pyproffit

Release 0.5

Nov 09, 2021

Contents:

CHAPTER 1

Introduction

`pyproffit` is a high-level Python package which aims to provide an easy and intuitive way of performing photometric analysis with X-ray images of galaxy clusters. It is essentially a Python replacement for the PROFFIT C++ interactive package (Eckert et al. 2011). It includes all features of the original PROFFIT package, and more. Available features include:

- Extraction of surface brightness profiles in circular and elliptical annuli, over the entire azimuth or any given sector
- Fitting of profiles with a number of built-in model or any given user-defined model, using chi-squared or C statistic
- Bayesian fitting using Emcee and/or PyMC3 with automatic or custom priors
- Non-parametric deprojection and extraction of gas density profiles and gas masses
- PSF deconvolution, count rate and luminosity reconstruction in any user defined radial range, surface brightness concentration
- Two-dimensional model images and surface brightness deviations
- Surface brightness fluctuation power spectra and conversion into 3D density power spectra

The current implementation has been developed in Python 3 and tested on Python 3.6+ under Linux and Mac OS.

1.1 Motivation

While the original PROFFIT package has attracted a substantial number of users, its structure was extremely rigid and outdated, making it difficult to maintain and very difficult for the user to add any custom features. `pyproffit` aims at providing all the popular features of PROFFIT in the form of an easy-to-use Python package. The modular structure of pyproffit allows the user to easily interface with other Python packages and develop additional features and models. The ultimate goal is to allow the user to perform any type of analysis directly within a Jupyter notebook.

1.2 Limitations

- The computation of the PSF mixing matrix currently only works with PSF images that have the same pixel size as the provided image.

CHAPTER 2

Installation

`pyproffit` is available on Github and PyPI.

The easiest way of installing `pyproffit` is obviously to use pip:

```
pip3 install pyproffit
```

The PyPI repository should contain the latest stable release (as judged by the developer), it may not be the latest version thus some features may be missing. To install the latest version from Github:

```
git clone https://github.com/domeckert/pyproffit.git
cd pyproffit
pip3 install .
```

`pyproffit` depends on numpy, scipy, astropy, matplotlib, iminuit, pymc3, and pystan.

A test dataset and a validation script are also available in the `validation` directory. To run the validation script:

```
cd validation
python3 test_script.py
```

CHAPTER 3

Tutorials

3.1 Example: Surface brightness profile extraction and fitting

This thread shows how to read data, extract surface brightness profiles, fit data and extract density profiles with *PyProffit*. The steps presented here can be replicated using the test data available in the *validation* folder of the *PyProffit* package.

3.1.1 Reading data

We start by loading the packages:

[1]:
```
import numpy as np
import pyproffit
import matplotlib.pyplot as plt
```

Then we can move to the data directory with the *os* package.

[2]:
```
import os

# Change this to the proper directory containing your run
os.chdir('../../validation/')
os.listdir()
```

[2]:
```
['test_sb.fits',
 'gsb.fits',
 'expose_mask_37.fits.gz',
 'test_density.pdf',
 'commands.xcm',
 'test_outmod.fits',
 'test_script.py',
 '.ipynb_checkpoints',
 'test_plot_fit.pdf',
 'test_rec_stan.pdf',
```

(continues on next page)

(continued from previous page)

```
 'reference_depr.fits',
 'test_save_all.fits',
 'pspcb_gain2_256.rsp',
 'b_37.fits.gz',
 'reference_psf.dat',
 'test_dmfilth.fits',
 'Untitled1.ipynb',
 'ign **-0.42',
 'back_37.fits.gz',
 'reference_pymc3.dat',
 'comp_rec.pdf',
 'test_region.reg',
 'test_mgas.pdf',
 'mybeta_GP.stan',
 'reference_OP.dat',
 'pspcb_gain2_256.fak',
 'Untitled.ipynb',
 'sim.txt',
 'lumin.txt']
```

Now we load the data inside a Data object in PyProffit structure:

[3]:
```
dat=pyproffit.Data(imglink='b_37.fits.gz',explink='expose_mask_37.fits.gz',
                   bkglink='back_37.fits.gz')
```

```
WARNING: FITSFixedWarning: RADECSYS= 'FK5 ' / Equatorial system reference
the RADECSYS keyword is deprecated, use RADESYSa. [astropy.wcs.wcs]
```

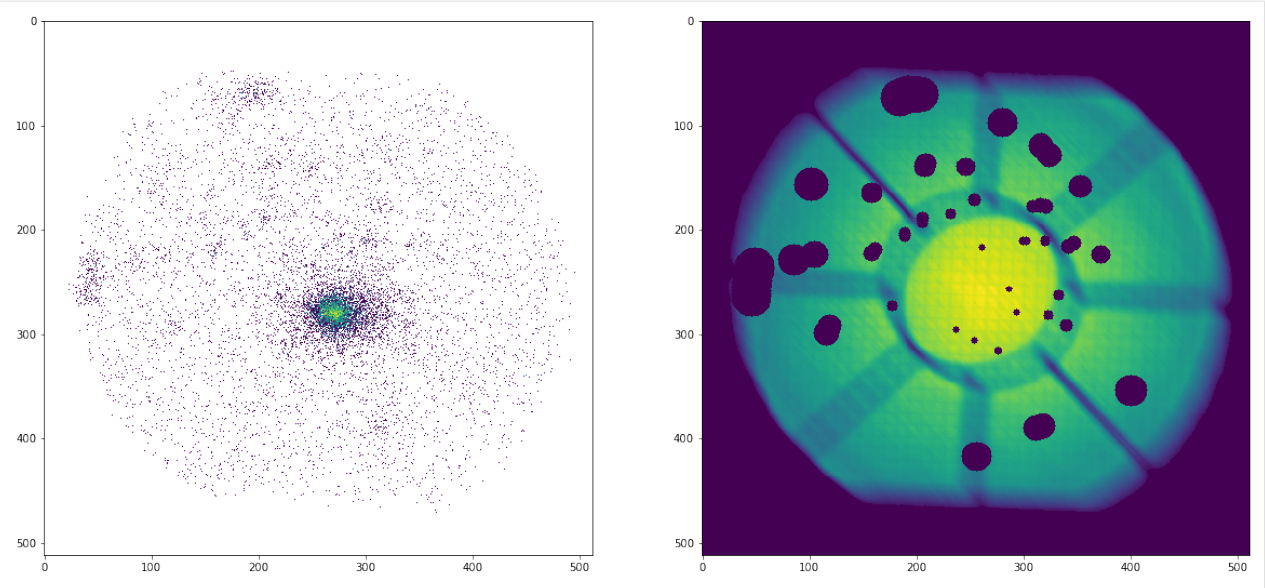
1. Here *imglink*='b_37.fits.gz'_ is the link to the image file (count map) to be loaded.
2. The option *explink*='expose_mask_37.fits.gz' allows the user to load an exposure map for vignetting correction. In case this option is left blank, a uniform exposure of 1s is assumed for the observation.
3. The option *bkglink*='back_37.fits.gz' allows to load an external background map, which will be used when extracting surface brightness profiles.

The images are then loaded into the Data structure and can be easily accessed as below:

[4]:
```
fig = plt.figure(figsize=(20,20))
s1=plt.subplot(221)
plt.imshow(np.log10(dat.img),aspect='auto')
s2=plt.subplot(222)
plt.imshow(dat.exposure,aspect='auto')
```

```
<ipython-input-4-00e9545509bd>:3: RuntimeWarning: divide by zero encountered in log10
  plt.imshow(np.log10(dat.img),aspect='auto')
```

[4]:
```
<matplotlib.image.AxesImage at 0x7f5cf4821460>
```



All the areas with zero exposure will be automatically excluded. We can ignore additional regions using the region method of the Data class, which loads a DS9 region file (in image or FK5 format):

[5]:
```
dat.region('test_region.reg')
```

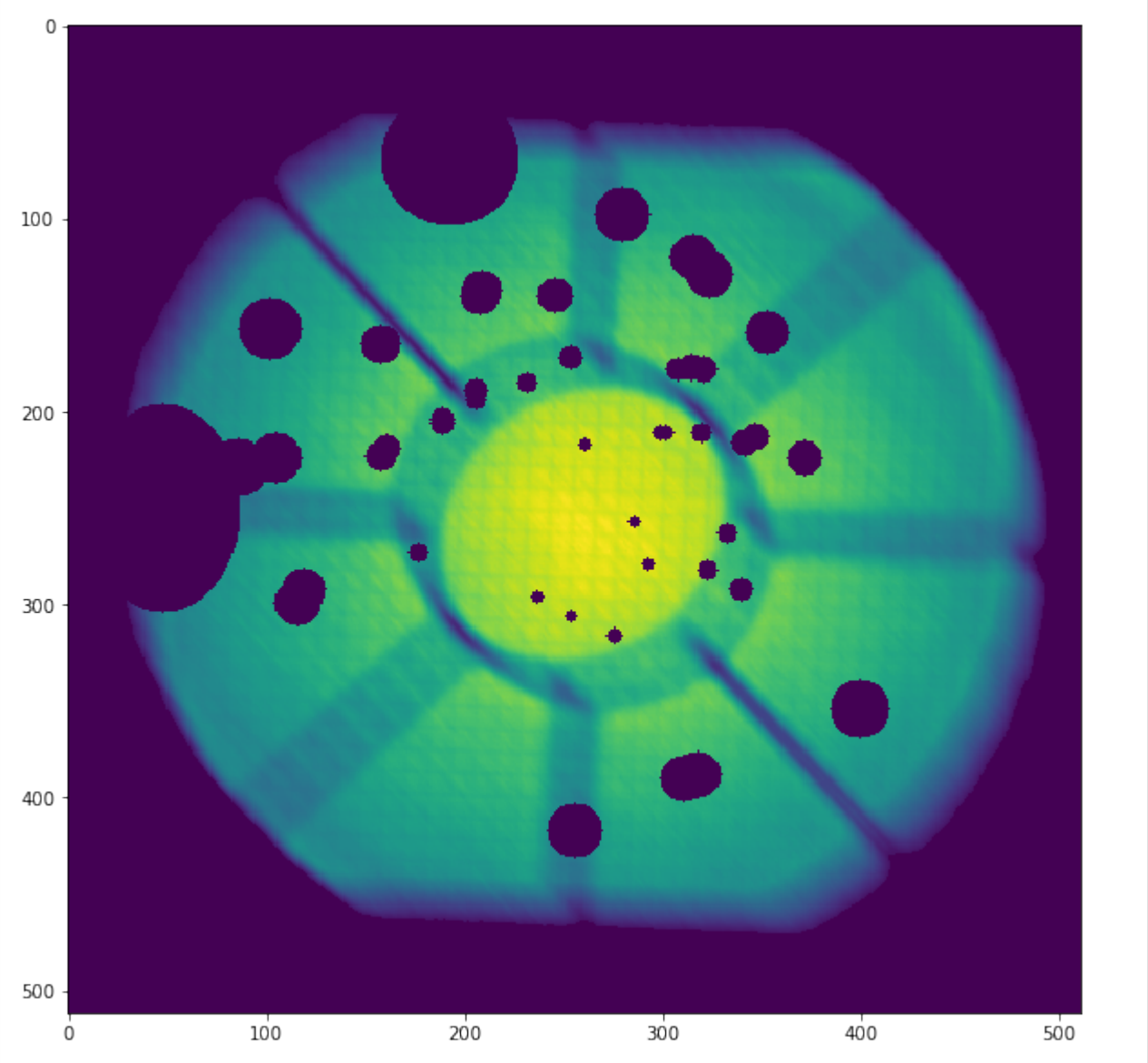
```
Excluded 2 sources
```

The exposure in the requested areas has been set to 0. Let's look at the output:

[6]:
```
plt.clf()
fig = plt.figure(figsize=(10,10))
plt.imshow(dat.exposure,aspect='auto')
```

[6]:
```
<matplotlib.image.AxesImage at 0x7f5cf3f98d00>
```

```
<Figure size 432x288 with 0 Axes>
```



The Data structure also contains the dmfilth method, which can be used to fill the masked areas. The method computes a 2D spline interpolation in between the gaps and generates a Poisson realization of the spline interpolated data, such that the filled holes have similar statistical properties to their surroundings

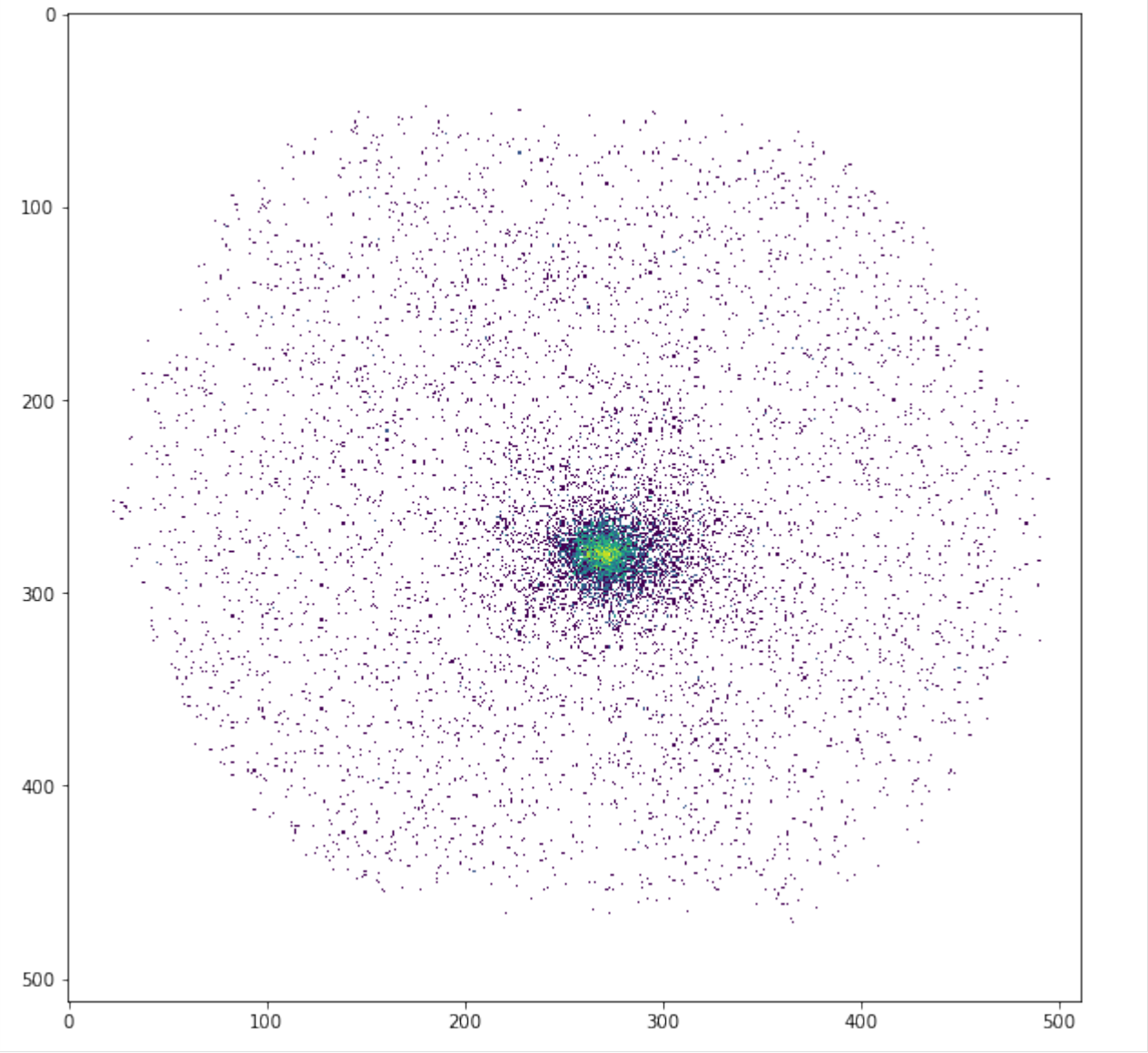
[7]:
```
dat.dmfilth()
```

```
Applying high-pass filter
Interpolating in the masked regions
Filling holes
```

[8]:
```
plt.clf()
fig = plt.figure(figsize=(10,10))
plt.imshow(np.log10(dat.filth),aspect='auto')
```

```
<ipython-input-8-010fe1205bf9>:3: RuntimeWarning: divide by zero encountered in log10
  plt.imshow(np.log10(dat.filth),aspect='auto')
```

[8]:
```
<matplotlib.image.AxesImage at 0x7f5cf471eeb0>
<Figure size 432x288 with 0 Axes>
```



The image produced by dmfilth is to be compared with the raw image shown above; it is apparent that the sources have been removed and their area has been replaced by a Poisson realization of their interpolated surroundings.

In case a dmfilth image has been generated, the computation of the image centroid and/or of the surface brightness peak to compute the center of the surface brightness profile is done on the dmfilth image rather than on the original image.

3.1.2 Profile extraction

Now we define a Profile object in the following way:

[9]:
```
prof=pyproffit.Profile(dat,center_choice='centroid',maxrad=45.,binsize=20.,centroid_
↪region=30.)
```

```
Computing centroid and ellipse parameters using principal component analysis
No approximate center provided, will search for the centroid within a radius of 30␣
↪arcmin from the center of the image
Denoising image...
Running PCA...
Centroid position: 272.6425582785415 277.47130902570234
Corresponding FK5 coordinates:  55.71693918958471 -53.642873306648674
Ellipse axis ratio and position angle: 1.112280720144629 -144.82405288826155
```

Profile class options

The class Profile is designed to contain all the Proffit profile extraction features (not all of them have been implemented yet). The "center_choice" argument specifies the choice of the center:

- center_choice='centroid': compute image centroid and ellipticity
- center_choice='peak': use brightness peak
- center_choice='custom_fk5': use custom center in FK5 coordinates (degrees), provided by the "center_ra" and "center_dec" arguments
- center_choice='custom_ima': like custom_fk but with input coordinates in image pixels

The other arguments are the following:

- maxrad: define the maximum radius of the profile (in arcmin)
- binsize: the width of the bins (in arcsec)
- center_ra, center_dec: position of the center (if center_choice='custom_fk5' or 'custom_ima')
- binsize: minimum bin size in arcsec
- binning=: specify binnig scheme: 'linear' (default), 'log', or 'custom'. In the 'custom' case, an array with the binning definition should be provided through the option bins=array
- centroid_region: for centroid calculation (center_choice='centroid'), optionally provide a radius within which the centroid will be computed, instead of the entire image.

Now let's extract the profile...

[10]:
```
prof.SBprofile(ellipse_ratio=prof.ellratio,rotation_angle=prof.ellangle+180.)
```

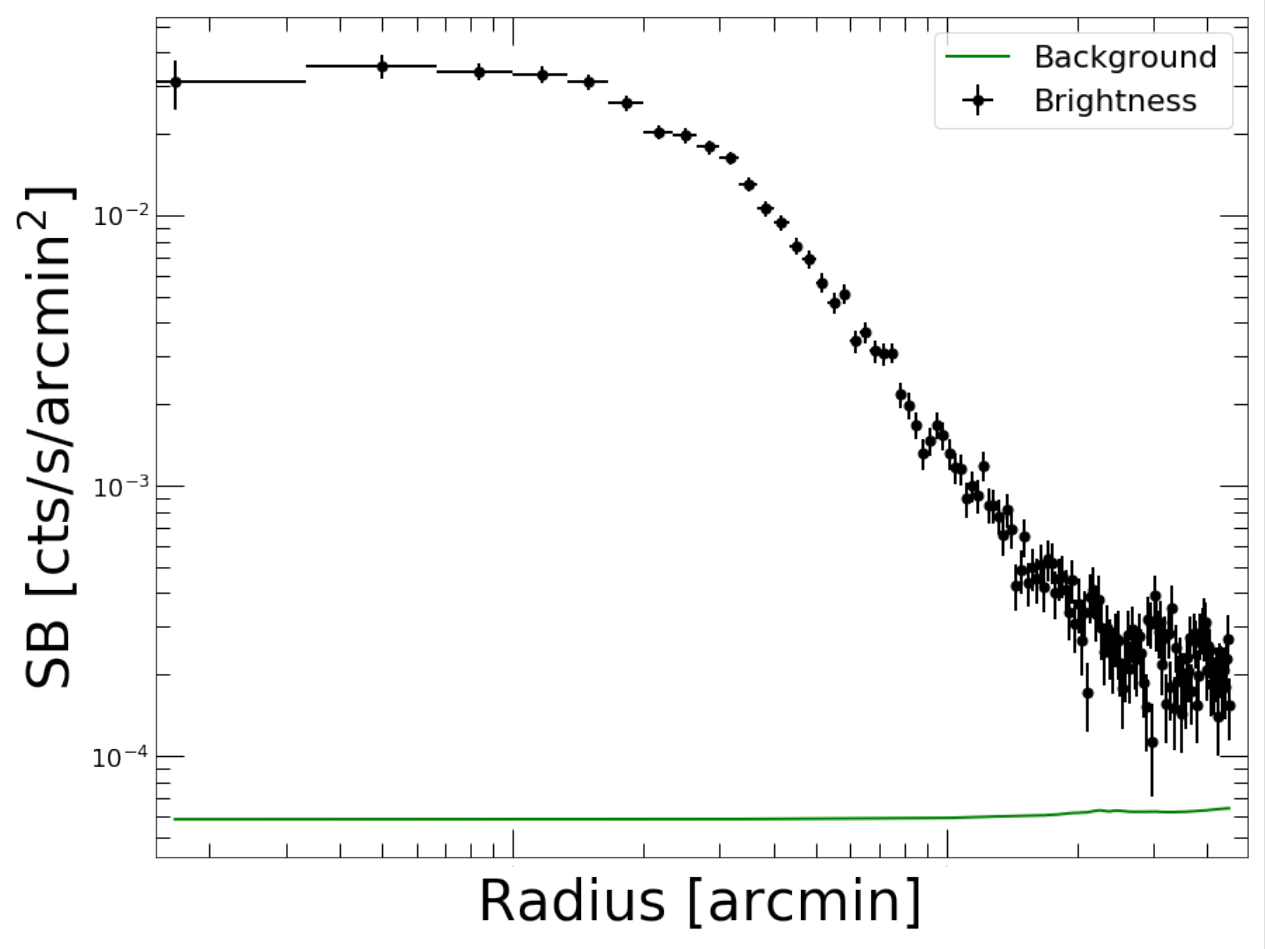
Here we have extracted a profile in elliptical annuli centered on the image centroid (see above), with an ellipse axis ratio (major/minor) and position angle calculated with principal component analysis. If ellipse_ratio and ellipse_angle are left blank circular annuli are used.

- ellipse_ratio: the ratio of major to minor axis (a/b) of the ellipse (default=1, i.e. circular annuli)
- rotation_angle: rotation angle of the ellipse from the R.A. axis (default=0)
- angle_low, angle_high: in case of profile extraction in sectors, the position angle of the minimum and maximum angles of the sector, with 0 equivalent to the R.A. axis (default=None, i.e. the entire azimuth)

Now let's plot the profile...

[11]:
```
prof.Plot()
```

```
<Figure size 432x288 with 0 Axes>
```



3.1.3 Defining a model

Models can be defined using the Model class. PyProffit includes several popular built-in models, however the Model structure is designed to be compatible with any custom Python function (see below)

```
[12]: mod=pyproffit.Model(pyproffit.BetaModel)
```

To check the parameters of the BetaModel function,

```
[13]: mod.parnames
[13]: ('beta', 'rc', 'norm', 'bkg')
```

Any user-defined Python function operating on NumPy arrays can be defined here, see below.

3.1.4 Fitting the data

To fit the extracted profiles PyProffit provides the Fitter class, which takes a Profile and a Model object as input:

```
[14]: fitobj=pyproffit.Fitter(model=mod, profile=prof, beta=0.7, rc=2.,norm=-2,bkg=-4)
```

(continues on next page)

(continued from previous page)

```
fitobj.Migrad()
```

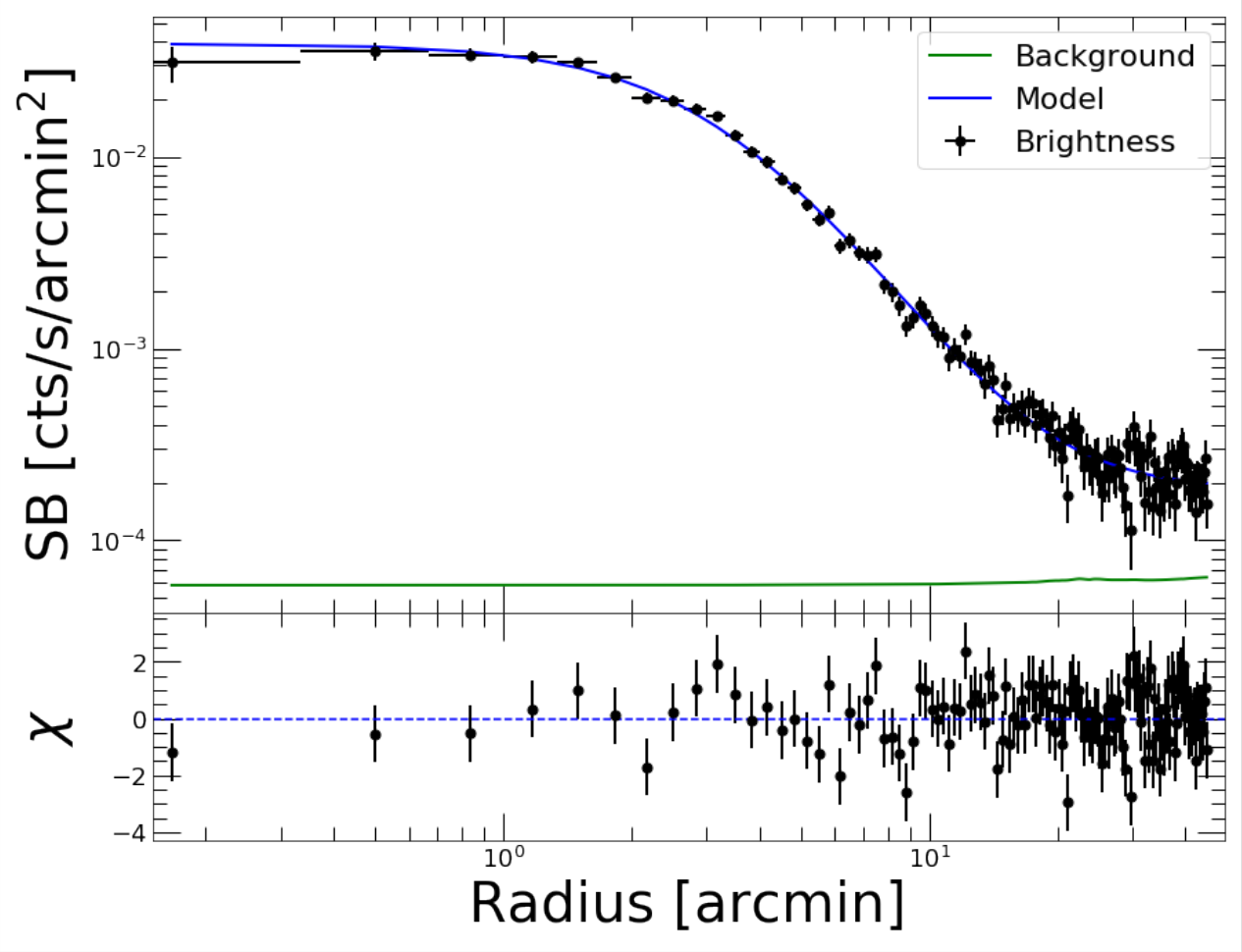
```
┌──────────────────────────────────────────────────────────────────────────────┐
│ FCN = 143.5                       │              Nfcn = 273                  │
│ EDM = 1.13e-05 (Goal: 0.0002)     │                                          │
├──────────────────────────────────────────────────────────────────────────────┤
│ Valid Minimum │ Valid Parameters  │        No Parameters at limit            │
├──────────────────────────────────────────────────────────────────────────────┤
│ Below EDM threshold (goal x 10)   │           Below call limit               │
├──────────────────────────────────────────────────────────────────────────────┤
│  Covariance   │     Hesse ok      │ Accurate  │  Pos. def.   │ Not forced    │
└──────────────────────────────────────────────────────────────────────────────┘
┌──────────────────────────────────────────────────────────────────────────────┐
│   │ Name │   Value    │ Hesse Err │ Minos Err- │ Minos Err+ │ Limit-  │ Limit+  │
↪Fixed │
├──────────────────────────────────────────────────────────────────────────────┤
│ 0 │ beta │   0.672    │   0.016   │            │            │         │         │
↪   │
│ 1 │ rc   │   3.27     │   0.15    │            │            │         │         │
↪   │
│ 2 │ norm │  -1.412    │   0.016   │            │            │         │         │
↪   │
│ 3 │ bkg  │  -3.733    │   0.020   │            │            │         │         │
↪   │
└──────────────────────────────────────────────────────────────────────────────┘
┌──────────────────────────────────────────────────────┐
│      │      beta        rc       norm       bkg     │
├──────────────────────────────────────────────────────┤
│ beta │  0.000267   0.00226 -0.000138  0.000215       │
│   rc │   0.00226    0.0222  -0.00184   0.00158       │
│ norm │ -0.000138  -0.00184  0.000258 -7.96e-05       │
│  bkg │  0.000215   0.00158 -7.96e-05  0.000386       │
└──────────────────────────────────────────────────────┘
Best fit chi-squared: 143.525 for 135 bins and 131 d.o.f
Reduced chi-squared: 1.09561
```

The fit provides the best-fit parameters with their errors and the covariance matrix. The fit statistic is provided at the bottom. All the results of the optimization can be accessed through the *fitobj.out* attribute of the Fitter class, which contains the output of the iminuit *migrad* command.

