Version 1.0

5 March 2001

Chandra **X-ray Center**
Smithsonian Astrophysical Observatory
60 Garden Street
Cambridge, MA 02138
USA

WWW: http://asc.harvard.edu/

Contents

List of Figures

List of Tables

Contributors to this document

Margarita Karovska	Lead Scientist
Ken Glotfelty	Software Engineer
H. Ebeling	Originator of the Asmooth algorithm
D. E. Harris	Editor, TCD User Guide
Joan Flanagan	Document Series Production (ASC)
R. Kilgard	Document Series Production (ASC)
D. Martinez-Galarce	Beta tester
Y. Zhou	Software Developer

Preface and Acknowledgements

This Guide is designed to serve as an introduction, user's guide, and a reference manual to the **TCD** suite of transforms and convolutions.

We gratefully acknowledge the aid and support of various members of the *Chandra* project including Science Data Systems group, and the Data Systems group.

- Margarita Karovska, March 5, 2001

Part I

Introduction to TCD

Chapter 1

Overview of TCD

1.1 What is TCD ?

Transforms-Convolutions-Deconvolutions (TCD) is a package of advanced analysis tools. The tools are designed as stand-alone tasks, although they also function as a library for use in other programs. The TCD package is expected to evolve with the addition of new tools from time to time. Currently, the following tasks are operational:

aconvolve: a convolution performed either in the map or Fourier plane; available functions include a box, a Gaussian, and a tophat; a user supplied file may also be used.

acrosscorr: cross correlation and autocorrelation

apowerspectrum: a power spectrum

atransform: a fast Fourier transform; other functions will be added. (NOT CURRENTLY RELEASED.)

csmooth: a set of smoothing algorithms. Currently it includes only 'asmooth', an adaptive smoothing code (Ebeling, 1999). Other functions will be added in the future.

For each of these tools we include a brief desription of the purpose of the tool and instructions on how to use it in a very basic way. Examples are given in Part II and the algorithm descriptions are given in Part III. All of the tools are designed to operate on image files, so if a 1-D input is desired, the user should be familiar with the DataModel filtering schemes.

1.2 Organization of this Guide

The **TCD** User Guide is divided into four parts:
I - This introduction
II - a cookbook with examples
III - the theory section describing the algorithms, and
IV - the reference section which describes the details on input parameters and data products.

1.3 Input file considerations

All the **TCD** tools only know how to deal with images. They can have any number of dimensions, but they still must be images. The user is advised to consult the DataModel documentation to ensure that the corrent syntax is used.

1.4 *aconvolve*

This is a simple smoothing tool with options for a Gaussian function, a tophat, a user defined kernel, or a user supplied map.

1.4.1 Parameters to set

`infile`

This should be an image; either in a FITS file or an iraf '.imh' file.

`kernelspec`

The generic syntax for `kernelspec` is:

key:parameters:origin

where 'key' is a filename, 'txt', or 'lib'. If the filename option is chosen, that file will be the convolving function. The 'txt' option allows an ASCII description of a function, and 'lib' allows a choice among 'box', 'gaus', and 'tophat' (from the library). The parameter specification will be described elsewhere (see section 8.1.2) and 'origin' is used to describe the placement of the convolving function. For a simple Gaussian smooth, we use lib:gaus(2,8,1,2.12,2.12); the leading '2' is for two dimensions, the '8' specifies how large to make the convolving function (in units of σ), the '1' is the normalization, and the last two values are the σ of the Gaussian in pixels.

`method`

There are two values permitted for this parameter: 'slide' and 'fft'. The default is 'slide' which is a brute force convolution in the map plane and the latter is a Fast Fourier Transform implementation. If the data array is small (less than 512) you can safely use 'slide'. Choosing 'fft' results in faster running, but with limited memory available, a segmentation violation may occur.

`outfile`

This is the filename for the output.

1.4.2 Running the task

For an initial test run, select an image that is not too large; say 512x512 or less. Check your setup, enter the necessary parameters, check the parameter file, and run the tool.

```
unix: which aconvolve
/home/ascds/DS.release/bin/aconvolve
unix: pset aconvolve infile=hrc_b8_250.fits
unix: pset aconvolve kernelspec= "lib:gaus(2,8,1,2.12,2.12)"
unix: pset aconvolve outfile=hrc_smo8.fits
unix: plist aconvolve
unix: aconvolve
```

The output image is a smoothed array. Note that the convolved image does not conserve counts, even when the normalization of the kernel is set to one.

1.5 *acrosscorr*

acrosscorr is a simple tool to perform an autocorrelation of an image, or to cross correlate two images.

1.5.1 Parameters to set

`infile1`

The filename of the first input image.

`infile2`

The filename of the second input image. If an autocorrelation of `infile1` is desired, `infile2` should be set to 'none'.

`outfile`

The name to assign to the output file.

`center`

Setting `center` to 'yes' will cause the zero-offset point to be in the center of the output image.

1.5.2 Running the task

Select an image, check your setup, enter the necessary parameters, check the parameter file, and run the tool.

```
unix: which acrosscorr
/proj/cm3/Release/install.R4CU3/bin/acrosscorr
unix: pset acrosscorr infile1="../120Bdew256.fits"
unix: pset acrosscorr infile2="../120Bdew256_10_smo.fits"
unix: pset acrosscorr outfile="cross120Bdew0_10.fits"
unix: pset acrosscorr center=yes
unix: plist acrosscorr
unix: acrosscorr
```

In this simple example, we cross correlated an image of 3C 120 containing counts with a smoothed version of the same image. Since the smoothing function was larger than the source structure, the cross corellation image reflected the size of the smoothing function and was circularly symetrical to first order.

1.6 *apowerspectrum*

The power spectrum of a function or distribution is the square of the amplitude of the Fourier transform. The current implementation employs a FFT and is designed to work on multi-dimensional images.

1.6.1 Parameters to set

`infilereal`

This is the filename of the real component of the input data.

`infileimag`

This is the filename of the imaginary component of the input data. For normal data, this entry should be 'none'.

`outfile`

The desired name of the output file.

1.6.2 Running the task

The tool is designed to run on images of arbitrary size, but for test runs, we suggest choosing an image that is 512x512 or less. Check your setup, enter the necessary parameters, check the parameter file, and run the tool.

```
unix: which apowerspectrum
/home/ascds/DS.release/bin/apowerspectrum
unix: pset apowerspectrum infilereal=120Bdew256.fits
unix: pset apowerspectrum infileimag=none
unix: pset apowerspectrum outfile=pow120.fits
unix: plist apowerspectrum
unix: apowerspectrum
```

The output map displays the amplitude of spatial frequencies which, if there is a dc signal (i.e. one or more strong sources), will be largest close to the origin (and aliased to other corners of the array).

1.7 *atransform* NOT CURRENTLY RELEASED

Eventually, this task will contain several different types of transforms. Currently, the only implementation is the fast Fourier transform.

1.7.1 Parameters to set

`infilereal, infileimag`

Normally, when doing a forward transform, the real input will be the data map and the imaginary part will be "none" (the default value). However, when reversing the direction, both parts must be specified.

`outfilereal, outfileimag`

Both of these names should be specified.

1.7.2 Running the task

Select an image, check your setup, set the filenames, check your parameter file, and run the program.

```
unix: which atransform
/home/ascds/DS.release/bin/atransform
unix: pset atransform infilereal=4c41_b8_512_sm20.fits
unix: pset atransform outfilereal=atran_4c41sm.fits
unix: pset atransform outfileimag=atrani_4c41sm.fits
```

```
unix: plist atransform
unix: atransform
```

The output image shows the non-vanishing amplitudes of spatial frequencies clustered around zero (and, from aliasing, the other corners of the image).

1.8 *csmooth*

csmooth is an implementation of an adaptive smoothing code ('ASMOOTH') conceived by Ebeling et al. (2000). Adaptive smoothing is a technique which adjusts the size of the smoothing kernal to match the local surface brightness (e.g. counts per pixel) so that some user defined significance level will be achieved throughout the image. Thus in regions of high brightness, the smoothing function is of a small scale; while at low brightness, the function automatically increases to larger scales.

At the present time, the code contains only two options for the smoothing kernal: a tophat or a circularly symetric Gaussian. The computation is done either by brute force ('in the map plane') or by the FFT method. The former is cpu intensive and users are warned to choose very small arrays initially (e.g. 64x64 or 128x128). The FFT method on the other hand is memory limited.

1.8.1 Parameters to set

There are a large number of parameters under the user's control to fine tune the operation. Many of these will be discussed in chapter 6 and section 7.5. Here we give a simple example using default values.

`infile`

This is the filename of the input image.

`outfile`

The name for the output file.

`outsigfile`

This is the name for the output map which has the significance for each output pixel.

`outsclfile`

The name for the output map which contains the smoothing scale for each pixel.

`conmeth`

This parameter can have one of two values: 'fft' or 'slide'. The 'slide' option operates in the map plane and take a long time to run unless the arrays are very small. The edges are dealt with by renormalizing the the convolving function as it runs off the edge of the map to compensate for the reduced area. The 'fft' option is much faster, but the edges are wrapped and thus artifacts are often present near edges.

1.8.2 Running the task

Select an image, check your setup, enter the necessary parameters, check the parameter file, and run the tool.

```
unix: which csmooth
/proj/cm3/Release/install.R4CU3/bin/csmooth
unix: pset csmooth infile="../acis_128_pts.fits"
unix: pset csmooth outfile="cs_acis_128_pts.fits"
unix: pset csmooth outsigfile="csig_acis_128_pts.fits"
unix: pset csmooth outsclfile="cscl_acis_128_pts.fits"
unix: plist csmooth
unix: csmooth
```

This run on a 128 image took 39 minutes on a SUN SPARCstaion 5. If we had not used the FFT method, it would have been considerably longer.

The low levels of the smoothed result show some unexpected asymetries. The significance map shows evidence of aliasing and computational effects at low levels.

Part II

TCD Cookbook

Chapter 2

Convolve

2.1 Description

aconvolve is an open-iraf C program and is used to convolve an N-dimensional image with a kernel. The kernel can be specified as either a file (image), a function, 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. kernel could be larger than image).

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.

The input parameter file controls the type of convolution. The following convolution methods are implemented:

- Sliding Cell

- FFT convolve

The 'sliding cell' is a 'brute-force' method which operates in the map plane whereas the FFT is just the usual Fourier Transform method. The former requires more cpu time for larger arrays.

2.2 Parameters

2.2.1 Input and output specifiers

`infile, outfile`

The input is commonly a FITS image but other formats supported by the DataModel such as IRAF .imh files are also possible. The input image 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 output file is a FITS image which has the same dimensions as the input image. The output data type is always "float" (BITPIX=-32).

`kernelspec`

The convolving function (a.k.a. 'kernel') can be defined in three ways: as a call to the library functions, as a user-provided file, or as an ASCII descriptor. These options, each with its own parameters, are selected via one input parameter, `kernelspec`. The generalized syntax is:

```
<key>:<parameters>:<origin>
```

Where 'key' can take on one of the three values: 'file', 'txt', or 'lib'. The possibilities for 'parameters' are determined by which 'key' is chosen, and the 'origin' specification is often not necessary, but can be useful for particular choices. A more extensive description of `kernelspec` is given in section 8.1.2.

`writekernel, kernelfile`

If the user wants to obtain a file with the kernel used (obviously not necessary if key=file), `writekernel` should be set to 'yes' and a filename should be supplied for `kernelfile`. The default value for `writekernel` is 'no'.

`writefft, fftroot, center`

If files of the Fourier Transform are to be preserved, `writefft` should be set to 'yes' and a root filename should be provided (`fftroot`). This option will provide the FFT of both the input data and the kernel. The default for `writefft` is 'no'. The parameter `center` is currently inoperative.

2.2.2 Processing parameters

`edges, const`

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. Four methods for handling the edges are implemented: For the 4 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 go ...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 go ...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 go ...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) therefore the data go ...0,0,1,2,3,0,0,..

Implicitly, using the FFT method of convolution implies wrapping edge treatment.

`method`

There are two options for the calculation: 'slide' and 'fft'. If the user is satisfied with `edges`=wrap, the fft option is preferred since it is faster and will produce the same results.

Sliding cell convolution is 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 buy summation signs]
```

For the FFT method, the Fourier Transform of the data and the kernel is computed. The arrays are multiplied and the inverse FFT is taken. Internally the kernel and data MUST be the same size. The program will pad the smaller with 0's. If the edges of the data are "wrapped" (see below) then the FFT and slide methods will yield the same results.

`pad`

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.

2.3 Side effects and Restrictions

All computations are done in single floating point precision.

If the program fails for some reason, it will exit and return with a status not equal to zero.

For the FFT method, if arrays are too large for the memory available, a segmentation fault or ERROR message about memory allocation may occur.

The current version of *aconvolve* does not conserve counts even when the normalization of the kernel is set to one.

2.4 Example 1

We wish to smooth a *Chandra* simulated image which is a 512x512 primary FITS image as seen in figure 2.1.

We check the setup, set the parameters, view the parameter file and run the program.

