sep Documentation

Release 0.5.2

Kyle Barbary

January 04, 2016

Contents

Source Extraction and Photometry in Python

SEP makes available some of the algorithms in Source Extractor as stand-alone functions and classes. These operate directly on in-memory numpy arrays (no FITS files, configuration files, etc). It's derived directly from (and tested against) the Source Extractor code base.

Some features:

- spatially variable background and noise estimation
- source extraction, with on-the-fly convolution and source deblending
- circular and elliptical aperture photometry
- fast: implemented in C with Python bindings via Cython

Additional features not in Source Extractor:

- Optimized matched filter for variable noise in source extraction
- circular annulus and elliptical annulus apertures
- Local background subtraction in shape consistent with aperture
- exact pixel overlap mode in all aperture photometry functions
- ellipse masking

Installation

SEP supports both Python 3 and Python 2 and requires only numpy. To install:

```
pip install --no-deps sep
```

Development version / source code: http://github.com/kbarbary/sep

CHAPTER 2

Usage Guide

2.1 Background estimation & source detection

Most optical/IR data must be background subtracted before sources can be detected. In SEP, background estimation and source detection are two separate steps.

2.1.1 Background estimation

Given a numpy array `data` containing the image data,

```
import numpy as np
import sep

# Measure a spatially variable background of some image data
# (a numpy array)
bkg = sep.Background(data)

# ... or with some optional parameters
bkg = sep.Background(data, mask=mask, bw=64, bh=64, fw=3, fh=3)
```

This creates a *Background* object which you can then use in several ways:

```
# Evaluate the spatially variable background:
back = bkg.back()  # creates an array, same shape and type as data
back = np.array(bkg)  # equivalent to the above

# Evaluate the spatially variable RMS of the background:
rms = bkg.rms()  # creates an array, same shape and type as data

# Subtract the background from the data
bkg_subtraced_data = data - bkg

# Directly subtract the background from the data in place
bkg.subfrom(data)  # data is modified in-place

# get global statistics
bkg.globalback  # Global "average" background level
bkg.globalrms  # Global "average" RMS of background
```

Note: If you are using SEP to analyze data read from FITS files with astropy.io.fits, you may see an error message such as:

```
ValueError: Input array with dtype '>f4' has non-native byte order.
Only native byte order arrays are supported. To change the byte
order of the array 'data', do 'data = data.byteswap().newbyteorder()'
```

It is usually easiest to do this byte-swap operation directly after reading the array from the FITS file. You can even perform the byte swap in-place by doing

```
>>> data = data.byteswap(inplace=True).newbyteorder()
```

If you do this in-place operation, ensure that there are no other references to `data`, as they will be rendered nonsensical.

For the interested reader, this byteswap operation is necessary because astropy.io.fits always returns big-endian byte order arrays, even on little-endian machines. For more on this, see this FAQ.

2.1.2 Source detection

Once the data has been background subtracted, detect objects in the data, given some threshold:

```
thresh = 1.5 * bkg.globalrms
objects = sep.extract(data, thresh)

# objects is a numpy structured array:
len(objects)  # number of objects
objects['x'][i]  # flux-weighted x-center coordinate of i-th object
...              # ... and many other fields.
```

See the reference section for all the fields available in the returned structured array.

> **Warning:** If the data array is not background-subtracted or the threshold is too low, you will tend to get one giant object or, more likely, an exception will be raised due to exceeding the internal memory constraints of the function.

2.1.3 The matched filter (convolution)

For source detection, SEP supports using a matched filter, which can give the optimal detection signal-to-noise for objects with some known shape. This is controlled using the `filter_kernel` keyword in *sep.extract*. For example:

```
kernel = np.array([[1., 2., 3., 2., 1.],
                   [2., 3., 5., 3., 2.],
                   [3., 5., 8., 5., 3.],
                   [2., 3., 5., 3., 2.],
                   [1., 2., 3., 2., 1.]])
objects = sep.extract(data, thresh, filter_kernel=kernel)
```

If `filter_kernel` is not specified, a default 3-by-3 kernel is used. To disable filtering entirely, specify `filter_kernel=None`.

What array should be used for `filter_kernel`? It should be approximately the shape of the objects you are trying to detect. For example, to optimize for the detection of point sources, `filter_kernel` should be set to shape of the point spread function (PSF) in the data. For galaxy detection, a larger kernel could be used. In practice, anything that is roughly the shape of the desired object works well since the main goal is to negate the effects of background noise, and a reasonable estimate is good enough.

