pawlikMorph-LSST

Lewis McMillan

Sep 04, 2020

CONTENTS

PawlikMorph-LSST is a python package for calculating various non-parametric morphological measures of galaxies. The package also allows the calculation of segmentation maps, and masking of stars that may contaminate any morphological measures. The code in this package is based upon/translation of pawlikMorph in IDL.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

1.1.1 Requirements

PawlikMorph-LSST depends on several other packages for some features:

- Python 3.7+
- Numpy 1.15.4+
- Numba 0.38.1+
- Astropy 3.0.3+
- Scikit-image 0.14.0+
- Pandas 0.25.3+
- photutils 0.7.1+
- scipy 1.3.2+
- parsl 0.9.0+

If diagnostic.py is used:

- matplotlib

1.1.2 Installing the latest version

First clone the GitHub repository:

- Via commandline:

```
git clone https://github.com/lewisfish/PawlikMorph-LSST.git
```

- Or via webpage:
  - https://github.com/lewisfish/PawlikMorph-LSST/archive/master.zip

Then install the dependancies:

- Via conda:

```
conda env create -f environment.yml
```

- Or via pip:

```
pip install --user --requirement requirements.txt
```

1.2 Overview

For a given image, pawlikMorph-LSST will calculate a segmentation map of the galaxy of interest, subtract the sky background, then analyse the image for the following statistics:

- Gini index
- M20
- CAS
- As
- Sérsic index

There are also options to provide pawlikMorph-LSST with a star catalogue to "clean" the image of external sources, so that they do not interfere with the morphology statistics.

The overall aim of this package is to be able to analyse the images that the LSST telescope will produce. This will involve being able to interface with the LSST data via Edinburgh's LSST servers, using LSST data Butler.

This package is based upon M. Pawlik's IDL code, and some functions are direct translations from IDL to python.

1.3 Getting Started

Tutorial:

For ease of use we also provide with the package, a commandline python script to run the code, imganalysis.py.

imganalysis can be used in the commandline by running:

```
python imganalysis.py
```

The script has several possible options:

-h, --help	show this help message and exit.
-A	Calculate asymmetry parameter.
-As	Calculate shape asymmetry parameter.
-Aall	Calculate all asymmetries parameters.
-sersic, --sersic	Calculate Sersic profile.
-spm, --savepixmap	Save calculated binary pixelmaps.
-sci, --savecleanimg	Save cleaned image.
-li, --largeimage	Use large cutout for sky background estimation.
-lif, --largeimagefactor	Factor to scale cutout image size (–imgsize) so that a better background value can be estimated.
-f FILE, --file FILE	File which contains list of images, and RA DECS of object in image.
-fo FOLDER, –folder FOLDER	Give location for where to save output files/images

-src {SDSS,LSST}, –imgsource {SDSS,LSST} Telescope source of the image. Default is SDSS. This option specifies method of ingestion of the FITS image files.

-s IMGSIZE, --imgsize IMGSIZE Size of image cutout to analyse.

-cc CATALOGUE, –catalogue CATALOGUE Check if any object in the provided catalogue occludes the analysed object.

-ns NUMSIG, –numsig NUMSIG Radial extent to which mask out stars if a catalogue is provided.

-fs {1,3,5,7,9,11,13,15}, –filtersize {1,3,5,7,9,11,13,15} Size of kernel for mean filter.

-par {multi,parsl,none}, –parlib {multi,parsl,none} Choose which library to use to parallelise script. Default is none.

-n CORES, --cores CORES Number of cores/process to use in calculation

-m, --mask If this option is provided then the script expects there to be precomputed masks in the format **pixelmap_** + filename in the same folder as the images for analysis

-cas, --cas If this option is enabled, the CAS parameters are calculated (Gini, M20, r20, r80, concentation, smoothness)

Example:

```
python imganalysis.py -f images.csv -fo output/ -Aall -spm -sci -src sdss -sersic -
↪cas -n 4
```

The above command will run the code over all the images in the file images.csv, and calculate all the asymmetry statistics, alongside the CAS and Sersic statistics. This will generate a folder output where pixelmaps of the object, cleaned images, and calculated parameters (parameters.csv) are stored. The command will also run the code in parallel over 4 processors. The input csv file (images.csv in this example) should have the following columns: filename, ra, dec The filename column should contain the path to the image relative to the directory you are runnning the script in. RA, and DEC shoul be the location of the galaxy/object to be analysed.

