imcascade

Release 0.1

Tim Miller & Pieter van Dokkum

Apr 07, 2023

CONTENTS:

Welcome to the documentation for `imcascade`. It is a non-parametric framework for fitting astronomical images to study the morphological properties of galaxies and other objects. This is accomplished by modelling them as a series (or cascade) of Gaussians

If you are planning on using `imcascade`, it is strongly recommended to read the paper describing the method, available here

CHAPTER ONE

INSTALLATION

The source code for `imcascade` is stored on github

To install `imcascade` simply clone the github repo and run the setup.py install script. This is the best way to make sure you are using the most up to date version.

```
$ cd < Directory where imcascade will be installed >
$ git clone https://github.com/tbmiller-astro/imcascade
$ cd imcascade
$ python setup.py install
```

We have also uploaded our code to PyPI so you can install `imcascade` with pip

```
$ pip install imcascade
```

1.1 Dependencies

`imcascade` is written purely in Python and requires the following packages, all of which can be installed using either `pip install` or `conda install`

- `numpy`
- `scipy`
- `matplotlib`
- `astropy`
- `numba`
- `sep`
- `dyensty`
- `asdf`

Optionally, you can also install `jax`, this speeds up the express sampling by a decent margin. It is left as optional as it cannot currently be installed easily on all systems. To install jax follow the instructions here.

`imcascade` was developed and tested using Python 3, but it may also work in Python 2 if the required packages can be installed, but be careful that it has not been properly vetted.

CHAPTER

TWO

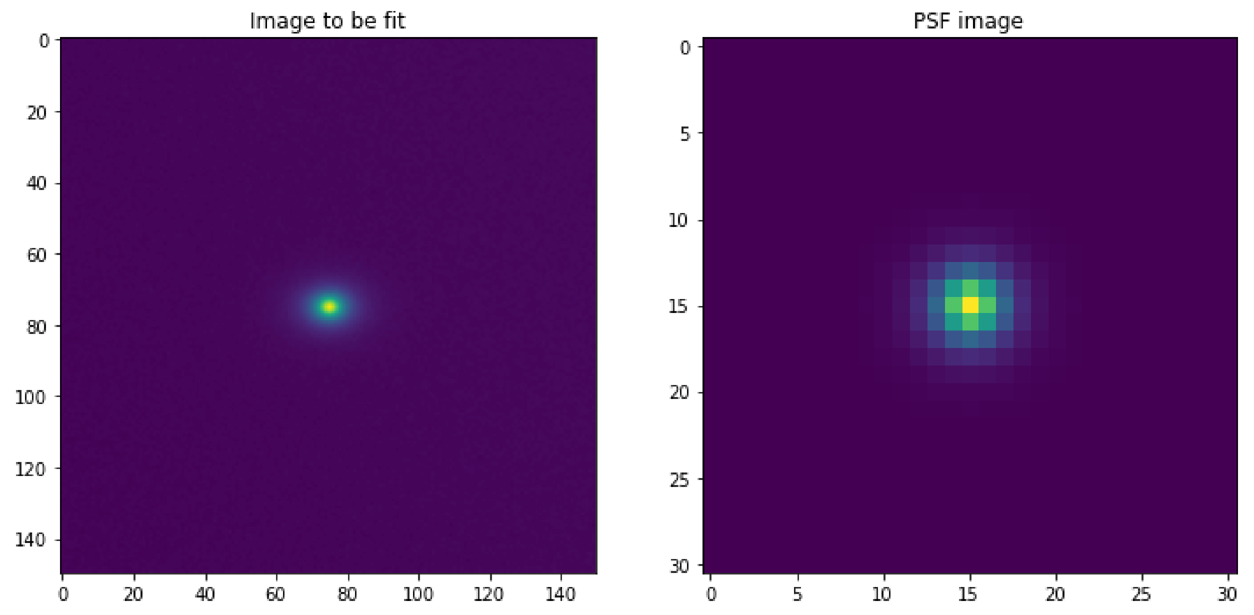
QUICKSTART GUIDE

`imcascade` is a method to fit sources in astronomical images. This is accomplished using the multi-Gaussian expansion method which models the galaxy as a mixture of Gaussians. For more details please read the in-depth example. What follows is a (very) brief introduction to the basic usage of `imcascade`.

In this short example we will fit an analytic, circular, Sersic profile with $n = 1.5$, $r_e = 6$ and total flux, $F = 250$, we have convolved the profile with a Moffat PSF with $\alpha = 3$ and $\gamma = 3$ and added purely Gaussian noise.

In a hidden cell I have intialized the cutout in the 2D array `sci` and the pixelized PSF saved in the 2D array `psf`. Below I show to 2D images of each

[2]:
```
fig,(ax1,ax2) = plt.subplots(1,2, figsize = (12,6))

ax1.imshow(sci)
ax1.set_title('Image to be fit')

ax2.imshow(psf)
ax2.set_title('PSF image')

plt.show()
```



The `initialize_fitter` function is designed to take a pixelized science images (as 2D arrays or fits files) and help initalize a `Fitter` instance which will be used below to fit galaxies. This function is designed to help users get started using our experiences to help guide some of the decisions, which may not be applicable in all situations. For more details about these choices or other options please see *the in depth example* for a longer discussion about all possibilities.

```
[3]: from imcascade.fitter import initialize_fitter

fitter = initialize_fitter(sci,psf)
```

```
2022-07-12 17:48:19,219 - Fit PSF with 4 components
2022-07-12 17:48:19,220 - Widths: 1.2,1.81,5.28,2.93
2022-07-12 17:48:19,221 - Fluxes: 0.27,0.51,0.03,0.19
2022-07-12 17:48:19,231 - Using 9 components with logarithmically spaced widths to fit␣
↪galaxy
2022-07-12 17:48:19,232 - 0.91, 1.52, 2.55, 4.27, 7.15, 11.97, 20.04, 33.54, 56.15
2022-07-12 17:48:19,233 - No mask was given, derriving one using sep
2022-07-12 17:48:19,246 - Using sep rms map to calculate pixel weights
```

This function uses the `psf_fitter` module to fit the given pixelized psf with a sum of Gaussians, which is required for our method. Next it estimates the effective radius and uses nine logarithmically spaced widths for the Gaussian components ranging from the PSF HWHM to $10 \times r_e$. It then derrives pixel weights and masks using sep (or the gain,exposure time and readnoise to calculate the rms). There are also options to use pre-calculated version of these if the user has them.

Now we will run our least-squares minimization routine

```
[4]: opt_param = fitter.run_ls_min()
print (opt_param)
```

```
2022-07-12 17:48:21,366 - Running least squares minimization
2022-07-12 17:48:47,347 - Finished least squares minimization
```

```
[ 7.60051373e+01  7.50005090e+01  6.87265676e-01  1.57715338e+00
  1.42502826e+00 -2.57490313e+00  1.53411116e+00  1.81786238e+00
  1.87217228e+00  1.63625068e+00  7.22478506e-01 -1.31211431e+00
  4.65480859e-01 -2.74388677e-05 -9.73687762e-05  1.91394558e-04]
```

We have printed out the parameters the desribe the best fit model. The first four are the structural parameters (x position, y position, axis ratio and position angle) then the next nine represent the fluxes or weights of the nine components (in logarithmic space) and the final three parameters for the tilted-plane sky model.