Correct treatment in the presence of variable noise

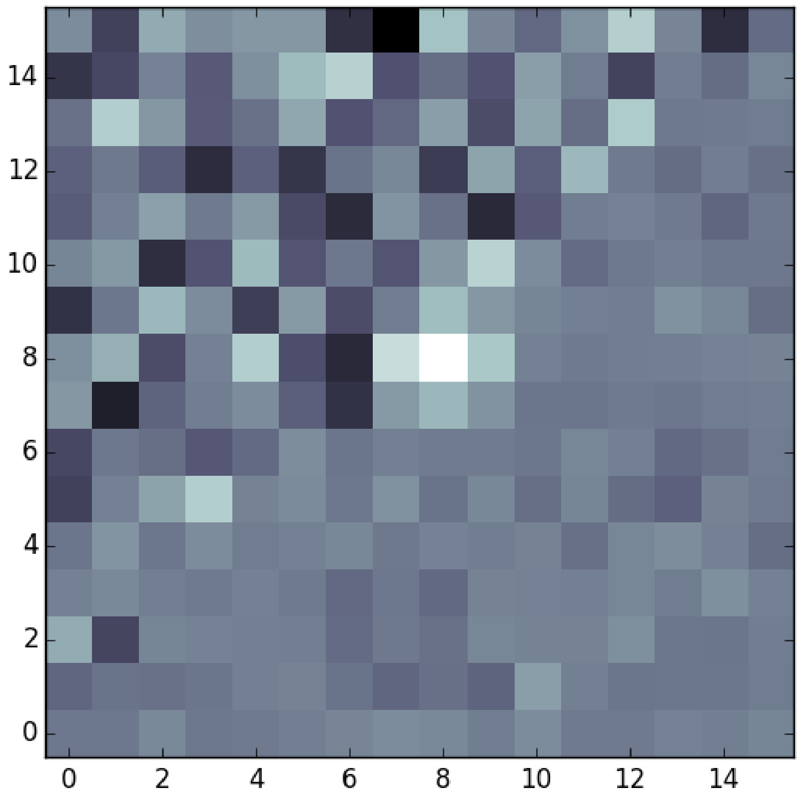
In Source Extractor, the matched filter is implemented assuming there is equal noise across all pixels in the kernel. The matched filter then simplifies to a convolution of the data with the kernel. In *sep.extract*, this is also the behavior when there is constant noise (when `err` is not specified).

In the presence of independent noise on each pixel, SEP uses a full matched filter implementation that correctly accounts for the noise in each pixel. This is not available in Source Extractor. Some benefits of this method are that detector sensitivity can be taken into account and edge effects are handled gracefully. For example, suppose we have an image with noise that is higher in one region than another. This can often occur when coadding images:

```
# create a small image with higher noise in the upper left
n = 16
X, Y = np.meshgrid(np.arange(n), np.arange(n))
mask = Y > X
error = np.ones((n, n))
error[mask] = 4.0
data = error * np.random.normal(size=(n, n))

# add source to middle of data
source = 3.0 * np.array([[1., 2., 1.],
                         [2., 4., 2.],
                         [1., 2., 1.]])
m = n // 2 - 1
data[m:m+3, m:m+3] += source

plt.imshow(data, interpolation='nearest', origin='lower', cmap='bone')
```



Specifying `filter_type='conv'` will use simple convolution, matching the behavior of Source Extractor. The object is not detected:

```
>>> objects = sep.extract(data, 3.0, err=error, filter_type='conv')
>>> len(objects)
0
```

Setting `filter_type='matched'` (the default) correctly deweights the noisier pixels around the source and detects the object:

```
>>> objects = sep.extract(data, 3.0, err=error, filter_type='matched')
>>> len(objects)
1
```

Derivation of the matched filter formula

Assume that we have an image containing a single point source. This produces a signal with PSF S_i and noise N_i at each pixel indexed by i. Then the measured image data D_i (i.e. our pixel values) is given by:

$$D_i = S_i + N_i$$

Then we want to apply a linear transformation T_i which gives an output Y:

$$Y = \sum_i T_i D_i = T^T D$$

We use matrix notation from here on and drop the explicit sums. Our objective is to find the transformation T_i which maximizes the signal-to-noise ratio SNR.

$$SNR^2 = \frac{(T^T S)^2}{E[(T^T N)^2]}$$

We can expand the denominator as:

$$E[(T^T N)^2] = E[(T^T N)(N^T T)] = T^T \cdot E[NN^T] \cdot T = T^T CT$$

Where C_{ik} is the covariance of the noise between pixels i and k. Now using the Cauchy-Schwarz inequality on the numerator:

$$(T^T S)^2 = (T^T C^{1/2} C^{-1/2} S)^2 \leq (T^T C^{1/2})^2 (C^{-1/2} S)^2 = (T^T CT)(S^T C^{-1} S)$$

since $C^T = C$. The signal-to-noise ratio is therefore bounded by:

$$\begin{aligned} SNR^2 &\leq \frac{(T^T CT)(S^T C^{-1} S)}{(T^T CT)} \\ SNR^2 &\leq S^T C^{-1} S \end{aligned}$$

