Table of Contents:

Compiling

The code is written in C++ and a Makefile is included. You can specify your favorite compiler there. Note that you will at minimum need to change some include paths in the Makefile.

Dependences

The following libraries are required to compile:

- GNU Scientific Library
- CFITSIO (In the Makefile, this is accomplished via my local HEASOFT installation, which includes CFITSIO.)
- WCSlib

Usage

Image types and file formats

The main inputs to the code are FITS files. Because my facility with FITSIO is limited, there are some requirements that these must follow, listed below. The program can also write new images (see next section), and output files will follow the same conventions.

- Gzipped files (.gz) are, optionally, allowed (both input and output).
- Standard FITS images please, i.e. primary extensions of the IMAGE type, not TABLE, and with the required NAXIS* keywords.

- Data type must be **single precision float** (BITPIX = -32). This applies even though some image files should contain integer-valued count maps.
- Image pixels that are not illuminated or are otherwise masked a priori should be given a **negative** value in the science image and/or a **zero** value in the exposure map.

The three types of image files we deal with are

- "Images", referring to the actual, integer-valued photon count maps of a science target.
- Optional "background" or "blank-sky" maps, meant to represent an empty patch of sky under equivalent observing conditions. Normally integer-valued by default, but can be real-valued.
- Optional "exposure" maps (real-valued) encoding the spatial variation of exposure, vignetting and instrument response.

The `morphology` executable

The compiled program has a multitude of command-line arguments, most of which you will not have to worry about.[1]

```
usage: morphology [-n] [--no-bg-fit] [--set-bg value] [--real-bg] [--bg-file file
factor fracerr] [--expmap file] [--exposure seconds] [--mask mask.reg] [--fix-center
imagex imagey] [--peaky-flux level] [--peaky-frac fraction] [--isoph-min-level level]
[--isoph-max-level level] [--num-isoph number] [--min-isoph-pix number] [-v verbose
level | --quiet | --chatty | --gregarious | --obnoxious] [--boot N] [--beta-rmin
radius] [--erase-rmin] [--gamma-k0 k0] [--sbthresh S/N] [--kpc kpc per pixel] [--Ralt
aperture in pix for w and P3/P0] [--write-flat file.fits[.gz]] [--write-error
file.fits[.gz]] [--write-smooth file.fits[.gz]] [--write-isophotes file.fits[.gz]]
[--write-filth file.fits[.gz]] [--just-merge | --just-center | --just-sbprof]
<image.fits[.gz]>

To combine observations, each argument to --set-bg, --bg-file, --expmap, --exposure and
--mask, as well as the required argument, should be a comma-, semicolon- or
space-separated list. Images are reprojected to the coordinate system of the first one
listed. --kpc refers to that pixel size.
```

Required arguments

Strictly speaking, the only required argument is one or more science images, following the conventions for FITS files above. However, for the results to be meaningful (i.e. to correspond to the procedure in our paper), some of the options are effectively required (see especially the Analysis options) even though there are default values in the code. The options that get regular use in our analysis are highlighted in blue below.

[1] In a known bug, the real usage message incorrectly calls the `--isoph-min-level` and `--isoph-max-level` options `--isoph-flux-min` and `--isoph-flux-max`. The correct names are the ones shown here in the README.

Options

Control options

- -n: don't actually do anything other than process the argument list (and parrot back, if requested)
- --just-merge | --just-center | --just-sbprof: exit after one of the early processing steps (merging of multiple images, center finding or generation of the surface brightness profile)
- --boot N: run the code N times, bootstrapping the input photons. This is how we determine uncertainties on the measurements. You definitely want minimum output (below) when running in this mode.

Output options

- -v verbose level | --quiet | --chatty | --gregarious | --obnoxious: determines the amount of feedback printed to the terminal (verbose level would be 0, 1, 2 or 3).
- --write-flat, --write-error, --write-smooth, --write-isophotes, --write-filth: write FITS image(s) of one or more of these intermediate products: flatfielded image, error on the flatfield, smoothed flat, isophote assignment, flat with masked regions filled in.

Background options

- --bg-file file factor fracerr: use file as a background map. After (possibly) bootstrapping, the map is scaled by factor (+- Gaussian fractional error fracerr, if bootstrapping) before being subtracted from the science image. (NB: In the case of multiple images, each of these arguments becomes a comma-separated list.)
- --real-bg: map specified by --bg-file is real-valued rather than integer-valued. In this case, it cannot be bootstrapped.
- --set-bg: just subtract this value from the image. Ignored if a --bg-file is passed.

Exposure options

- --expmap: use this exposure map
- --exposure: specify the total exposure in seconds. Otherwise, the value of the EXPOSURE keyword of the image file's FITS header is used.

Warning: these keywords can be confusing. The image-minus-background map (counts) needs to be divided by an exposure map with units of cm^2s. In practice, we divide by the specified exposure map (if any) and *t*, where *t* is from either the EXPOSURE keyword of the image file (default) or the --exposure option. This works famously for Chandra data, where by convention the "exposure" maps (units of cm^2) encode spatial *variations* in exposure and vignetting as well as the effective area, but **not** the overall exposure. More commonly, the total exposure appears in both the keyword and the exposure map (units of s), and the overall normalization of the effective area appears in neither. In that case, one would want to set --exposure to the overall area (usually the on-axis value) for the energy range corresponding to the data. It is the user's responsibility to make sure that the exposure time and effective area are included in the calculation exactly once!

