Morphology Extractor `MEx` Standalone Manual

Vitor Medeiros Sampaio

May 18, 2025

Contents

1 Motivation

Galaxy morphology has long been recognized as a key observable for understanding galaxy evolution. Morphological properties are correlated with a variety of physical characteristics, such as stellar mass, star formation rate, and environment. These connections suggest that morphology is not merely descriptive but encodes the outcome of fundamental processes shaping galaxies across cosmic time.

Despite its importance, the reliable and reproducible measurement of morphology remains a persistent challenge in extragalactic astronomy. Visual classifications, while intuitive and effective at low redshift, are subjective and increasingly impractical in the era of large-scale surveys. Parametric approaches based on profile fitting (e.g., Sérsic or bulge+disk models) require assumptions that may not hold in irregular or interacting systems. Recent deep learning methods offer powerful classification capabilities but typically rely on large training sets and are often viewed as "black boxes", lacking interpretability and fine control.

Non-parametric indices—such as those in the CAS (Concentration, Asymmetry, Smoothness) and MEGG (M20, Entropy, Gini, Gradient Pattern Asymmetry) systems—offer an alternative path: they quantify structure directly from the image, without assuming any particular model. These indices have been shown to correlate well with visual morphology and can be applied across redshift and survey conditions. However, their implementation is not straightforward: results can vary significantly depending on image pre-processing, segmentation masks, and even subtle choices in pixel handling. Moreover, different papers and codes adopt slightly different definitions for the same index, particularly for Asymmetry and Smoothness.

`MEx` (Morphology Extractor) is designed to address these issues. It is a modular, reproducible, and fully configurable Python package for extracting morphological indices from galaxy images. `MEx` includes multiple implementations of each major index from the literature, enabling consistent comparisons. It also emphasizes transparency in pre-processing, user-defined segmentation strategies, and output control—ensuring that morphology measurements can be interpreted and compared robustly across datasets and methods.

This manual documents the key classes and functions provided by `MEx`, and serves as a guide to integrating morphological analysis into your scientific pipeline.

2 Overview

`MEx` (Morphology Extractor) is built around a modular and extensible architecture, designed to facilitate reliable and customizable measurement of non-parametric morphological indices. Its internal structure reflects a clear separation of tasks: each major step in the morphological analysis pipeline is comprised within its own dedicated **class**, and each class provides a suite of specialized methods to perform related operations.

The key modules of `MEx` correspond to distinct phases in the processing of galaxy images:

- **Background estimation** (`BackgroundEstimator`): Multiple strategies for subtracting background light, including flat, edge-based, and SEP-based methods.
- **Object detection** (`ObjectDetector`): Source identification using either `SEP` or external SExtractor execution, with unified output.
- **Image cleaning** (`GalaxyCleaner`): Techniques to remove or replace secondary objects from the image, including flat filling, Gaussian noise replacement, and isophotal interpolation.
- **Photometry and light profile analysis** (`PetrosianCalculator`): Growth curves, Petrosian radius, half-light radius, and other scale-related measurements.

- **Segmentation control** (`SegmentImage`): Definition of which pixels belong to the galaxy, including elliptical cuts, surface brightness limits, and custom masks.

Each class in `MEx` can be used independently or in combination with others, making the toolkit adaptable to different pipelines and levels of user control.

3 Installation and Setup

The `MEx` (Morphology Extractor) package is available on PyPI and can be installed with:

```
pip install morphology-extractor
```

The full source code is hosted on GitHub: `https://github.com/vitorms99/MorphologyExtractor`

Jupyter Notebook: A detailed demonstration of all functionalities is provided in the notebook `Functions_description.ipynb`, located in the `examples/` folder. All images shown in this manual were generated directly from that notebook.

Dependencies: MEx depends on standard scientific Python packages such as `numpy`, `scipy`, `matplotlib`, `scikit-image`, and `astropy`. If you clone the repo manually, you can install the required packages using:

```
pip install -r requirements.txt
```

SExtractor Integration (Optional):

If you intend to use `SExtractor` for object detection (via `detection_mode="sex"`), you must have SExtractor installed and callable using the command:

```
sex
```

The simplest way to install SExtractor with this command available is via `conda-forge`. Run:

```
conda config --add channels conda-forge
conda config --set channel_priority strict
conda install -c conda-forge astromatic-source-extractor
```

This will install SExtractor and register the `sex` command in your environment automatically.

If you install SExtractor manually or if the command is not recognized, you may need to add an alias manually. In that case, add the following lines to your shell configuration (e.g., `.bashrc`, `.zshrc`):

```
export PATH="$PATH:/path/to/sextractor"
alias sex='sextractor'
source ~/.bashrc  # or source ~/.zshrc
```

To verify the installation was successful, run:

```
sex -h
```

The following sections document each class in detail, including its initialization parameters, available methods, expected inputs and outputs, and example usage. Throughout this manual, I use the galaxy shown in Figure 1 to perform comparison between the different pre-processing methods.

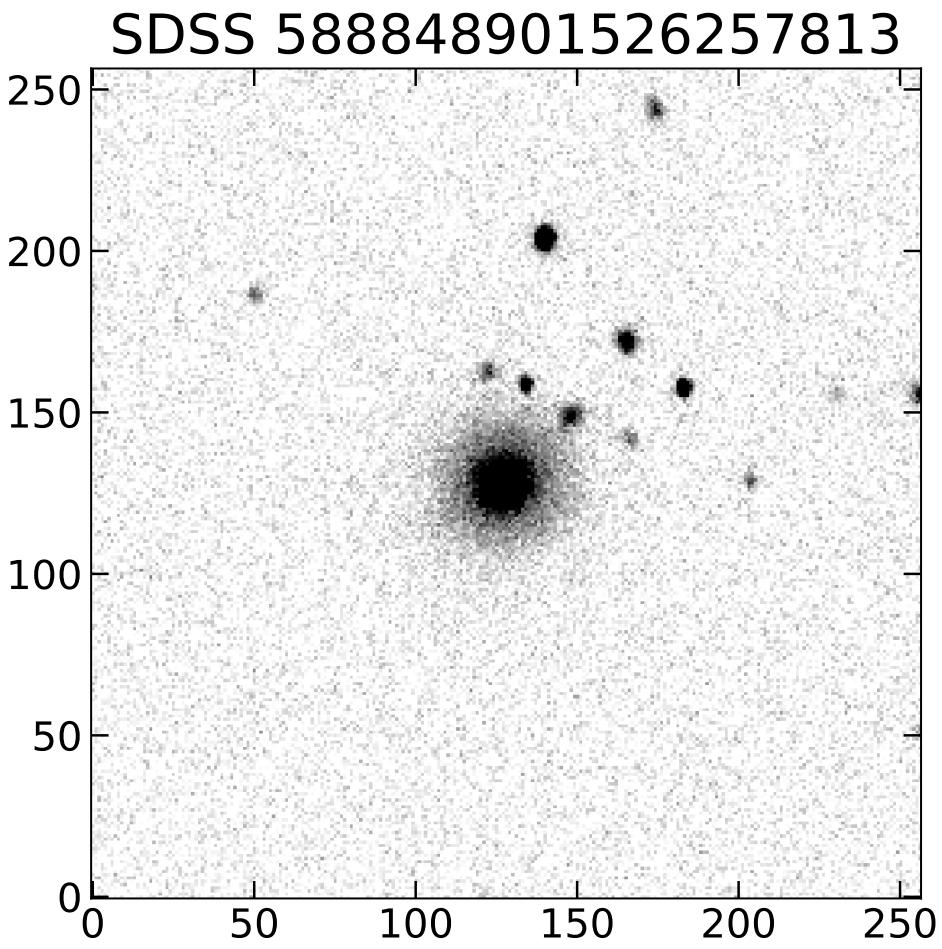

Figure 1: Galaxy example.

4 Background Estimation

The `BackgroundEstimator` class provides flexible and reproducible methods for estimating and subtracting the background in astronomical images. It supports four primary strategies: flat background, image-edge statistics, SEP-based background maps, and externally provided background images. The `BackgroundEstimator` is initialized as follows.

```
from mex.Background_module import BackgroundEstimator
bkg = BackgroundEstimator(galaxy_name,
                          image)
```

- `galaxy_name` (`str`): Object identifier.
- `image` (`numpy.ndarray`): 2D-array representing the input image.

Once initialized, the background subtraction functions return the following variables:

- `bkg_median` (float): Estimated median background level.
- `bkg_std` (float): Estimated standard deviation of background.
- `bkg_image` (ndarray): 2D array representing the background map.
- `galaxy_nobkg` (ndarray): Background-subtracted version of the input image.

The available methods are:

4.1 Flat Background

Method: `flat_background(value, std)`

Assumes a spatially uniform background with constant value across the image.

Config parameters:

- `value` (float): Constant background value to subtract (default = 0).
- `std` (float): Standard deviation to assign to the background (default = 1).

Example call:

```
median, std, bkg_image, galaxy_nobkg = bkg.flat_background(value = 1,
                                                           std = 0.2)
```

4.2 Frame-Based Estimation

Method: `frame_background(image_fraction, sigma_clipping, clipping_threshold)`

Computes the background statistics from the outer edges of the image.

Config parameters:

- `image_fraction` (float): Fraction of image edges to use (default = 0.1).
- `sigma_clipping` (bool): Whether to apply sigma clipping (default = True).
- `clipping_threshold` (float): Clipping threshold in sigma (default = 3).

Example usage:

```
median, std, bkg_image, galaxy_nobkg = bkg.frame_background(image_fraction = 0.1,
                                                            sigma_clipping = True,
                                                            clipping_threshold = 3)
```

4.3 SEP-Based Estimation

Method: `sep_background(bw, bh, fw, fh)`

Uses the `sep.Background()` function to compute a 2D background model via gridding and filtering.

Config parameters:

- `bw, bh` (int): Box width and height for local background estimation (both default to 32).
- `fw, fh` (int): Filter width and height for smoothing (both default to 3).

Example config:

```
median, std, bkg_image, galaxy_nobkg = bkg.frame_background(bw = 32,
                                                            bh = 32,
                                                            fw = 3,
                                                            fh = 3)
```

4.4 External Background Image

Method: `load_background()`

Loads a background image from a user-provided FITS file. This is useful when using custom background models generated externally (e.g., from pipelines or simulations).

