always "float" (BITPIX=−32). The autoname filename suffix is "_conv".

The output format is controlled by the kernel parameter. The output will either be a FITS image (kernel="fits"), an IRAF .imh/.pix file (kernel="iraf"), or will default to whatever format the input is in (kernel="default").

Note: To completely specify the file name one should specify the file and extension name like:

- myfile.fits[foo] −− for fits
- or ./myout.imh −− for iraf

Not specifying the filename with the block specified (i.e. the part in [ ]'s) may result in some odd names.

Parameter=kernelspec `(string required)`

Kernel specification.

The generic syntax for the kernelspec specification is: [key]:[parameters]:[origin]

Where −

- [key] = file : This tells the program to read the kernel from the image stored in the file [parameters]. The format of the specified file can be a FITS image of any of the data types specified for the input file.
- [key] = txt : This tells the program to parse the string [parameters] to create the kernel. Ex. see example #3 (example of a 2D kernel).
- [key] = lib : This tells the program to parse the string [parameters] for two things a) which library and b) parameters needed for that library call. The parameters needed follow the library specification in "()"'s.

Currently supported libraries are:

box

A multi−dimensional array with constant value. The parameters are (D, N, D1, D2,...DD) where:

- D = number of dimensions
- N = normalization (constant value)
- D1 − DD = length of box in each dimension.

gaus

A multi−dimensional, non−rotated Gaussian. The parameters are (D, M, N, S1, S2, ... SD) where:

- D = number of dimensions
- M = number of sigma to extend in each direction
- N = normalization

- S1−SD = sigmas in each direction (units = pixels)

tophat

A two−dimensional non−rotated elliptical top−hat function. The parameters are (D, N, D1, D2) where:

- D = number of dimension (ONLY = 2)!
- N = normalization (what value)
- D1 D2 = radii of ellipse axes along 1st and 2nd axes.

mexhat

A multi−dimensional, mexican hat function.

```
mexhat = Norm * exp( −0.5 * (x/s1)^2 + (y/s2)^2 + ...)
```

The parameters are (D, M, N, S1, S2, ... SD) where:

- D = number of dimensions
- M = number of sigma to extend in each direction
- N = normalization
- S1−SD = sigmas (width of mexican hat; units = pixels)

exp

A multi−dimensional exponential function.

```
exp = Norm * exp(−abs(x/s1)) * exp(−abs(y/s2)) * ...
```

The parameters are (D, M, N, S1, S2, ... SD) where:

- D = number of dimensions
- M = number of sigma to extend in each direction
- N = normalization
- S1 − SD = sigmas (width of exponential function; units = pixels)

power

A multi−dimensional power law function.

```
power = Norm * abs(x)^s1 * abs(y)^s2 * ...
```

The parameters are (D, M, N, S1, S2, ... SD) where:

- D = number of dimensions
- M = number of sigma to extend in each direction

- N = normalization
- S1 − SD = sigmas (width of power law function; units = pixels)

beta / lorentz

A multi−dimensional beta/lorentzian function.

```
beta = Norm / ((x^2 +y^2+...)/ s1^2)
```

The parameters are (D, M, N, S1) where:

- D = number of dimensions
- M = number of sigma to extend in each direction
- N = normalization
- S1 = sigma (width of beta/lorentzian function; units = pixels)

sinc

A multi−dimensional sinc function.

```
sinc= Norm * sin( 2*pi * x^2/s1 * y^2/s1 * ...
```

The parameters are (D, M, N, S1) where:

- D = number of dimensions
- M = number of sigma to extend in each direction
- N = normalization
- S1 = sigma (width of sinc function; units = pixels)

The (optional) origin specification is used to locate the origin of the kernel. Typically 2D kernels have the origin in the "center" of the image. Many 1D kernels have the origin as the first element in the array.

The user can explicitly specify the origin. This alleviates common restrictions in some other tools (e.g. length of data axes must be an odd number).

If no origin is specified, the default is to assign the origin to the center of the array (rounded down). Thus a (5,4) array will have its origin set at (2,2).

Parameter=writekernel `(boolean default=no)`

Output kernel to an image?

Parameter=kernelfile `(file filetype=output default=./. autoname=yes)`

File name for kernel image. Autoname suffix is "_kernel".

Parameter=writefft (boolean default=no)

Output FFT of data and kernel?

Parameter=fftroot (file filetype=output default=./. autoname=yes)

Root file name for FFT output files. The autoname suffix for the real image is "_kernel_real". For the imaginary image, "_kernel_imag". The data file suffixes are "_data_real" and "_data_imag". ALWAYS use autonaming in this parameter.

