Adaptive Binning with Weighted Voronoi Tesselations: WVT_BINNING

User's Manual

Steven Diehl

July 2005

Preface

This manual is intended to give a hands-on introduction on how to use the main features of the adaptive binning technique `WVT_BINNING`. Throughout the document, we will denote all pixel indices with the letters k and l. Bins will be denoted with i and j to avoid confusion. Accordingly, the signal and noise per pixel will be denoted as S_k and N_k, respectively. In addition, we will also use the slightly different calligraphic characters for bins: $\mathcal{S}_i$ and $\mathcal{N}_i$. All bins will be abbreviated with the letter $\mathcal{V}$.

Each chapter of this manual is divided into several sections, starting with a general description of the required input parameters, followed by some examples. Among those, we start with the easiest case and increase in complexity (but also correctness) toward the end. The last chapter contains a general description of keywords which can improve the performance of the algorithm.

The examples in this manual use various procedures or functions from the Astro IDL library, which is available at http://idlastro.gsfc.nasa.gov/homepage.html.

Table of Contents

Chapter 1

Desciption of the adaptive binning method (WVT_BINNING)

1.1 Modified bin accretion

Show pictures of example after bin accretion.

1.2 Redistribute unbinned pixels

Show pictures of example after redistribution.

1.3 WVT iteration

Show pictures of example after WVT is done. + S/N distribution

Wait with this chapter until the paper is done completely, since it will be mostly the same text to describe the method

Chapter 2

WVT_IMAGE: Intensity binning of X-ray images

2.1 General Description

WVT_IMAGE provides an interface to WVT_BINNING to produce adaptively binned X-ray images. These can be raw counts, exposure and/or background corrected images. The algorithm operates directly on 2-dimensional images, and has a straightforward basic syntax:

```
WVT_IMAGE, signal, noise, targetSN, binnedimage
```

Although WVT_IMAGE will work with only these few arguments, it is far more flexible. The complete syntax is the following:

```
WVT_IMAGE, signal, noise, targetSN, binnedimage, xnode, ynode, weight $
     [, snbin=snbin, mask=mask, ctsimage=ctsimage,
        binnumber=binnumber, binvalue=binvalue, center=center,
        plotit=plotit, resume=resume, save_all=save_all,
        max_area=max_area, gersho=gersho, keepfixed=keepfixed,
        quiet=quiet ]
```

For X-ray data, it is recommended always to supply the counts image (CTSIMAGE, s6.1.10), as well as the mask file (MASK, s6.1.9)

2.2 Main Parameters

Here is a short, general description of the main parameters. For a complete description of all the keywords, please refer to chapter 6 or the IDL files directly. Have a look at section 2.4

2.2.1 `SIGNAL` [input, required]

A two-dimensional image, containing the signal per pixel. The image can be background subtracted or exposure map corrected, as long as the NOISE image reflects this. If the 'pixels' are actually the apertures of an integral-field spectrograph, then the signal can be defined as the total flux in the spectral range under study, for each aperture.

2.2.2 `NOISE` [input, required]

Two-dimensional image (same size as `SIGNAL`), containing the noise associated with each pixel (sqrt(variance)).

2.2.3 `TARGETSN` [input, required]

The desired signal-to-noise ratio in the final 2D-binned data. A `TARGETSN` between $\sim 4 - 10$ is standard for X-ray images, a temperature map would require higher `TARGETSN` ($\sim 30 - 100$), depending on the spectral model used. For integral field spectroscopy, a `TARGETSN` of ~ 50 per bin may be a reasonable value to extract stellar kinematics information from galaxy spectra. In general, the higher `TARGETSN` is, the fewer bins will be computed, and thus the lower the resolution will be.

2.2.4 `BINNEDIMAGE` [output, required]

The final binned image will have the same size as the input image `SIGNAL`.

2.2.5 `XNODE, YNODE, WEIGHT` [output, optional]

The locations of the WVT bin generators, together with the associated weights for the WVT. This set of three 3 parameter values is sufficient to recon-

struct the complete binning scheme and/or to apply it to different data sets. This represents the most efficient way to save and/or distribute the binning structure.

2.3 Prescription to compute signal to noise

`WVT_IMAGE` assumes that the signal-to-noise ratio of a bin can be computed in the following manner:

$$(\mathcal{S}/\mathcal{N})_i = \frac{\sum_{k\in\mathcal{V}_i} \mathcal{S}_k}{\sqrt{\sum_{k\in\mathcal{V}_i} \mathcal{N}_k^2}}, \tag{2.1}$$

Note: If this is not the case for your type of data, you have to adjust the functions `ADD_SIGNAL` and/or `ADD_NOISE` as described in section A.

2.4 Applications

Although all of our examples are drawn from applications to X-ray data, we should emphasize that the algorithm is not restricted to X-ray analysis, but rather to any type of 2-dimensional data.

2.4.1 Binning of X-ray counts images

The easiest way to get a first impression of X-ray images is to have a look at the raw counts image C_k. Due to the sparse nature of X-ray data, it is in most cases necessary to bin the data in order to gain some kind of insights about the spatial distribution. Here, the signal and noise per pixel are defined as follows:

$$S_k = C_k \tag{2.2}$$

$$N_k = \sqrt{C_k}. \tag{2.3}$$

`WVT_IMAGE` will correctly compute the signal and noise in the bins according to equation 2.1. Thus, after simplifying equation 2.1, the S/N per bin can be expressed as

$$(\mathcal{S}/\mathcal{N})_i = \sqrt{\sum_{k\in\mathcal{V}_i} C_k}, \tag{2.4}$$

and binning to a constant target S/N is equivalent to binning to a constant number of counts per bin[1] : $\mathcal{C}_{\rm Target} = (\mathcal{S}/\mathcal{N})^2_{\rm Target}$

Example:

```
;Read in the counts image
ctsimage=mrdfits('img.fits',0, header)
signal=ctsimage
noise=sqrt(ctsimage)
targetSN=5d0  ; ~25 counts per bin

;Perform the binning, with target signal-to-noise of 5
wvt_image, signal, noise, targetSN, binnedimage

;Save the output in another fits image, keeping the WCS information (header)
mwrfits, binnedimage, 'abinned.fits', header
```