Choosing $T = \alpha C^{-1} S$ where α is an arbitrary normalization constant, we get equality. Hence this choise of T is the optimal linear tranformation. We normalize this linear transformation so that if there is no signal and only noise, we get an expected signal-to-noise ratio of 1. With this definition, the output SNR represents the number of standard deviations above the background. This gives:

$$\begin{aligned} &E[(T^T N)^2] = T^T CT = \alpha^2 S^T C^{-1} C C^{-1} S = \alpha^2 S^T C^{-1} S = 1 \\ &\alpha = \frac{1}{\sqrt{S^T C^{-1} S}} \end{aligned}$$

Putting everything together, our normalized linear transformation is:

$$T = \frac{C^{-1} S}{\sqrt{S^T C^{-1} S}}$$

And the optimal signal-to-noise is given in terms of the known variables as:

$$SNR = \frac{S^T C^{-1} D}{\sqrt{S^T C^{-1} S}}$$

2.2 Aperture photometry

There are four aperture functions available:

Function	Sums data within...
`sep.sum_circle`	circle(s)
`sep.sum_circann`	circular annulus/annuli
`sep.sum_ellipse`	ellipse(s)
`sep.sum_ellipann`	elliptical annulus/annuli

The follow examples demonstrate options for circular aperture photometry. The other functions behave similarly.

```
# sum flux in circles of radius=3.0
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0)

# x, y and r can be arrays and obey numpy broadcasting rules.
```

```
# Here, r is an array instead of a single number:
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'],
                                     3.0 * np.ones(len(objs)))

# use a different subpixel sampling (default is 5; 0 means "exact")
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0,
                                     subpix=0)
```

Error calculation

In the default modes illustrated above, the uncertainty `fluxerr` is not calculated and values of 0 are simply returned. The uncertainty can be flexibly and efficiently calculated, depending on the characteristics of your data. The presence of an `err` or `var` keyword indicates a per-pixel noise, while the presense of a `gain` keyword indicates that the Poisson uncertainty on the total sum should be included. Some illustrative examples:

```
# Specify a per-pixel "background" error and a gain. This is suitable
# when the data have been background subtracted.
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0,
                                     err=bkg.globalrms, gain=1.0)

# Variance can be passed instead of error, with identical results.
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0,
                                     var=bkg.globalrms**2, gain=1.0)

# Error or variance can be arrays, indicating that the background error
# is variable.
bkgrms = bkg.rms()  # array, same shape as data
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0,
                                     err=bkgrms, gain=1.0)

# If your uncertainty array already includes Poisson noise from the object,
# leave gain as None (default):
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0,
                                     err=error_array)

# If your data represent raw counts (it is not background-subtracted),
# set only gain to get the poisson error:
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0,
                                     gain=1.0)
```

The error is calculated as

$$\sigma_F^2 = \sum_i \sigma_i^2 + F/g$$

where the sum is over pixels in the aperture, σ_i is the noise in each pixel, F is the sum in the aperture and g is the gain. The last term is not added if `gain` is `None`.

Masking

Apply a mask (same shape as data). Pixels where the mask is True are "corrected" to the average value within the aperture.

```
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0,
                                     mask=mask)
```

Local background subtraction

The *sum_circle* and *sum_ellipse* functions have options for performing local background subtraction. For example, to subtract the background calculated in an annulus between 6 and 8 pixel radius:

```
flux, fluxerr, flag = sep.sum_circle(data, objs['x'], objs['y'], 3.0,
                                     mask=mask, bkgann=(6., 8.))
```

Pixels in the background annulus are not subsampled and any masked pixels in the annulus are completely igored rather than corrected. The inner and outer radii can also be arrays. The error in the background is included in the reported error.

2.2.1 Equivalent of FLUX_AUTO (e.g., MAG_AUTO) in Source Extractor

This is a two-step process. First we calculate the Kron radius for each object, then we perform elliptical aperture photometry within that radius:

```
kronrad, krflag = sep.kron_radius(data, x, y, a, b, theta, 6.0)
flux, fluxerr, flag = sep.sum_ellipse(data, x, y, a, b, theta, 2.5*kronrad,
                                      subpix=1)
flag |= krflag  # combine flags into 'flag'
```

This specific example is the equilvalent of setting `PHOT_AUTOPARAMS 2.5, 0.0` in Source Extractor (note the 2.5 in the argument to `sep.sum_ellipse`). The second Source Extractor parameter is a minimum diameter. To replicate Source Extractor behavior for non-zero values of the minimum diameter, one would put in logic to use circular aperture photometry if the Kron radius is too small. For example:

```
r_min = 1.75  # minimum diameter = 3.5
use_circle = kronrad * np.sqrt(a * b) < r_min
cflux, cfluxerr, cflag = sep.sum_circle(data, x[use_circle], y[use_circle],
                                        r_min, subpix=1)
flux[use_circle] = cflux
fluxerr[use_circle] = cfluxerr
flag[use_circle] = cflag
```

2.2.2 Equivalent of FLUX_RADIUS in Source Extractor

In Source Extractor, the FLUX_RADIUS parameter gives the radius of a circle enclosing a desired fraction of the total flux. For example, with the setting `PHOT_FLUXFRAC 0.5`, FLUX_RADIUS will give the radius of a circle containing half the “total flux” of the object. For the definition of “total flux”, Source Extractor uses its measurement of FLUX_AUTO, which is taken through an elliptical aperture (see above). Thus, with the setting `PHOT_FLUXFRAC 1.0`, you would find the circle containing the same flux as whatever ellipse Source Extractor used for `FLUX_AUTO`.

