colimplot

August 16, 2005

Abstract

The task produces image plots, colour coded to represent the relative intensities in either two or three spectral bands.

1 Instruments/Modes

Instrument	Mode
EPIC MOS:	IMAGING
EPIC PN:	IMAGING
OM:	IMAGING

2 Use

pipeline processing	yes
interactive analysis	yes

3 Description

3.1 Introduction

This task is designed to produce colour plots, the displayed colour at any pixel being a function of the relative intensities in the supplied energy/wavelength bands. Although designed originally to display EPIC X-ray images, it can also be used to create false-colour plots from OM images. The assignment of colour attempts to mimic that of visible light, ie objects bright at long wavelengths are coloured red, those bright at short wavelengths, blue.

The task takes for its input any number of FITS images, each made in a different band of energy or wavelength. However at least 2 images are necessary for colour rendering. If the colour coding of x-ray hardness is to be preserved, the images should be listed on the command line in order of increasing energy or hardness (ie of decreasing wavelength for OM images).

The created colour image may either be stored in one of the file formats accessible via the pgplot library, or as a 3-plane fits file, or as a ppm image. The last format can be converted to gif or png by use

of external (eg netpbm) utilities which bypass the 256-colour-name pgplot limitation on the number of colours available.

Processing of the input is divided into two broad sections: manipulations of the colour values of each pixel; and distributing the pixel colours among a much less numerous set (256 or even less) of pgplot-allowed colour indices.

3.2 Colour manipulations

3.2.1 Dividing by the reference multiplet

After the N input files are read and confirmed to be of the same size, they are combined in a single array; each array pixel being thus associated with a multiplet of N flux values derived from the inputs. The first processing step is to scale these multiplets with respect to a reference multiplet. The derivation of the reference multiplet is carried out in one of two basic styles: an 'internal' style in which a kind of average multiplet is calculated from the values in the image themselves, without reference to any other information, or a 'black body' (bb) style in which the reference multiplet is calculated using the widths of the energy bands (in the case of EPIC images) or the filter bandpasses (in the case of the OM). (Future developments may encompass a third style in which the reference multiplet values are entered directly by the user.) In the black body case the reference multiplet is calculated from the integrated effective area (EPIC) or integrated filter bandpass × QE (OM), as the multiplet expected from a black body (BB) at at user-definable temperature.

These style choices are controlled firstly by the parameter `refclrstyle`, which can be either 'internal', 'bb' or 'none'. If this is 'internal', the user can further specify via the parameter `internalrefstyle` whether the reference multiplet is to be calculated by taking the mean or the rms of the image multiplet. Only pixels having a fractional total flux greater than `cutoff` are used in this calculation. However if `refclrstyle`='bb', the user can specify (through the parameter `bbrefstyle`) the BB curve either directly as a temperature or via the energy or wavelength of the photons occupying the peak of the BB energy spectrum. (Good physics students will not need to be reminded that the peak of the BB curve plotted against energy is not in the same location as the peak of the curve when plotted against wavelength!)

For any of the 'bb' reference multiplet styles, **colimplot** needs access to the boundaries of the energy or wavelength bands within which each of the input images was accumulated. The task looks first in the data subspace (DSS); if nothing is found there it looks for `BAND` attributes; if these aren't found either, an error is raised. The 'bb' style also requires CCF access in order to calculate the integrated effective area etc, hence the environment variable SAS_CCF should be properly set if this style is desired.

Note that **colimplot**, although designed to operate on XMM images, will accept any fits image. However only the 'internal' style of reference multiplet calculation is available for non-XMM images, because these are not likely to store the information about energy bands in the XMM manner.

3.2.2 Forcing to 'thermal' colours

It is desirable that the output image should look 'right'. This is a psychological term and therefore has a degree of slipperiness of definition. However to my eye the colour scheme that looks best is what might be described as a 'thermal' one, ie one in which cool objects (objects with a long-wavelength excess) are coloured red and hot objects blue, with the intermediate sequence going from red through yellow and white to blue. Greens and purples look wrong to my eye, in excess anyway. So the next part of the colour manipulations is to ensure that all colours fall on this 'thermal' sequence. This is not difficult in the case that 2 input images are supplied, since these 2 degrees of freedom can fairly easily be mapped onto the

2 degrees of freedom, namely brightness and temperature (or analogue thereof), of the 'thermal' scheme. But where there are 3 input images, one of the three degrees of input freedom must to a large degree be thrown away. (The three-image facility is retained however because there may be occasions when one *wants* the weird colours.)