Now we can plot the data together with the best fitting model

[15]:
```
prof.Plot(model=mod)
```

```
<Figure size 432x288 with 0 Axes>
```



That's nice; now if instead of *:math:*chi^2‘ <https://pyproffit.readthedocs.io/en/latest/pyproffit.html#pyproffit.fitting.ChiSquared>‘__ we want to fit the counts with C statistic

[16]:
```
fitobj=pyproffit.Fitter(model=mod, method='cstat', profile=prof, beta=0.7, rc=2.,
↪norm=-2, bkg=-4, fitlow=0., fithigh=30.)

fitobj.Migrad()
```

FCN = 86.67	Nfcn = 316
EDM = 9.35e-06 (Goal: 0.0002)	
Valid Minimum	Valid Parameters	No Parameters at limit
Below EDM threshold (goal x 10)	Below call limit
Covariance	Hesse ok	Accurate	Pos. def.	Not forced

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	beta	0.658	0.019					

(continues on next page)

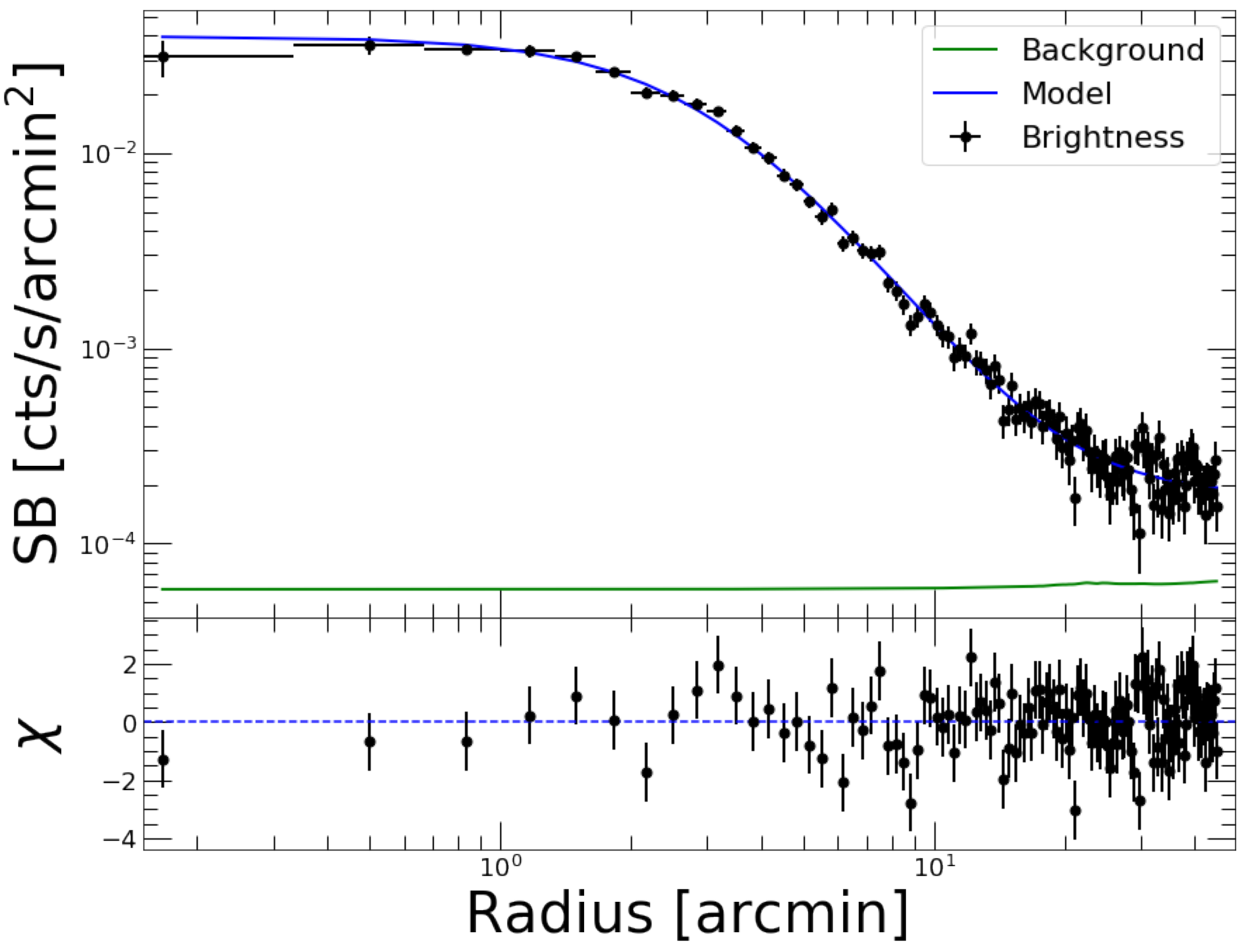
(continued from previous page)

```
│ 1 │ rc    │   3.17    │   0.17    │            │            │          │          │   ␣
↪   │
│ 2 │ norm │  -1.406   │   0.017   │            │            │          │          │   ␣
↪   │
│ 3 │ bkg  │   -3.75   │   0.04    │            │            │          │          │   ␣
↪   │
└────────────────────────────────────────────────────────────────────────────────────

──────────────────────────────────────────────
│       │       beta        rc      norm        bkg │
│───────────────────────────────────────────────
│  beta │  0.000372   0.00298 -0.000178  0.000673 │
│    rc │   0.00298    0.0273  -0.00216   0.00474 │
│  norm │ -0.000178  -0.00216  0.000284 -0.000235 │
│   bkg │  0.000673   0.00474 -0.000235   0.00191 │
└──────────────────────────────────────────────
Best fit C-statistic: 86.6662 for 135 bins and 131 d.o.f
Reduced C-statistic: 0.661574
```

Here we have restricted the fitting range to be between 0 and 30 arcmin through the *fitlow* and *fithigh* arguments. Note that the "reduced C-statistic" cannot be used directly as a goodness-of-fit indicator.

[17]:
```
prof.Plot(model=mod)
```

```
<Figure size 432x288 with 0 Axes>
```



3.1.5 Inspecting the results

After running *Migrad* the Fitter object contains a *minuit* object which can be used to run all iminuit tools. For instance, we can run *minos* to get more accurate errors on the parameters:

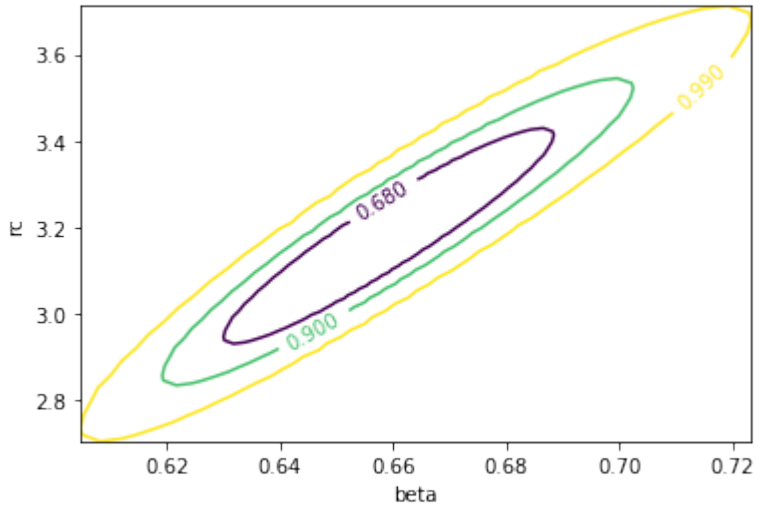
[18]:
```
fitobj.minuit.minos()
```

[18]:
```
┌──────────────────────────────────┬──────────────────────────────────────┐
│ FCN = 86.67                      │              Nfcn = 673              │
│ EDM = 9.35e-06 (Goal: 0.0002)    │                                      │
├──────────────────────────────────┼──────────────────────────────────────┤
│ Valid Minimum │ Valid Parameters │        No Parameters at limit        │
├──────────────────────────────────┴──────────────────────────────────────┤
│ Below EDM threshold (goal x 10)  │           Below call limit           │
├──────────────┬───────────────────┬──────────┬──────────┬────────────────┤
│  Covariance  │     Hesse ok      │ Accurate │ Pos. def.│ Not forced     │
└──────────────┴───────────────────┴──────────┴──────────┴────────────────┘
┌───┬──────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┬
│   │ Name │   Value  │ Hesse Err│ Minos Err-│ Minos Err+│ Limit-  │ Limit+  │
↪Fixed │
├───┼──────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┼
│ 0 │ beta │   0.658  │   0.019  │   -0.019 │   0.020  │          │          │
↪   │
│ 1 │ rc   │   3.17   │   0.17   │   -0.16  │    0.17  │          │          │
↪   │
│ 2 │ norm │  -1.406  │   0.017  │   -0.017 │   0.017  │          │          │
↪   │
│ 3 │ bkg  │   -3.75  │   0.04   │   -0.05  │   0.04   │          │          │
↪   │
└───┴──────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴
┌──────────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┐
│          │         beta        │          rc         │        norm         │
↪        bkg          │
├──────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│  Error   │  -0.019  │   0.020  │   -0.16  │   0.17   │  -0.017  │   0.017  │
↪ -0.05   │   0.04   │
│  Valid   │   True   │   True   │   True   │   True   │   True   │   True   │
↪ True     │   True   │
│ At Limit │   False  │   False  │   False  │   False  │   False  │   False  │
↪ False    │   False  │
│ Max FCN  │   False  │   False  │   False  │   False  │   False  │   False  │
↪ False    │   False  │
│ New Min  │   False  │   False  │   False  │   False  │   False  │   False  │
↪ False    │   False  │
└──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘
┌──────┬──────────┬──────────┬──────────┬──────────┐
│      │      beta│        rc│      norm│       bkg│
├──────┼──────────┼──────────┼──────────┼──────────┤
│ beta │  0.000372│   0.00298│-0.000178 │  0.000673│
│   rc │   0.00298│    0.0273│  -0.00216│   0.00474│
│ norm │-0.000178 │  -0.00216│  0.000284│-0.000235 │
│  bkg │  0.000673│   0.00474│-0.000235 │   0.00191│
└──────┴──────────┴──────────┴──────────┴──────────┘
```

The draw_mncontour method allows to compute parameter covariance plots

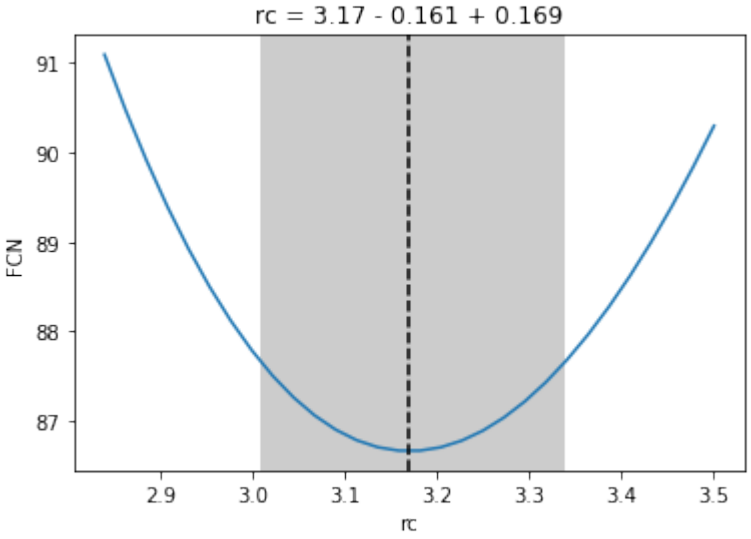
[19]:
```
fitobj.minuit.draw_mncontour('beta', 'rc', cl=(0.68, 0.9, 0.99))
```

[19]:
```
<matplotlib.contour.ContourSet at 0x7f5cf418c0a0>
```



Likelihood profiles for each parameter can be created using the draw_mnprofile method,

[20]:
```
rcval, fcnval = fitobj.minuit.draw_mnprofile('rc')
```



3.1.6 Setting constraints on the parameters

To help convergence or avoid bad results it is often useful to restrict the range of acceptable values for a parameter or fix its value. This can be done easily by accessing the *minuit* attribute of the Fitter class.

For instance, let's assume we want to fix the value of the *beta* parameter to 0.7 and refit

[21]:
```
fitobj.minuit.fixed['beta'] = True

fitobj.minuit.migrad()
```

[21]:

FCN = 86.67			Nfcn = 15613				
EDM = 1.38e-13 (Goal: 0.0002)							
Valid Minimum	Valid Parameters		No Parameters at limit				
Below EDM threshold (goal x 10)			Below call limit				
Covariance	Hesse ok		Accurate	Pos. def.	Not forced		

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	beta	0.658	0.019	-0.019	0.020			yes
1	rc	3.17	0.06	-0.16	0.17			
2	norm	-1.406	0.014	-0.017	0.017			
3	bkg	-3.750	0.026	-0.047	0.041			

	beta		rc		norm		bkg	
Error	-0.019	0.020	-0.16	0.17	-0.017	0.017	-0.05	0.04
Valid	True	True	True	True	True	True	True	True
At Limit	False	False	False	False	False	False	False	False
Max FCN	False	False	False	False	False	False	False	False
New Min	False	False	False	False	False	False	False	False

	beta	rc	norm	bkg
beta	0	0	0	0
rc	0	0.00338	-0.000729	-0.000651
norm	0	-0.000729	0.000199	8.67e-05
bkg	0	-0.000651	8.67e-05	0.000698

If on top of that we want to restrict parameter *rc* to lie within the range (0,10):

[22]:
```
fitobj.minuit.limits['rc'] = (0, 10)

fitobj.minuit.migrad()
```

[22]:

FCN = 86.67			Nfcn = 15649				
EDM = 6.24e-13 (Goal: 0.0002)							
Valid Minimum	Valid Parameters		No Parameters at limit				

(continues on next page)

(continued from previous page)

```
┌──────────────────────────────────────────────────────────────────────────────────────┐
│ Below EDM threshold (goal x 10) │           Below call limit           │
├─────────────────────────────────┼──────────────────────────────────────┤
│ Covariance │ Hesse ok │ Accurate │ Pos. def. │ Not forced │
└─────────────────────────────────┴──────────────────────────────────────┘
┌──┬──────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────┐
│  │ Name │  Value   │ Hesse Err│ Minos Err-│ Minos Err+│ Limit-   │ Limit+   │Fixed │
├──┼──────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────┤
│ 0│ beta │   0.658  │   0.019  │  -0.019  │   0.020  │          │          │yes   │
│ 1│ rc   │   3.17   │   0.06   │  -0.16   │    0.17  │    0     │   10     │      │
│ 2│ norm │  -1.406  │   0.014  │  -0.017  │   0.017  │          │          │      │
│ 3│ bkg  │  -3.750  │   0.026  │  -0.047  │   0.041  │          │          │      │
└──┴──────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────┘
┌──────────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┐
│          │          beta          │          rc            │         norm           │
│          bkg          │
├──────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│  Error   │  -0.019  │   0.020  │  -0.16   │   0.17   │ -0.017   │   0.017  │
│  -0.05   │   0.04   │
│  Valid   │   True   │   True   │   True   │   True   │   True   │   True   │
│  True    │   True   │
│ At Limit │  False   │  False   │  False   │  False   │  False   │  False   │
│  False   │  False   │
│ Max FCN  │  False   │  False   │  False   │  False   │  False   │  False   │
│  False   │  False   │
│ New Min  │  False   │  False   │  False   │  False   │  False   │  False   │
│  False   │  False   │
└──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘
┌──────┬──────────┬──────────┬──────────┬──────────┐
│      │     beta │       rc │     norm │      bkg │
├──────┼──────────┼──────────┼──────────┼──────────┤
│ beta │        0 │        0 │        0 │        0 │
│   rc │        0 │  0.00338 │-0.000729 │-0.000651 │
│ norm │        0 │-0.000729 │  0.00019 │ 8.67e-05 │
│  bkg │        0 │-0.000651 │ 8.67e-05 │  0.00069 │
└──────┴──────────┴──────────┴──────────┴──────────┘
```

3.1.7 Subtracting the background

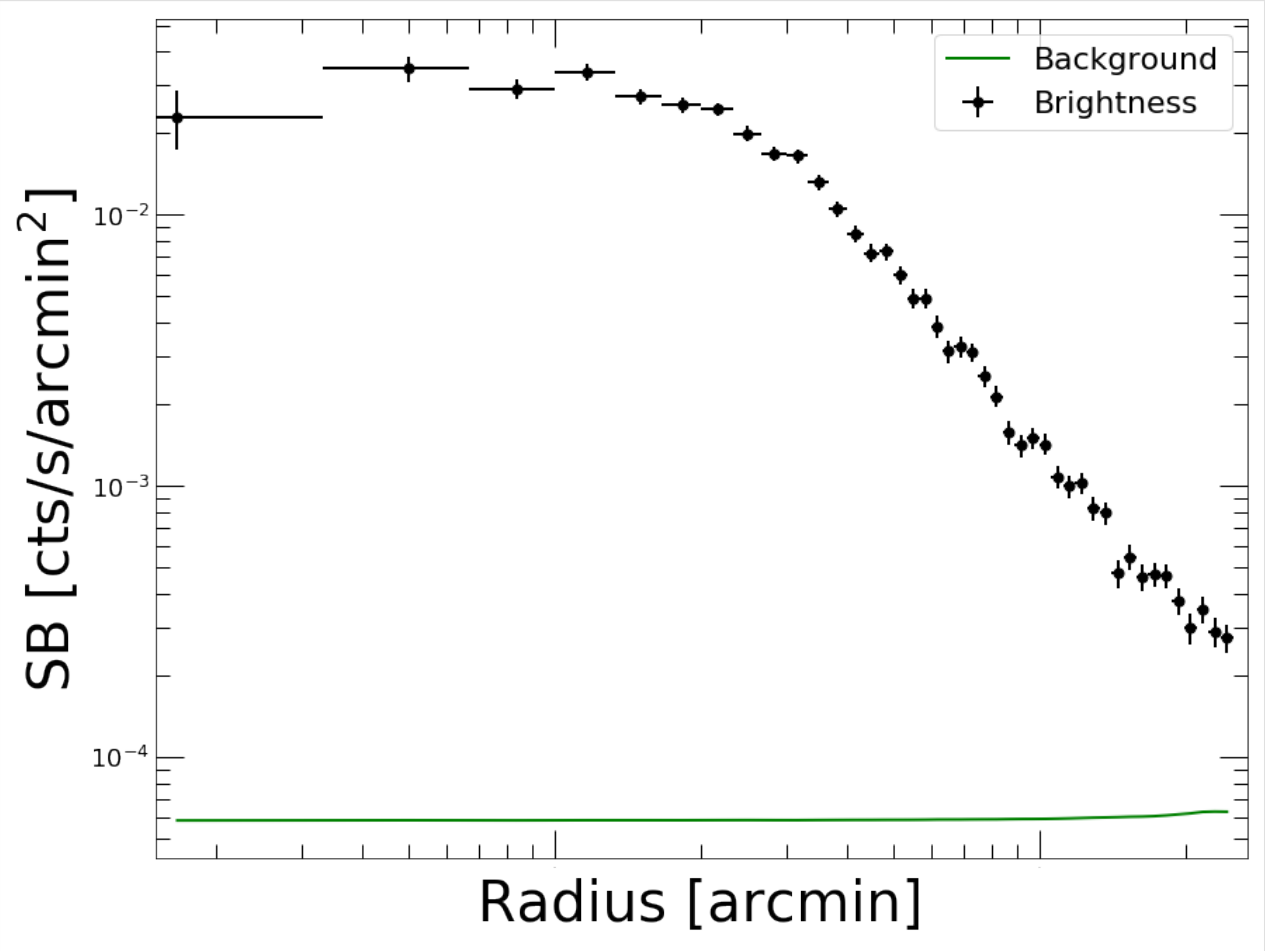
Now if we want to re-extract the profile with a different binning and subtract the background, we can create a new Profile object...

[23]:
```
p2=pyproffit.Profile(dat,center_choice='custom_ima',center_ra=prof.cx,center_dec=prof.cy,
                     maxrad=25.,binsize=20.,binning='log')

p2.SBprofile(ellipse_ratio=prof.ellratio, rotation_angle=prof.ellangle)
```

```
Corresponding FK5 coordinates:  55.72393261623938 -53.647031714234124
```

[24]:
```
p2.Plot()
```

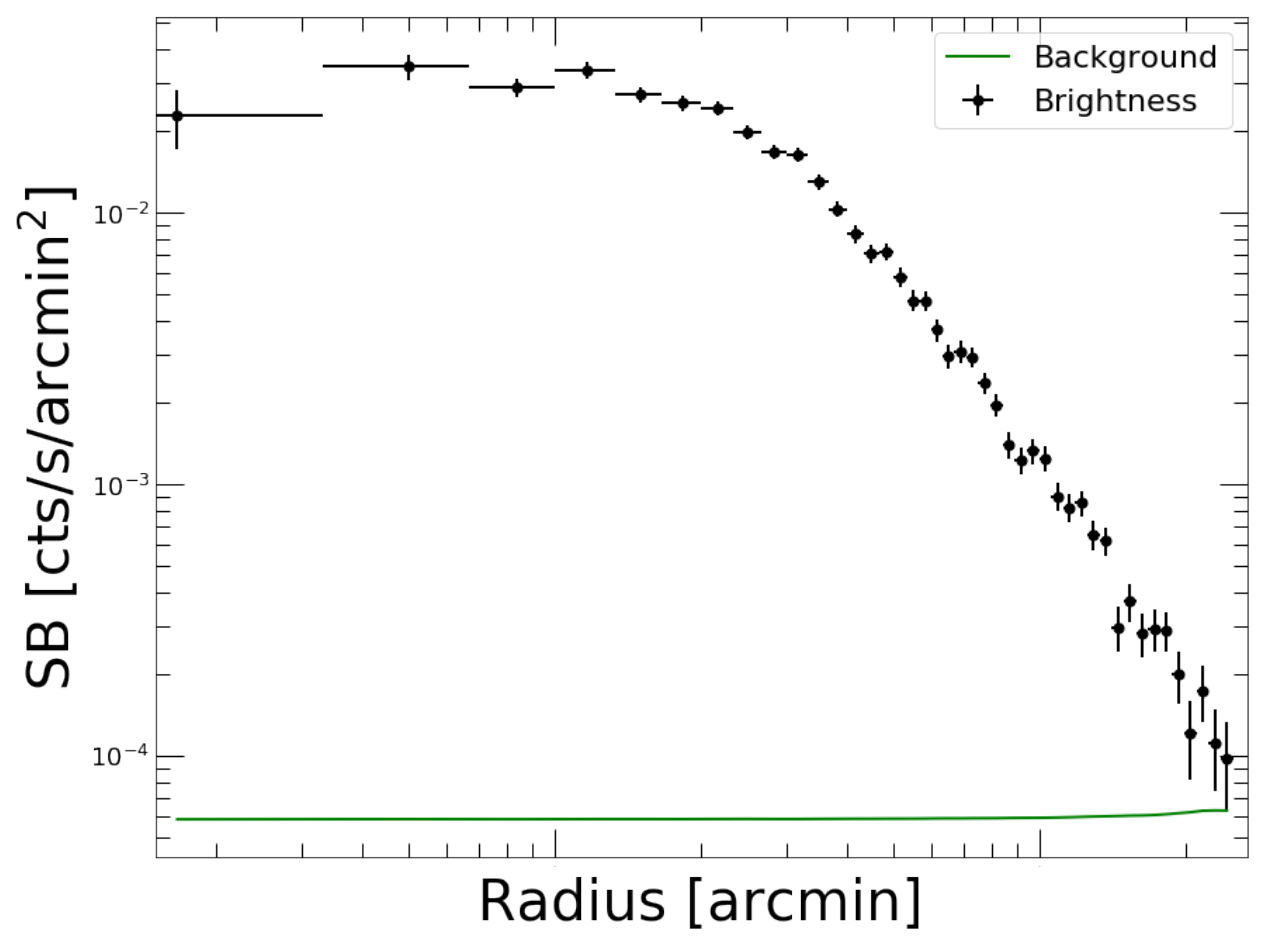
```
<Figure size 432x288 with 0 Axes>
```



We will use the results of the previous fit stored in the Fitter object to subtract the background. The Backsub method of the Profile class reads the *bkg* parameter, subtracts its value from the brightness profile and adds the statistical error in quadrature:

[25]:
```
p2.Backsub(fitobj)
p2.Plot()
```

```
<Figure size 432x288 with 0 Axes>
```



3.1.8 Fitting with a custom function

To fit the data with any custom function we can simply define a Python function with the desired model and pass it to a PyProffit Model object, with the following structure:

[26]:
```
# Create your own model, here some sort of cuspy beta model

def myModel(x,beta,rc,alpha,norm,bkg):
    term1 = np.power(x/rc,-alpha) * np.power(1. + (x/rc) ** 2, -3 * beta + alpha/2.)
    n2 = np.power(10., norm)
    b2 = np.power(10., bkg)
    return n2 * term1 + b2
```

[27]:
```
# Pass the user-defined function to a new Model structure

cuspmod = pyproffit.Model(myModel)
```

Now we fit the model to the data, as done previously.

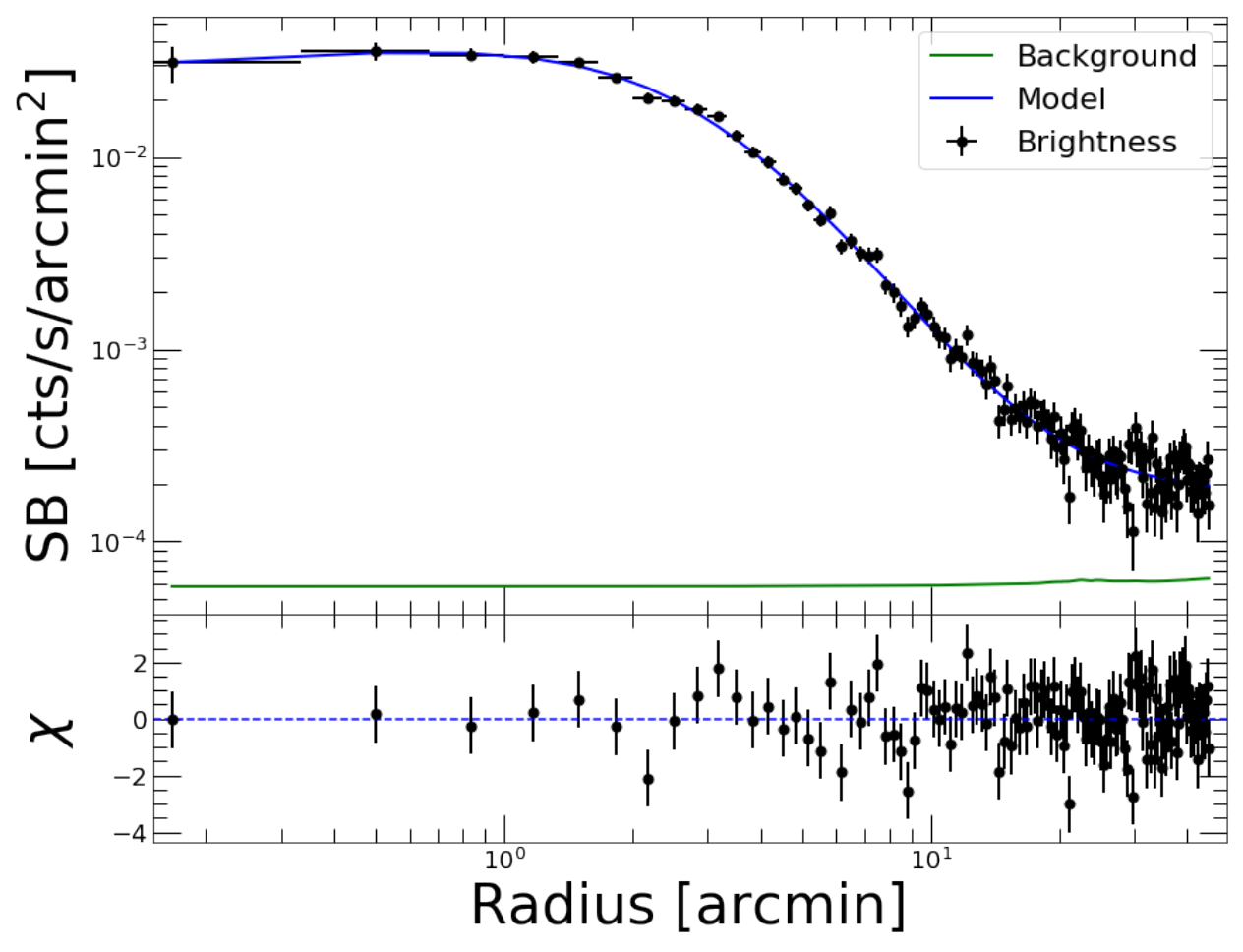
We can also fix the value of some of the parameters to help the convergence. This is done easily by modifying the *iminuit.fixed* variable for the parameter, as shown above.

[28]:
```
fitcusp = pyproffit.Fitter(model=cuspmod, profile=prof, beta=0.35, rc=1., alpha=0.,
↪norm=-1.7, bkg=-3.8)
fitcusp.Migrad()
```

```
┌──────────────────────────────────┬──────────────────────────────────────┐
│ FCN = 140.6                      │              Nfcn = 407              │
│ EDM = 3.49e-05 (Goal: 0.0002)    │                                      │
├──────────────┬───────────────────┼──────────────────────────────────────┤
│ Valid Minimum │ Valid Parameters │        No Parameters at limit        │
├──────────────┴───────────────────┼──────────────────────────────────────┤
│ Below EDM threshold (goal x 10)  │           Below call limit           │
├──────────────┬─────────────┬─────┴────────┬─────────────┬─────────────────┤
│  Covariance  │     Hesse ok      │ Accurate  │  Pos. def.  │ Not forced │
└──────────────┴─────────────┴──────────┴─────────┴─────────┘
┌───┬──────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┐
│   │ Name │   Value  │ Hesse Err │ Minos Err- │ Minos Err+ │ Limit-  │ Limit+  │
↪Fixed │
├───┼──────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ 0 │ beta │   0.490  │   0.017  │          │          │          │          │
↪   │
│ 1 │ rc   │   2.95   │   0.23   │          │          │          │          │
↪   │
│ 2 │ alpha │  -0.14  │   0.09   │          │          │          │          │
↪   │
│ 3 │ norm │   -1.33  │   0.05   │          │          │          │          │
↪   │
│ 4 │ bkg  │  -3.742  │   0.021  │          │          │          │          │
↪   │
└───┴──────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘
┌──────┬──────────┬──────────┬──────────┬──────────┬──────────┐
│      │     beta │       rc │    alpha │     norm │      bkg │
├──────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ beta │ 0.000303 │  0.00353 │ 0.000822 │ -0.00054 │ 0.000253 │
│   rc │  0.00353 │   0.0533 │   0.0172 │  -0.0105 │  0.00247 │
│alpha │ 0.000822 │   0.0172 │  0.00819 │  -0.0043 │ 0.000495 │
│ norm │ -0.00054 │  -0.0105 │  -0.0043 │   0.0025 │-0.000328 │
│  bkg │ 0.000253 │  0.00247 │ 0.000495 │-0.000328 │ 0.000438 │
└──────┴──────────┴──────────┴──────────┴──────────┴──────────┘
Best fit chi-squared: 140.643 for 135 bins and 130 d.o.f
Reduced chi-squared: 1.08187
```

[29]:
```
prof.Plot(model=cuspmod)
```

```
<Figure size 432x288 with 0 Axes>
```



3.2 Example: Deprojection, gas density profiles and gas masses

This thread shows how to use *PyProffit* to extract gas density profiles, gas masses, integrated count rates and luminosities from imaging data. This code implements the method presented in Eckert et al. (2020) to apply deprojection and PSF deconvolution.