2.4.2 Binning of exposure corrected flux images

If one is interested in removing artifacts in the counts image due to instrument structures such as node boundaries or chip edges, one should use the exposure map E_k. In order to convert counts to physical units of photons $\sec^{-1}\,\mathrm{cm}^{-2}\,\mathrm{arcsec}^{-2}$ one has to divide by the exposure map and the effective exposure time[2].

$$S_k = C_k/E_k \tag{2.5}$$

$$N_k = \sqrt{C_k}/E_k. \tag{2.6}$$

[1] This is one of the few cases, where you can make use of the keyword `GERSHO` (see s6.1.3 for more details).
[2] For examples on how to use exposure maps, please refer to the official CIAO website

Example:

```
;Read in the counts image and exposure map
ctsimage=mrdfits('ctsimg.fits',0, header)
expmap=mrdfits('expmap.fits',0)
dim=size(ctsimage,/dimension)

; Create a mask from the exposure map (1=good, 0=bad),
; to avoid regions outside the chip boundaries
mask=intarr(dim[0],dim[1])
wh=where(expmap GT 0)
mask[wh]=1

; Create the flux image and the associated noise image
signal=dblarr(dim[0],dim[1])
noise=dblarr(dim[0],dim[1])
signal[wh]=ctsimage[wh]/expmap[wh]
noise[wh]=sqrt(ctsimage[wh]/expmap[wh]^2)

; Perform the binning, with target signal-to-noise of 5
targetSN=5d0
wvt_image, signal, noise, targetSN, binnedimage, mask=mask

; Save the output in another fits image, keeping the WCS
mwrfits, binnedimage, 'abinned.fits', header
```

2.4.3 Binning of background corrected images

Let's assume a spatially variable background B_k across the image that you know with a 1σ accuracy $\sigma_{B,k}$. Then, you can define the signal and noise in as

$$S_k = C_k/E_k - B_k \tag{2.7}$$

$$N_k = \sqrt{C_k/E_k^2 + \sigma_{B,k}^2}. \tag{2.8}$$

There are various possibilities for how to find such a background: The easiest way is to use blank sky fields, normalized to the appropriate exposure time. These images can either be subtracted directly, or a fit can be used alternatively (uniform background, tilted plane, etc.). Another option is to use a local background on the same chip, derived from source free regions, or a surface brightness profile fit.

Example:

```
; Read in fluxed signal and noise (no background correction)
ctsimage=mrdfits('cts.fits',0, header)
expmap=mrdfits('expmap.fits',0)
dim=size(signal,/dimension)

; Create a mask from the exposure map (1=good, 0=bad),
mask=intarr(dim[0],dim[1])
wh=where(expmap GT 0)
mask[wh]=1

; Define a flat background value (from a model fit, etc.)
bgvalue=1d-10
sigma_bg=0.1*bgvalue ; assume a 10% error on the bg value (example)

; Create the flux image and the associated noise image
signal=dblarr(dim[0],dim[1])
noise=dblarr(dim[0],dim[1])
signal[wh]=ctsimage[wh]/expmap[wh]-bgvalue
noise[wh]=sqrt( ctsimage[wh]/expmap[wh]^2 + sigma_bg^2 )

; Perform the binning, with target signal-to-noise of 5
targetSN=5d0
wvt_image, signal, noise, targetSN, binnedimage, ctsimage=ctsimage, mask=mask

; Save the output in another fits image, keeping the WCS
mwrfits, binnedimage, 'abinned.fits', header
```

2.4.4 Isolating components of linear combinations of images

In X-ray astronomy, images are often composed of various components that contribute in spectrally distinct ways to the total image. Thus, the image can be considered as a linear combination of these components. A good example is the X-ray emission of elliptical galaxies, where one is often faced with the problem of removing the contribution of unresolved point sources to the diffuse emission. We will use this particular example to demonstrate the capabilities of the binning algorithm, although it can easily be generalized to other problems.

To remove the contribution of unresolved point sources, we use the fact that the hot gas and point sources contribute differently to the soft and the hard bands. Let F_S and F_H represent the background-subtracted and exposure map corrected

soft and hard images. We can express both in terms of the unresolved point source emission P, the gas emission G, and their respective softness ratios γ and δ:

$$F_S = \gamma P + \delta G \tag{2.9}$$

$$F_H = (1-\gamma)P + (1-\delta)G. \tag{2.10}$$

The uncontaminated gas image is then given by

$$G = \frac{1-\gamma}{\delta-\gamma}\left[F_S - \left(\frac{\gamma}{1-\gamma}\right) F_H\right]. \tag{2.11}$$

For more details and instructions on how to determine the constants gamma and delta with spectral models, please refer to Diehl and Statler (2005). If we use the gas image G_k as our signal S_k, we have to define the noise accordingly.

Depending on your specific background values for F_S and F_H and their respective uncertainties, it could happen that you create "artificially" high S/N values in pixels that contain no counts. To avoid these artifacts, always supply the counts image of the combined bands (`CTSIMAGE`). See s6.1.10 for more details.

$$S_k = G_k \tag{2.12}$$

$$N_k = \frac{1-\gamma}{\delta-\gamma}\sqrt{\sigma_{S,k}^2 + \left(\frac{\gamma}{1-\gamma}\right)^2 \sigma_{H,k}^2} \tag{2.13}$$

Example:

```
;Read in the soft and hard bands
hard=mrdfits('hard.fits',0, header)
hard_noise=mrdfits('hardnoise.fits',0)
soft=mrdfits('soft.fits',0)
soft_noise=mrdfits('softnoise.fits',0)

; Define your constants gamma and delta
alpha=.5
delta=.9

;Isolate the gas emission
signal=(1d0-gamma)/(delta-gamma)*(soft-(gamma/(1d0-gamma))*hard)
noise=(1d0-gamma)/(delta-gamma)* $  $
        sqrt(softnoise^2+(gamma/(1d0-gamma))^2*hardnoise^2)

;Perform the binning, with target signal-to-noise of 5
targetSN=5d0
wvt_image, signal, noise, targetSN, binnedimage, ctsimage=ctsimage

;Save the output in another fits image, keeping the WCS
mwrfits, binnedimage, 'abinned.fits', header
```

Chapter 3

WVT_XRAYCOLOR: Binning of Hardness Ratios

3.1 General Description

Another useful tool in X-ray astronomy is the generation of so-called color maps. An X-ray "color" is generally defined as the quotient between the fluxes in two different bands A and B. Depending on the choice of energy bands, this "hardness ratio" map can be used as diagnostics for temperature gradients or photoelectric absorption features for example (Sanders and Fabian 2001). A general discussion about the physical interpretation of these maps and an appropriate choice of bands can be found in Fabian et al. (2000), for example.

