asmooth

November 4, 2014

Abstract

This task smooths FITS images either simply or adaptively, with options to mask, weight and exposure-correct.

1 Instruments/Modes

asmooth is not XMM-specific: it can be applied to any FITS image.

2 Use

pipeline processing	yes
interactive analysis	yes

3 Description

This task smooths FITS images by a weighted cyclic convolution. The convolution procedure itself is described in section 3.1. There are many ways in which the user can specify the convolver (see section 3.3). The user can also supply an exposure map (section 3.5), a weight image (section 3.4) and/or an output mask image (also section 3.4). An image of the variance (square of the standard deviation) of the input image may also be supplied, and an image of the variance of the smoothed image obtained (section 3.2).

3.1 Convolution details

A cyclic convolution of an input image $I_{x,y}$ weighted by $w_{x,y}$ gives output $O_{x,y}$ according to the following prescription:

$$O_{x,y} = C \; \frac{\sum_{i=L}^{M} \sum_{j=P}^{Q} c_{i,j} \; w_{x-i,y-j} \; I_{x-i,y-j}}{\sum_{i=L}^{M} \sum_{j=P}^{Q} c_{i,j} \; w_{x-i,y-j}}, \tag{1}$$

where $c_{x,y}$ is the convolver and

$$C = \sum_{x=L}^{M} \sum_{y=P}^{Q} c_{x,y}.$$

The indices $x - i$ and $y - j$ are calculated modulo the image sizes X and Y respectively, that is, they wrap around the image limits. This cyclic property is unavoidable if it is desired to make efficient use of the Fourier transform in performing the convolution. Potentially this is a nuisance because it means, for example, that values at the left-hand edge of the image can become mixed with values on the right-hand edge, and so forth. The task avoids this by padding the image with zero-valued pixels in both x and y directions. The pad size is equal to the width of the convolver in that direction. If there are multiple convolvers (see sections 3.3.4, 3.3.5 and 3.3.6), the largest convolver sizes are used.

In fact the 'blank space' areas around the edges of the image are handled in three steps, as below.

1. If a smaller rectangle can be excised from the input image such that all the pixels of the excess are zero-valued, this is done.
2. The pad described above is added.
3. The Fast Fourier Transform (FFT) algorithm used by the task works most efficiently if each dimension of the image is a multiple of 2, 3, 5 or 7. For each dimension of the working image, **asmooth** calculates the next largest integer which obeys this condition and adds further blank space to increase the dimension to that value.

After the convolution is done, before the output is written to file, the process is reversed: the pad is first cut away, then the original amount of blank space is restored.

The convolution may be done either directly or via FFT. If left to itself, the task will try to use the method which is quickest. If the image is large and the number of convolvers small, this will generally be the FFT method; however direct convolution becomes more efficient if the image is divided into many small patches of different convolution. If the user desires the convolutions to be unilaterally performed using one method or another, they should make use of the `forcecalctype` and `calcbyfft` parameters.

The two methods produce identical outputs except for small differences caused by different propagation of rounding errors. However the FFT method will, because it always acts on the whole image, in general leave few if any pixels in the output which are exactly equal to zero. Use of a mask (section 3.4) or an exposure map (section 3.5) can be helpful in this instance.

3.2 Variance images

The user has the opportunity both to supply a variance image of the input and to receive a variance image of the smoothed output. (A variance image is an image of the variances, that is the squares of the standard deviations, in the values of the input or output images.) A variance image is supplied by setting parameter `readvarianceset`='yes' and naming the image dataset in `invarianceset`; the variance of the `outset` can be obtained via parameters `writevarianceset` and `outvarianceset`.

If the task is required to make use of the variance (either because the user has set `writevarianceset`='yes' or `smoothstyle`='adaptive'), but none is supplied by the user, the task assumes that the input image `inset` is Poissonian - that is, that the image itself is a reasonable estimate of its own variance.[1] In this

[1]For small values this becomes a poor approximation. If the matter is of importance, an iterative procedure in which the smoothed variance is fed back into the task might be worth trying.

case naturally only `inset`s with pixel values >= 0 are accepted. If a variance set is supplied, all pixel values of the variance image must be >= 0, but the pixels of `inset` may have any value.

If `writevarianceset`='yes', the task calculates the variance in the smoothed output image `outset`. The variance $\sigma^2(y)$ of a linear combination $y = \sum_i a_i x_i$ of independent variates x_i is given by

$$\sigma^2(y) = \sum_i a_i^2 \, \sigma^2(x).$$

(Note there is no connection between the σ used here and the σ used in section 3.3.1 to signify the characteristic width of a gaussian! I'm using the same symbol for two different things, but hopefully this won't be too confusing.) By applying this to equation 1 we get

$$\sigma(O)_{x,y} = C \, \frac{\sqrt{\sum_{i=L}^{M} \sum_{j=P}^{Q} c_{i,j}^2 \; w_{x-i,y-j}^2 \; \sigma^2(I)_{x-i,y-j}}}{\sum_{i=L}^{M} \sum_{j=P}^{Q} c_{i,j} \; w_{x-i,y-j}}, \tag{2}$$

where $\sigma(I)_{x,y}$ is the standard deviation of the input image and $\sigma(O)_{x,y}$ that of the output.

3.3 Ways of specifying the convolvers

In all the following descriptions, the convolving function is designated by $c_{x,y}$, where x and y are integer pixel numbers. Convolvers must necessarily be of finite extent, hence are only defined over a rectangular array.

It is necessary to have some way to determine the 'phase' of any convolver: ie, which pixel of the convolver array should be taken to have row and column indices y and x equal to zero. The rule applied by **asmooth** is as follows. If the convolver has for example N rows, then row 1 plus the integer[2] part of$N/2$ is taken as the zeroth row. This means that if N is odd, the central row is the zeroth row, ie the row indices are taken to extend from $-(N-1)/2$ to $(N-1)/2$; if N is even, the row indices run from $-N/2$ to $N/2-1$. The same rule is applied to the columns.