The details of processing are probably not of much interest to the user; more important to know is that this forcing is controlled by the parameters `weirdness`, `heat` and `heatspread`. A `weirdness` value of 0 causes nothing to be done to the data; a value of -1 forces all pixels to have strictly thermal colours; whereas a value of +1 does the opposite, ie *none* of the consequent pixel colours are 'thermal', they are all strange purples, greens and cyans. The parameter `heat` shifts colours along the thermal sequence: at `heat` = -1 all colours go to red, at +1 they are all blue, with again 0 having no effect. The third parameter, `heatspread`, stretches (positive values) or compresses (negaive values) colours along the thermal sequence.

At the end of this processing, each pixel is associated with a triplet of red, green and blue (RGB) values, each of which can have a value between 0 and 1.

3.2.3 Flux scaling

This is achieved via the parameters `gainstyle`, `usergain`, `pixelfraction`, `tofluxfraction` and `nhotpixels`. If `gainstyle`='user', the gain is entered directly. A gain value of 1 will leave the image in the default situation in which the pixel with the largest net flux is saturated, ie has an RGB with at least one of its three values equal to 1.0. However it is often the case that the brightest few pixels in an image are contributed by a single strong source, or even that they are 'hot' pixels of the ccd, and that most of the sources of interest are considerably fainter. In this case it is desirable to multiply all the pixel values by a gain value > 1 in order to increase the brightness.

The algorithm which is employed to apply the gain is more complicated than just a simple multiplication by a scalar gain value. It is necessary to ensure that all RGB values remain less than or equal to 1. A hyperbolic transform

$$f_{\rm new} = \frac{g\ f_{\rm old}}{1 + f_{\rm old}(g-1)}, \tag{1}$$

where $f = I/I_{\rm max}$ is flux fraction and g is the gain, is used to accomplish this. The effect of this for small flux values is to approximately multiply them by g, but $f_{\rm old}$ values just below 1 never become larger than 1 for any positive gain value. The colour saturation of each pixel decreases smoothly with increase in gain, but the hue is maintained at a constant value.

Since images can vary widely in the difference between their brightest pixel and the brightness of the features of interest, it can be difficult to choose a correct value for the gain without several time-consuming trials. The task contains two mechanisms that help make choosing the gain easier. Firstly, the user can set the expected number of 'hot' pixels via the parameter `nhotpixels`. Say this value is set to N. The image is then pre-scaled so that the saturation level is no longer set to the flux of the brightest pixel, but the Nth brightest. An under-estimate of `nhotpixels` can result in an image which is mostly way too dim, but a mild overestimate is fairly innocuous.

The second feature is complete automatic scaling, actuated by setting `gainstyle` to 'auto'. In this setting the task attempts to calculate the best gain value itself. At present this is computed depending upon the distribution of pixels as a function of flux, and is designed to scale the image so that the background is just visible. The user can exert some control over this via the parameters `pixelfraction` and `tofluxfraction`. This is done as follows. In the case that `gainstyle`='auto', **colimplot** does two things: first, it creates (internally) a histogram of numbers of image pixels against net flux; using the histogram, it then estimates a value of net flux, call it $I(p)$, such that `pixelfraction` pixels fall below this

value. The value of gain is then chosen so that $I(p)$ becomes equal to `tofluxfraction` of the saturated level of net flux. From equation 1, this is given by

$$g = \frac{f_{\rm tff}\ (1 - f(p))}{f(p)\ (1 - f_{\rm tff})}$$

where $f_{\rm tff}$ = `tofluxfraction` and $f(p) = I(p)/I_{\rm max}$. For example, suppose that `pixelfraction`=0.5 and `tofluxfraction`=0.4. Further suppose that, after constructing the histogram, **colimplot** estimates that the flux fraction $f(0.5)$ in the unscaled image is 0.25. In other words, the dimmest 50% of pixels are found to have net flux values of less than 0.25 of the maximum net flux value. The necessary gain is then

$$g = \frac{0.4\ (1 - 0.25)}{0.25\ (1 - 0.4)}$$

ie, equal to 2.0.

3.3 Distribution of colours

Since XMM images are typically several hundred pixels on a side, there can be tens of thousands of separate colours on the image. Unfortunately the pgplot library allocates only 8 bits to colour names and hence is limited to 256 colours. If pgplot output is desired, a way must therefore be found to bin up the thousands of colours into at most 256. To see how this is done, it helps to visualise the colours as points inside a cube with edges 1 unit in length. Each pixel is associated with a different point in the cube. The task of allocating these colours is equivalent to constructing 256 boxes inside the cube, each box to contain a subset of the thousands of colour points. Clearly it is inefficient to have boxes that either overlap or that are much larger than the spread of the points they contain. The boxes are assigned within the task using an algorithm called a kd-tree. At present the boxes are allocated so that they all tend to be of equal size, but it may make more sense to allocate them so that they all tend to contain equal numbers of points. This may be explored in a future version of **colimplot**.