The basic, required syntax to use WVT_XRAYCOLOR successfully, is as follows:

```
WVT_XRAYCOLOR, signal, signal2, noise, noise2, targetSN, binnedimage
```

The complete, more flexible syntax with all keywords (§6) is given here:

```
WVT_XRAYCOLOR, signal, signal2, noise, noise2, targetSN, binnedimage, xnode, ynode, weight $
  [, snbin=snbin, mask=mask, ctsimage=ctsimage, binnumber=binnumber, $
     binvalue=binvalue, center=center, plotit=plotit, resume=resume, $
     save_all=save_all, max_area=max_area, keepfixed=keepfixed ]
```

Depending on your specific background values for the soft and hard band, and their respective uncertainties, it could happen that you create "artificially" high

S/N values in pixels that contain no counts. To avoid these artifacts, always supply the counts image (`CTSIMAGE`) of the combined bands. See s6.1.10 for more details.

3.2 Main Parameters

3.2.1 `SIGNAL` [input, required]

A two-dimensional image, containing the signal per pixel for band A. The image can be background subtracted or exposure map corrected, as long as the NOISE image reflects this.

3.2.2 `NOISE` [input, required]

Two-dimensional image (same size as `SIGNAL`), containing the noise associated with each pixel (sqrt(variance)) for band A.

3.2.3 `SIGNAL2` [input, required]

The equivalent to `SIGNAL` for band B.

3.2.4 `NOISE2` [input, required]

The equivalent to `NOISE` for band B.

3.2.5 `TARGETSN` [input, required]

The desired signal-to-noise ratio in the final 2D-binned data. A `TARGETSN` between $\sim 4 - 10$ is standard for X-ray color images, but will in general depend on what type of features you want to show. As always, the higher `TARGETSN`, the fewer detail you will see.

3.2.6 `BINNEDIMAGE` [output, required]

The final binned image will have the same size as the input images.

3.2.7 `XNODE, YNODE, WEIGHT` [**output, optional**]

The locations of the WVT bin generators, together with the associated weights for the WVT. This set of three 3 parameter values is sufficient to reconstruct the complete binning scheme and/or to apply it to different data sets.

3.3 Prescription to compute signal to noise

Here, our signal consists of the hardness ratio of the two flux values. We define $F_{A,k}$ and $F_{B,k}$ as the flux per pixel k in the bands A and B, respectively. Thus, we can write our signal as

$$\mathcal{S}_i = \frac{\sum_{k\in\mathcal{V}_i} F_{A,k}}{\sum_{k\in\mathcal{V}_i} F_{B,k}}. \tag{3.1}$$

The associated error in the hardness ratio can be expressed in terms of the noise in the individual bands:

$$\mathcal{N}_i = \mathcal{S}_i \sqrt{\frac{(\sum_{k\in\mathcal{V}_i} \sigma^2_{A,k})}{(\sum_{k\in\mathcal{V}_i} F_{A,k})^2} + \frac{(\sum_{k\in\mathcal{V}_i} \sigma^2_{B,k})}{(\sum_{k\in\mathcal{V}_i} F_{B,k})^2}} \tag{3.2}$$

If necessary, these definitions can be extended for the general use of n different bands (for more details refer to Sanders and Fabian 2001).

3.4 Applications

3.4.1 Count hardness ratios

The easiest way to compute an X-ray color map is to use simply count hardness ratios between the counts in band A and B (C_A and C_B, respectively):

$$\mathcal{S}_i = \frac{\sum_{k\in\mathcal{V}_i} F_{A,k}}{\sum_{k\in\mathcal{V}_i} F_{B,k}}, \tag{3.3}$$

$$\mathcal{N}_i = \mathcal{S}_i \left(\frac{1}{\sum_{k\in\mathcal{V}_i} C_{A,k}} + \frac{1}{\sum_{k\in\mathcal{V}_i} C_{B,k}} \right)^{-1/2} \tag{3.4}$$

This is good to get a first impression of the general structure of the target, but shouldn't be used for the final analysis, since changes in the fractional contribution

of the background, as well as energy dependent features of the exposure map can introduce unreal artifacts.

In this simpler case, the S/N ratio is reduced to:

$$\mathcal{S}_i/\mathcal{N}_i = \left(\frac{1}{\sum_{k \in \mathcal{V}_i} C_{A,k}} + \frac{1}{\sum_{k \in \mathcal{V}_i} C_{B,k}} \right)^{-1/2} \tag{3.5}$$

Example:

```
; Read in soft and hard counts images (with header!) and the mask
soft=mrdfits('softcts.fits',0, header)
hard=mrdfits('hardcts.fits',0)
softnoise=sqrt(soft)
hardnoise=sqrt(hard)
ctsimage=soft+hard

; Bin the color map to a signal-to-noise ratio of 5
targetSN=5d0
wvt_xraycolor, soft, hard, softnoise, hardnoise, targetSN, $
  binnedimage, ctsimage=ctsimage

; Save the color map
mwrfits, binnedimage, 'abinned.fits', header
```

3.4.2 Background corrected flux hardness ratios

A better way to compute hardness ratios, is to use background and exposure corrected fluxes instead of counts. In our example, we will use a flat background for each band. However, we could have used a spatially dependent background model or a blank sky image as well (see also §2.4).