```
unix: which aconvolve
/proj/cm3/Release/install.R4CU3/bin/aconvolve
unix: pset aconvolve infile ="../hrci512ext.fits"
unix: pset aconvolve kernelspec ="lib:gaus(2,5,1,10,10)"
unix: pset aconvolve outfil=hrci512ext_10.fits
unix: plist aconvolve
unix: aconvolve
```

This run took 5 hours on a SPARCstation 5. If the fft option had been chosen, it would have been much faster. In this example we chose a Gaussian kernel from the library. The 5 numerical parameters are '2' for the number of dimensions, '5' for the size of the kernel in units of σ, '1' for the amplitude of the kernel, and a '10' for the x and y values of σ for the kernel (in pixels).

The results are shown in figure 2.2.

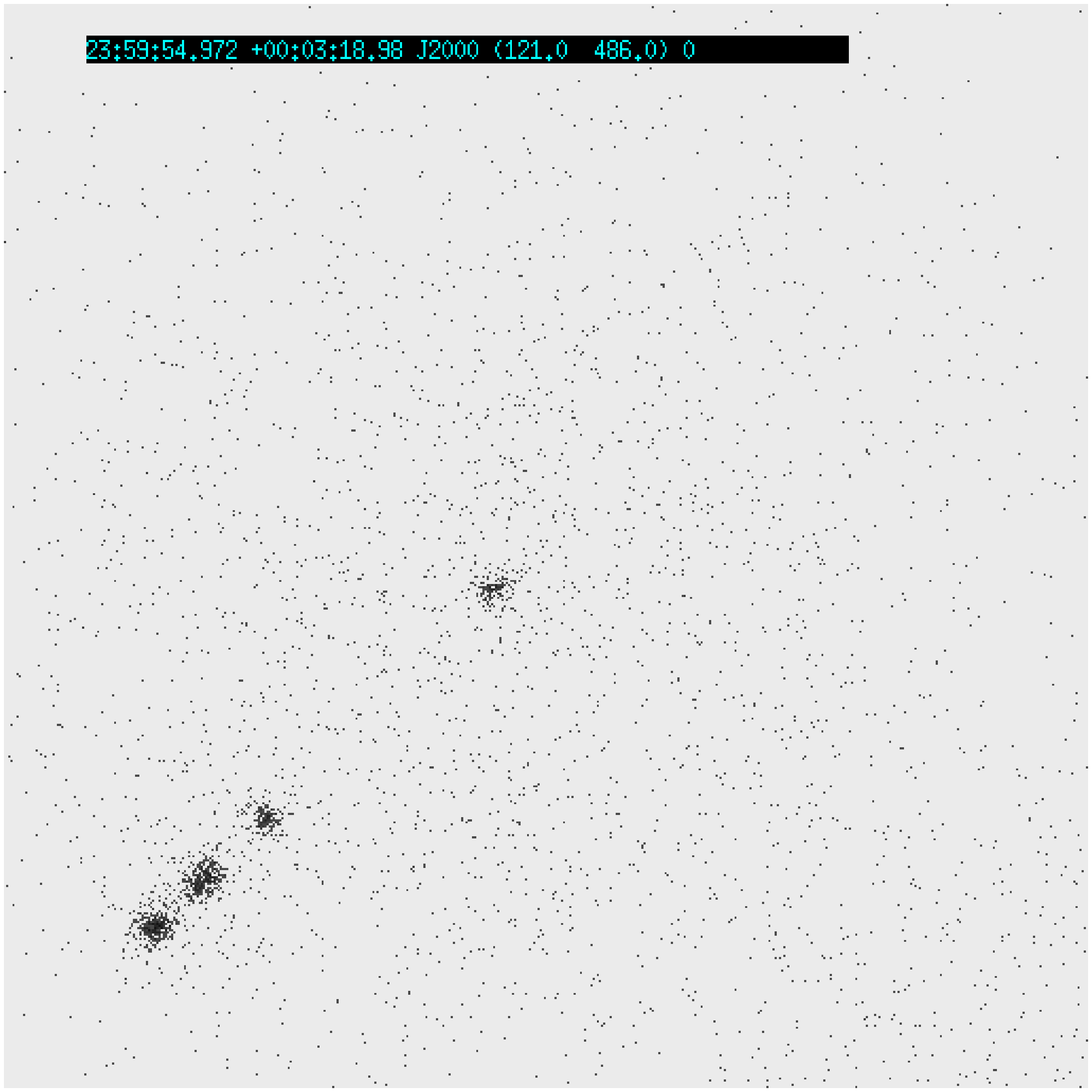

Figure 2.1: An HRC simulation with 6 point sources in a line plus an extended Gaussian source with a σ of 25 arcsec.

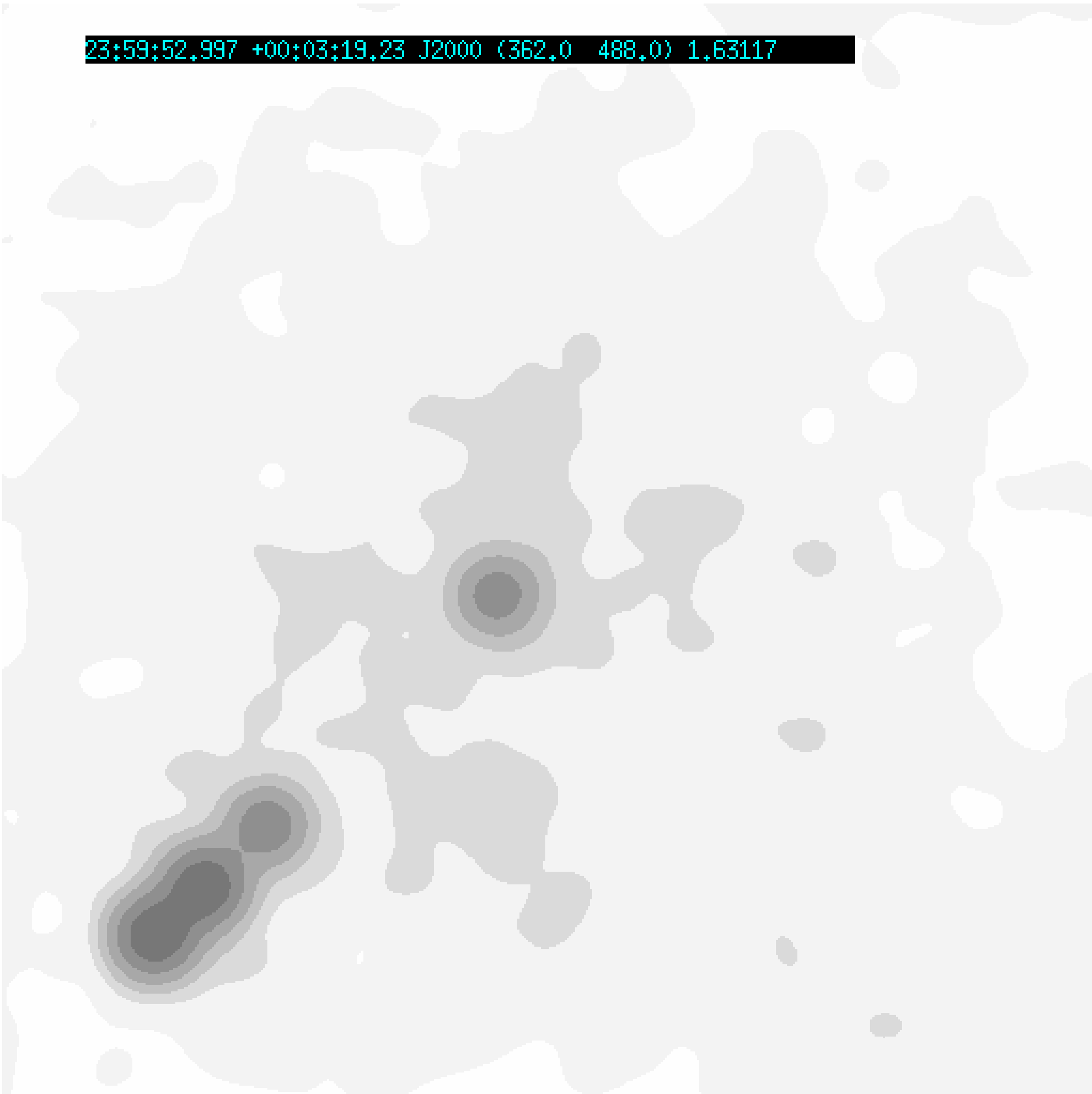

Figure 2.2: The results of running *aconvolve* with a Gaussian kernel with σ=10 pixels on the HRC simulation.

Chapter 3

Crosscorrelate

3.1 Description

The tool *acrosscorr* performs a cross correlation or an auto-correlation using Fourier transforms. For a cross correlation, the two input images should have the same number of dimensions.

3.2 Parameters

There are very few parameters for this simple tool.

`infile1, infile2, and outfile`

For a cross correlation, provide the appropriate names for the two input images. If `infile2` is set to 'none', this is the switch which produces an auto-correlation instead of a cross correlation.

`crop, pad, and center`

These 3 boolean parameters (values of 'yes' or 'no') provide the only choices for this tool. If `crop` is set to 'yes', the output image is cropped to the size of `infile1`. If `pad` is set to 'yes', the input data are padded to the size of `infile1` + `infile2`. If `center` is set to 'yes', then the zero shift point appears in the center of the output map rather than in the corner (at pixel 1,1).

3.2.1 Example: Cross correlation

In figure 3.1 we show the two input maps. The first consists of 6 sources in a line, The first two are separated by only 0.5 arcsec so appear as a single source with the HRC. The next two are separated by one arcsec, and produce a clearly extended distribution. The second image contains the same 6 sources but has in addition

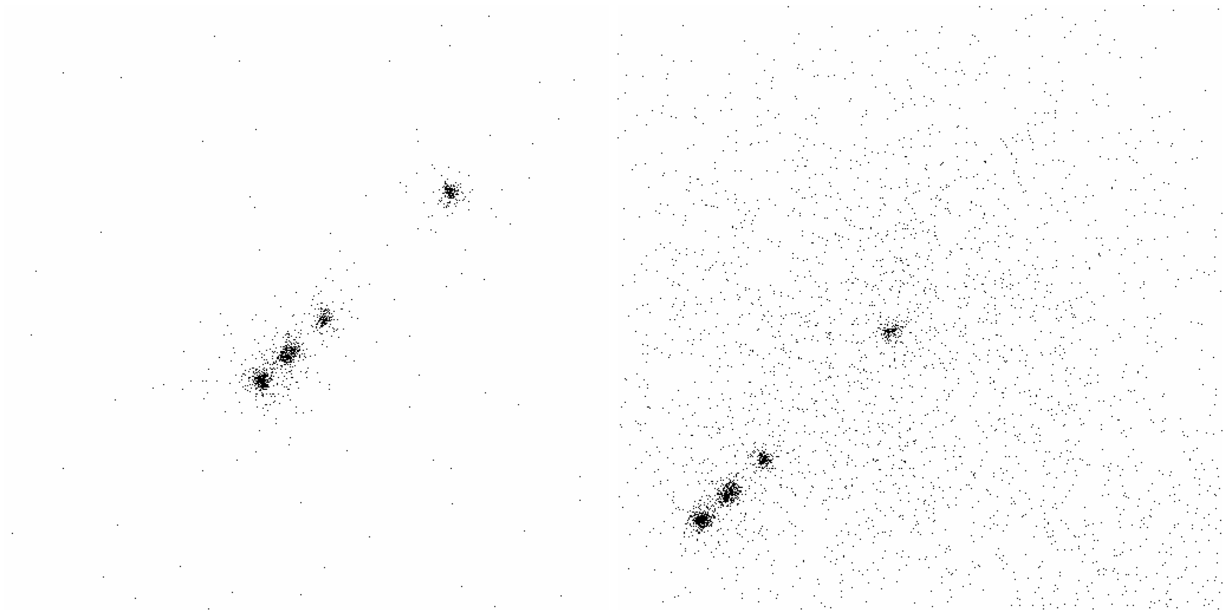

Figure 3.1: Simulated HRC data. The first field has 6 unresolved sources in a line. The other field shows the second input image which has the same sources but in addition, has an extended source.

an extended source centered at the same position as the sixth unresolved source. As can be seen, the lines of sources are not at the same pixel location in the images.

We check the setup, set the relevant parameters, review the parameter list, and run the program.

```
unix: which acrosscorr
/proj/cm3/Release/install.R4CU3/bin/acrosscorr
unix: pset acrosscorr infile1="../hrci512pts.fits"
unix: pset acrosscorr infile2="../hrci512ext.fits"
unix: pset acrosscorr outfile=cross_hrc512pts_ext.fits
unix: pset acrosscorr center=yes
unix: plist acrosscorr

Parameters for /home/user/ascds_iraf/uparm/acrosscorr.par

#
# acrosscorr.par file
#
# inputs
#
       infile1 = ../hrci512pts.fits Input file name #1
       infile2 = ../hrci512ext.fits Input file name #2. Use none for
# autocorrelate
#
# output
#
       outfile = cross_hrc512pts_ext.fits Output file name
```

```
#
# processing params
#
        (crop = no)             Crop output to size of infile1
         (pad = no)             Pad data to size of infile1 +
# infile2
      (center = yes)            Center output
#
# user
#
     (clobber = yes)            Clobber existing output file
     (verbose = 0)              Debug level
      (kernel = default)        Output format kernel
#
# mode
#
        (mode = ql)