Obviously this is non-trivial to parse, which is why we will use our `results` module and the `ImcascadeResults` class to help us analyze the results

```
[7]: from imcascade.results import ImcascadeResults
res_class = ImcascadeResults(fitter)
res_class.run_basic_analysis()
```

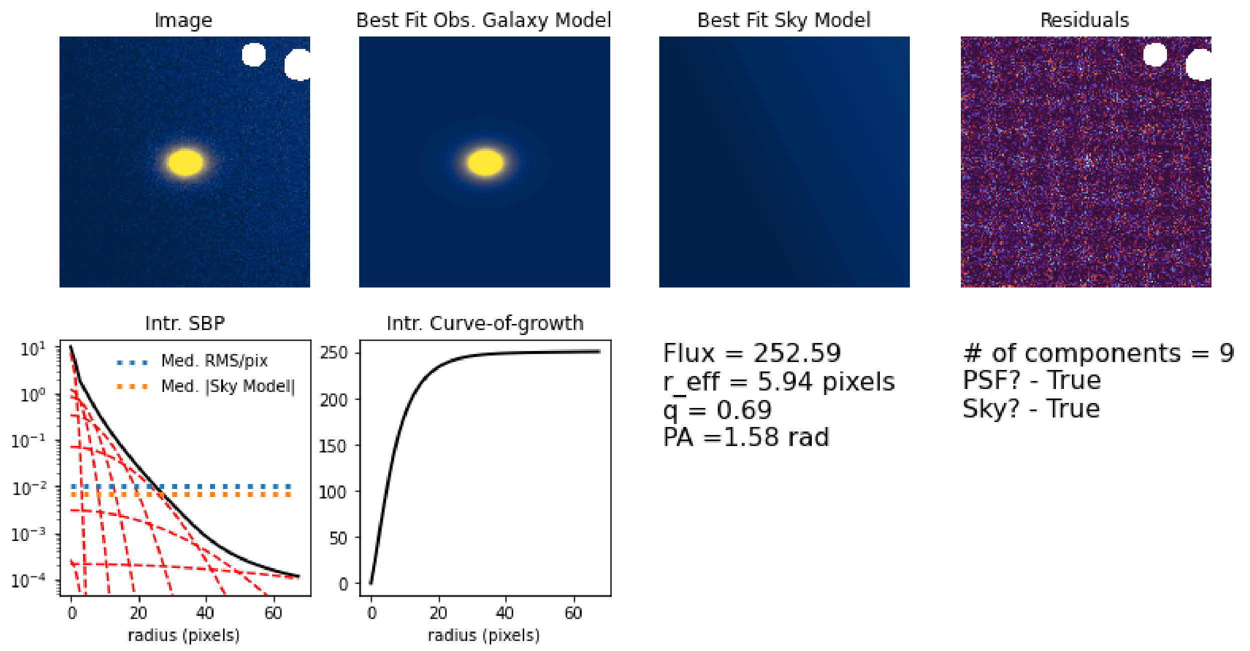
```
[7]: {'flux': 252.58989931777214,
 'r20': 2.305274076411796,
 'r50': 5.940674533616872,
 'r80': 12.340978912963354,
 'r90': 17.43337150701717,
 'C80_20': 5.353367323755417,
 'C90_50': 2.934577783779577}
```

The function `.run_basic_analysis()` calculates some basic morphological quantities like the total flux and half-light radius. We can see that the best fit parameters match the inputs pretty well!

Another very useful function is the `.make_diagnostic_fig()` this makes a figure which helps inspect the fit

```
[8]: fig = res_class.make_diagnostic_fig()
```

```
/mnt/c/Users/timbl/Documents/files/research/packages/imcascade/imcascade/results.py:707:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  rms_med = np.median(1./np.sqrt(fitter.weight) )
```



```
Flux = 252.59
r_eff = 5.94 pixels
q = 0.69
PA =1.58 rad
```

```
# of components = 9
PSF? - True
Sky? - True
```

This makes it easier to see if the fit went catastrophically wrong. This fit looks pretty good! Some examples of issues are if the curve-of-growth does not converge or if there are systematic issues in the residuals. To remedy this one could try using a different set of widths for the components, altering the inital guesses or making sure all the input data is correct.

Next we will explore the Posterior distribution using Dynesty. Specifically we will use the 'express' method which uses pre-rendered images to help speed up the run time. The code also automatically checks to see if the package `jax` is installed. This additionally helps to speed up the computation if availible.

```
[ ]: post = fitter.run_dynesty(method = 'express')
```

Now if we re-initialize the results class, we can calculate uncertainties on the morphological values.

```
[9]: res_class_w_post = ImcascadeResults(fitter)
     res_class_w_post.run_basic_analysis()
```

```
[9]: {'flux': array([249.31304659,   1.36354565,   1.27251959]),
      'r20': array([2.3907311 , 0.01458799, 0.01400204]),
      'r50': array([5.88510278, 0.03870749, 0.03532017]),
      'r80': array([12.00401554,  0.13302446,  0.1238212 ]),
      'r90': array([16.70281417,  0.28503741,  0.2769434 ]),
      'C80_20': array([5.0199006 , 0.03832022, 0.03772956]),
      'C90_50': array([2.83774779, 0.03312112, 0.03330771])}
```

CHAPTER THREE

IN-DEPTH EXAMPLE

In this notebook we will be going through an example of running `imcascade` in a realistic setting and discussing issues along the way

[1]:
```
#Load all neccesary packages
import numpy as np
import matplotlib.pyplot as plt
import time
import sep

import astropy.units as u
from astropy.coordinates import SkyCoord
```

For this example we will be running `imcascade` on HSC data on a MW mass galaxy at z=0.25. The data is attained in the cell below and is retrieved using the `unagi` python package availible *here*, written by Song Huang. The cell below is used to download the data and the PSF

[2]:
```
from unagi import hsc
from unagi.task import hsc_psf,hsc_cutout
pdr2 = hsc.Hsc(dr='pdr2',rerun = 'pdr2_wide')

#Downloaded from HSC archive, this a MW mass galaxy at z~0.25 at the sky location below
ra,dec = 219.36054754*u.deg, -0.40994375*u.deg
examp_coord = SkyCoord(ra = ra, dec = dec)
cutout = hsc_cutout(examp_coord, cutout_size=20*u.arcsec, filters='i', archive = pdr2,␣
↪dr = 'pdr2', verbose=True, variance=True, mask=True, save_output = False)
psf = hsc_psf(examp_coord, filters='i', archive=pdr2, save_output = False)

#Retrieve science and variance images
img = cutout[1].data.byteswap().newbyteorder()
var = cutout[3].data.byteswap().newbyteorder()
psf_data = psf[0].data
```

```
# Get table list from /home/tbm/anaconda3/envs/py3/lib/python3.8/site-packages/unagi/
↪data/pdr2_wide/pdr2_wide_tables.fits
# Retrieving cutout image in filter: i
# Retrieving coadd PSF model in filter: i
```