Parameter=method (string default=slide)

Convolution method: (slide|fft).

The convolution of a data set with a kernel yields a measure of "how much" the data set looks like the kernel at every location in the data set. Another way to think about the convolution operation is a filtering of the dataset by the kernel.

Two convolution methods are implemented. Under some simple constraints they will provide the same results.

method=slide

Sliding cell convolution is a convolution from first principles.

```
O(a1,a2,...aN) = S_x1 S_x2... S_xN
```

```
I(x1,x2,..xN)*K(a1−x1,a2−x2,...aN−xN).
```

[replace S_x by summation signs]

method=fft

The FFT of the data and the kernel is computed. The arrays are multiplied and the inverse FFT is taken. The FFT route used is FFTW.

Internally the kernel and data MUST be the same size. The program will pad either/or both the data and/or kernel to the maximum length along either axis, defined by the larger of the two datasets.

If the edges of the data are "wrapped" (see below) then the FFT and slide methods will yield the same results.

For moderately−large to large kernels, the FFT convolution is much faster (O(N log N) as opposed to O(N**2)), however a) it requires much more memory and b) it restricts the edge treatment to "wrap".

Parameter=edges (string default=wrap)

How should the code handle edges? (wrap/nearest/mirror/constant/renorm).

As the kernel moves across the data, the edge of the kernel may fall off the data space. When this happens the program needs to know how the user wants to handle the edges. Five methods for handling the edges are implemented: For the 5 cases described below, consider the following example data space:

```
  data = { 1, 2, 3}
```

- edges = wrap : When the kernel runs off the data space, it is wrapped around the data. Thus the data goes {...1,2,3,1,2,3,1,2,3...}
- edges = nearest : When the kernel runs off the data space, extrapolate the data nearest to the edge, e.g. data goes {...1,1,1,2,3,3,3,3,3...}
- edges = mirror : When the kernel runs off the data space, reflect the data nearest the edge, e.g. data goes {...3,2,1,1,2,3,3,2,1...}
- edges = constant : When the kernel runs off the data space, use the constant supplied by the const parameter, e.g. const=0, data goes {...0,0,1,2,3,0,0,..}
- edges = renorm : When the kernel runs off the data space, use a constant = 0 at the edge; however, re−normalize the kernel by the amount of area that remains on the data space.

If you set method="fft" then you are implicitly using the edges="wrap" option.

Parameter=const `(real default=0)`

Constant value to use at edges.

Parameter=pad `(boolean default=no)`

Pad the data to the next integer power of 2.

The user can specify that the data be padded such that the length of each data axes is promoted to the next integer power of 2. The data are always padded on the "right" hand side, thus the array 1,2,3,4,5 would be padded to 1,2,3,4,5,0,0,0. This way of padding results in no change in the origin.

Parameter=center `(boolean default=no)`

Center FFT output?

If center = yes, zero frequency will appear at the center of the FFT images. This option is currently inoperative.

Parameter=normkernel `(string default=area)`

Normalize the kernel?

Allows aconvolve to automatically normalize the kernel. There are three options.

- area − divide the kernel values by the sum of the values. NB: negative values in kernel may produce undesired result.
- max − divide the kernel values by the maximum kernel value (thus the max value is normalized to 1).
- none − do not normalize the kernel.

Parameter=clobber `(boolean default=no)`

Clobber existing output?

Parameter=verbose `(integer default=0 min=0 max=5)`

Verbosity level: 0 − no output, 5 − max display.

Parameter=kernel `(string default=default)`

This sets the output format. The options are fits file and default, where default will use the format of the infile.

CHANGES IN CIAO 3.3

Several new kernels have been added to the tool: mexhat, power, exp, beta/lorentz, and sinc. See the "kernelspec" parameter information for details.

CHANGES IN CIAO 3.0

The normkernel parameter has been added to the parameter list. This allows you to set the kernel normalisation to one of several methods. For instance, to ensure that counts are preserved you should set normkernel to "area" (assuming that your kernel is positive).

Bugs

See the bugs page for this tool on the CIAO website for an up−to−date listing of known bugs.

See Also

sherpa
: tpsf, tpsf1d

tools
: acrosscorr, apowerspectrum, arestore, csmooth, dmcoords, dmfilth, dmregrid, mkpsf, psf_project_ray

URL: http://cxc.harvard.edu/ciao3.4/aconvolve.html
Last modified: December 2006