unix: acrosscorr
Input file name #1 (../hrci512pts.fits):
Input file name #2. Use none for autocorrelate (../hrci512ext.fits):
Output file name (cross_hrc512pts_ext.fits):
```

The resulting map is shown in figure 3.2.

3.3 Example: Auto-correlation

We take the same images shown in figure 3.1 to illustrate the auto-correlation simply by setting `infile2` = none. The results for the field containing the extended emission are shown in figure 3.3. A similar run on the other field (without the additional extended source) produces essentially the same pattern, but the absolute value of the correlation peak is lower and, without the extended source, the lower levels (away from the main line of correlation features) are a smaller percentage of the peak value.

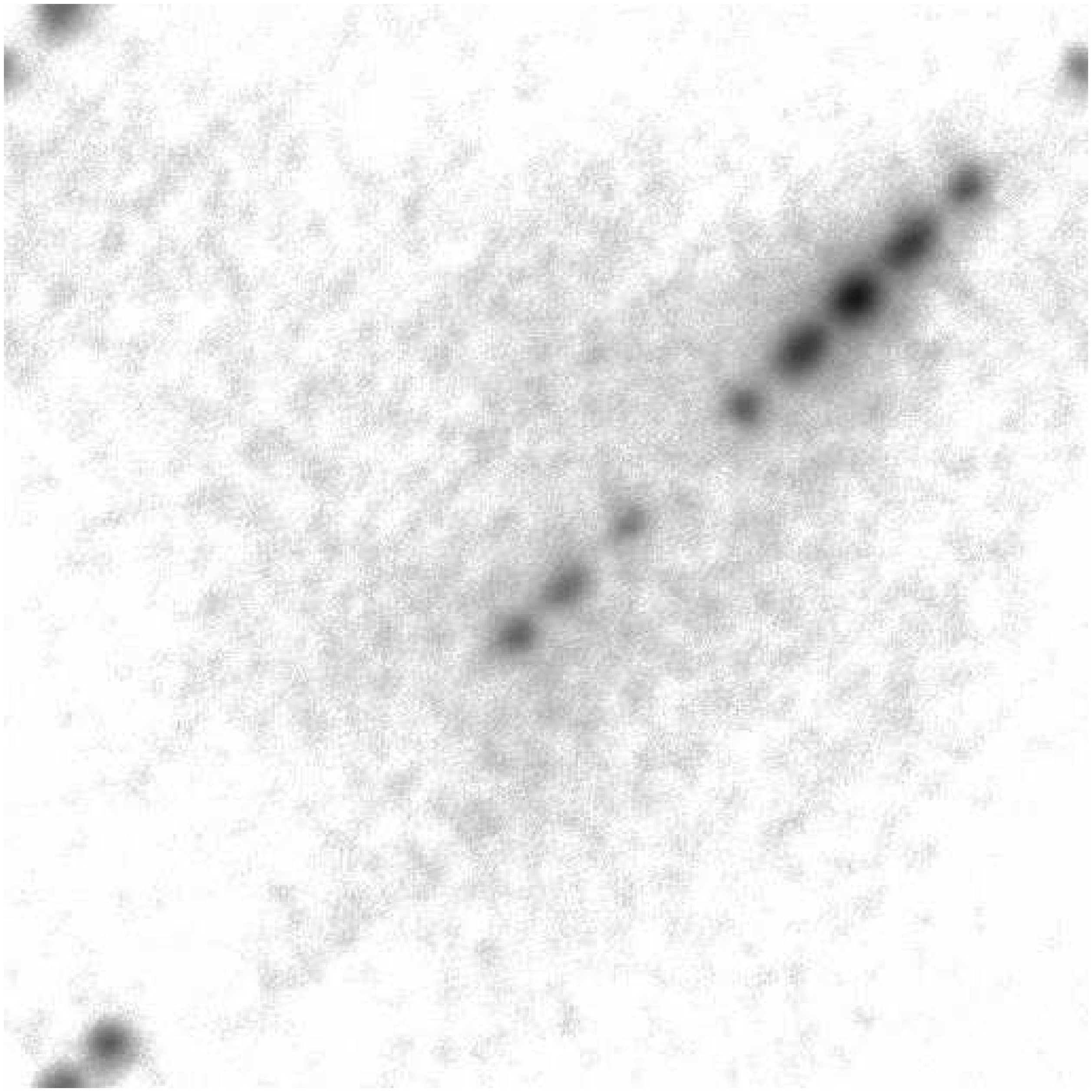

Figure 3.2: The results of *acrosscorr* run on the two fields shown in fig. 3.1. Since `center` was set to 'yes', the maximum correlation occurs towards the upper right at a distance from the center corresponding to the shift necessary so that the array of 6 unresolved sources in map 2 are aligned with their counterparts in map 1.

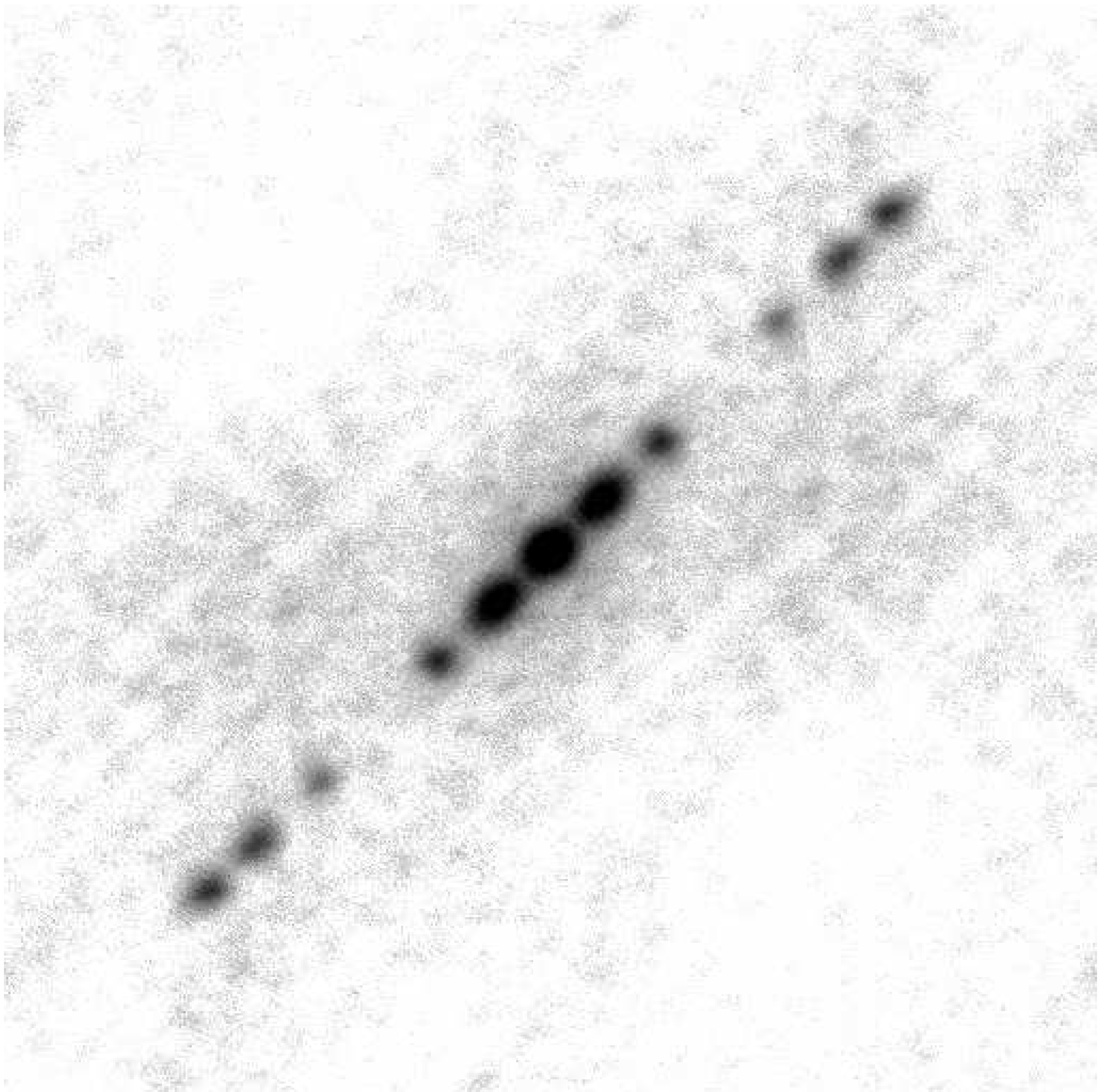

Figure 3.3: The auto-correlation image for the HRC example of 6 unresolved sources plus an extended source. Since `center` was set to 'yes', the peak occurs at the map center.

Chapter 4

Powerspectrum

4.1 Description

apowerspectrum uses an FFT on an N-dimensional image and produces the square of the amplitudes for the frequency components of the transform.

4.2 Parameters

`infilereal, infileimag, outfile`

These filenames are required, although 'none' is the default for `infileimag`, the imaginary part of the input. The usual DataModel conventions are followed for input and output.

`pad`

This is boolean parameter. If set to 'yes', then the input data will be padded in each dimension to attain the next power of two.

`center`

Setting `center` to 'yes' causes the zero frequency point to be at the center of the output array.

`scale`

This parameter simply determines the absolute value of the output map. The choices are 'linear' (which is the default) meaning simply the square of the amplitude of the frequency components; 'log' which gives the logorithm to the base ten; and 'db' ('decibels') which gives 10 times the log value.

`crop`

If `crop` is set to 'yes', only frequencies up to the Nyquist frequency will be output.

4.3 Example

To examine the distribution of spatial frequencies in the maps used in chapters 2 and 3, we make a power spectrum for the array of 6 point sources, the same plus the extended source; and the smoothed version of the latter. These input maps are shown in figures 3.1 and 2.2.

One of these runs was performed thusly:

```
unix: which apowerspectrum
/proj/cm3/Release/install.R4CU3/bin/apowerspectrum
unix: pset apowerspectrum infilereal=''../hrci512ext.fits''
unix: pset apowerspectrum outfile=''pow_hrci512ext.fits''
unix: pset apowerspectrum center=yes
unix: plist apowerspectrum

Parameters for /home/user/ascds_iraf/uparm/apowerspectrum.par

#
# apowerspectrum tool
#
# inputs
#
    infilereal = ../hrci512ext.fits Input file name for real part
    infileimag = none             Input file name for imaginary part
#
# output
#
       outfile = pow_hrci512ext.fits File name for output
#
# processing
#
          (pad = no)              Pad data array to next power of 2
       (center = yes)             Center 0 frequency at center of
# array
        (scale = linear)          Output scale
         (crop = no)              Crop output at Nyquist frequency
#
# user
#
      (clobber = yes)             Delete existing output
      (verbose = 0)               Debug level
       (kernel = default)         Output format kernel
#
# mode
#
         (mode = ql)
```

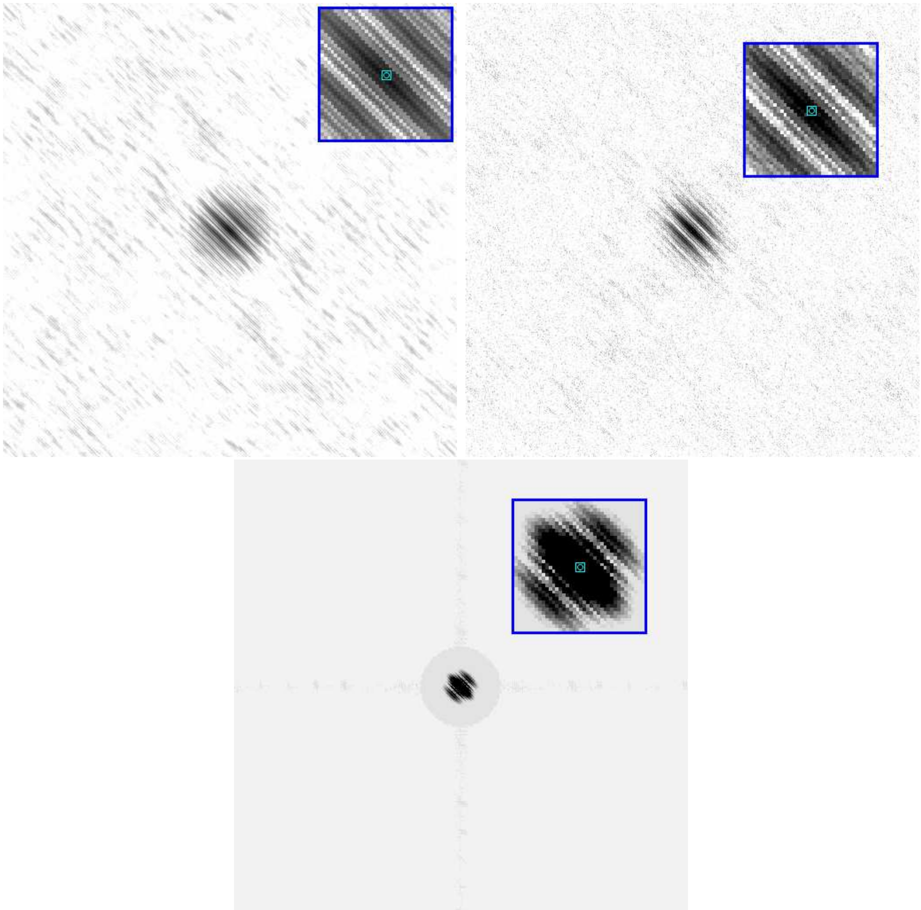

Figure 4.1: Power spectra of HRC simulations. The first map is for the line of 6 unresolved sources. The second is for the same sources but with an extended source added. The last map is the same as the second except it has been smoothed with a Gaussian of σ=10 pixels. The peak amplitudes are 5.82, 52.6, and 2×10^7, respectively. The noise level in the first two maps is $\approx$0.1, but in the last it is $<10^{-11}$. Inserts show the central region in more detail.

```
unix: apowerspectrum
Input file name for real part (../hrci512ext.fits):
Input file name for imaginary part (none):
File name for output (pow_hrci512ext.fits):
```

The resulting maps are shown in figure 4.1. Since we set `center`='yes', the non-zero amplitudes of spatial frequencies appears at the center of the maps. The basic spatial frequency structure of an individual source is represented by the distribution in PA=45°. In PA -45° there is more structure caused by enhancement of those frequencies which correspond to the source placement along the line. The power spectrum of the smoothed map is essentially noise free and the the size of the region containing significant amplitudes is significantly reduced.

Chapter 5

Transform - NOT CURRENTLY RELEASED

5.1 Description

Transform is an open-iraf C program and is used to compute various mathemtical transforms of an input data array.

The available transforms are:

- Fourier Transform

Additional functions will be added in the future. The transform and direction parameters control the type of transform.

5.2 Parameters

5.2.1 Filenames

`infilereal, infileimag`

The input files are FITS images or IRAF .imh files. The input images can have any number of dimensions.

The input can have the following data type: "btye" (BITPIX=8), "short" (BITPIX=16), "long" (BITPIX=32), "float" (BITPIX=-32), and "double" (BITPIX=-64).

The real and imaginary parts are input seperately. If there is no real or imaginary part, then set the appropriate file name to "none" or "NONE".

Currently, there MUST be a real part to the data. The imaginary part can be "none".

`outfilereal, outfileimag`

Two output data files are produced. They are both FITS images; the data type is always "float" (BITPIX=-32). There is one file for the real part of the transform and one for the imaginary part of the transform.

5.2.2 Processing parameters

`transform`

At this time the only allowed value is 'fft', and this is the default value.

`direction`

This parameter determines the sense of the transform. The two allowed values are 'forward' and 'reverse'. The default value is 'forward'.

`pad`

This parameter can either be 'no' (the default) or 'yes'. The latter value will cause each data axis to be padded with zeros (on the 'right hand' side) so that the dimension of each axis is a power of two.

`center`

The default for `center` is 'no', and this causes the zero frequency point to be located at pixel zero. Setting `center` to 'yes' causes the zero frequency location to be at the center of the output array.

5.3 Syntax

Compute the N-D FFT of the data in the file my_data.fits and save the real + imagainary parts in my_out_xxxx.fits.