If imganalysis.py is run with the -spm and -sci options, then it will automatically plot the various outputs using matplotlib at the end of a run. This can be useful to asses by eye the various setting used in that run for generating the segmentation map, and size of star to mask out.

imganalysis.py is provided as an easy option for calculating various morphology statistics. However, the package can also be used as a general purpose library for building your own scripts. The package publicly exposes several of the functions in order to achieve this.

1.4 LSST Support

In order to be able ingest images via the LSST pipeline you must first install the LSST pipeline.

Note LSST support is currently experimental. The LSST API is currently not finalised so using the latest version of the LSST pipeline may break the code. If in doubt try and use version 19, the last version this code was tested against.

To install the LSST pipeline please refer to https://pipelines.lsst.io/#installation. We recommend using Docker as the installation method for the LSST pipeline. The following instructions assume that Docker has been used to install the LSST pipeline.

To use run pawlikMorph-LSST with the LSST pipeline first mount the directory where the pawlikMorph-LSST code is stored (i.e the directory above pawlikMorphLSST/).

```
$ docker run -it -v `pwd`:/home/lsst/mnt lsstsqre/centos:7-stack-lsst_distrib-v19_0_0
```

Setup the LSST environment using which ever shell you are using (if in doubt try .bash)

```
$ source loadLSST.bash
$ setup lsst_distrib
```

Next you need to change directory into the directory with the code and data

```
$ cd
$ cd mnt/
```

Finally you need to install the dependacies that do not come with the LSST Docker image

```
$ pip install numba parsl scikit-image photutils
```

If everything was correctly installed then running

```
$ python imganalysis --help
```

Should output the help for the imganalysis script.

For more help on Docker https://docs.docker.com/get-started/.

1.5 Using the LSST databutler

Currently this code is only setup for reading in SDSS images using the LSST pipeline. The SDSS images must be available on the machine the script is running on. The folder structure for the SDSS images that the LSST pipeline expects is the same as found in http://das.sdss.org/imaging/, i.e . /path/to/data/756/41/corr/1/fpC-000756-r1-0234.fits etc. Where 756 must be the run number, 41 is the rerun number (must be greater than 40), and 1 is the camcol number.

The script expects a csv file passed via the –file flag with the following columns: ra, dec, run, rerun, camCol, and field. The only other difference for the user along with the different csv file, is to pass the LSST option to the –imgsource flag. Below is an example command for using the LSST pipeline:

```
$ python imganalysis.py --file sdssimgsinfo.csv --imgsource LSST -Aall -sersic -cas -
↪n 4
```

CHAPTER

TWO

USER DOCUMENTATION

2.1 Apertures

`pawlikMorphLSST.apertures.`**`makeaperpixmaps`**(*npix: int*, *folderpath=None*) → None

Writes the aperture binary masks out after calculation.

Parameters

- **`npix`** (*`int`*) – Width of aperture image.
- **`folderpath`** (*`Pathlib object`*) – Path to the folder where the aperture masks should be saved.

Returns

Return type None

`pawlikMorphLSST.apertures.`**`distarr`**(*npixx: int*, *npixy: int*, *cenpix: numpy.ndarray*) → numpy.ndarray

Creates an array of distances from given centre pixel.

Near direct translation of IDL code.

Parameters

- **`npixx`** (*`int`*) – Number of x pixels in the aperture mask.
- **`npixy`** (*`int`*) – Number of y pixels in the aperture mask.
- **`cenpix`** (*`np.ndarray`*) – Location of central pixels.

Returns array of distances.

Return type np.ndarray

`pawlikMorphLSST.apertures.`**`subdistarr`**(*npix: int*, *nsubpix: int*, *cenpix: List[int]*) → numpy.ndarray

Writes the aperture binary masks out after calculation.

Near direct translation of IDL code.

Parameters

- **`npix`** (*`int`*) – Number of pixels in the aperture mask.
- **`nsubpix`** (*`int`*) – Number of subpixels.
- **`cenpix`** (*`List[int]`*) – Location of central pixels.

Returns Array of sub-distances.

Return type np.ndarray

`pawlikMorphLSST.apertures.`**`apercentre`**(*apermask: numpy.ndarray*, *pix: numpy.ndarray*) → numpy.ndarray

Function that centers a precomputed aperture mask on a given pixel.

Parameters

- **`apermask`** (*`np.ndarray`*) – Aperture mask that is to be centred.
- **`pix`** (*`List[int]`*) – Central pixel indicies.

Returns **mask** – Returns aperture mask centered on central pixel, pix.