3.3.1 Single gaussian convolver

`smoothstyle`='simple', `convolverstyle`='gaussian'.

The convolving kernel is a rotationally-symmetric gaussian, truncated onto a square array of side length $2N+1$:

$$c_{x,y} = A \, \exp(-[x^2+y^2]/2\sigma^2) \text{ for } |x|,|y| \le N, \text{ 0 else.}$$

The width σ is read from the parameter `width` and N is set equal to 1 plus the integer part of 5σ. If `normalize`='yes', A is set from

$$A^{-1} = \sum_{x=-N}^{N} \sum_{y=-N}^{N} \exp(-[x^2+y^2]/2\sigma^2);$$

[2]'Integer' in the present document always means truncated rather than rounded.

otherwise $A = 1$.

3.3.2 Single top-hat convolver

`smoothstyle`='simple', `convolverstyle`='tophat'.

The convolving kernel approximates a filled circle, viz:

$$c_{x,y} = A \text{ for } x^2 + y^2 \leq r^2, \text{ 0 else.}$$

The radius r is read from the parameter `width`. The convolver array is again square and of side length $2N + 1$, where N equals the integer part of r. If `normalize`='yes', A^{-1} is set from $\sum\sum c$ as before, otherwise $A = 1$.

3.3.3 Single square-box convolver

`smoothstyle`='simple', `convolverstyle`='squarebox'.

This simple convolver is a square array of side equal to $1 + 2\times$(the integer part of `width`) and value specified by A, which is calculated as in the previous two sections.

3.3.4 Multi-convolver smoothing

`smoothstyle`='withset'.

In this mode, different parts of the image may be smoothed by different convolvers. A list or library of convolvers is required, also an 'index' image which shows those parts of the image to be smoothed by each convolver. These are supplied in the form of a FITS dataset, as described in section 7. Separate parameters `inconvolversarray` and `inindeximagearray` are provided for the convolver list and the index image, so these may be stored either in the same dataset or in two different datasets. The image `inindeximagearray` must have the same dimensions as `inset`, and the maximum value of `inindeximagearray` must not exceed the number of convolvers in `inconvolversarray`.

Regardless of the value of `withweightset` (see section 3.4 for the use of weight and mask images), the convolution within each index 'patch' is always a weighted convolution, the weight being 1 (or $1\times$ the `weightset` image if this is supplied) within the patch allocated to that convolver and 0 outside it. Likewise, the result of each patch convolution is masked by the patch (ANDed with the `outmaskset` mask if this is supplied) before being added to the output image. In future a parameter may be provided to allow the user to switch off this patch weighting and masking if desired.

If the user desires the convolvers to be normalized, the `normalizeset` parameter should be set.

3.3.5 Adaptive smoothing

`smoothstyle`='adaptive'.

This is exactly the same as the patchwork convolution described in section 3.3.4, except here the task calculates the library of convolvers and the associated index image itself.

Adaptive smoothing is designed for Poissonian images made by counting photons or events. (Note that all images made by XMM instruments have this character.) For such non-negative images it is convenient to define the signal-to-noise ratio (SNR) at each pixel as the value at that pixel divided by its standard deviation. The intent of adaptive smoothing is to produce an output image in which the SNR at each pixel is as close as possible to the constant value given in parameter `desiredsnr`. Through this process, fainter areas become more thoroughly smoothed than brighter areas. This implies that the detail which one wishes to preserve from smoothing should be bright rather than dark - it would not be advisable, for example, to adaptively smooth an image of absorbing dust filaments on a bright background.

The convolvers are normalized gaussians of the form described in section 3.3.1. The exact distribution of their widths is not of primary importance so long as there are enough of them within a wide enough range to cater for the variations in SNR of the input image. There are several ways in which the widths of the gaussians can be established. Firstly, the user can provide a list of widths directly via the `userwidths` parameter; alternatively, the user can provide minimum and maximum values (parameters `minwidth` and `maxwidth`), specify the number of convolvers (`nconvolvers`) and the scaling rule to be used (`widthliststyle`), and allow the task to calculate the widths.

The relation between input and output standard deviation is given by equation 2. For the square, symmetric gaussian convolvers of section 3.3.1 this becomes

$$\sigma(O)_{x,y} = C\,\frac{\sqrt{\sum_{i=-N}^{N}\sum_{j=-N}^{N} c_{i,j}^2\ w_{x-i,y-j}^2\ \sigma^2(I)_{x-i,y-j}}}{\sum_{i=-N}^{N}\sum_{j=-N}^{N} c_{i,j}\ w_{x-i,y-j}},$$

giving the following equation for the signal-to-noise ratio of the smoothed image:

$$SNR(O)_{x,y} = \frac{\sum_{i=-N}^{N}\sum_{j=-N}^{N} c_{i,j}\ w_{x-i,y-j}\ I_{x-i,y-j}}{\sqrt{\sum_{i=-N}^{N}\sum_{j=-N}^{N} c_{i,j}^2\ w_{x-i,y-j}^2\ \sigma^2(I)_{x-i,y-j}}}. \tag{3}$$

As described in section 3.2, the user can either supply an explicit image of the variance $\sigma^2(I)_{x,y}$ of the input image, or this can be left implicit by leaving `readvarianceset` at its default value of 'no'. In the latter case, the task works under the assumption that the input image is Poissonian and thus may be used as an estimate of its own variance.

It is not possible to invert equation 3 so as to arrive at the required width of the gaussian convolver needed to give $SNR(O)_{x,y}$ =`desiredsnr` at each (x, y). Instead the solution must be found iteratively. The procedure used by **asmooth** is simple and therefore robust: for each image pixel, the task starts at the broadest gaussian in its library and works through the library until $|SNR(O)_{x,y}-$`desiredsnr`$|$ reaches a minimum.

The task actually makes two passes through the library of gaussians: the first as described above, to calculate the index image; the second refers to the index image while performing the convolution, exactly as described in section 3.3.4.