3.1 Setting up

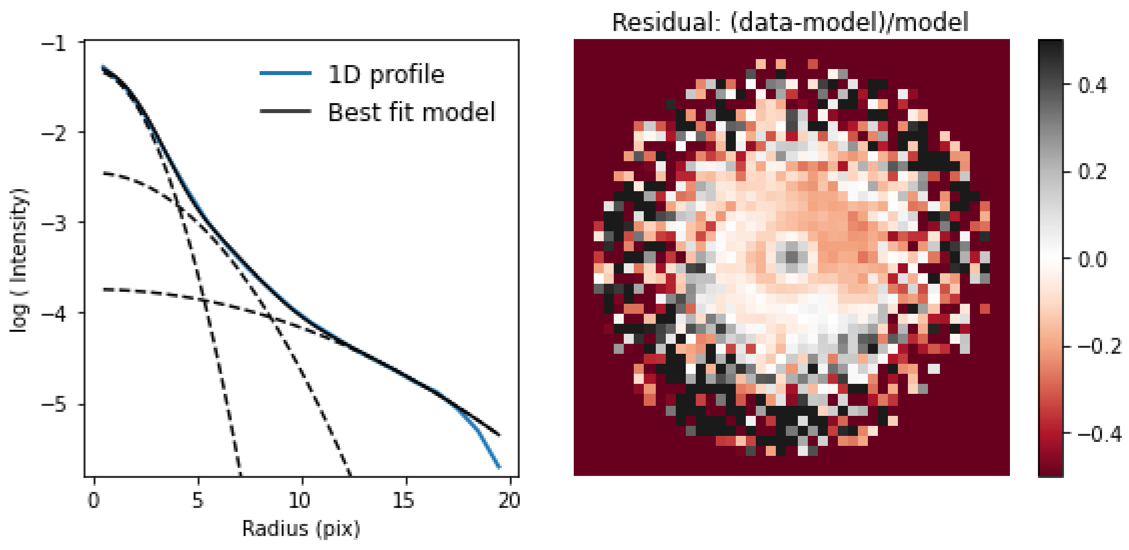
3.1.1 Modelling the PSF

To use `imcascade` while accounting for the PSF, you need to have a Gaussian decomposition of the PSF. While this is availible for some surveys, you can use the `imcascade.psf_fitter` module to help if you have a pixelized version.

The following function first fits the PSF profile in 1D to decide what the best widths are. Then a 2D fit is used to find the correct weights

[3]:
```
from imcascade.psf_fitter import PSFFitter
psf_fitter = PSFFitter(psf_data)
psf_sig,psf_a,chi2, fig = psf_fitter.fit_N(3, plot = True) # Can choose number
print (psf_sig,psf_a)
plt.show()
```

```
[1.56676938 3.15624179 7.16928138] [0.72289466 0.21654144 0.05830564]
```

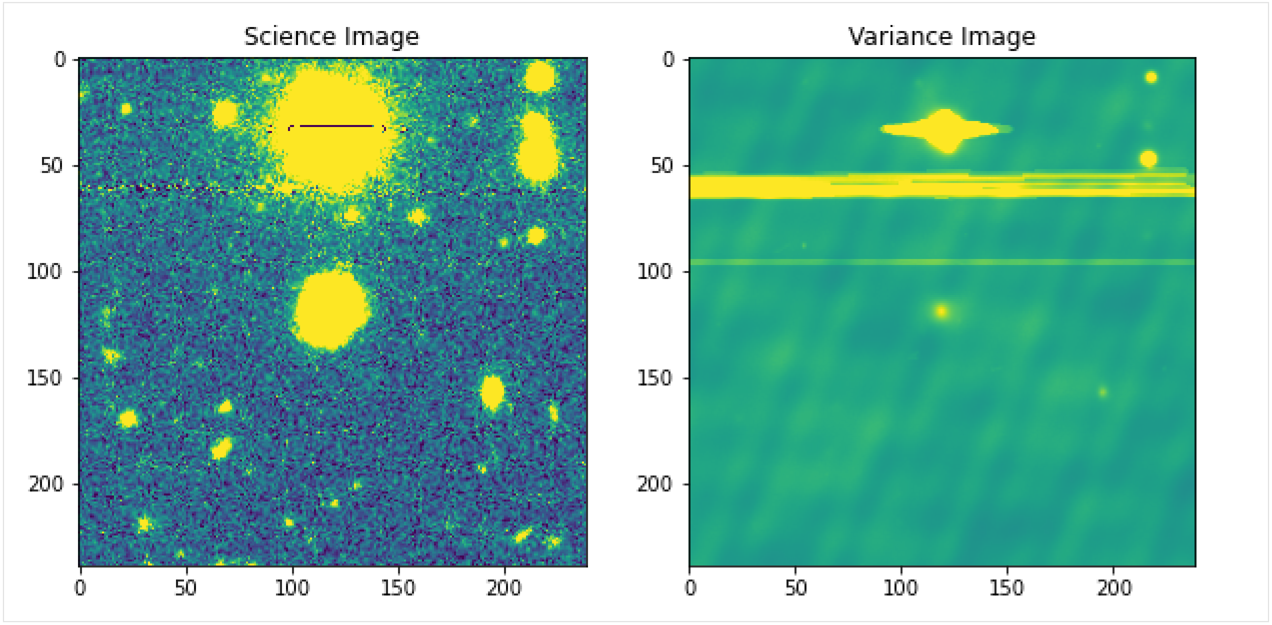


We can see that a model with three gaussians provides a pretty good fit! Generally we find 2-3 components works well for standard ground based telescopes and for more complicated PSFs, like HST WFC3, we find 4 works well. There is some incentive to use a small number of gaussians to define the PSF as it decreasese the time to render a model image. Additionally it is good to check that the sum of the weights, `psf_a`, is close to one. This ensures the PSF, and the fit are properly normalized