```
transform infilereal=mydata.fits infileimag=none
outfilereal=myoutreal.fits outfileimag=myoutimag.fits
transform=fft direction=forward
```

5.4 FFT

The Fast Fourier Transform (FFT) (invoked by setting `transform` = 'fft') computes the discrete Fourier Transform of a dataset with respect to one or more axes.

By standard convention, in the forward direction the sign of the complex exponential is negative and in the reverse direction the sign of the complex exponential is positive. In the forward direction, the data are normalized by the area of the data.

The FFT algorithm was adapted from the STSDAS FFT routine converted from FORTRAN to C and made to work in multiple dimensions.

5.4.1 Padding

Although the algorithm works on datasets of any length, to sped up the process 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.

5.4.2 Sideeffects and Restrictions

All computations are done in single floating point precision.

If the program fails for some reason, it will exit and return with a status not equal to zero.

5.4.3 Example 1

Suppose we wish to determine the spatial frequencies of the ACIS simulation shown in figure 6.1.

We set the relevent parameters, check the complete parameter file, and run the program:

```
unix: which atransform
/proj/cm3/Release/install.R4CU3/bin/atransform
unix: pset atransform infilereal="../Smooth/acis_128_extmap.fits"
unix: pset atransform infilereal="../Smooth/acis_128_extmap.fits"
unix: pset atransform outfilereal="tr_acis128ext.fits"
unix: pset atransform outfileimag="ti_acis128ext.fits"
unix: plist atransform

Parameters for /home/user/ascds_iraf/uparm/atransform.par

#
# atransform.par file
#
# inputs
    infilereal = ../Smooth/acis_128_extmap.fits Input file name for
# real part
    infileimag = none             Input file name for imaginary part
#
# outputs
#
```

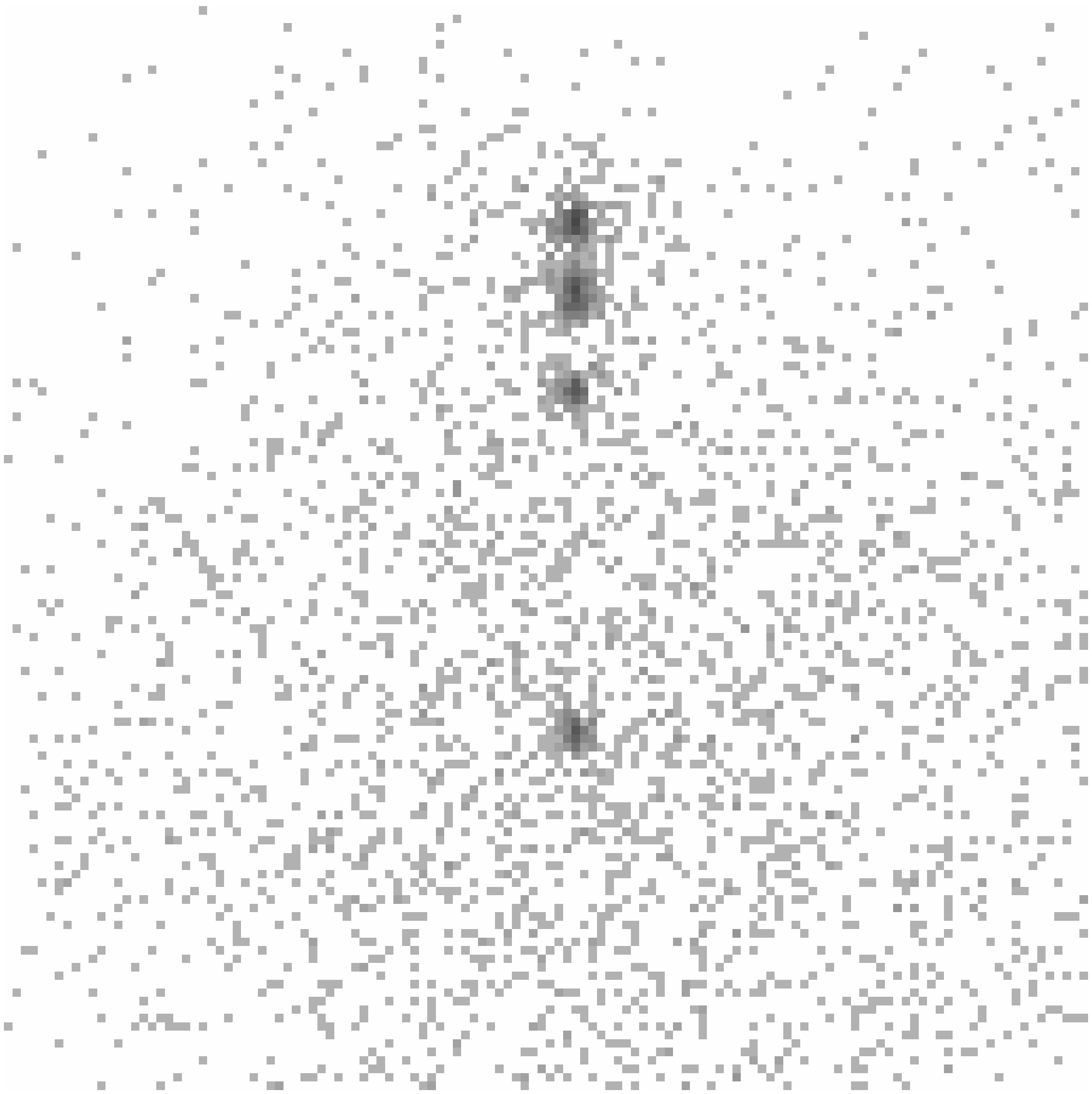

Figure 5.1: A 128x128 ACIS simulation containing 6 unresolved sources in a line plus an extended source. Only 4 discrete sources are visible since the close separation for two pairs is not resolvable by ACIS.

```
   outfilereal = tr_acis128ext.fits File name for real part of output
   outfileimag = ti_acis128ext.fits File name for imaginary part of
# output
#
# processing parameters
#
     transform = fft              Transform type
     direction = forward          Transform direction
          (pad = no)              Pad data array to next power of 2
       (center = no)              Center 0 frequency at center of
# array
#
# user preferences
#
      (clobber = yes)             Delete existing output
      (verbose = 0)               Debug level
       (kernel = default)         Output format kernel
#
# mode
#
         (mode = ql)

unix: atransform
Input file name for real part (../Smooth/acis_128_extmap.fits):
Input file name for imaginary part (none):
File name for real part of output (tr_acis128ext.fits):
File name for imaginary part of output (ti_acis128ext.fits):
Transform type (fft) (fft):
Transform direction (forward|reverse) (forward):
```

NOTE: we cannot finish this example until the center option is repaired. deh 11 July 1999.

Chapter 6

Csmooth

6.1 Description

csmooth is a tool to perform adaptive smoothing. If normal smoothing is required, the user should use *aconvolve* (chapter 2). The basic premise of adaptive smoothing attempts to adapt the size of the smoothing kernel to the local s/n ratio. Thus in regions containing strong point sources, a small kernel is used preserving the inherent resolution of the data. For lower brightness regions (e.g. extended sources or the background) the size of the kernel increases, usually attempting to include enough counts so as to achieve a specified s/n ratio. In this chapter we give a simple example. Users are urged to explore the potential of the tool by experimenting with adjustment of hidden parameters (see section 8.5.2 or the help file for parameter description).

This version of *csmooth* is based on H. Ebeling's IDL code called 'ASMOOTH'. The code is now in C and modifications have been made to permit the use of FFTs which are much faster but have the disadvantage that they may introduce artifacts because the only possible edge treatment is wrapping. The sliding cell convolution provides an option to use a renormalization routine for dealing with image edges which compensates for that fraction of the kernel which is outside the image, thus ensuring that no artifacts are introduced at the field edges.

6.1.1 Parameters

Here we consider the parameters required to run a simple example. For a complete description of all parameters see section 8.5.2.

`infile`

This is the designation of the 2D image for the input.

`outfile, outsigfile, outsclfile`

The result of running *csmooth* is `outfile`. `outsigfile` is a map containing the significance of each pixel in `outfile`. The scale size of the kernel used at each pixel is given in `outsclfile`.

`conmeth`

This parameter can have the value 'fft' (the default) or 'slide'. In the former case the FFT method is used and in the latter case, a brute force implementation is used. Note that designating 'fft' means that the data are wrapped around to pad the edges so as to accommodate the kernel as it moves toward the edge of the original data. For the 'slide' method, the edges are padded with the particular constant=0, but the kernel is renormalized to compensate for the smaller area of the actual (non-zero) data.

The FFT method requires substantial memory resources whereas the slide method requires a longer run time. For these reasons, we suggest that first time users choose a small array (e.g. 128x128) and, if FFT is chosen, to select an array with dimensions that are 2^N in size.

`conkerneltype`

The convolving function can be either 'gauss' or 'tophat'

6.2 Example

To illustrate *csmooth* we take a subimage from an ACIS simulation. This is a 128x128 map which has 6 unresolved sources in a line, and an extended source comprised of a Gaussian with $\sigma = 25\,''$ (see Figure 6.1).

We check the setup, set the parameters, examine the parameter file, and run the program.

```
unix: which csmooth
/soft/ciao/bin/csmooth
unix: pset csmooth infile="../acis_128_ext.fits"
unix: pset csmooth outfile="cs_acis_128_ext.fits"
unix: plist csmooth

Parameters for /home/user/cxcds_param/csmooth.par

#
# csmooth.par file
#
#
        infile = ../acis_128_ext.fits input file name (raw image)
       outfile = cs_acis_128_ext.fits output file name (adaptively smoothed image)
    outsigfile = .                output file name (image of the significance of the signal at each location of the smoothed image)
    outsclfile = .                output file name (image of the smoothing scales [kernel sizes] used at each location of the image)
#
# processing parameters
#
       conmeth = fft              Convolution method.
 conkerneltype = gauss            Convolution kernel type.
#
# Significance numbers
```

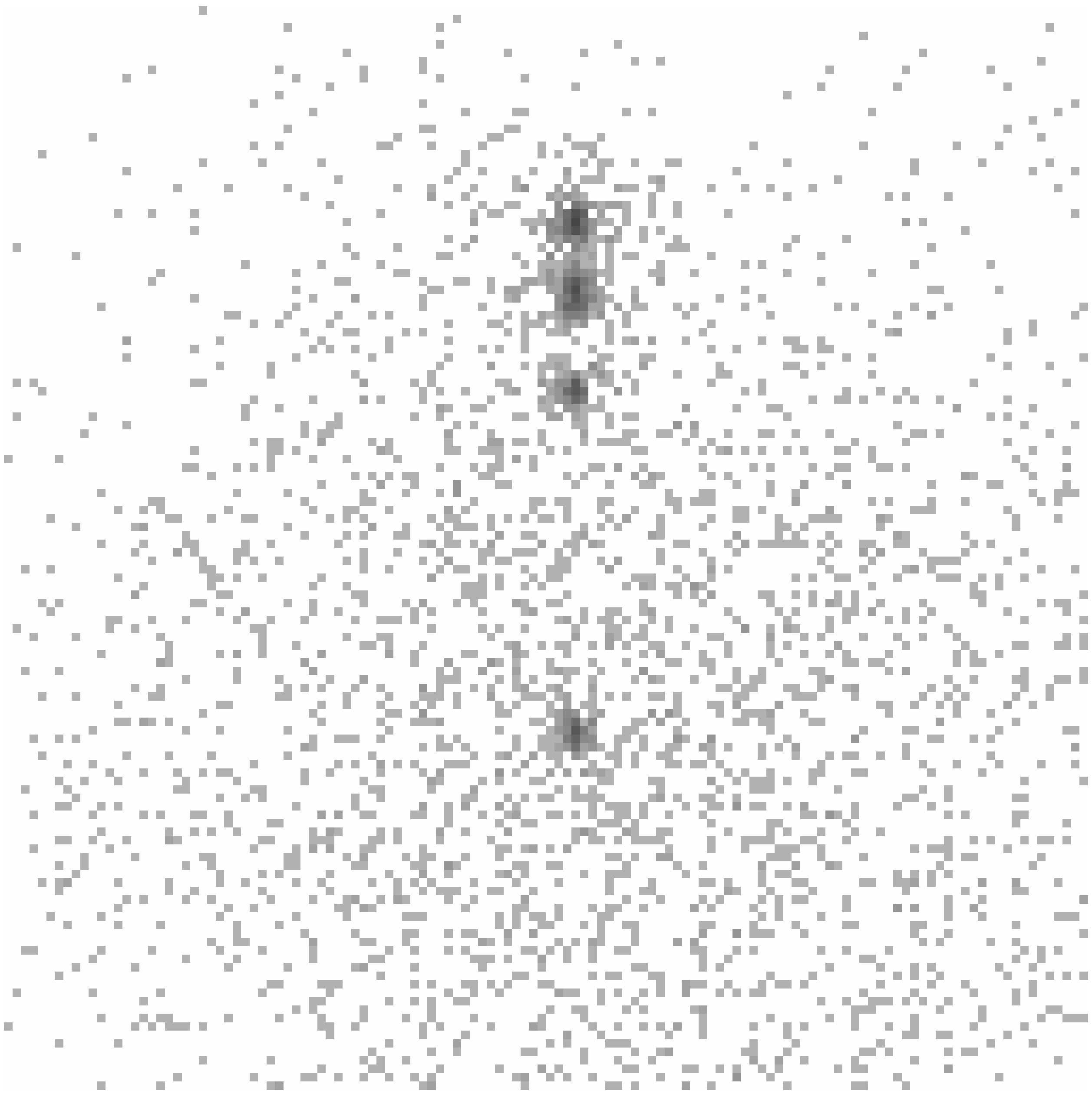

Figure 6.1: A 128x128 ACIS simulation containing 6 unresolved sources in a line plus an extended source. Only 4 discrete sources are visible since the close separation for two pairs is not resolvable by ACIS.