Note that variance values = 0 are permitted. This leaves open the possibility that the denominator of equation 3 may be zero-valued at some (x, y). Equation 3 is not calculated for such pixels: they are instead simply allocated in the index image to the first, ie broadest, gaussian.

Occasionally it is useful to be able to apply the adaptive smoothing calculated for one image to another. The smoothing information can be stored to file in two ways: either directly as a set of convolver images plus an index image (via parameters `writeconvolvers`, `outconvolversset` and `outindeximageset`), or via a template image (`writetemplateset` and `outtemplateset`). To find out how to make use of these files, look in sections 3.3.4 and 3.3.6 respectively.

3.3.6 Smoothing from an adaptive template

`smoothstyle`='template'.

Suppose you wish to make a false-colour image from several images of the same piece of sky taken in different energy bands (or at different values of any other quantity). If the input images are event-based images of low exposure it may be rewarding to adaptively smooth them before combining them into a colour image. However, if each image is smoothed separately, in general convolvers of different widths will be allocated to the same place in the sky in the different images. This can cause a sort of chromatic aberration. To avoid this it is necessary to smooth each image according to the same template. This template might for example be established by adaptively smoothing the sum of all input images, then writing the result to file via the parameters `writetemplateset` and `outtemplateset`. This template image just stores the characteristic width of the gaussian used to smooth each pixel of the input image. If you select `smoothstyle`='template' and supply the template set to parameter `smoothstyle`='intemplateset', you will get the same smoothing pattern for any `inset`.

The same result can be achieved through `outconvolversset` and `outindeximageset`, which is a more robust method. I'll probably abandon the template facility eventually.

3.4 Weighting and masking

Images necessarily have sharp boundaries at their edges, and may in addition have sharp internal steps between zero- and non-zero-valued pixels. Such steps become 'smeared out' through the convolution. Smearing manifests itself as both 'drooping' of non-zero values as the edge is approached and 'bleeding' of non-zero values into the previously zero-valued area. A glance at equation 1 shows that drooping can be avoided by supplying as weight image (parameters `withweightset` and `weightset`) something which exhibits the same steps as the input image. Where this is the case, both numerator and denominator of equation 1 fall to approximately half their usual values at pixels adjacent to an image step to zero; if no weight image is supplied, all values of $w_{x,y}$ are set to 1, and thus only the numerator of 1 falls by about half at such pixels, so also the result as a whole. An exposure map often serves very well as a weight image. This can also prevent large noise fluctuations in areas of low exposure when the input image is exposure-corrected before smoothing (see section 3.5).

Droop correction by weighting carries on in principle even outside the area of the input image which was non-zero-valued. This facility can be used to interpolate over gaps or holes in the input and weight images. However such extrapolation becomes increasingly noisy far from the step as fewer and fewer non-zero pixels of the input and weight images overlap the convolver. Indeed such extrapolation cannot extend further than the convolver array dimensions from non-zero areas of the weight image, else the denominator of equation 1 would become zero.

This extrapolation can be controlled by supplying **asmooth** with a mask image via parameters `withoutmaskset` and `outmaskset`. The mask actually has two effects: only pixels for which the mask is TRUE contribute to the convolution; and convolution is only performed for pixels for which the mask is TRUE. In fact we should rewrite equation 1 to read

$$O_{x,y} = C\ \delta_{x,y}\ \frac{\sum_{i=L}^{M}\sum_{j=P}^{Q}\delta_{x-i,y-j}\ c_{i,j}\ w_{x-i,y-j}\ I_{x-i,y-j}}{\sum_{i=L}^{M}\sum_{j=P}^{Q}\delta_{x-i,y-j}\ c_{i,j}\ w_{x-i,y-j}} + (1-\delta_{x,y})I_{x,y}, \qquad (4)$$

where $\delta_{x,y}$ represents the mask.

Note from section 7 that the image supplied to `outmask` may have any numeric data type. The exposure

map for example is often a convenient choice.

If the absolute value of the numerator of equation 1 falls below a certain minimum value, the task will do the following: (i) issue the warning outMaskTooNarrow; (ii) set the value of that pixel in the outset to zero; (iii) set the respective pixel of an internal logical image to TRUE. This logical image of pixels where weighted smoothing could not be carried out can be written to file by setting `writebadmaskset`='yes'.

An example of a situation in which a small amount of controlled extrapolation is desirable is in the creation of background maps by smoothing. Bright sources should be removed from any image before smoothing it to create such a map. However if pixels in the neighbourhood of sources are simply set to zero, smoothing won't remove the resulting holes in the image, just blur them. The surrounding background values can be interpolated into the holes by supplying a weight image (eg the exposure map) with a matching set of holes. Provided the holes do not approach the convolver array dimensions in size, the holes should become completely filled in. Note that if a mask is also used, the mask should *not* have holes cut in it at the source positions. In total, the recommended procedure is as follows:

1. Make a source-mask image which has holes at source locations. From its appearence the name of 'cheese mask' suggests itself. (Unfortunately the present sas offers no easy way to do this.)
2. Multiply the input image by the cheese mask. Let us call the result the cheesed input image.
3. Multiply the exposure map by the cheese mask. Let us call the result the cheesed weight image.
4. Call asmooth as follows:

```
asmooth inset=<cheesed input image>
withweightset=yes weightset=<cheesed weight image>
withoutmaskset=yes outmaskset=<original exposure map>
```

If you still get the 'outMaskTooNarrow' warning, you will either have to reduce the size of some of the holes or use a broader convolver.

Note that, although the exposure map is recommended for three separate inputs of **asmooth** (`weightset`, `outmaskset` and `expimageset`), these three functions are separate and should not be confused.

3.5 Correcting for exposure before smoothing

Different sections of an image may have different exposure. Few problems are likely to be encountered where the exposure gradient is small compared to the characteristic width of the convolver. Sharp variations in exposure (such as occur for example in image mosaics) will however become blurred through the smoothing. The obvious remedy for this is to divide the input image by the exposure map before smoothing. This facility is provided in **asmooth** through the parameters `withexpimageset` and `expimageset`.