Return type np.ndarray

`pawlikMorphLSST.apertures.`**`aperpixmap`**(*npix: int*, *rad: float*, *nsubpix: int*, *frac: float*) → numpy.ndarray

Calculate aperture binary mask.

Calculates the aperture binary mask through pixel sampling knowing the aperture radius and number of subpixels.

Near direct translation of IDL code.

Parameters

- **`npix`** (*`int`*) – Width of aperture image.
- **`rad`** (*`float`*) – Radius of the aperture.
- **`nsubpix`** (*`int`*) – Number of subpixels
- **`frac`** (*`float`*) – Fraction of something… Maybe due to Petrosian magnitude?

Returns Numpy array that stores the mask.

Return type np.ndarry

2.2 CASgm

Module contains routines to calculate CAS parameters, as well as Gini, M20, R20, R80.

`pawlikMorphLSST.casgm.`**`gini`**(*image: numpy.ndarray*, *mask: numpy.ndarray*) → float

Calculation of the Gini index of a Galaxy.

$$g = \frac{1}{2\bar{X}n(n-1)} \sum (2i - n - 1) |X_i|$$

Where $\bar{X}$ is the mean over all intensities n is the total number of pixels X_i are the pixel intensities in increasing order

see Lotz et al. 2004 https://doi.org/10.1086/421849

Parameters

- **`image`** (*`float, 2d np.ndarray`*) – Image from which the Gini index shall be calculated
- **`mask`** (*`int, 2D np.ndarray`*) – TMask which contains the galaxies pixels

Returns **G** – The Gini index.

Return type float

`pawlikMorphLSST.casgm.`**`m20`**`(`*image: numpy.ndarray*, *mask: numpy.ndarray*`)` → float

Calculate the M20 statistic.

$$M_{20} = log_{10}\left(\frac{\sum M_i}{M_{tot}}\right)$$

$$While \sum f_i < 0.2 f_{tot}$$

$$M_{tot} = \sum M_i = \sum f_i[(x - x_c)^2 + (y - y_c)^2]$$

see Lotz et al. 2004 https://doi.org/10.1086/421849

Adapted from statmorph: https://github.com/vrodgom/statmorph

Parameters

- **`image`** (*`float, 2d np.ndarray`*) – Image of galaxy
- **`mask`** (*`float [0. - 1.], 2d np.ndarray`*) – Mask which contains the pixels belonging to the galaxy of interest.

Returns **m20** – M20 statistic

Return type float

`pawlikMorphLSST.casgm.`**`concentration`**`(`*r20: float*, *r80: float*`)` → float

Calculation of the concentration of light in a galaxy.

$$C = 5log_{10}(\frac{r_{80}}{r_{20}})$$

see Lotz et al. 2004 https://doi.org/10.1086/421849

Parameters

- **`r20`** (*`float`*) – Radius at 20% of light
- **`r80`** (*`float`*) – Radius at 80% of light

Returns **C** – The concentration index

Return type float

`pawlikMorphLSST.casgm.`**`smoothness`**`(`*image: numpy.ndarray*, *mask: numpy.ndarray*, *centroid: List[float]*, *Rmax: float*, *r20: float*, *sky: float*`)` → float

Calculate the smoothness or clumpiness of the galaxy of interest.

$$S = \frac{\sum |I - I_s| - B_s}{\sum |I|}$$

Where I is the image I_s is the smoothed image B_s is the background smoothness

see Lotz et al. 2004 https://doi.org/10.1086/421849

Parameters

- **`image`** (*`float, 2d np.ndarray`*) – Image of galaxy
- **`mask`** (*`float [0. - 1.], 2d np.ndarray`*) – Mask which contains the pixels belonging to the galaxy of interest.
- **`centroid`** (*`List[float]`*) – Pixel location of the brightest pixel in galaxy.
- **`Rmax`** (*`float`*) – Distance from brightest pixel to furthest pixel in galaxy
- **`r20`** (*`float`*) – Distance from brightest pixel to radius at which 20% of light of galaxy is enclosed.

- **sky** (*float*) – Value of the sky background.

Returns **Result** – The smoothness or clumpiness parameter, S.

Return type float

`pawlikMorphLSST.casgm.`**`calcR20_R80`**(*image: numpy.ndarray*, *centroid: List[float]*, *radius: float*) → Tuple[float, float]

Calculation of r_{20}, and r_{80}

Parameters

- **image** (*float, 2d np.ndarray*) – Image of galaxy
- **centroid** (*List[float]*) – Location of the brightest pixel
- **radius** (*float*) – Radius in which to measure galaxy light out to.