Config parameters:

- `bkg_file` (optional): Full filename of the background FITS file. If not provided, a filename will be constructed automatically.
- `bkg_image_path` (str): Directory containing the FITS background image.
- `bkg_image_prefix` (str): Optional prefix added to the filename.
- `bkg_image_sufix` (str): Optional suffix added to the filename (before the `.fits` extension).
- `bkg_image_HDU` (int): FITS HDU index to load data from (default = 0).

Automatic filename construction:

If `bkg_file` is not provided, the background image is assumed to follow the naming convention:

bkg_image_path/bkg_image_prefix + galaxy_name + bkg_image_sufix + ".fits"

That is, the filename is assembled using the `galaxy_name` passed during class initialization. This allows seamless per-object background handling without hard-coding paths.

Example config:

```
median, std, bkg_image, galaxy_nobkg = bkg.load_background(bkg_image_path = "./backgrounds"
                                                           bkg_image_prefix = "bkg_",
                                                           bkg_image_sufix = "_final",
                                                           bkg_image_HDU = 0)
```

This will attempt to load a file named:

`./backgrounds/bkg_<galaxy_name>_final.fits`

and select the HDU[0] data for bkg estimation.

4.5 Methods comparison

Figure 2 shows a comparison between the different background subtraction methods.

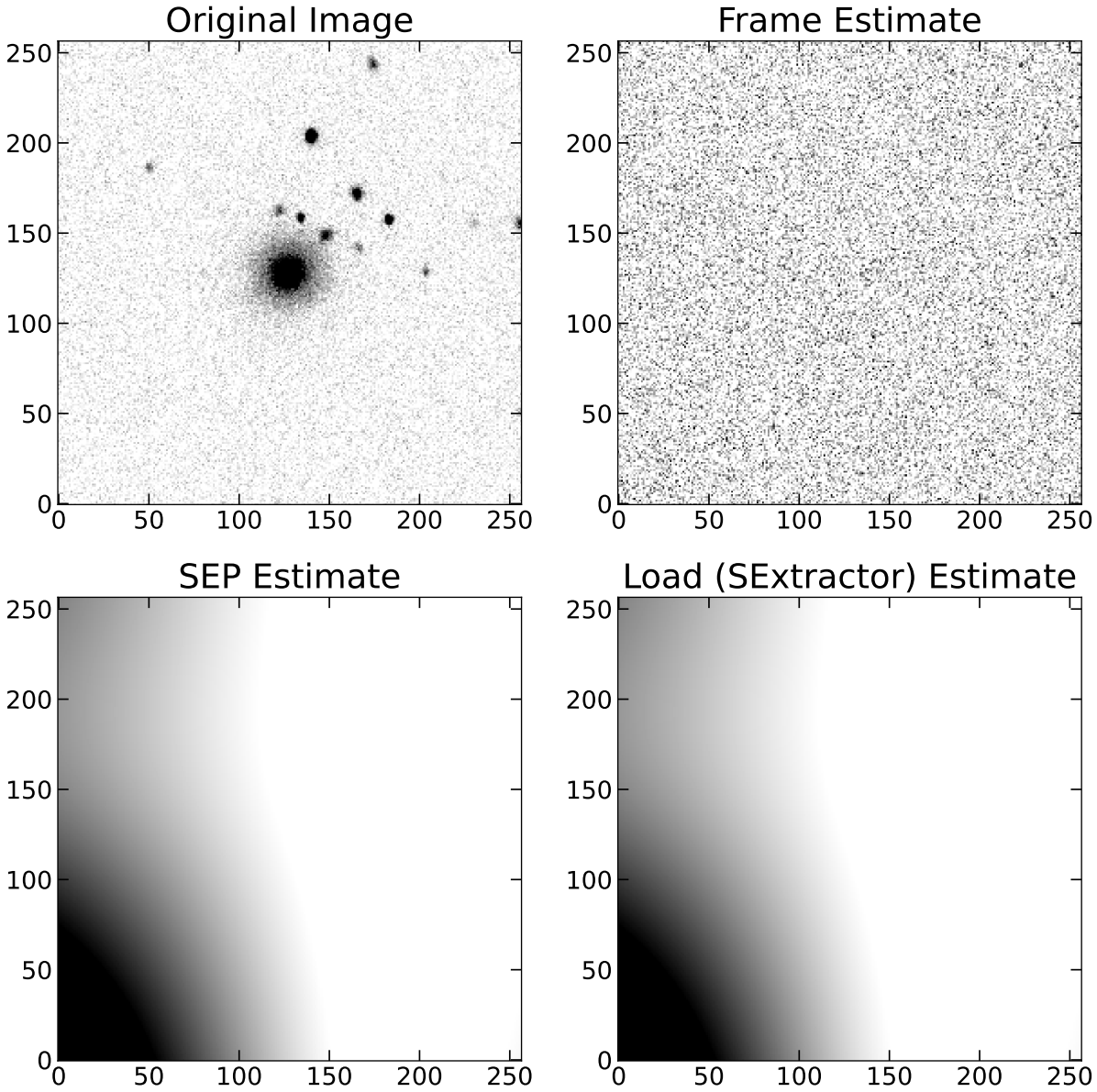

Figure 2: Comparison between the different background subtraction methods.

5 Object Detection

`ObjectDetector` is responsible for identifying sources in astronomical images and generating corresponding segmentation maps. It supports both native `SEP`-based detection and external `SExtractor` execution, offering flexible integration with existing pipelines.

```
from mex.Detection_module import ObjectDetector
detector = ObjectDetector(galaxy_name,
                          image)
```

- `galaxy_name` (`str`): Galaxy identifier.
- `image` (`numpy.ndarray`): 2D array containing the background-subtracted galaxy image.

5.1 SEP-Based Detection

Method: `sep_detector()`

Uses the `sep.extract()` function to detect sources in the image.

Config parameters:

- `thresh` (float): Detection threshold in units of `bkg_std`.
- `minarea` (int): Minimum number of connected pixels above threshold (default = 10).
- `deblend_nthresh` (int): Number of deblending thresholds (default = 32).
- `deblend_cont` (float): Minimum contrast ratio for deblending (default = 0.005).
- `filter_type` (str): Filtering mode ("matched" or "conv"), default to `"matched"`.

Example config:

```
objectes_detected, segmentation_map = detector.sep_detector(thresh = 1.5,
                                                            minarea = 10,
                                                            deblend_nthresh = 32,
                                                            deblend_cont = 0.005,
                                                            filter_type = "matched")
```

5.2 SExtractor-Based Detection

Method: `sex_detector()`

Executes `SExtractor` as an external command-line program. A temporary FITS image is saved, passed to SExtractor, and then cleaned after catalog and segmentation map extraction.

Config parameters:

- `sex_files_folder` (str): Folder containing the `.sex` configuration file.
- `default_file` (str): Name of the SExtractor configuration file (e.g., `default.sex`).
- `sex_output_folder` (str): Folder to store temporary output files.
- `sex_keywords` (dict): Additional key-value parameters passed to SExtractor.
- `clean_sex_output` (bool): If `True`, deletes temporary files after detection (default = True).

```
### You can include any sextractor parameter in the
### "sex_keywords" dictionary, to iteratively change the value
sex_keywords = {"DETECT_MINAREA": 10,
                "DETECT_THRESH": 1,
                "VERBOSE_TYPE": "QUIET"}

objects, segmentation = detector.sex_detector(sex_folder = "./sextractor_files/",
                                              sex_default = "default.sex",
                                              sex_keywords = sex_keywords,
                                              sex_output_folder = "./",
                                              clean_up = True)

```

5.3 Methods comparison

In Figure 3, I show a comparison between the results running SExtractor and SEP exactly with the same setup.

6 Image cleaning

`GalaxyCleaner` is responsible for removing secondary sources from galaxy images and filling the affected pixels with appropriate background or interpolated values. This step is essential to prevent contamination of non-parametric indices by foreground stars, nearby galaxies, or segmentation artifacts.

```
from mex.Cleaning_module import GalaxyCleaner

cleaner = GalaxyCleaner(image,
                        segmentation)
```

- `image` (`ndarray`): 2D array of the target galaxy image.
- `segmentation` (`ndarray`): Segmentation map where each pixel is labeled by object ID.

Upon instantiation, the class determines the ID of the central object based on the segmentation value at the center of the image. This ID is then used to preserve only the main galaxy during cleaning.

6.1 Flat Replacement

Method: `flat_filler(median)`

Pixels with segmentation_map $\neq$ main ID and segmentation_map $\neq 0$ are replaced by the scalar `median`.