```
#
       sigmin = 4                minimal significance (S/N ratio) of the signal under the kernel
       sigmax = 5                maximal significance (S/N ratio) of the signal under the kernel
#
# Scales
#
       sclmin = INDEF            initial (minimal) smoothing scale in pixel, use INDEF for default (~1pixel)
       sclmax = INDEF            maximal smoothing scale, use INEF for default(~image size)
#
# User supplied scale map
#
      sclmode = compute          compute smoothing scales or user user-supplied map
       sclmap =                  input file name (image of user-supplied map of smoothing scales)
    (stepzero = 0.01)            initial stepsize by which smoothing scale increases
#
# background method
#
     (bkgmode = local)           background treatment
      (bkgmap = )                input file name (image of user-supplied background)
      (bkgerr = )                input file name (image of user-supplied background error)
#
# user specific comments
#
     (clobber = yes)             clobber existing output
     (verbose = 1)               verbosity of processing comments
      (kernel = default)         kernel of output format
        (mode = ql)


unix: csmooth
input file name (raw image) (../acis_128_ext.fits):
input file name (image of user-supplied map of smoothing scales) ():
output file name (adaptively smoothed image) (cs_acis_128_ext.fits):
output file name (image of the significance of the signal at each
location of the smoothed image) (.):
output file name (image of the smoothing scales [kernel sizes] used at
each location of the image) (.):
Convolution kernel type. (gauss|tophat) (gauss):
Convolution method. (slide|fft) (fft):
initial (minimal) smoothing scale in pixel, use INDEF for default
(~1pixel) (INDEF):
maximal smoothing scale, use INEF for default(~image size) ()sclmax)
(INDEF):
minimal significance (S/N ratio) of the signal under the kernel (4):
maximal significance (S/N ratio) of the signal under the kernel
()sigmin) (5):
compute smoothing scales or user user-supplied map (compute|user)
(compute):