Returns **r20, r80** – The radii where 20% and 80% light falls within

Return type Tuple[float, float]

`pawlikMorphLSST.casgm.`**`calculateCSGM`**(*image: numpy.ndarray*, *mask: numpy.ndarray*, *skybgr: float*) → Tuple[float]

Helper function that calculates the CSGM parameters

Parameters

- **image** (*np.ndarray, 2D float*) – Image of a galxy for which the CSGM parameters are to be calculated
- **mask** (*np.ndarray, 2D uint8*) – Image for which only the pixels that belong to the galaxy in "image" are "hot".
- **skybgr** (*float*) – The value of the sky background in the given image

Returns **C, S, gini, m20** – The CSGM parameters

Return type Tuple[float]

2.3 Asymmetry

`pawlikMorphLSST.asymmetry.`**`minapix`**(*image: numpy.ndarray*, *mask: numpy.ndarray*, *apermask: numpy.ndarray*, *starMask=None*) → List[int]

Find the pixel that minimises the asymmetry parameter, A

Selects a range of candidate centroids within the brightest region that compromises of 20% of the total flux within object. Then measures the asymmetry of the image under rotation around that centroid. Then picks the centroid that yields the minimum A value.

Parameters

- **image** (*np.ndarray*) – Image that the minimum asymmetry pixel is to be found in.
- **mask** (*np.ndarray*) – Precomputed mask that describes where the object of interest is in the image
- **apermask** (*np.ndarray*) – Precomputed aperture mask
- **starMask** (*np.ndarray*) – Precomputed mask that masks stars that interfere with object measurement

Returns **Centroid** – The minimum asymmetry pixel position.

Return type List[int]

`pawlikMorphLSST.asymmetry.`**`calcA`**(*img: numpy.ndarray*, *pixmap: numpy.ndarray*, *apermask: numpy.ndarray*, *centroid: List[int]*, *angle: float*, *starMask=None*, *noisecorrect=False*) → List[float]

Function to calculate A, the asymmetry parameter.

$$A = \frac{\sum |I_0 - I_\theta|}{2 \sum I_0} - A_{bgr}$$

Where I_0 is the original image, I_θ is the image rotated by θ degrees, and A_{bgr} is the asymmerty of the sky.

See Conselice et al. for full details.

Near direct translation of IDL code.

Parameters

- **`img`** (*`np.ndarray`*) – Image to be analysed.
- **`pixmap`** (*`np.ndarray`*) – Mask that covers object of interest.
- **`apermask`** (*`np.ndarray`*) – Array of the aperture mask image.
- **`centroid`** (*`np.ndarray`*) – Pixel position of the centroid to be used for rotation.
- **`angle`** (*`float`*) – Angle to rotate object, in degrees.
- **`starMask`** (*`np.ndarray`*) – Precomputed mask that masks stars that interfere with object measurement
- **`noisecorrect`** (*`bool, optional`*) – Default value False. If true corrects for background noise

Returns **A, Abgr** – Returns the asymmetry value and its background value.

Return type List(float)

`pawlikMorphLSST.asymmetry.`**`calculateAsymmetries`**(*image: numpy.ndarray*, *pixelmap: numpy.ndarray*) → Tuple[float]

helper function to calculate all asymmetries

Parameters

- **`image`** (*`np.ndarray, 2d float`*) – image of a galaxy for which the asymmetries should be calculated.
- **`pixelmap`** (*`np.ndarray, 2d uint8`*) – Pixel mask of the galaxy calculated from image.

Returns **A, As, As90** – The calculated asymmetry values.

Return type Tuple, float

2.4 Sersic

`pawlikMorphLSST.sersic.`**`fitSersic`**(*image: numpy.ndarray*, *centroid: List[float]*, *fwhms: List[float]*, *theta: float*, *starMask=None*)

Function that fits a 2D sersic function to an image of a Galaxy.

Parameters

- **`image`** (*`np.ndarray`*) – image to which a 2D Sersic function will be fit

- **`centroid`** (*`List[float]`*) – Centre of object of interest
- **`fwhms`** (*`List[float]`*) – Full width half maximums of object
- **`theta`** (*`float`*) – rotation of object anticlockwise from positive x axis
- **`starMask`** (*`np.ndarray`*) – Mask contains star locations.

Returns **Parameters** – Collection of best fit parameters for the 2D sersic function

Return type astropy.modeling.Model object

2.5 Pixmap

This module contains the routines to calculate the segmentation map based upon the 8 connected pixel method, and methods related to the segmentation map.