We start by loading the data and extracting a profile...

[1]:
```
import numpy as np
import pyproffit
import matplotlib.pyplot as plt
```

[2]:
```
import os

# Change this to the proper directory containing the test data
os.chdir('../../validation/')
```

Let's load the data into a Data structure...

[3]:
```
dat=pyproffit.Data(imglink='b_37.fits.gz',explink='expose_mask_37.fits.gz',
                   bkglink='back_37.fits.gz')
```

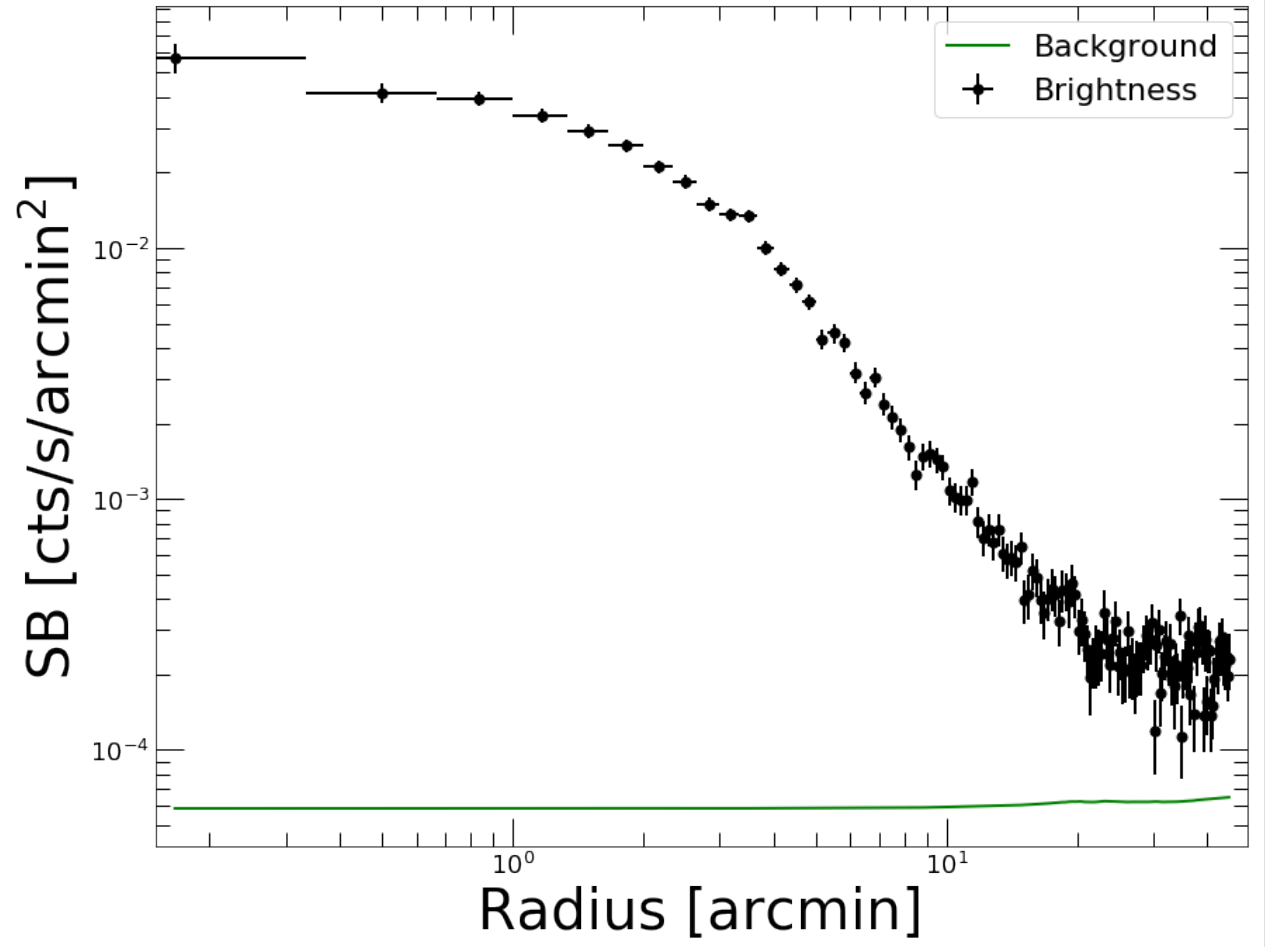
```
WARNING: FITSFixedWarning: RADECSYS= 'FK5 ' / Equatorial system reference
the RADECSYS keyword is deprecated, use RADESYSa. [astropy.wcs.wcs]
```

Now we extract a profile in circular annuli from the surface brightness peak...

[4]:
```
prof=pyproffit.Profile(dat,center_choice='peak',maxrad=45.,binsize=20.)

prof.SBprofile()

prof.Plot()
```

```
Determining X-ray peak
Coordinates of surface-brightness peak: 272.0 281.0
Corresponding FK5 coordinates:  55.72147733144434 -53.628226297404545
```

```
<Figure size 432x288 with 0 Axes>
```



3.2.1 PSF modeling

To correct for PSF smearing, we need to create a PSF mixing matrix which describes the leaking of photons from each annulus to the others. The shape of the PSF needs to be known a priori. The definition of the PSF can be done either by providing an input image or through an analytical function.

In the following example we model the ROSAT/PSPC PSF as a King function with a core radius of 25 arcsec

[5]:
```
def fking(x):
    r0=25./60. # arcmin
    alpha=1.5 # King slope
    return np.power(1. + (x/r0) ** 2, -alpha)
```

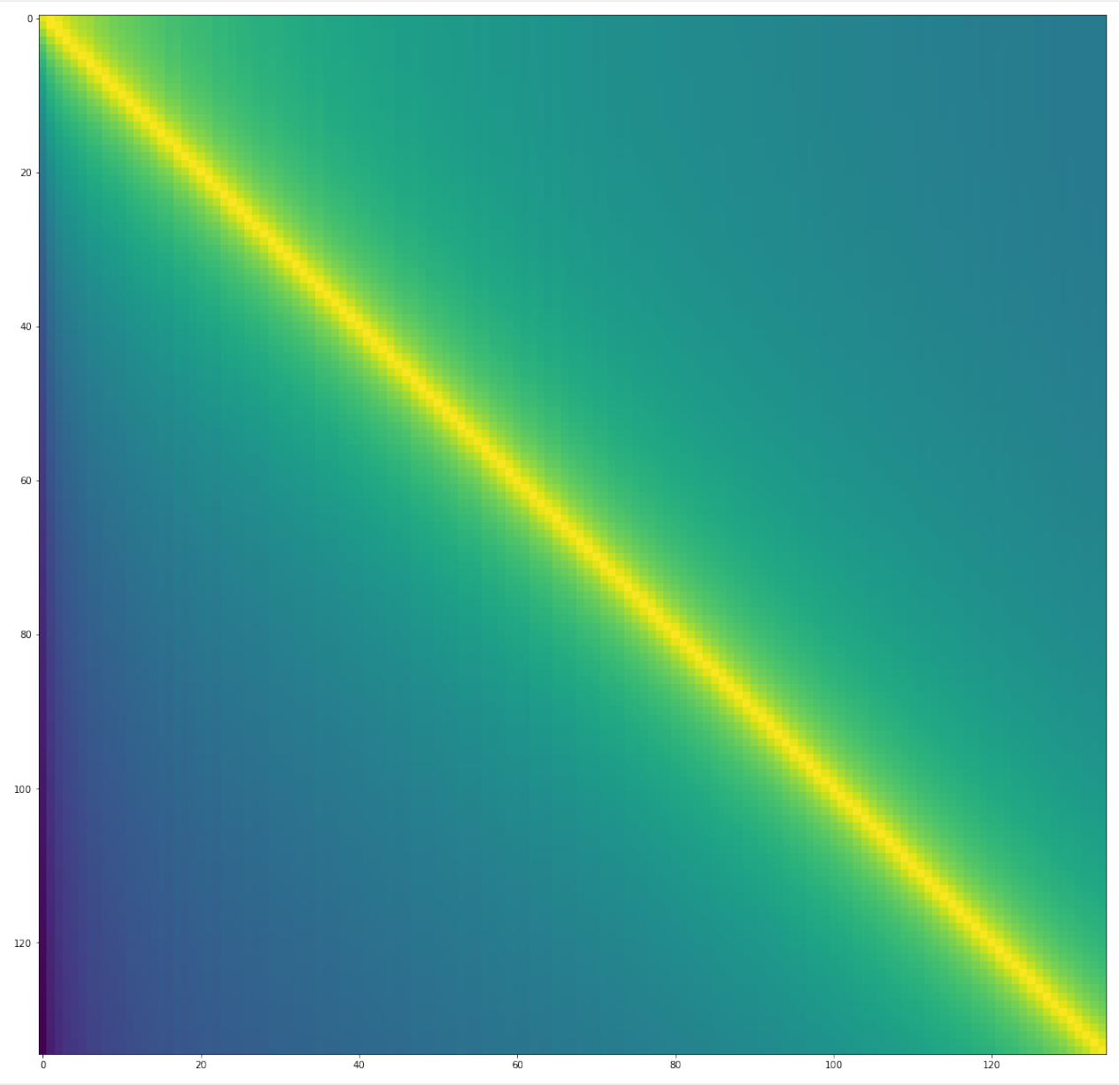
Now we pass this function to the PSF method of the Profile class. To create the mixing matrix, the method creates normalized images of each annulus individually, convolves them with the PSF using FFT and computes the fraction of events in the FFT-convolved image that fall within each annulus. See Eckert et al. (2020) for more details.

[6]:
```
prof.PSF(psffunc = fking)
```

Let's inspect the PSF mixing matrix. Each row describes the fraction originating from that annulus which are in fact recorded in any other row

[7]:
```
fig = plt.figure(figsize=(20,20))
plt.imshow(np.log10(prof.psfmat), aspect='auto')
```

[7]:
```
<matplotlib.image.AxesImage at 0x7f8ab85e3310>
```



3.2.2 Deprojection

For deprojection, PyProffit implements both the standard onion-peeling deprojection similar to that of plain Proffit and a new method based on multiscale decomposition of the observed profile. The new method is suitable in the low count-rate regime, provides on-the-fly propagation of the background value and PSF deconvolution.

The two methods can be accessed with the Deproject class. For the extraction of density profiles the class requires the following input:

- profile=prof: a Profile object containing the data
- z=redshift: the redshift of the source
- cf=factor: the conversion between count rate and emission measure

The conversion factor can be computed using the calc_emissivity function, which goes through XSPEC to simulate an absorbed APEC model with an emission measure of 1 and retrieve the corresponding count rate. XSPEC needs to be accessible in the PATH for this command to work properly.

Note: Since we have corrected from vignetting using the exposure map, the count rates are all rescaled to equivalent *on-axis* count rates. Therefore the *on-axis* effective area of the telescope must be used to convert correctly count rates into emission measure.

Note: In some cases (in particular *Chandra*), the exposure maps are usually provided in units of cm^2 s instead of just s, such that the unit of the surface brightness measured by *PyProffit* is phot/cm^2/s instead of count/s. In this case, the *type="cr"* option should be used.

[8]:
```
z_a3158 = 0.059 # Source redshift, here 0.059 for the test cluster A3158
kt_a3158 = 5.0 # Plasma temperature; if a soft band is used the profile is mildly␣
↪dependent on it
nh_a3158 = 0.0118 # Source NH in 1e22 cm**(-2) unit
rsp = 'pspcb_gain2_256.rsp' # Response file, here ROSAT/PSPC in RSP format
elow = 0.42 # Lower energy boundary of the image
ehigh = 2.01 # Upper energy boundary of the image

cf = prof.Emissivity(z=z_a3158,
                     kt=kt_a3158,
                     nh=nh_a3158,
                     rmf=rsp,
                     elow=elow,
                     ehigh=ehigh)

print(cf)
```

```
43.53
```

We are now ready to declare the Deproject object. Note that in case the redshift and conversion factor are not known, it is still possible to run the PSF deconvolution and profile reconstruction, however the gas density profile and gas mass cannot be computed.

[9]:
```
depr = pyproffit.Deproject(z=z_a3158, cf=cf, profile=prof)
```

Let's start with the multiscale decomposition method. It can be launched with the Multiscale method of the Deproject class. The parameters of the method are the following:

- backend='pymc3': choose whether the optimization will be performed with the PyMC3 or the Stan backend
- nmcmc=1000: number of points in Hamiltonian Monte Carlo chain
- tune=500: number of tuning steps to set up the NUTS sampler
- bkglim=rad: radius beyond which it is assumed that the source is 0 (i.e. background only)
- samplefile=file.dat: output file where the HMC samples will be stored

The sampling time with HMC will depend on a number of factors, including the number of bins in the profile, the number of points in the output chain, and the bkglim value.

[10]:
```
depr.Multiscale(nmcmc=1000, tune=1000, bkglim=30.)
```

```
Running MCMC...
```

```
<IPython.core.display.HTML object>
```

```
/home/deckert/.local/lib/python3.8/site-packages/pyproffit/deproject.py:628:␣
↪FutureWarning: In v4.0, pm.sample will return an `arviz.InferenceData` object␣
↪instead of a `MultiTrace` by default. You can pass return_inferencedata=True or␣
↪return_inferencedata=False to be safe and silence this warning.
  trace = pm.sample(nmcmc, tune=tune, start=tm)
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [bkg, coefs]
```

```
<IPython.core.display.HTML object>
```

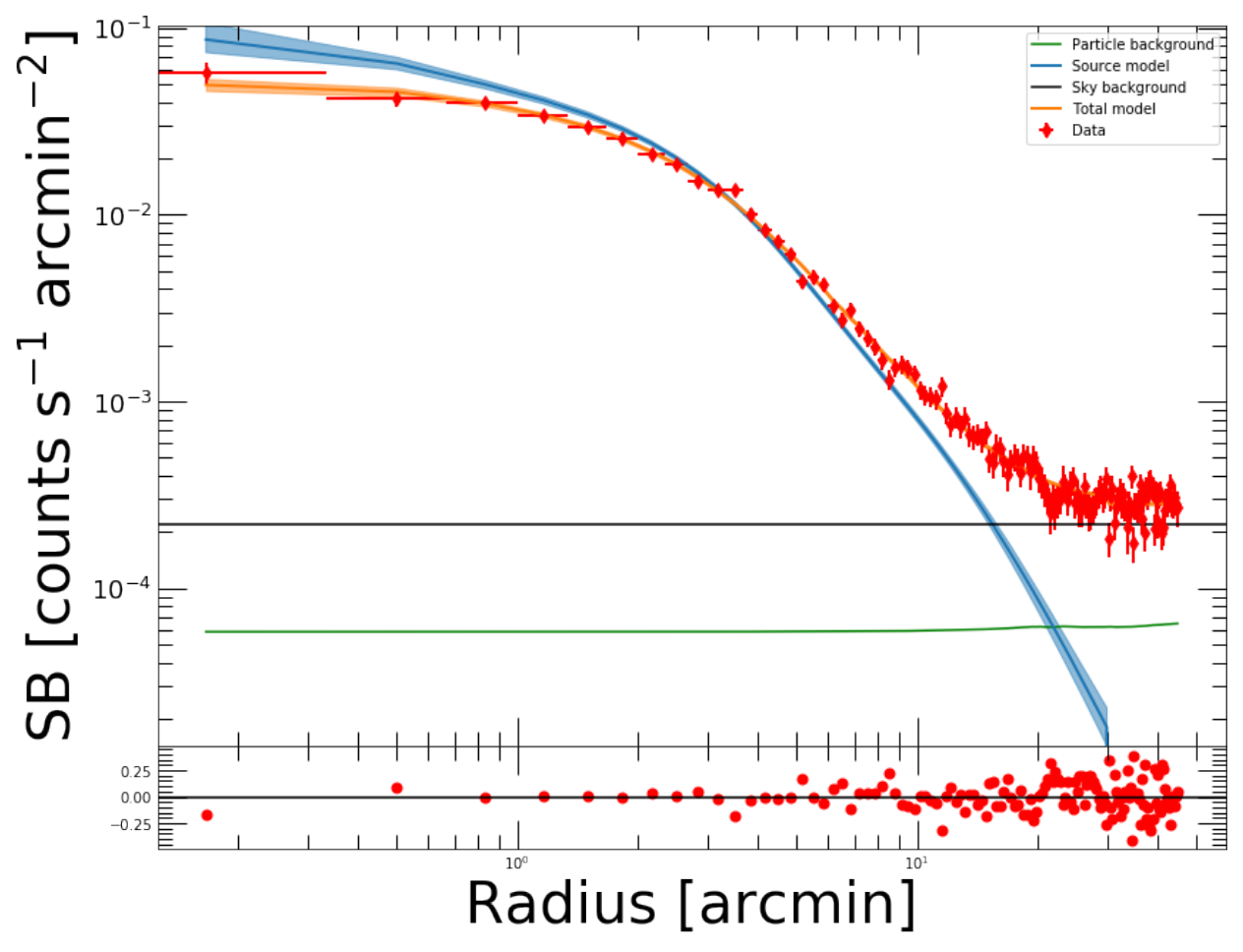
```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws␣
↪total) took 171 seconds.
There were 738 divergences after tuning. Increase `target_accept` or reparameterize.
There were 841 divergences after tuning. Increase `target_accept` or reparameterize.
There were 861 divergences after tuning. Increase `target_accept` or reparameterize.
There were 934 divergences after tuning. Increase `target_accept` or reparameterize.
The rhat statistic is larger than 1.05 for some parameters. This indicates slight␣
↪problems during sampling.
The estimated number of effective samples is smaller than 200 for some parameters.
```

```
Done.
 Total computing time is:  2.9870954672495524  minutes
```

The quality of the restruction can be viewed through the PlotSB method of the Deproject class

[11]:
```
depr.PlotSB()
```

```
<Figure size 432x288 with 0 Axes>
```



Here the total model (PSF convolved source + background) is shown in orange. The reconstructed source profile (PSF deconvolved) is shown in blue, and the fitted sky background is in black. The residuals (bottom panel) allow the user to assess the quality of the reconstruction.

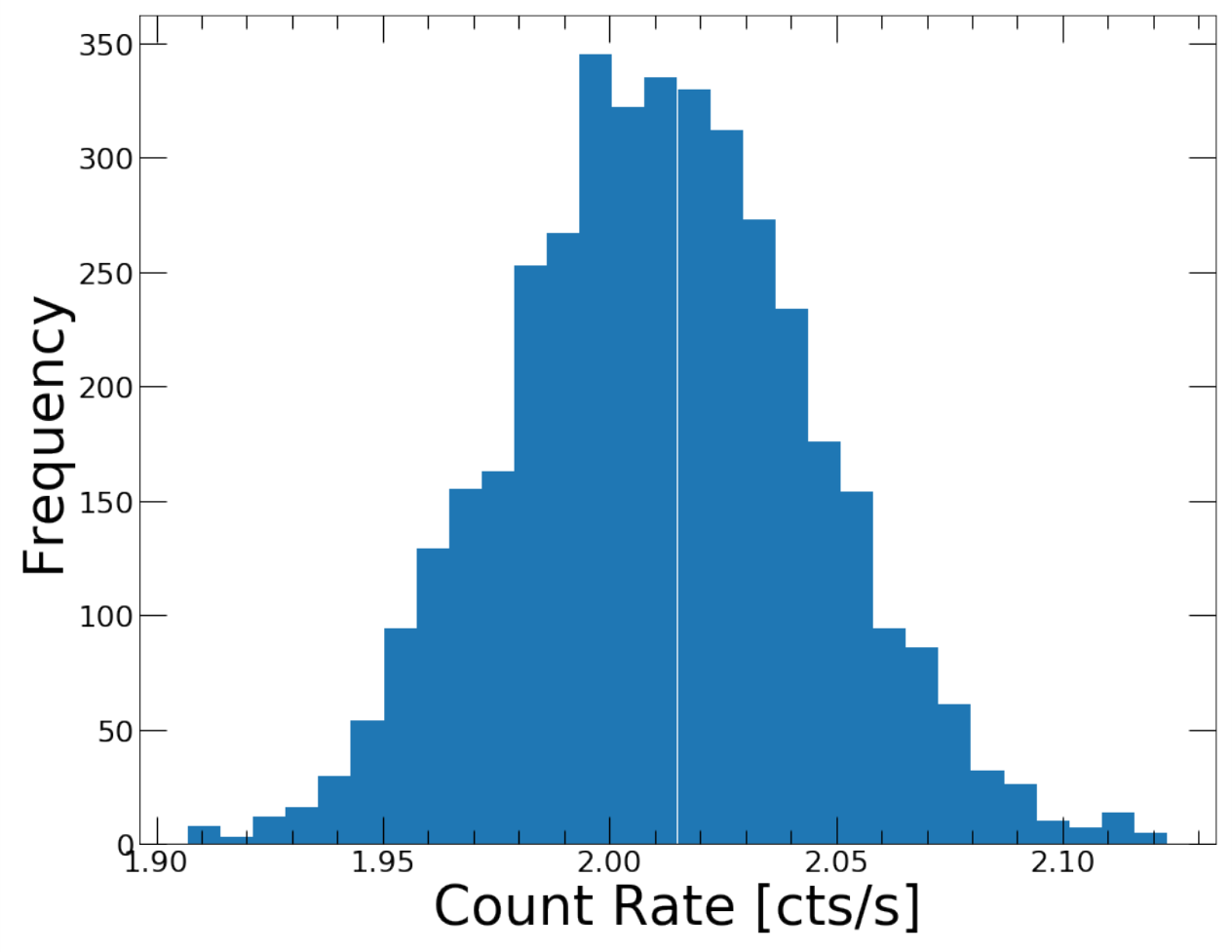
3.2.3 Count Rates and luminosities

The count rates can be computed easily from the reconstructed surface brightness profile within any user given apertures. This is done through the CountRate method, which integrates the PSF deconvolved model over the area. The posterior distribution of the count rate can be displayed as well.

[12]:
```
cr, cr_lo, cr_hi = depr.CountRate(0., 30.)
```

```
Reconstructed count rate: 2.01158 (1.97752 , 2.04536)
```

```
<Figure size 432x288 with 0 Axes>
```

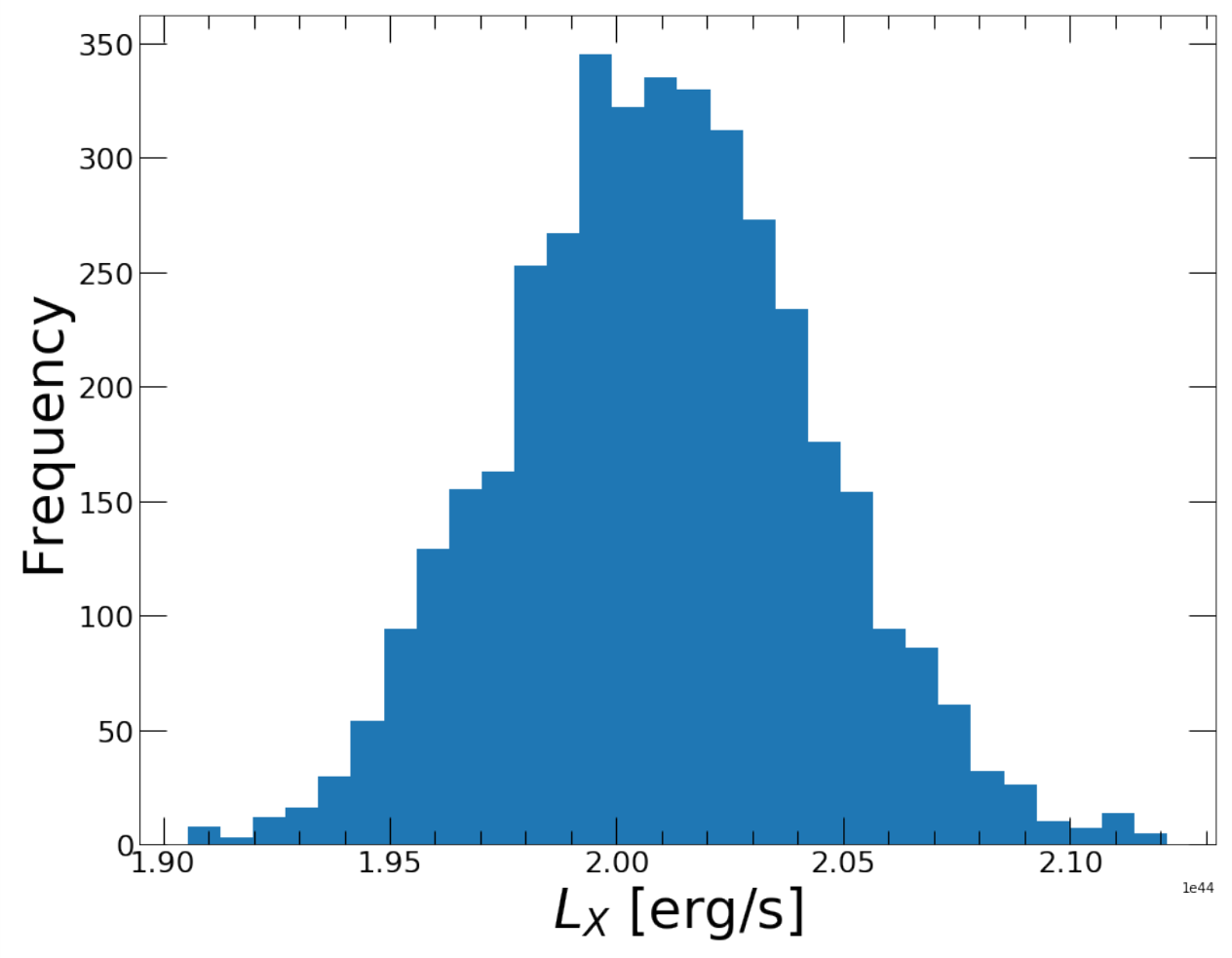


And the luminosity can be obtained similarly through the Luminosity method. Similarly to the gas density, the luminosity requires the emissivity conversion to be calculated

[13]:
```
lum, lum_lo, lum_hi = depr.Luminosity(0., 30.)
```

```
Reconstructed luminosity: 2.01001e+44 (1.97598e+44 , 2.04376e+44)
```

```
<Figure size 432x288 with 0 Axes>
```

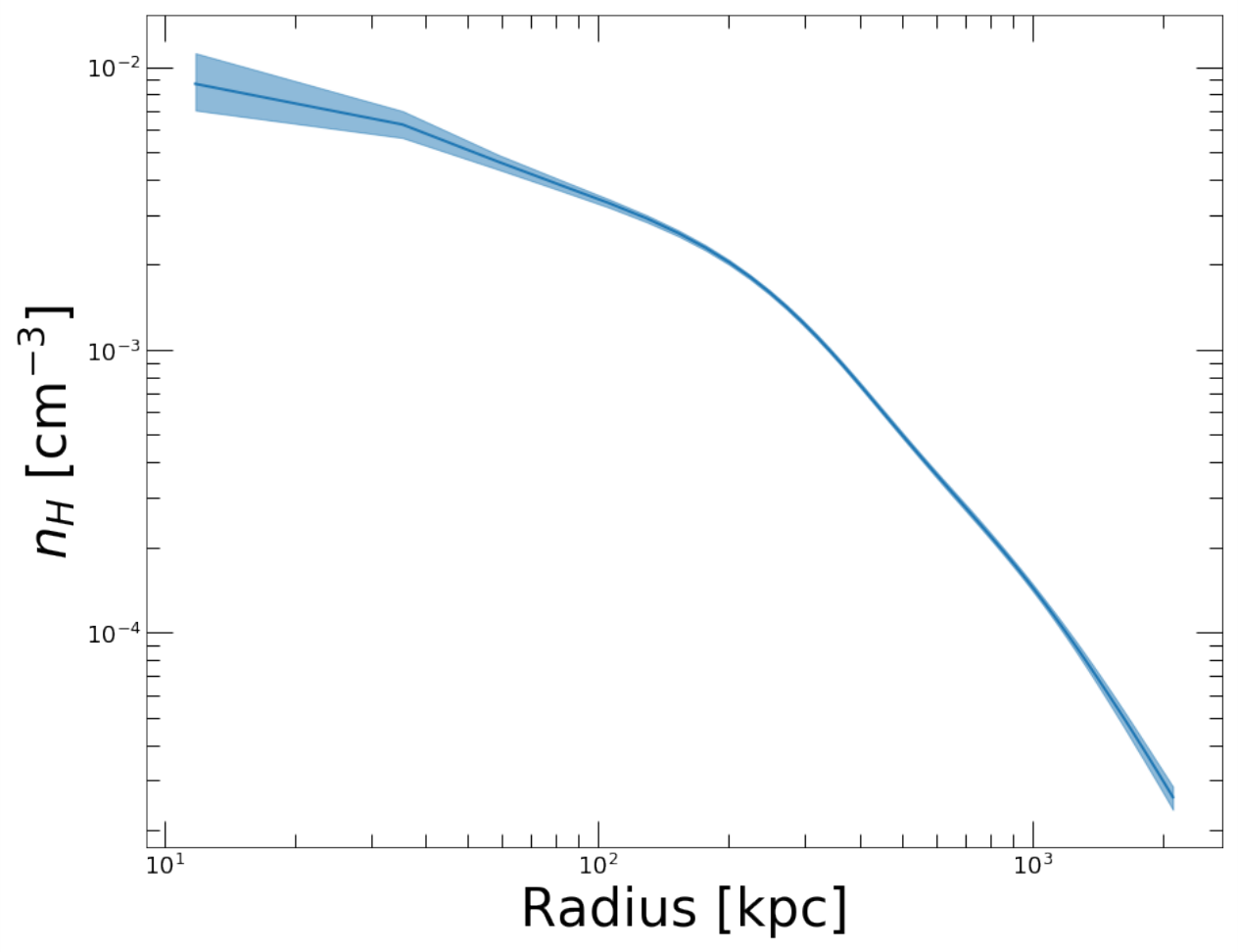


3.2.4 Gas density profile

Once a Multiscale reconstruction has been performed, and if the source redshift and the emission measure conversion factor have been provided, it is straightforward to measure the gas density profile of the source. This is done through the Density method of the Deproject class. The gas density profile can then be plotted through the PlotDensity method

[14]:
```
depr.Density()

depr.PlotDensity()
```

```
<Figure size 432x288 with 0 Axes>
```



3.2.5 Onion Peeling deprojection

If instead of the multiscale approach one wishes to compute the deprojected profile using the classical *onion peeling* approach, in which case the projection kernel is directly inverted, the Deproject class contains the OnionPeeling method.