Example usage:

```
galaxy_clean = cleaner.flag_filler(median = 0)
```

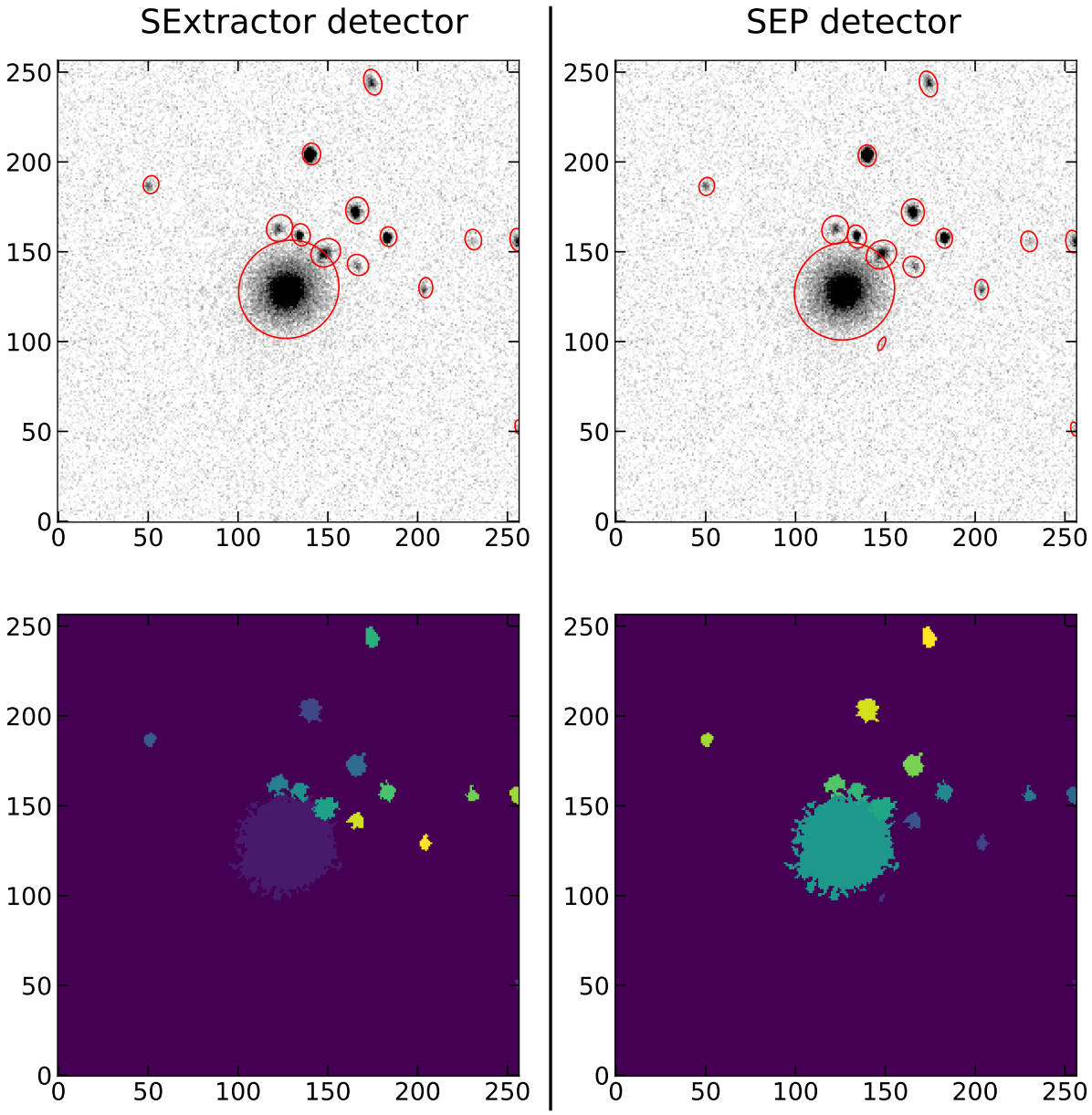

Figure 3: Comparison between objects detected properties and segmentation mask of the SExtractor and SEP methods.

6.2 Gaussian Replacement

Method: `gaussian_filler(mean, std)`

Fills contaminated regions with values drawn from a Gaussian distribution with mean μ and standard deviation σ provided as input (it is usually assumed $\mu = 0$ and σ_{bkg} if the image is already background subtracted). Replacement pixels are drawn from $\mathcal{N}(\mu, \sigma)$.

Example usage:

```
galaxy_clean = cleaner.gaussian_filler(mean = 0,
                                       std = 7)
```

6.3 Elliptical Isophotal Interpolation

Method: `isophotes_filler(theta)`

Performs elliptical interpolation of missing pixels using the light distribution of the main galaxy along elliptical annuli.

Requirements:

- `theta` (float): Position angle of the galaxy (in radians) must be provided.

Example usage:

```
galaxy_clean = cleaner.isophote_filler(theta = np.pi/6)
```

6.4 Methods comparison

In Figure 4, I show the result for the different cleaning (removal of secondary objects) methods.

7 Light Profile and Characteristic Radii

`PetrosianCalculator` computes key radial metrics for galaxy morphology, including the Petrosian radius (R_p), growth curves, and fractional light radii such as R_{50}. These metrics are used in concentration measurements and for defining consistent segmentation apertures. The initialization is as follows:

```
from mex.Petrosian_module import PetrosianCalculator

petro_rad = PetrosianCalculator(galaxy_image,
                                x, y, a, b, theta),
```

where

- `galaxy_image` (`ndarray`): 2D image of the galaxy.
- `x, y` (float): Galaxy centroid coordinates.
- `a, b` (float): Semi-major and semi-minor axes.
- `theta` (float): Position angle of the galaxy (in radians).

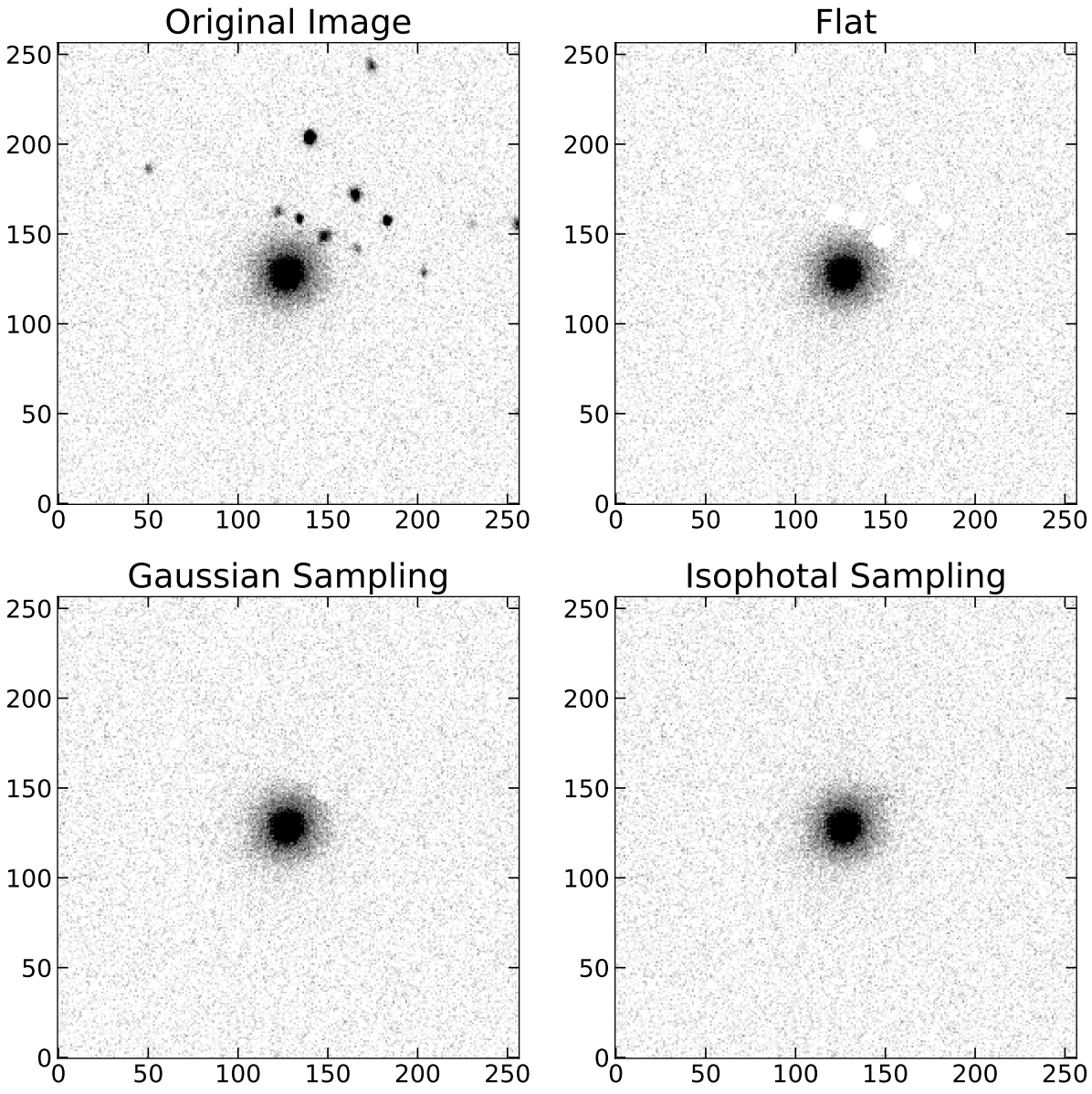

Figure 4: Comparison between the different cleaning methods.

7.1 Petrosian Radius Estimation

Method: `calculate_petrosian_radius()`

This method computes the Petrosian radius R_p, defined as the radius at which the ratio $\eta(r)$ between the local surface brightness in an elliptical annulus and the mean surface brightness within r equals a threshold value η_{thresh}, typically 0.2.

$$\eta(r) = \frac{\mu_{\text{annulus}}(r)}{\mu_{\text{enclosed}}(r)} = \frac{F(0.8r < r < 1.25r)/A_{\text{ann}}}{F(< r)/A(< r)} \tag{1}$$

The method supports two operating modes:

- **Optimized mode:** Uses bisection method to define where the $\eta(r)$ curve crosses η_{thresh}, being more computationally efficient;
- **Standard mode:** Returns the closest radius r where $\eta(r) \approx \eta_{\text{thresh}}$, after calculating the eta profile for the entire image extension.

Parameters:

- `rp_thresh` (float): Threshold η value for defining R_p (default = 0.2).
- `aperture` (str): Shape of apertures, either `"elliptical"` (default) or `"circular"`.
- `optimize_rp` (bool): Whether to use bissection method to avoid unnecessary calculations.
- `interpolate_order` (int): Polynomial order used for spline interpolation (typically 3).
- `Naround` (int): Number of neighboring points used for fitting around the threshold.
- `rp_step` (float): Step size in units of semi-major axis a for radial sampling.

Returns:

- `eta` (ndarray): Array of η values as a function of radius.
- `growth_curve` (ndarray): Cumulative flux as a function of radius.
- `radii` (ndarray): Radii sampled during the calculation.
- `rp` (float): Estimated Petrosian radius.
- `eta_flag` (int): Warning flag (1 if eta never reaches the desired threshold).

Note: Internally, the method relies on aperture photometry via `sep.sum_ellipse()` using subpixel sampling (`subpix = 100`) to ensure reliable radial profiles.

Example usage:

```
eta, gc, rad, rp, flag = petro_rad.calculate_petrosian_radius(rp_thresh = 0.2,
                                                              aperture = "elliptical",
                                                              optimize_rp = True,
                                                              interpolate_order = 3,
                                                              Naround = 3,
                                                              rp_step = 0.05)
```

7.2 Fractional Light Radius Estimation

Method: `calculate_fractional_radius(fraction = 0.5, sampling = 0.05, aperture = "elliptical")`

This method computes the radius at which a specified fraction of the total light of the galaxy is enclosed. The default behavior estimates the half-light radius (R_{50}), but arbitrary fractions (e.g., R_{80}) can also be requested.

Definition:

$$F(< R_f) = f \cdot F_{\text{total}}, \quad \text{with } f \in (0, 1) \tag{2}$$

where $F(< R_f)$ is the cumulative flux inside radius R_f, and F_{total} is the total flux enclosed within an extended elliptical aperture (typically $8a$).