Message: edge treatment options will be ignored
```

```
                  smoothing    out/in      pixels    counts    significance
 n_max  m_krnl_min  radius    diffl  cumul  done (%)  done (%)     range
                                                                 min  med   max
---------------------------------------------------------------------------------
 63.00     63.92     0.188    1.000  1.000    0.01      1.72    4.30  4.30  4.30
...(etc).....
  3.00    170.57    13.477    1.000  1.000   22.63     58.58    4.00   4.15  4.72
  2.00    204.53    14.631    1.000  1.000   25.25     61.15    4.00   4.30  5.01
  2.00    199.12    15.315    1.000  1.000   26.42     62.43    4.00   4.14  4.54
CSMOOTH: kernel cannot be larger than image
setting kernel size to size of image
CSMOOTH: remainder will be smoothed on scale of 15.736570
  3.00     64.21    15.737    1.000  1.000  100.00    100.00  -14.90   0.54  4.75
```

This run took 13 minutes on a SUN Ultra 1 and 37 minutes on a SparcStation5. The results are shown in figure 6.2. This rendition can be compared with the constant kernel implementation of *aconvolve*, see figure 2.2. (This is just an illustrative comparison since the pixel size of the HRC is smaller than that of ACIS.)

The scale map is shown in figure 6.3 and the significance map is shown in figure 6.4.

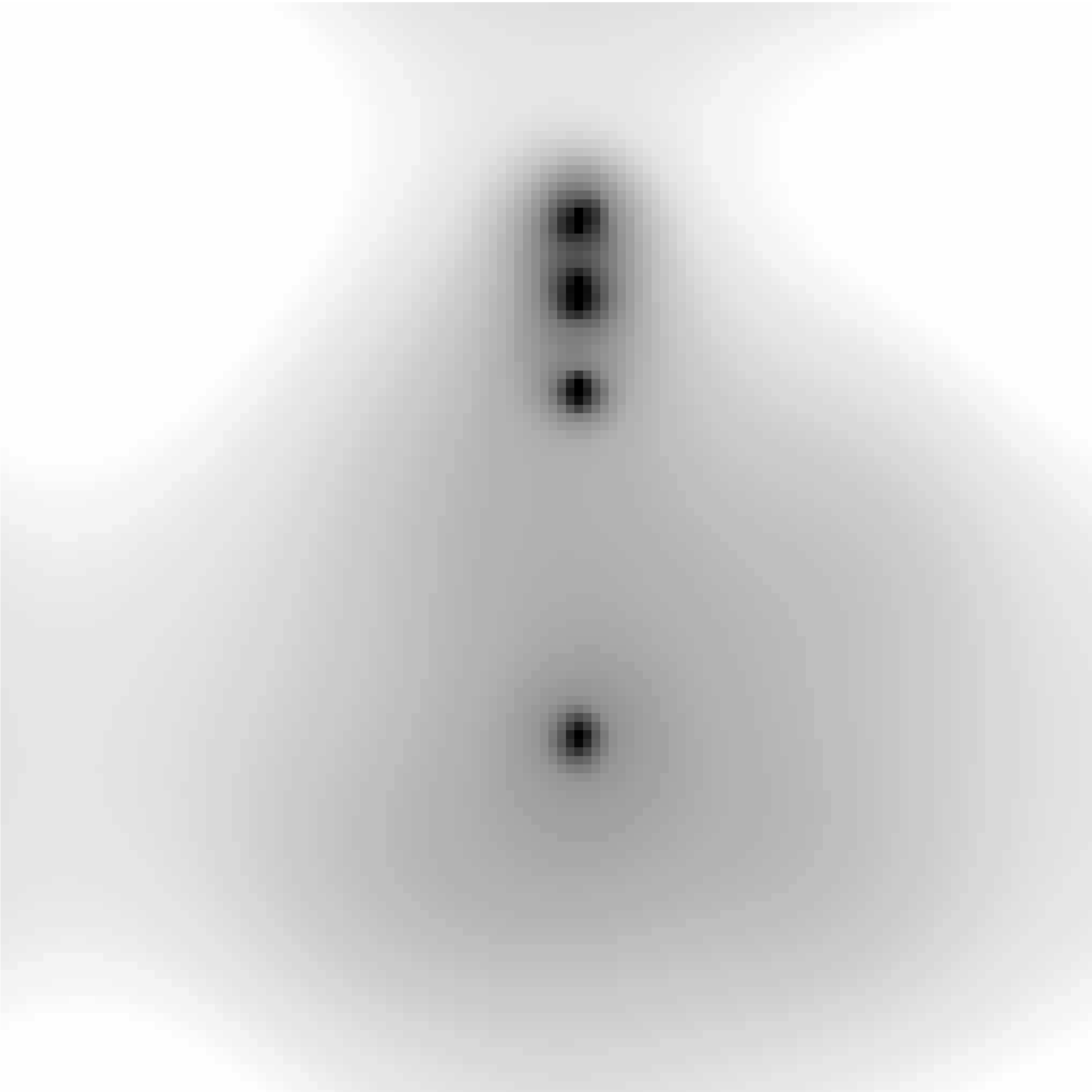

Figure 6.2: The results of running *csmooth* on an ACIS simulation.

Figure 6.3: This map shows the scale of the kernel used at each pixel location for the ACIS 128x128 simulation. In the white areas, the scale is 14 to 16 pixels whereas in the darkest area the scale is the minimum value of 0.188 pixels.

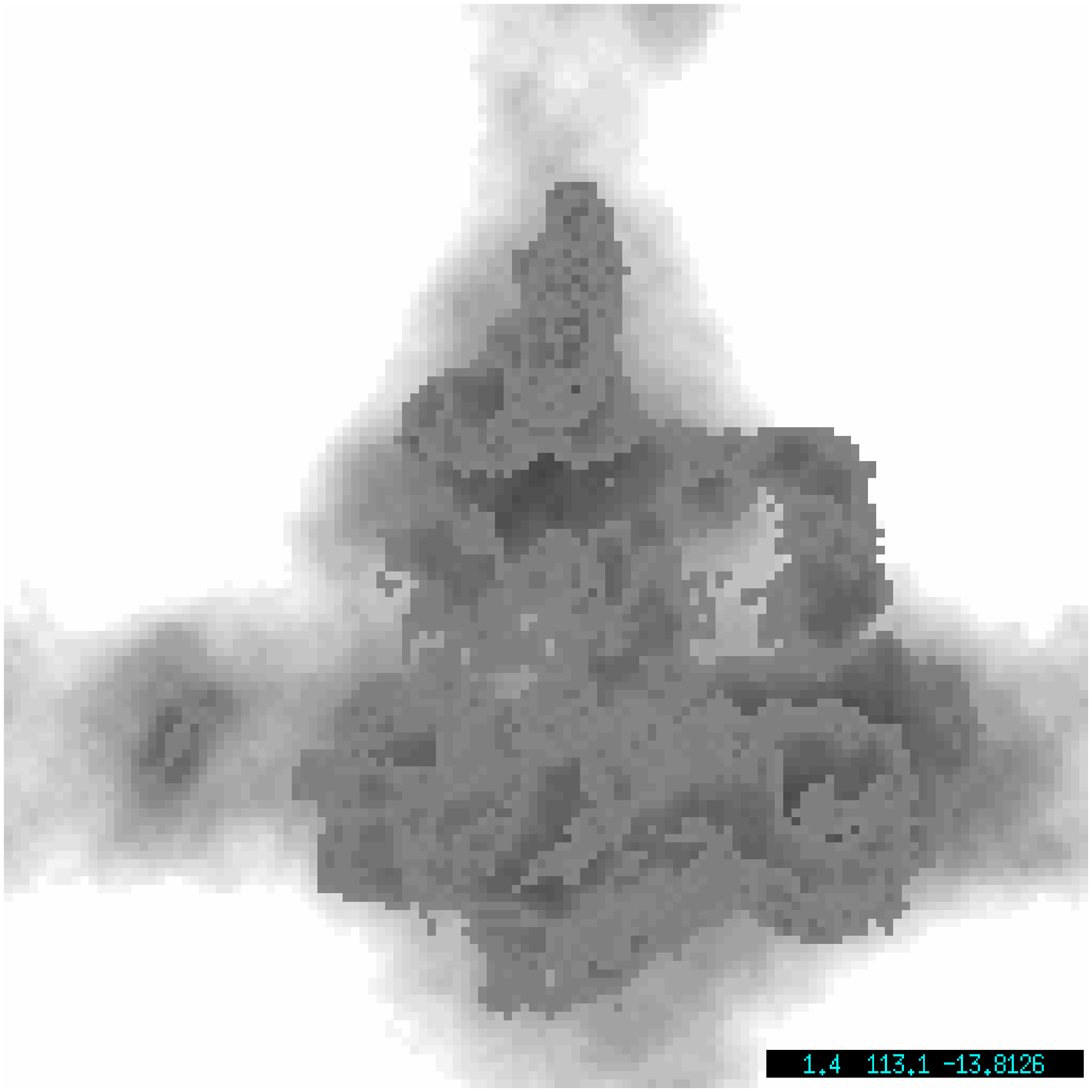

Figure 6.4: For each pixel in the smoothed map, this image shows the significance. Darker areas are higher; the white areas are negative. Since *csmooth* is attempting to keep the significance between 3 and 4, there is not much contrast in the significance map.

Part III

TCD Theory

Chapter 7

Theory

7.1 *aconvolve*

ASC Name: aconvolve

Desription: Calculate the convolution of one-D or two-D arrays using Slide cell and Fourier transform methods.

Algorithm:

In the following we give a simple example for a convolution of two 1-d arrays. We assume that the arrays ware originally shifted from the origin by a and b, respectively, and have been shifted back.

g(k) k=0,1,....,A-1

h(k) k=0,1,....,B-1

A and B are the number of samples in g and h, respectively.

First pad with zeros to a size N>= A+B -1, (if running FFT on power of 2 size arrays then N=2 γ (where γ is an integer value) then chose the convolution method:

- FFT Convolve
- Slide Cell Convolve

1. Slide Cell Convolve:

To calculate the convolution using the Slide Cell method, calculate the following sum:

$$z(k) = \sum_{n=0}^{N-1} [g(n)h(k-n)] \tag{7.1}$$

2. FFT Convolve:

To calculate the convolution using the FFT method:

- Calculate the FFT of g and h:

$$G(n) = \sum_{k=0}^{N} g(k)e^{-j2\pi nk/N} \tag{7.2}$$

$$H(n) = \sum_{k=0}^{N} h(k)e^{-j2\pi nk/N} \tag{7.3}$$

and then calculate:

$$Z(n) = G(n)H(n) \tag{7.4}$$

-Finally, calculate the convolution by calculating the inverse FFT of Z(n):

$$z(k) = \sum_{n=0}^{N-1} [N^{-1} Z^*(n)e^{-j2\pi nk/N} \tag{7.5}$$

Note: (see e.g. Brigham 1974).

7.2 Correlation

ASC Name: *acrosscorr*

Desription: Calculate the correlation of one-D or two-D arrays using Fourier transforms. (Special case - autocorrelation).

Parameters:

Algorithm:

The correlation of two finite-length functions g(x) and h(x) with a number of samples A and B, respectively, can be calculated as follows:

- Assume that g and h are shifted from the origin by a and b, respectively, and are defined as follows:

g(k)=0 k=0,1, N-A

g(k)=g(kT+a) k=N-A+1, N-A+2,..., N-1

h(k)=h(kT+b) k=0,1,....,B-1

h(x)=0 x=B,B+1,...., N-1

where N is larger or equal to the sum of A+B-1 and N=2 γ (where γ is an integer value)

- Calculate the discrete transforms of g and h:

$$G(n) = \sum_{k=0}^{N} g(k) e^{-j2\pi nk/N} \tag{7.6}$$

$$H(n) = \sum_{k=0}^{N} h(x) e^{-j2\pi nk/N} \tag{7.7}$$

- Change the sign in the imaginary part of H(n) to obtain $H^*(n)$ and then calculate:

$$Z(n) = G(n) H^*(n) \tag{7.8}$$

-Finally, calculate the correlation:

$$z(k) = \sum_{k=0}^{N-1} [N^{-1} Z^*(n) e^{-j2\pi nk/N} \tag{7.9}$$

Autocorrelation can be calculated using the same algorithm by replacing h with g.

Note: (from E.Oran Brigham "The Fast Fourier Transform").

7.3 Power Spectrum

ASC Name: *apowerspectrum*

Desription: Calculate the powerspectrum of one-D or two-D arrays using Fourier transforms.

Algorithm:

The powerspectrum of a finite-length function h(x) with a number of samples N can be calculated as follows:

-Calculate the FFT of h(k)

-Calculate the powerspectrum by taking the the modulus-squared of the fourier transform of h(k).

-Normalize in a such a way that the the sum of the pixel values of the power spectrum is equal to the sum of the squares of the pixel values of the input image.

7.4 Transforms

ASC Name: *atransform*

FFT

Description:

Calculate the Fast Fourier Transform of N-D dataset with respect to one or more axes.

Algorithm:

Use the IRAF STSDAS code by Phil Hodge. FFTPACK code obtained by Chris Biemesderfer from Argonne National Lab; math library over Arpanet (via netlib@argonne). Written at NCAR by Paul Swarztrauber in Apr 85.

OPTIONS:

1. Forward FFT - computes the forward Fourier transform of an image;

2. Inverse FFT - computes the inverse Fourier transform of an image.

7.5 Csmooth: Adaptive Smoothing

7.5.1 Introduction

The ASMOOTH algorithm used in *csmooth* was conceived and developed by Harald Ebeling, David White and Vijay Rangarajan. A detailed description of the algorithm and applications to X-ray imaging data can be found in Ebeling H., White D.A., Rangarajan F.V.N. ASMOOTH: A simple and efficient algorithm for adaptive kernel smoothing of two-dimensional imaging data, 2000, MNRAS, accepted

Smoothing of two-dimensional event distributions is a procedure routinely used in many fields of data analysis.

In practice, smoothing is generally achieved by a convolution

$$I'(\vec{r}\,) \equiv I(\vec{r}\,) \otimes \mathcal{K}(\vec{r}\,) = \int_{\mathbb{R}^2} \mathcal{K}(\vec{r} - \vec{r}\,')\, I(\vec{r}\,')\, d\vec{r}\,' \qquad (7.10)$$

$$\left(\int_{\mathbb{R}^2} \mathcal{K}(\vec{r}\,)\, d\vec{r}\,' = 1 \right)$$

of the measured data $I(\vec{r}\,)$ with a kernel function $\mathcal{K}$ (often also called 'filter' or 'window function'). Although the raw data may be an image in the term's common meaning [i.e. the data set can be represented as a function $I(x, y)$ where I is some intensity, and x and y are spatial coordinates], the two coordinates x and y can, in principle, describe any two-dimensional parameter space. The coordinates x and y are assumed to take only discrete values, i.e. the events are binned into (x, y) intervals. The only requirement on I that we shall assume in all of the following is that I is the result of a counting process in some detector, such that $I(x, y) \in \mathbb{N}_0$.

An image, as defined above, is a two-dimensional histogram and is thus often a coarse representation of the underlying probability density distribution (e.g. Merrit & Tremblay 1994, Vio et al. 1994). Although it is true that binned data contain less information than a discrete event distribution, there are also other considerations. For certain experiments, an unbinned event distribution may not exist – for instance, if the x and y values correspond to discrete PHA (spectral energy) channels. Also, there are circumstances where *some* binning can be desirable for analysis purposes. For example, in cases where the dynamic range of the data under consideration is large, the amount of data that need to be dealt with in the analysis can be reduced drastically by replacing the raw event distribution with image pixels. If the bin size is sufficiently small, the unavoidable loss of spatial resolution may be a small price to be paid for a data array of manageable size.

Smoothing of an observed, high-resolution image is of interest whenever the observed number of counts per resolution element of the instrumental set-up (in either x or y) approaches the expected background level. A practical criterion that tests for this condition is whether the signal (defined as the number of counts per pixel above the expected background) in the region of interest in $x - y$ space is in the Poissonian regime, i.e. is less than, or of the order of, 10, after the raw event distribution has been sorted into intervals whose size matches approximately the instrumental resolution. It is crucial in this context that the observed count statistics are not taken at face value but are corrected for background: imagine a data set featuring a high background level. (This background may be internal, i.e. originating from the detector [more general: the instrumental setup], or external.) In such a case the observed intensity (counts) distribution $I(x, y)$ may be high across the region of interest, suggesting good count statistics, although the signal above the background that we are interested in is actually very faint and poorly sampled. The statistics of the observed counts alone can thus be a poor indicator of the need for image smoothing.

Rebinning the data set into larger, and thus fewer, intervals improves the count statistics per pixel and reduces the need for smoothing. This is also the basic idea behind smoothing with a kernel of the form

$$\mathcal{K}(\vec{r}\,', \sigma) = \begin{cases} 1/(\pi\sigma^2) & \text{where } |\vec{r}\,'| < \sigma \\ 0 & \text{elsewhere} \end{cases} \qquad (7.11)$$

(circular 'top-hat' or 'box-car' filter of radial size σ), the only difference being that smoothing occurs semi-continuously (the step size being given by the bin size of the original data) whereas rebinning requires an additional phase information [the offset of the boundaries of the first bin with respect to some point of reference such as the origin of the (x, y) coordinate system]. However, when starting from an image binned at about the instrumental resolution, both rebinning and conventional smoothing share a well known drawback, namely that any improvement in the count statistics occurs at the expense of spatial resolution.

Although conventional smoothing algorithms usually employ more sophisticated functional forms for the kernel than the above 'top-hat' filter (the most popular probably being a Gaussian), the problem remains that a kernel of fixed size is ill-suited for images that feature real structure on various scales, some of which may be much smaller or much larger than the kernel size. In such a situation, small-scale features tend to get over-smoothed while large-scale structure remains under-smoothed. *Adaptive-kernel smoothing* (AKS) is the generic name for an approach developed to overcome this intrinsic limitation by allowing the kernel to vary over the image and adopt a position-dependent 'natural' size.

AKS is closely related to the problem of finding the optimal adaptive kernel estimator of the probability density distribution underlying a measured, unbinned event distribution. The advantages of adaptive kernel estimators for the analysis of discrete, and in particular one-dimensional, astronomical data have been discussed by various authors (e.g. Thompson 1990, Pisani 1993, Merritt and Tremblay 1994, Vio et al. 1994). An overview of adaptive filtering techniques in two dimensions is given by Lorenz et al. (1993).

A common feature of all non-parametric adaptive kernel algorithms is that the 'natural' smoothing scale for any given position is determined from the number of counts accumulated in its immediate environment. Following the aforementioned principle, smoothing occurs over a large scale where few counts have been recorded, and over a small scale where count statistics are good. AKS algorithms differ, however, in the prescription that defines how the amplitude of the local signal is to be translated into a smoothing scale.

A criterion widely used for discrete data is that of Silverman (1986). It determines the size, σ, of the local kernels relative to that of some global (i.e. non-adaptive, fixed) kernel ($\sigma_{\rm const}$) by introducing a scaling factor which is the inverse square root of the ratio of the globally smoothed data to their logarithmic mean. For images, and using the same notation as before, this means

$$\sigma(\vec{r}\,) = \sqrt{\frac{\langle I'_{\rm const}(\vec{r}\,)\rangle_{\log}}{I'_{\rm const}(\vec{r}\,)}}, \tag{7.12}$$

where $\langle I'_{\rm const}(\vec{r}\,)\rangle_{\log} = {\rm dex}\langle \log_{10} I'_{\rm const}(\vec{r}\,)\rangle$, and $I'_{\rm const}(\vec{r}\,)$ represents the convolution of the measured data with a kernel of fixed size $\sigma_{\rm const}$. However, whether or not this approach yields satisfactory results depends strongly on the choice of the global smoothing scale $\sigma_{\rm const}$ (Vio et al. 1994). In the context of discrete data sets, Pisani (1993) suggested a least-squares cross-validation procedure to determine an optimal global kernel size in an iterative loop. However, for binned data covering a large dynamical range, the dependence of the result on the size of the global kernel becomes very sensitive indeed, and the iteration becomes very time-consuming. Also the dependence on the somewhat arbitrary scaling law (eq. 7.12) remains. Other adaptive filtering techniques discussed recently in the literature include the HFILTER algorithm for square images (Richter et al. 1991, see also Lorenz et al. 1993) and the AKIS algorithm of Huang & Sarazin (1996).

7.5.2 Characteristics of ASMOOTH

The algorithm discussed in more detail in the following, ASMOOTH, is an AKS algorithm for images, i.e. binned, two-dimensional datasets of any size, which determines the local smoothing scale from the requirement that the significance *above the background* of any signal enclosed by the kernel must exceed a certain, preset value. The algorithm is similar to AKIS (Huang & Sarazin 1996) in that it employs a signal-to-noise (s.n.r.) criterion to determine the smoothing scale. However, other than AKIS, ASMOOTH does not require any initial fixed-kernel smoothing but determines the size of the adaptive kernel directly and unambiguously from the unsmoothed input data. ASMOOTH also goes beyond existing AKS algorithms in that its s.n.r. criterion takes the background (instrumental or other) of the raw image into account. This leads to significantly improved noise suppression in the case of large-scale features embedded in high background (which may be another real feature at even larger scale). Our approach yields smoothed images which feature a near-constant (or, alternatively, minimal) signal-to-noise ratio above the local background in all pixels. In contrast to most other algorithms which require threshold values to be set (e.g., for the H coefficients in the case of the HFILTER technique), ASMOOTH is intrinsically non-parametric. The only external parameters that need to be specified are the minimal and maximal signal-to-noise ratios (above the background) required under the kernel.

The simplicity of the determination of the local smoothing scale from the counts under the kernel and an estimate of the background (which can, but need not be specified by the user) greatly facilitates the translation of the smoothing prescription into a simple and robust computer algorithm, and also allows a straightforward interpretation of the resulting smoothed image.

7.5.3 Description of the algorithm

ASMOOTH adjusts the smoothing scale such that, at every position in the image, the resulting smoothed data values share the same signal-to-noise ratio (s.n.r.) above the background; one may call this the 'uniform significance' approach. The only external parameter required by ASMOOTH is the desired minimal s.n.r., $\tau_{\rm min}$. In order to ensure that statistically significant structure is not over-smoothed to a level of significance much higher than $\tau_{\rm min}$, a range of signal-to-noise ratios can be specified as a pair of $\tau_{\rm min}$, $\tau_{\rm max}$ values. Note, however, that the maximal significance criterion is a soft one and, also, is applied only at scales larger than the instrumental resolution (which is assumed to be similar to the pixel scale); under no circumstances will ASMOOTH blur pointlike features (pixels whose significance in the unsmoothed image exceeds $\tau_{\rm min}$) in order to bring their significance down below the $\tau_{\rm max}$ threshold. This implies also that, on the smallest scales, pixel-to-pixel variations (even insignificant ones) may still exist in the adaptively smoothed image. Besides the desired significances $\tau_{\rm min,\, max}$, estimates of the background $I_{\rm bkg}$ and the associated background error $\Delta I_{\rm bkg}$ are optional additional parameters.

The background is, by default, measured in an annulus surrounding the smoothing kernel thus providing a *local* estimate on the current smoothing scale. However, external background estimates ($I_{\rm bkg}$ and $\Delta I_{\rm bkg}$) can be supplied by the user. To allow background variations across the image to be taken into account, $I_{\rm bkg}$ and $\Delta I_{\rm bkg}$ can be supplied as images of the same dimensions as the raw image; in the case of a flat background $I_{\rm bkg}$ and $\Delta I_{\rm bkg}$ reduce to global estimates of the background and background error per pixel, i.e., single numbers. Note, however, that features in the adaptively smoothed image will not necessarily be *locally* significant at the specified level if an external background estimate is provided.

Internally, the threshold significances $\tau_{\rm min}$, $\tau_{\rm max}$ are translated into a minimal and a maximal integral number of counts, $N_{\rm min}$, $N_{\rm max}$, to be covered by the kernel. More precisely, the criterion is that

$$N_{\rm min} \le I'(\vec{r})/\mathcal{K}(\vec{0},\sigma(\vec{r})) \lesssim N_{\rm max} \tag{7.13}$$

where $\sigma(\vec{r})$ is the characteristic, position-dependent scale of the respective kernel. $N_{\rm min,max}$ in eq. 7.13 are determined from the definition of the minimal and maximal s.n.r. value $\tau_{\rm min,max}$,

$$\tau_{\rm min,max} = \frac{N_{\rm min,max} - N_{\rm bkg}}{\sqrt{N_{\rm min,max} + \Delta N_{\rm bkg}^2}}, \tag{7.14}$$

where, in analogy to the definition of $N_{\rm min,max}$ (cf. eqs. 7.10,7.13), $N_{\rm bkg}$ and $\Delta N_{\rm bkg}$ are the integral number of background counts under the respective kernel and the associated error. From eq. 7.14 follows

$$\begin{aligned} N_{\rm min,max} = N_{\rm bkg} \quad &+ \quad \frac{1}{2}\,\tau_{\rm min,max}^2 \\ &+ \quad \tau_{\rm min,max}\sqrt{N_{\rm bkg} + \Delta N_{\rm bkg}^2 + \frac{1}{4}\,\tau_{\rm min,max}^2}. \end{aligned} \tag{7.15}$$

For an adaptive circular top-hat kernel of size $\sigma(\vec{r})$ (cf. eq. 7.11), eq. 7.13 translates into $N_{\rm min} \leq \pi\,\sigma(\vec{r})^2\,I'(\vec{r}) \leq N_{\rm max}$, and the interpretation is straightforward: at least $N_{\rm min}$, but no more than $N_{\rm max}$, counts are required to lie within the area $\pi\,\sigma(\vec{r})^2$ that the smoothing occurs over. In the case of a uniform background, the value of $N_{\rm bkg}$ in eq. 7.15 is simply given by $n_{\rm bkg}\pi\,\sigma(\vec{r})^2$ where $n_{\rm bkg}$ is the global background level per pixel in the input image.

For any given pair of $(N_{\rm min}, N_{\rm max})$ values, a Gaussian kernel

$$\mathcal{K}(\vec{r}-\vec{r}\,', \sigma(\vec{r})) = \frac{1}{2\,\pi\,\sigma(\vec{r})^2}\ \exp\left(-\frac{|\vec{r}-\vec{r}\,'|^2}{2\sigma(\vec{r})^2}\right) \tag{7.16}$$

will yield considerably larger effective smoothing scales than a top-hat, as, in two dimensions, more than 60 per cent of the integral weight fall outside a $1\,\sigma$ radius, whereas, in the case of a circular top-hat kernel, all of $N_{\rm min}$ needs to be accumulated within a $1\,\sigma$ radius. (Note that, according to Eq. 7.16, it is the weights *per unit area* that follow a Gaussian distribution. The weights per radial annulus *do not*, which is why, for the kernel defined in Eq. 7.16, the fraction of the integral weight that falls outside the $1\,\sigma$ radius is much larger than the 32 per cent found for a one-dimensional Gaussian.) Which kernel to use is up to the user: ASMOOTH offers a choice of Gaussian and circular top-hat but any other, user-defined kernel can be specified as an optional argument in the function call.

The algorithm is coded such that the adaptively smoothed image is accumulated in discrete steps as the smoothing scale increases gradually, i.e.

$$I'_{\rm AKS}(\vec{r}) = \sum_i I'_i(\vec{r}) = \sum_i I_i(\vec{r}) \otimes \mathcal{K}(\vec{r}, \sigma_i)\ , \tag{7.17}$$

where σ_i starts from an initial value σ_0 which is matched to the intrinsic resolution of the raw image (i.e., the pixel size), and $I_i(\vec{r})$ is given by

$$I_i(\vec{r}) = \begin{cases} I(\vec{r}) & \text{where } N_{\rm min} \leq I'(\vec{r})/\mathcal{K}(\vec{0}, \sigma_i) \leq N_{\rm max} \\ & \quad \text{and } I(\vec{r}) \notin I_j(\vec{r}), j < i \\ 0 & \text{elsewhere.} \end{cases} \tag{7.18}$$

The adaptively smoothed image is thus accumulated in a "top-down" fashion with respect to the observed intensities as ASMOOTH starts at small kernel sizes to smooth the vicinity of the brightest pixels, and then increases the kernel size until, eventually, only background pixels contribute. Note that condition 7.18 ensures that pixels found to contain significant signal at a scale σ_i will not contribute to the image layers $I'_j, (j > i)$ subsequently produced with smoothing scales $\sigma_j > \sigma_i$. Consequently, each feature is smoothed at the smallest scale at which it reaches the required significance (see eq. 7.13), and low-s.n.r. regions are smoothed at an appropriately large scale even in the immediate vicinity of image areas with very high s.n.r.

In order to take full advantage of the resolution of the unbinned image, the size σ_0 of the smallest kernel is chosen such that the area enclosed by $\mathcal{K}(\vec{r}, \sigma_0)$ is about one pixel. For the circular top-hat filter of eq. 7.11 this means $\sigma_0 = 1/\sqrt{\pi}$; for the Gaussian kernel of eq. 7.16 we have $\sigma_0 = 1/\sqrt{9\pi}$. Subsequent values of $\sigma_i\,(i > 0)$ are determined from the requirement that eq. 7.13 be true. If a near-constant s.n.r. value is aimed at with high accuracy, i.e., if a $\tau_{\rm max}$ value very close to $\tau_{\rm min}$ is chosen, the smoothing scale σ_i will grow in very small increments, and the smoothing will proceed only slowly. In all our applications we found values of $\tau_{\rm max} \gtrsim 1.1 \times \tau_{\rm min}$ to yield a good compromise between CPU time considerations and a sufficiently constant signal-to-noise ratio of the smoothed image.

While the intrinsic resolution of the raw image (i.e. the pixel size) determines the smallest kernel size σ_0, the size of the image as a whole represents an upper limit to the size of the kernel. Although the convolution can be carried out until the numerical array representing the kernel is as large as the image itself, this process becomes very CPU time intensive as σ_i increases. Once the smoothing scale has exceeded that of the largest structure in the image, the criterion of eq. 7.13 can never be met as only background pixels contribute. Since the only features left unsmoothed at this stage are insignificant background fluctuations, the algorithm then smoothes the remaining pixels with the largest possible kernel. Unavoidably, the signal-to-noise ratio of these last background pixels to be smoothed does not meet the condition of eq. 7.13.

In order to allow the convolution to be performed over the whole image for any kernel size up to the size of the image itself, boundary conditions have to be specified. Of the three main possibilities truncation (also commonly referred to as padding), duplication, and wrapping, we found padding with zeroes to be the only feasible option if, firstly, the total number of counts in the unsmoothed image is to be preserved and, secondly, the creation of spurious structure at the field edges is to be avoided. (Duplication, where the values at the edges of the data array are repeated outside the image boundaries, does not always meet the first criterion due to problems at the corners of the image; wrapping can violate the second criterion if the image intensities do not fall off to some uniform background value at the field edges.) We ensure that the zero-padded regions outside the actual image do not affect the convolution result by dynamically renormalizing the kernel to the area inside the image boundaries.

The smoothed image obtained from the above procedure strictly conserves total counts (within the limitations set by the computational accuracy) and provides a fair representation of the original data at all positions.

Part IV

TCD Reference Manual

Chapter 8

TCD Tools: Input Parameters & Data Products

In this chapter we give a detailed account for each task in the **TCD** suite. We give the parameter file, a description of each parameter, and a discussion of the data products.

NB: The term 'Required' indicates that the user must provide a value for the indicated parameter before the program will run; i.e. the default parameter file does not contain a valid value.

8.1 *aconvolve*

8.1.1 *aconvolve*: input parameter file - with default values