Clearly, the division by exposure cannot be done at pixels where the exposure map is zero-valued. The task handles such pixels by setting the corresponding pixels of the weight image to zero. (Whether or not the user explicitly supplies a weight image, an internal weight image is always constructed; the difference being that if `withweightset`='no', the entire internal weight image is initially set equal to 1.) Note that this also takes care of 'droop' at image edges in the same way as is described in section 3.4; however the output image may still contain noise at places where the exposure gradient is very large. The only way to avoid such noise is to supply the exposure map also as a weight image.

If you wish to create a background map in counts per pixel from an XMM exposure in which the exposure time varies significantly between CCDs, then you will need to remultiply the the output image by the exposure map; if however you wish to create a mosaic from several different cameras or pointings, you may wish not to do so. For this reason, remultiplication by the exposure map has been made optional via the `remultiply` parameter.

3.5.1 Exposure division, variance and adaptive smoothing

A further matter to consider is the effect of exposure correction on the variance map. Since variance is the square of standard deviation, the obvious thing to to is to divide the input variance image by the square of the exposure map ϵ. (Remember that, regardless of the value of `readvarianceset`, the task makes use of an internal variance image if either `writevarianceset`='yes' or `smoothstyle`='adaptive'.) For all choices of `smoothstyle` but 'adaptive', this makes sense. However, recall that adaptive smoothing makes use of the input variance to calculate the signal-to-noise ratio (SNR) of the input image. From equation 3 one can see that correct division of variance by exposure squared produces the following equation for SNR:

$$SNR(O)_{x,y} = \frac{\sum_{i=-N}^{N}\sum_{j=-N}^{N} c_{i,j}\ (w_{x-i,y-j}/\epsilon_{x-i,y-j})\ I_{x-i,y-j}}{\sqrt{\sum_{i=-N}^{N}\sum_{j=-N}^{N} c_{i,j}^2\ (w_{x-i,y-j}^2/\epsilon_{x-i,y-j}^2)\ \sigma^2(I)_{x-i,y-j}}}. \quad (5)$$

This treatment of variance will produce an image in which short-exposure areas are more heavily smoothed than long-exposure areas. Short-exposure areas will in other words exhibit broader spatial variation - they will appear more 'out of focus'. This can be irritating. An alternative approach is provided via the parameter `expmapuse`. If this is set to 'samesnr' (the default), equation 5 is employed. If `expmapuse` is instead set to 'samesize', the input variance is divided by $\epsilon \times \max(\epsilon)$ instead of by ϵ^2. The effect on the output of the 'samesize' selection is to leave areas of the image of shorter exposure with a similar distribution of spatial frequencies to, but larger noise brightness amplitude than, areas of longer exposure. Note that this ONLY affects the SNR calculation: if the variance of the output image is also desired, this is calculated according to equation 2 in the standard way.

3.5.2 Smoothing of mosaics

Image mosaics assembled from several separate pointings look better if the result is divided by the mosaiced exposure and then smoothed. The `withexpimageset` facility of **asmooth** allows you to do this. Users who wish to make mosaics from XMM EPIC data however need to exercise some caution. The largest component of the exposure spatial variation in XMM EPIC images is the mirror vignetting function, which changes typically by a factor of three between the optic axis and the edge of the field of view. The problem is that a significant fraction of the background in such images arises from sources (eg cosmic-ray fluorescence, soft protons, electronic noise) which are subject to little or no vignetting. Dividing an image which comprises a flat part plus a vignetted part by a purely vignetted function is going to leave one with an output image which appears to have a hollow centred on the optic axis - ie, is anti-vignetted. It is therefore advisable to subtract the flat background component first before dividing by the exposure map. NOTE however that this will destroy the Poisson nature of the input image, which introduces a couple of complications: firstly, if you also want the variance of the output, you will need to explicitly supply an `invarianceset` and not leave it to the task to calculate it; secondly, if you want to adaptively smooth your mosaic (see section 3.3.5) you will have to adopt a more complicated procedure than usual.

The recommended procedure for producing simply-smoothed mosaics is as follows:

1. Make a mosaic of the raw images.
2. Make a mosaic of the exposure maps.
3. Make a mosaic of the nonvignetted background (NVB) maps. (It is hoped that in future there will be a sas task which will be able to decompose XMM EPIC background into its various components; unfortunately, for the present you are on your own.)
4. Subtract the NVB mosaic from the raw mosaic.
5. Do

```
asmooth inset=<(raw-NVB) mosaic> smoothstyle=simple
withexpimageset=yes expimageset=<exp img mosaic> remultiply=no
```

   If you also want a map of the output variance, add to this

```
readvarianceset=yes invarianceset=<raw mosaic>
writevarianceset=yes outvarianceset=<pick a name>
```

The recommended procedure for producing adaptively-smoothed mosaics is as follows:

1. Make a mosaic of the raw images.
2. Make a mosaic of the exposure maps.
3. Make a mosaic of the nonvignetted background (NVB) maps. (It is hoped that in future there will be a sas task which will be able to decompose XMM EPIC background into its various components; unfortunately, for the present you are on your own.)
4. Adaptively smooth the raw mosaic and save the convolvers using parameters `writeconvolvers`, `outconvolversset` and `outindeximageset`. I don't at present have a recommendation for use of `withexpimageset` or `expmapuse` at this stage: best to experiment to see which produces the most acceptable eventual output.
5. Subtract the NVB mosaic from the raw mosaic.
6. Do

```
asmooth inset=<(raw-NVB) mosaic> smoothstyle=withset
inconvolversarray=<conv set saved from step 4>
inindeximagearray=<index img saved from step 4>
readvarianceset=yes invarianceset=<raw mosaic>
withexpimageset=yes expimageset=<exp img mosaic> remultiply=no
```

4 Parameters

This section documents the parameters recognized by this task (if any).

