Université de Montréal

Documentation for my Weighted Voronoi Tessellations Algorithm.

Contents

1 Weighted Voronoi Tessellations

1.1 Introduction

Before delving into Weighted Voronoi Tessellations, I would first like to simply describe why we care about them (other than the fact that they are pretty cool). In a few words: WVT allows for a binning scheme that is unbiased when compared with Voronoi Tessellations, Centroidal Voronoi Tesselations, or other binning methods. Once we have the WVT bins in place, we can use that information to general temperature maps for X-ray data (That's what I am interested in).

See [1], [2], and [3] for more information.

1.2 Signal-to-Noise

For our particular application in X-ray astronomy, we are going to be interested in a signal-to-noise ratio dominated by count rates. Count Rates are a relatively easy quantity for us to compute. We use the following bash commands to calculate the **FLUX** at a pixel (This is assuming you skipped ahead and did all of the preprocessing as discussed in section 2!!).

```
punlearn eff2evt
pset eff2evt clobber=yes
eff2evt "input.fits" "output.fits"
```

The following equation are for pixel number k.

$$Signal_k = Flux_k = \frac{C_k}{E_k \tau} - B_k \tag{1}$$

$$Noise_k = \sigma_k^2 = \frac{C_k}{E_k^2 \tau^2} + \sigma_{B_k}^2 \tag{2}$$

Parameter	Definition
C_k	Raw Counts
E_k	Effective Area
τ	Exposure Time
B_k	Background Flux
σ_{B_k}	Uncertainty in Background Flux

HOWEVER, since we have done all of that wonderful data analysis described above, we have taken into account the Effective Area and Background Flux! Therefore our Signal-to-Noise calculation is reduced to the following simple expression:

$$SignalToNoise_k = \sqrt{Flux_k} \tag{3}$$

This calculation and data-gathering occurs in the READ_IN subroutine.

That sums up signal-to-noise calculations in the current version of the code.

1.3 Algorithm Run-Down

This section will contain the pseudo-algorithm of my python code.

Before we are able to begin our algorithm (WVT) in earnest, we must have a decent initial guess!

Enter **Bin Accretion**....

1.3.1 Bin Accretion

```
Data: Pixels
Result: Initial Guess for WVT
initialization;
while Pixels left to be assigned do
    while Criteria not all met do
        Calculate pixel closest to bin centroid;
        Calculate Adjacency;
        Calculate Roundness;
        Calculate Potential Signal-to-Noise;
        if Criteria all met then
            Add pixel to Bin;
        else
            Start new bin;
        end
        if Signal-to-Noise too low then
            Mark pixels unassigned;
        end
    end
end
Reassign all unassigned pixels to nearest bin;
```

Algorithm 1: Bin Accretion Algorithm

With that out of the way we can discuss how I actually calculated the three different criteria: Adjacency, Roundness, Potential Signal-to-Noise.

- Adjaceny: This one is simple: just check if the pixels are neighbors!
- Roundness: Ok so now we finally need to calculate some stuff... More explicity R_{equiv} and R_{max}. R_{max} is the maximum distance from the centroid for ALL pixels in bin. R_{equiv} is a wonderfully horrendous quantity which is "the radius of a disk around the bin". Lets do a quick little derivation...

$$Area_{circle} = \pi r_{circle}^2 == n\pi \frac{d_{pix}}{2} r = Area_{bin} \tag{4}$$

Solving for r_{circle} we get,

$$r_{circle} = \sqrt{\frac{n}{\pi}}\, d \tag{5}$$

And now we can calculate the roundness parameter which describes the compactness of the bin:

$$Roundness = \frac{R_{max}}{R_{equiv}} - 1 \tag{6}$$

We generally shoot for a value of $Roundness = 0.3$.

- Potential Signal-to-Noise: We add the pixel to the bin and then recompute the Signal-to-Noise. This needs to be less than the target Signal-to-Noise.