```
#
# aconvolve.par file
#
#
        infile =                  Input IMAGE file name
    kernelspec =                  Kernel specification
       outfile =                  Output IMAGE file name
#
# auxiliary outputs
#
  (writekernel = no)              Output kernel
   (kernelfile = ./.)             Output kernel file name
     (writefft = no)              Write fft outputs
      (fftroot = ./.)             Root name for FFT files
#
```

```
# processing parameters
#
       (method = slide)           Convolution method
        (edges = wrap)            Edge treatment
        (const = 0)               Constant value to use at edges with edges=constant
          (pad = no)              Pad data axes to next power of 2^n
       (center = no)              Center FFT output
#
# user specific comments
#
      (clobber = yes)             Clobber existing output
      (verbose = 0)               Debug level
       (kernel = default)         Output format kernel
         (mode = ql)
```

8.1.2 *aconvolve*: Input Parameter Description

center

Optional
Data Type: boolean
Default: `no`
Allowed range: `yes, no`

Center FFT output. This parameter is currently inoperative.

clobber

Optional
Data Type: boolean
Default: `yes`
Allowed range: `yes, no`

Clobber existing output.

const

Optional
Data Type: real
Default: `0`

Constant value to use when `edges=constant` (see `edges` parameter).

edges

Optional
Data Type: string
Default: `wrap`
Allowed range: `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, eg 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, eg 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 (eg `const=0`), so 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.

Implicitly, using the FFT method of convolution implies wrapping edge treatment.

fftroot
Optional
Data Type: string
Default: `./.`

root file name for FFT output files

infile
Required
Data Type: string
Default: `""`

The input is an image. The input image 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.

kernel
Optional
Data Type: string
Default: `default`
Allowed range: `fits, iraf, default`

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`).

kernelfile

Optional
Data Type: string
Default: `./.`

file name for kernel image

kernelspec

Required
Data Type: string
Default: `""`

The examples show the syntax of the 3 `kernelspec` specifications that are available. The generic syntax of the `kernelspec` specification is:

```
<key>:<parameters>:<origin>
```

The required 'key' part of the specification (3 choices):

`<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 or any of the data types specified for the input image. For example,

```
kernelspec="file:/tmp/foo.fits"
```

`<key> = txt`

This tells the program to parse the string `<parameters>` to create the kernel. For example (example of a 2D kernel),

```
kernelspec="txt:((1,1,1),(1,1,1),(1,1,1)):(1,1)"
```

`<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, for example,

```
kernelspec="lib:box(2,1,3,3):(1,1)"
```

Currently supported libraries are:

box - an N-D 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.

Example: `box(2,1,3,3)` = 3x3 box with unit (1) amplitude.

gaus - an N-D non-rotated Gaussian. 'non-rotated' means you can specify a size for each axis separately, but not an angle for the major axis.

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

Example: `gaus(2,5,1,2,3)` = an elliptical 2-D Gaussian with unit (1) amplitude that extends out to 5 sigma in each direction. The sigma along the first axis = 2, in the 2nd = 3.

tophat - a 2D 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.

Example: `tophat(2,1,3,3)` = a unit high circular top hot that has a radius = 3 pixels.

The optional 'origin' part of the specification for `kernelspec`

The origin specification allows the user to specify the origin of the kernel within the kernal array used in the convolution. By 'origin' we mean the center (of a function like the tophat), or peak value (of a function such as a Gaussian). Typically 2D kernels have the origin in the "center" of the array. Many 1D kernels have the origin as the first element in the array.

To make the tool as generic as possible, the user is allowed to explicitly specify where the origin is. This alleviates some common restrictions that other tools may have (i.e. 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).

method

Optional
Data Type: string
Default: `slide`
Allowed range: `slide, fft`

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

method=slide

Sliding cell convolution is a convolution from first principles.

method=fft

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

a) 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.

If the edges of the data are "wrapped" (see `edges` parameter) 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".

outfile

Required
Data Type: string
Default: `""`

output image file name

The output file is an image. The output image has the same dimensions as the input image. The output data type is always "float" (BITPIX=-32).

The output format is controlled by the `kernel` parameter.

Note: To completely specify the file name one should include the extension name:

`myfile.fits[foo]` – for fits or `./[blah]` – for iraf

Not specifying the file block, (i.e. the part in []'s) may results in some odd name conventions (especially for the iraf kernel).

Note: If the output filename is "." or "path/." (where path is some directory path), then the output file will automatically be named by deriving a root from `infile`.

pad

Optional
Data Type: boolean
Default: `no`
Allowed range: `yes, no`

pad data array to length = 2^N.

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.

verbose

Optional
Data Type: integer
Default: `0`
Allowed range: `0-5`

Debugging information is provided at various steps thru the program. The verbosity of the debugging messages is controlled by the `verbose` parameter.

Debug	Output
0	Nothing (quite)
1	echo back parameters
2	report when entering/exiting routines
3	report size of data arrays
4	report kernel building steps
5	verbose

writefft

Optional
Data Type: boolean
Default: `no`
Allowed range: `yes, no`

output FFT of data and kernel (iff method=fft). The choice indicated by `writekernel` has no effect on `writefft`. The coordinates of the FFT output will be improved at a later time.

writekernel

Optional
Data Type: boolean
Default: `no`
Allowed range: `yes, no`

output kernel to an image. If `writefft` is set to 'yes', then the fft of the kernel will be output to real and imaginary files regardless of the value of `writekernel`.

8.1.3 *aconvolve*: Data Products Description

image

The output file is an image. The output image has the same dimensions as the input image. The output data type is always "float" (BITPIX=-32).

The output format is controlled by the `kernel` parameter.

8.2 *acrosscorr*

8.2.1 *acrosscorr*: input parameter file - with default values

```
  infile1 =                 Input file name #1
  infile2 =                 Input file name #2. Use none for autocorrelate
  outfile =                 Output file name
    (crop = no)             Crop output to size of infile1
     (pad = no)             Pad data to size of infile1 + infile2
  (center = no)             Center output
 (clobber = yes)            Clobber existing output file
 (verbose = 0)              Debug level
  (kernel = default)        Output format kernel
    (mode = ql)
```

8.2.2 *acrosscorr*: Input Parameter Description

center

Optional
Data Type: boolean
Default: `no`
Allowed range: `yes, no`

center output

If `center=yes`, the zero-offset point will be in the center of the output data array, otherwise, it will be at the 0 pixel location.

clobber

Optional
Data Type: boolean
Default: `yes`
Allowed range: `yes, no`

remove output file if it exists

crop

Optional
Data Type: boolean

Default: `no`
Allowed range: `yes, no`

crop output to size of 1st image

If `crop=yes`, the output is cropped to the size of `infile1`.

infile1

Required
Data Type: string
Default: `""`

input filename for 1st image

The input is a FITS image or IRAF .imh image file. The input can have the following data types: "short" (BITPIX=16), "long" (BITPIX=32), "float" (BITPIX=-32), and "double" (BITPIX=-64). Complex inputs are not currently supported.

Alternatively a FITS binary table can be binned using the datamodel syntax to specify the image (see example below).

If `infile2` is "NONE", then the autocorrelation of the `infile1` is computed.

infile2

Required
Data Type: string
Default: `""`

input filename for 2nd image

The input is a FITS image or IRAF .imh image file. The input can have the following data types: "short" (BITPIX=16), "long" (BITPIX=32), "float" (BITPIX=-32), and "double" (BITPIX=-64). Complex inputs are not currently supported.

Alternatively a FITS binary table can be binned using the datamodel syntax to specify the image (see example below).

If `infile2` is "NONE", then the autocorrelation of the `infile1` is computed.

kernel

Optional
Data Type: string
Default: `default`
Allowed range: `fits, iraf, default`

output format

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`).

outfile

Required
Data Type: string
Default: `""`

output image file name

The output image is an image of type FLOAT (32bit IEEE floating point number).

The `center`, `pad`, and `crop` parameters determine the output size.

By default, the output image is the maximum size in each direction from both input files. Thus if `infile1` is 5x2 and `infile2` is 3x3, the output will be 5x3.

The output format is controlled by the `kernel` parameter.

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

`myfile.fits[foo]` – for fits or `./[blah]` – for iraf

Not specifying the filename with the block specified, ie the part in []'s may results in some add name conventions (especially for the iraf kernel).

Note: If the output filenames are "." or "path/." (where path is some directory path), then the output files will automatically be named by deriving a root from the `infilereal` and adding a "real" and "imag" suffix.

pad

Optional
Data Type: boolean
Default: `no`
Allowed range: `yes, no`

pad data to size of image1+image2

If `pad=yes`, the data is padded to the size of `infile1` + `infile2`, so in the example above the output would be 8x5.

Padding is done before the correlation is performed. Cropping is done after, so if both are set to "yes", the final output will be cropped.

verbose

Optional
Data Type: integer
Default: `0`
Allowed range: `0-5`

processing verbosity

8.2.3 *acrosscorr*: Data Products Description

image The output image is an image of type FLOAT (32bit IEEE floating point number).

The `center`, `pad`, and `crop` parameters determine the output size.

By default, the output image is the maximum size in each direction from both input files. Thus if `infile1` is 5x2 and `infile2` is 3x3, the output will be 5x3.

If `crop` = yes, the output is cropped to the size of `infile1`.

If `pad` = yes, the data is padded to the size of `infile1` + `infile2`, so in the example above the output would be 8x5.

Padding is done before the correlation is performed. Cropping is done after, so if both are set to "yes", the final output will be cropped.

If `center`=yes, the zero-offset point will be in the center of the output data array, otherwise, it will be at the 0 pixel location.

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 something the file and extension name like:

myfile.fits[foo] – for fits or ./[blah] – for iraf

Not specifying the filename with the block specified, ie the part in []'s may results in some add name conventions (especially for the iraf kernel).

Note: If the output filenames are "." or "path/." (where path is some directory path), then the output files will automatically be named by deriving a root from the infilereal and adding a "real" and "imag" suffix.

8.3 *apowerspectrum*

8.3.1 *apowerspectrum*: input parameter file - with default values