Parameter	Mand	Type	Default	Constraints
inset	yes	dataset		

The image dataset to be smoothed.

outset	no	dataset	outimage.ds	

The resulting smoothed data set.

tempset	no	dataset	tempimage.ds	

Name for a temporary dataset (only important if one is running several parallel asmooths).

smoothstyle	no	string	adaptive	simple—adaptive—withset—template

The type of smoothing desired.

convolverstyle	no	string	gaussian	gaussian—tophat—squarebox

This parameter is read if `smoothstyle`='simple' is chosen and prescribes the shape or type of convolver to use to smooth the image.

width	no	real	5.0 pixels	0.0 $\leq$ `width` $\leq$ 100.0 pixels

This parameter is read if `smoothstyle`='simple' is chosen. It governs the width of the various types of simple convolver. See section 3.3 for further details.

normalize	no	boolean	yes	

This parameter is read if `smoothstyle`='simple' is chosen. If set, the convolver is divided by its array sum before use. See section 3.3 for further details.

withuserwidths	no	boolean	no	

This parameter is read if `smoothstyle`='adaptive' is chosen. If set, the task reads a list of gaussian-convolver sigma values from the `userwidths` parameter. See section 3.3.5 for further details.

userwidths	yes	real list	0	0.0 $\leq$ `userwidths` $\leq$ 100.0 pixels

The list of gaussian-convolver sigma values read when `withuserwidths`='yes'. The values must occurr in a monotonically increasing sequence. See section 3.3.5 for further details.

nconvolvers	no	integer	20	2 $\leq$ `nconvolvers` $\leq$ 126

If `smoothstyle`='adaptive' is chosen but `withuserwidths`='no', the task constructs a library of `nconvolvers` gaussian convolvers. See section 3.3.5 for further details.

minwidth	no	real	0.0 pixels	0.0 $\leq$ `minwidth` $\leq$ 100.0 pixels

If `smoothstyle`='adaptive' is chosen but `withuserwidths`='no', the task constructs a library of gaussian convolvers which have sigma values ranging from `minwidth` to `maxwidth`. See section 3.3.5 for further details.

maxwidth	no	real	10.0 pixels	0.0 $\leq$ `maxwidth` $\leq$ 100.0 pixels

If `smoothstyle`='adaptive' is chosen but `withuserwidths`='no', the task constructs a library of gaussian convolvers which have sigma values ranging from `minwidth` to `maxwidth`. See section 3.3.5 for further details.

widthliststyle	no	string	linear	linear—log—sqrt

If `smoothstyle`='adaptive' is chosen but `withuserwidths`='no', the task constructs a library of gaussian convolvers. The sigma values of successive gaussians are spaced according to `widthliststyle`. See section 3.3.5 for further details.

desiredsnr	no	real	10.0	0.0 < `desiredsnr`

Desired signal-to-noise ratio in an adaptively-smoothed image.

writetemplateset	no	boolean	yes	

After completion of (adaptive) smoothing, save an image of the convolver widths to a file name specified by `outtemplateset`. See section 3.3.6 for further details.

outtemplateset	no	dataset	template.ds	

Name of the template image to be written when `writetemplateset`='yes'. See section 3.3.6 for further details.

writeconvolvers	no	boolean	no	

After completion of (adaptive) smoothing, save details of the convolvers used to files named in parameters `outconvolversset` and `outindeximageset`. See section 3.3.4 for a description of how these files can be used.

outconvolversset	no	dataset	outconvolvers.ds	

If `writeconvolvers`='yes', the task writes images of all the convolvers to this dataset. It is recommended that `outindeximageset` and `outconvolversset` be the same dataset. See section 3.3.4 for a description of how this file can be used.

outindeximageset	no	dataset	outindeximage.ds	

If `writeconvolvers`='yes', the task writes an index image to this dataset. It is recommended that `outindeximageset` and `outconvolversset` be the same dataset. See section 3.3.4 for a description of how this file can be used.

inconvolversarray	no	array	inconvolvers.ds:CONV_000	

This parameter is read if `smoothstyle`='withset'. `inconvolversarray` is the name of a dataset + array which contains images of all the convolvers to be used. An index image specified in parameter `inindeximagearray` is also needed. See section 3.3.4 for further details.

withindeximagearray	no	boolean	yes	

This parameter is read if `smoothstyle`='withset'. It specifies whether to also look for an index image dataset+array.

inindeximagearray	no	array	inindeximage.ds:INDEXIMG	

This parameter is read if `smoothstyle`='withset' and `withindeximagearray`='yes'. `inindeximagearray` is the name of a dataset + array which contains an index image, which shows which convolver should be used to smooth which part of the input image in `inset`. Images of the convolvers are read from parameter `inconvolversarray`. See section 3.3.4 for further details.

normalizeset	no	boolean	no	

This parameter is read if `smoothstyle`='withset'. If set, the convolvers read from parameter `inconvolversarray` are divided by their array sums before use.

intemplateset	no	dataset	template.ds	

This parameter is read if `smoothstyle`='template' and contains the name of the template image to be read. See section 3.3.6 for further details.

nopslimit	no	real	500000	> 0

See developers' notes.

forcecalctype	no	boolean	no	

Force the use of either all direct processing or all Fourier-domain (see developers' notes).

calcbyfft	no	boolean	yes	

If `forcecalctype` is set, this parameter controls which of the two methods to use for the whole image: FFT convolution if true, direct convolution otherwise (see developers' notes).

readvarianceset	no	boolean	no	

If set, the task reads from parameter `invarianceset` an image of the variances (squares of the standard deviations) of the input image values.

invarianceset	no	dataset	invariance.ds	

The name of the dataset which contains an image of the variances (squares of the standard deviations) of the input image values.

writevarianceset	no	boolean	no	

If set, the task writes to parameter `outvarianceset` an image of the variances (squares of the standard deviations) of the output image values.

outvarianceset	no	dataset	outvariance.ds	

The name of the dataset to contain an image of the variances (squares of the standard deviations) of the output image values.