Given a previous calculation of `flux` as above, calculate the radius for a flux fraction of 0.5:

```
r, flag = sep.flux_radius(data, objs['x'], objs['y'], 6.*objs['a'], 0.5,
                          normflux=flux, subpix=5)
```

And for multiple flux fractions:

```
r, flag = sep.flux_radius(data, objs['x'], objs['y'], 6.*objs['a'],
                          [0.5, 0.6], normflux=flux, subpix=5)
```

2.2.3 Equivalent of XWIN_IMAGE, YWIN_IMAGE in Source Extractor

Source Extractor’s XWIN_IMAGE, YWIN_IMAGE parameters can be used for more accurate object centroids than the default X_IMAGE, Y_IMAGE. Here, the `winpos` function provides this behavior. To match Source Extractor exactly, the right `sig` parameter (giving a description of the effective width) must be used for each object.

Source Extractor uses `2. / 2.35 * (half-light radius)` where the half-light radius is calculated using `flux_radius` with a fraction of 0.5 and a normalizing flux of `FLUX_AUTO`. The equivalent here is:

```
sig = 2. / 2.35 * r  # r from sep.flux_radius() above, with fluxfrac = 0.5
xwin, ywin, flag = sep.winpos(data, objs['x'], objs['y'], sig)
```

2.2.4 Masking image regions

Create a boolean array with elliptical regions set to True:

```
mask = np.zeros(data.shape, dtype=np.bool)
sep.mask_ellipse(mask, objs['x'], objs['y'], obs['a'], objs['b'],
                 objs['theta'], r=3.)
```

2.3 Reference/API

Background estimation & source detection

sep.Background(data[, mask, maskthresh, bw, ...])	Representation of spatially variable image background and noise.
sep.extract(data, thresh[, err, mask, ...])	Extract sources from an image.

2.3.1 sep.Background

class `sep.`**`Background`**(*data*, *mask=None*, *maskthresh=0.0*, *bw=64*, *bh=64*, *fw=3*, *fh=3*, *fthresh=0.0*)

Representation of spatially variable image background and noise.

Parameters **data** : 2-d `ndarray`

Data array.

mask : 2-d `ndarray`, optional

Mask array, optional

maskthresh : float, optional

Mask threshold. This is the inclusive upper limit on the mask value in order for the corresponding pixel to be unmasked. For boolean arrays, False and True are interpreted as 0 and 1, respectively. Thus, given a threshold of zero, True corresponds to masked and False corresponds to unmasked.

bw, bh : int, optional

Size of background boxes in pixels. Default is 64.

fw, fh : int, optional

Filter width and height in boxes. Default is 3.

fthresh : float, optional

Filter threshold. Default is 0.0.

`__init__`(*data*, *mask=None*, *maskthresh=0.0*, *bw=64*, *bh=64*, *fw=3*, *fh=3*, *fthresh=0.0*)

Methods

`back`([dtype])	Create an array of the background.
`rms`([dtype])	Create an array of the background rms.
`subfrom`(data)	Subtract the background from an existing array.

Attributes

`globalback`	Global background level.
`globalrms`	Global background RMS.

back (*dtype=None*)
Create an array of the background.

Parameters **dtype** : `dtype`, optional

Data type of output array. Default is the dtype of the original data.

Returns **back** : `ndarray`

Array with same dimensions as original data.

globalback
Global background level.

globalrms
Global background RMS.

rms (*dtype=None*)
Create an array of the background rms.

Parameters **dtype** : `dtype`, optional

Data type of output array. Default is the dtype of the original data.

Returns **rms** : `ndarray`

Array with same dimensions as original data.

subfrom (*data*)
Subtract the background from an existing array.

Like `data = data - bkg`, but avoids making a copy of the data.

Parameters **data** : `ndarray`

Input array, which will be updated in-place. Shape must match that of the original image used to measure the background.

2.3.2 sep.extract

sep.**extract** (*data, thresh, err=None, mask=None, minarea=5, filter_kernel=default_kernel, filter_type='matched', deblend_nthresh=32, deblend_cont=0.005, clean=True, clean_param=1.0, segmentation_map=False*)
Extract sources from an image.