3.1.2 Organizing all the inputs

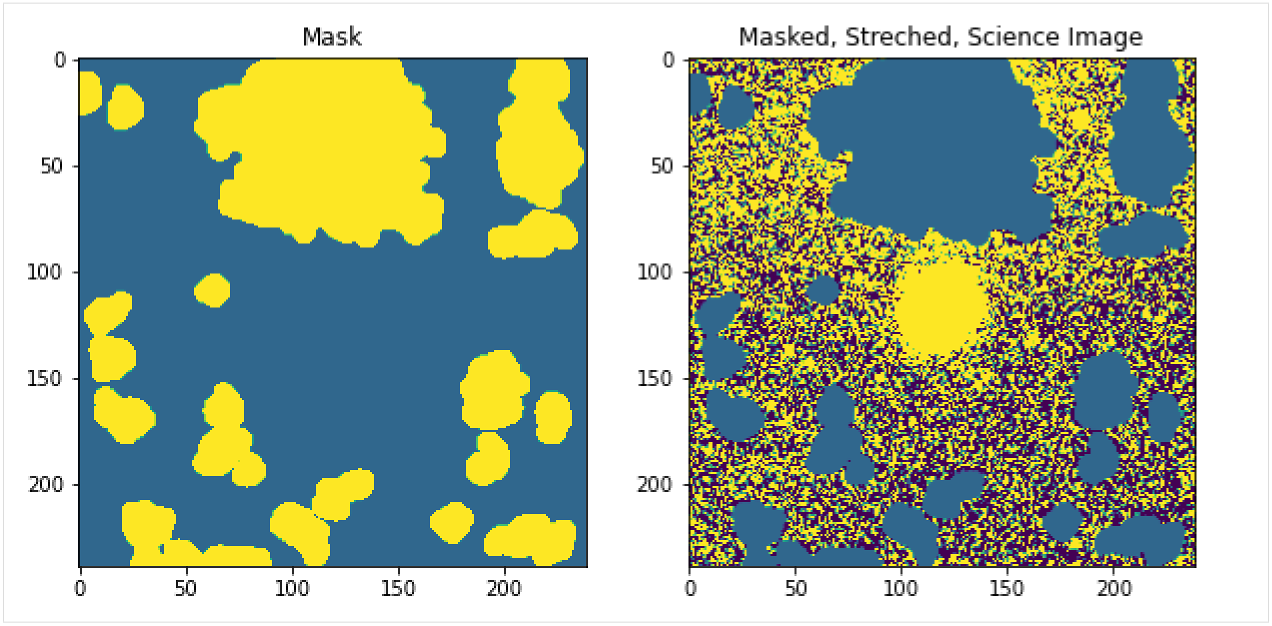
First let's take a quick look at the science and variance images. We will be fitting a model to the science image and the inverse of the variance image will be used as the pixel weights when fitting

[4]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize = (10,5))
ax1.imshow(img, vmin = -0.1, vmax = 0.2)
ax1.set_title('Science Image')
ax2.imshow(var,vmin = 0, vmax = 0.005)
ax2.set_title('Variance Image')
plt.show()
```



Additionally we will be building a mask to mask contaminating sources that we don't want affecting the fit

```
[5]: # Use sep to detect sources
bkg = sep.Background(img)
x_cent,y_cent = int(img.shape[0]/2.) , int(img.shape[1]/2.)
obj,seg = sep.extract(img - bkg.back(), 1.5, err = np.sqrt(var), deblend_cont = 0.005,␣
↪segmentation_map = True)
seg[np.where(seg == seg[x_cent,y_cent])] = 0
mask_raw = seg > 0

#Convolve mask with a gaussian to expand it and make sure that all low-SB emission is␣
↪masked
from imcascade.utils import expand_mask
mask = expand_mask(mask_raw, radius = 1.5)
mask = np.array(mask, dtype = bool)


fig, (ax1,ax2) = plt.subplots(1,2, figsize = (10,5))
ax1.imshow(mask, vmin = -0.1, vmax = 0.2)
ax1.set_title('Mask')
ax2.imshow(img*np.logical_not(mask),vmin = -0.01, vmax = 0.02)
ax2.set_title('Masked, Streched, Science Image')
plt.show()
```



3.1.3 Choosing the widths for the Gaussian components

The next major decision is what set of widths to use for the Gaussian components. In general we reccomend logarithmically spaced widths. This means there are more components are smaller radii where the signal is the largest and the profile changes the quickest. asinh scaling can also work.

Next we have to choose start and end points. This should be 0.75-1 pixel (or half the PSF width) to roughly 8-10 times the effective radius. The estimate of the effective radius does not need to be perfect, for example the Kron radius for sep or Sextractor works well. This should help decide the size of the cutout too. In order to properly model the sky the cutout size should be at least 3-4 times larger then the largest width, so 30-40 times the effective radius.

Finally we have to choose the number of components. In our testing somewhere around 9-11 seems to work.

Note: These decisions are not trivial and can affect on the outcome of an `imcascade` fit. However reasonable changes withn the confines discussed here shouldn't greatly affect the results. You should run tests to ensure you have chosen a reliable set of widths. If the results are very sensitive to the choice of widths, you should be wary and there may be other issues at play.

In this example we estimate the effective radius to be roughly 6 pixels so we use 9 components with logarithmically spaced widths from 1 pixels to 60 pixels (~10 x r eff) and use a cutout size of 240 pixels, roughly 40 times the effective radius.

```
[6]: sig = np.logspace(np.log10(1),np.log10(60), num = 9)
```

We can also specify inital conditions to help make inital guesses. Here we specify the estimated half light radii and total flux. The code make some intelligent guesses on the inital conditions and the bounds but this may help ensure a quick and reliable fit. It is also possible to specify guesses and bounds for individual components, sky values etc. but this is more involved. See the user's guide for more details

```
[7]: init_dict  = {'re':6., 'flux': 1000.}
```

3.2 Running imcascade

3.2.1 Least squares-minimization

To run imcascade we first need to intialize a `Fitter` instance with all the inputs discussed above. This class organizes all the data and contains all the methods used to fit the image

```
[8]: from imcascade import Fitter
fitter = Fitter(img,sig, psf_sig, psf_a, weight = 1./var, mask = mask, init_dict = init_
↪dict)
```

Now we can run least squares minimization using the command below

```
[9]: min_res = fitter.run_ls_min()
```

```
2022-06-22 12:30:26,609 - Running least squares minimization
2022-06-22 12:30:53,368 - Finished least squares minimization
```