Parameters:

- `fraction` (float): Desired light fraction to enclose (default = 0.5).
- `sampling` (float): Step size, in terms of semi-major axis, for radial sampling (in pixels, default = 0.05).
- `aperture` (str): Aperture shape, either `"elliptical"` (default) or `"circular"`.

Returns:

- `r_frac` (float): Estimated fractional light radius (in units of semi-major axis a).
- `cum_flux` (ndarray): Cumulative flux profile over sampled radii.
- `sma_values` (ndarray): Semi-major axis values corresponding to each flux measurement.

Important to highlight that:

- Fluxes are integrated using `sep.sum_ellipse()` over elliptical annuli centered at `(x, y)` and oriented by `theta`.
- To ensure robustness, the flux array is NaN-cleaned and interpolated linearly to locate the radius corresponding to the target light fraction.
- The search range extends from 1 pixel up to $8a$.

Example:

```
r50, cumflux, sma = petro_rad.calculate_fractional_radius(fraction=0.5)
```

7.3 Method Comparison

In Figure 5, I show the η profile and growth curve for different setups.

8 Segmentation

The `SegmentImage` class is responsible for defining the set of pixels associated with the main galaxy. This segmentation step is essential to ensure that non-parametric indices are calculated over consistent and physically meaningful regions. It can be initialized as:

```
from mex.Segmentation_module import SegmentImage

segmenter = SegmentImage(image, segmentation, rp, x, y, a, b, theta)
```

Parameters:

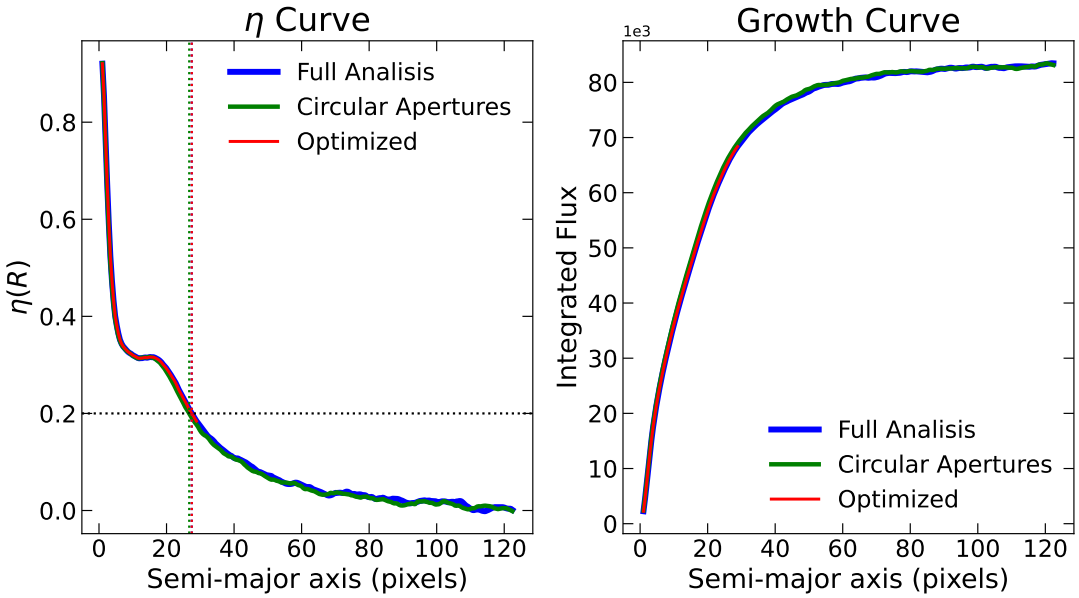

Figure 5: Comparison between the different setups to measure the η profile and growth curve. Irrespective of the method, all result in consistent Petrosian radius estimates.

- `image` (`ndarray`): 2D input image.
- `segmentation` (`ndarray`): Initial segmentation map (e.g., from detection step).
- `rp` (float): characteristic radius used in segmentation limits – it can be any radii, despite the variable name suggesting the use of petrosian radius.
- `x, y` (float): Central coordinates of the main object.
- `a, b` (float): Semi-major and semi-minor axes of the galaxy.
- `theta` (float): Orientation angle (radians).

Upon initialization, the main object is identified from the segmentation label at the image center. If this pixel is not associated with any object (label = 0), a `ValueError` is raised.

8.1 Original Segmentation

Method: `_get_original()`

This mode retains only pixels assigned to the main object in the original segmentation map.

Usage:

```
segmentation_mask = segmenter.get_original()
```

8.2 Limit by distance from central coordinates

Method: `_limit_to_ellipse(k_segmentation)`

This method limits the segmentation region to an ellipse centered on the galaxy, with semi-axes scaled using the characteristic radius provided in the class initialization:

$$a_{\text{new}} = k \cdot rp, \quad b_{\text{new}} = a_{\text{new}} \cdot \frac{b}{a} \tag{3}$$

Usage:

```
segmentation_mask = segmenter._limit_to_ellipse(k_segmentation = 1.5)
```

Orientation is determined by `theta`. Useful to ensure consistent scale definitions across the sample.

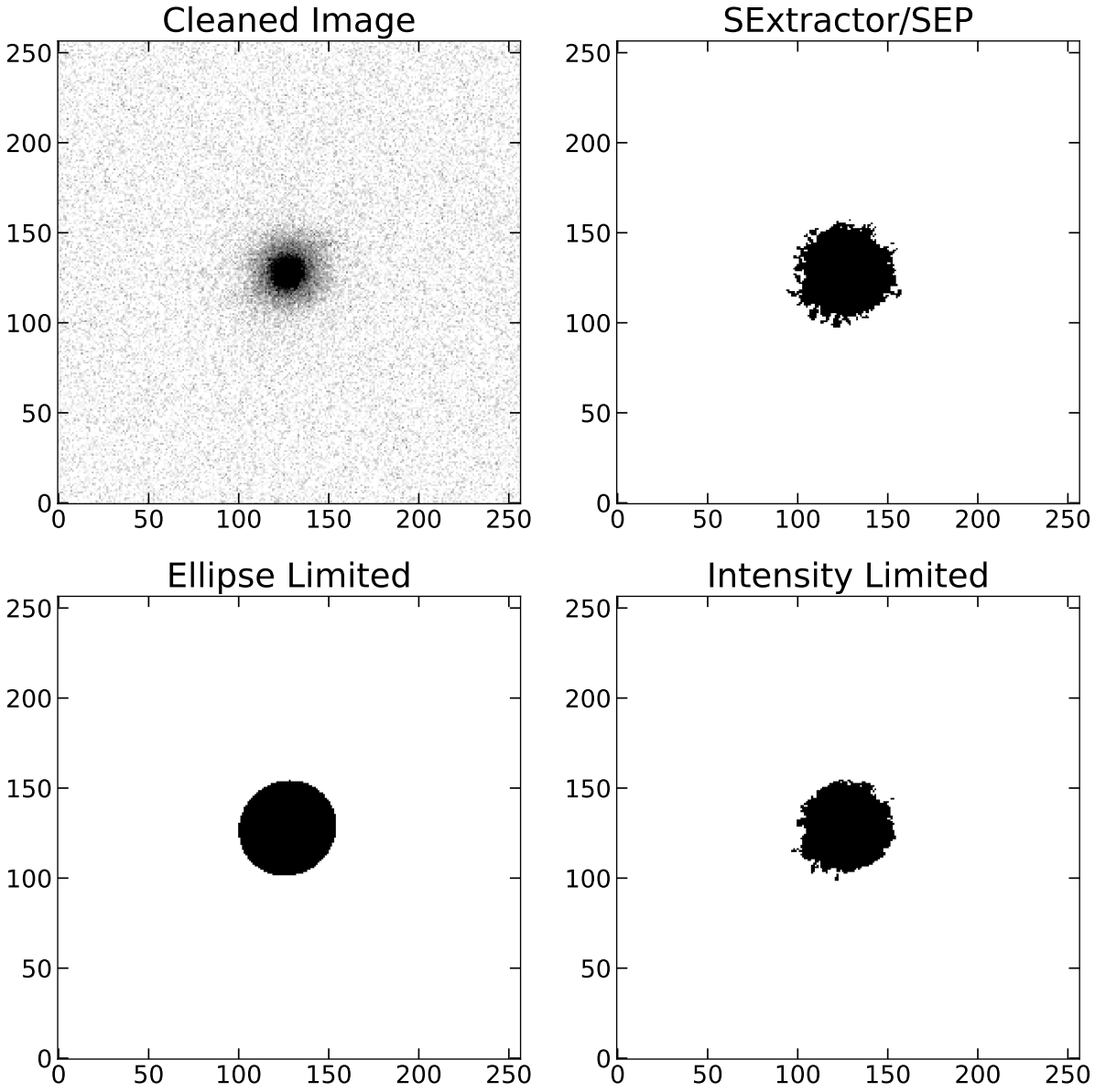

Figure 6: Comparison between different segmentation methods.

8.3 Surface Brightness Thresholding

Method: `_limit_to_intensity(k_segmentation)`

This method refines the segmentation mask by applying a surface brightness threshold derived from the $k \cdot rp$ region. The threshold μ_p is computed as:

$$\mu_p = \frac{F(1.1 \cdot k \cdot rp) - F(0.9 \cdot k \cdot rp)}{A(1.1 \cdot k \cdot rp) - A(0.9 \cdot k \cdot rp)} \tag{4}$$

Important: The image is first smoothed using a box kernel with size $\sim rp/5$, to slightly enhance signal to noise, before selection. Then, only pixels above μ_p and part of the main segmentation label are retained.

Usage:

```
segmentation_mask = segmenter._limit_to_intensity(k_segmentation = 1.5)
```

8.4 Method comparison

In Figure 6, I show the comparison between different segmentation methods. In the two lower panels, I used k_segmentation = 1.

9 CAS-System: Concentration + Asymmetry + Smoothness

The CAS system **REF** has become one of the most widely adopted frameworks for quantifying galaxy morphology using three complementary structural descriptors: **Concentration (C)**, **Asymmetry (A)**, and **Smoothness (S)**. These parameters are particularly useful for identifying and classifying galaxy morphologies in large surveys, and have been shown to correlate with key physical properties such as star formation rate, merger history, and stellar mass.