```
#
# apowerspectrum tool
#
# inputs
#
    infilereal =                  Input file name for real part
    infileimag =                  Input file name for imaginary part
#
# output
#
       outfile =                  File name for output
#
# processing
#
          (pad = no)              Pad data array to next power of 2
       (center = no)              Center 0 frequency at center of array
        (scale = linear)          Output scale
         (crop = no)              Crop output at Nyquist frequency
#
# user
#
      (clobber = yes)             Delete existing output
      (verbose = 0)               Debug level
       (kernel = default)         Output format kernel
#
# mode
#
         (mode = ql)
```

8.3.2 *apowerspectrum*: Input Parameter Description

center

Optional
Date Type: boolean
Default: `no`
Allowed range: `yes, no`

center data

The `center` parameter controls whether the zero frequency point is at pixel=0 or pixel=N/2.

Note: both `center` and `crop` cannot be set to "yes".

clobber

Optional
Date Type: boolean
Default: `yes`
Allowed range: `yes, no`

clobber output file if it exists

crop

Optional
Date Type: boolean
Default: `no`
Allowed range: `yes, no`

crop output N/2

The `crop` parameter controls whether the entire powerspectrum is output or just up to the Nyquist frequency (N/2).

Note: both `center` and `crop` cannot be set to "yes".

infileimag

Required
Date Type: string
Default: `""`

input image for imaginary part of data

The input is an image. The input image can have any number of dimensions.

The input can have the following data type: "byte" (BITPIX=8), "short" (BITPIX=16), "long" (BITPIX=32), "float" (BITPIX=-32), and "double" (BITPIX=-64).

The real and imaginary parts of the input are input separately. If there is no real or imaginary part, then set the file name to "none" or "NONE".

(TBR) Currently, there MUST be a real part to the data. The imaginary part can be "none".

If `pad` is set to yes, the data is padded in size to the next power of 2 in all dimensions.

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.

infilereal

Required
Date Type: string
Default: `""`

input image for real part of data

The input is an image. The input image can have any number of dimensions.

The input can have the following data type: "byte" (BITPIX=8), "short" (BITPIX=16), "long" (BITPIX=32), "float" (BITPIX=-32), and "double" (BITPIX=-64).

The real and imaginary parts of the input are input separately. If there is no real or imaginary part, then set the file name to "none" or "NONE".

(TBR) Currently, there MUST be a real part to the data. The imaginary part can be "none".

If `pad` is set to yes, the data is padded in size to the next power of 2 in all dimensions.

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.

kernel

Optional
Date Type: string
Default: `default`
Allowed range: `fits, iraf, default`

output format kernel

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`).

outfile

Required
Date Type: string
Default: `""`

output file name

The output data file is a "float" image with the power spectrum computed.

The output format is controlled by the `kernel` parameter.

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

`myfile.fits[foo]` – for fits or `./[blah]` – for iraf

Not specifying the filename with the block specified, ie the part in []'s may results in some add name conventions (especially for the iraf kernel).

Note: If the output filenames are "." or "path/." (where path is some directory path), then the output files will automatically be named by deriving a root from the `infilereal` and adding a "real" and "imag" suffix.

pad

Optional
Date Type: boolean
Default: `no`
Allowed range: `yes, no`

pad data to next power of 2

If `pad` is set to yes, the data is padded in size to the next power of 2 in all dimensions.

scale

Optional
Date Type: string
Default: `linear`
Allowed range: `linear, log, db`

scale data by

The output is scaled according to the `scale` parameter.

The `scale` parameter can take the following values.

linear	linear scaling: output = `|FFT(a)|^2`
log	log (base 10) scaling: output = `log(|FFT(a)|^2)`
db	10 * log() scaling: output = `10*log(|FFT(a)|^2)`

Note: For `scale=log` and `scale=db`, IEEE NaN's may be generated in the output files, log(0) = NaN.

verbose

Optional
Date Type: integer
Default: `0`
Allowed range: `0-5`

processing info verbosity

Debugging information is provided at various steps thru the program. The verbosity of the debug messages is controlled by the `verbose` parameter.

8.3.3 *apowerspectrum*: Data Products Description

image The output data file is a "float" image with the power spectrum computed.

The output is scaled according to the `scale` parameter.

The output format is controlled by the `kernel` parameter.

8.4 *atransform*

8.4.1 *atransform*: input parameter file - with default values

```
#
# atransform.par file
#
# inputs
    infilereal =                  Input file name for real part
    infileimag = none             Input file name for imaginary part
#
# outputs
#
   outfilereal =                  File name for real part of output
   outfileimag =                  File name for imaginary part of output
#
# processing parameters
#
     transform = fft              Transform type
     direction = forward          Transform direction
          (pad = no)              Pad data array to next power of 2
       (center = no)              Center 0 frequency at center of array
#
# user preferences
#
      (clobber = yes)             Delete existing output
      (verbose = 0)               Debug level
       (kernel = default)         Output format kernel
#
# mode
#
         (mode = ql)
```

8.4.2 *atransform*: Input Parameter Description

center

Optional
Date Type: boolean
Default: `no`
Allowed range: `yes, no`

center output so DC is in middle

The `center` parameter controls whether the zero frequency point is at pixel=0 or pixel=N/2.

clobber

Optional
Date Type: boolean
Default: `yes`
Allowed range: `yes, no`

clobber output file if it exists

direction

Required

Date Type: string
Default: `forward`
Allowed range: `forward, reverse`

direction of transform

infileimag

Required
Date Type: string
Default: `none`

input image for imaginary part of data

The input is a FITS image (possibly IRAF .imh file). The input image can have any number of dimensions.

The input can have the following data type: "byte" (BITPIX=8), "short" (BITPIX=16), "long" (BITPIX=32), "float" (BITPIX=-32), and "double" (BITPIX=-64).

The real and imaginary parts of the input are input separately. If there is no real or imaginary part, then set the file name to "none" or "NONE".

(TBR) Currently, there MUST be a real part to the data. The imaginary part can be "none".

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.

infilereal

Required
Date Type: string
Default: `""`

input image for real part of data

The input is a FITS image (possibly IRAF .imh file). The input image can have any number of dimensions.

The input can have the following data type: "byte" (BITPIX=8), "short" (BITPIX=16), "long" (BITPIX=32), "float" (BITPIX=-32), and "double" (BITPIX=-64).

The real and imaginary parts of the input are input separately. If there is no real or imaginary part, then set the file name to "none" or "NONE".

(TBR) Currently, there MUST be a real part to the data. The imaginary part can be "none".

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.

kernel

Optional
Date Type: string
Default: `default`
Allowed range: `fits, iraf, default`

output format

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`).

outfileimag

Required
Date Type: string
Default: `""`

output file name for imaginary part of data

Two output data files are produced.

The data type is always "float" (BITPIX=-32). There is one file for the real part of the transform and one for the imaginary part of the transform.

The `center` parameter controls whether the zero frequency point is at pixel=0 or pixel=N/2.

The output format is controlled by the `kernel` parameter.

outfilereal

Required
Date Type: string
Default: `""`

output file name for real part of data

Two output data files are produced.

The data type is always "float" (BITPIX=-32). There is one file for the real part of the transform and one for the imaginary part of the transform.

The `center` parameter controls whether the zero frequency point is at pixel=0 or pixel=N/2.

The output format is controlled by the `kernel` parameter.

pad

Optional
Date Type: boolean
Default: `no`
Allowed range: `yes, no`

pad data to next 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 is 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.

transform

Required
Date Type: string
Default: `fft`
Allowed range: `fft`

type of transform to perform

The available transforms are listed below:

transform=fft

Compute the discrete Fourier Transform of the data using an FFT algorithm. The FFT algorithm was adapted from the STSDAS FFT routine: converted from FORTRAN to C and made to work in multiple dimensions. By standard convention, in the forward direction the sign of the complex exponential is negative and in the reverse direction the sign of the complex exponential is positive. In the forward direction, the data is normalized by the area of the data.

verbose

Optional
Date Type: integer
Default: `0`
Allowed range: `0-5`

processing info verbosity

Debugging information is provided at various steps thru the program. The verbosity of the debug messages is controlled by the `verbose` parameter.

8.4.3 *atransform*: Data Products Description

Two output data files are produced.

The data type is always "float" (BITPIX=-32). There is one file for the real part of the transform and one for the imaginary part of the transform.

The output format is controlled by the `kernel` parameter.

8.5 *csmooth*

8.5.1 *csmooth*: input parameter file - with default values

```
#
# csmooth.par file
#
#
        infile =                  input file name (raw image)
       outfile =                  output file name (adaptively
# smoothed image)
    outsigfile = .                output file name (image of the
# significance of the signal at each location of the smoothed image)
    outsclfile = .                output file name (image of the
# smoothing scales [kernel sizes] used at each location of the image)
#
# processing parameters
#
       conmeth = fft              Convolution method.
 conkerneltype = gauss            Convolution kernel type.
#
# Signifigance numbers
#
        sigmin = 4                minimal significance (S/N ratio) of
# the signal under the kernel
```

```
       sigmax = 5                maximal significance (S/N ratio) of
# the signal under the kernel
#
# Scales
#
       sclmin = INDEF            initial (minimal) smoothing scale in
# pixel, use INDEF for default (~1pixel)
       sclmax = INDEF            maximal smoothing scale, use INEF
# for default(~image size)
#
# User supplied scale map
#
      sclmode = compute          compute smoothing scales or user
# user-supplied map
       sclmap =                  input file name (image of
# user-supplied map of smoothing scales)
    (stepzero = 0.01)            initial stepsize by which smoothing
# scale increases
#
# background method
#
      (bkgmode = local)          background treatment
       (bkgmap = )               input file name (image of
# user-supplied background)
       (bkgerr = )               input file name (image of
# user-supplied background error)
#
# user specific comments
#
      (clobber = yes)            clobber existing output
      (verbose = 1)              verbosity of processing comments
       (kernel = default)        kernel of output format
         (mode = ql)
```

8.5.2 *csmooth*: Input Parameter Description

bkgerr

Optional
Data Type: string
Default: ‘‘’’

The input file name of a user-supplied background error map.

bkgmap

Optional
Data Type: string

Default: ‘‘’’

The name of a user supplied background map.

bkgmode

Optional
Data Type: string
Default: `local`
Allowed range: `local, user`

The parameter which selects how the background is to be computed. If set to 'local', the data surrounding the kernel will be used. If set to 'user', then `bkgmap, bkgerr` must contain the names of files containing a user supplied background map and a background error map.

clobber

Optional
Data Type: boolean
Default: `yes`
Allowed range: `yes, no`

Clobber existing output

conkerneltype

Data Type: string
Default: `gauss`
Allowed range: `gauss, tophat`

Convolution kernel type.

conmeth

Data Type: string
Default: `fft`
Allowed range: `slide, fft`

The convolution method. If the original 'Asmooth' algorithm is desired, choose 'slide'.

infile

Required
Data Type: string
Default: `""`

Input IMAGE file name

kernel

Optional
Data Type: string
Default: `default`
Allowed range: `fits, iraf, default`

Output format kernel

outfile

Required
Data Type: string

Output smoothed image file name

outsclfile

Data Type: string
Default: `.`

Output file name of an image, where each pixel has the value of the scale used at that location.

outsigfile

Data Type: string
Default: `.`

Output file name of an image, where each pixel has the value of the significance (sigma) at that location.

sclmap

Optional
Data Type: string
Default: `‘‘’’`

If `sclmode`=user, then a map must be supplied by the user such that each pixel has the value of the scale to use at that location.

sclmode

Optional
Data Type: string
Default: `compute` Allowed range: `compute, user`

`sclmode` controls whether *csmooth* computes all smoothing scales internally based on the specified significance threshold (the default), or whether the smoothing scales are to be taken from a user-supplied map.

If a map of predefined smoothing scales is supplied by the user (through the `sclmap` parameter), the values of the parameters `sigmin`, `sigmax`, `sclmin`, `sclmax`, and `stepzero` are ignored, and any features in the smoothed output image will, in general, not be significant at any uniform level.

sclmax

Data Type: real
Default: `INDEF`

`sclmax` is the maximum scale size allowed (in pixels). The default value (INDEF) will allow the scale increase so as to obtain a significance within the range specified by `sigmin, sigmax`, which often means that the largest scale approaches the size of the map.

sclmin

Required
Data Type: real
Default: `INDEF`

`sclmin` is the minimum scale size to use, in pixels. The default value (INDEF) will cause the program to compute the minimum scale, which will often be of order one pixel (depending on the significance level).

sigmax

Data Type: real
Default: `5`

`sigmax` defines the upper boundary of significance of the (background corrected) signal under the kernel used in selecting scale sizes.

sigmin

Data Type: real
Default: `4`

`sigmin` defines the lower boundary of significance of the (background corrected) signal under the kernel used in selecting scale sizes.

stepzero

Data Type: real
Default: `0.01`

`stepzero` is the initial step size for increasing the scale of the convolution kernel.

verbose

Optional
Data Type: integer
Default: `0`
Allowed range: `0-5`

Debug level

8.5.3 *csmooth*: Data Products Description

outfile The output map is the adaptively smoothed version of the data. The output file is an image of the same dimensions as the input image. The output data type is always “float” (BITPIX=-32). The output format is controlled by the kernel parameter.

outsigfile This is a map of the significance of the signal in each pixel of the smoothed image. The significance is computed using Gaussian statistics and taking into account the expectation value of the background in the kernel area. If not supplied by the user, the background expectation is computed from a local estimate obtained from the counts in an annulus surrounding the kernel.

outsclfile An image of the smoothing scales (kernel sizes) used at each location of the smoothed image. The smoothing scales are the smallest that allow the significance threshold to be reached. The scales are the sizes of the smoothing kernel used at any location of the image: standard deviation in the case of a Gaussian kernel, radius in the case of a tophat kernel.

REFERENCES

R. Bracewell “The Fourier Transform and Its Applications”, 2nd Ed.; McGraw-Hill, Inc.; 1985.

Brigham, E.Oran “The Fast Fourier Transform”, 1974, Prentice-Hall, Inc

Ebeling H., White D.A., Rangarajan F.V.N. “ASMOOTH: A simple and efficient algorithm for adaptive kernel smoothing of two-dimensional imaging data”, 2000, MNRAS, submitted.