```
[10]: print (min_res)
```

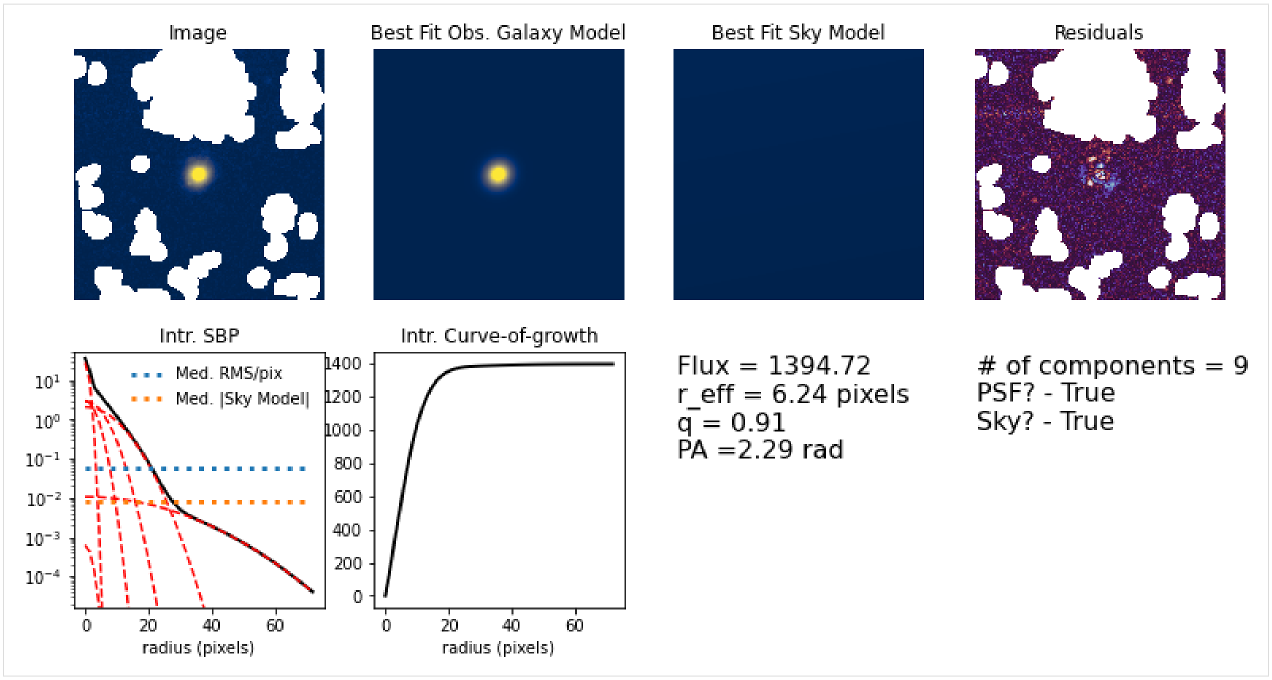
```
[ 1.20224028e+02  1.18955019e+02  9.11389758e-01  2.29047642e+00
  2.20703755e+00 -1.99999486e+00  2.13014635e+00  2.55804376e+00
  2.85067400e+00 -1.99999840e+00  1.44969096e+00 -1.99999947e+00
 -1.99999996e+00  7.17048936e-03 -1.24392340e-04 -2.20482392e-05]
```

Here we have printed out the best-fit parameters. They are, in order, x_0,y_0, axis ratio and position angle. Than the next 9 values are the best fit weights for the Gaussian components. Note that by default `imcascade` explores these in log scale. This can be changes by passing the option `log_weight_scale = False` when intializing. The final three parameters describe the best fit tilted-plane sky model. The can also be disabled when intializing with `sky_model = False` or can be set to a flat sky model with `sky_type = 'flat'`.

These aren't super easy to parase as is, which is why we use the `ImcascadeResults` class, described below

```
[11]: from imcascade import ImcascadeResults

#Initialized using `imcascade.Fitter.fitter` instance
res_class = ImcascadeResults(fitter)
fig = res_class.make_diagnostic_fig()
```

```
/mnt/c/Users/timbl/Documents/files/research/packages/imcascade/imcascade/results.py:697:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  rms_med = np.median(1./np.sqrt(fitter.weight) )
```



Here we have used the `make_diagnostic_fig()` function to help diagnose the fit. This figures shows the (masked) observed image, best fit model, sky and residuals. Along with some facts about the fit and the intrinsic surface brightness profile and curve-of-growth. From this it is easy to diagnose if something has gone catastrophically wrong. Such as: Major source not masked properly, poor convergence on fit, or the COG not asymptoting. Here we can see the fit performs pretty well, with no major issues!

3.2.2 Posterior estimation

Below we show the commands that could be used to use Bayesian techniques to explore the posterior distributions. Specifically we are using the "express" method discussed in the paper based on pre-rendered images. There are additional options when running `dynesty` using `imcascade`. Specifically we offer two choices of priors, the default is based on the results of the least-squares minimization, the other being uniform priors. We found the former runs quicker and more reliably as the priors are not as broad. It is also possible to set you own priors, see the Advanced section for more details.

```
> fitter.run_dynesty(method = 'express')
> fitter.save_results('./examp_results.asdf')
```

This is much quicker than the traditional method. However it still took about 15 min to run on my laptop. So I have run it previously and we will load the saved data.

3.3 Analyzing the results

Since when using `imcascade` the paramters that are fit are the fluxes of each Gaussian component, the analysis is more involved then other parameterized models, which fit directly for the total flux, radius etc. To assist in this we have written the `results` module and the `ImcascadeResults` class. This class can be initialized multiple ways. First you can pass it a `Fitter` instance after running `run_ls_min()` and/or `run_dynesty()`. Alternatively it can be passed a string which denotes the location of an ASDF file of the saved results

```
[12]: #Initialized with saved file
res_class_w_post = ImcascadeResults('examp_results.asdf')
```

`ImcascadeResults` will default to using the posterior to derrive morphological parameters if it is availible.

There are a number of functions we have written to calculate various morpological quantities, please see the API reference for all functions. For a lot of applications, one can simple run `run_basic_analysis()` which calculates a series of common morpological quantities

```
[13]: #we can also specify the percentiles used to calculate the error bars, here we use the␣
↪5th-95th percentile
res_class_w_post.run_basic_analysis(zpt = 27, errp_lo = 5, errp_hi = 95)
```

```
[13]: {'flux': array([1392.30063682,    8.20183351,    8.24031006]),
 'mag': array([1.91406674e+01, 6.40697201e-03, 6.41482156e-03]),
 'r20': array([2.45196295, 0.01713131, 0.01708627]),
 'r50': array([6.22617613, 0.03933737, 0.04009991]),
 'r80': array([11.42789865,  0.11402678,  0.11763424]),
 'r90': array([14.71073612,  0.22744139,  0.23404145]),
 'C80_20': array([4.66096268, 0.02856234, 0.02881198]),
 'C90_50': array([2.36285489, 0.02340729, 0.02374323])}
```

In addition to these integrated quantities, we can calculate the surface brightness profile and curve-of-growth as a function of semi-major axis

```
[14]: #Set of radii to calculate profiles at
rplot = np.linspace(0, 50, num = 100)

#Here we will calculate the posterier distribution of the surface_brightness profile
sbp_all = res_class_w_post.calc_sbp(rplot)
print (sbp_all.shape)

#Here we calculate the curve-of-growth for the posterier
cog_all = res_class_w_post.calc_cog(rplot)
```

```
(100, 29206)
```

If you use `res_class_w_post` where the postior distribution is availible it will return a 2D array containing the SBP of each sample of the posterior.

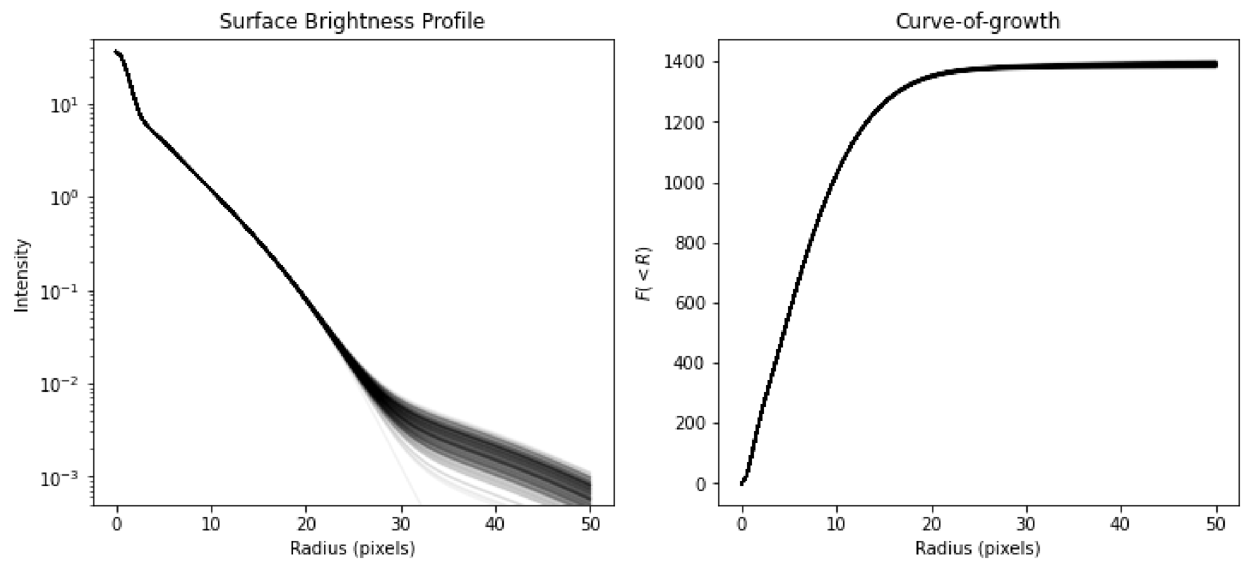
Now we plot many grey lines which show individual samples from the posterior