Note that in this case the background is not reconstructed on-the-fly, thus this method should be used directly on background subtracted profiles. Here we provide an example of the use of the OnionPeeling method. First, let us fit the surface brightness profile beyond 30 arcmin with a constant,

[15]:
```
mod = pyproffit.Model(pyproffit.Const)

fitconst = pyproffit.Fitter(model=mod, profile=prof, fitlow=30., fithigh=40., bkg=-3.
↪5)

fitconst.Migrad()
```

```
┌──────────────────────────────────┬──────────────────────────────────────┐
│ FCN = 47.4                       │              Nfcn = 22               │
│ EDM = 5.94e-06 (Goal: 0.0002)    │                                      │
├──────────────┬───────────────────┼──────────────────────────────────────┤
│ Valid Minimum │ Valid Parameters │        No Parameters at limit        │
```

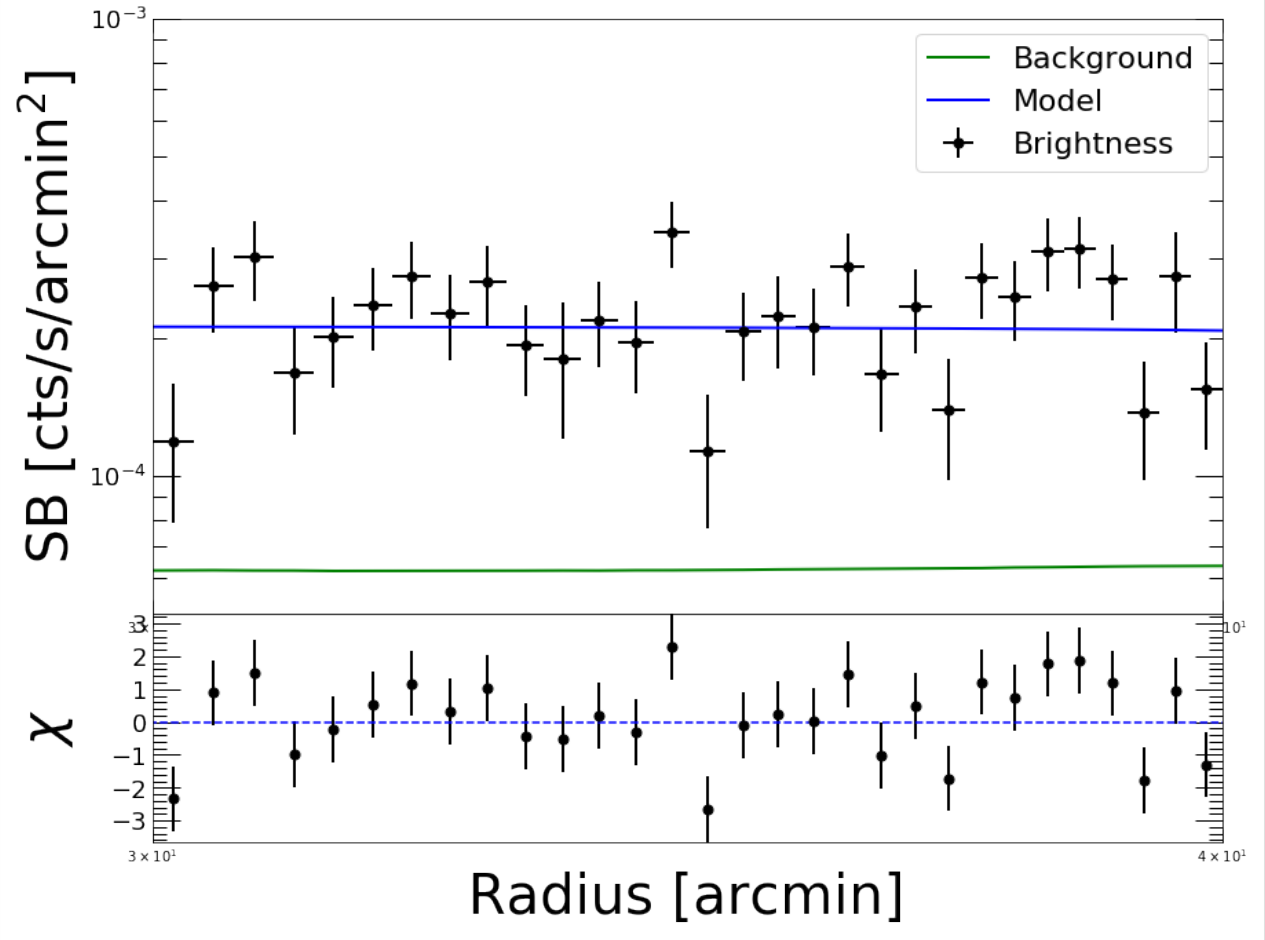
(continues on next page)

(continued from previous page)

```
┌──────────────────────────────────────────────────────────────────────┐
│ Below EDM threshold (goal x 10) │           Below call limit         │
├─────────────────────────────────┼────────────────────────────────────┤
│ Covariance │ Hesse ok │ Accurate │ Pos. def. │ Not forced │
└─────────────────────────────────┴────────────────────────────────────┘
┌──┬──────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────┐
│  │ Name │  Value   │ Hesse Err│ Minos Err-│ Minos Err+│ Limit-   │ Limit+   │Fixed │
├──┼──────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────┤
│ 0│ bkg  │  -3.670  │   0.018  │          │          │          │          │      │
└──┴──────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────┘
┌──────┬──────────┐
│      │      bkg │
├──────┼──────────┤
│ bkg  │ 0.000322 │
└──────┴──────────┘
Best fit chi-squared: 47.3985 for 135 bins and 134 d.o.f
Reduced chi-squared: 0.35372
```

[16]:
```
prof.Plot(model=mod, axes=[30., 40., 5e-5, 1e-3])
```

```
<Figure size 432x288 with 0 Axes>
```



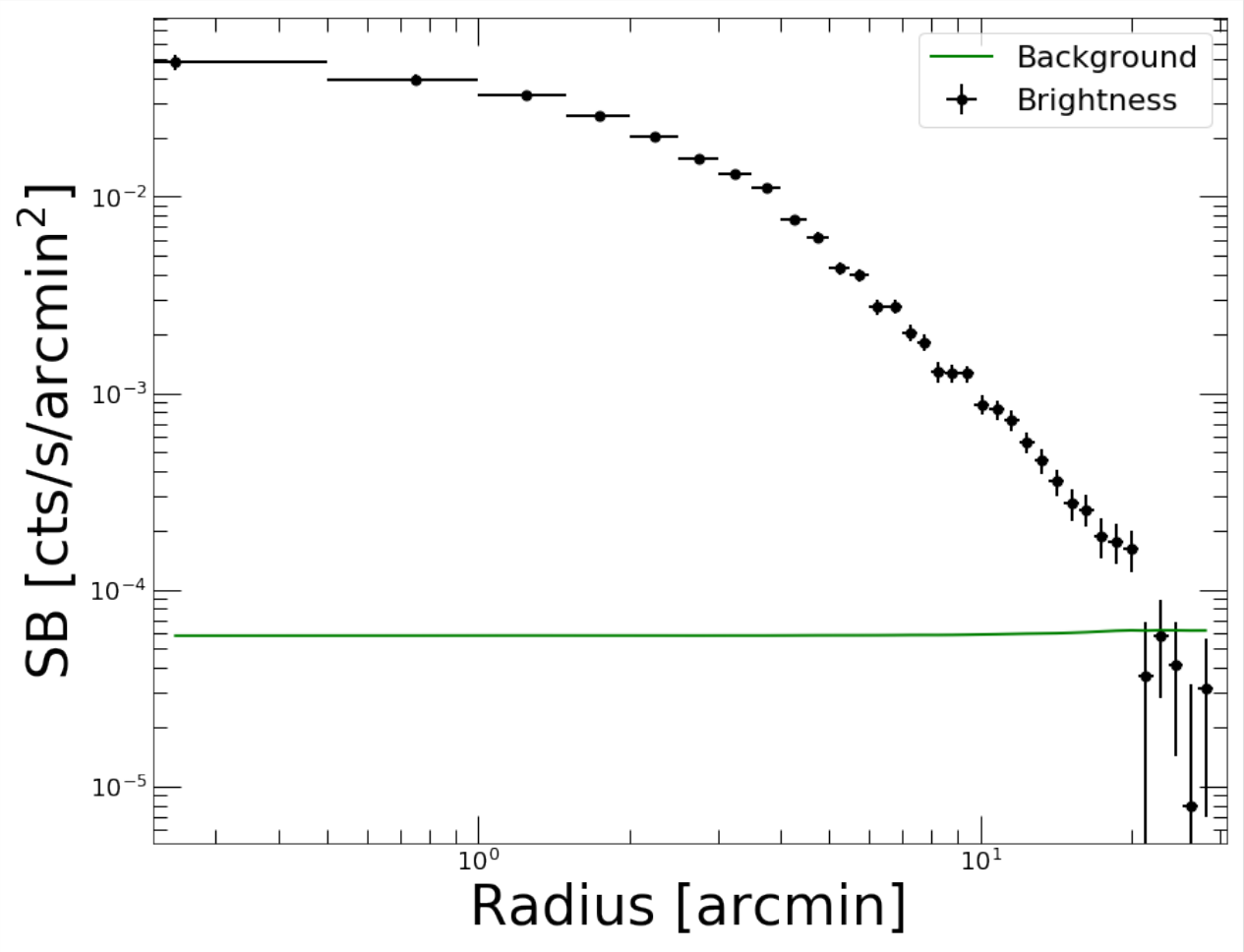
Now we define a new Profile object with a logarithmic binning, from which we will subtract the fitted background

[17]:
```
prof2 = pyproffit.Profile(dat, center_choice='peak', binsize=30, maxrad=30., binning=
↪'log')

prof2.SBprofile()

prof2.Backsub(fitconst)
```

```
Determining X-ray peak
Coordinates of surface-brightness peak: 272.0 281.0
Corresponding FK5 coordinates:  55.72147733144434 -53.628226297404545
```

[18]:
```
prof2.Plot()
```

```
<Figure size 432x288 with 0 Axes>
```

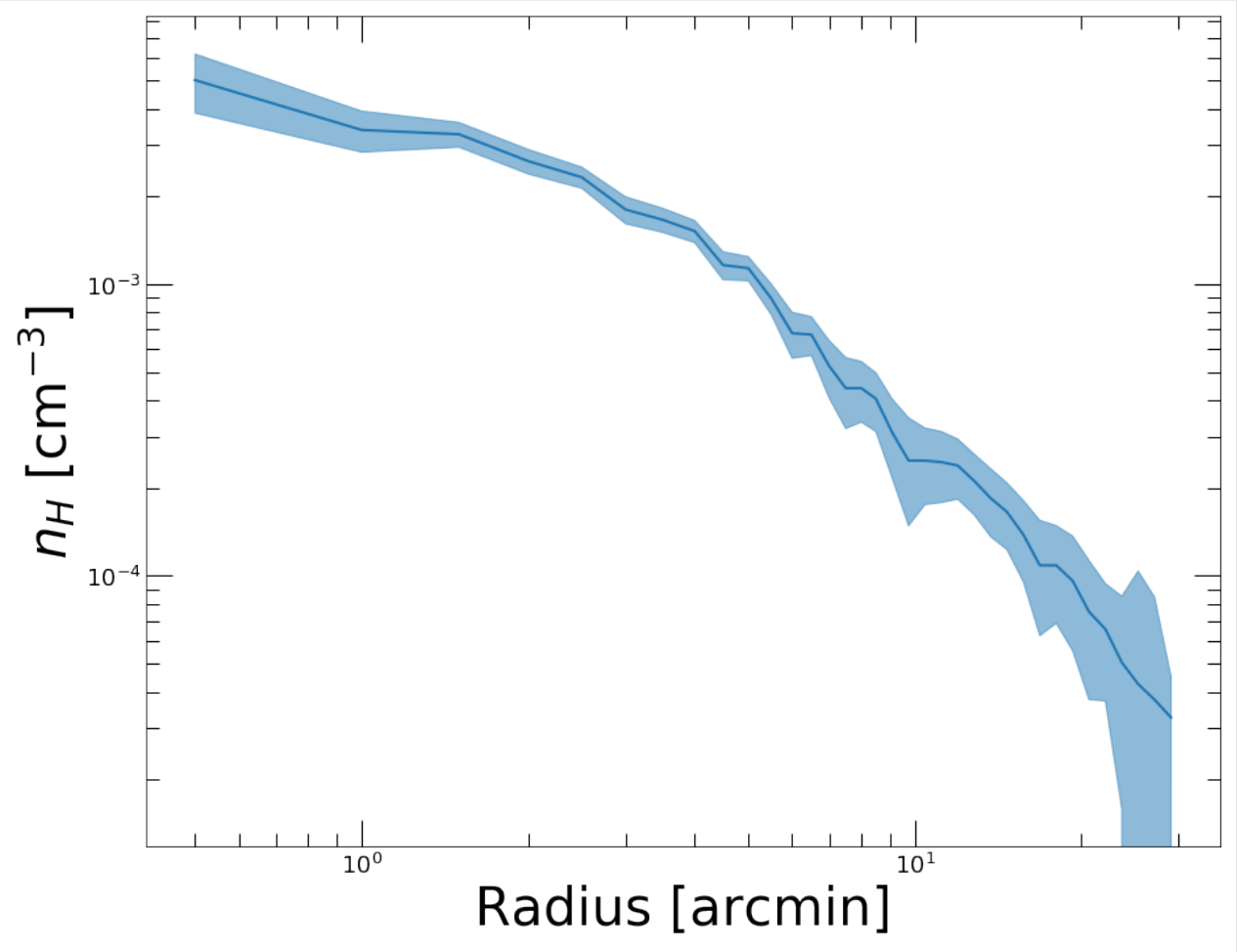


Now we are ready to define a new Deproject object and apply the OnionPeeling method,

[19]:
```
depr_op = pyproffit.Deproject(profile=prof2, cf=cf, z=z_a3158)

depr_op.OnionPeeling()
```

[20]:
```
depr_op.PlotDensity(xunit='arcmin')
```

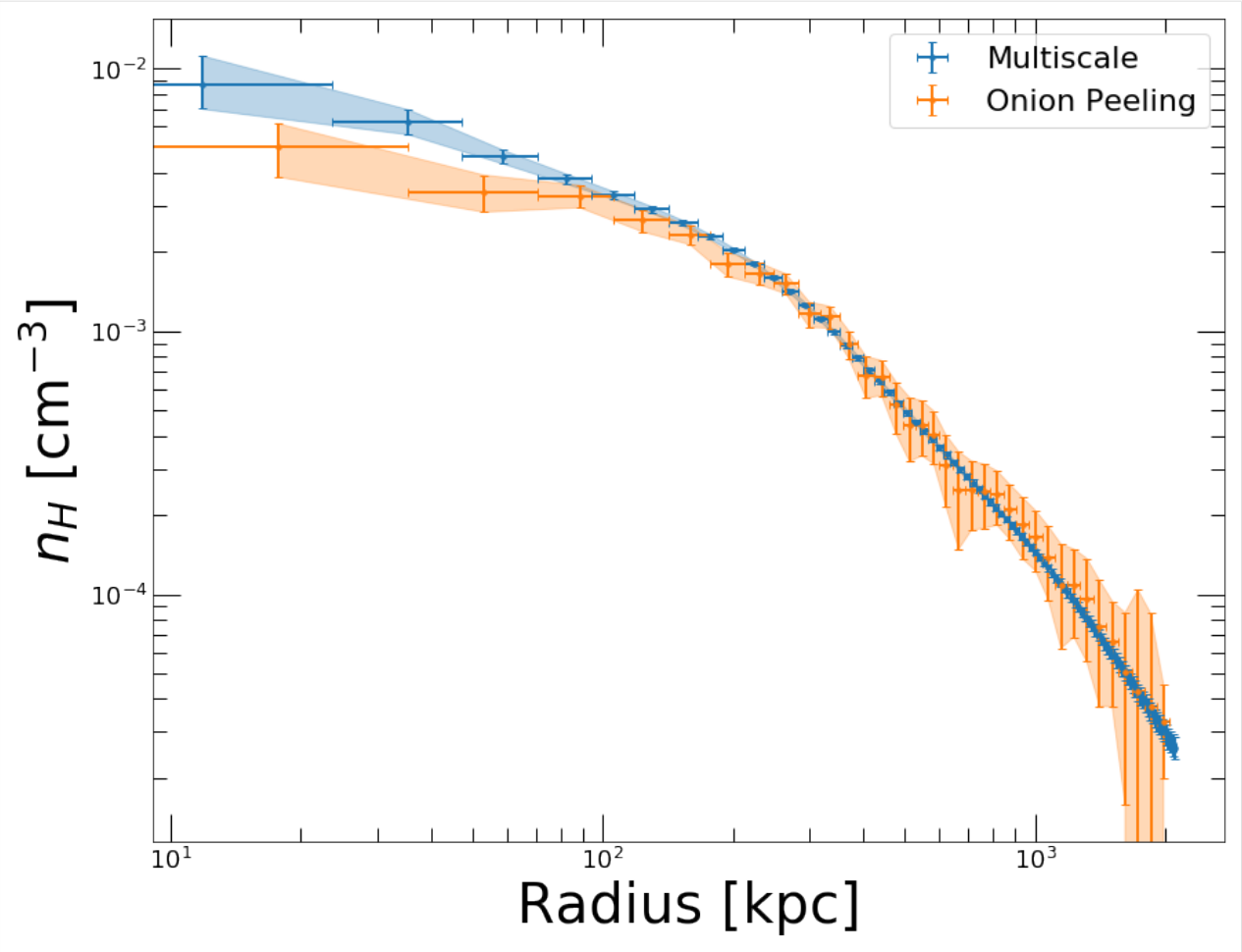
```
<Figure size 432x288 with 0 Axes>
```



Clearly we want to know how the two methods compare. The plot_multi_methods function allows the user to easily compare the results of several density profile reconstructions

[21]:
```
outfig = pyproffit.plot_multi_methods(deps = (depr, depr_op),
                            profs = (prof, prof2),
                            labels = ('Multiscale', 'Onion Peeling'))
```

```
Showing 2 density profiles
```



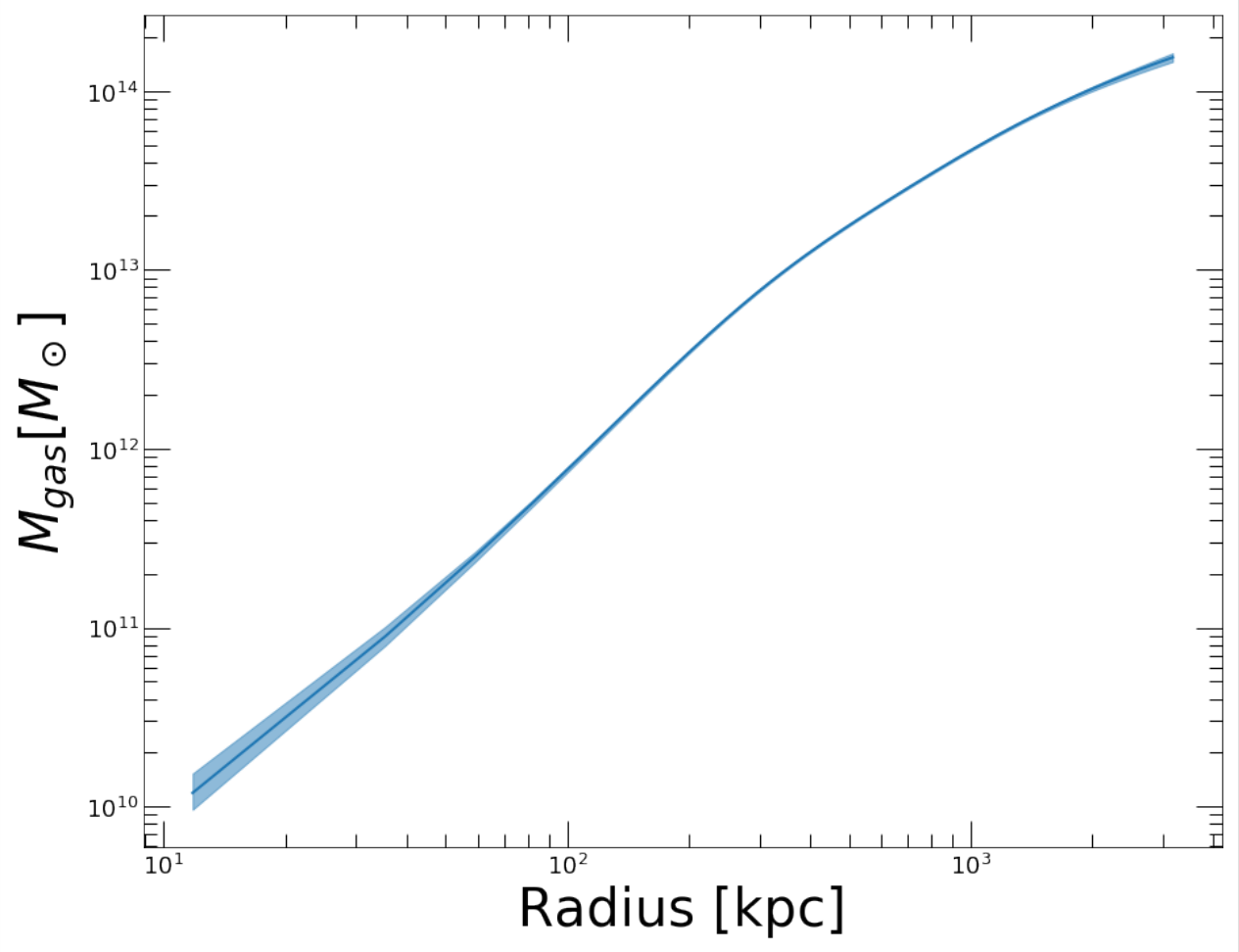
Clearly the two methods are consistent, but the Multiscale approach is much less noisy. In the central regions we can easily notice the effect of the PSF deconvolution in the Multiscale case; in the OnionPeeling case no PSF deconvolution can be applied.

If instead of PyMC3 one wishes to use the (usually more computationally efficient) Stan backend, it is easy to set it up when calling Multiscale by using the *backend='stan'* option. The results of PyMC3 and Stan are usually indistinguishable.

3.2.6 Gas masses

The gas mass profile is the integral of the gas density profile over the volume. The Deproject class contains two methods to compute the gas mass profile and the posterior distribution of M_{gas} evaluated at a specific radius. PlotMgas allows the user to view the total reconstructed M_{gas} profile, wherease Mgas computes M_{gas} at any user given radius (in kpc) and plot the posterior distribution of this value.

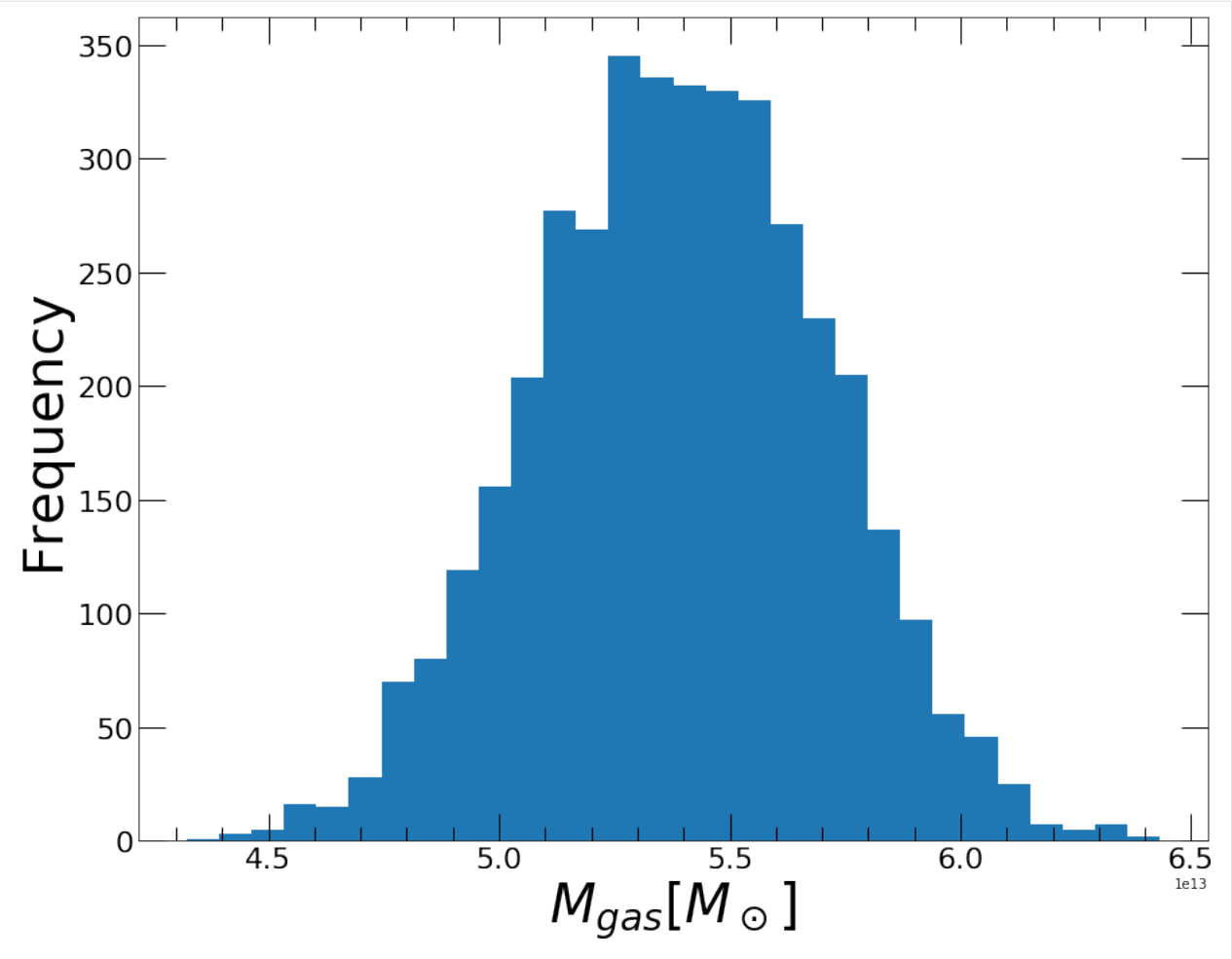
[22]:
```
depr.PlotMgas()
```



Now let's say for instance that we want to compute M_{gas} at $R_{500} = 1123 \pm 50$ kpc. The Mgas method evaluates the gas mass at the provided overdensity radius. The uncertainty in the overdensity radius can be propagated to the posterior M_{gas} distribution by randomizing the radius out to which the profile is integrated

[23]:
```
mg_r500, mg_lo, mg_hi, rho = depr.Mgas(radius = 1123., radius_err=50.)
```

```
<Figure size 432x288 with 0 Axes>
```



3.3 Example: Modeling surface brightness discontinuities

This tutorial shows how to use PyProffit to model surface brightness discontinuities in galaxy clusters (shocks and cold fronts), in particular to determine the density compression factor. Here we take the example of *XMM-Newton* observations of A2142 (z=0.09), which hosts one of the most famous cold fronts and the first to be recognized as such (Markevitch et al. 2000).