There is one parameter that affects this process, namely `rgbsysstyle`. If this is set to 'polar', the cartesian RGB triplets are transformed to cylindrical polar coordinates, the z value representing net flux or brightness, the radius r representing colour saturation and the angle θ being proportional to the hue. Use of this basis for binning up the colours helps to prevent jitter in the average hue of the boxes with increase in brightness over the final image, since the pixels that correspond to a single source, for instance, tend to be aligned closely in their values of r and θ. It may be of use to investigate other transforms which help to maximize the fineness of separation of colours in the middle range of brightness values.

3.4 Alternative output

The parameter `pgdev` allows the user to specify a pgplot device, but, in contrast to the pgplot standard, the file name should not be included, but instead supplied via the parameter `plotfile`. Some examples of values of `pgdev` and `plotfile`, and their interpretations:

- To display to an xserve device:

  `pgdev=/xs`

- To write a GIF file called myfile.gif:

```
pgdev=/gif plotfile=myfile.gif
```

- To write a colour postscript file called op.ps:

```
pgdev=/cps plotfile=op.ps
```

However note that at present, if a colour postscript device (`pgdev` = '/cps'), a fits file (`pgdev` = '/fits') or a ppm file (`pgdev` = '/ppm') are selected, the task in fact bypasses pgplot and writes the postscript file directly. This is because all the pgplot devices use at largest an 8-bit colour palette: this does not result in optimum plots. Writing the output files directly, on the other hand, allows one to use a much larger number of colours.

3.5 Image rebinning

In the usual case that the image dimensions are hundreds of pixels, **colimplot** can take quite a while to run if a pgplot output is selected. On the other hand it is often necessary to make several trials before one arrives at parameter values that produce the desired output, because the initial appearence on any pgplot output device may not at first be very pleasing, due partly to the aforementioned limit on the number of colours available. This adds up to an irritating situation for the user. A way around this is to produce an intermediate output with a reduced number of image pixels, via the parameters `rebinimage`, `rebinstyle`, `dividexby`, `divideyby`, `newnxbins` and `newnybins`. The way these work is probably best illustrated by a couple of examples. So, to rebin the intermediate image to 40 in the x direction by 50 in y, use the following values:

```
rebinimage=yes
rebinstyle=tonumber
newnxbins=40
newnybins=50
```

If, on the other hand, you wish to rebin by dividing the number of x pixels by 3.3 and the number of y by 10, use the following values:

```
rebinimage=yes
rebinstyle=divideby
dividexby=3.3
divideyby=10
```

Note that it is only possible to decrease the resolution of the image. An error will result if the requested number of bins in either direction is larger than the relevant dimension of the input images.

3.6 Masking

XMM images typically contain quite a lot of pixels which are outside the field of view of the instrument. These contain zero values and therefore result in a lot of black ink being used in hard copy output. It may be desirable to the user to print or display these out-of-FOV pixels as white, or even to manipulate the image size so as to maximise the relative size of those parts within the FOV. To do this, an exposure map can be loaded as a mask, or the mask can be deduced from the input images themselves. The parameter that enables this functionality is `withmask`. If this is set true, the user can specify the colour of the unmasked bits at the edges of the image via the parameter `greylevel`. If parameter `maskstyle`='user',

the name of a mask set is expected to be supplied via `maskset`; if `maskstyle`='self', the task attempts to use the input images for this purpose. The final relevant parameter, `expandtomask`, controls whether the full image is displayed or whether it is expanded until the masked central section just touches the borders of the display.