Input massaging

- --mask mask.reg: ds9-style regions in **image coordinates** to be masked before doing anything else. Only circles and ellipses are supported. This is for those pesky point sources that you didn't notice until the image FITS files were already made.

Analysis options

The morphology code does not know any physics! This means that the all-important surface brightness scaling and our isophote definitions need to be passed somewhat awkwardly through the following keywords. Let norm = $K(z,T,N_H)$ (kT/keV) $E(z)^3/(1+z)^4$ $(p/0.984'')^2$ as in eqn 3 of the paper, where p is your pixel size. The SPA analysis uses

- --isoph-max-level 0.05*norm
- --isoph-min-level 2e-3*norm
- --num-isoph 5
- --peaky-flux 0.0475*norm (This is a vestigial option that just needs to be smaller than isoph-flux-max.)

And some more miscellaneous analysis options:

- --no-bg-fit: disables the fitting and subtraction of a constant residual background from the surface brightness profile.
- --fix-center: normally, the code should be allowed to find the cluster center on its own. If that fails spectacularly, as a last resort, you can specify the center in image coordinates here. (For multiple images, these are coordinates into the merged image, which you can work out by writing the flatfield, for example.)
- --min-isoph-pix: number of pixels required in a given isophote to proceed with the analysis of said isophote. Defaults to 100. May need to be lower for very course pixelization.
- --beta-rmin: normally, we fit a beta model+constant to the outer half of an adaptively binned surface profile. This seems to work well when the background subtraction is good. Otherwise, specify an inner radius in pixels for the fit.
- --erase-rmin: erase everything in the flatfielded image at radii (in pixels) greater than this. Use if there's lots of crap around the edges of your image for some reason. Happens after center-finding.
- --gamma-k0: affects the variance assigned to individual pixels. You do not want to know the details, I promise.
- --sbthresh: change the signal-to-noise used in adaptively binning the surface brightness profile. Why would you want to do this?
- --kpc: the size of a pixel in kpc. If specified, the Santos c_{SB} surface brightness concentration will be calculated.
- --Ralt: if passed, the centroid variance and power ratios within this radius (in pixels) will be calculated.
- --peaky-frac: deprecated

Output

Depending on the verbosity setting, a large amount of feedback may be written to the terminal. In addition to that (or alone, if using --quiet), the numerical results of the algorithm will be written. There will be one line per bootstrap iteration, or a single line corresponding to the input data if not bootstrapping. The values written do not include peakiness/symmetry/alignment, but these can be derived from them. Each line consists of:

`<center x> <center y> <csb> <filth fraction> <cen var> <4 power ratios> <avg central sb> <4 more power ratios> <ellipses>`

The number of ellipses depends on the --num-isoph setting. Each `<ellipse>` expands to:

`<x> <y> <semimajor axis> <semiminor axis> <position angle> <exposed frac> <avg cosine> <avg sine>`

where the latter 3 are used to reject "bad" elliptical fits (see the paper).

You can now either calculate the SPA quantities from this output, or use the `reduce2.R` script described below to do so.

Supporting scripts

You have probably gathered that the exact command line you will be using is telescope- and cluster-dependent. I've provided a bash script which calls the code as an example appropriate for Chandra data (with various cluster and observation properties assumed to be saved in appropriate files). This is unlikely to be directly useful other than as an example, apart from the following aspect.

Typically, we first do a non-bootstrap analysis with the highest verbosity setting, parsing the output into a few different files (see the awk commands at the bottom of the script). This allows us to produce some informative plots (using the provided makeplots.R script) to verify that everything is working reasonably on a given data set. If so, we do the bootstrap analysis with minimal output. The results can be translated into SPA quantities using the reduce2.R script.

To use the R scripts, you will need (in addition to R) the FITSio package, which is easy enough to install. Just typing `install.packages("FITSio")` in R should do it. Both scripts are quick & dirty, but I think at least there are no hidden dependences on other script files.

R is freely available for just about any platform at http://www.r-project.org/. If you've never used it, you may as well start now.

Notes on reduce2.R

The calling sequence is

```
./reduce2.R <morph.log> <redshift> [morph_boot.dat]
```

where results are based on the non-bootstrap run in morph.log if the last argument is not included. It assumes that a line with "SB scaling factor" appears in the log file, which will be the case if you haven't changed my calling script too much. (This is the “norm” discussed in the Analysis Options section, above.)

When I test it, the output looks something like this:

```
0.963
1.015536 -0.5167867 1.39044
0.008999406 -0.000231339 -0.001232162
-0.000231339 0.001317087 -0.0001095687
-0.001232162 -0.0001095687 0.02545858
```

which means

P(relaxed) (using the Chandra cuts; this can be adjusted in the script)
mean symmetry, peakiness, alignment
3x3 covariance of s, p, a

In case the fit has < 2 good ellipses, I think you should just get the mean and variance of p.