In the `MEx` package, we implement a modular and extensible framework for computing all CAS indices. Each metric is encapsulated in its own class, with clearly separated methods for calculating, optimizing, and visualizing the result. Notably, MEx supports multiple distinct implementations of both **Asymmetry** and **Smoothness**, reflecting methodological diversity in the literature. For asymmetry, we include the original definition from **REF**, a pixel-normalized formulation introduced by Sampaio et al. (in prep.), and a correlation-based estimator proposed by **REF**, and adopted in **REF**. Similarly, for smoothness, MEx supports the Conselice-style residual approach, the normalized residual method from Sampaio et al. (in prep.), and the correlation-based smoothness proposed by Barchi et al (2020).

This section provides a detailed description of each class and method used to compute CAS indices in `MEx`, including input requirements, configuration options, return values, and visual diagnostics.

9.1 Generating a noise map

Non-parametric estimators such as Asymmetry and Smoothness require the construction of a control image that contains only background noise. This is crucial to properly account for residual signal in empty regions, and to avoid biasing the measurements in the presence of sky structure or faint undetected objects. The `MEx` package provides a built-in procedure for estimating such a noise image by extracting a clean patch from the corners of the original image — a strategy that has proven effective in preserving realistic noise properties. This functionality is implemented through the utility function:

```
from mex.Utils_module import extract_cutouts

clean_m, segmented_m, rng, noise_m, best_c = extract_cutouts(galaxy_clean_iso,
                                                             segmentation_ellipse,
                                                             expansion_factor=1.2,
                                                             estimate_noise=True)
```

When `estimate_noise=True`, the function searches for the cleanest (least contaminated) corner among the four image quadrants and extracts a patch with the same shape as the galaxy cutout. If some contamination from nearby sources is still present, it replaces the contaminated

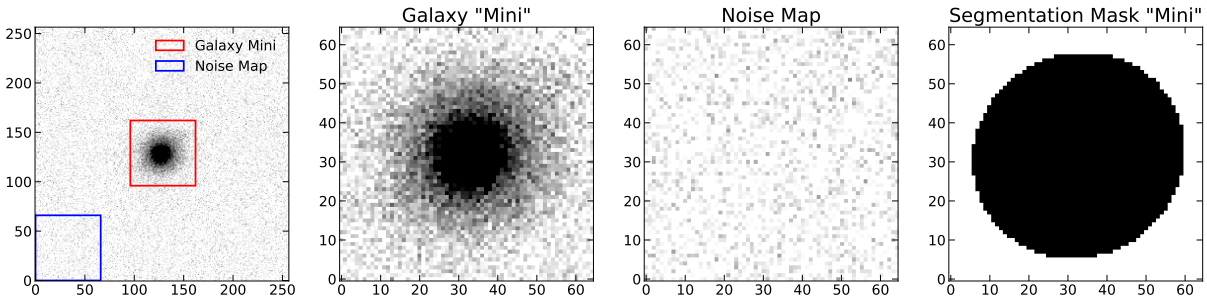

Figure 7: Example of noise procedure.

pixels with values drawn from a normal distribution modeled on the remaining clean background. The process ensures that the noise image is statistically consistent with the true background, but free from contaminating sources signal.

The selection process proceeds as follows:

- All four corners of the image are tested for usable cutouts.
- Each patch is scanned for contamination using the segmentation map.
- The patch with the fewest galaxy pixels is selected as the noise source.
- If any contaminating pixels remain, they are replaced by Gaussian noise:

$$N(x, y) \sim \mathcal{N}(0, \sigma_{\text{bkg}})$$

The returned outputs include the extracted image cutout, segmentation mask, noise image (if requested), and the name of the selected corner (e.g., "top_right"). This mechanism enables robust estimation of noise asymmetry and smoothness following the prescriptions in **REF**, **REF**, and Sampaio et al. (in prep.). I highlight in Figure 7 the procedure to generate the noise map.

9.2 Concentration (C)

The `Concentration` class calculates the concentration index of a galaxy, defined as the ratio between the radii that enclose two specified fractions of the total light. This class supports two main definitions:

- The classical **Conselice (2003)** definition:

$$C = 5 \log_{10} \left(\frac{R_{80}}{R_{20}} \right)$$

- A version of the same ratio used by **Barchi et al. (2020)** (the only difference is the multiplicative factor):

$$C = \log_{10} \left(\frac{R_{80}}{R_{20}} \right)$$

```
from mex.Metrics_module import Concentration
conc = Concentration(image)
```

Parameters:

- `image` (`ndarray`): 2D array of the background-subtracted galaxy image.

9.2.1 Calculate concentration index

Method: `get_concentration()`

Main routine to compute the concentration index using the selected method.

```
C, rinner, routter = conc.get_concentration(x, y, a, b, theta,
                                            method="conselice",
                                            f_inner=0.2,
                                            f_outter=0.8,
                                            rmax=32,
                                            sampling_step=1,
                                            Naround=3,
                                            interp_order=3)
```

Parameters:

- `x, y` (float): central coordinates of the main object.
- `a, b, theta` (float): parameters of the best fitting ellipse of the main object.
- `method` (str): Either `"conselice"` or `"barchi"`.
- `f_inner`, `f_outter` (float): Light fractions for inner/outer radii (defaults: 0.2 and 0.8).
- `rmax` (float): Maximum radius to sample (default: $8a$).
- `sampling_step` (float): Semi-major axis step in pixels.
- `Naround` (integer): Number of points around closest point to interpolate.
- `interp_order` (integer): order of interpolation to get radius values at the exact fraction.

Returns:

- `c` (float): Concentration index.
- `rinner`, `routter` (float): Light radii enclosing `f_inner` and `f_outter`.

9.2.2 Sanity check plot

Method: `plot_growth_curve()`

Plots the light growth curve and visually marks the inner and outer radii.

```
conc.plot_growth_curve(x, y, a, b, theta,
                       rmax=None,
                       sampling_step=1,
                       f_inner=0.2,
                       f_outter=0.8,
                       Naround=2,
                       interp_order=3)
```

Description: Draws the normalized growth curve and overlays vertical/horizontal lines indicating the radii and flux fractions used in the concentration calculation, as shown in Figure 8.

Example:

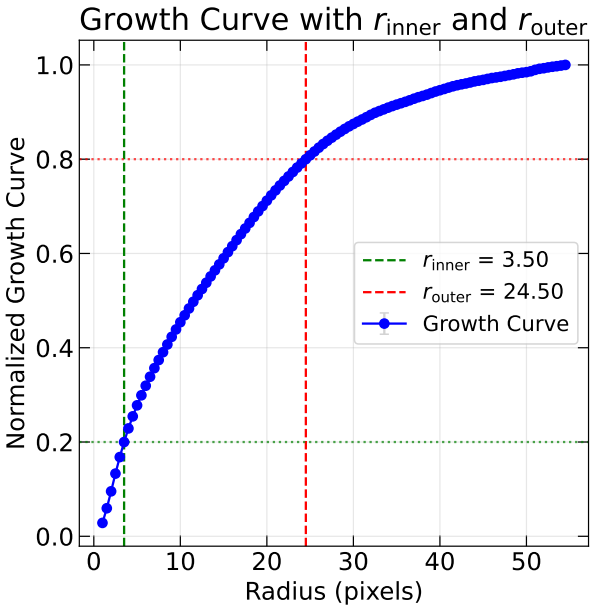

Figure 8: Normalized growth curve. Green and red dashed lines show the radii containing the inner and outer flux fraction.

```
conc = Concentration(image)
c, r20, r80 = conc.get_concentration(x, y, a, b, theta,
                                     method="barchi")
conc.plot_growth_curve(x, y, a, b, theta)
```

9.2.3 Auxiliary functions

Method: `get_growth_curve()`

Computes the cumulative light growth curve using elliptical apertures centered on the galaxy.

```
radii, growth_curve, growth_err = conc.get_growth_curve(x, y, a, b, theta,
                                                        rmax=None,
                                                        sampling_step=1)
```

Returns:

- `radii` (`ndarray`): Sampled semi-major axis values.
- `growth_curve` (`ndarray`): Normalized cumulative flux.
- `growth_err` (`ndarray`): Estimated error on cumulative flux.

Method: `get_radius()`

Interpolates the radius at which a given light fraction is enclosed.

```
r_i = conc.get_radius(radius, curve, fraction=0.5, Naround=3, interp_order=3)
```

Returns:

- `r_i` (float): Radius corresponding to the given light fraction.

9.3 Asymmetry (A)

The `Asymmetry` class quantifies the rotational asymmetry of a galaxy using three distinct methods available in the literature:

1. **Conselice-style asymmetry** based on the pixel-wise difference between the image and its 180° rotation.
2. **Sampaio-style asymmetry** using improved pixel-wise comparison in normalization.
3. **Barchi-style asymmetry** based on $1 - r$ where r is the correlation coefficient between the original and rotated image.

All methods support segmentation masking, center optimization, optional noise correction, and visual diagnostics.

```
from mex.Metrics_module import Asymmetry

asy = Asymmetry(image, angle=180, segmentation=None, noise=None)
```

Parameters:

- `image` (`ndarray`): 2D galaxy image.
- `angle` (float): Rotation angle in degrees (default = 180).
- `segmentation` (`ndarray`, optional): Binary segmentation mask (default = entire image).
- `noise` (`ndarray`, optional): 2D noise-only image for noise asymmetry subtraction.

9.3.1 Conselice Asymmetry

Method: `get_conselice_asymmetry(method='absolute', pixel_comparison='equal', max_iter=50)`

Computes the residual-based asymmetry by subtracting the rotated image from the original. Minimization over small center displacements is performed iteratively.

```
A_f, A_g, A_n, center_g, center_n, n_g, n_n = asy.get_conselice_asymmetry(method='absolute'
                                                                          pixel_comparison=
                                                                          max_iter=50)
```

Parameters:

- `method` (str): `"absolute"` or `"rms"` residual.
- `pixel_comparison` (str): `"equal"` (intersection) or `"simple"` (union) for pixel-wise comparison.
- `max_iter` (int): Maximum iterations for center optimization.

Returns:

- `A_f`: Final asymmetry ($A_{\text{gal}} - A_{\text{noise}}$)
- `A_g`: Minimized asymmetry of galaxy
- `A_n`: Background noise asymmetry (if provided)
- `center_g`, `center_n`: Optimal centers for galaxy and noise, respectively.
- `n_g`, `n_n`: Number of iteration until reaching minima for galaxy and noise, respectively.