Parameters **data** : `ndarray`

Data array (2-d).

thresh : float

Threshold pixel value for detection. If an `err` array is not given, this is interpreted as an absolute threshold. If `err` is given, this is interpreted as a relative threshold: the absolute threshold at pixel (j, i) will be `thresh * err[j, i]`.

err : `ndarray`, optional

Noise array for specifying a pixel-by-pixel detection threshold.

mask : `ndarray`, optional

Mask array. `True` values, or numeric values greater than 0, are considered masked. Masking a pixel is equivalent to setting data to zero and noise (if present) to infinity.

minarea : int, optional

Minimum number of pixels required for an object. Default is 5.

filter_kernel : `ndarray` or None, optional

Filter kernel used for on-the-fly filtering (used to enhance detection). Default is a 3x3 array: [[1,2,1], [2,4,2], [1,2,1]]. Set to `None` to skip convolution.

filter_type : {'matched', 'conv'}, optional

Filter treatment. This affects filtering behavior when a noise array is supplied. `'matched'` (default) accounts for pixel-to-pixel noise in the filter kernel. `'conv'` is simple convolution of the data array, ignoring pixel-to-pixel noise across the kernel. `'matched'` should yield better detection of faint sources in areas of rapidly varying noise (such as found in coadded images made from semi-overlapping exposures). The two options are equivalent when noise is constant.

deblend_nthresh : int, optional

Number of thresholds used for object deblending. Default is 32.

deblend_cont : float, optional

Minimum contrast ratio used for object deblending. Default is 0.005. To entirely disable deblending, set to 1.0.

clean : bool, optional

Perform cleaning? Default is True.

clean_param : float, optional

Cleaning parameter (see SExtractor manual). Default is 1.0.

segmentation_map : bool, optional

If True, also return a "segmentation map" giving the member pixels of each object. Default is False.

Returns objects : `ndarray`

Extracted object parameters (structured array). Available fields are:

- `thresh` (float) Threshold at object location.
- `npix` (int) Number of pixels belonging to the object.
- `tnpix` (int) Number of pixels above threshold (unconvolved data).
- `xmin`, `xmax` (int) Minimum, maximum x coordinates of pixels.
- `ymin`, `ymax` (int) Minimum, maximum y coordinates of pixels.
- `x`, `y` (float) object barycenter (first moments).

- x2, y2, xy (float) Second moments.
- a, b, theta (float) Ellipse parameters.
- cxx, cyy, cxy (float) Alternative ellipse parameters.
- cflux (float) Sum of member pixels in convolved data.
- flux (float) Sum of member pixels in unconvolved data.
- cpeak (float) Peak value in convolved data.
- peak (float) Peak value in unconvolved data.
- xcpeak, ycpeak (int) Coordinate of convolved peak pixel.
- xpeak, ypeak (int) Coordinate of convolved peak pixel.
- flag (int) Extraction flags.

segmap : ndarray, optional

Array of integers with same shape as data. Pixels not belonging to any object have value 0. All pixels belonging to the i-th object (e.g., objects[i]) have value i+1. Only returned if segmentation_map=True.

Aperture photometry

sep.sum_circle(data, x, y, r[, err, var, ...])	Sum data in circular aperture(s).
sep.sum_circann(data, x, y, rin, rout[, ...])	Sum data in circular annular aperture(s).
sep.sum_ellipse(data, x, y, a, b, theta, r)	Sum data in elliptical aperture(s).
sep.sum_ellipann(data, x, y, a, b, theta, ...)	Sum data in elliptical annular aperture(s).

2.3.3 sep.sum_circle

sep.**sum_circle**(*data*, *x*, *y*, *r*, *err=None*, *var=None*, *mask=None*, *maskthresh=0.0*, *bkgann=None*, *gain=None*, *subpix=5*)

Sum data in circular aperture(s).

Parameters **data** : ndarray

2-d array to be summed.

x, y, r : array_like

Center coordinates and radius (radii) of aperture(s). x corresponds to the second ("fast") axis of the input array and y corresponds to the first ("slow") axis. x, y = (0.0, 0.0) corresponds to the center of the first element of the array. These inputs obey numpy broadcasting rules.

err, var : float or ndarray

Error *or* variance (specify at most one).

mask : ndarray, optional

Mask array. If supplied, a given pixel is masked if its value is greater than maskthresh.

maskthresh : float, optional

Threshold for a pixel to be masked. Default is 0.0.

bkgann : tuple, optional

Length 2 tuple giving the inner and outer radius of a "background annulus". If supplied, the background is estimated by averaging unmasked pixels in this annulus. If supplied, the inner and outer radii obey numpy broadcasting rules along with `x`, `y` and `r`.

gain : float, optional

Conversion factor between data array units and poisson counts, used in calculating poisson noise in aperture sum. If `None` (default), do not add poisson noise.

subpix : int, optional

Subpixel sampling factor. If 0, exact overlap is calculated. Default is 5.

Returns **sum** : `ndarray`

The sum of the data array within the aperture.

sumerr : `ndarray`

Error on the sum.

flags : `ndarray`

Integer giving flags. (0 if no flags set.)

2.3.4 sep.sum_circann

`sep.`**`sum_circann`**(*data*, *x*, *y*, *rin*, *rout*, *err=None*, *var=None*, *mask=None*, *maskthresh=0.0*, *gain=None*, *subpix=5*)

Sum data in circular annular aperture(s).