Example:

```
; Read in soft and hard counts images (with header!)
softcts=mrdfits('softcts.fits',0, header)
hardcts=mrdfits('hardcts.fits',0)

; Read in the exposure maps for soft and hard band
softexpmap=mrdfits('expmap_soft.fits',0)
hardexpmap=mrdfits('expmap_hard.fits',0)

; Define the background values (assumed flat) and their uncertainties
softbg=1d-10
hardbg=2d-10
softbgsigma=0.1*softbg
hardbgsigma=0.1*hardbg

; Compute the fluxed images and their associated noise
softflx=softcts/softexpmap - softbg
softnoise=sqrt( softcts/softexpmap^2 + softbgsigma^2 )
hardflx=hardcts/hardexpmap - hardbg
softnoise=sqrt( hardcts/hardexpmap^2 + hardbgsigma^2 )
ctsimage=softcts + hardcts

; Create a mask from the exposure maps:
mask=(softexpmap GT 0) AND (hardexpmap GT 0)

; Bin the color map to a signal-to-noise ratio of 5
targetSN=5d0
wvt_xraycolor, softflx, hardflx, softnoise, hardnoise, targetSN, $
    binnedimage, mask=mask, ctsimage=ctsimage

; Save the color map
mwrfits, binnedimage, 'abinned.fits', header
```

Chapter 4

WVT_PIXELLIST: Binning of pixel lists

YET TO COME:

- Include example from IFS data by Cappellari and Copin

Chapter 5

Spectral maps with SHERPA/CIAO

In this chapter, I will briefly describe how to use `WVT_BINNING` to generate two-dimensional maps of spectral parameters. I will focus on the most common application in X-ray astronomy, namely the creation of temperature maps. However, the scripts are completely general and can be easily modified for the use with any spectral parameter (e.g. metallicity, absorption, etc.).

The "cookbook" section will give a brief overview about how to apply the scripts to a typical example. The following section gives detail on how to use each script individually and explains its functionality.

This chapter is not intended to explain the usage of CIAO, Sherpa or S-lang tools or scripts. The intention is rather to give you an example on how one can use the spatial binning for spectral fitting using these systems. A basic familiarity with all the former languages will be necessary to modify these scripts to suit your needs. You might also decide to trust me and use them as a black box. The scripts were written and tested with CIAO 3.1 and CALDB 2.28. on a Sun Ultra80 station running SunOS 5.8. They might be compatible with other unix systems or CIAO versions, though I cannot guarantee its proper functionality there.

5.1 Cookbook

Before you start using the scripts, it is necessary to follow the general setup that this script requires. First, the event2 file should be in the working directory and

named evt2.fits. There should also be a mask file mask.fits that contains 1 for good data and 0 for bad data. Look at the examples in chapter 2 for ideas on how to create such a mask. One should also have the bad pixel file in the directory or a parallel ../primary or ../secondary directory, for the CIAO script `acis_set_ardlib` to find it. You will also need a copy of your ciao.csh (in CIAO/bin) in this directory.

```
# 1) General Setup

# 1.0) Copy event 2 file and bpix files into the working directory,
# and create the mask file

# 1.1) Unzip the tempmap package in the working directory
tar -xvzf tempmap.tar.gz

# 1.2) Sky coordinates to derive an image from, must match the mask file
setenv XMIN             3400
setenv XMAX             4500
setenv YMIN             3700
setenv YMAX             4800

# 1.3) Start CIAO
source ciao.csh -o

# 1.4) Execute the ciao script acis_set_ardlib
chmod +x acis_set_ardlib
punlearn ardlib
./acis_set_ardlib

# 1.5) Create the counts image
dmcopy "evt2.fits[bin x=${XMIN}:${XMAX}:1,y=${YMIN}:${YMAX}:1]" \
  cts.fits clobber=yes verbose=2
```

```
# 2) Start IDL batch file wvt_temperaturemap to bin the image and extract
# event sublists for each bin
idl -queue wvt_temperaturemap

# 3) Start the csh-script tempmap_generic.csh.
# 3.1) If you decide to only use one rmf and arf file for the whole chip,
# i.e. you want to neglect the spatial dependence of the instrument
# response, create a file called "use_one_rmf":
touch use_one_rmf
# If this is not the case, the script will create a new rmf file for each bin.
# For a large number of bins, this will be very time consuming!

# 3.2) Modify the files redshift.dat and n_h.dat which contain the values for
# the redshift and the column density in the APEC model, which are read in
# during the fitting process and held fixed in this example.
echo 0.0002925 > redshift.dat
echo 1.6200000 > n_h.dat

# 3.3) Execute the automatic fitting script
chmod +x tempmap_generic.csh
./tempmap_generic.csh

# 4) Evaluate the output to create a temperature map
idl -queue wvt_evaltemperaturemap
```

5.2 Details of Single Steps

5.2.1 General setup

The temperature map script requires the following files to be present in the working directory:

- `evt2.fits`: The file containing the event 2 list of X-ray photons. For computational reasons, it might be useful to restrict the event list to the CCD that you are analysing.
- `*bpix*.fits`: Observation specific bad pixel file. The `acis_set_ardlib` scripts will set the parameters for the RMF and ARF generators later, which will need this information to function correctly.

- `cts.fits`: The counts image that has to be analysed. Here is an example on how to create this image from the evt2 file:

```
punlearn dmcopy
dmcopy "evt2.fits[bin x=3400:4500:1,y=3000:4100:1]" cts.fits clobber=yes verbose=2
```

- `mask.fits`: File containing the mask for the field of analysis. Here is an example on how to create a mask file from an exposure map and a CIAO region file:

```
# Create an empty image with the same dimensions and world coordinate system
# as your image file
dmimgcalc cts.fits cts.fits zeroimage.fits sub clobber=yes verbose=2
# Create the mask from the exposure map, by setting everything above 40% of
# the exposure map's peak value to 1, the rest to 0
punlearn dmimgthresh
dmimgthresh infile=zeroimage.fits outfile=exposure_mask.fits \
  expfile=expmap.fits cut=":40%" value=1 clobber=yes verbose=2
# Cut out the region that is contained in excludeme.reg
dmcopy "exposure_mask.fits[exclude sky=region(excludeme.reg)]" mask.fits clobber=yes verbose=2
```

- `mark_bgd.pi`: The background spectrum file. This can be a extracted from a local background region on the same chip, or taken from the Markevitch background compilation, as shown in this example.

```
acis_bkgrnd_lookup $EVTFILE
setenv BGFILE `pget acis_bkgrnd_lookup outfile`
# Attention: in this example, I do not ensure that the gain files of the bg and
# evt2 file match. See the CIAO homepage for more details.
cp $BGFILE bg.fits
# Reproject background events to fill the sky column correctly
punlearn reproject_events
reproject_events infile="bg.fits[cols -time]" outfile=bg_reproj.fits \
  aspect=pcad_asol1.fits match=evt2.fits random=0 clobber=yes verbose=2
# Fill the time column
dmlist "bg.fits[#row=1][cols time]" data,clean >temp.dat
sed -e 1d temp.dat >time.dat
setenv TIME `sed -e 's/ //g' time.dat`
punlearn dmtcalc
dmtcalc infile=bg_reproj.fits outfile=bg_final.fits expr="TIME=$TIME" clobber=yes verbose=2
# Create the bg pi spectrum
punlearn dmextract
dmextract infile="bg_final.fits[bin pi]" outfile=mark_bgd.pi
```

- `wvt_*.pro`: The WVT binning suite of programs should be in the current directory or in a directory that is included in you $IDLPATH variable.
- `acis_set_ardlib`: CIAO script that sets parameters necessary for ARF and RMF generation.
- `ciao.csh`: A copy of your PATH_TO_CIAO/bin/ciao.csh file.
- `acisspec`: CIAO script that creates ARF and RMF.
- `fitsingletemp.sl`: Sherpa S-lang script that does the automatic spectral fitting.
- `is_file.sl`: Sherpa S-lang script that determines if a file exists or not.
- `n_h.dat, redshift.dat`: Files that are used during fitting and contain the Galactic value for the column density and redshift of the target.