9.3.2 Sampaio Normalized Asymmetry

Method: `get_sampaio_asymmetry(method='absolute', pixel_comparison='equal', max_iter=50)`

```
A_f, A_g, A_n, center_g, center_n, n_g, n_n = asy.get_sampaio_asymmetry(method='absolute',
                                                                       pixel_comparison='e
                                                                       max_iter=50)
```

Normalizes pixel-wise differences by the flux of the original image to take into account relative differences:

$$A = \frac{1}{2N} \sum \left| \frac{I - R}{I} \right|$$

Returns and inputs: Same as Conselice method, but follows aforementioned equation.

9.3.3 Barchi Correlation Asymmetry

Method: `get_barchi_asymmetry(corr_type='pearson', pixel_comparison='equal', max_iter=50)`

```
A_b, r_max, center, niter = asy.get_barchi_asymmetry(corr_type='spearman',
                                                     pixel_comparison='equal',
                                                     max_iter=50)
```

Computes:

$$A = 1 - r(I, R)$$

using Pearson or Spearman correlation. Center is optimized by maximizing r.

Parameters:

- `corr_type` (str): `"pearson"` or `"spearman"` for correlation computation.
- `pixel_comparison` (str): `"equal"` (intersection) or `"simple"` (union) for correlation computation.
- `max_iter` (int): Maximum iterations for center optimization.

Returns:

- `A_b` (float): Final asymmetry value.
- `r` (float): Maximum correlation coefficient.
- `center` (tuple): Optimal rotation center.
- `niter` (integer): Number of iterations until reaching minimum asymmetry.

9.3.4 Sanity check plots

Method: `plot_asymmetry_comparison()`

Displays side-by-side images of original, rotated, and residual maps. If noise is provided, it shows the corresponding maps for the noise image as well. An example is shown in Figure 9. **I highlight that the subtraction between the original and rotated images are not done using the center that minimized asymmetry. It is simply a subtraction between the provided image and its rotated version.**

Example:

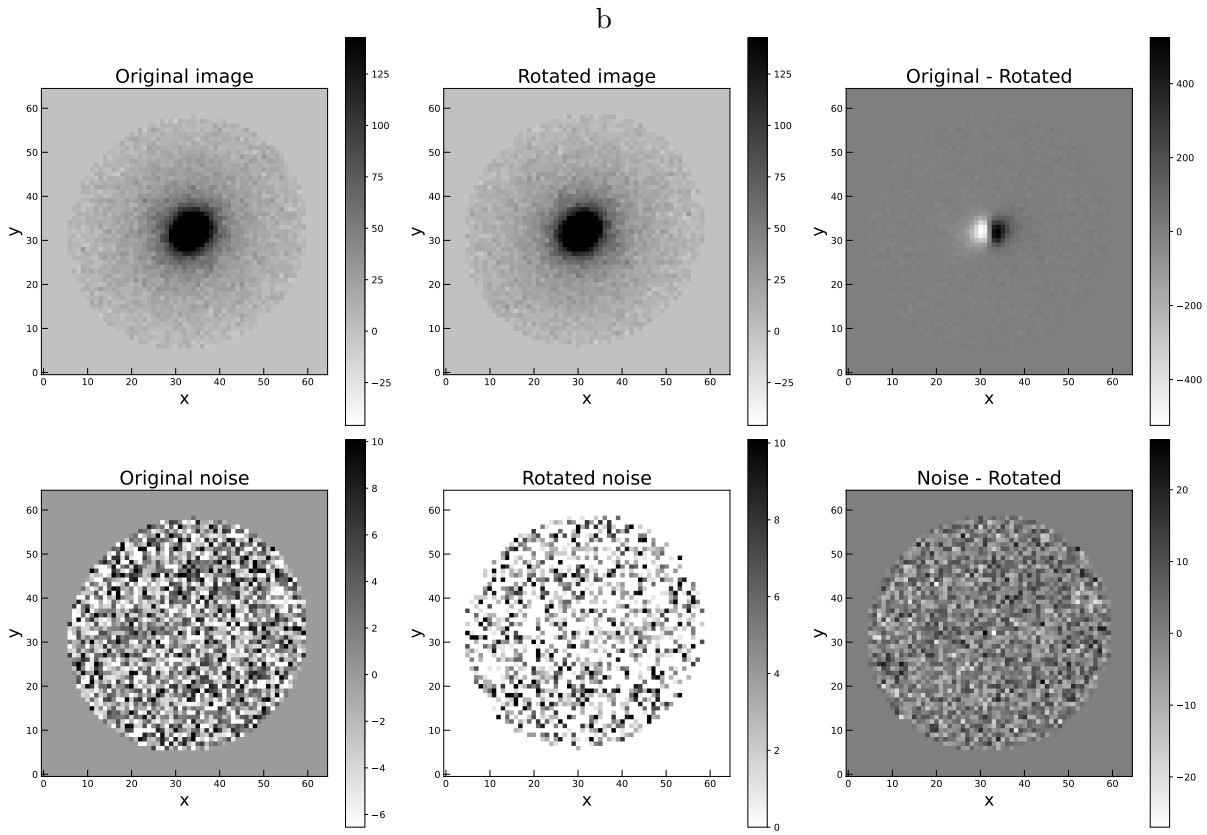

Figure 9: Comparison between original and rotated images. Please note that comparison images is done by subtracting the provided and rotated images with respect to the center of the image, and not the center that minimizes asymmetry.

```
asy.plot_asymmetry_comparison()
```

Method: `plot_asymmetry_scatter(comparison='equal')`

Shows scatter plot of pixel values before and after rotation for both galaxy and (if provided) noise image, as shown in Figure 10. Again, **I highlight that the subtraction between the original and rotated images are not done using the center that minimized asymmetry.**

9.4 Smoothness

The `Smoothness` class measures the small-scale structure or "clumpiness" in a galaxy image by comparing it with a smoothed version of itself. It implements three distinct formulations:

1. **Conselice-style smoothness** based on residual flux between the original and smoothed image.
2. **Sampaio-style smoothness** (in prep.), using pixel-wise normalization.
3. **Barchi-style correlation smoothness** computed as $1 - \rho$ where ρ is the correlation between original and smoothed image.

b

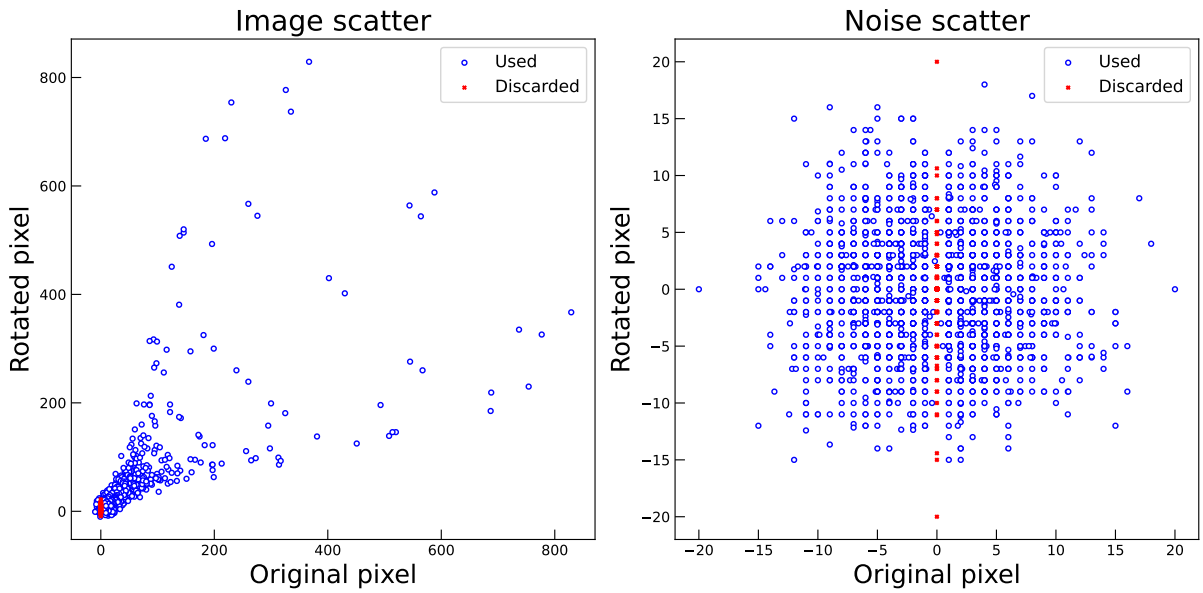

Figure 10: Scatter plot showing rotated pixel values as a function of original image values. Just as in previous case, the comparison is done with respected to the center of the image, and not the center that minimizes asymmetry.

The class also supports masking, smoothing kernel configuration, optional noise correction, and diagnostic plots.

```
from mex.Metrics_module import Smoothness

smoo = Smoothness(image, segmentation=None, noise=None,
                  smoothing_factor=5, smoothing_filter="box")
```

Parameters:

- `image` (`ndarray`): Input galaxy image.
- `segmentation` (`ndarray`): Binary segmentation mask.
- `noise` (`ndarray`, optional): Background-only image.
- `smoothing_factor` (int): smoothing kernel size (default = 5).
- `smoothing_filter` (str): `"box"`, `"gaussian"`, or `"tophat"`.

9.4.1 Conselice Smoothness

Method: `get_smoothness_conselice()`

```
S = smoo.get_smoothness_conselice()
```

Implements the definition from **REF**:

$$S = \frac{\sum(I - I_s - B)}{\sum I}$$

where I is the original image, I_s is the smoothed image, and B is the smoothed background (optional).

Returns:

- `S` (float): Total smoothness value.

9.4.2 Sampaio Normalized Smoothness

Method: `get_smoothness_sampaio()`

```
S_final, S_gal, S_noise = smoo.get_smoothness_sampaio()
```

Computes pixel-normalized smoothness as:

$$S = \frac{1}{2N} \sum \left| \frac{I - I_s}{I} \right| \quad \text{(with optional noise subtraction)}$$

Returns:

- `S_final` (float): Net smoothness (galaxy minus noise).
- `S_gal` (float): Galaxy-only contribution.
- `S_noise` (float): Noise-only contribution.

9.4.3 Barchi Correlation Smoothness

Method: `get_smoothness_barchi(method="spearman")`

```
S_final, r = smoo.get_smoothness_barchi()
```

Computes:

$$S = 1 - \rho$$

where ρ is the Pearson or Spearman correlation between I and I_s within the segmentation mask.