[1]:
```
import numpy as np
import pyproffit
import matplotlib.pyplot as plt
```

We use the publicly available *XMM-Newton* mosaic of A2142 extracted by the X-COP team and available here. We start by loading the image, exposure map and background map into a Data structure

[2]:
```
dat = pyproffit.Data(imglink='/home/deckert/Documents/Work/cluster_data/VLP/a2142/
↪mosaic_a2142.fits.gz',
                     explink='/home/deckert/Documents/Work/cluster_data/VLP/a2142/
↪mosaic_a2142_expo.fits.gz',
                     bkglink='/home/deckert/Documents/Work/cluster_data/VLP/a2142/
↪mosaic_a2142_bkg.fits.gz')
```

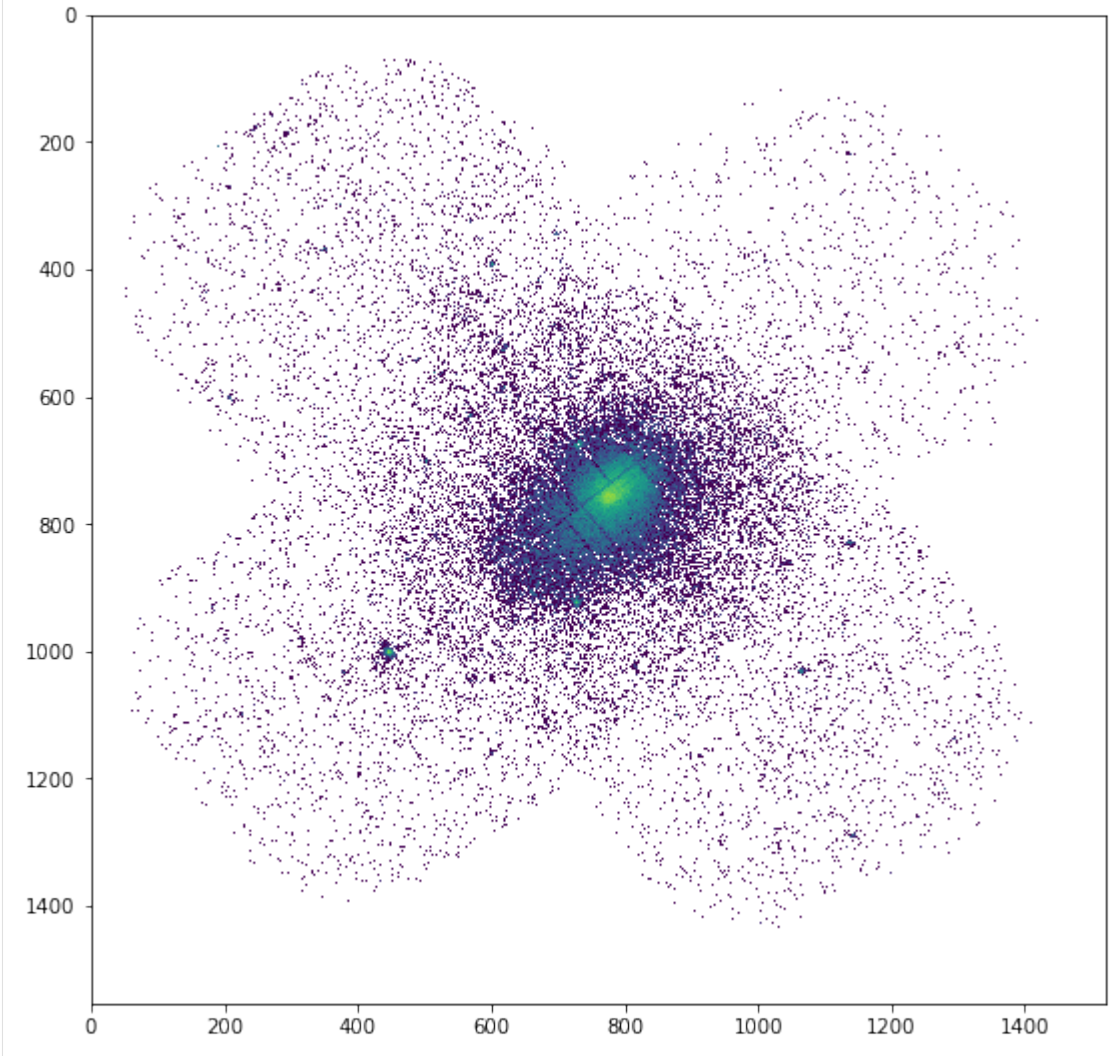
```
WARNING: FITSFixedWarning: RADECSYS= 'FK5 ' / Stellar reference frame
the RADECSYS keyword is deprecated, use RADESYSa. [astropy.wcs.wcs]
WARNING: FITSFixedWarning: EQUINOX = '2000.0 ' / Coordinate system equinox
a floating-point value was expected. [astropy.wcs.wcs]
```

Let's have a look at the data. Here it is a mosaics of several *XMM-Newton* pointings:

[3]:
```
fig = plt.figure(figsize=(20,20))
s1=plt.subplot(221)
plt.imshow(np.flipud(np.log10(dat.img)),aspect='auto')
```

```
<ipython-input-3-9eeab1757565>:3: RuntimeWarning: divide by zero encountered in log10
  plt.imshow(np.flipud(np.log10(dat.img)),aspect='auto')
```

[3]:
```
<matplotlib.image.AxesImage at 0x7fb1404f58b0>
```



We mask the detected point sources to avoid contaminating the profile

[4]:
```
dat.region('/home/deckert/Documents/Work/cluster_data/VLP/a2142/src_ps.reg')
```

```
Excluded 226 sources
```

The cold front is located about 3 arcmin North-West of the cluster core, i.e. in the top-right direction in the above plot. By inspecting the image with DS9 we need to get an idea of the geometry of the front, i.e. we need to define a sector across which the front will be sharpest. In this case the front is highly elliptical, with a position angle that is inclinated by ~40 degrees with respect to the Right Ascension axis.

We now define a Profile to which we pass the necessary information. We will extract a profile centered on R.A.= 239.5863, Dec=27.226989 with a linear binning of 5 arcsec width out to 10 arcmin

[5]:
```
prof = pyproffit.Profile(data=dat, binsize=5., maxrad=10.,
                         center_choice='custom_fk5', center_ra=239.5863, center_
↪dec=27.226989)
```

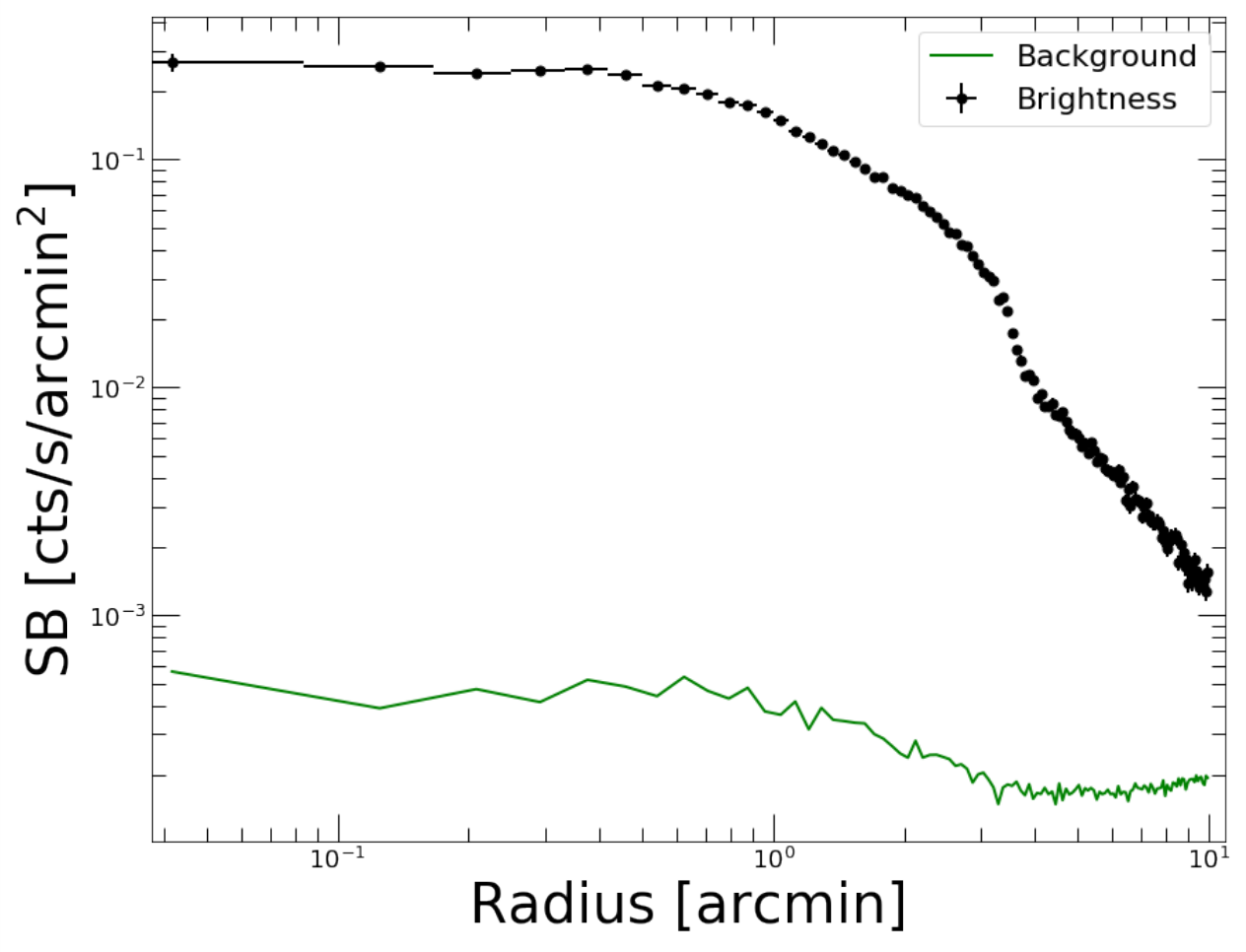
```
Corresponding pixels coordinates:  775.303810518434 791.9785944739778
```

Now we extract the profile using the SBprofile method of the Profile class. We select the data in a sector between position angles 10 and 70 degrees, across an ellipse rotated by 40 degrees and with a major-to-minor axis ratio of 1.65. Note that *PyProffit* follows the DS9 convention, i.e. the zero point refers to the Right Ascension axis.

[6]:
```
prof.SBprofile(rotation_angle=40., ellipse_ratio=1.65,
               angle_low=10., angle_high=70.)
```

[7]:
```
prof.Plot()
```

```
<Figure size 432x288 with 0 Axes>
```



3.3.1 Comparing sectors

The break in the profile between 3 and 4 arcmin is well visible. We can inspect it further by comparing the brightness across several sectors; this is done by defining other Profile objects and comparing them using the plot_multi_profiles function

[8]:
```
prof_se = pyproffit.Profile(data=dat, binsize=5., maxrad=10.,
                            center_choice='custom_fk5', center_ra=239.5863, center_
↪dec=27.226989)

prof_ne = pyproffit.Profile(data=dat, binsize=5., maxrad=10.,
                            center_choice='custom_fk5', center_ra=239.5863, center_
↪dec=27.226989)

prof_sw = pyproffit.Profile(data=dat, binsize=5., maxrad=10.,
                            center_choice='custom_fk5', center_ra=239.5863, center_
↪dec=27.226989)
```

```
Corresponding pixels coordinates:  775.303810518434 791.9785944739778
Corresponding pixels coordinates:  775.303810518434 791.9785944739778
Corresponding pixels coordinates:  775.303810518434 791.9785944739778
```

In the new Profile structures we now load brightness profiles in sectors of 60 degree opening along 4 perpendicular

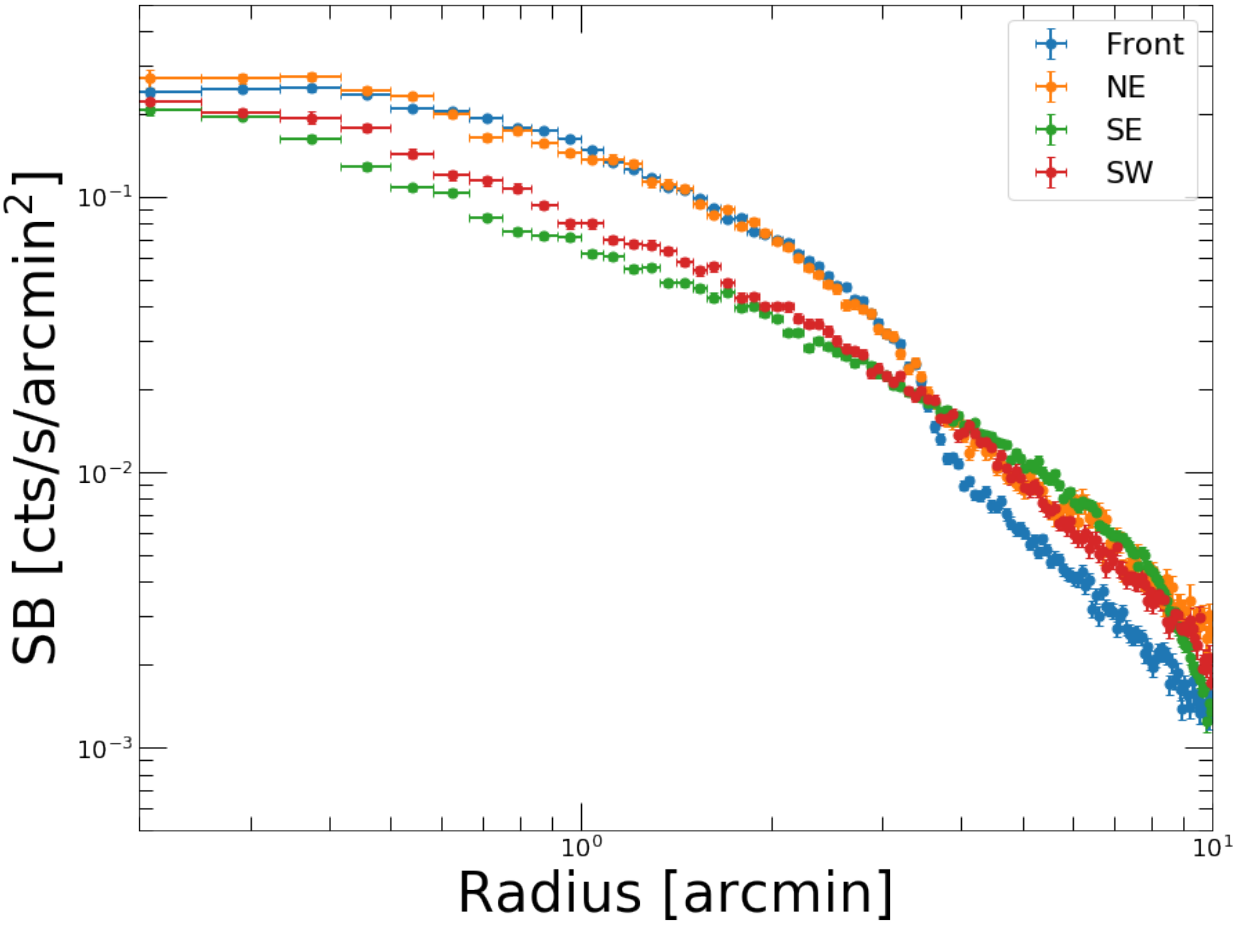
directions

[9]:
```
prof_se.SBprofile(rotation_angle=40., ellipse_ratio=1.65,
                  angle_low=190., angle_high=250.)

prof_ne.SBprofile(rotation_angle=40., ellipse_ratio=1.65,
                  angle_low=100., angle_high=160.)

prof_sw.SBprofile(rotation_angle=40., ellipse_ratio=1.65,
                    angle_low=280., angle_high=340.)
```

We can now display all 4 profiles together using the plot_multi_profiles function

[10]:
```
fig = pyproffit.plot_multi_profiles(profs=(prof, prof_ne, prof_se, prof_sw),
                                    labels=('Front', 'NE', 'SE', 'SW'),
                                    axes=[0.2, 10., 5e-4, 0.5])
```

```
Showing 4 brightness profiles
```



We can see clearly the difference between the various sectors. The sectors on the South show no discontinuity around 3-4 arcmin. The front can be observed as well in the NE direction, although it is not as sharp as in the direction that we previously identified for the front.

To search for deviations from symmetry we can also look at the *azimuthal scatter*, i.e. the quantity

$$\Sigma_X(r) = \left(\sum_{i=1}^{N} \frac{(S_i(r) - \langle S(r) \rangle)^2}{\langle S(r) \rangle^2} \right)^{1/2}$$

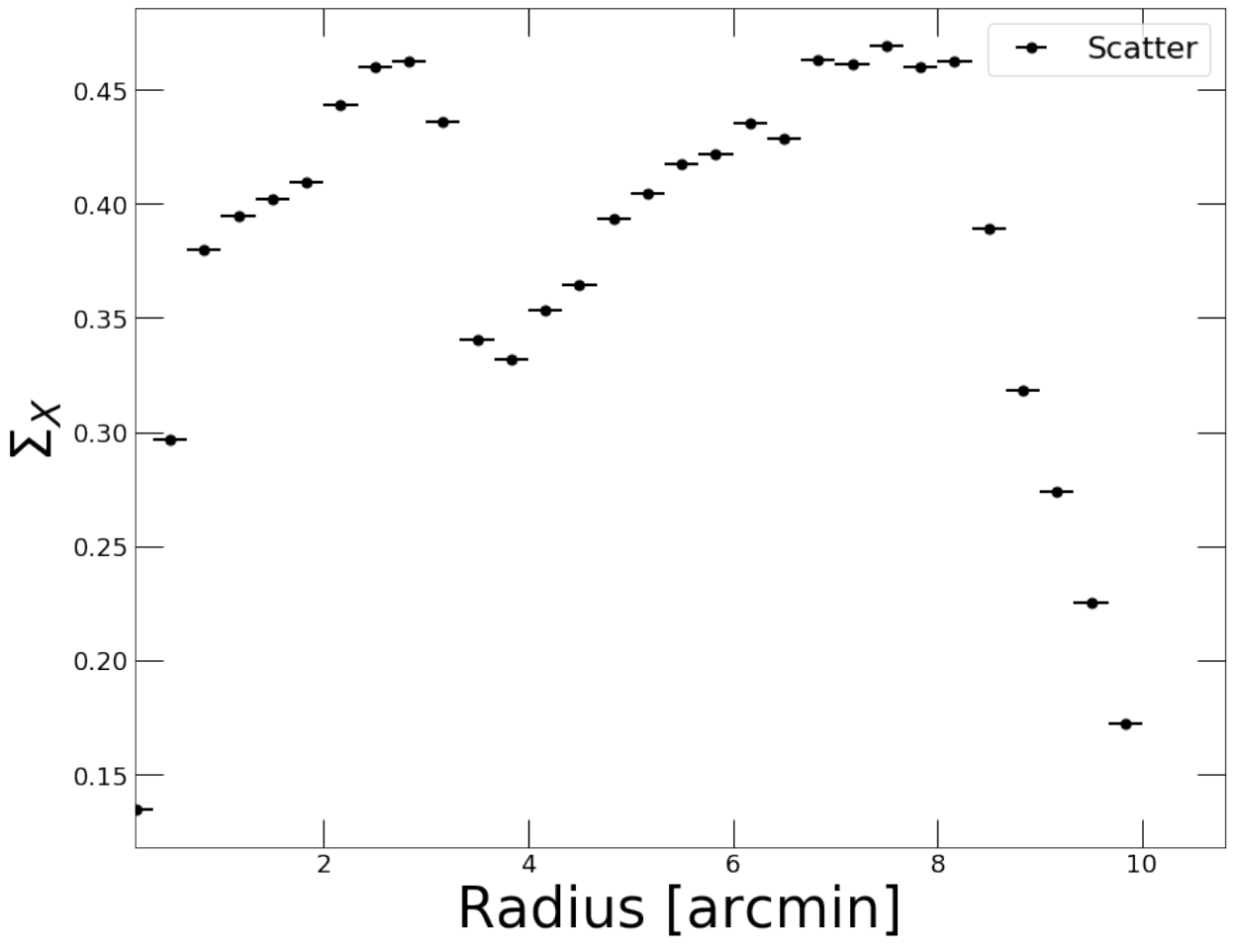
with $S_i(r)$ the surface brightness profile determined in N individual sectors covering the azimuth from 0 to 360 degrees with an opening angle of $360/N$ degrees, and $\langle S(r) \rangle$ a loaded mean surface brightness profile

[11]:
```
prof_tot = pyproffit.Profile(data=dat, binsize=20., maxrad=10.,
                             center_choice='custom_fk5', center_ra=239.5863, center_
↪dec=27.226989)

prof_tot.SBprofile()

prof_tot.AzimuthalScatter(nsect=12)
```

```
Corresponding pixels coordinates:  775.303810518434 791.9785944739778
```

[12]:
```
prof_tot.Plot(scatter=True, yscale='linear', xscale='linear')
```

```
<Figure size 432x288 with 0 Axes>
```



The azimuthal scatter shows two regions of enhanced scatter followed by two sharp drops. The enhanced scatter is induced by sloshing gas extending out to the cold fronts; beyond the cold fronts the scatter from one sector to the other decreases sharply. This method can be useful to pinpoint the radii of the cold fronts.

3.3.2 Modeling the brightness profile

Now that we are confident that we have identified the feature of interest, let's try to model it. First, we need to account for the *XMM-Newton* PSF, which smears the gradient across the front and would lead to an underestimation of the compression factor. To this aim, we create a function describing the *XMM-Newton* PSF as a function of distance, and we use the PSF method to generate a PSF mixing matrix. We describe the *XMM-Newton* PSF as a King function with parameters provided in the calibration files

[13]:
```
# Function describing the PSF
def fking(x):
    r0=0.0883981 # core radius in arcmin
    alpha=1.58918 # outer slope
    return np.power(1.+(x/r0)**2,-alpha)

prof.PSF(psffunc=fking)
```

As is usually done in these cases, we assume that the 3D distribution is described as two power laws with an infinitely small discontinuity. The 3D broken power law is then projected onto the line of sight:

$$I(r) = I_0 \int F(\omega)^2 \, d\ell + B$$

with $\omega^2 = r^2 + \ell^2$ and

$$F(\omega) = \begin{cases} \omega^{-\alpha_1}, & \omega < r_f \\ \frac{1}{C}\omega^{-\alpha_2}, & \omega \geq r_f \end{cases}$$

PyProffit includes the BknPow function which implements this model. We now define a Model object containing the appropriate model to describe the front

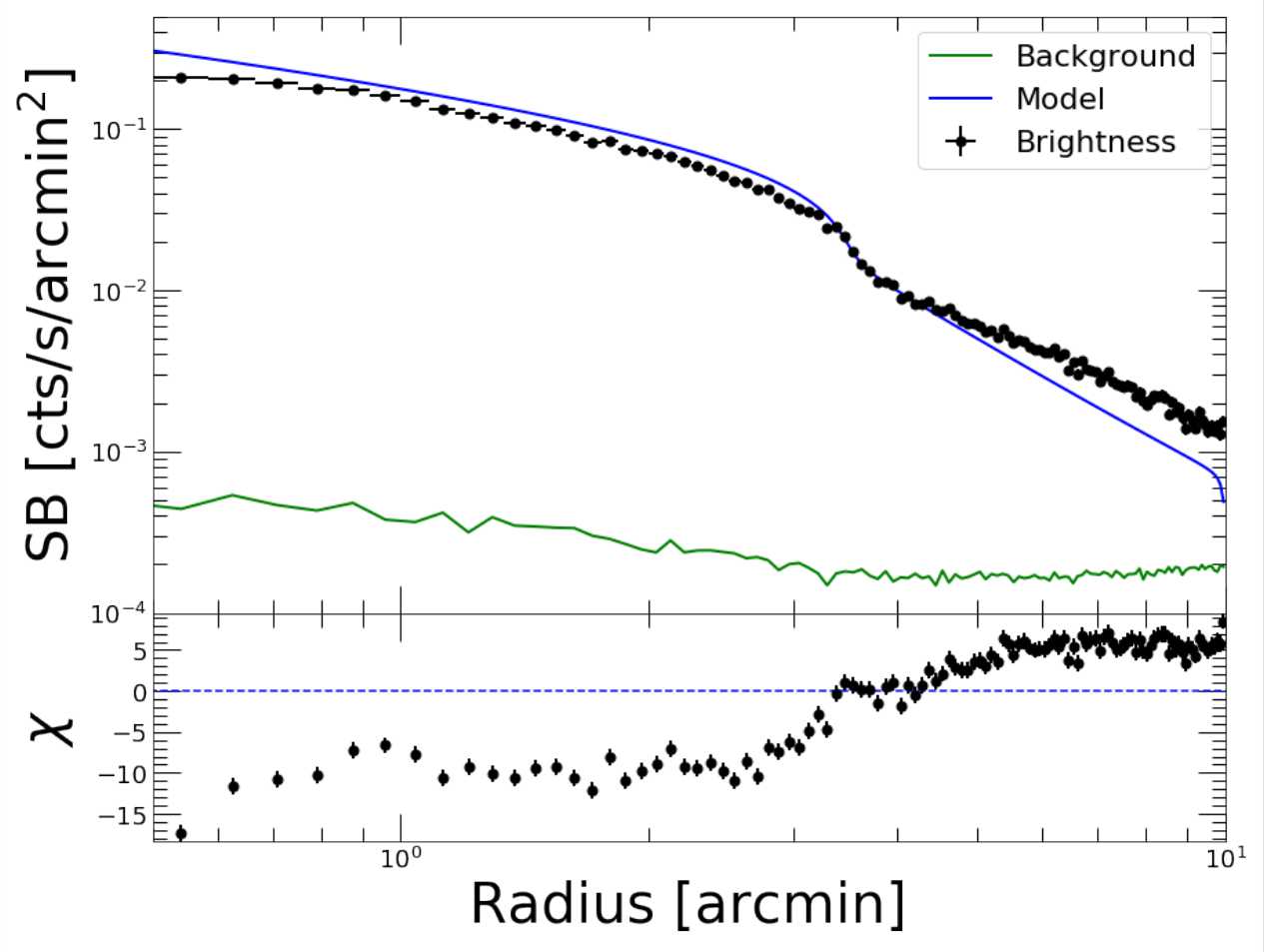
[14]:
```
modbkn = pyproffit.Model(pyproffit.BknPow)

print(modbkn.parnames)
```

```
('alpha1', 'alpha2', 'rf', 'norm', 'jump', 'bkg')
```

To choose appropriate starting points for the parameter, we can set up initial values and inspect how the model compares to the data

[15]:
```
modbkn.SetParameters([0.8, 2., 3.5, -1.8, 1.8, -4.])

prof.Plot(model=modbkn, axes=[0.5, 10., 1e-4, 0.5])
```

```
<Figure size 432x288 with 0 Axes>
```



We are now ready to optimize the model. To do this, we set up a Fitter object and pass to it the data and the model. We run the optimization using the Migrad method of the Fitter class.

To focus on the region surrounding the front, we fit the data between 1 and 7 arcmin such that we still have a good handle of the slopes in the upstream and downstream regions, whilst being insensitive to the behavior of the profile far away from the front. The fitting range is specified using the *fitlow* and *fithigh* parameters of the Migrad method

[16]:
```
fitobj = pyproffit.Fitter(model=modbkn, profile=prof, alpha1=0.8, alpha2=2.0, rf=3.5,␣
↪jump=1.8, norm=-1.8, bkg=-4.0,
              fitlow=1.0, fithigh=7.0)

fitobj.Migrad()
```

```
┌──────────────────────────────────┬──────────────────────────────────────┐
│ FCN = 71.63                      │              Nfcn = 962              │
│ EDM = 7.84e-05 (Goal: 0.0002)    │                                      │
├──────────────┬───────────────────┼──────────────────────────────────────┤
│ Valid Minimum │ Valid Parameters │        No Parameters at limit        │
├──────────────┴───────────────────┼──────────────────────────────────────┤
│ Below EDM threshold (goal x 10)  │           Below call limit           │
├──────────────┬───────────────┬───┴──────────┬─────────────┬─────────────┤
│  Covariance  │     Hesse ok  │ Accurate     │  Pos. def.  │ Not forced  │
└──────────────┴───────────────┴──────────────┴─────────────┴─────────────┘

┌───┬──────┬───────────┬───────────┬────────────┬────────────┬─────────────┬─────────────┬␣
↪─────────┐
│   │ Name │   Value   │ Hesse Err │ Minos Err- │ Minos Err+ │   Limit-    │   Limit+    │␣
↪ Fixed │
```

(continues on next page)

(continued from previous page)

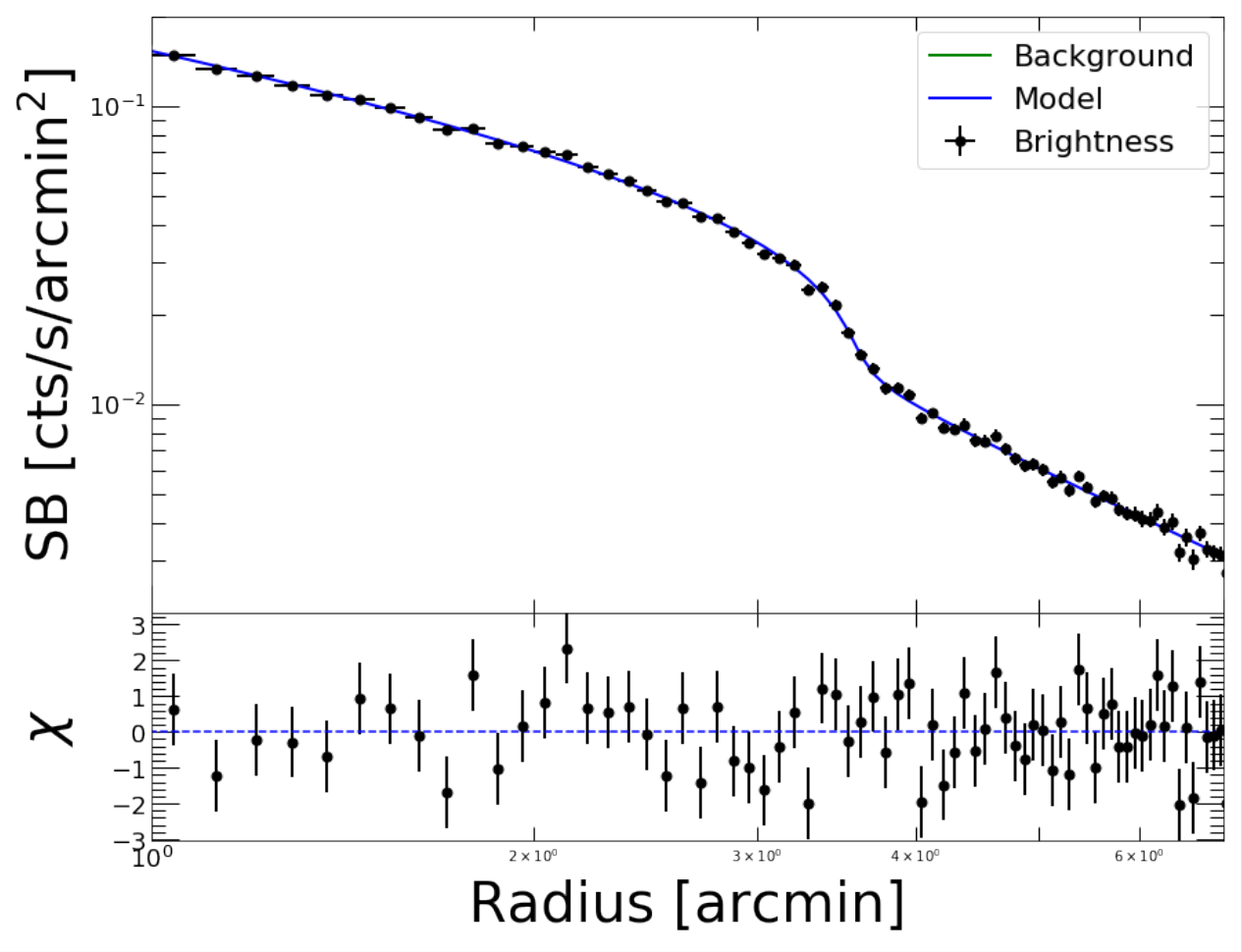
```
╒═══╤═══════╤═══════╤═══════╤═══════╤═══════╤═══════╤═══════╕
│ 0 │ alpha1 │   0.869   │   0.013   │           │           │         │         │
↪      │
│ 1 │ alpha2 │   1.51    │   0.14    │           │           │         │         │
↪      │
│ 2 │ rf     │   3.612   │   0.026   │           │           │         │         │
↪      │
│ 3 │ norm   │  -1.948   │   0.011   │           │           │         │         │
↪      │
│ 4 │ jump   │   1.90    │   0.06    │           │           │         │         │
↪      │
│ 5 │ bkg    │   -4.0    │    3.5    │           │           │         │         │
↪      │
╘═══╧═══════╧═══════╧═══════╧═══════╧═══════╧═══════╧═══════╛

╒═══════╤═══════════╤═══════════╤═══════════╤═══════════╤═══════════╤═══════════╕
│       │    alpha1 │    alpha2 │        rf │      norm │      jump │       bkg │
╞═══════╪═══════════╪═══════════╪═══════════╪═══════════╪═══════════╪═══════════╡
│ alpha1 │   0.00016 │ 4.76e-05  │   0.00017 │ -0.000122 │ -0.000125 │   0.00161 │
│ alpha2 │ 4.76e-05  │    0.0199 │  -0.00136 │  0.000196 │  -0.00661 │     0.484 │
│    rf │   0.00017 │  -0.00136 │  0.000653 │ -0.000252 │  0.000585 │   -0.0288 │
│  norm │ -0.000122 │  0.000196 │ -0.000252 │   0.00013 │ -3.45e-05 │   0.00349 │
│  jump │ -0.000125 │  -0.00661 │  0.000585 │ -3.45e-05 │   0.00306 │    -0.144 │
│   bkg │   0.00161 │     0.484 │   -0.0288 │   0.00349 │    -0.144 │      12.1 │
╘═══════╧═══════════╧═══════════╧═══════════╧═══════════╧═══════════╧═══════════╛
Best fit chi-squared: 71.6327 for 120 bins and 66 d.o.f
Reduced chi-squared: 1.08534
```

The *Valid Minimum* output indicates that the minimization was performed successfully. The best-fit parameters are now officially loaded into the Model object. The retrieved compression factor (the *jump* parameter) of 1.90 ± 0.06 agrees well with the value measured by *Chandra* for this front, 2.0 ± 0.1 (Owers et al. 2009).