And now to the main event (which is not very much coding wise compared to bin accretion).

1.3.2 WVT

Data: Bins
Result: WVT Data
initialization $--> $ read in Bin Accretion Data;
while *Bins not converged* **do**
  for *bin in Bins* **do**
    Recalculate Signal-to-Noise;
    Calculate Area;
    Calculate Centroid;
    Calculate Scale Length;
  end
  Reassign Pixel to Closest Bin;
end

Algorithm 2: Weighted Voronoi Tessellation Algorithm

And again we need to define two properties: Area and Scale Length.

- Area: Area of bin. This is basically just the area of the pixels in the bin...

$$A_{bin} = (Pixel\_Length)^2 * N_{pixels} \tag{7}$$

- Scale Length: This scale length is the KEY to the WEIGHTED part of WVT.

$$\sqrt{\frac{A_{bin}}{\pi}\frac{S/N}{(S/N)_{target}}} \tag{8}$$

And really thats it!

2 Preprocessing of Data

I finally decided to put this section as the second chapter since the main goal of this document is to outline the WVT algorithm. However, I also wanted to include this information for anyone interested in X-ray data reduction from *CHANDRA*.

2.1 Download Data

Before we can begin to preprocess the data, we obviously need to have the data. Thankfully, this is incredibly simple! After downloading the data from the chandra website or using

$$download\_chandra\_obsid\ \#OBSID$$

There exists a considerable amount of documentation on each file downloaded at

$$http://cxc.harvard.edu/ciao/threads/intro\_data/$$

2.2 Reduce Data

We reduced the data with the following command:

$$chandra\_repro$$

. For details on what the reprocessing does visit the following website:

$$http://cxc.harvard.edu/ciao/ahelp/chandra\_repro.html$$

2.3 Region of Interest

Once we have the reduction completed (which it is easily done using either of these algorithms...) we can move onto dealing with the data we are interested in... We are assuming that you have already done the following to create a region of interest:

- opened ds9 and picked the region of interest: "ds9 ...evt2.fits &" and RofI is called "simple.reg"
- Also created background image "simple_bkgd.reg"

I saved them as physical units

Running this will create several files: PI, ARF, RMA. While I will leave the ARF and RMA file discussion to the webpage[1], we should note what our PI file gives us since it is critical in all of our analysis...

[1] http://cxc.harvard.edu/ciao/threads/extended/

2.4 Spectrum

The $*.pi$ file will enable us to calculate the $counts/cm^2/s$ for a given Energy bin (KeV). This basically gives us everything we need to use $sherpa$[2] or $xspec$ to model the data. It is worth noting the following equation:

$$PI = \left(\frac{Energy}{14.6KeV} + 1 \right) \tag{9}$$

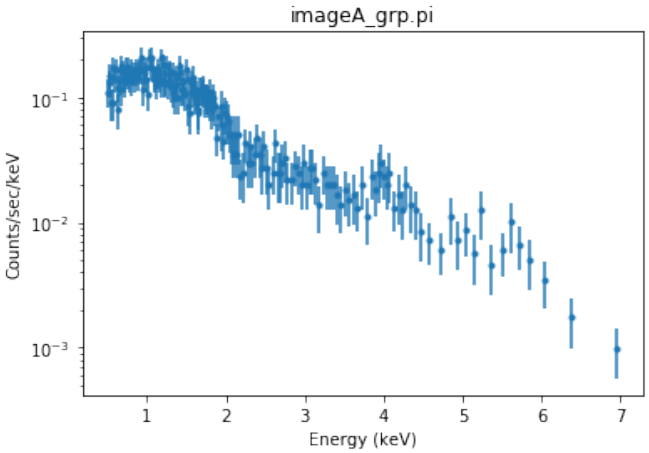

Figure 1: Example data taken from RXJ1131 OBSID 12833 Image A (Middle left)