Returns:

- `S` (float): Final smoothness value.
- `rho` (float): Correlation coefficient.

9.4.4 Plotting Utilities

Method: `plot_smoothness_comparison()`

Displays side-by-side panels of the original, smoothed, and residual images. If noise is provided, the same is shown for the background. An example is shown in Figure 11.

Example:

```
smoo.plot_smoothness_comparison()
```

Method: `plot_smoothness_scatter()`

Shows scatter plots of pixel intensities before and after smoothing. Noise can be visualized on a second panel if available. An example is shown in Figure 12.

Example:

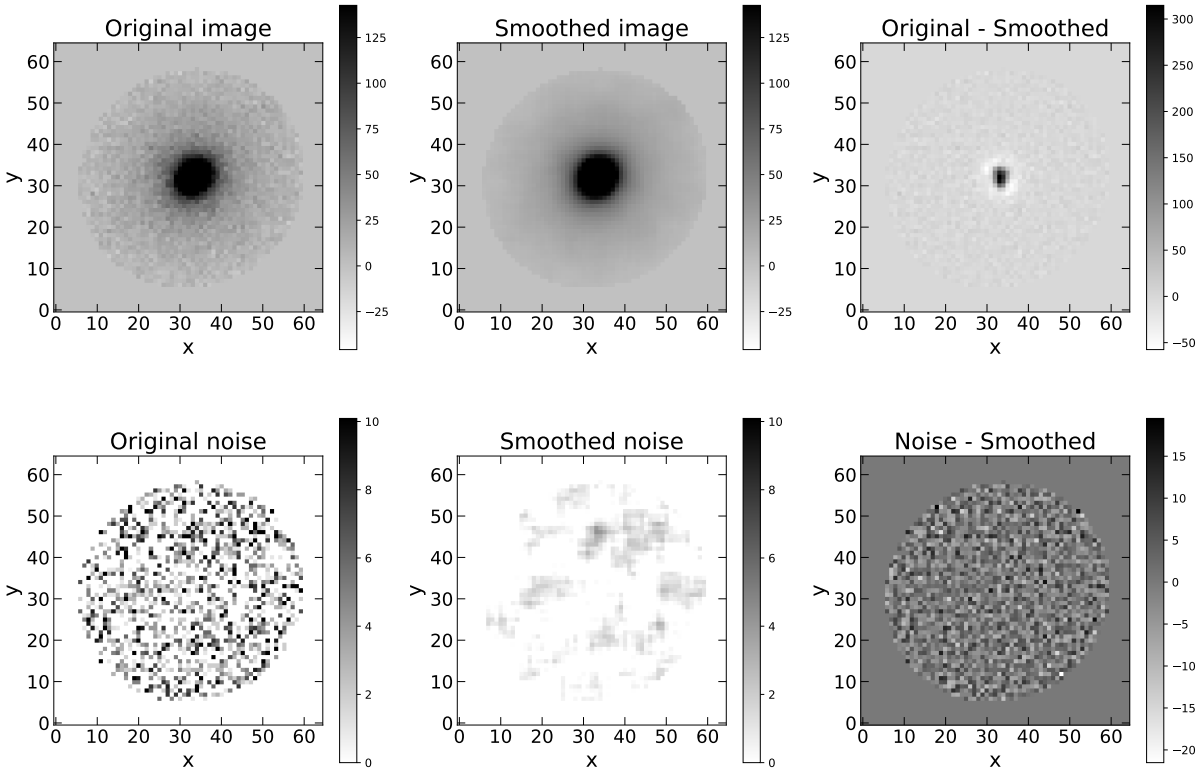

Figure 11: Panels showing original, smoothed and residual image.

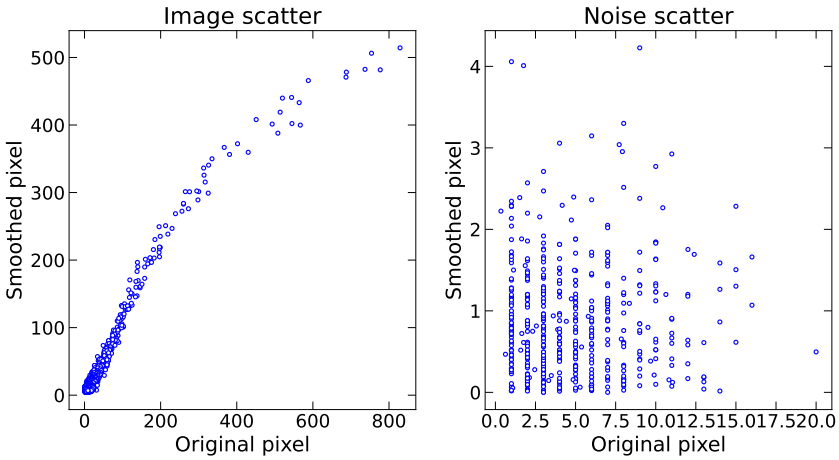

Figure 12: Scatter plot between original and smoothed image, and the noise (if provided).

```
smoo.plot_smoothness_scatter()
```

10 MEGG-System: Second moment of light + shanon Entropy + Gini index + Gradient pattern asymmetry

The MEGG system encompasses a distinct set of non-parametric morphological indicators that capture distinct statistical and structural properties of galaxy light distributions. The MEGG suite includes the following metrics:

- **M20** – the normalized second-order moment of the brightest 20% of the light, sensitive to spatial concentration and off-center clumps.
- **Shannon Entropy** – a global measure of image information content, often linked to clumpiness or texture complexity.
- **Gini Index** – a measure of flux inequality among pixels, with higher values corresponding to centrally concentrated or bulge-dominated systems.
- **Gradient Pattern Asymmetry (G2)** – a texture-based asymmetry metric derived from the divergence in gradient orientation distributions.

Together, these indices provide a robust and model-independent view of galaxy structure, extending the interpretability of morphological analysis beyond symmetric or parametric forms. In `MEx`, each MEGG index is implemented in an independent class, allowing for flexible computation, masking, and optional diagnostic visualization. This section documents the functionality and usage of each metric.

10.1 Second moment of light: (M20)

The `Moment_of_light` class computes the M_{20} statistic, a non-parametric indicator that quantifies the spatial distribution of the brightest regions of a galaxy image. M_{20} is defined as the logarithmic second-order moment of the brightest 20% of the galaxy's flux, normalized by the total second-order moment of light. This index is sensitive to features like off-center clumps, star-forming knots, or double nuclei.

```
from mex.Metrics_module import Moment_of_light
mol = Moment_of_light(image, segmentation)
```

Parameters:

- `image` (`ndarray`): 2D galaxy image.
- `segmentation` (`ndarray`): Binary mask of galaxy pixels.

10.1.1 Calculate M20

Method: `get_m20()`

Computes the M_f value based on the galaxy center and the top f% of flux.

```
m_f = mol.get_m20(x0, y0, f=0.2)
```

Parameters:

- `x0, y0` (float): Coordinates of the galaxy center.

- `f` (float): Fraction of brightest flux used (default = 0.2, which means M20).

Returns:

- `m_f` (float): M_{20} index, defined as:

$$M_{20} = \log_{10}\left(\frac{\sum_i M_i \mid \sum_i f_i < 0.2\, f_{\text{tot}}}{M_{\text{tot}}}\right)$$

Implementation details:

- Flux is sorted globally within the segmentation region.
- The brightest f fraction of total flux is isolated.
- Central second-order moments are computed using `skimage.measure.moments_central`.

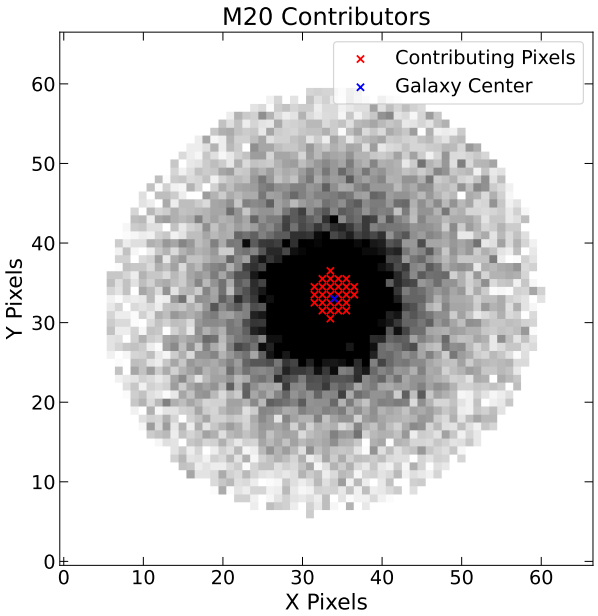

Figure 13: Original image, with pixels contributing to M20 highlighted by the red crosses. The center used for calculation is shown by a blue cross.

10.1.2 Plot pixels contributing to M20

Method: `plot_M20_contributors()`

Displays the original image with overlaid pixels that contribute to the $M_{\text{brightest}}$ moment, as in Figure 13.

Example:

```
M20_value = mol.get_m20(x0, y0)
mol.plot_M20_contributors(x0, y0)
```

10.2 Shannon entropy: (E)

The `Shannon_entropy` class measures the global entropy of the galaxy light distribution, a quantity that quantifies the degree of unpredictability or information content in the image. Higher entropy indicates more uniform, clumpy, or complex light distributions. The Shannon entropy E is computed as:

$$E = -\sum_i p_i \log_{10}(p_i)$$

where p_i is the probability of pixel intensity falling within bin i of the histogram. MEx implements two variants of this metric:

1. The **Kolesnikov-style entropy**, computed directly from the histogram of pixel values.
2. The **Barchi-style normalized entropy** , in which the raw entropy is normalized by its theoretical maximum.

$$E_{\text{norm}} = \frac{E}{\log_{10}(N_{\text{bins}})}$$

```
from mex.Metrics_module import Shannon_entropy
ent = Shannon_entropy(image, segmentation)
```

Parameters:

- `image` (`ndarray`): Input galaxy image.
- `segmentation` (`ndarray`): Binary segmentation mask.

10.2.1 Calculate shannon entropy

Method: `get_entropy()`

Computes the Shannon entropy using a fixed number of bins and optional normalization.

```
entropy = ent.get_entropy(normalize=True, nbins=100)
```

Parameters:

- `normalize` (`bool`): whether to normalize the entropy according to bins number.
- `nbins` (`integer`): number of bins used to build the histogram.