To calculate a segmentation map (pixelmap), we first apply a mean filter of size {filterSize}. This filter size can be set using the –filterSize command line option when using imganalysis.py. Currently the filter size is constrained to be an odd integer between 3 and 15.

The next step is to reduce the image size by a factor of filterSize. We then check if the central pixel is brighter than the sky value, if it is not we raise an ImageError.

We then set the central pixel as part of the segmentation map and start the 8 connected algorithm. This algorithm checks the surrounding 8 pixels that are connected to the current pixel, starting with the central pixel. Each of the connecting pixels are only added to the segmentation map if and only if that pixel's flux is greater than $sky + \sigma_{sky}$. This whole process repeats for each pixel that is added to segmentation map until no more pixels can be added.

If a starmask is provided, the above process still takes place, except some of the pixels where there are stars are pre-masked out before the 8 connected algorithm starts.

The other functions in this module are methods that operate on the mask. For instance, calcRmax, calculates the maximum distance from a pixel on the edge of the segmentation map to the central pixel. calcMaskedFraction is only used if a star catalogue is provided. This calculates the amount of pixels masked out of object of interest due to masking of nearby stars. This is calculated by first rotating the masked output and then taking the residual of this in order to offset any artificially induced asymmetry. Finally, checkPixelmapEdges gives a warning if any pixels in the segmentation map are on the edge of the image.

Below are the function signatures for pixelmap, and the other functions in this module.

`pawlikMorphLSST.pixmap.`**`pixelmap`**(*image: numpy.ndarray*, *threshold: float*, *filterSize: int*, *starMask=None*) → numpy.ndarray

Calculates an object binary mask.

This is acheived using a mean filter, 8 connected pixels, and a given threshold.

Parameters

- **`image`** (*`np.ndarray`*) – Image from which the binary mask is calculated.
- **`threshold`** (*`float`*) – Threshold for calculating 8 connectedness
- **`filterSize`** (*`int`*) – Size of the mean filter. Must be odd
- **`starMask`** (*`optional, None or np.ndarray`*) – Mask that mask out nuisance stars

Returns **objectMask** – Calculated binary object mask

Return type np.ndrray

`pawlikMorphLSST.pixmap.`**`calcMaskedFraction`**(*oldMask: numpy.ndarray*, *starMask: numpy.ndarray*, *cenpix: List[float]*) → float

Calculates the fraction of pixels that are masked.

Parameters

- **`oldMask`** (*`np.ndarray`*) – Mask containing object of interest.
- **`starMask`** (*`np.ndarray`*) – Mask containing location of star
- **`cenpix`** (*`List[float]`*) – Centre of asymmetry in pixels.

Returns Fraction of object pixels masked by stars

Return type float

`pawlikMorphLSST.pixmap.`**`calcRmax`**(*image: numpy.ndarray*, *mask: numpy.ndarray*) → float

Function to calculate the maximum extent of a binary pixel map

Parameters

- **`image`** (*`float, 2d np.ndarray`*) – Image of galaxy.
- **`mask`** (*`np.ndarray`*) – Binary pixel mask

Returns **rmax** – the maximum extent of the pixelmap

Return type float

`pawlikMorphLSST.pixmap.`**`checkPixelmapEdges`**(*mask: numpy.ndarray*) → bool

Flag image if galaxy pixels are on edge of image.

Parameters **`mask`** (*`np.ndarray`*) – pixelmap to check

Returns **flag** – If true then the pixelmap hits the edge of the image

Return type bool

2.6 ObjectMasker

This module is only used if a star catalogue is used via the command line option –cc. The catalogue is expected to be a comma separated CSV file with the following header and columns:

```
`objID, ra, dec, psfMag_r, type`
```

Where ObjId is the object ID, RA is the right ascension, DEC is the Declination, psfMAG_r is the PSF magnitude for the r band, and type is the object type from {STAR, GALAXY, COSMICRAY, UNKNOWN}.

The function simply reads in the star catalogue and checks for which object to match with. By default this is only STAR, however the other types can be enabled. The function then converts RA DEC pairs to pixel coordinates and uses these to check if the object is in the image. If the object is in the image it then checks to see if occludes the segmentation map.

`pawlikMorphLSST.objectMasker.`**`objectOccluded`**(*mask: numpy.ndarray*, *radec: Tuple[float, float]*, *catalogue: str*, *header*, *galaxy=False*, *cosmicray=False*, *unknown=False*) → Tuple[bool, List[float]]

Function gets list of objects near the object of interest, and determines if that objects light occludeds the object of interest light.