The low reduced chi-squared value implies the model provides a good description of the data. Now let us check the quality of the fit

[17]:
```
prof.Plot(model=modbkn, axes=[1., 7., 2e-3, 0.2])
```

```
<Figure size 432x288 with 0 Axes>
```



That looks very good. By default, the code will run a chi-square minimization; if instead we wish to minimize the C statistic, we can run the minimization again using the *method='cstat'* option.

We can also fix the *bkg* parameter since it is not very relevant in this region and its value is not well constrained

[18]:
```
fitobj = pyproffit.Fitter(model=modbkn, profile=prof, method='cstat',
                          alpha1=0.9, alpha2=1.5, rf=3.609, jump=1.92,
                          norm=-1.9, bkg=-3.8,
                          fitlow=1.0, fithigh=7.0)

fitobj.minuit.fixed['bkg'] = True

fitobj.Migrad()
```

FCN = 81.15 EDM = 6.08e-06 (Goal: 0.0002)		Nfcn = 192		
Valid Minimum	Valid Parameters	No Parameters at limit		
Below EDM threshold (goal x 10)		Below call limit		
Covariance	Hesse ok	Accurate	Pos. def.	Not forced

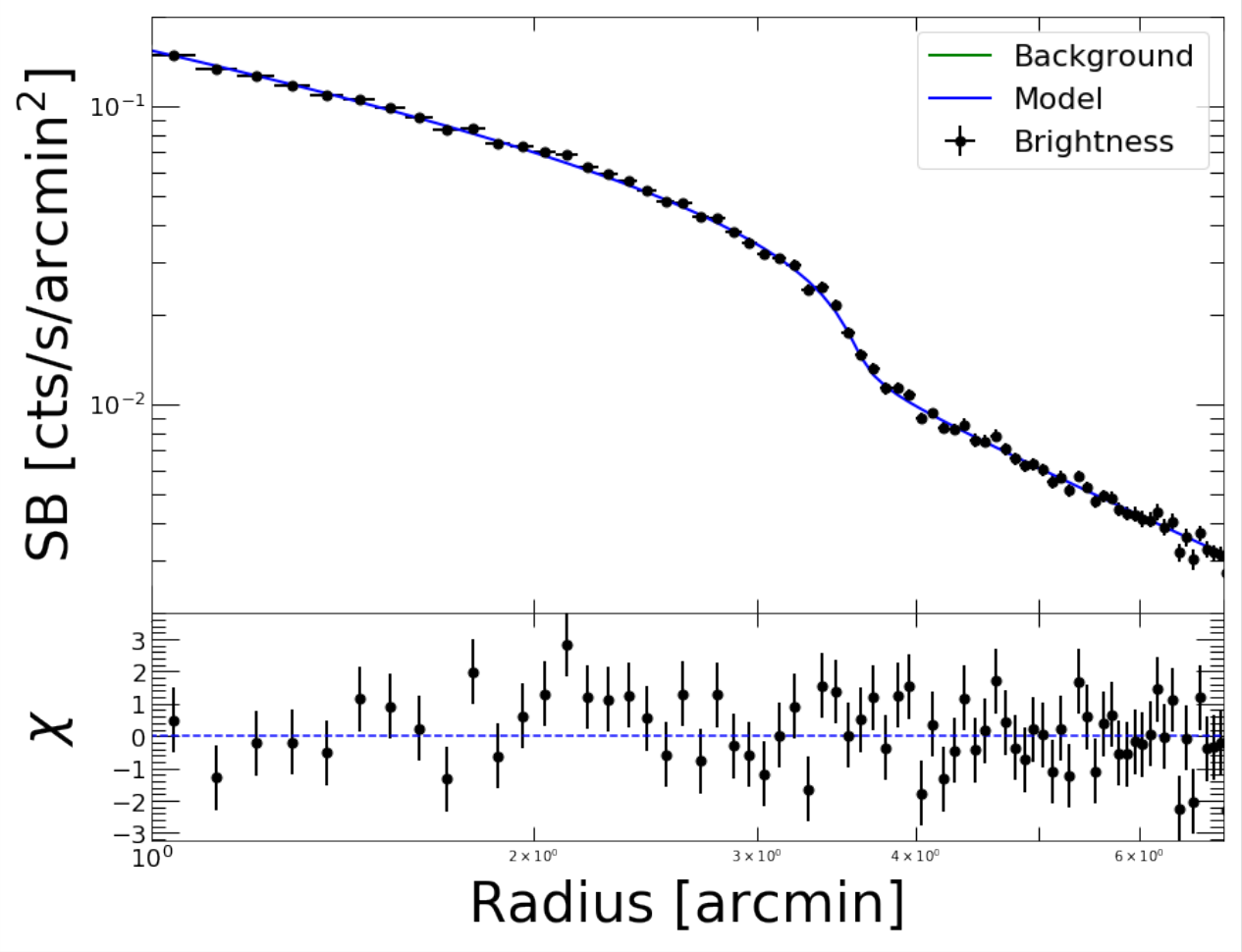
(continues on next page)

(continued from previous page)

```
└──────────────────────────────────────────────────────────────────────────────┘
┌──────────────────────────────────────────────────────────────────────────────┐
│   │ Name    │   Value    │ Hesse Err │ Minos Err- │ Minos Err+ │ Limit-   │ Limit+   │
↪Fixed │
├──────────────────────────────────────────────────────────────────────────────┤
│ 0 │ alpha1 │   0.884    │   0.012   │            │            │          │          │
↪      │
│ 1 │ alpha2 │   1.503    │   0.025   │            │            │          │          │
↪      │
│ 2 │ rf     │   3.615    │   0.023   │            │            │          │          │
↪      │
│ 3 │ norm   │  -1.959    │   0.011   │            │            │          │          │
↪      │
│ 4 │ jump   │   1.90     │   0.04    │            │            │          │          │
↪      │
│ 5 │ bkg    │   -3.80    │   -0.04   │            │            │          │          │
↪yes   │
└──────────────────────────────────────────────────────────────────────────────┘
┌──────────────────────────────────────────────────────────────────────┐
│        │    alpha1    alpha2        rf      norm      jump       bkg │
├──────────────────────────────────────────────────────────────────────┤
│ alpha1 │  0.000151 -1.66e-05  0.000162 -0.000115 -9.97e-05         0 │
│ alpha2 │ -1.66e-05  0.000627 -0.000205  5.61e-05 -0.000808         0 │
│    rf  │  0.000162 -0.000205  0.000541 -0.000228  0.000226         0 │
│  norm  │ -0.000115  5.61e-05 -0.000228  0.000121  6.07e-06         0 │
│  jump  │ -9.97e-05 -0.000808  0.000226  6.07e-06   0.00126         0 │
│   bkg  │         0          0         0         0         0         0 │
└──────────────────────────────────────────────────────────────────────┘
Best fit C-statistic: 81.1496 for 120 bins and 67 d.o.f
Reduced C-statistic: 1.21119
```

[19]:
```
prof.Plot(model=modbkn, axes=[1., 7., 2e-3, 0.2])
```

```
<Figure size 432x288 with 0 Axes>
```



The results obtained with the two likelihood functions are nicely consistent. In case of low quality data, however, the results obtained with C-statistic should be preferred.

3.3.3 Results and uncertainties

The results of the fitting procedure are stored in the *params* and *errors* attributes of the Fitter object, which can be displayed using the *out* attribute

[20]:
```
fitobj.out
```

[20]:
```
┌──────────────────────────────────┬──────────────────────────────────────┐
│ FCN = 81.15                      │              Nfcn = 192              │
│ EDM = 6.08e-06 (Goal: 0.0002)    │                                      │
├──────────────────────────────────┼──────────────────────────────────────┤
│ Valid Minimum │ Valid Parameters │        No Parameters at limit        │
├──────────────────────────────────┴──────────────────────────────────────┤
│ Below EDM threshold (goal x 10)  │           Below call limit           │
├──────────────┬───────────────────┬──────────┬──────────┬────────────────┤
│  Covariance  │     Hesse ok      │ Accurate │ Pos. def.│ Not forced     │
└──────────────┴───────────────────┴──────────┴──────────┴────────────────┘

┌───┬──────┬───────────┬───────────┬────────────┬────────────┬─────────┬─────────┬─
│   │ Name │   Value   │ Hesse Err │ Minos Err- │ Minos Err+ │ Limit-  │ Limit+  │ 
↪Fixed │
```

(continues on next page)

(continued from previous page)

```
┌──────────────────────────────────────────────────────────────────────────────────────┐
│  0 │ alpha1 │   0.884   │   0.012   │           │           │         │         │ ␣
↪     │
│  1 │ alpha2 │   1.503   │   0.025   │           │           │         │         │ ␣
↪     │
│  2 │ rf     │   3.615   │   0.023   │           │           │         │         │ ␣
↪     │
│  3 │ norm   │  -1.959   │   0.011   │           │           │         │         │ ␣
↪     │
│  4 │ jump   │   1.90    │   0.04    │           │           │         │         │ ␣
↪     │
│  5 │ bkg    │   -3.80   │   -0.04   │           │           │         │         │ ␣
↪yes  │
└──────────────────────────────────────────────────────────────────────────────────────┘
```

	alpha1	alpha2	rf	norm	jump	bkg
alpha1	0.000151	-1.66e-05	0.000162	-0.000115	-9.97e-05	0
alpha2	-1.66e-05	0.000627	-0.000205	5.61e-05	-0.000808	0
rf	0.000162	-0.000205	0.000541	-0.000228	0.000226	0
norm	-0.000115	5.61e-05	-0.000228	0.000121	6.07e-06	0
jump	-9.97e-05	-0.000808	0.000226	6.07e-06	0.00126	0
bkg	0	0	0	0	0	0

[21]:
```
fitobj.params['jump']
```

[21]:
```
1.901961720701023
```

The *Migrad* function of iminuit is a very efficient optimization algorithm, however it is not designed to determine accurate, asymmetric error bars. For this purpose, iminuit includes the *Minos* algorithm, which can be ran easily from *PyProffit*

[22]:
```
minos_result = fitobj.minuit.minos()
```

The uncertainties in the *jump* parameter can be viewed and accessed in the following way

[23]:
```
minos_result.params
```

[23]:
```
┌──────────────────────────────────────────────────────────────────────────────────────┐
│    │ Name   │   Value   │ Hesse Err │ Minos Err- │ Minos Err+ │ Limit-  │ Limit+  │␣
↪Fixed │
├──────────────────────────────────────────────────────────────────────────────────────┤
│  0 │ alpha1 │   0.884   │   0.012   │   -0.012   │   0.012    │         │         │ ␣
↪     │
│  1 │ alpha2 │   1.503   │   0.025   │   -0.025   │   0.025    │         │         │ ␣
↪     │
│  2 │ rf     │   3.615   │   0.023   │   -0.022   │   0.014    │         │         │ ␣
↪     │
│  3 │ norm   │  -1.959   │   0.011   │   -0.011   │   0.010    │         │         │ ␣
↪     │
│  4 │ jump   │   1.902   │   0.036   │   -0.035   │   0.036    │         │         │ ␣
↪     │
│  5 │ bkg    │   -3.80   │   -0.04   │            │            │         │         │ ␣
↪yes  │
└──────────────────────────────────────────────────────────────────────────────────────┘
```

[24]:
```
minos_result.merrors['jump']
```

[24]:

	jump	
Error	-0.035	0.036
Valid	True	True
At Limit	False	False
Max FCN	False	False
New Min	False	False

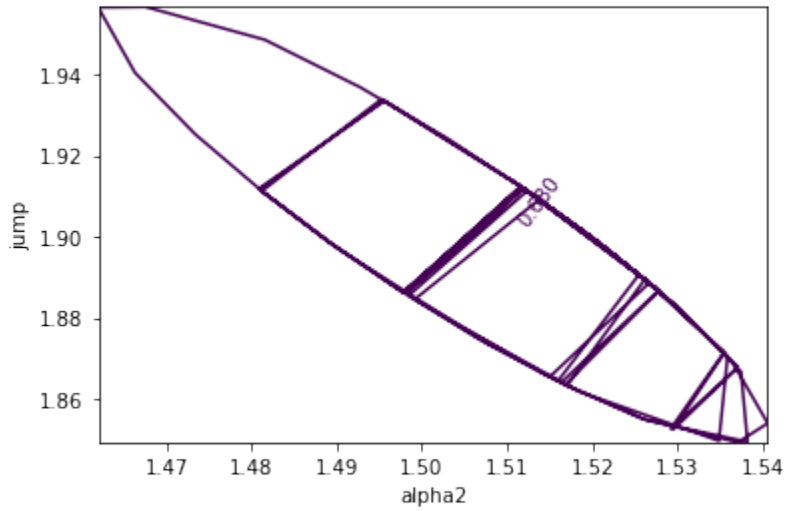
[25]:
```
print('Best fitting compression factor : %g (%g , %g)'
      % (fitobj.params['jump'], minos_result.merrors['jump'].lower, minos_result.
↪merrors['jump'].upper))
```

```
Best fitting compression factor : 1.90196 (-0.0351364 , 0.0360152)
```

Correlations between parameters can be investigated using the draw_mncontour method. Here we show the usual correlation between the outer slope of the profile α_2 and the compression factor

[26]:
```
fitobj.minuit.draw_mncontour('alpha2', 'jump')
```

[26]:
```
<matplotlib.contour.ContourSet at 0x7fb135758b50>
```



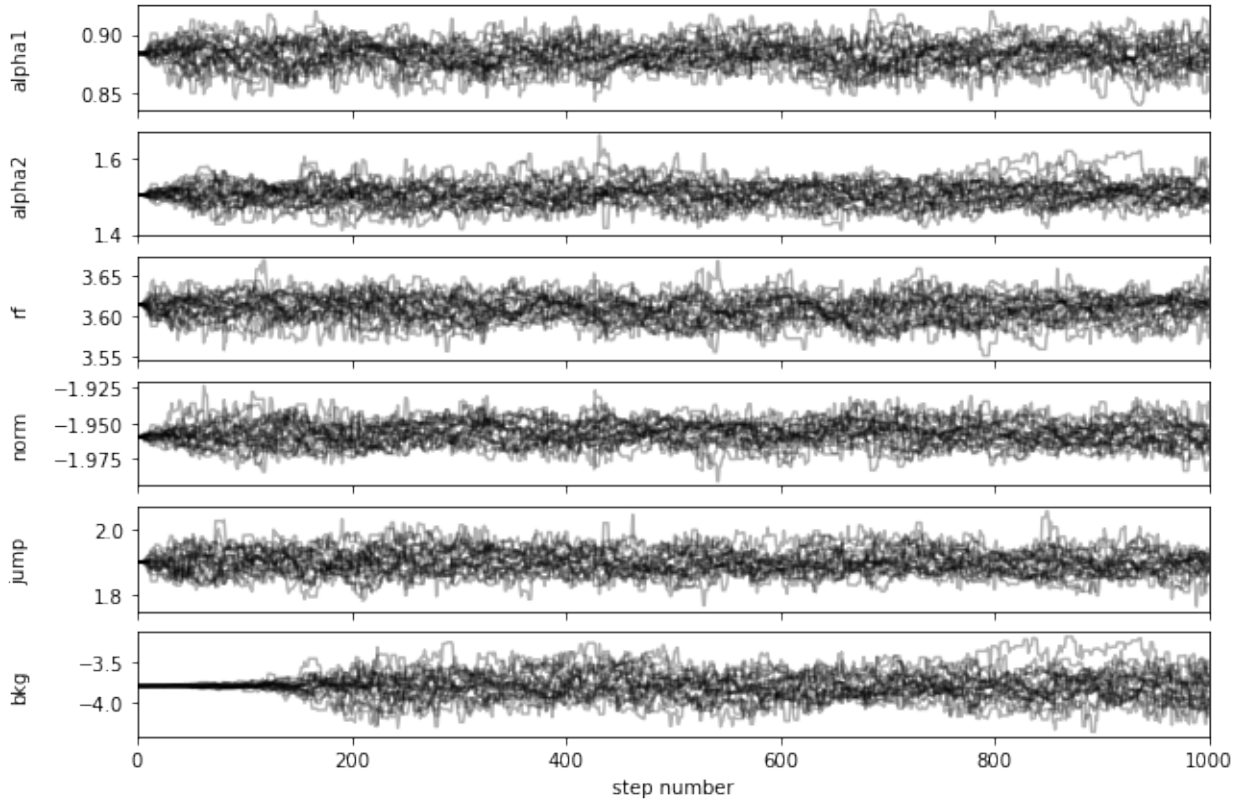
3.3.4 Running Monte Carlo Markov Chain

To check the results and inspect the correlations further, we can now run a Monte Carlo Markov Chain (MCMC) using the affine invariant sampler emcee assuming the corresponding package is installed. This can be done easily using the Emcee method of the Fitter class.

If a pre-existing maximum-likelihood fit can be found, as in the previous case, the code will start from the best-fit values and automatically assign a broad Gaussian prior on each of the parameters. Alternatively, any custom prior function computing the prior probability of an input parameter set can be passed to the code via the *prior=function* option.

[27]:
```
fitobj.Emcee(nmcmc=1000, walkers=20, burnin=200)
```

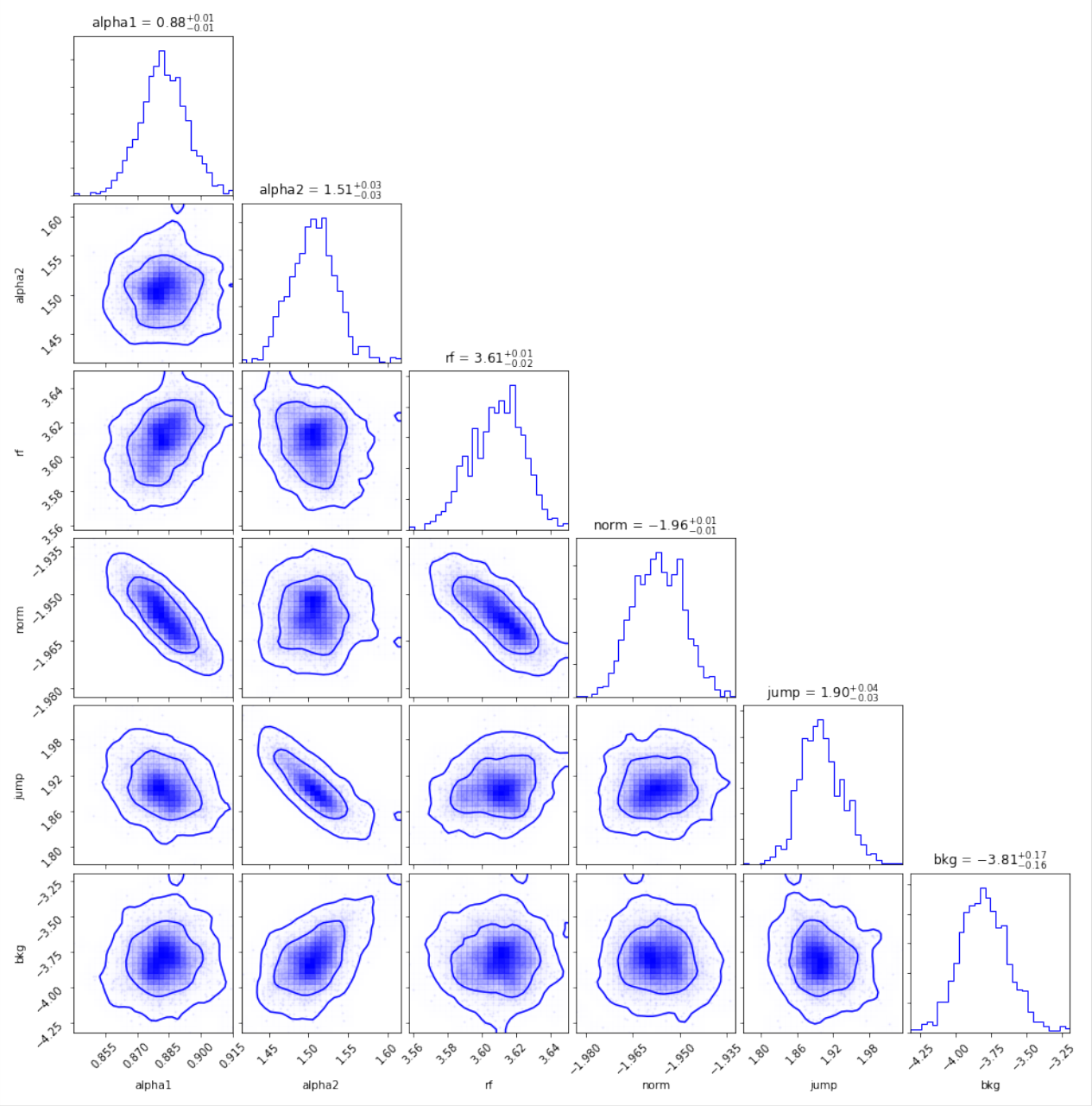
```
100%|| 1000/1000 [01:23<00:00, 11.94it/s]
```



We can see that all the chains are well converged after step 200.

We can inspect the output chain and create a two-dimensional corner plot using the Corner method if the corner library is installed. All the available arguments of the corner library can be passed directly to the code as shown in the example below

[28]:
```
fig_corner = fitobj.Corner(show_titles=True, levels=(0.68, 0.95), bins=(30), no_fill_
↪contours=True, color='blue', smooth=1.2)
```



Again, we can see the clear anti-correlation between the density jump and the outer slope. The uncertainties in the jump parameter are consistent with the *Minos* results shown above. The chain can be accessed via the *samples* attribute of the Fitter class, e.g. to check the individual posterior probability distributions

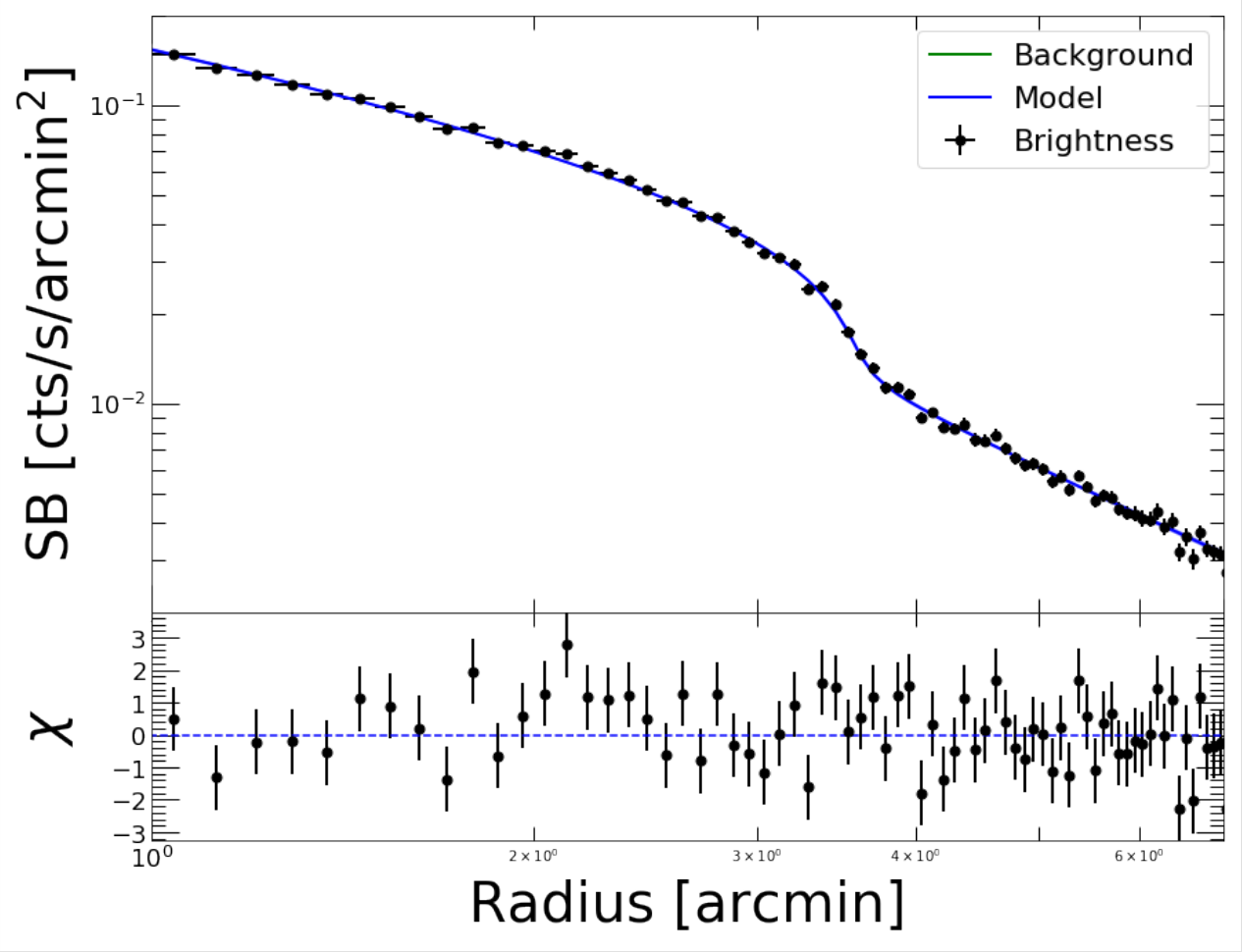
[29]:
```
chain_jump = fitobj.samples[:,4]

median_jump, jump_lo, jump_hi = np.percentile(chain_jump, [50., 50.-68.3/2., 50.+68.3/
↪2.])

print('Best fitting compression factor : %g (%g , %g)'
      % (median_jump, median_jump-jump_lo, jump_hi-median_jump))
```

```
Best fitting compression factor : 1.8983 (0.0327633 , 0.0401024)
```

We can then look at the fitted model envelope and compare it with the data by passing the output Emcee samples to the Plot method of the Profile class,

[30]:
```
prof.Plot(model=modbkn, samples=fitobj.samples, axes=[1., 7., 2e-3, 0.2])
```

```
<Figure size 432x288 with 0 Axes>
```



CHAPTER 4

pyproffit package

4.1 Submodules

4.2 pyproffit.data module

class pyproffit.data.**Data**(*imglink*, *explink=None*, *bkglink=None*, *voronoi=False*, *rmsmap=None*)
Bases: object

Class containing the data to be loaded and used by other pyproffit routines

Parameters

- **imglink** (*str*) – Path to input image
- **explink** (*str , optional*) – Path to exposure map. If none, assume a flat exposure of 1s or an input error map provided through rmsmap
- **bkglink** (*str , optional*) – Path to background map. If none, assume zero background
- **voronoi** (*bool , optional*) – Define whether the input image is a Voronoi image or not (default=False)
- **rmsmap** (*str , optional*) – Path to error map if the data is not Poisson distributed

dmfilth(*outfile=None*, *smoothing_scale=8*)
Mask the regions provided in a region file and fill in the holes by interpolating the smoothed image into the gaps and generating a Poisson realization

Parameters

- **outfile** (*str , optional*) – If outfile is not None, file name to output the dmfilth image into a FITS file
- **smoothing_scale** (*int*) – Size of smoothing scale (in pixel) to estimate the surface brightness distribution outside of the masked areas

region (*regfile*)
Filter out regions provided in an input DS9 region file

Parameters **regfile** (*str*) – Path to region file. Accepted region file formats are fk5 and image.

reset_exposure ()
Revert to the original exposure map and ignore the current region file

pyproffit.data.**get_extnum** (*fitsfile*)
Find the extension number of the first IMAGE extension in an input FITS file

Parameters **fitsfile** (*str*) – Input FITS file to be read

Returns extension number

Return type int

4.3 pyproffit.deproject module

class pyproffit.deproject.**Deproject** (*z=None*, *profile=None*, *cf=None*, *f_abund='aspl'*)
Bases: object

Class to perform all calculations of deprojection, density profile, gas mass, count rate, and luminosity

Parameters

- **z** (*float*) – Source redshift. If z=None, only the surface brightness reconstruction can be done.
- **profile** (class:*pyproffit.profextract.Profile*) – Object of type *pyproffit.profextract.Profile* containing the surface brightness profile data
- **cf** (*float*) – Conversion factor from count rate to emissivity. If cf=None, only the surface brightness reconstruction can be done.
- **f_abund** (*str*) – Solar abundance table to compute the electron-to-proton ratio and mean molecular weight. Available tables are 'aspl' (Aspling+09, default), 'angr' (Anders & Grevesse 89), and 'grsa' (Grevesse & Sauval 98)

CSB (*rin=40.0*, *rout=400.0*, *plot=True*, *outfile=None*, *figsize=(13, 10)*, *nbins=30*, *fontsize=40*, *yscale='linear'*, ***kwargs*)
Compute the surface brightness concentration from a loaded brightness profile reconstruction. The surface brightness concentration is defined as the ratio of fluxes computed within two apertures.