5.2.2 `WVT_TEMPERATUREMAP`: Create the binning and extract event lists

`wvt_temperaturemap.pro` is actually an IDL batch file, rather that a procedure or function. The reason for this is that you can execute it from within an

automated script with the simple command

```
idl -queue wvt_temperaturemap
```

The -queue option ensures that idl automatically waits for a license to be available before starting, so you avoid the interactive question to wait for a license.

The default setup is such that `WVT_TEMPERATUREMAP` reads in the counts image (`cts.fits`) and bins it to a constant signal-to-noise per bin value of 30. This corresponds to binning to 900 (30^2) counts per bin. The sophistication of your spectral model determines what the value should be in your situation. In spectral fitting, it is recommended to bin the spectrum within Sherpa to a minimum of 20 counts later, i.e. 900 counts would correspond to about 45 spectral bins. This will decrease depending on your background contribution to the counts. If you want to take the background into account, edit `wvt_temperaturemap.pro` (see chapter 2) for more detail.

After binning, the program will automatically save the results and split the evt2.fits file up into several files. The files with the names `evt2.fits.X` contain the evt lists for the bins X. The corresponding `evt2.fits.X.BACKSCAL` files have the BACKSCAL factor (i.e. the relative size of the bin X), needed by Sherpa to get the normalization of the spectrum correct. The program will also create a file called `nbins.dat`, that simply specifies the number of bins that were created.

5.2.3 `tempmap_generic.csh`: Fit a spectral model with `Sherpa`

The main work is done by the C-shell script `tempmap_generic.csh` and the S-lang script `fitsingletemp.sl`. First, the CIAO procedure `dmextract` is evoked to create a spectrum from the event list of the current bin. The CIAO script `acisspec` is used to create the weighted ARF and RMF, corresponding to the bin region. If a file named `use_one_rmf` is present, only the ARF and RMF of the first bin will be computed and used for all other bins, in order to save time. If this file is not present, the ARF and RMF will be computed separately for each bin. After this step, the header of the spectrum file is automatically updated to include the correct BACKSCAL (area), ANCRFILE (ARF) and RESPFILE (RMF) keywords.

`fitsingletemp.sl` lets Sherpa automatically read in the bin source spectrum, the background spectrum, the response, and parameter files. Then it adaptively bins the spectrum to a minimum of 20 counts per spectral bin and fits an APEC (newer version of MEKAL) model to the spectrum by minimizing the χ^2 deviation. The temperature is then saved in the ascii file `evt2.fits.X.temp`, its positive and negative 1σ error bounds in `evt2.fits.X.tempcov`. Please note that so far, there are no "backup" or restart features built in for non-convergent fits. For cases where the fit converges onto a non-physical solution, i.e. where the temperature parameter bounds are stepped over or the fit didn't converge, the temperature will be simply reported as -1.

5.2.4 `WVT_EVALTEMPERATUREMAP`: Generate the temperature map

Again, `wvt_evaltemperaturemap.pro` is an IDL batch file that reads in the fitted temperature values from `evt2.fits.X.temp` and applies it to the saved binning scheme. This creates the 2-dimensional temperature map `tempmap.fits`

Chapter 6

Detailed description of parameters

Most keywords are common to the main binning algorithm, as well as their different interfaces. The brackets indicate, for which algorithm the keyword is valid.

6.1 Keywords

6.1.1 `PLOTIT` [all]

This keyword regulates the amount of graphical output during the session. The default value is "`PLOTIT=0`", i.e. no output. Set this keyword to 1 (either via "`PLOTIT=1`" or "`/PLOTIT`") to produce a plot of the two-dimensional bin distribution and of the corresponding S/N at the end of the computation. A value of "`PLOTIT=2`" will produce a similar plot after the bin accretion step. Setting `PLOTIT` to 3, will result in renewing the plots after each iteration. Having a file named "`plotit`" in the working directory, has the same effect as setting "`PLOTIT=3`". Note that plotting can significantly slow down the speed of the binning algorithm, since `WVT_BINNING` operates on pixel lists. Thus, the plotting procedure is designed to plot each pixels individually and can take rather long depending on the graphical capabilities of your machine.

6.1.2 `QUIET` [all]

By default the program shows the progress while accreting pixels and then while iterating the CVT ("`QUIET=0`"). Set this keyword to avoid printing progess

results. Be aware that the binning algorithm can run rather long in some cases, depending on image size and bin sizes. Having text output does not negatively affect the speed of the algorithm.

6.1.3 GERSHO [WVT_BINNING, WVT_IMAGE, WVT_PIXELLIST]

WVT_BINNING is based on an algorithm based on unweighted, centroidal Voronoi tesselations (CVT Cappellari and Copin 2003), which exploits a property of CVTs, known as Gersho's conjecture. If you set the GERSHO keyword, the output will be very similar to the output of Cappellari & Copin's code VORONOI_2D_BINNING, except for some modifications in the bin accretion steps. However, be aware, that Gersho's conjecture is *only* valid for strictly positive data, where the S/N adds in quadrature!

6.1.4 SAVE_ALL [WVT_IMAGE, WVT_XRAYCOLOR, WVT_PIXELLIST]

Set this keyword to a variable (will be in structure format) that holds all information that is necessary in order to restart the program from any given point. Simply supply the SAVE_ALL output and set the RESUME keyword. Its contents will be overwritten with the updated binning information at the end. If this keyword is supplied, *all other input information will be ignored.*

Example:

```
; Run wvt_image and save everything in
wvt_image,signal,noise,targetSN,binnedimage, save_all=save_all

; Save the structure save_all in a fits file
mwrfits, save_all, 'abinned_data.fits'
```

Have a look at the RESUME keyword to find out to restart from this point.