2.5 2D Plots

We also can do some really nice 2D plots after running the following commands in the terminal:

```
dmcopy "acisf12833_repro_evt2.fits[sky=region(simple.reg)]" source.fits
dmcopy "source.fits[events][bin x=::.1,y=::.1][IMAGE]" source_img.fits
```

Now with our source image created we can go about making 2D plots several ways using python...

```
#OPTION A
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
from astropy.io import fits
from matplotlib.colors import LogNorm
from astropy.wcs import WCS

image_file = "full_source_img.fits"
hdu_list = fits.open(image_file)
image_data = hdu_list[0].data
hdu_list.close()

hdu = fits.open(image_file)[0]
```

[2] http://cxc.harvard.edu/sherpa/snapshot.html

```
wcs = WCS(hdu.header)

plt.subplot(projection=wcs)
plt.imshow(image_data, cmap='jet', norm=LogNorm())
plt.colorbar()
plt.title(r'RX J1131-1231 Image A')
plt.xlabel('RA')
plt.ylabel('DEC')
```

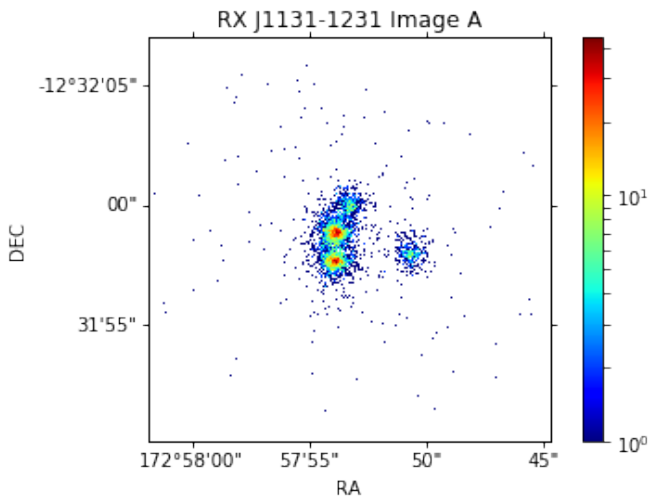

Figure 2: Option A

And after a little bit of fun with python coordinates:

```
#OPTION B
import aplpy
from astropy.coordinates import SkyCoord
c1 = SkyCoord(ra=51.60*u.arcsec, dec=59*u.arcsec, distance=d_RXJ, frame='icrs')
c2 = SkyCoord(ra=51.58*u.arcsec, dec=59*u.arcsec, distance=d_RXJ, frame='icrs')
sep = c1.separation_3d(c2)
fig = aplpy.FITSFigure(image_file)
fig.show_colorscale(cmap='jet')
fig.add_colorbar()
fig.colorbar.set_axis_label_text(r'Photon Flux $photons/cm^2/s$')
fig.add_scalebar(0.2*u.arcsec)
#fig.scalebar.set_length(0.02 * u.arcsecond)
fig.scalebar.set_color('white')
fig.scalebar.set_corner('top left')
fig.scalebar.set_label('%.2f Kpc' %(sep/u.kpc))
```