Parameters

- **`mask`** (*`np.ndarray`*) – Object mask

- **radec** (*Tuple[float, float]*) – Tuple of ra, dec
- **catalogue** (*str*) – Name of object catalogue to check against Expected format is objID: float, ra: float, dec: float, type: str
- **header** – fits image header
- **galaxy** (*bool, optional*) – Option to include galaxy objects
- **cosmicray** (*bool, optional*) – Option to include cosmic rays
- **unknown** (*bool, optional*) – Option to include unknown objects

Returns Returns true alongside list of objects that occlude object mask. Otherwise returns false and an empty list

Return type Tuple[bool, List[float]]

2.7 Result

This module contains the data class which acts as a container for all the results and parameters calculated or needed for the analysis code. Default value for numerical values is -99. If in the final results -99 appears this means that the value was either not clacluated or there has been an error with that part of the analysis. Default value for strings is an empty string.

class pawlikMorphLSST.result.**Result** (*file: str*, *outfolder: Any*, *occludedFile: str*, *pixelMapFile: Any = ''*, *cleanImage: Any = ''*, *starMask: Any = ''*, *objList: Any = <factory>*, *A: List[float] = <factory>*, *As: List[float] = <factory>*, *As90: List[float] = <factory>*, *rmax: float = -99*, *apix: Tuple[float] = (-99.0, -99.0)*, *sky: float = -99.0*, *sky_err: float = 99.0*, *fwhms: List[float] = <factory>*, *theta: float = -99.0*, *r20: float = -99.0*, *r80: float = -99.0*, *C: float = -99.0*, *gini: float = -99.0*, *m20: float = -99.0*, *S: float = -99.0*, *sersic_amplitude: float = -99.0*, *sersic_r_eff: float = -99.0*, *sersic_n: float = -99.0*, *sersic_x_0: float = -99.0*, *sersic_y_0: float = -99.0*, *sersic_ellip: float = -99.0*, *sersic_theta: float = -99.0*, *time: float = 0.0*, *star_flag: bool = False*, *maskedPixelFraction: float = -99.0*, *objectEdge: bool = False*)

Data class that stores the results of image analysis.

A: List[float] = None
Calculated asymmetry value, format [A, A_error]

As: List[float] = None
Calculated shape asymmetry value, format [As, As_error]

As90: List[float] = None
Calculated shape asymmetry 90 value, format [As90, As90_error]

C: float = -99.0
Concentraion value

S: float = -99.0
Smoothness value.

apix: Tuple[float] = (-99.0, -99.0)
Asymmetry (A) minimised central pixel

`cleanImage: Any = ''`
path to clean image.

`file: str = None`
Filename of image

`fwhms: List[float] = None`
FWHM's of the fitted 2D Gaussian

`gini: float = -99.0`
Gini index

`m20: float = -99.0`
M20 value

`maskedPixelFraction: float = -99.0`
Fraction of pixels masked due to occluding star/object.

`objList: Any = None`
List of objects RA, DECs that occlude objects segmentation map.

`objectEdge: bool = False`
If true then the segmentation map extends to an edge of the image.

`occludedFile: str = None`
Filename of output data for objects that occlude with segmentation map.

`outfolder: Any = None`
Output folder for saving data

`pixelMapFile: Any = ''`
Path to segmentation map

`r20: float = -99.0`
Radius in which 20% of total light flux is contained

`r80: float = -99.0`
Radius in which 80% of total light flux is contained

`rmax: float = -99`
Maxmimum radius of the segmentation map

`sersic_amplitude: float = -99.0`
Sersic amplitude.

`sersic_ellip: float = -99.0`
Sersic ellipticity.

`sersic_n: float = -99.0`
Sersic index.

`sersic_r_eff: float = -99.0`
Sersic effective radius

`sersic_theta: float = -99.0`
Sersic rotation.

`sersic_x_0: float = -99.0`
Sersic x centre

`sersic_y_0: float = -99.0`
Sersic y centre

sky: float = -99.0
Sky background value.

sky_err: float = 99.0
Sky background error.

starMask: Any = ''
path to star mask

star_flag: bool = False
If true means that there is a star in the catalogue occluding the objects segmentation map

theta: float = -99.0
Theta of the fitted 2D Gaussian

time: float = 0.0
Time taken to analyse image.

write(*objectfile*)
Write out result as a row to a csv file

2.8 Image

This module contains the Image abstract class, with the SDSS and SDSS-LSST class that inherit from it. The Image abstract class is meant to be a base class for future expansion of the code for other sources of images. The SDSS image class should function as a catch all for any source of FITS images, not just SDSS images. All that is needed to use the SDSS image class is a FITS image and a RA, and DEC value to pinpoint the object of interest.