```
[15]: fig, (ax1,ax2) = plt.subplots(1,2, figsize = (12,5))
ax1.plot(rplot, sbp_all[:,::100], 'k-', alpha = 0.05)

ax1.set_yscale('log')
ax1.set_ylim([5e-4,5e1])
```

(continues on next page)

(continued from previous page)

```
ax1.set_title('Surface Brightness Profile')
ax1.set_xlabel('Radius (pixels)')
ax1.set_ylabel('Intensity')

ax2.plot(rplot,cog_all[:,::100], 'k-', alpha = 0.05)
ax2.set_title('Curve-of-growth')
ax2.set_xlabel('Radius (pixels)')
ax2.set_ylabel(r'$F(<R)$')
plt.show()
```



If you are interested in morphological quantity that is not included, it is likely that it will be easy to calculate and code up, so please contact us!

CHAPTER

FOUR

ADVANCED

A guide of more advanceded and in-depth features and possibilities when runnning `imcascade`. Currently in progress of being written

If you have additional questions please feel free to reach out to me or raise an issue on github!

4.1 Adjusting inital values and bounds

By default *imcascade* attempts to make ‘’ smart ‘’ guesses for the initial values and bounds based on the input variables. These can be easily changed by using the `init_dict` or `bounds_dict` arguments when intializing a `Fitter` instance These are both dictionaries, with matching keys, that will be discussed below, where each key should match to a float for `init_dict` and 2 length tuple or list for `bounds_dict`.

- `x0` and `y0` - The central location of the object. By default this intial value is the center of the image and the bounds are +/- 10 pixels
- `phi` - The position angle in radians, default is $\pi/2$ with bounds 0 to π
- `q` - The axis ratio initial value is 0.5 and bounds are 0 to 1

The inital values and boudns for the fluxes of each gaussian components can be adjusted in several ways. The easist is the specify ‘re’ and ‘flux’ in `init_dict`. This uses a polynomial fit to the exponential relationship derrived in Hogg & Lang (2013) to guess the inital values. This lower bounds on the flux of each component is then $flux/10^5$ (or 0 if using linear weight scaling) and the upper bound is the `flux` value.

You can also specify using the keys `a_max` and `a_min` in `bounds_dict` to directly specify the upper and lower bound for the flux of every component. Note that these need to be specified in logarithmic scale if exploring the fluxes in log scale. These will override the bounds discussed above.

Finally you can specify `a_i`, where i is the component of interest, in `init_dict` and `bounds_dict` to make changes to specific components.

Again these will need to be in logarithmic scale if using `log_weight_scale = True`

If you are using the tilted plane sky model, you can additionally specify those parameters

- `sky0` - Overal sky background, estimated as the median of the image
- `sky1` - X slope of sky, estimated using the edges of the image
- `sky2` - Y slope of sky, estimated using the edges of the image

4.2 Fitting options

imcascade utilized previously written routines for the optimization procedures. For the least squares minimization we use the scipy.optimize.least_squares routine. when using run_ls_min() the ls_kwargs input can be used to specify keywords to be passed to the least_squares function.

Similarly we implemented dynesty for Bayesian inference. You can pass arguments through run_dynesty() for when defining the sampler using sampler_kwargs or running the nested sampling using run_nested_kwargs(). Both of these should be dictionaries.

We have found the default options for both these packages work fairly well however performce can likely be increased by tweaking one or more parameters when using these packages.

4.3 Model averaging

As discussed in our paper, a powerful extension of imcascade is to run Bayesian inference on a galaxy multiple times with different using different set of widths. These can then be combined using a Bayesian model averaging approach. For ease of use we have written a class MultiResults in the results module. The input for this class is a list of ImcascadeResuts instances which are meant meant to be combined. It contains some (but not all) function contained in ImcascadeResuts and calculates the joint posterier distribution of these quantities by weighting each model (i.e. set of widths) by their relative evidence. It is important to make sure you are using results run on the same images etc. or else it will produce non-sensical results.

4.4 Non-circular PSF

As a extension to the normal mode with a circular PSF, we have implemented the ability to specify a non-circular PSF. To do this use the parameter PSF_shape when initializing a Fitter instance. This should be a dictionary with the keys q for the axis ratio and phi for the position angle. Currently all components of the psf must have the same shape parameters

This increases the time to render a model as no each individual component in the observed profile must be rotated individually as they will have different position angles. **It has also not been thoroughly tested so use with caution!**

4.5 Changing the Likelihood function

In our implementation of imcascade we have assumed Gaussian statistics and errors when using the usual χ^2 method. However, In some use cases (or just in general, see Erwin (2015)) it is preferable to assume Poisson statistics and use the Cash statistic as the likelihood (Cash (1979)). For the “express” method the function that gets called to calculate the likelihood is log_like_express, we can simply re-assign the to a function of our choosing, using the setattr method built in to python. First we have initialized a Fitter instance under the name fitter_cash as usual (but with the pixel weights equal to one) and then we run the following code block

```python
fitter_cash = initialize_fitter(img, psf, sky_model = False, err = np.ones(img.shape))

def log_like_cash(self, exp_params):
    "log likelihood using the Cash statistic"

    #Use this function to generate model
    model = self.make_express_model(exp_params)
```

(continues on next page)

(continued from previous page)