6.1.5 RESUME [all]

Set this keyword, if you want to start from an existing WVT, which is uniquely defined by the vectors XNODE, YNODE, and SNBIN. The WVT iteration scheme will be applied to the supplied WVT, and the values overwritten with the final output. If you are using one of the interfaces (WVT_IMAGE, WVT_XRAYCOLOR, or WVT_PIXELLIST),

you should use this keyword in conjunction witht the structure SAVE_ALL.

Example:

```
; Restore the old session
save_all=mrdfits('abinned_data.fits',1)

; Pick up from this point and restart wvt_image
wvt_image,signal, noise, targetSN, binnedimage, save_all=save_all, /resume
```

6.1.6 KEEPFIXED [all]

Optional input vector containing x and y coordinates of bin generators that you want to keep fixed in their position. The binning algorithm will move all other bins around as usual. The size of this vector should be: (2, # of fixed bins). Example use: Keep one bin fixed on the center of a galaxy.

Example:

```
; Define the center
center=[512.,512.]

; Run wvt_image, keeping the center fixed
WVT_IMAGE, signal, noise, targetSN, binnedimage, center=center, keepfixed=center
```

6.1.7 CENTER [WVT_IMAGE, WVT_XRAYCOLOR]

Optional input vector (size: 2) containing x and y values for the center. If this keyword is *not* supplied, [0,0] will be assumed as the center and the algorithm will start at the highest S/N pixel in the bin accretion step. If the center is given, bin accretion will start at the center. For work with pixel lists (WVT_BINNING or WVT_PIXELLIST), the X and Y coordinates should already have the center coordinates subtracted.

6.1.8 MAX_AREA [all]

Optional scalar, specifying a maximum bin size (in square pixels). We generally recommend the use of this keyword. bins. Essential in cases where there is

essentially no signal in a certain region (otherwise, the empty bins will "eat" into the region with signal) or if spatial resolution is more important than a smooth S/N distribution. This boundary is only approximate, but bins stay in general within a few percent. *Attention*: In area where the bin size hits `MAX_AREA`, the algorithm does no longer enforce a uniform S/N anymore. Thus, be careful when interpreting the resulting images. Always interpret in conjunction with the output S/N map (see also `SNBIN` keyword).

Example:

```
; Adaptively bin the signal with a maximum bin size of a circle
; with a radius of approximately 25pixel:
max_area=!pi*25.^2
WVT_IMAGE, signal, noise, targetSN, binnedimage, max_area=max_area,$
  snbin=snbin, binnumber=binnumber

; Create an S/N map from the bin distribution and its associated S/N
; values
sn_image=snbin[binnumber-1]
```

6.1.9 `MASK` [`WVT_IMAGE`, `WVT_XRAYCOLOR`]

Optional input, two-dimensional image (same size as the signal and noise images). The `MASK` specifies, which pixels should be included in the WVT binning algorithm. Valid pixels have to be designated as "1", excluded pixels as "0" (integer or byte). If your detector does not fill the field of view, this keyword is essential, otherwise you will attempt to bin empty regions.

Example:

```
; Read in the exposure map
expmap=mrdfits('expmap.fits',0)

; Create a mask from the complete exposure map
mask=expmap GT 0

; To avoid large fluctuations in the border region of your
; fluxed image, you can use this simple prescription instead:
mask=expmap GT .1*max(expmap)

; To remove point sources from the binning, use the
; ''source cell mask'' (e.g. output of CIAO's wavdetect algorithm)
cellmask=mrdfits('cell.fits',0)
mask[where(cellmask GT 0)]=0

; Supply the mask as a keyword to the binning algorithm
WVT_IMAGE, signal, noise, targetSN, binnedimage, mask=mask
```

6.1.10 CTSIMAGE [WVT_IMAGE, WVT_XRAYCOLOR]

Optional input, two-dimensional image (same size as the signal and noise images) containing the counts per pixel. Often necessary for very sparse X-ray data, where some pixels can contain no counts. In certain cases, you can then "artificially" have a high S/N value for these empty pixel, e.g. when you ; subtract two background corrected components (example: remove the point source contribution in ellipticals) or compute X-ray colors. Supply the counts image in order to avoid that the binning algorithm produces bins without any counts. In case of producing a color image from two distinct X-ray bands, one should supply the combined counts image of both bands, as this image gives information about where the signal is located.

Example:

```
; Read in the signal and noise (here: fluxed image and variance)
signal=mrdfits('fluximg.fits',0, header)
noise=sqrt(mrdfits('fluximg_variance.fits',0))

; Read in the associated counts image
ctsimage=mrdfits('ctsimg.fits',0)

;Perform the binning, with target signal-to-noise of 5
targetSN=5d0
wvt_image, signal, noise, targetSN, binnedimage, ctsimage=ctsimage

;Save the output in another fits image, keeping the WCS
mwrfits, binnedimage, 'abinned.fits', header
```

6.1.11 BINNUMBER [all]

Optional output (size: n_{pixels} for WVT_PIXELLIST and WVT_BINNING, [n_x,n_y for WVT_IMAGE and WVT_XRAYCOLOR) that contains the indices of the bin distribution. The labels range from 1 to n_{bins}, 0 is reserved for pixels outside the field of view, as specified by the MASK keyword.

Example:

```
; Perform the binning, with target signal-to-noise of 5
targetSN=5d0
wvt_image, signal, noise, targetSN, binnedimage, binnumber=binnumber

; Save the bin distribution over the field of view,
mwrfits, binnumber, 'abinned_binnumbers.fits'
```

6.1.12 SNBIN [all]

Optional output vector (size: n_{bins}) that keeps track of the S/N for all individual bins. In conjunction with BINNUMBER (see s6.1.11), this can be used to create a map of the S/N distribution.