4 Parameters

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

Parameter	Mand	Type	Default	Constraints
insets	yes	dataset list		

List (in order of increasing hardness) of input image sets.

withmask	no	boolean	yes	yes—no

If 'yes', the task attempts to access or create a mask, depending on the setting of `maskstyle`. Parameters `greylevel` and `expandtomask` are also enabled by this switch.

maskstyle	no	string	self	self—user

If 'self', the task attempts to create the mask by using the input images. If 'user', the task looks for the name of the mask image set in the parameter `maskset`.

maskset	no	dataset	mask.ds	

The name of the mask set (usually an exposure map).

greylevel	no	real	0.0	0.0 $\leq$ `greylevel` $\leq$ 1.0

The grey level to paint those areas of the output image that fall outside the mask. Where the background level of the image is visible, the default value (= black) is preferable; however, in the case that large areas of the image fall outside the mask (but see also the parameter `expandtomask`), a value of 1.0 may be better, if only to save ink on printed output.

expandtomask	no	boolean	yes	yes—no

When set, that part of the image where the mask is true is expanded until it just fits into the frame. This is to prevent large areas of non-mask and therefore blank image from being displayed.

rebinimage	no	boolean	no	yes—no

Rebin the image to the specifications dictated by the parameters `dividexby` and `divideyby` or `newnxbins` and `newnybins`.

rebinstyle	no	string	tonumber	tonumber—divideby

This parameter controls the style in which the image rebinning occurs. If 'tonumber', the parameters `newnxbins` and `newnybins` are read; if 'divideby', the parameters `dividexby` and `divideyby` are read.

dividexby	no	real	5.0	1.0 $\leq$ `dividexby`$\leq$ 50.0

Divide the number of x pixels by this value, and convert to integer; the result is the number of x pixels in the rebinned image.

divideyby	no	real	5.0	1.0 ≤ `divideyby`≤ 50.0

Divide the number of y pixels by this value, and convert to integer; the result is the number of y pixels in the rebinned image.

newnxbins	no	integer	100	10 ≤ `newnxbins`≤ 1000

The number of x pixels in the rebinned image.

newnybins	no	integer	100	10 ≤ `newnybins`≤ 1000

The number of y pixels in the rebinned image.

negremovalstyle	no	string	truncate	truncate—offset

If negative values are detected in any of the input images, the task removes them by whichever of these two methods is specified.

refclrstyle	no	string	internal	internal—bb—none—user

Dictates the style in which the reference multiplet is calculated.

internalrefstyle	no	string	rms	rms—mean

If `refclrstyle`='internal', this is read, and controls whether the reference multiplet is calculated from the root mean squared (rms) or the mean of all pixels which have a total flux above the cutoff value (see parameter `cutoff`).

cutoff	no	real	0.05	0.0 ≤ `cutoff` ≤ 1.0

Pixels which have a total flux which is less than `cutoff` times the maximum total flux are not included in calculations of the reference multiplet. The purpose of this is to prevent such calculations being skewed by background values, which usually dominate an image in terms of numbers of pixels involved.

bbrefstyle	no	string	kev	kev—angstroms—kelvin

For `refclrstyle`='bb', the value of the reference multiplet is set to the image multiplet that would have been observed from a black-body source with a given temperature. The different values possible to `bbrefstyle` just allow the user some flexibility in entering the temperature. For `bbrefstyle`='kelvin' the temperature is entered directly in millions of degrees kelvin; for `bbrefstyle`='kev' the peak of the BB curve is given in keV; whereas for `bbrefstyle`='angstroms' the peak is given in angstroms. Note that these peak values refer to the peak of the BB spectrum as a function of energy or frequency, rather than wavelength.

whitekev	no	real	2.0	0.01 ≤ `whitekev`≤ 20.0

Enabled if `refclrstyle`='bb' and `bbrefstyle`='kev'. It is the energy of a photon at the peak of the reference black body spectrum.

whiteangstroms	no	real	100.0	0.01 ≤ `whiteangstroms`≤ 8000.0

Enabled if `refclrstyle`='bb' and `bbrefstyle`='angstroms'. It is the wavelength of a photon at the peak of the reference black body spectrum.

whitemegakelvin	no	real	1.0	0.01 ≤ whitemegakelvin≤ 100.0

Enabled if refclrstyle='bb' and bbrefstyle='kelvin'. It is the temperature of the reference black body spectrum.

refclr	no	real list		> 0

List of 3 RGB weights to use for reference colour.

weirdness	no	real	-0.7	−1.0 ≤ weirdness≤ 1.0

This parameter exerts control over the colour values of the plot. Values of weirdness that approach -1 give output colours in the so-called 'thermal' sequence, ie that are similar to those acquired by heated black bodies; values that approach 1 give highly non-thermal colours such as greens and violets.

heat	no	real	0.0	−1.0 ≤ heat≤ 1.0

This parameter exerts control over the colour values of the plot. Smaller values of heat make all the pixels 'cooler' in the thermal sequence of colours (ie redder); larger values in contrast 'heat up' the colour values, ie make them bluer.