The LSST image class for now only works with SDSS images in a certain folder structure. However, this is provided in order to help any future developers

This package can be extended by sub classing Image and implementing the required methods, and _IMAGE_TYPE.

class pawlikMorphLSST.Image.**Image**(*filename=None*)
Abstract base class for images

abstract getHeader()

abstract getImage()

pawlikMorphLSST.Image.**readImage**(*filename: str, ra: float, dec: float, npix=128, header=False*)

Helper function that can be used to read images directly without need to manually create Image class.

Parameters

- **filename** (*sty*) – location of image to read
- **ra** (*float*) – RA, right ascension of object of interest in image
- **dec** (*float*) – DEC, declination of object of interest in image
- **npix** (*int, optional*) – Size of cutout to return from larger image. Default is 128
- **header** (*bool, optional*) – If true return header information as well. Default is False

Returns

- **img** (*np.ndarray, 2D, float*) – Cutout image.
- *If header=True then also returns the header from the FITS file.*

class pawlikMorphLSST.Image.**sdssImage**(**args*, ***kwargs*)
Class for SDSS images, as ingested by standard 'method'

getHeader() → astropy.io.fits.header.Header
Returns the image header

Returns **self.header** – The header for the cutout.

Return type astropy.io.fits.Header

getImage() → numpy.ndarray
Returns the cutout image

Returns **self.image** – The cutout image centered on the view provided in setview.

Return type np.ndarray, float (2D)

setView(*ra=None*, *dec=None*, *npix=128*)

Get the correct view in larger image, and create the cutout on the correct view

Parameters

- **ra** (*float*) – RA position
- **dec** (*float*) – DEC position
- **npix** (*int*) – Size to make cutout

class pawlikMorphLSST.Image.**lsstImage**(**args*, ***kwargs*)
Class for SDSS images ingested via LSST dataButler.

Some metadata not available, this includes wcs, and pixel value conversion information (bscale, bzero etc). Shouldn't really recreate butler on each image call. . .

This code is far from the optimal way to read images. https://github.com/LSSTScienceCollaborations/StackClub

The above source maybe of help for future developer.

getHeader()

getImage()

setView(*ra*, *dec*, *run*, *camCol*, *field*, *filter='r'*, *npix=128*)

2.9 ImageUtils

This module contains several utility function that operate on the image.

maskstarSEG "cleans" the image of stars and other objects that emit light if they do not occlude the segmentation map.

maskstarPSF uses the star catalogue and information on the PSF in the FITS image header if both are available in order to mask out stars using the PSF. maskSstarPSF is experimental and has not been fully developed. The method first tries to read in information on the PSF and other viewing parameters. These values are used to calculate the sky's magnitude. It then uses the objects that occlude the segmentation map as calculated by *ObjectMasker* module, and calculates the radius of that object based upon its psfMAG_r. The algorithm then masks that object out to its radius $\times$ {numsig}, where numsig is the number factor tha allows the user some control of this process. If the adaptive option is enabled then the algorithm calculates the radius based upon following a line of flux from the centre of the object in the opposite direction of the object of interests centre, and checking for a discontinuity in the flux. It then extends the masking out until there is no discontinuity in the flux.

pawlikMorphLSST.imageutils.**maskstarsSEG**(*image: numpy.ndarray*) → numpy.ndarray

'Cleans' image of external sources. For example will remove all stars that do not interfere with the object of interest.

Parameters **`image`** (*`np.ndarray`*) – Image to be cleaned.

Returns **imageClean** – Image cleaned of external sources.

Return type np.ndarray

`pawlikMorphLSST.imageutils.`**`maskstarsPSF`**(*image: numpy.ndarray*, *objs: List*, *header*, *skyCount: float*, *numSigmas=5.0*, *adaptive=True*, *sky_err=0.0*) → numpy.ndarray

Use the PSF to estimate stars radius, then masks them out.

Parameters

- **`image`** (*`np.ndarray`*) – Image that is to be masked of nuisance stars.
- **`objs`** (*`List[float, float, str, float]`*) – List of objects. [RA, DEC, type, psfMag_r]
- **`header`** (*`astropy.io.fits.header.Header`*) – The header of the current image. Contains information on PSF and various other parameters.
- **`skyCount`** (*`float`*) – Sky background in counts.
- **`numSigmas`** (*`optional, float`*) – Number of sigmas that the stars radius should extend to

Returns **mask** – Array that masks stars on original image.