```
    #Employing the Cash statistic
    return -1.*np.sum( self.weight *(model - self.img*np.log(model)) )

setattr(fitter_cash, 'log_like_express', log_like_cash)
```

In this example we have written our on function to replace the original function. Now when we use `run_dynesty` it will call our new function instead. The function `self.make_express_model` generates the model image and the self weight variable contains the pixel weights (which we have set to 1.) and any mask we have supplied. `self.img` contains the input science cutout.

Using this example as a blueprint, it is possible to change the likelihood function to anything your heart desires! It is important to test whatever function you have used to make sure the answer is what you expect.

4.6 Changing the Priors

CHAPTER

FIVE

API REFERENCE

This page contains auto-generated API reference documentation[1].

5.1 `imcascade`

imcascade: Fitting astronomical images using a 'cascade' of Gaussians

5.1.1 Submodules

`imcascade.fitter`

Module Contents

Classes

Fitter	A Class used fit images with MultiGaussModel

Functions

initialize_fitter(im, psf[, mask, err, x0, y0, re, ...])	Function used to help Initialize Fitter instance from simple inputs
fitter_from_ASDF(file_name[, init_dict, bounds_dict])	Function used to initalize a fitter from a saved asdf file

[1] Created with sphinx-autoapi

Attributes

log2pi

imcascade.fitter.**log2pi**

imcascade.fitter.**initialize_fitter**(*im*, *psf*, *mask=None*, *err=None*, *x0=None*, *y0=None*, *re=None*, *flux=None*, *psf_oversamp=1*, *sky_model=True*, *log_file=None*, *readnoise=None*, *gain=None*, *exp_time=None*, *num_components=None*, *component_widths=None*, *log_weight_scale=True*)

Function used to help Initialize Fitter instance from simple inputs

Parameters

- **im** (*str or 2D Array*) – The image or cutout to be fit with imcascade. If a string is given, it is interpretted as the location of a fits file with the cutout in it's first HDU. Otherwise is a 2D numpy array of the data to be fit
- **psf** (*str, 2D Array or None*) – Similar to above but for the PSF. If not using a PSF, the use None
- **mask** (*2D array (optional)*) – Sources to be masked when fitting, if none is given then one will be derrived
- **err** (*2D array (optional)*) – Pixel errors used to calculate the weights when fitting. If none is given will use readnoise, gain and exp_time if given, or default to sep derrived rms
- **x0** (*float (optional)*) – Inital guess at x position of center, if not will assume the center of the image
- **y0** (*float (optional)*) – Inital guess at y position of center, if not will assume the center of the image
- **re** (*float (optional)*) – Inital guess at the effective radius of the galaxy, if not given will estimate using sep kron radius
- **flux** (*float (optional)*) – Inital guess at the flux of the galaxy, if not given will estimate using sep flux
- **psf_oversamp** (*float (optional)*) – Oversampling of PSF given, default is 1
- **sky_model** (*boolean (optional)*) – Whether or not to model sky as tilted-plane, default is True
- **log_file** (*str (optional)*) – Location of log file
- **readnoise,gain,exp_time** (*float,float,float (all optional)*) – The read noise (in electrons), gain and exposure time of image that is used to calculate the errors and therefore pixel weights. Only used if `err = None`. If these parameters are also None, then will estimate pixel errors using sep rms map.

Returns **Fitter** – Returns intialized instance of imcascade.fitter.Fitter which can then be used to fit galaxy and analyze results.

Return type *imcascade.fitter.Fitter*

imcascade.fitter.**fitter_from_ASDF**(*file_name*, *init_dict={}*, *bounds_dict={}*)

Function used to initalize a fitter from a saved asdf file

This can be useful for re-running or for initializing a series of galaxies beforehand and then transferring to somewhere else or running in parallel

Parameters

- **file_name** (*str*) – location of asdf file containing saved data. Often this is a file created by Fitter.save_results
- **init_dict** (*dict (optional)*) – Dictionary specifying initial guesses for least_squares fitting to be passed to Fitter instance.
- **bounds_dict** (*dict (optional)*) – Dictionary specifying bounds for least_squares fitting to be passed to Fitter instance.

class `imcascade.fitter.`**`Fitter`**(*img*, *sig*, *psf_sig*, *psf_a*, *weight=None*, *mask=None*, *sky_model=True*, *sky_type='tilted-plane'*, *render_mode='hybrid'*, *log_weight_scale=True*, *verbose=True*, *psf_shape=None*, *init_dict={}*, *bounds_dict={}*, *log_file=None*)

Bases: `imcascade.mgm.MultiGaussModel`

A Class used fit images with MultiGaussModel

This is the main class used to fit `imcascade` models

Parameters

- **img** (*2D Array*) – Data to be fit, it is assumed to be a cutout with the object of interest in the center of the image
- **sig** (*1D Array*) – Widths of Gaussians to be used in MultiGaussModel
- **psf_sig** (*1D array, None*) – Width of Gaussians used to approximate psf
- **psf_a** (*1D array, None*) – Weights of Gaussians used to approximate psf If both psf_sig and psf_a are None then will run in Non-psf mode
- **weight** (*2D Array, optional*) – Array of pixel by pixel weights to be used in fitting. Must be same shape as 'img' If None, all the weights will be set to 1.
- **mask** (*2D Array, optional*) – Array with the same shape as 'img' denoting which, if any, pixels to mask during fitting process. Values of '1' or 'True' values for the pixels to be masked. If set to 'None' then will not mask any pixels. In practice, the weights of masked pixels is set to '0'.
- **sky_model** (*bool, optional*) – If True will incorperate a tilted plane sky model. Reccomended to be set to True
- **sky_type** (*str, 'tilted-plane' or 'flat'*) – Function used to model sky. Default is tilted plane with 3 parameters, const bkg and slopes in each directin. 'flat' uses constant background model with 1 parameter.
- **render_mode** (*'hybrid', 'erf' or 'gauss'*) – Option to decide how to render models. 'erf' analytically computes the integral over the pixel of each profile therefore is more accurate but more computationally intensive. 'gauss' assumes the center of a pixel provides a reasonble estimate of the average flux in that pixel. 'gauss' is faster but far less accurate for objects which vary on O(pixel size), so use with caution. 'hybrid' is the defualt, uses 'erf' for components with width < 5 to ensure accuracy and uses 'gauss' otherwise as it is accurate enough and faster. Also assumes all flux > 5 sigma for components is 0.
- **log_weight_scale** (*bool, optional*) – Wether to treat weights as log scale, Default True
- **verbose** (*bool, optional*) – If true will log and print out errors

- `psf_shape` (*dict, Optional*) – Dictionary containg at 'q' and 'phi' that define the shape of the PSF. Note that this slows down model rendering significantly so only reccomended if neccesary.
- `init_dict` (*dict, Optional*) – Dictionary specifying initial guesses for least_squares fitting. The code is desigined to make 'intelligent' guesses if none are provided
- `bounds_dict` (*dict, Optional*) – Dictionary specifying boundss for least_squares fitting and priors. The code is desigined to make 'intelligent' guesses if none are provided

`resid_1d`(*params*)

Given a set of parameters returns the 1-D flattened residuals when compared to the Data, to be used in run_ls_min Function

Parameters `params` (*Array*) – List of parameters to define model

Returns **resid_flatten** – 1-D array of the flattened residuals

Return type array

`run_ls_min`(*ls_kwargs={}*)

Function to run a least_squares minimization routine using pre-determined inital guesses and bounds.