withweightset	no	boolean	no	

If this is set, the task reads an image of weights from parameter `weightset`. See section 3.4 for further details.

weightset	no	dataset	weight.ds	

The image of weights read by the task if `withweightset`='yes'. See section 3.4 for further details.

withoutmaskset	no	boolean	no	

If this is set, the task reads a mask image from parameter `outmaskset`. See section 3.4 for further details.

outmaskset	no	dataset	outmask.ds	

The mask image read by the task if `withoutmaskset`='yes'. All pixels of `inset` for which the mask is FALSE are left unchanged. `outmaskset` may be of any numeric data type: values > 0 translate to TRUE, the rest FALSE. See section 3.4 for further details.

writebadmaskset	no	boolean	no	

If this is set, the task writes to parameter `badmaskset` a logical-valued image which flags those pixels for which the weighted convolution could not be performed. See section 3.4 for further details.

badmaskset	no	dataset	badmask.ds	

The image of bad-result flags which is written by the task if `writebadmaskset`='yes'. See section 3.4 for further details.

withexpimageset	no	boolean	no	

If this is set, the task reads an exposure map from parameter `expimageset`. See section 3.5 for further details.

expimageset	no	dataset	expmap.ds	

The exposure map read by the task if `withexpimageset`='yes'. See section 3.5 for further details.

expmapuse	no	string	samesnr	samesize—samesnr

This parameter is read if `withexpimageset`='yes'. It governs the treatment of variance when adpative smoothing is desired. See section 3.5 for further details.

remultiply	yes	boolean	no	

This parameter is read if `withexpimageset`='yes'. If `remultiply`='yes', the output image is remultiplied by `expimageset` before it is saved to file; otherwise not. See section 3.5 for further details.

5 Errors

This section documents warnings and errors generated by this task (if any). Note that warnings and errors can also be generated in the SAS infrastructure libraries, in which case they would not be documented here. Refer to the index of all errors and warnings available in the HTML version of the SAS documentation.

outMaskWrongSize *(error)*
`outmaskset` and `inset` have different dimensions.

wholeImageUnmasked *(error)*
There are no TRUE-valued pixels in `outmaskset`.

weightImageWrongSize *(error)*
`weightset` and `inset` have different dimensions.

exposureMapWrongSize *(error)*
`expimageset` and `inset` have different dimensions.

weightImageNegativeValues *(error)*
Some of the pixels of `weightset` have negative values.

weightImageAllZero *(error)*
All the pixels of `weightset` are zero-valued.

varianceImageWrongSize *(error)*
`invarianceset` and `inset` have different dimensions.

varianceImageNegativeValues *(error)*
Some of the pixels of `invarianceset` have negative values.

inImageNegativeValues *(error)*
No variance image was supplied, so the task has tried to use the `inset` as its own variance image. However this is not possible because some of the pixels of `inset` are negative-valued.

badConvolverStyle *(error)*
The value of parameter `convolverstyle` was not recognized.

badExpMapUse *(error)*
The value of parameter `expmapuse` was not recognized.

badSmoothStyle *(error)*
The value of parameter `smoothstyle` was not recognized.

badUserWidths *(error)*
The list of convolver widths `userwidths` was found not to be monotonically increasing, as it should be.

badWidthRange *(error)*
The value of `maxwidth` was found to be less than or equal to `minwidth`.

tooManyConvolvers *(error)*
More than 999 convolvers were generated while trying to calculate a library of convolvers from the template image `intemplateset`.

badWidthScaleStyle *(error)*
The value of parameter `widthliststyle` was not recognized.

noConvolvers *(error)*
The user has set `smoothstyle`='adaptive' and `withuserwidths`='yes', but hasn't supplied any convolvers in `userwidths`.

convolverWidthsBadOrder *(error)*
The list of convolver widths `userwidths` was found not to be monotonically increasing, as it should be.

minConvolverWidthEqualsZero *(error)*
This won't work with `widthliststyle`=log - in this case the user must specify a `minwidth` greater than zero.

badIndexImage *(error)*
The maximum index found in the index image exceeds the number of convolvers.

zeroNormConvolver *(error)*
The sum of the convolver elements is zero. Such convolvers are not allowed.

indexImageWrongSize *(error)*
`inindeximagearray` and `inset` have different dimensions.

indexImageNegativeValues *(error)*
Some of the pixels of `inindeximagearray` have negative values.

uselessVarianceImage *(warning)*
The user has requested `writevarianceset`='yes', but the configuration of the task is such that no variance set has been calculated. That is, a variance set for the smoothed output is only calculated if `readvarianceset`='yes' or `smoothstyle`='adaptive'.
corrective action: A zero-valued variance set is written anyway.

outMaskTooNarrow *(warning)*
This happens if there are substantial zero-valued areas of `weightset` which are not masked by `outmaskset`. The user can avoid this by either reducing the size of the holes in `weightset` or increasing the coverage of `outmaskset`.
corrective action: Pixels where this occurs are set to zero in `outset` and flagged in `badmaskset`.

dynamicRangeTooLarge *(warning)*
The dynamic range is calculated by dividing max(abs(I)) by median(abs(I), I ne 0), where I is the weighted, exposure-corrected input image. If this value is larger than a cutoff (currently 40000), this warning is issued.
corrective action: No action.

onlyOneConvolver *(warning)*
The user has set `smoothstyle`='adaptive' and `withuserwidths`='yes', but has only supplied one convolver in `userwidths`.
corrective action: Task proceeds as normal.

negativeValuesInOutput *(warning)*
This warning only occurs if `inset`, `weightset` and all convolvers are everywhere non-negative. In this case one might expect that all values of the output image would also be non-negative. However, small negative values can sometimes occur if the FFT was used in the convolution. The user probably ought to check the output to make sure that it looks ok.
corrective action: Such pixels are set to zero.

negativeValuesInVarianceOutput *(warning)*
Small negative values can sometimes occur if the FFT was used in the convolution. Even if this was the case, the user probably ought to check the output variance image to make sure that it looks ok. If the FFT was *not* used then it probably indicates a bug and you should contact the code developer.
corrective action: Such pixels are set to zero.