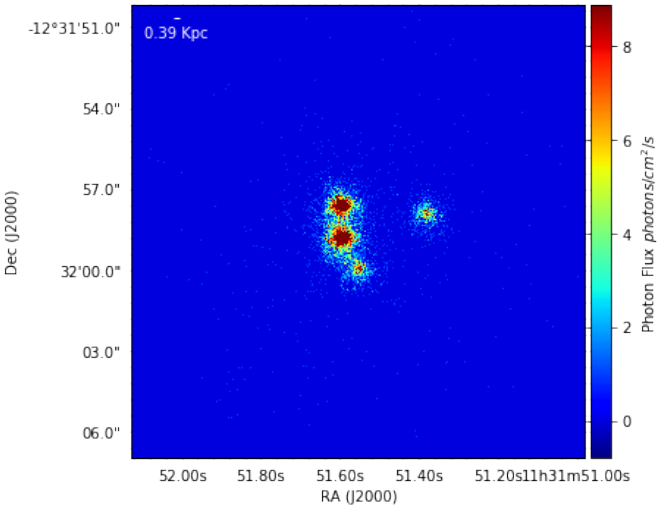

Figure 3: Option B

We also have the option of creating a true color image with the following bash script:

```
#!/bin/bash

punlearn dmcopy
pset dmcopy infile="full_source.fits[energy=200:1500][bin x=::.1,y=::.1]"
pset dmcopy outfile=soft_img.fits
pset dmcopy clobber=yes
dmcopy

punlearn dmcopy
pset dmcopy infile="full_source.fits[energy=1500:2500][bin x=::.1,y=::.1]"
pset dmcopy outfile=med_img.fits
pset dmcopy clobber=yes
dmcopy

punlearn dmcopy
pset dmcopy infile="full_source.fits[energy=2500:8000][bin x=::.1,y=::.1]"
pset dmcopy outfile=hard_img.fits
pset dmcopy clobber=yes
dmcopy

punlearn dmimg2jpg
pset dmimg2jpg infile=soft_img.fits
pset dmimg2jpg greenfile=med_img.fits
pset dmimg2jpg bluefile=hard_img.fits
pset dmimg2jpg outfile=truecolor.jpg
pset dmimg2jpg clobber=yes
dmimg2jpg
```

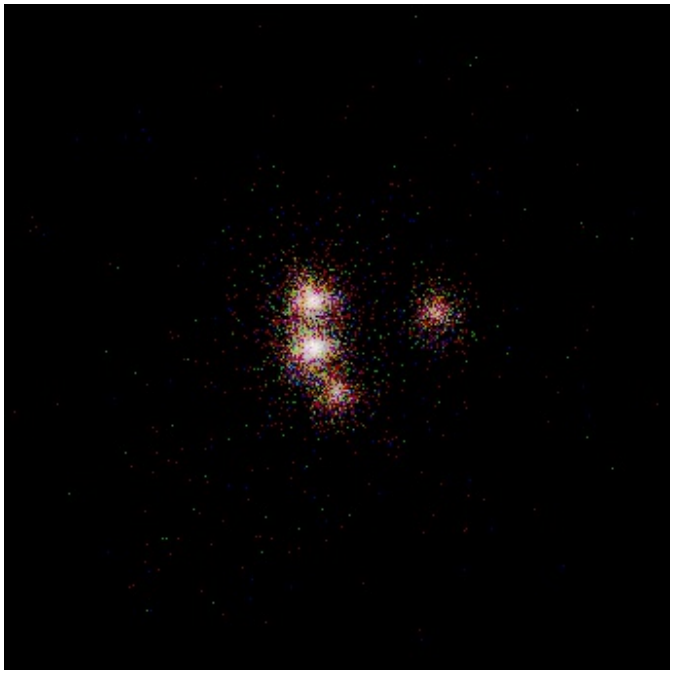

Figure 4: True Color

3 Additional Python Fun

3.1 Lookback Time

```
from astropy import units as u
from astropy import cosmology
from astropy.coordinates import *
from astropy.cosmology import Planck13, default_cosmology
default_cosmology.set(Planck13)
z_RXJ = 0.658
d_RXJ = Distance(z=z_RXJ, unit=u.kpc)
```

Bibliographies

References

[1] Yannick Copin. Adaptive spatial binning of integral-field spectroscopic data using Voronoi tessellations. 10(January):1–10, 2018.

[2] Steven Diehl and Thomas S Statler. Adaptive Binning of X-ray data with Weighted Voronoi Tesselations. 14(February):1–14, 2008.

[3] J S Sanders and A C Fabian. Adaptive binning of X-ray galaxy cluster images. 000(May), 2018.