Returns:

- `entropy` (float): Entropy of pixel distribution, optionally normalized to $\log_{10}$(`nbins`).

10.2.2 Sanity check histogram

Method: `plot_entropy_frame()`

Generates a combined histogram and cumulative distribution function (CDF) for the pixel intensities within the segmentation region, as shown in Figure 14.

Example:

```
ent.plot_entropy_frame()
```

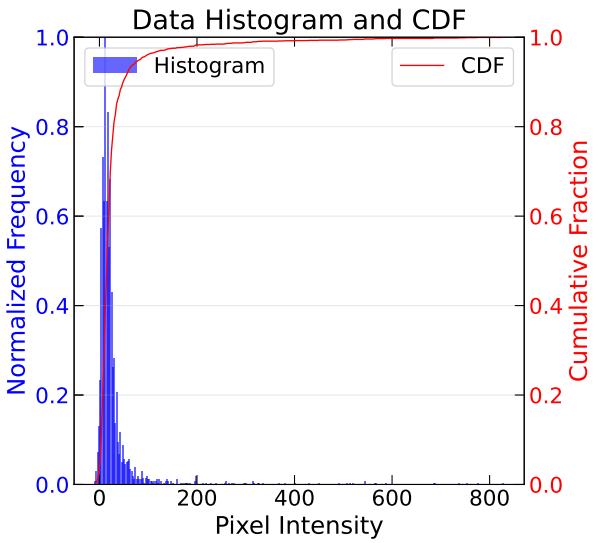

Figure 14: Histogram (left axis) and cumulative distribution function (right axis) for the galaxy pixel intensities.

10.3 Gini index: (G)

The `Gini_index` class calculates the Gini coefficient of a galaxy image, which measures the inequality in the distribution of pixel flux values. A Gini index of 0 represents complete equality (all pixels have the same flux), while a value near 1 indicates extreme inequality (e.g., all flux is concentrated in a few pixels). In morphological studies, the Gini index helps distinguish between concentrated systems (e.g., ellipticals) and diffuse or clumpy galaxies.

```
from mex.Metrics_module import Gini_index
gini = Gini_index(image, segmentation)
```

Parameters:

- `image` (`ndarray`): Input 2D galaxy image.
- `segmentation` (`ndarray`): Binary segmentation mask.

10.3.1 Calculate gini-index

Method: `get_gini()`

Computes the Gini index from the sorted pixel intensity distribution:

$$G = \frac{1}{\bar{f}N(N-1)} \sum_{i=1}^{N} (2i - N - 1) f_i$$

where f_i is the i-th sorted pixel value, N is the total number of pixels, and $\bar{f}$ is the mean intensity.

Example:

```
gini_value = gini.get_gini()
```

Returns:

- `gini` (float): Gini index of the image within the segmentation mask.

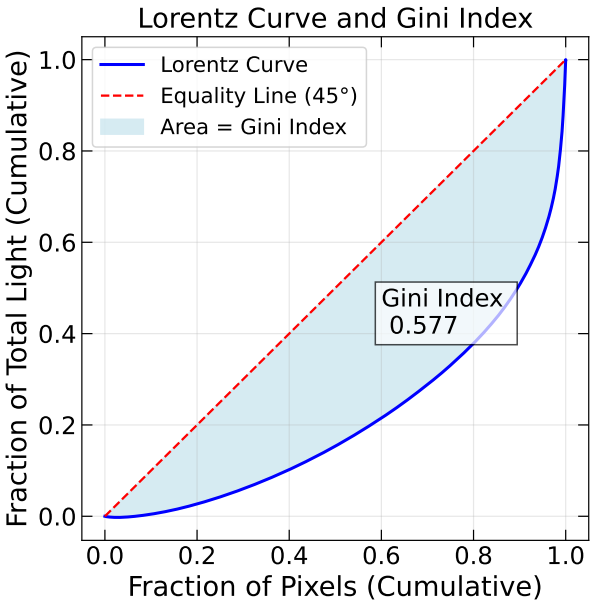

Figure 15: Diagram showing the area defining the Gini index.

10.3.2 Plot Gini diagram

Method: `compute_lorentz_curve()`

Generates the Lorentz curve used to visualize flux inequality.

Returns:

- `cumulative_pixels` (`ndarray`): Cumulative fraction of pixels.
- `cumulative_light` (`ndarray`): Corresponding cumulative light.

Method: `plot_gini_rep()`

Visualizes the Lorentz curve, equality line, and shaded area representing the Gini index, as shown in Figure 15.

Example:

```
gini_val = gini.get_gini()
x, y = gini.compute_lorentz_curve()
gini.plot_gini_rep(x, y, gini_val)
```

10.4 Gradient pattern asymmetry (G2)

The `GPA` class implements the Gradient Pattern Asymmetry (G2) index, a texture-based morphological metric. G2 quantifies the asymmetry in the spatial distribution of gradient vectors in a galaxy image, and is especially effective at detecting subtle disturbances in otherwise smooth systems.

This method decomposes the image into gradient fields, identifies asymmetric components based on module and phase tolerances, and evaluates the fraction of asymmetric vectors weighted by a "confluence" term that penalizes directional dispersion.

```
gpa = GPA(image, segmentation=None)
```

Parameters:

- `image` (`ndarray`): Input galaxy image.
- `segmentation` (`ndarray`, optional): Binary segmentation mask. If not provided, all pixels are considered valid.

10.4.1 G2 Calculation

Method: `get_g2()`

Computes the G2 asymmetry index based on vector field symmetry.

```
g2 = gpa.get_g2(mtol=0, ptol=0, remove_outliers='')
```

Parameters:

- `mtol` (float): Module tolerance threshold for symmetric matching.
- `ptol` (float): Phase tolerance threshold for symmetric matching (degrees).
- `remove_outliers` (str): If set to `"new"`, removes gradient outliers at the 3σ level.

Returns:

- `g2` (float): Gradient Pattern Asymmetry index:

$$G2 = \left(\frac{N_{\text{asym}}}{N_{\text{valid}}}\right) \cdot (1 - C)$$

where C is the confluence of asymmetric vectors and N_{asym} is the number of unmatched gradient pairs.

Notes:

- The image must be square and of type `float32`.
- The vector field is rotated by 180° to identify symmetric counterparts.
- Outliers in module and angle space may be masked to increase robustness.

10.4.2 Sanity checks

Method: `plot_gradient_field(mtol, ptol)`

Displays side-by-side vector field plots of the original and asymmetric gradient vectors, as shown in Figure 16.

Example:

```
gpa.plot_gradient_field(mtol=0.05, ptol=15)
```

Method: `plot_hists()`

Plots the normalized histograms of the gradient module and orientation phase distributions used in the G2 calculation, as shown in Figure 17.

Example:

```
gpa.plot_hists()
```

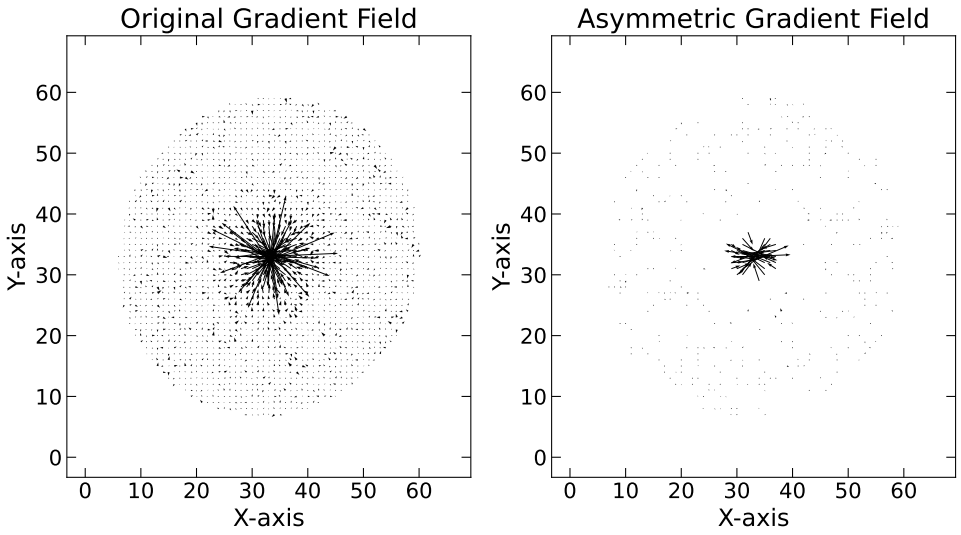

Figure 16: Left: gradient field for the observed galaxy. Right: Asymmetric vector field.

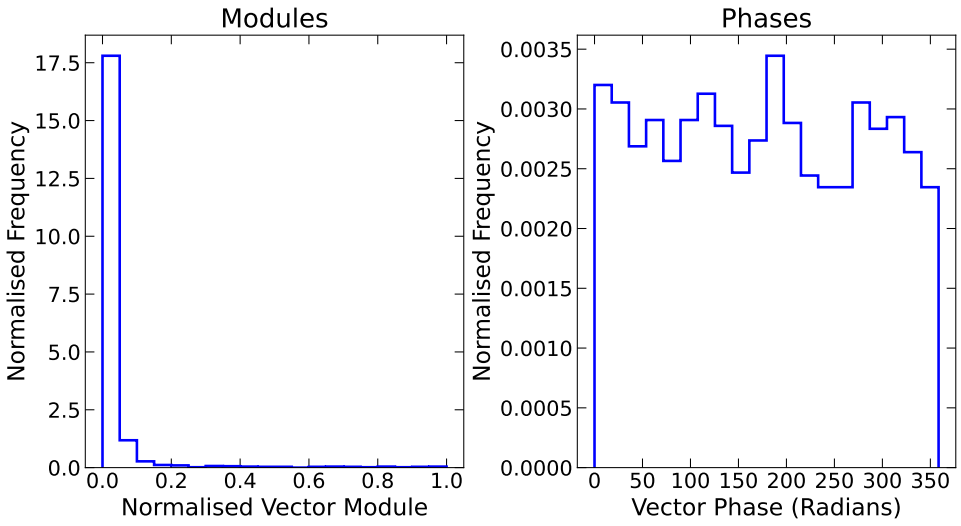

Figure 17: Left: gradient field for the observed galaxy. Right: Asymmetric vector field.