Example:

```
; Perform the binning, with target signal-to-noise of 5
targetSN=5d0
wvt_image, signal, noise, targetSN, binnedimage, $
  snbin=snbin, binnumber=binnumber

; Save the distribution of SN over the field of view,
; remember that binnumber starts at 1!
snbin=[0,snbin]
mwrfits, snbin[binnumber], 'abinned_snr.fits'
```

6.2 External Interactive Controls

6.2.1 FILE "plotit" [all]

A good way to check the progress of the adaptive binning algorithm is to have graphical output on the way (see also the PLOTIT keyword). If you decide that you want to plot the iteration steps, create a file in the current directory with the name 'plotit'. A simple way to do this is to issue the command 'touch plotit' in a shell. Remove the file if you want to stop plotting. This is useful when you want to decide if stopping the binning algorithm prematurely will give you useful results (see also "stopmenow").

6.2.2 FILE "stopmenow" [all]

Sometimes, it is useful to be able to stop the algorithm manually. In cases where computing time is an issue, you need to shut down the system, etc. If you want to terminate the iteration at the next possible time, simply create a file named 'stopmenow' in the current directory (e.g ; 'touch stopmenow'). If necessary, you can resume the stopped session later with the help of the RESUME and SAVE_ALL keywords.

Chapter 7

Sample Output

7.1 Text Output

The following example uses a subset of a Chandra observation of the Perseus cluster core. The data are freely available on our WVT website in the package `test_wvt_image.tar.gz` . In this section we will go through the example step by step and include all text output that you will see. The optional graphical output is described in the following section.

```
IDL> ; Run the adaptive binning algorithm
IDL> wvt_image,image,noise,targetSN,binnedimage,xnode,ynode,snbin=snbin,$
IDL>     mask=mask, ctsimage=ctsimage, binnumber=binnumber, binvalue=binvalue, $
IDL>     center=center, save_all=save_all, max_area=max_area, $
IDL>     keepfixed=keepfixed
% Compiled module: WVT_IMAGE.
Bin-accretion...
...making neighbor list...
           0 pixels done
       10000 pixels done
       20000 pixels done
       30000 pixels done
       40000 pixels done
       50000 pixels done
       60000 pixels done
```

This is the start of the binning process, where each pixel gets its associated neighbor list computed. Every 10000 steps, you will get a line noting the progress of the procedure.

```
Bin accretion started at the specified center:
First bin starts at       127.000      127.000
bin:        1 | S/N:   10.11 | n_pixels:     215 |   0.33 % done
bin:        2 | S/N:   10.01 | n_pixels:     171 |   0.59 % done
bin:        3 | S/N:   10.05 | n_pixels:     116 |   0.77 % done
bin:        4 | S/N:   10.02 | n_pixels:     123 |   0.95 % done
bin:        5 | S/N:   10.02 | n_pixels:     229 |   1.31 % done
[...]
bin:      366 | S/N:   10.07 | n_pixels:     207 |  97.57 % done
bin:      367 | S/N:   10.00 | n_pixels:     258 |  97.98 % done
bin:      368 | S/N:   10.04 | n_pixels:     239 |  98.36 % done
bin:      369 | S/N:    8.50 | n_pixels:     248 |  98.74 % done
bin:      370 | S/N:   10.00 | n_pixels:     306 |  99.21 % done
bin:      371 | S/N:    8.24 | n_pixels:     138 |  99.94 % done
607 initial bins.
Reassign bad bins...
371 good bins.
```

This is the output of the bin accretion step. The columns denote the current number of the accepted bins, the signal-to-noise value of the bin, its size in pixels and the total fraction of pixels that have been binned. After the bin accretion step, all pixels of "bad bins" that didn't meet S/N or roundness criteria are reassigned to its closest neighbor.

```
(Extremely) modified Lloyd algorithm...
Initial WVT done.
Iteration 2,  pixels that switched bins: 1.83868%
Iteration 3,  pixels that switched bins: 1.28479%
Iteration 4,  pixels that switched bins: 1.02386%
Iteration 5,  pixels that switched bins: 0.849915%
[...]
Iteration 70,  pixels that switched bins: 0.00152588%
Iteration 71,  pixels that switched bins: 0.00152588%
Iteration 72,  pixels that switched bins: 0.00000%
Iteration converged to a stable WVT solution
72 iterations.
Fractional S/N scatter (%) around target S/N:       4.4022827
Average S/N:       10.137795
Fractional S/N scatter (%) around average S/N:       4.3424460
```

The modified Lloyd algorithm represents the "heart" of the algorithm, where the binning scheme continuously tries to reach a more uniform and self-consistent bin distribution. Each iteration names the number and percentage of bins that have

switched bins. It is often advisable to stop the algorithm before one of the convergence criteria is fulfilled to save computing time. However, this depends on your analysis goal.

7.2 Graphical Output

Figure 7.1 give an example on what a typical graphical output from `WVT_BINNING` looks like. The top panel shows the two-dimensional distribution of bins, each bin colored differently. The crosses indicate the locations of the bin generators. The bottom panel shows the signal-to-noise distribution of the final bins, excluding those with less than 2 pixels in size, as well as those reaching `MAX_AREA` in size.