Parameters **data** : `ndarray`

2-d array to be summed.

x, y, rin, rout : array_like

Center coordinates and inner and outer radii of aperture(s). `x` corresponds to the second ("fast") axis of the input array and `y` corresponds to the first ("slow") axis. `x, y = (0.0, 0.0)` corresponds to the center of the first element of the array. These inputs obey numpy broadcasting rules. It is required that `rout >= rin >= 0.0`.

err, var : float or ndarray

Error *or* variance (specify at most one).

mask : `ndarray`, optional

Mask array. If supplied, a given pixel is masked if its value is greater than `maskthresh`.

maskthresh : float, optional

Threshold for a pixel to be masked. Default is `0.0`.

gain : float, optional

Conversion factor between data array units and poisson counts, used in calculating poisson noise in aperture sum. If `None` (default), do not add poisson noise.

subpix : int, optional

Subpixel sampling factor. Default is 5.

Returns **sum** : `ndarray`

The sum of the data array within the aperture.

sumerr : `ndarray`

Error on the sum.

flags : `ndarray`

Integer giving flags. (0 if no flags set.)

2.3.5 sep.sum_ellipse

`sep.`**`sum_ellipse`**(*data*, *x*, *y*, *a*, *b*, *theta*, *r*, *err=None*, *var=None*, *mask=None*, *maskthresh=0.0*, *bkgann=None*, *gain=None*, *subpix=5*)

Sum data in elliptical aperture(s).

Parameters data : `ndarray`

2-d array to be summed.

x, y : array_like

Center coordinates and radius (radii) of aperture(s). `x` corresponds to the second ("fast") axis of the input array and `y` corresponds to the first ("slow") axis. `x, y = (0.0, 0.0)` corresponds to the center of the first element of the array. These inputs obey numpy broadcasting rules.

a, b, theta : array_like

Ellipse parameters. These inputs, along with `x`, `y`, and `r`, obey numpy broadcasting rules. `a` is the semi-major axis, `b` is the semi-minor axis and `theta` is angle in radians between the positive x axis and the major axis. It must be in the range `[-pi/2, pi/2]`. It is also required that `a >= b >= 0.0`.

r : array_like, optional

Scaling factor for the semi-minor and semi-major axes. The actual ellipse used will have semi-major axis `a * r` and semi-minor axis `b * r`. Setting this parameter to a value other than 1.0 is exactly equivalent to scaling both `a` and `b` by the same value. Default is 1.0.

err, var : float or `ndarray`

Error *or* variance (specify at most one).

mask : `ndarray`, optional

Mask array. If supplied, a given pixel is masked if its value is greater than `maskthresh`.

maskthresh : float, optional

Threshold for a pixel to be masked. Default is `0.0`.

bkgann : tuple, optional

Length 2 tuple giving the inner and outer radius of a "background annulus". If supplied, the background is estimated by averaging unmasked pixels in this annulus. If supplied, the inner and outer radii obey numpy broadcasting rules, along with `x`, `y`, and ellipse parameters.

gain : float, optional

Conversion factor between data array units and poisson counts, used in calculating poisson noise in aperture sum. If `None` (default), do not add poisson noise.

subpix : int, optional

Subpixel sampling factor. Default is 5.

Returns sum : `ndarray`

The sum of the data array within the aperture.

sumerr : `ndarray`

Error on the sum.

flags : `ndarray`

Integer giving flags. (0 if no flags set.)

2.3.6 sep.sum_ellipann

`sep.`**`sum_ellipann`**(*data*, *x*, *y*, *a*, *b*, *theta*, *rin*, *rout*, *err=None*, *var=None*, *mask=None*, *maskthresh=0.0*, *gain=None*, *subpix=5*)

Sum data in elliptical annular aperture(s).

Parameters data : `ndarray`

2-d array to be summed.

x, y : array_like

Center coordinates and radius (radii) of aperture(s). `x` corresponds to the second ("fast") axis of the input array and `y` corresponds to the first ("slow") axis. `x, y = (0.0, 0.0)` corresponds to the center of the first element of the array. These inputs obey numpy broadcasting rules.

a, b, theta, rin, rout : array_like

Elliptical annulus parameters. These inputs, along with `x` and `y`, obey numpy broadcasting rules. `a` is the semi-major axis, `b` is the semi-minor axis and `theta` is angle in radians between the positive x axis and the major axis. It must be in the range `[-pi/2, pi/2]`. It is also required that `a >= b >= 0.0` and `rout >= rin >= 0.0`

err, var : float or `ndarray`

Error *or* variance (specify at most one).

mask : `ndarray`, optional

Mask array. If supplied, a given pixel is masked if its value is greater than `maskthresh`.

maskthresh : float, optional

Threshold for a pixel to be masked. Default is `0.0`.

gain : float, optional

Conversion factor between data array units and poisson counts, used in calculating poisson noise in aperture sum. If `None` (default), do not add poisson noise.

subpix : int, optional

Subpixel sampling factor. Default is 5.