6 Input Files

The input FITS images listed below need not be XMM images and all of them (even the mask) can be of any numeric data type output by **evselect**, eg int8, int16, int32, real32 or real64. All optional images (except the convolvers in `inconvolversarray`) must however be of the same dimensions as the `inset`.

1. (Mandatory) `inset`: the image to be smoothed.
2. (Optional) `inconvolversarray`: both this file and `inindeximagearray` are read when `smoothstyle` = 'withset'. `inconvolversarray` should contain a cube or 3-dimensional array which is a stack of convolver images. The first two dimensions of the cube must be the common x and y dimensions of the convolver arrays; the third dimension must equal the number of convolvers. Convolvers are assigned an index which is their position (starting with 1) along the third-dimension sequence. A convolver of index i is then used to smooth portions of the image for which the `inindeximagearray` value is also i.

   In the future this specification may be expanded to accommodate convolvers of varying array size.
3. (Optional) `inindeximagearray`: both this file and `inconvolversarray` are read when `smoothstyle` = 'withset'. `inindeximagearray` should contain a 2-dimensional array. The values of this array after rounding to the nearest integer are taken to refer to the convolvers in the list read from `inconvolversarray`. The ith convolver in this list is then used to smooth portions of the image for which the `inindeximagearray` value is also i.
4. (Optional) `intemplateset`: this file is read when `smoothstyle`='template'. It should contain a 2-dimensional array in the primary extension. The value of a given pixel of `intemplateset` is taken to be the characteristic width (sigma value) of the gaussian convolver to be used to smooth the corresponding pixel of `inset`. This facility doesn't offer any advantages over the `smoothstyle`='withset' and I will eventually delete it.
5. (Optional) `invarianceset`: this file is read when `readvarianceset`='true'. It should contain a 2-dimensional array in the primary extension. The array values should be the variances (ie, squares of standard deviations) in the values of `inset`. If the task needs these variances, but `readvarianceset`='no', the task assumes that `inset` is Poissonian and thus can be used as an approximation of its own variance.
6. (Optional) `weightset`: this file is read when `withweightset`='true'. It should contain a 2-dimensional array in the primary extension. The array values are used as the weights w in equation 1 or 4.
7. (Optional) `outmaskset`: this file is read when `withoutmaskset`='true'. It should contain a 2-dimensional array in the primary extension. The array values are translated to logicals by replacing values > 0 by TRUE, the rest FALSE. The array values perform as the δ values in equation 4.

   The parameter name is now (with the abolition of the corresponding 'inmask') is a little misleading and I think I'll replace it in the next version with plain 'maskset'.

8. (Optional) `expimageset`: this file is read when `withexpimageset`='true'. It should contain a 2-dimensional array in the primary extension. The array values should record the exposure of `inset`, if this is a well-defined quantity.

7 Output Files

All outputs are arrays in FITS datasets.

1. `outset`: The smoothed image. This has data type REAL32. Attributes (and DSS, if present) are copied from the input image.
2. (Optional) `outtemplateset`: this is only available from `smoothstyle`='adaptive'. The array has data type REAL32. The value of a given pixel of `outtemplateset` is the characteristic width (sigma value) of the gaussian convolver which was used to smooth the corresponding pixel of `inset`. This facility doesn't offer any advantages over the `smoothstyle`='withset' and I will eventually delete it.
3. (Optional) `outconvolversset`: this is only available from `smoothstyle`='adaptive' and is written in conjunction with `outindeximageset`. The sequence of convolver images used in the adaptive smoothing is written to a cube or 3-dimensional array of REAL32 data type, named `CONV_000`. (It is recommended that `outindeximageset` and `outconvolversset` be the same dataset.) Convolver arrays which are smaller than the largest convolver array are padded to the maximum size with zeros. The first two dimensions of the cube record the now common x and y dimensions of these arrays; the third dimension records their position in the sequence.
4. (Optional) `outindeximageset`: this is only available from `smoothstyle`='adaptive' and is written in conjunction with `outconvolversset`. The index image is written to a 2-dimensional array, of INTEGER32 data type, named `INDEXIMG`. (It is recommended that `outindeximageset` and `outconvolversset` be the same dataset.) Its values correspond to the sequence position of the corresponding convolver: the ith convolver in the sequence stored in `outconvolversset` was used to smooth those portions of the image for which the `outindeximageset` value is also i.
5. (Optional) `outvarianceset`: this records the variances (squares of standard deviations) in the smoothed image. Equation 2 shows how these values are calculated. The array has data type REAL32.
6. (Optional) `badmaskset`: this ought to be a boolean array but the best I can manage at the moment is INTEGER8. Non-zero values are taken to indicate TRUE. Values are set TRUE where the task was unable to evaluate the fraction in equation 4 because the denominator was close to zero.