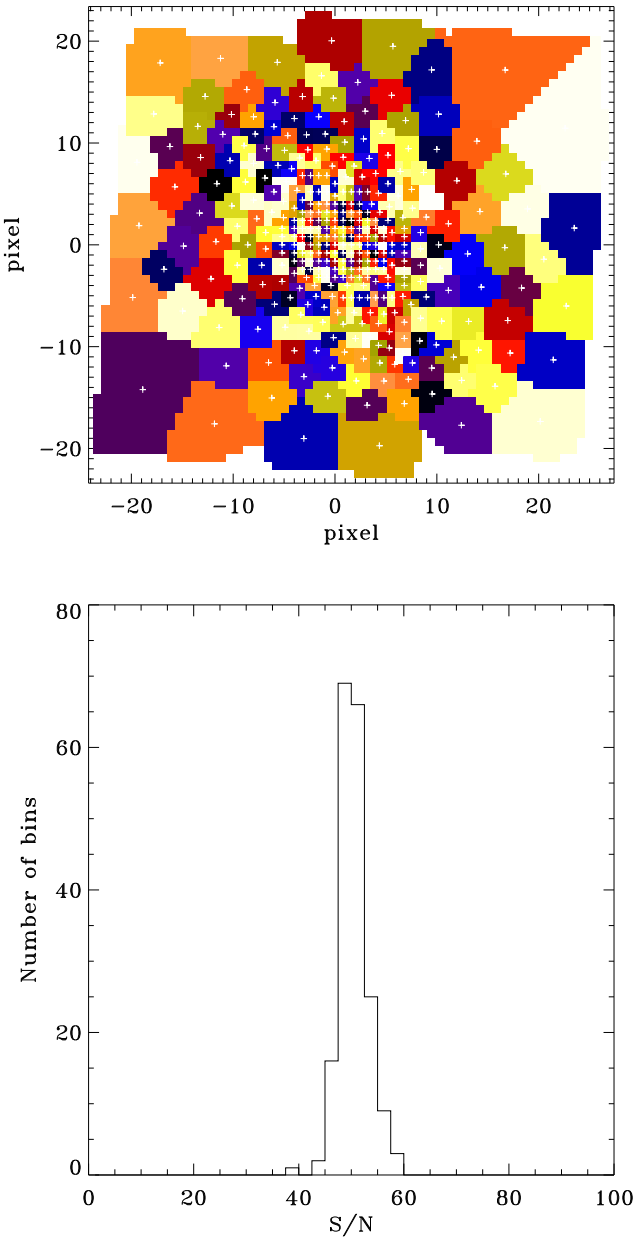

Figure 7.1: Top: Bin distribution, bin colors are random; Bottom: signal-to-noise distribution, excluding bins with less than 2 pixel and bins approaching `MAX_AREA`

Chapter 8

Caveats

- The bin accretion algorithm is based on a neighbor search to increase speed. Thus, bins are not able to cross gaps in the data (e.g. due to chip boundaries, readout streaks, etc.) in the bin accretion stage.
- The algorithm can run rather slowly for large images with large bins. We recommend the use of the keyword `MAX_AREA` in this case.
- The functions `ADD_SIGNAL` and `ADD_NOISE` are part of the wrappers `WVT_IMAGE`, `WVT_XRAYCOLOR` and `WVT_PIXELLIST` and not compatible with each other. So you have to recompile the program you want to use first to make sure you use the correct function. The wrappers have checks built in to warn you and will exit if you use incorrect functions.

Bibliography

M. Cappellari and Y. Copin. Adaptive spatial binning of integral-field spectroscopic data using Voronoi tessellations. *MNRAS*, 342:345–354, June 2003.

S. Diehl and T. S. Statler. An x-ray gas fundamental plane for elliptical galaxies. *ApJ*, 2005.

A. C. Fabian, J. S. Sanders, S. Ettori, G. B. Taylor, S. W. Allen, C. S. Crawford, K. Iwasawa, R. M. Johnstone, and P. M. Ogle. Chandra imaging of the complex X-ray core of the Perseus cluster. *MNRAS*, 318:L65–L68, November 2000.

J. S. Sanders and A. C. Fabian. Adaptive binning of X-ray galaxy cluster images. *MNRAS*, 325:178–186, July 2001.

Appendix A

WVT_GENERIC: Adopting the binning algorithm to your needs

A.1 General Description

The structure of the main binning algorithm `WVT_BINNING` was held in a very general way, in order to make the algorithm more flexible and applicable to a broader variety of problems. The file `wvt_generic.pro` contains a template on how to adjust the algorithm to your own needs. It is also very instructive to have a look at `wvt_image.pro`, `wvt_xraycolor.pro` and `wvt_pixellist.pro` and to use them as guidance.

For `WVT_BINNING`, all information that you need in order to calculate the combined signal of the bin, has to be contained in a global variable named `P`. `P` has to be a structure containing scalars and/or vectors, which are generally of length n_pixels. The functions `ADD_SIGNAL` and `ADD_NOISE` determine how to compute the combined signal and noise for the bin. The only parameter given to these function are the indices of the bin members, locating the correct pixel properties in the arrays of `P`.

The next section will give you a hands-on tutorial on how to implement this.

A.2 Step-by-step Guide

Step 1: The variable P

In order to compute the signal-to-noise ratio of a bin, one needs two types of information: The properties of all pixels and the information on which pixels belong to the bin in question. In step 1, we define a globally accessible structure P to deal with the first issue. P stores all of the pixel properties and constants necessary to compute the signal-to-noise ratio of a bin.

```
COMMON DATAVALUES, P
P={ pixprop1:  dblarr(npix),   $
    pixprop2:  fltarr(npix),   $
    pixprop3:  lonarr(npix),   $
    constant1: 3.142d0     ,   $
    constant2: 9L          }
```

The number of tags (here: 5), as well as their names (here: pixprop1, pixprop2, pixprop3, constant1 and constant2 are irrelevant and only used in the functions ADD_SIGNAL and ADD_NOISE, which are defined by the user. The structure should hold *all* the information that you need to add signal and noise correctly. There is no limit on how much information P can contain, and which nature it should be. In this example, the pixprop* tags could describe pixel properties (since they are vectors of length n_{pixels}), such as the flux, counts, noise or the exposure in each pixel.

Step 2: Define the function ADD_SIGNAL

```
FUNCTION ADD_SIGNAL, index
  COMMON DATAVALUES, P
  RETURN, total(P.pixprop1[index])
END
```

This is the simplest example on how your ADD_SIGNAL function could look like. This function takes the global variable P and adds up the pixel property pixprop1 of all bin members (→ INDEX). The recipe on how compute the signal can be as easy or complicated as is required by the specific problem. This would be a real example if pixprop1 was something like a flux per pixel.

Step 3: Define the function ADD_NOISE

```
FUNCTION ADD_NOISE, index
  COMMON DATAVALUES, P
  RETURN, sqrt(total(P.pixprop2[index]^2))
END
```

The usage of ADD_NOISE is completely analogous to ADD_SIGNAL, with the only difference that the combined noise of the bin should be returned. In the example case, the square root of the sum of all squares of pixprop2, similar to a real example of adding gaussian errors.

Step 4: Start WVT_BINNING

Now we have everything set up to successfully start the adaptive binning. All we have to do is to call the main binning program WVT_BINNING.

```
WVT_BINNING, x, y, pixelSize, targetSN, $
    class, xNode, yNode, area, QUIET=quiet, $
    dens=dens, binvalue=binvalue, snbin=snbin, plotit=plotit, $
    resume=resume, neighborlist=neighborlist, max_area=max_area, $
    gersho=gersho, keepfixed=keepfixed
```

As you can see, nothing has changed in the way WVT_BINNING is being used. All we did was to introduce a new way to compute the signal-to-noise ratio of the bins. We have to be sure that the correct functions ADD_SIGNAL and ADD_NOISE are precompiled and that we have the global variable P initiated.