Returns sum : `ndarray`

The sum of the data array within the aperture(s).

sumerr : `ndarray`

Error on the sum.

flags : `ndarray`

Integer giving flags. (0 if no flags set.)

Aperture utilities

sep.kron_radius(data, x, y, a, b, theta, r)	Calculate Kron "radius" within an ellipse.
sep.flux_radius(data, x, y, rmax, frac[, ...])	Return radius of a circle enclosing requested fraction of total flux.
sep.winpos(data, xinit, yinit, sig[, mask, ...])	Calculate more accurate object centroids using 'windowed' algorithm.
sep.mask_ellipse(arr, x, y, a, b, theta[, r])	Mask ellipse(s) in an array.
sep.ellipse_axes(cxx, cyy, cxy)	Convert from coefficient ellipse representation to ellipse axes and angle.
sep.ellipse_coeffs(a, b, theta)	Convert from ellipse axes and angle to coefficient representation.

2.3.7 sep.kron_radius

sep.**kron_radius**(*data*, *x*, *y*, *a*, *b*, *theta*, *r*, *mask=None*, *maskthresh=0.0*)

Calculate Kron "radius" within an ellipse.

The Kron radius is given by

$$\sum_i r_i I(r_i) / \sum_i I(r_i)$$

where the sum is over all pixels in the aperture and the radius is given in units of `a` and `b`: `r_i` is the distance to the pixel relative to the distance to the ellipse specified by `a`, `b`, `theta`. Equivalently, after converting the ellipse parameters to their coefficient representation, `r_i` is given by

$$r_i^2 = cxx(x_i - x)^2 + cyy(y_i - y)^2 + cxx(x_i - x)(y_i - y)$$

Parameters **data** : `ndarray`

Data array.

x, y : array_like

Ellipse center(s).

a, b, theta : array_like

Ellipse parameters.

r : array_like

"Radius" of ellipse over which to integrate. If the ellipse extent correponds to second moments of an object, this is the number of "isophotal radii" in Source Extractor parlance. A Fixed value of 6 is used in Source Extractor.

mask : `numpy.ndarray`, optional

An optional mask.

maskthresh : float, optional

Pixels with mask > maskthresh will be ignored.

Returns **kronrad** : array_like

The Kron radius.

flag : array_like

Integer value indicating conditions about the aperture or how many masked pixels it contains.

2.3.8 sep.flux_radius

sep.**flux_radius**(*data*, *x*, *y*, *rmax*, *frac*, *normflux=None*, *mask=None*, *maskthresh=0.0*, *subpix=5*)

Return radius of a circle enclosing requested fraction of total flux.

Parameters **data** : `ndarray`

2-d array to be summed.

x, y : array_like

Center coordinates and radius (radii) of aperture(s). `x` corresponds to the second ("fast") axis of the input array and `y` corresponds to the first ("slow") axis. `x, y = (0.0, 0.0)` corresponds to the center of the first element of the array. Shapes must match.

rmax : array_like

Maximum radius to analyze. Used as normalizing flux if `normflux` is None. Shape must match x and y.

frac : array_like

Requested fraction of light (in range 0 to 1). Can be scalar or array.

normflux : array_like, optional

Normalizing flux for each position. If not given, the sum within `rmax` is used as the normalizing flux. If given, shape must match x, y and rmax.

mask : `ndarray`, optional

Mask array. If supplied, a given pixel is masked if its value is greater than `maskthresh`.

maskthresh : float, optional

Threshold for a pixel to be masked. Default is `0.0`.

subpix : int, optional

Subpixel sampling factor. Default is 5.

Returns **radius** : `ndarray`

The sum of the data array within the aperture(s). Shape is same as `x`, except if `frac` is an array; then the dimension of `frac` will be appended. For example, if `x` and `frac` are both 1-d arrays, the result will be a 2-d array with the trailing dimension corresponding to `frac`.

flags : `ndarray`

Integer giving flags. Same shape as `x`. (0 if no flags set.)

2.3.9 sep.winpos

`sep.`**`winpos`**(*data*, *xinit*, *yinit*, *sig*, *mask=None*, *maskthresh=0.0*, *subpix=11*)

Calculate more accurate object centroids using 'windowed' algorithm.

Starting from the supplied initial center position, an iterative algorithm is used to determine a better object centroid. On each iteration, the centroid is calculated using all pixels within a circular aperture of `4*sig` from the current position, weighting pixel positions by their flux and the amplitude of a 2-d Gaussian with sigma `sig`. Iteration stops when the change in position falls under some threshold or a maximum number of iterations is reached. This is equivalent to `XWIN_IMAGE` and `YWIN_IMAGE` parameters in Source Extractor (for the correct choice of sigma for each object).

Parameters **data** : `ndarray`

Data array.

xinit, yinit : array_like

Initial center(s).

sig : array_like

Gaussian sigma used for weighting pixels. Pixels within a circular aperture of radius 4*sig are included.

mask : `numpy.ndarray`, optional

An optional mask.

maskthresh : float, optional

Pixels with mask > maskthresh will be ignored.

subpix : int

Subpixel sampling used to determine pixel overlap with aperture. 11 is used in Source Extractor. For exact overlap calculation, use 0.