heatspread	no	real	0.3	−1.0 ≤ heatspread≤ 1.0

This parameter exerts control over the colour values of the plot. Smaller values of heatspread pull all the pixels in towards white, larger values spread them out more along the thermal sequence of colours.

rgbsysstyle	no	string	polar	cartesian—polar

This parameter controls whether the colour points are described in a RGB basis or in a brightness-saturation-hue basis when an 8-bit colour palette is shared out.

gainstyle	no	string	auto	user—auto

If 'user', a value of gain can be entered directly via the parameter usergain. If gainstyle='auto', the task attempts to calculate it such that the background is just bright enough to be visible.

usergain	no	real	8.0	0.0 < usergain

The value by which the net flux values should be multiplied before display.

pixelfraction	no	real	0.5	0.0 < pixelfraction≤ 1.0

For gainstyle='auto', the task calculates the gain required to bring the maximum net flux fraction of the dimmest pixelfraction pixels to the value of tofluxfraction.

tofluxfraction	no	real	0.4	0.0 < tofluxfraction≤ 1.0

For gainstyle='auto', the task calculates the gain required to bring the maximum net flux fraction of the dimmest pixelfraction pixels to the value of tofluxfraction.

nhotpixels	no	integer	10	0 ≤ nhotpixels

The image brightness is scaled so as to saturate the nhotpixels brightest pixels.

pgdev	no	string	/cps	

The pgplot device name.

plotfile	no	string	test.ps	

If the pgplot device is one that requires an output file, this gives the name of the file.

withframe	no	boolean	no	yes—no

If `pgdev`='/ppm' and `withframe`='yes', the task constructs a frame plot around the image, containing various pieces of information such as the name of the observer and the target. This is written to a .gif file named frame.gif, which can be combined with the output image by **colimchain**.

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.

badPgDev *(error)*
The `pgdev` string is not in the correct <optional outFilename>/<device> format.

zeroUnsatFrac *(error)*
The value of `unsatfrac` must be greater than zero.

unknownEband *(error)*
The `BAND` attribute of one of the input image arrays has an unrecognised value.

zeroWhiteFlux *(error)*
One of the three RGB values of the reference colour equals zero. (This can happen if `kevofwhite` is too small.)

badEbandOrder *(error)*
The images are not listed in order of increasing energy band.

differentSizes *(warning)*
The three input images do not all have the same size.
corrective action: The smaller images are padded with zeros to the size of the largest.

totallySaturated *(warning)*
The whole image is saturated.
corrective action: Run the task again with a larger value of `unsatfrac`.

zeroClrVolume *(warning)*
The entire image has just one colour.
corrective action: A monochrome plot is produced, but this won't be very interesting. Figure out the problem and run the task again with better images.

sameEband *(warning)*
Two or more of the images have the same value of the `BAND` keyword.
corrective action: The task proceeds anyway - maybe the user wants it this way.

colimplot is designed to work with images produced from XMM data, but any FITS images can be used, provided that they meet the required data type criteria.

6 Input Files

1. Any number of FITS datasets. These must have primary image arrays of the same size. They can be of assorted data types from any of the set output by **evselect**, namely int8, int16, int32, real32 and real64.

7 Output Files

One of the following:

1. An image in one of the PGPLOT file formats.
2. An image directly written in postscript.
3. An image directly written as a 3-plane FITS image.
4. An image directly written in ppm (Portable Pixel Map) format.

8 Algorithm

```
# Read in the three input files.

# Rebin these files if desired.

# Load the mask set if desired.

foreach (pixel) {
  netFlux(pixel) = 0.0
  foreach (band) {
    netFlux(pixel) = netFlux(pixel) + image(band)(pixel)
  }
}
maxNetFlux = maxval(netFlux)

# Calculate the reference multiplet 'ref'.

foreach (pixel) {
  foreach (band) {
    image(band)(pixel) = image(band)(pixel) / ref(band)
  }
}

# Force to thermal -> RGB triplet for each pixel.

# Transform the net flux scale if desired.

# Recombine netFlux and RGB -> RGB.

# Saturation calculation.
```

```
# Final colour tweak.

# Separate the filename from the pgplot device 'pgDev'.

if (pgDev eq 'cps') {
  # Draw the postscript image.
} else
  if (rgbsysstyle eq 'polar') then
    # Transform RGB values to polar coordinates.
  end if

  # Assign colour values to 256 bins.

  if (rgbsysstyle eq 'polar') then
    # Transform bin colours back from polar coordinates.
  end if

  call pgimag(RGB)
}
```

9 Comments

References