Return type np.ndarray

2.10 SkyBackground

This module contains the routines used to calculate the sky background value. It does this by first fitting a Gaussian to the object of interest in the image.

This routine carries out the following procedures:

First it checks if there are any sources of light in the image. If there are none then the algorithm raises a SkyError.

The next step is to fit a 2D Gaussian to the light source. There are two 2D Gaussian fitting routines used. The one used first is less robust but faster, and usually gives an accurate fit. However, if this fitter fails then the 2nd Gaussian fitter is used, which is robust, but a lot slower.

Next the algorithm checks the size of the sky area around the object of interest. If this area is less than 300 pixels or the radius of the Gaussian is larger than 3 times the image size, then more robust Gaussian fitter is used. This is done as most of the time the less robust fitter is used and it cant cope with objects that are on the same size scale as the image itself.

Again the sky area is checked, this time if it is less than 100 we raise a SkyError.

The final step is to calculate the sky background value. This is achieved by first calculating the mean, median, and std of the sky region just calculated. If the median is greater than or equal to the mean, then the sky background value is just the mean, and the error in this is the std. If the mean is greater than the median, them we use sigma clipping, with a sigma of 3 to recalculate the mean, median, and std. We then set the sky background value to that of $sky = 3median - 2mean$, and the error in this to that of the recalculated std.

The function then returns the value of the sky, its error, and the FWHM's, and theta of the fitted Gaussian.

In order to get a better estimation of the sky background value, a larger image can be passed to the routine. If a larger image is passed to the routine, then the routine will use this larger image to estimate the sky background rather than the smaller cutout. The larger image size can be defined by the imganalysis.py command line option -largeimagefactor. This factor essentially makes a larger cutout image.

Below is the exposed function for calculating the sky background value.

`pawlikMorphLSST.skyBackground.`**`skybgr`**(*img: numpy.ndarray*, *largeImage=None*, *file=None*, *imageSource=None*) → Tuple[float, float, List[float], float]

Helper function for calculating skybgr

Parameters

- **`img`** (*`np.ndarray`*) – image from which a sky background will be calculated
- **`file`** (*`Path object, optional`*) – Path to image
- **`largeImage`** (*`np.ndarray`*) – If not None, algorithm uses larger image to estimate sky background.
- **`imageSource`** (*`str, optional`*) – Telescope source of the image. Default is SDSS

Returns

- **sky** (*float*) – Estimation of the sky background value in counts
- **sky_err** (*float*) – Error in sky background value in counts
- **fwhms** (*List[float]*) – FWHM in x and y direction of the fitted Gaussian.
- **theta** (*float*) – Angle of the fitted Gaussian in radians measured from the +ive x axis anti-clockwise

2.11 Helpers

`pawlikMorphLSST.helpers.`**`getLocation`**(*folder*)

Helper function that returns a Path object containg the output folder

Parameters **`folder`** (*`str`*) – Path to a folder for output files.

Returns **outfolder** – Path to folder where data from analysis will be saved.

Return type Path object

`pawlikMorphLSST.helpers.`**`analyseImage`**(*info: List[Union[float, str]]*, **args*) → List[Union[float, str]]

Helper function that calculates CASGM including As and AS90

Parameters **`info`** (*`List[str, float, float]`*) – List of filename, RA, and DEC

Returns A, As, AS90, C, S, gini, M20, filename , RA, DEC

Return type Tuple[float, float, float, float, float, float, float, str, float, float]

`pawlikMorphLSST.helpers.`**`getFiles`**(*file: str*)

`pawlikMorphLSST.helpers.`**`getFilesLSST`**(*file: str*, *folder: str*)

2.12 Diagnostics

`pawlikMorphLSST.diagnostic.`**`make_figure`**(*result: Type[pawlikMorphLSST.result.Result]*, *folder: bool*, *save=False*, *show=False*) → None

Function plots results from image analysis.

Plots two or four images. Top row: original image and object map with stars overplotted if any. bottom row: Sersic fit and residual with stars overplotted if any.

Parameters

- **`result`** (*`Type[Result]`*) – Data class container of calculated results. Must have clean image and pixelmap in order to run this function.
- **`folder`** (*`bool`*) – If True then adjusts path to read file from.
- **`save`** (*`bool, optional`*) – If true function saves generated figure.
- **`show`** (*`bool, optional`*) – If true open interactive matplotlib plot.

Returns

Return type None

CHAPTER

THREE

MODULES (API)

- modindex
- genindex