Returns **x, y** : np.ndarray

New x and y position(s).

flag : np.ndarray

Flags.

2.3.10 sep.mask_ellipse

`sep.`**`mask_ellipse`**(*arr*, *x*, *y*, *a*, *b*, *theta*, *r=1.0*)

Mask ellipse(s) in an array.

Set array elements to True (or 1) if they fall within the given ellipse. The `r` keyword can be used to scale the ellipse. Equivalently, after converting `a`, `b`, `theta` to a coefficient ellipse representation (`cxx`, `cyy`, `cxy`), pixels that fulfill the condition

$$cxx(x_i - x)^2 + cyy(y_i - y)^2 + cxx(x_i - x)(y_i - y) < r^2$$

will be masked.

Parameters **arr** : `ndarray`

Input array to be masked. Array is updated in-place.

x, y : array_like

Center of ellipse(s).

a, b, theta : array_like

Parameters defining the extent of the ellipe(s).

r : array_like, optional

Scale factor of ellipse(s). Default is 1.

2.3.11 sep.ellipse_axes

`sep.`**`ellipse_axes`**(*cxx*, *cyy*, *cxy*)

Convert from coefficient ellipse representation to ellipse axes and angle.

Parameters **cxx, cyy, cxy** : array_like

Describes the ellipse(s) `cxx * x**2 + cyy * y**2 + cxy * x * y = 1`

Returns **a, b, theta** : `ndarray`

Ellipse(s) semi-major, semi-minor axes and position angle respectively. Position angle is radians counter clockwise from positive x axis to major axis, and lies in range `(-pi/2, pi/2)`

Raises **ValueError** :

If input parameters do not describe an ellipse.

2.3.12 sep.ellipse_coeffs

`sep.`**`ellipse_coeffs`**(*a*, *b*, *theta*)

Convert from ellipse axes and angle to coefficient representation.

Parameters **a, b, theta** : array_like

Ellipse(s) semi-major, semi-minor axes and position angle respectively. Position angle is radians counter clockwise from positive x axis to major axis, and lies in range `[-pi/2, pi/2]`

Returns **cxx, cyy, cxy** : `ndarray`

Describes the ellipse(s) `cxx * x^2 + cyy * y^2 + cxy * xy = 1`

Low-level utilities

`sep.get_extract_pixstack`()	Get the size in pixels of the internal pixel buffer used in extract().
`sep.set_extract_pixstack`(size)	Set the size in pixels of the internal pixel buffer used in extract().

2.3.13 sep.get_extract_pixstack

`sep.`**`get_extract_pixstack`**()

Get the size in pixels of the internal pixel buffer used in extract().

2.3.14 sep.set_extract_pixstack

`sep.`**`set_extract_pixstack`**(*size*)

Set the size in pixels of the internal pixel buffer used in extract().

The current value can be retrieved with get_extract_pixstack. The initial default is 300000.

Flags

Flag	Description
`sep.OBJ_MERGED`	object is result of deblending
`sep.OBJ_TRUNC`	object is truncated at image boundary
`sep.OBJ_SINGU`	x, y fully correlated in object
`sep.APER_TRUNC`	aperture truncated at image boundary
`sep.APER_HASMASKED`	aperture contains one or more masked pixels
`sep.APER_ALLMASKED`	aperture contains only masked pixels
`sep.APER_NONPOSITIVE`	aperture sum is negative in `kron_radius`

To see if a given flag is set in `flags`:

```
is_merged = (flags & sep.OBJ_MERGED) != 0
```

Note: The coordinate convention in SEP is that (0, 0) corresponds to the center of the first element of the data array. This agrees with the 0-based indexing in Python and C. However, note that this differs from the FITS convention where the center of the first element is at coordinates (1, 1). As Source Extractor deals with FITS files, its outputs follow the FITS convention. Thus, the coordinates from SEP will be offset from Source Extractor coordinates by -1 in x and y.

For complete API documentation, see Reference/API.

CHAPTER 3

License and Citation

The license for SEP is the Lesser GNU Public License (LGPL), granted with the permission from the original author of Source Extractor.

If you use SEP in a publication, please cite the DOI 10.5281/zenodo.15669. The link provides a variety of citation styles and BibTeX export. For example:

```
@misc{kyle_barbary_2015_15669,
  author       = {Kyle Barbary and
                  Kyle Boone and
                  Christoph Deil},
  title        = {sep: v0.3.0},
  month        = feb,
  year         = 2015,
  doi          = {10.5281/zenodo.15669},
  url          = {http://dx.doi.org/10.5281/zenodo.15669}
}
```

You may also wish to cite the original SExtractor paper: Bertin & Arnouts 1996.

Symbols

B

E

F

G

K

M

R

S

W