Parameters

- **rin** (*float*) – Lower aperture value in kpc (default=40)
- **rout** (*float*) – Higher aperture value in kpc (default=400)
- **plot** (*bool*) – Plot the posterior CSB distribution (default=True)
- **outfile** (*str*) – Output file name to save the figure. If outfile=None, plot only to stdout
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **nbins** (*int , optional*) – Number of bins on the X axis to construct the posterior distribution. Defaults to 30
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **yscale** (*str , optional*) – Scale on the Y axis. Defaults to 'linear'

- **kwargs** – Options to be passed to `matplotlib.pyplot.hist`

Returns Median count rate, 16th and 84th percentiles

Return type float

CountRate(*a*, *b*, *plot=True*, *outfile=None*, *figsize=(13, 10)*, *nbins=30*, *fontsize=40*, *yscale='linear'*, ***kwargs*)

Compute the model count rate integrated between radii a and b. Optionally, the count rate distribution can be plotted and saved.

Parameters

- **a** (*float*) – Inner integration boundary in arcmin
- **b** (*float*) – Outer integration boundary in arcmin
- **plot** (*bool*) – Plot the posterior count rate distribution (default=True)
- **outfile** (*str*) – Output file name to save the figure. If outfile=None, plot only to stdout
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **nbins** (*int , optional*) – Number of bins on the X axis to construct the posterior distribution. Defaults to 30
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **yscale** (*str , optional*) – Scale on the Y axis. Defaults to 'linear'
- **kwargs** – Options to be passed to `matplotlib.pyplot.hist`

Returns Median count rate, 16th and 84th percentiles

Return type float

Density(*rout=None*)

Compute a density profile from a multiscale reconstruction

Parameters **rout** (*numpy.ndarray*) – Radial binning of the density profile. If rout=None, the original binning of the surface brightness profile is used

Luminosity(*a*, *b*, *plot=True*, *outfile=None*, *figsize=(13, 10)*, *nbins=30*, *fontsize=40*, *yscale='linear'*, ***kwargs*)

Compute the luminosity integrated between radii a and b. Optionally, the luminosity distribution can be plotted and saved. Requires the luminosity factor to be computed using the *pyproffit.profextract.Profile.Emissivity()* method.

Parameters

- **a** (*float*) – Inner integration boundary in arcmin
- **b** (*float*) – Outer integration boundary in arcmin
- **plot** (*bool*) – Plot the posterior count rate distribution (default=True)
- **outfile** (*str*) – Output file name to save the figure. If outfile=None, plot only to stdout
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **nbins** (*int , optional*) – Number of bins on the X axis to construct the posterior distribution. Defaults to 30
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **yscale** (*str , optional*) – Scale on the Y axis. Defaults to 'linear'
- **kwargs** – Options to be passed to `matplotlib.pyplot.hist`

Returns Median luminosity, 16th and 84th percentiles

Return type float

Mgas(*radius*, *radius_err=None*, *plot=True*, *outfile=None*, *rad_scale='normal'*, *figsize=(13, 10)*, *nbins=30*, *fontsize=40*, *yscale='linear'*, ***kwargs*)

Compute the posterior cumulative gas mass within a given radius. Optionally, the posterior distribution can be plotted and saved.

Parameters

- **radius** (*float*) – Gas mass integration radius in kpc
- **radius_err** (*float , optional*) – (Gaussian) error on the input radius to be propagated to the gas mass measurement. To be used in case one wants to evaluate M_{gas} at an overdensity radius with a known uncertainty
- **plot** (*bool*) – Plot the posterior Mgas distribution (default=True)
- **outfile** (*str , optional*) – Output file name to save the figure. If outfile=None, plot only to stdout
- **rad_scale** (*str*) – If radius_err is not None, specify whether the distribution of radii is drawn from a normal distribution (rad_scale='normal') or a log-normal distribution (rad_scale='lognormal'). Defaults to 'normal'.
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **nbins** (*int , optional*) – Number of bins on the X axis to construct the posterior distribution. Defaults to 30
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **yscale** (*str , optional*) – Scale on the Y axis. Defaults to 'linear'
- **kwargs** – Options to be passed to `matplotplib.pyplot.hist`

Returns Median M_{gas}, 16th and 84th percentiles

Return type float

Multiscale(*backend='pymc3'*, *nmcmc=1000*, *tune=500*, *bkglim=None*, *back=None*, *samplefile=None*, *nrc=None*, *nbetas=6*, *depth=10*, *min_beta=0.6*)

Run Multiscale deprojection using the method described in Eckert+20

Parameters

- **backend** (*str*) – Backend to run the optimization problem. Available backends are 'pymc3' (default) and 'stan'
- **nmcmc** (*int*) – Number of HMC points in the output sample
- **tune** (*int*) – Number of HMC tuning steps
- **bkglim** (*float*) – Limit beyond which it is assumed that the background dominates, i.e. the source is set to 0. If bkglim=None (default), the entire radial range is used
- **back** (*float*) – Input value for the background, around which a gaussian prior is set. If back=None (default), the input background value will be computed as the average of the source-free region
- **samplefile** (*str*) – Path to output file to write the output samples. If samplefile=None (default), the data are not written to file and only loaded into memory
- **nrc** (*int*) – Number of core radii. If nrc=None (default), the number of core radiis will be defined on-the-fly

- **nbetas** (*int*) – Number of beta values. Default=6
- **depth** (*int*) – Set the max_treedepth parameter for Stan (default=10)
- **min_beta** (*float*) – Minimum value of beta. Default=0.6

Ncounts(*plot=True*, *outfile=None*, *figsize=(13, 10)*, *nbins=30*, *fontsize=40*, *yscale='linear'*, ***kwargs*)

Compute the total model number of counts. Optionally, the posterior distribution can be plotted and saved.

Parameters

- **plot** (*bool*) – Plot the posterior distribution of number of counts (default=True)
- **outfile** (*str*) – Output file name to save the figure. If outfile=None, plot only to stdout
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **nbins** (*int , optional*) – Number of bins on the X axis to construct the posterior distribution. Defaults to 30
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **yscale** (*str , optional*) – Scale on the Y axis. Defaults to 'linear'
- **kwargs** – Options to be passed to `matplotplib.pyplot.hist`

Returns Median number of counts, 16th and 84th percentiles

Return type float

OnionPeeling(*nmc=1000*)

Run standard Onion Peeling deprojection using the Kriss+83 method and the McLaughlin+99 edge correction

Parameters **nmc** (*int*) – Number of Monte Carlo realizations of the profile to compute the uncertainties (default=1000)

PlotDensity(*outfile=None*, *xunit='kpc'*, *figsize=(13, 10)*, *fontsize=40*, *color='C0'*, *lw=2*, ***kwargs*)

Plot the loaded density profile

Parameters

- **outfile** (*str*) – Output file name. If outfile=None (default) plot only to stdout
- **xunit** (*str*) – Choose whether the x axis should be in unit of 'kpc' (default), 'arcmin', or 'both', in which case two axes are drawn at the top and the bottom of the plot
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **color** (*str , optional*) – Line color following matplotlib conventions. Defaults to 'blue'
- **lw** (*int , optional*) – Line width. Defaults to 2
- **kwargs** – Additional arguments to be passed to `matplotlib.pyplot.plot`

PlotMgas(*rout=None*, *outfile=None*, *xunit='kpc'*, *figsize=(13, 10)*, *color='C0'*, *lw=2*, *fontsize=40*, *xscale='log'*, *yscale='log'*)

Plot the cumulative gas mass profile from the output of a reconstruction

Parameters

- **rout** (*numpy.ndarray*) – Radial binning of the gas mass profile. If rout=None, the original binning of the surface brightness profile is used

- **outfile** (*str*) – Output file name to save the figure. If outfile=None, plot only to stdout
- **xunit** (*str*) – Choose whether the x axis should be in unit of 'kpc' (default), 'arcmin', or 'both', in which case two axes are drawn at the top and the bottom of the plot
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **color** (*str , optional*) – Line color following matplotlib conventions. Defaults to 'C0'
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **xscale** (*str , optional*) – Scale of the X axis. Defaults to 'log'
- **yscale** (*str , optional*) – Scale of the Y axis. Defaults to 'log'
- **lw** (*int , optional*) – Line width. Defaults to 2

PlotSB (*outfile=None*, *figsize=(13, 10)*, *fontsize=40*, *xscale='log'*, *yscale='log'*, *lw=2*, *fmt='d'*, *markersize=7*, *data_color='red'*, *bkg_color='green'*, *model_color='C0'*, *skybkg_color='k'*)

Plot the surface brightness profile reconstructed after applying the multiscale deprojection and PSF deconvolution technique, and compare it with the input brightness profile

Parameters

- **outfile** (*str*) – File name of saving the output figure. If outfile=None (default), plot only to stdout
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **xscale** (*str , optional*) – Scale of the X axis. Defaults to 'log'
- **yscale** (*str , optional*) – Scale of the Y axis. Defaults to 'log'
- **lw** (*int , optional*) – Line width. Defaults to 2
- **fmt** (*str , optional*) – Marker type following matplotlib convention. Defaults to 'd'
- **markersize** (*int , optional*) – Marker size. Defaults to 7
- **data_color** (*str , optional*) – Color of the data points following matplotlib convention. Defaults to 'red'
- **bkg_color** (*str , optional*) – Color of the particle background following matplotlib convention. Defaults to 'green'
- **model_color** (*str , optional*) – Color of the surface brightness model following matplotlib convention. Defaults to 'C0'
- **skybkg_color** (*str , optional*) – Color of the fitted sky background model following matplotlib convention. Defaults to 'k'

Reload (*samplefile*, *bkglim=None*)

Reload the samples stored from a previous reconstruction run

Parameters

- **samplefile** (*str*) – Path to file containing the saved HMC samples
- **bkglim** (*float*) – Limit beyond which it is assumed that the background dominates, i.e. the source is set to 0. This parameter needs to be the same as the value used to run the reconstruction. If bkglim=None (default), the entire radial range is used

SaveAll(*outfile=None*)

Save the results of a profile reconstruction into an output FITS file First extension is data Second extension is density Third extension is Mgas Fourth extension is PSF

Parameters **outfile** (*str*) – Output file name

pyproffit.deproject.**Deproject_Multiscale_PyMC3**(*deproj*, *bkglim=None*, *nmcmc=1000*, *tune=500*, *back=None*, *samplefile=None*, *nrc=None*, *nbetas=6*, *min_beta=0.6*)

Run the multiscale deprojection optimization using the PyMC3 backend

Parameters

- **deproj** (class:*pyproffit.deproject.Deproject*) – Object of type *pyproffit.deproject.Deproject* containing the data and parameters
- **bkglim** (*float*) – Limit beyond which it is assumed that the background dominates, i.e. the source is set to 0. If bkglim=None (default), the entire radial range is used
- **nmcmc** (*int*) – Number of HMC points in the output sample
- **back** (*float*) – Input value for the background, around which a gaussian prior is set. If back=None (default), the input background value will be computed as the average of the source-free region
- **samplefile** (*str*) – Path to output file to write the output samples. If samplefile=None (default), the data are not written to file and only loaded into memory
- **nrc** (*int*) – Number of core radii. If nrc=None (default), the number of core radiis will be defined on-the-fly
- **nbetas** (*int*) – Number of beta values. Default=6
- **min_beta** (*float*) – Minimum value of beta. Default=0.6

pyproffit.deproject.**Deproject_Multiscale_Stan**(*deproj*, *bkglim=None*, *nmcmc=1000*, *back=None*, *samplefile=None*, *nrc=None*, *nbetas=6*, *depth=10*, *min_beta=0.6*)

Run the multiscale deprojection optimization using the Stan backend

Parameters

- **deproj** (class:*pyproffit.deproject.Deproject*) – Object of type *pyproffit.deproject.Deproject* containing the data and parameters
- **bkglim** (*float*) – Limit beyond which it is assumed that the background dominates, i.e. the source is set to 0. If bkglim=None (default), the entire radial range is used
- **nmcmc** (*int*) – Number of HMC points in the output sample
- **back** (*float*) – Input value for the background, around which a gaussian prior is set. If back=None (default), the input background value will be computed as the average of the source-free region
- **samplefile** (*str*) – Path to output file to write the output samples. If samplefile=None (default), the data are not written to file and only loaded into memory
- **nrc** (*int*) – Number of core radii. If nrc=None (default), the number of core radiis will be defined on-the-fly
- **nbetas** (*int*) – Number of beta values. Default=6
- **depth** (*int*) – Set the max_treedepth parameter for Stan (default=10)

- **min_beta** (*float*) – Minimum value of beta. Default=0.6

pyproffit.deproject.**EdgeCorr**(*nbin*, *rin_cm*, *rout_cm*, *em0*)

For Onion Peeling deprojection, correct for edge effects by estimating the contribution of flux beyond the edge, using McLaughlin+99 method

Parameters

- **nbin** (*int*) – Number of bins in input profile
- **rin_cm** (*numpy.ndarray*) – Inner radii of the bins in cm
- **rout_cm** (*numpy.ndarray*) – Outer radii of the bins in cm
- **em0** (*numpy.ndarray*) – Deprojected emissivity profile before edge correction

Returns Edge-corrected deprojected profile

Return type numpy.ndarray

class pyproffit.deproject.**MyDeprojVol**(*radin*, *radout*)

Bases: object

Class to compute the projection volumes

Parameters

- **radin** (class:*numpy.ndarray*) – Array of inner radii of the bins
- **radout** (class:*numpy.ndarray*) – Array of outer radii of the bins

deproj_vol()

Compute the projection volumes

Returns Volume matrix

Return type numpy.ndarray

pyproffit.deproject.**OP**(*deproj*, *nmc=1000*)

Run standard Onion Peeling deprojection including edge correction

Parameters

- **deproj** (class:*pyproffit.deproject.Deproject*) – Object of type *pyproffit.deproject.Deproject* containing the input data
- **nmc** (*int*) – Number of Monte Carlo realizations of the profile to compute the uncertainties (default=1000)

pyproffit.deproject.**calc_density_operator**(*rad*, *pars*, *z*, *cosmo*)

Compute linear operator to transform parameters into gas density profiles

Parameters

- **rad** (*numpy.ndarray*) – Array of input radii in arcmin
- **pars** (*numpy.ndarray*) – List of beta model parameters obtained through list_params
- **z** (*float*) – Source redshift

Returns Linear operator for gas density

Return type numpy.ndarray

pyproffit.deproject.**calc_int_operator**(*a*, *b*, *pars*)

Compute a linear operator to integrate analytically the basis functions within some radial range and return count rate and luminosities

Parameters

- **a** (*float*) – Lower integration boundary
- **b** (*float*) – Upper integration boundary
- **pars** (*numpy.ndarray*) – List of beta model parameters obtained through list_params

Returns Linear integration operator

Return type numpy.ndarray

pyproffit.deproject.**calc_linear_operator**(*rad*, *sourcereg*, *pars*, *area*, *expo*, *psf*)
Function to calculate a linear operator transforming parameter vector into predicted model counts

Parameters

- **rad** (*numpy.ndarray*) – Array of input radii in arcmin
- **sourcereg** (*numpy.ndarray*) – Selection array for the source region
- **pars** (*numpy.ndarray*) – List of beta model parameters obtained through list_params
- **area** (*numpy.ndarray*) – Bin area in arcmin^2
- **expo** (*numpy.ndarray*) – Bin effective exposure in s
- **psf** (*numpy.ndarray*) – PSF mixing matrix

Returns Linear projection and PSF mixing operator

Return type numpy.ndarray

pyproffit.deproject.**calc_sb_operator**(*rad*, *sourcereg*, *pars*)
Function to calculate a linear operator transforming parameter vector into surface brightness

Parameters

- **rad** (*numpy.ndarray*) – Array of input radii in arcmin
- **sourcereg** (*numpy.ndarray*) – Selection array for the source region
- **pars** (*numpy.ndarray*) – List of beta model parameters obtained through list_params

Returns Linear projection operator

Return type numpy.ndarray

pyproffit.deproject.**calc_sb_operator_psf**(*rad*, *sourcereg*, *pars*, *area*, *expo*, *psf*)
Same as calc_sb_operator but convolving the model surface brightness with the PSF model

Parameters

- **rad** (*numpy.ndarray*) – Array of input radii in arcmin
- **sourcereg** (*numpy.ndarray*) – Selection array for the source region
- **pars** (*numpy.ndarray*) – List of beta model parameters obtained through list_params
- **area** (*numpy.ndarray*) – Bin area in arcmin^2
- **expo** (*numpy.ndarray*) – Bin effective exposure in s
- **psf** (*numpy.ndarray*) – PSF mixing matrix

Returns Linear projection and PSF mixing operator

Return type numpy.ndarray

pyproffit.deproject.**fbul19**(*R*, *z*, *cosmo*, *Runit='kpc'*)
Compute Mgas from input R500 using Bulbul+19 M-Mgas scaling relation

Parameters

- **R** (*float*) – Input R500 value
- **z** (*float*) – Input redshift
- **Runit** (*str*) – Unit of input radis, kpc or arcmin (default='kpc')

Returns Mgas

Return type float

pyproffit.deproject.**list_params**(*rad*, *sourcereg*, *nrc=None*, *nbetas=6*, *min_beta=0.6*)
Define a list of parameters to define the dictionary of basis functions

Parameters

- **rad** (*numpy.ndarray*) – Array of input radii in arcmin
- **sourcereg** (*numpy.ndarray*) – Selection array for the source region
- **nrc** (*int*) – Number of core radii. If nrc=None (default), the number of core radiis will be defined on-the-fly
- **nbetas** (*int*) – Number of beta values. Default=6
- **min_beta** (*float*) – Minimum value of beta. Default=0.6

Returns Array containing sets of values to set up the function dictionary

Return type numpy.ndarray

pyproffit.deproject.**list_params_density**(*rad*, *sourcereg*, *z*, *cosmo*, *nrc=None*, *nbetas=6*, *min_beta=0.6*)
Define a list of parameters to transform the basis functions into gas density profiles

Parameters

- **rad** (*numpy.ndarray*) – Array of input radii in arcmin
- **sourcereg** (*numpy.ndarray*) – Selection array for the source region
- **z** (*float*) – Source redshift
- **nrc** (*int*) – Number of core radii. If nrc=None (default), the number of core radiis will be defined on-the-fly
- **nbetas** (*int*) – Number of beta values. Default=6
- **min_beta** (*float*) – Minimum value of beta. Default=0.6

Returns Array containing sets of values to set up the function dictionary

Return type numpy.ndarray

pyproffit.deproject.**medsmooth**(*prof*)
Smooth a given profile by taking the median value of surrounding points instead of the initial value

Parameters **prof** (*numpy.ndarray*) – Input profile to be smoothed

Returns Smoothd profile

Return type numpy.ndarray

pyproffit.deproject.**plot_multi_methods**(*profs*, *deps*, *labels=None*, *outfile=None*, *xunit='kpc'*, *figsize=(13, 10)*, *fontsize=40*, *xscale='log'*, *yscale='log'*, *fmt='.'*, *markersize=7*)

Plot multiple gas density profiles (e.g. obtained through several methods, centers or sectors) to compare them

Parameters

- **profs** (*tuple*) – List of Profile objects to be plotted
- **deps** (*tuple*) – List of Deproject objects to be plotted
- **labels** (*tuple*) – List of labels for the legend (default=None)
- **outfile** (*str*) – If outfile is not None, path to file name to output the plot
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **xscale** (*str , optional*) – Scale of the X axis. Defaults to 'log'
- **yscale** (*str , optional*) – Scale of the Y axis. Defaults to 'log'
- **fmt** (*str , optional*) – Marker type following matplotlib convention. Defaults to 'd'
- **markersize** (*int , optional*) – Marker size. Defaults to 7

4.4 pyproffit.emissivity module

pyproffit.emissivity.**calc_emissivity**(*cosmo*, *z*, *nh*, *kt*, *rmf*, *Z=0.3*, *elow=0.5*, *ehigh=2.0*, *arf=None*, *type='cr'*, *lum_elow=0.5*, *lum_ehigh=2.0*, *abund='aspl'*)

Function calc_emissivity. The function computes the scaling factor between count rate and APEC/MEKAL norm using XSPEC, which is needed to extract density profiles. Requires XSPEC to be in PATH

Parameters

- **cosmo** (*class: astropy.cosmology*) – Astropy cosmology object
- **z** (*float*) – Source redshift
- **nh** (*float*) – Source NH in units of 1e22 cm**(-2)
- **kt** (*float*) – Source temperature in keV
- **rmf** (*str*) – Path to response file (RMF/RSP)
- **Z** (*float*) – Metallicity with respect to solar (default = 0.3)
- **elow** (*float*) – Low-energy bound of the input image in keV (default = 0.5)
- **ehigh** (*float*) – High-energy bound of the input image in keV (default = 2.0)
- **arf** (*str*) – Path to on-axis ARF (optional, in case response file is RMF)
- **type** (*str*) – Specify whether the exposure map is in units of sec (type='cr') or photon flux (type='photon'). By default type='cr'.
- **lum_elow** (*float*) – Low energy bound (rest frame) for luminosity calculation. Defaults to 0.5
- **lum_ehigh** (*float*) – High energy bound (rest frame) for luminosity calculation. Defaults to 2.0

Returns Conversion factor

Return type float

pyproffit.emissivity.**is_tool**(*name*)
Check whether *name* is on PATH.

4.5 pyproffit.fitting module

class pyproffit.fitting.**ChiSquared**(*model*, *x*, *dx*, *y*, *dy*, *psfmat=None*, *fitlow=None*, *fithigh=None*)

Bases: object

Class defining a chi-square likelihood based on a surface brightness profile and a model. Let S_i be the measured surface brightness in annulus i and σ_i the corresponding Gaussian error. The likelihood function to be optimized is

$$-2\log\mathcal{L} = \sum_{i=1}^{N}\frac{(S_i - f(r_i))^2}{\sigma_i^2}$$

This class is called by the Fitter object when using the method='chi2' option.

Parameters

- **model** (class:*pyproffit.models.Model*) – Model definition. A *pyproffit.models.Model* object defining the model to be used.
- **x** (class:*numpy.ndarray*) – x axis data
- **dx** (class:*numpy.ndarray*) – x bin size data. dx is defined as half of the total bin size.
- **y** (class:*numpy.ndarray*) – y axis data
- **dy** (class:*numpy.ndarray*) – y error data
- **psfmat** (class:*numpy.ndarray* , optional) – PSF convolution matrix
- **fitlow** (*float , optional*) – Lower fitting boundary in arcmin. If fitlow=None the entire radial range is used, default to None
- **fithigh** (*float , optional*) – Upper fitting boundary in arcmin. If fithigh=None the entire radial range is used, default to None

errordef = 1.0

class pyproffit.fitting.**Cstat**(*model*, *x*, *dx*, *counts*, *area*, *effexp*, *bkgc*, *psfmat=None*, *fitlow=None*, *fithigh=None*)

Bases: object

Class defining a C-stat likelihood based on a surface brightness profile and a model. Let A_i , T_i be the area and the effective exposure time of annulus i. We set $F_i = f(r_i)A_iT_i$ the predicted number of counts in the annulus. The Poisson likelihood is then given by

$$-2\log\mathcal{L} = 2\sum_{i=1}^{N} F_i - C_i\log F_i - C_i + C_i\log C_i$$

with C_i the observed number of counts in annulus i.

This class is called by the Fitter object when using the method='cstat' option.

Parameters

- **model** (class:*pyproffit.models.Model*) – Model definition. A *pyproffit.models.Model* object defining the model to be used.

- **x** (*numpy.ndarray*) – x axis data
- **counts** (*numpy.ndarray*) – counts per bin data
- **area** (*numpy.ndarray*) – bin are in arcmin^2
- **effexp** (*numpy.ndarray*) – bin effective exposure in s
- **bkgc** (*numpy.ndarray*) – number of background counts per bin
- **psfmat** (*numpy.ndarray*) – PSF convolution matrix
- **fitlow** (*float*) – Lower fitting boundary in arcmin. If fitlow=None (default) the entire radial range is used
- **fithigh** (*float*) – Upper fitting boundary in arcmin. If fithigh=None (default) the entire radial range is used

errordef = 1.0

class pyproffit.fitting.**Fitter**(*model*, *profile*, *method='chi2'*, *fitlow=None*, *fithigh=None*, ***kwargs*)

Bases: object

Class containing the tools to fit surface brightness profiles with a model. Sets up the likelihood and optimizes for the parameters.

Parameters

- **model** (class:*pyproffit.models.Model*) – Object of type *pyproffit.models.Model* defining the model to be used.
- **profile** (class:*pyproffit.profextract.Profile*) – Object of type *pyproffit.profextract.Profile* containing the surface brightness profile to be fitted
- **method** (*str*) – Likelihood function to be optimized. Available likelihoods are 'chi2' (chi-squared) and 'cstat' (C statistic). Defaults to 'chi2'.
- **fitlow** (*float*) – Lower boundary of the active fitting radial range. If fitlow=None the entire range is used. Defaults to None
- **fithigh** (*float*) – Upper boundary of the active fitting radial range. If fithigh=None the entire range is used. Defaults to None
- **kwargs** – List of arguments to be passed to the iminuit library. For instance, setting parameter boundaries, optimization options or fixing parameters. See the iminuit documentation: https://iminuit.readthedocs.io/en/stable/index.html

Corner(*labels=None*, ***kwargs*)

Produce a parameter corner plot from a loaded set of samples. Uses the corner library: https://corner.readthedocs.io/en/latest/

Parameters

- **labels** (*list*) – List of names to be used
- **kwargs** – Any additional parameter to be passed to the corner library. See https://corner.readthedocs.io/en/latest/api.html

Returns Output matplotlib figure

Emcee(*nmcmc=5000*, *burnin=100*, *start=None*, *prior=None*, *walkers=32*, *thin=15*)

Run a Markov Chain Monte Carlo optimization using the affine-invariant ensemble sampler emcee. See https://emcee.readthedocs.io/en/stable/ for details.

Parameters

- **nmcmc** (*int*) – Number of MCMC samples. Defaults to 5000
- **burnin** (*int*) – Size of the burn-in phase that will eventually be ignored. Defaults to 100
- **start** (class:*numpy.ndarray*) – Array of input parameter values. If None, the code will look for the results of a previous Migrad optimization and use the corresponding parameters as starting values. Defaults to None
- **prior** (*function*) – Function defining the priors on the parameters. The function should take the parameter set as input and return the log prior probability. If None, the code will search for the results of a previous Migrad optimization and set up broad Gaussian priors on each parameter with sigma set to 5 times the Migrad errors. Defaults to None.
- **walkers** (*int*) – Number of emcee walkers. Defaults to 32.
- **thin** (*int*) – Thinning number for the output samples. The total number of sample values will be nmcmc*walkers/thin. Defaults to 15.

Migrad(*fixed=None*)

Perform maximum-likelihood optimization of the model using the MIGRAD routine from the MINUIT library.

Parameters **fixed** (class:*numpy.ndarray*) – A boolean array setting up whether parameters are fixed (True) or left free (False) while fitting. If None, all parameters are fitted. Defaults to None.