8 Algorithm

```
read parameters;
read --inset;

if (--withoutmaskset) {
  read --outmaskset;
} else {
```

```
  outMask = TRUE;
}

if (--withweightset) {
  read --weightset;
} else {
  weightImage = 1;
}

weightImage = weightImage / maxval(weightImage)

(xyLimits) = &findWidthOfBlackBorder(inImage);
inImage     = &cutBorder(inImage,     xyLimits);
outMask     = &cutBorder(outMask,     xyLimits);
weightImage = &cutBorder(weightImage, xyLimits);

if (--withexpimageset) {
  read --expimageset;
  expImage = &cutBorder(expImage, xyLimits);

  where(expImage>0) {
    inImage = inImage / expImage;
  } elsewhere {
    inImage = 0;
    weightImage = 0;
  }
}

# Choice of smoothing type:

if (--smoothstyle='simple') {
  if (--convolverstyle='gaussian') {
    convolvers(1) = &makeGaussianConvolver;
  } elsif(--convolverstyle='tophat') {
    convolvers(1) = &makeTopHatConvolver;
  } elsif(--convolverstyle='squarebox') {
    convolvers(1) = &makeBoxConvolver;
  }
} elsif(--smoothstyle='template') {
  (indexImage, convolvers) = &makeConvolversFromTemplate;
} elsif(--smoothstyle='withset') {
  (indexImage, convolvers) = &readConvolversFromSets;

} elsif(--smoothstyle='adaptive') {

  # Adaptive smoothing calculation of convolvers:

  if (--readvarianceset) {
    read --invarianceset;
    varianceImage = &cutBorder(varianceImage, xyLimits);
  } else {
    if (--withexpimageset) {
      varianceImage = inImage * expMapImage;
      # Because inImage has already been divided by expMapImage, and we
      # need to reverse that.
```

```
  } else {
    varianceImage = inImage;
  }
}

if (--withexpimageset) {
  if (--expmapuse='samesnr') {
    where(expImage>0) {
      varianceImage = varianceImage / expImage / expImage;
    } elsewhere {
      varianceImage = 0;
    }
  } else {
    where(expImage>0) {
      varianceImage = varianceImage / expImage / maxval(expImage);
    } elsewhere {
      varianceImage = 0;
    }
  }
}

convolvers = &calculateConvolverLibrary;

where(outMask) {
  indexImage = 1;
} elsewhere {
  indexImage = 0;
}

smoothedImage = 0.0;
rmsSmoothedImage = 0.0;

i = 1;
smoothedImage    = smoothedImage         + &patchSmooth(convolvers(i)
  , weightImage, mask=(indexImage==i));
rmsSmoothedImage = rmsSmoothedImage + sqrt(&patchSmooth(convolvers**2(i)
  , weightImage, mask=(indexImage==i)));

where(indexImage == i) {
  where(! failureMask) {
    where(rmsSmoothedImage>0 && smoothedImage>0) {
      bestSnr = smoothedImage / rmsSmoothedImage;

      where(bestSnr > desiredSnr) {
        # This means that we are still
        # smoothing too hard, and need to try this pixel again with a
        # narrower gaussian. Therefore increment the index:

        indexImage = i + 1;
      }
    }
  }
}

foreach(i = 2, numConvolvers) {
```

```
    secondBestSnr = bestSnr;

    smoothedImage    = smoothedImage         + &patchSmooth(convolvers(i)
      , weightImage, mask=(indexImage==i));
    rmsSmoothedImage = rmsSmoothedImage + sqrt(&patchSmooth(convolvers**2(i)
      , weightImage, mask=(indexImage==i)));

    where(indexImage == i) {
      where(failureMask) {
        indexImage = i - 1; # This pixel didn't fail for the previous
        # value of index (=> next larger gaussian), otherwise it would
        # never have got here. Hence we will drop the index value for
        # this pixel back to this last known 'good' value.

      } elsewhere {
        where(rmsSmoothedImage>0 && smoothedImage>0) {
          bestSnr = smoothedImage / rmsSmoothedImage;
        } elsewhere {
          bestSnr = 0;
        }

        where(bestSnr > desiredSnr) {
          # This means that we are still
          # smoothing too hard, and need to try this pixel again with a
          # narrower gaussian. Therefore increment the index:

          indexImage = i + 1;
        } elsewhere { # go to the value that gave the best SNR:
          where(abs(secondBestSnr-desiredSnr)<abs(bestSnr-desiredSnr)) {
            indexImage = i - 1;
            # Otherwise, leave indexImage at i.
          }
        }
      }
    }
  }

  where(indexImage > numConvolvers) {indexImage = numConvolvers;}
}

# Do the smoothing:

smoothedImage = 0.0;
where(outMask) {smoothedImage = inImage;}
failureMask = FALSE;
foreach(i = 1, numConvolvers) {
  patchMask = outMask && (indexImage == i);

  # The actual convolution:
  foreach(xi = outStartX, outFinisX) {
    foreach(yi = outStartY, outFinisY) {
    next if (! outMask(xi, yi));

      summ = 0.0;
      weight = 0.0;
```

```
      foreach(cxi = 1, convolverXSize) {
        xxi = (cxi - 1 - halfConvolverXSize) - xi;
        foreach(cyi = 1, convolverYSize) {
          yyi = (cyi - 1 - halfConvolverYSize) - yi;
        next if (weightImage(xxi, yyi) <= 0.0);
          summ   = summ   + convolver(cxi, cyi) *     inImage(xxi, yyi);
          weight = weight + convolver(cxi, cyi) * weightImage(xxi, yyi);
        }
      }

      if (weight >= minAllowedWeight) {
        outImage(xi, yi) = summ * norm / weight;
      } else {
        outImage(xi, yi) = 0.0;
        failureMask(xi, yi) = TRUE; # indicates those pixels where the
        # weight is too small.
      }
    }
  }
}

# Outputs:

if (--writevarianceset) {
  outVarianceImage = &calculateOutVariance(inImage, weightImage, outMask
    , indexImage, convolvers);

  outVarianceImage = &restoreBlankBorder(outVarianceImage, xyLimits);
  write to --outvarianceset;
}

where(failureMask) {smoothedImage = 0;}

if (--writebadmaskset) {
  failureMask = &restoreBlankBorder(failureMask, xyLimits);
  write to --badmaskset;
}

if (--smoothstyle='adaptive') {
  if (--writetemplateset) {
    templateImage = &calculateTemplateImage;
    templateImage = &restoreBlankBorder(templateImage, xyLimits);
    write to --outtemplateset;
  }

  if (--writeconvolvers) {
    indexImage = &restoreBlankBorder(indexImage, xyLimits);
    write indexImage to --outindeximageset;
    write convolvers to --outconvolversset;
  }
}

if (--withexpimageset) {
  if (--remultiply) {
    smoothedImage = smoothedImage * expMapImage;
```

```
  }
}

smoothedImage = &restoreBlankBorder(smoothedImage, xyLimits);

write to --outset;
```

9 Comments

-

References