4.6 pyproffit.hmc module

pyproffit.hmc.**BetaModelPM**(*x*, *beta*, *rc*, *norm*, *bkg*)

pyproffit.hmc.**BknPowPM**(*x*, *alpha1*, *alpha2*, *norm*, *jump*, *bkg*, *rf=3.0*)

Broken power law 3D model projected along the line of sight for discontinuity modeling

$$I(r) = I_0 \int F(\omega)^2 d\ell + B$$

with $\omega^2 = r^2 + \ell^2$ and

$$F(\omega) = \begin{cases} \omega^{-\alpha_1}, & \omega < r_f \\ \frac{1}{C}\omega^{-\alpha_2}, & \omega \geq r_f \end{cases}$$

Parameters

- **x** (*numpy.ndarray*) – Radius in arcmin
- **alpha1** (class:*theano.tensor*) – α_1 parameter
- **alpha2** (class:*theano.tensor*) – α_2 parameter
- **rf** (*float*) – rf parameter
- **norm** (class:*theano.tensor*) – log of I0 parameter
- **jump** (class:*theano.tensor*) – C parameter
- **bkg** (class:*theano.tensor*) – log of B parameter

Returns Calculated model

Return type numpy.ndarray

pyproffit.hmc.**ConstPM**(*x*, *bkg*)

pyproffit.hmc.**DoubleBetaPM**(*x*, *beta*, *rc1*, *rc2*, *ratio*, *norm*, *bkg*)

class pyproffit.hmc.**HMCModel**(*model*, *start=None*, *sd=None*, *limits=None*, *fix=None*)
Bases: object

Class containing pyproffit model structure for HMC optimization

Parameters

- **model** (*function*) – Function to be used as surface brightness model
- **vals** (numpy.ndarray) – Array containing initial values for the parameters (optional)

SetPriors(*start=None*, *sd=None*, *limits=None*, *fix=None*)
Set prior definition for the function parameters

Parameters

- **start** –
- **sd** –
- **limits** –
- **fix** –

pyproffit.hmc.**IntFuncPM**(*omega*, *rf*, *alpha*, *xmin*, *xmax*)

pyproffit.hmc.**PowerLawPM**(*x*, *alpha*, *norm*, *pivot*, *bkg*)

pyproffit.hmc.**VikhlininPM**(*x*, *beta*, *rc*, *alpha*, *rs*, *epsilon*, *gamma*, *norm*, *bkg*)

pyproffit.hmc.**fit_profile_pymc3**(*hmcmod*, *prof*, *nmcmc=1000*, *tune=500*, *find_map=True*, *fitlow=0.0*, *fithigh=10000000000.0*)

4.7 pyproffit.miscellaneous module

pyproffit.miscellaneous.**bkg_smooth**(*data*, *smoothing_scale=25*)
Smooth an input background image using a broad Gaussian

Parameters

- **data** (class:*numpy.ndarray*) – 2D array
- **smoothing_scale** (*float*) – Size of Gaussian smoothing kernel in pixel. Defaults to 25

Returns smoothed image

Return type class:*numpy.ndarray*

pyproffit.miscellaneous.**clean_bkg**(*img*, *bkg*)
Subtract statistically the background from a Poisson image

Parameters

- **img** (class:*numpy.ndarray*) – Input image
- **bkg** (class:*numpy.ndarray*) – Background map

Returns Background subtracted Poisson image

Return type class:*numpy.ndarray*

pyproffit.miscellaneous.**dist_eval**(*coords2d*, *index=None*, *x_c=None*, *y_c=None*, *metric='euclidean'*, *selected=None*)

Computed distance of a set of points to a centroid

Parameters

- **coords2d** (class:*numpy.ndarray*) – 2D array with shape (N,2), N is the number of points
- **index** (class:*numpy.ndarray*) – index defining input value for the centroid
- **x_c** (*float*) – input centroid X axis coordinate
- **y_c** (*float*) – input centroid Y axis coordinate
- **metric** (*str , optional*) – metric system (defaults to 'euclidean')
- **selected** (class:*numpy.ndarray* , optional) – index defining a selected subset of points to be used

Returns distance

Return type class:*numpy.ndarray*

pyproffit.miscellaneous.**get_bary**(*x*, *y*, *x_c=None*, *y_c=None*, *weight=None*, *wdist=False*)

Compute centroid position and ellipse parameters from a set of points using principle component analysis

Parameters

- **x** (class:*numpy.ndarray*) – Array of positions on the X axis
- **y** (class:*numpy.ndarray*) – Array of positions on the Y axis
- **x_c** (*float , optional*) – Initial guess for the centroid X axis value
- **y_c** (*float , optional*) – Initial guess for the centroid X axis value
- **weight** (class:*numpy.ndarray* , optional) – Weights to be applied to the points when computing the average
- **wdist** (*bool*) – Switch to apply the weights. Defaults to False

Returns

- x_c_w (float): Centroid X coordinate
- y_c_w (float): Centroid Y coordinate
- sig_x (float): X axis standard deviation
- sig_y (float): Y axis standard deviation
- r_cluster (float): characteristic size of cluster
- semi_major_angle (float): rotation angle of ellipse
- pos_err (float): positional error on the centroid

pyproffit.miscellaneous.**heaviside**(*x*)

Heavyside theta function

Parameters **x** (class:*numpy.ndarray*) – 2D array

Returns filter

Return type class:*numpy.ndarray*

pyproffit.miscellaneous.**logbinning**(*binsize*, *maxrad*)

Set up a logarithmic binning scheme with a minimum bin size

Parameters

- **binsize** (*float*) – Minimum bin size in arcsec
- **maxrad** (*float*) – Maximum extraction radius in arcmin

Returns Bins and bin width

Return type class:*numpy.ndarray*

pyproffit.miscellaneous.**median_all_cov**(*dat*, *bins*, *ebins*, *rads*, *nsim=1000*, *fitter=None*, *thin=10*)

Generate Monte Carlo simulations of a Voronoi image and compute the median profile for each of them. The function returns an array of size (nbin, nsim) with nbin the number of bins in the profile and nsim the number of Monte Carlo simulations.

Parameters

- **dat** (class:*pyproffit.data.Data*) – A *pyproffit.data.Data* object containing the input Voronoi image and error map
- **bins** (class:*numpy.ndarray*) – Central value of radial binning
- **ebins** (class:*numpy.ndarray*) – Half-width of radial binning
- **rads** (class:*numpy.ndarray*) – Array containing the distance to the center, in arcmin, for each pixel
- **nsim** (*int*) – Number of Monte Carlo simulations to generate
- **fitter** (class:*pyproffit.fitter.Fitter*) – A pyproffit.fitter.Fitter object containing the result of a fit to the background region, for subtraction of the background to the resulting profile
- **thin** (*int*) – Number of blocks into which the calculation of the bootstrap will be divided. Increasing thin reduces memory usage drastically, at the cost of a modest increase in computation time.

Returns

- Samples of median profiles
- Area of each bin

Return type class:*numpy.ndarray*

pyproffit.miscellaneous.**medianval**(*vals*, *errs*, *nsim*)

Compute the median value of a sample of values and compute its uncertainty using Monte Carlo simulations

Parameters

- **vals** (class:*numpy.ndarray*) – Array containing the set of values in the sample
- **errs** (class:*numpy.ndarray*) – Array contatining the error on each value
- **nsim** (*int*) – Number of Monte Carlo simulations to be performed

Returns

- med (float): Median of sample
- err (float): Error on median

pyproffit.miscellaneous.**model_from_samples**(*x*, *model*, *samples*, *psfmat=None*)

Compute the median model and 1-sigma model envelope from a loaded chain, either from HMC or Emcee

Parameters

- **x** (class:*numpy.ndarray*) – Vector containing the X axis definition

- **model** (class:*pyproffit.models.Model*) – Fitted model
- **samples** (class:*numpy.ndarray*) – 2-dimensional array containing the parameter samples

Returns

- median (class:*numpy.ndarray*): Median model array
- model_lo (class:*numpy.ndarray*): Lower 1-sigma envelope array
- model_hi (class:*numpy.ndarray*): Upper 1-sigma envelope array

4.8 pyproffit.models module

pyproffit.models.**BetaModel**(*x*, *beta*, *rc*, *norm*, *bkg*)

Single Beta model

$$I(r) = I_0 \left(1 + (x/r_c)^2\right)^{-3\beta+0.5} + B$$

Parameters

- **x** (*numpy.ndarray*) – Radius in arcmin
- **beta** (*float*) – β parameter
- **rc** (*float*) – rc parameter
- **norm** (*float*) – log of I0 parameter
- **bkg** (*float*) – log of B parameter

Returns Calculated model

Return type numpy.ndarray

pyproffit.models.**BknPow**(*x*, *alpha1*, *alpha2*, *rf*, *norm*, *jump*, *bkg*)

Broken power law 3D model projected along the line of sight for discontinuity modeling

$$I(r) = I_0 \int F(\omega)^2 d\ell + B$$

with $\omega^2 = r^2 + \ell^2$ and

$$F(\omega) = \begin{cases} \omega^{-\alpha_1}, & \omega < r_f \\ \frac{1}{C}\omega^{-\alpha_2}, & \omega \geq r_f \end{cases}$$

Parameters

- **x** (*numpy.ndarray*) – Radius in arcmin
- **alpha1** (*float*) – α_1 parameter
- **alpha2** (*float*) – α_2 parameter
- **rf** (*float*) – rf parameter
- **norm** (*float*) – log of I0 parameter
- **jump** (*float*) – C parameter
- **bkg** (*float*) – log of B parameter

Returns Calculated model

Return type numpy.ndarray

pyproffit.models.**Const**(*x*, *bkg*)
Constal model for background fitting

Parameters

- **x** (*numpy.ndarray*) – Radius in arcmin
- **bkg** (*float*) – log of B parameter

Returns Calculated model

Return type numpy.ndarray

pyproffit.models.**DoubleBeta**(*x*, *beta*, *rc1*, *rc2*, *ratio*, *norm*, *bkg*)
Double beta model

$$I(r) = I_0 \left[(1 + (x/r_{c,1})^2)^{-3\beta+0.5} + R(1 + (x/r_{c,2})^2)^{-3\beta+0.5}\right] + B$$

Parameters

- **x** (*numpy.ndarray*) – Radius in arcmin
- **beta** (*float*) – β parameter
- **rc1** (*float*) – rc1 parameter
- **rc2** (*float*) – rc2 parameters
- **ratio** (*float*) – R parameter
- **norm** (*float*) – log of I0 parameter
- **bkg** (*float*) – log of B parameter

Returns Calculated model

Return type numpy.ndarray

pyproffit.models.**IntFunc**(*omega*, *rf*, *alpha*, *xmin*, *xmax*)
Numerical integration of a power law along the line of sight

$$\int_{x_{min}}^{x_{max}} \left(\frac{\omega^2 + \ell^2}{r_f^2}\right)^{-\alpha} d\ell$$

Parameters

- **omega** (*float*) – Projected radius
- **rf** (*float*) – rf parameter
- **alpha** (*float*) – α parameter
- **xmin** (*float*) – xmin parameter
- **xmax** (*float*) – xmax parameter

Returns Line-of-sight integral

Return type float

class pyproffit.models.**Model**(*model*, *vals=None*)
Bases: object

Class containing pyproffit models

Parameters

- **model** (*function*) – Function to be used as surface brightness model

- **vals** (numpy.ndarray) – Array containing initial values for the parameters (optional)

SetErrors(*vals*)
Set input values for the errors on the parameters

Parameters **vals** (numpy.ndarray) – Array containing initial values for the errors

SetParameters(*vals*)
Set input values for the model parameters

Parameters **vals** (numpy.ndarray) – Array containing initial values for the parameters

pyproffit.models.**PowerLaw**(*x*, *alpha*, *norm*, *pivot*, *bkg*)
Single power law model in projected space

$$I(r) = I_0 \left(\frac{x}{x_p}\right)^{-\alpha} + B$$

Parameters

- **x** (*numpy.ndarray*) – Radius in arcmin
- **alpha** (*float*) – α parameter
- **norm** (*float*) – log of I0 parameter
- **pivot** (*float*) – x_p parameter
- **bkg** (*float*) – log of B parameter

Returns Calculated model

Return type numpy.ndarray

pyproffit.models.**Vikhlinin**(*x*, *beta*, *rc*, *alpha*, *rs*, *epsilon*, *gamma*, *norm*, *bkg*)
Simplified Vikhlinin+06 surface brightness model used in Ghirardini+19

$$I(r) = I_0 \left(\frac{x}{r_c}\right)^{-\alpha} \left(1 + (x/r_c)^2\right)^{-3\beta+\alpha/2} \left(1 + (x/r_s)^\gamma\right)^{-\epsilon/\gamma}$$

Parameters

- **x** (*numpy.ndarray*) – Radius in arcmin
- **beta** (*float*) – β parameter
- **rc** (*float*) – rc parameter
- **norm** (*float*) – log of I0 parameter
- **alpha** (*float*) – α parameter
- **rs** (*float*) – rs parameter
- **epsilon** (*float*) – ϵ parameter
- **gamma** (*float*) – γ parameter
- **bkg** (*float*) – log of B parameter

Returns Calculated model

Return type numpy.ndarray

4.9 pyproffit.power_spectrum module

class pyproffit.power_spectrum.**PowerSpectrum**(*data*, *profile*)
Bases: object

Class to perform fluctuation power spectrum analysis from Poisson count images. This is the code used in Eckert et al. 2017.

Parameters

- **data** (class:*pyproffit.data.Data*) – Object of type *pyproffit.data.Data* including the image, exposure map, background map, and region definition
- **profile** (class:*pyproffit.profextract.Profile*) – Object of type *pyproffit.profextract.Profile* including the extracted surface brightness profile

MexicanHat(*modimg_file*, *z*, *region_size=1.0*, *factshift=1.5*)
Convolve the input image and model image with a set of Mexican Hat filters at various scales. The convolved images are automatically stored into FITS images called conv_scale_xx.fits and conv_beta_xx.fits, with xx the scale in kpc.

Parameters

- **modimg_file** (*str*) – Path to a FITS file including the model image, typically produced with *pyproffit.profextract.Profile.SaveModelImage()*
- **z** (*float*) – Source redshift
- **region_size** (*float*) – Size of the region of interest in Mpc. Defaults to 1.0
- **factshift** (*float*) – Size of the border around the region, i.e. a region of size factshift * region_size is used for the computation. Defaults to 1.5

PS(*z*, *region_size=1.0*, *radius_in=0.0*, *radius_out=1.0*)
Function to compute the power spectrum from existing Mexican Hat images in a given circle or annulus

Parameters

- **z** (*float*) – Source redshift
- **region_size** (*float*) – Size of the region of interest in Mpc. Defaults to 1.0. This value must be equal to the region_size parameter used in *pyproffit.power_spectrum.PowerSpectrum.MexicanHat()*.
- **radius_in** (*float*) – Inner boundary in Mpc of the annulus to be used. Defaults to 0.0
- **radius_out** (*float*) – Outer boundary in Mpc of the annulus to be used. Defaults to 1.0

Plot(*save_plots=True*, *outps='power_spectrum.pdf'*, *outamp='a2d.pdf'*, *plot_3d=False*, *cfact=None*)
Plot the loaded power spectrum

Parameters

- **save_plots** (*bool*) – Indicate whether the plot should be saved to disk or not. Defaults to True.
- **outps** (*str*) – Name of output file. Defaults to 'power_spectrum.pdf'
- **outamp** (*str*) – Name of output file to save the 2D amplitude plot. Defaults to 'a2d.pdf'
- **plot_3d** (*bool*) – Add or not the 3D power spectrum to the plot. Defaults to False
- **cfact** (class:*numpy.ndarray* , optional) – 2D to 3D projection factor

ProjectionFactor(*z*, *betaparams*, *region_size=1.0*)
Compute numerically the 2D to 3D deprojection factor. The routine is simulating 3D fluctuations using the surface brightness model and the region mask, projecting the 3D data along one axis, and computing the ratio of 2D to 3D power as a function of scale.

Caution: this is a memory intensive computation which will run into memory overflow if the image size is too large or the available memory is insufficient.

Parameters

- **z** (*float*) – Source redshift
- **betaparams** (class:*numpy.ndarray*) – Parameters of the beta model or double beta model
- **region_size** (*float*) – Size of the region of interest in Mpc. Defaults to 1.0. This value must be equal to the region_size parameter used in *pyproffit.power_spectrum.PowerSpectrum.MexicanHat()*.

Returns Array of projection factors

Return type class:*numpy.ndarray*

Save(*outfile*, *outcov='covariance.txt'*)
Save the loaded power spectra to an output ASCII file

Parameters

- **outfile** (*str*) – Name of output ASCII file
- **outcov** (*str*) – Output covariance matrix. Defaults to 'covariance.txt'

pyproffit.power_spectrum.**betamodel**(*x*, *par*)

pyproffit.power_spectrum.**calc_mexicanhat**(*sc*, *img*, *mask*, *simmod*)
Filter an input image with a Mexican-hat filter

Parameters

- **sc** (*float*) – Mexican Hat scale in pixel
- **img** (class:*numpy.ndarray*) – Image to be smoothed
- **mask** (class:*numpy.ndarray*) – Mask image
- **simmod** (class:*numpy.ndarray*) – Model surface brightness image

Returns Mexican Hat convolved image and SB model

Return type class:*numpy.ndarray*

pyproffit.power_spectrum.**calc_projection_factor**(*nn*, *mask*, *betaparams*, *scale*)
Compute numerically the 2D to 3D deprojection factor. The routine is simulating 3D fluctuations using the surface brightness model and the region mask, projecting the 3D data along one axis, and computing the ratio of 2D to 3D power as a function of scale.

Caution: this is a memory intensive computation which will run into memory overflow if the image size is too large or the available memory is insufficient.

Parameters

- **nn** (*int*) – Image size in pixel
- **mask** (class:*numpy.ndarray*) – Array defining the mask of active pixels, of size (nn , nn)
- **betaparams** (class:*numpy.ndarray*) – Parameters of the beta model or double beta model

- **scale** (class:*numpy.ndarray*) – Array of scales at which the projection factor should be computed

Returns Array containing wave number, 2D power, 2D amplitude, and 3D power

Return type class:*numpy.ndarray*

pyproffit.power_spectrum.**calc_ps**(*region*, *img*, *mod*, *kr*, *nreg*)

Function to compute the power at a given scale kr from the Mexican Hat filtered images

Parameters

- **region** (class:*numpy.ndarray*) – Index defining the region from which the power spectrum will be extracted
- **img** (class:*numpy.ndarray*) – Mexican Hat filtered image
- **mod** (class:*numpy.ndarray*) – Mexican Hat filtered SB model
- **kr** (*float*) – Extraction scale
- **nreg** (*int*) – Number of subregions into which the image should be splitted to perform the bootstrap

Returns

- ps (float): Power at scale kr
- psnoise (float): Noise at scale kr
- vals (class:*numpy.ndarray*): set of values for bootstrap error calculation

pyproffit.power_spectrum.**do_bootstrap**(*vals*, *nsample*)

Compute the covariance matrix of power spectra by bootstrapping the image

Parameters

- **vals** (class:*numpy.ndarray*) – Set of values
- **nsample** (*int*) – Number of bootstrap samples

Returns 2D covariance matrix

Return type class:*numpy.ndarray*

pyproffit.power_spectrum.**doublebeta**(*x*, *pars*)

4.10 pyproffit.profextract module

class pyproffit.profextract.**Profile**(*data=None*, *center_choice=None*, *maxrad=None*, *binsize=None*, *center_ra=None*, *center_dec=None*, *binning='linear'*, *centroid_region=None*, *bins=None*, *cosmo=None*)

Bases: object

pyproffit.Profile class. The class allows the user to extract surface brightness profiles and use them to fit models, extract density profiles, etc.

Parameters

- **data** (class:*pyproffit.Data*) – Object of type *pyproffit.data.Data* containing the data to be used
- **center_choice** (*str*) – Choice of the center of the surface brightness profile. Available options are “centroid”, “peak”, “custom_ima” and “custom_fk5”. Args:

  - 'centroid': Compute image centroid and ellipticity. This is done by performing principle component analysis on the count image. If a dmfilth image exists, it will be used instead of the original count image.
  - 'peak': Compute the surface brightness peak, The peak is computed as the maximum of the count image after a light smoothing. If a dmfilth image exists, it will be used instead of the original count image.
  - 'custom_fk5': Use any custom center in FK5 coordinates, provided by the "center_ra" and "center_dec" arguments
  - 'custom_ima': Similar to 'custom_fk5' but with input coordinates in image pixels
- **maxrad** (*float*) – The maximum radius (in arcmin) out to which the surface brightness profile will be computed
- **binsize** (*float*) – Minumum bin size (in arcsec).
- **center_ra** (*float*) – User defined center R.A. If center_choice='custom_fk5' this is the right ascension in degrees. If center_choice='custom_ima' this is the image pixel on the X axis. If center_choice='peak' or 'centroid' this is not used.
- **center_dec** (*float*) – User defined center declination. If center_choice='custom_fk5' this is the declination in degrees. If center_choice='custom_ima' this is the image pixel on the Y axis. If center_choice='peak' or 'centroid' this is not used.
- **binning** (*str*) – Binning type. Available types are 'linear', 'log' or 'custom'. Defaults to 'linear'. Args:
  - 'linear': Use a linear radial binning with bin size equal to 'binsize'
  - 'log': Use logarithmic binning, i.e. bin size increasing logarithmically with radius with a minimum bin size given by 'binsize'
  - 'custom': Any user-defined binning in the form of an input numpy array provided through the 'bins' option
- **centroid_region** (*float*) – If center_choice='centroid', this option defines the radius of the region (in arcmin), centered on the center of the image, within which the centroid will be calculated. If centroid_region=None the entire image is used. Defaults to None.
- **bins** (class:*numpy.ndarray*) – in case binning is set to 'custom', a numpy array containing the binning definition. For an input array of length N, the binning will contain N-1 bins with boundaries set as the values of the input array.
- **cosmo** (class:*astropy.cosmology*) – An `astropy.cosmology` object containing the definition of the cosmological model. If cosmo=None, Planck 2015 cosmology is used.

AzimuthalScatter(*nsect=12*, *model=None*)

Compute the azimuthal scatter profile around the loaded profile. The azimuthal scatter is defined as the standard deviation of the surface brightness in equispaced sectors with respect to the azimuthal mean,

$$\Sigma_X(r) = \frac{1}{N}\sum_{i=1}^{N}\frac{(S_i(r) - \langle S(r)\rangle)^2}{\langle S(r)\rangle^2}$$

with N the number of sectors and $\langle S(r)\rangle$ the loaded mean surface brightness profile.

Parameters

- **nsect** (*int*) – Number of sectors from which the azimuthal scatter will be computed. Defaults to nsect=12

- **model** (class:*pyproffit.models.Model*) – A `pyproffit.models.Model` object containing the background to be subtracted, in case the scatter is to be computed on background-subtracted profiles. Defaults to None (i.e. no background subtraction).

Backsub(*fitter*)

Subtract a fitted background value from the loaded surface brightness profile. Each pyproffit model contains a 'bkg' parameter, which will be fitted and loaded in a Fitter object. The routine reads the value of 'bkg', subtracts it from the data, and adds its error in quadrature to the error profile.

Parameters **fitter** (class:*pyproffit.fitting.Fitter*) – Object of type `pyproffit.fitting.Fitter` containing a model and optimization results

Emissivity(*z=None*, *nh=None*, *kt=6.0*, *rmf=None*, *Z=0.3*, *elow=0.5*, *ehigh=2.0*, *arf=None*, *type='cr'*, *lum_elow=0.5*, *lum_ehigh=2.0*)

Use XSPEC to compute the conversion from count rate to emissivity using the pyproffit.calc_emissivity routine (see its description)

Parameters

- **z** (*float*) – Source redshift
- **nh** (*float*) – Source NH in units of 1e22 cm**(-2)
- **kt** (*float*) – Source temperature in keV. Default to 6.0
- **rmf** (*str*) – Path to response file (RMF/RSP)
- **Z** (*float*) – Metallicity with respect to solar. Defaults to 0.3
- **elow** (*float*) – Low-energy bound of the input image in keV. Defaults to 0.5
- **ehigh** (*float*) – High-energy bound of the input image in keV. Defaults to 2.0
- **arf** (*str , optional*) – Path to on-axis ARF in case response file type is RMF)
- **type** (*str*) – Specify whether the exposure map is in units of sec (type='cr') or photon flux (type='photon'). Defaults to 'cr'
- **lum_elow** (*float*) – Low energy bound (rest frame) for luminosity calculation. Defaults to 0.5
- **lum_ehigh** (*float*) – High energy bound (rest frame) for luminosity calculation. Defaults to 2.0

Returns Conversion factor

Return type float

MedianSB(*ellipse_ratio=1.0*, *rotation_angle=0.0*, *nsim=1000*, *outsamples=None*, *fitter=None*, *thin=10*)

Extract the median surface brightness profile in circular annuli from a provided Voronoi binned image, following the method outlined in Eckert et al. 2015

Parameters

- **ellipse_ratio** (*float*) – Ratio a/b of major to minor axis in the case of an elliptical annulus definition. Defaults to 1.0, i.e. circular annuli.
- **rotation_angle** (*float*) – Rotation angle of the ellipse or box respective to the R.A. axis. Defaults 0.
- **nsim** (*int*) – Number of Monte Carlo realizations of the Voronoi image to be performed
- **outsamples** (*str*) – Name of output FITS file to store the bootstrap realizations of the median profile. Defaults to None

- **fitter** (class:*pyproffit.fitter.Fitter*) – A `pyproffit.fitter.Fitter` object containing the result of a fit to the background region, for subtraction of the background to the resulting profile
- **thin** (*int*) – Number of blocks into which the calculation of the bootstrap will be divided. Increasing thin reduces memory usage drastically, at the cost of a modest increase in computation time.

PSF (*psffunc=None*, *psffile=None*, *psfimage=None*, *psfpixsize=None*, *sourcemodel=None*)
Function to calculate a PSF convolution matrix given an input PSF image or function. To compute the PSF mixing matrix, images of each annuli are convolved with the PSF image using FFT and determine the fraction of photons leaking into neighbouring annuli. FFT-convolved images are then used to determine a mixing matrix. See Eckert et al. 2020 for more details.

Parameters

- **psffunc** (*function*) – Function describing the radial shape of the PSF, with the radius in arcmin
- **psffile** (*str*) – Path to file containing an image of the PSF. The pixel size must be equal to the pixel size of the image.
- **psfimage** (class:*numpy.ndarray*) – Array containing an image of the PSF. The pixel size must be equal to the pixel size of the image.
- **psfpixsize** (*float*) – (currently inactive) Pixel size of the PSF image in arcsec. Currently not implemented.
- **sourcemodel** (class:*pyproffit.models.Model*) – Object of type *pyproffit.models.Model* including a surface brightness model to account for surface brightness gradients across the bins. If sourcemodel=None a flat distribution is assumed across each bin. Defaults to None

Plot (*model=None*, *samples=None*, *outfile=None*, *axes=None*, *scatter=False*, *figsize=(13, 10)*, *fontsize=40.0*, *xscale='log'*, *yscale='log'*, *fmt='o'*, *markersize=7*, *lw=2*, *data_color='black'*, *bkg_color='green'*, *model_color='blue'*, ***kwargs*)
Plot the loaded surface brightness profile

Parameters

- **model** (class:*pyproffit.models.Model* , optional) – If model is not None, plot the provided model of type *pyproffit.models.Model* together with the data. Defaults to None
- **samples** (class:*numpy.ndarray*) – Use parameter samples outputted either by Emcee or HMC optimization to compute the median optimized models and upper and lower envelopes. Defaults to None
- **outfile** (*str , optional*) – If outfile is not None, name of output file to save the plot. Defaults to None
- **axes** (*list , optional*) – List of 4 numbers defining the X and Y axis ranges for the plot. Gives axes=[x1, x2, y1, y2], the X axis will be set between x1 and x2, and the Y axis will be set between y1 and y2.
- **scatter** (*bool*) – Set whether the azimuthal scatter profile will be displayed instead of the surface brightness profile. Defaults to False.
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **xscale** (*str , optional*) – Scale of the X axis. Defaults to 'log'
- **yscale** (*str , optional*) – Scale of the Y axis. Defaults to 'log'

- **lw** (*int , optional*) – Line width. Defaults to 2
- **fmt** (*str , optional*) – Marker type following matplotlib convention. Defaults to 'd'
- **markersize** (*int , optional*) – Marker size. Defaults to 7
- **data_color** (*str , optional*) – Color of the data points following matplotlib convention. Defaults to 'red'
- **bkg_color** (*str , optional*) – Color of the particle background following matplotlib convention. Defaults to 'green'
- **model_color** (*str , optional*) – Color of the model curve following matplotlib convention. Defaults to 'blue'
- **kwargs** – Arguments to be passed to `matplotlib.pyplot.errorbar`

SBprofile(*ellipse_ratio=1.0*, *rotation_angle=0.0*, *angle_low=0.0*, *angle_high=360.0*, *box=False*, *width=None*)

Extract a surface brightness profile and store the results in the input Profile object

Parameters

- **ellipse_ratio** (*float*) – Ratio a/b of major to minor axis in the case of an elliptical annulus definition. Defaults to 1.0, i.e. circular annuli.
- **rotation_angle** (*float*) – Rotation angle of the ellipse or box respective to the R.A. axis. Defaults 0.
- **angle_low** (*float*) – In case the surface brightness profile should be extracted across a sector instead of the whole azimuth, lower position angle of the sector respective to the R.A. axis. Defaults to 0
- **angle_high** (*float*) – In case the surface brightness profile should be extracted across a sector instead of the whole azimuth, upper position angle of the sector respective to the R.A. axis. Defaults to 360
- **voronoi** (*bool*) – Set whether the input data is a Voronoi binned image (True) or a standard raw count image (False). Defaults to False.
- **box** (*bool*) – Define whether the profile should be extract along an annulus or a box. The parameter definition of the box matches the DS9 definition. Defaults to False.
- **width** (*float*) – In case box=True, set the full width of the box (in arcmin)

Save(*outfile=None*, *model=None*)

Save the data loaded in the Profile class into an output FITS file.

Parameters

- **outfile** (*str*) – Output file name
- **model** (class:*pyproffit.models.Model* , optional) – If model is not None, Object of type *pyproffit.models.Model* including the fitted model. Defaults to None

SaveModelImage(*outfile*, *model=None*, *vignetting=True*)

Compute a model image and output it to a FITS file

Parameters

- **model** (class:*pyproffit.models.Model*) – Object of type *pyproffit.models.Model* including the surface brightness model. If model=None (default), the azimuthally averaged radial profile is interpolated at each point.
- **outfile** (*str*) – Name of output file

- **vignetting** (*bool*) – Choose whether the model will be convolved with the vignetting model (i.e. multiplied by the exposure map) or if the actual surface brightness will be extracted (False). Defaults to True

pyproffit.profextract.**plot_multi_profiles**(*profs*, *labels=None*, *outfile=None*, *axes=None*, *figsize=(13, 10)*, *fontsize=40*, *xscale='log'*, *yscale='log'*, *fmt='o'*, *markersize=7*)

Plot multiple surface brightness profiles on a single plot. This feature can be useful e.g. to compare profiles across multiple sectors

Parameters

- **profs** (*tuple*) – List of Profile objects to be plotted
- **labels** (*tuple*) – List of labels for the legend (default=None)
- **outfile** (*str*) – If outfile is not None, path to file name to output the plot
- **axes** (*list , optional*) – List of 4 numbers defining the X and Y axis ranges for the plot. Gives axes=[x1, x2, y1, y2], the X axis will be set between x1 and x2, and the Y axis will be set between y1 and y2.
- **figsize** (*tuple , optional*) – Size of figure. Defaults to (13, 10)
- **fontsize** (*int , optional*) – Font size of the axis labels. Defaults to 40
- **xscale** (*str , optional*) – Scale of the X axis. Defaults to 'log'
- **yscale** (*str , optional*) – Scale of the Y axis. Defaults to 'log'
- **lw** (*int , optional*) – Line width. Defaults to 2
- **fmt** (*str , optional*) – Marker type following matplotlib convention. Defaults to 'd'
- **markersize** (*int , optional*) – Marker size. Defaults to 7

4.11 pyproffit.reload module

pyproffit.reload.**Reload**(*infile*, *model=None*)

Reload the results of a previous session into new Profile and Data objects. If model=None and a Model structure is present, it will be read and its values will be stored into the input Model structure.

Parameters

- **infile** (*str*) – Path to a FITS file saved through the *pyproffit.profextract.Profile.Save* method
- **model** (class:*pyproffit.models.Model* , str) – A *pyproffit.models.Model* object including the model to be saved

Returns

- dat: a *pyproffit.data.Data* object including the reloaded data files
- prof: a *pyproffit.profextract.Profile* object including the reloaded profile

4.12 Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

H

I

L

M

N

O

P

R

S

V