Utilizes the scipy least_squares routine (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html)

Parameters `ls_kwargs` (*dict, optional*) – Optional list of arguments to be passes to least_squares routine

Returns **min_param** – Returns a 1D array containing the optimized parameters that describe the best fit model.

Return type 1D array

`set_up_express_run`(*set_params=None*)

Function to set up 'express' run using pre-rendered images with a fixed x0,y0, phi and q. Sets class attribute 'express_gauss_arr' which is needed to run dynesty or emcee in express mode

Parameters `set_params` (*len(4) array-like, optional*) – Parameters (x0,y0,q,phi) to use to per-render images. If None will call `run_ls_min()` or use stored `min_res` to find parameters.

Returns **express_gauss_arr** – Returns a 3-D with pre-rendered images based on input parameters

Return type array (shape[0],shape[1], Ndof_gauss)

`make_express_model`(*exp_params*)

Function to generate a model for a given set of paramters, specifically using the pre-renedered model for the 'express' mode

Parameters `exo_params` (*Array*) – List of parameters to define model. Length is Ndof_gauss + Ndof_sky since the structural parameters (x0,y0,q, PA) are set

Returns **model** – Model image based on input parameters

Return type 2D-array

`chi_sq`(*params*)

Function to calculate chi_sq for a given set of paramters

Parameters `params` (*Array*) – List of parameters to define model

Returns **chi^2** – Chi squared statistic for the given set of parameters

Return type float

`log_like`(*params*)
Function to calculate the log likeliehood for a given set of paramters

Parameters **params** (*Array*) – List of parameters to define model

Returns **log likeliehood** – log likeliehood for a given set of paramters, defined as -0.5*chi^2

Return type float

`ptform`(*u*)
Prior transformation function to be used in dynesty 'full' mode

Parameters **u** (*array*) – array of random numbers from 0 to 1

Returns **x** – array containing distribution of parameters from prior

Return type array

`log_like_exp`(*exp_params*)
Function to calculate the log likeliehood for a given set of paramters, specifically using the pre-renedered model for the 'express' mode

Parameters **exo_params** (*Array*) – List of parameters to define model. Length is Ndof_gauss + Ndof_sky since the structural parameters (x0,y0,q, PA) are set

Returns **log likeliehood** – log likeliehood for a given set of paramters, defined as -0.5*chi^2

Return type float

`ptform_exp_ls`(*u*)
Prior transformation function to be used in dynesty 'express' mode using gaussian priors defined by the results of the least_squares minimization

Parameters **u** (*array*) – array of random numbers from 0 to 1

Returns **x** – array containing distribution of parameters from prior

Return type array

`ptform_exp_lin_cauchy`(*u*)
Prior transformation function to be used in dynesty 'express' mode using gaussian priors defined by the results of the least_squares minimization

Parameters **u** (*array*) – array of random numbers from 0 to 1

Returns **x** – array containing distribution of parameters from prior

Return type array

`ptform_exp_unif`(*u*)
Prior transformation function to be used in dynesty 'express' mode using unifrom priors defined by self.lb and self.ub

Parameters **u** (*array*) – array of random numbers from 0 to 1

Returns **x** – array containing distribution of parameters from prior

Return type array

`run_dynesty`(*method='full'*, *sampler_kwargs={}*, *run_nested_kwargs={}*, *prior='min_results'*)
Function to run dynesty to sample the posterior distribution using either the 'full' methods which explores all paramters, or the 'express' method which sets the structural parameters.

Parameters

- **method** (*str: 'full' or 'express'*) – Which method to use to run dynesty
- **sampler_kwargs** (*dict*) – set of keyword arguments to pass the the dynesty DynamicNestedSampler call, see: https://dynesty.readthedocs.io/en/latest/api.html#dynesty.dynesty.DynamicNestedSampler
- **run_nested_kwargs** (*dict*) – set of keyword arguments to pass the the dynesty run_nested call, see: https://dynesty.readthedocs.io/en/latest/api.html#dynesty.dynamicsampler.DynamicSampler.run_nested
- **prior** (*'min_results' or 'uniform'*) – Which of the two choices of priors to use. The *min_results* priors are Gaussian, with centers defined by the best fit paramters and variance equal to 4 times the variance estimated using the Hessian matrix from the run_ls_min() run. *uniform* is what it sounds like, uniform priors based on the the lower and upper bounds Defualt is *min_results*

Returns **Posterior** – posterior distribution derrived. If method is 'express', the first 4 columns, containg x0, y0, PA and q, are all the same and equal to values used to pre-render the images

Return type Array

`save_results`(*file_name*)

Function to save results after run_ls_min, run_dynesty and/or run_emcee is performed. Will be saved as an ASDF file.

Parameters **file_name** (*str*) – Str defining location of where to save data

Returns **dict_saved** – Dictionary containing all the save quantities

Return type dict

`imcascade.mgm`

Module Contents

Classes

MultiGaussModel	A class used to generate models based series of Gaussians

Functions

rot_im_jax_exp(img, phi, x0, y0)	Experimental, do not use yet!
rot_im(img, phi, x0, y0)	Function to rotate image around a given point
get_ellip_conv_params(var_all, q, phi, psf_var_all, . . .)	Function used to derrive the observed Gaussian Parameters for a non-circular PSF
_erf_approx(x)	Approximate erf function for use with numba
_get_hybrid_stack(x0, y0, final_q, final_a, final_var, . . .)	Wrapper Function used to calculate render model using the hybrid method

class imcascade.mgm.**MultiGaussModel**(*shape*, *sig*, *psf_sig*, *psf_a*, *verbose=True*, *sky_model=True*, *sky_type='tilted-plane'*, *render_mode='hybrid'*, *log_weight_scale=True*, *psf_shape=None*)

A class used to generate models based series of Gaussians

Parameters

- **shape** (*2x1 array_like*) – Size of model image to generate
- **sig** (*1-D array*) – Widths of Gaussians used to genrate model
- **psf_sig** (*1-D array, None*) – Width of Gaussians used to approximate psf
- **psf_a** (*1-D array, None*) – Weights of Gaussians used to approximate psf, must be same length as 'psf_sig'. If both psf_sig and psf_a are None then will run in Non-psf mode
- **verbose** (*bool, optional*) – If true will print out errors
- **sky_model** (*bool, optional*) – If True will incorperate a tilted plane sky model
- **render_mode** (*'gauss' or 'erf'*) – Option to decide how to render models. Default is 'erf' as it computes the integral over the pixel of each profile therefore is more accurate but more computationally intensive. 'gauss' assumes the center of a pixel provides a reasonble estimate of the average flux in that pixel. 'gauss' is faster but far less accurate for objects with size O(pixel size), so use with caution.
- **log_weight_scale** (*bool, optional*) – Wether to treat weights as log scale, Default True

get_gauss_stack(*x0*, *y0*, *q_arr*, *a_arr*, *var_arr*)

Function used to calculate render model using the 'Gauss' method

Parameters

- **x0** (*float*) – x position of center
- **y0** (*float*) – y position of center
- **q_arr** (*Array*) – Array of axis ratios
- **a_arr** – Array of Gaussian Weights
- **var_arr** – Array of Gassian widths, note this the variance so sig^2

Returns **Gauss_model** – Array representing the model image, same shape as 'shape'

Return type array

get_erf_stack(*x0*, *y0*, *final_q*, *final_a*, *final_var*)

Function used to calculate render model using the 'erf' method

Parameters

- **x0** (*float*) – x position of center
- **y0** (*float*) – y position of the center
- **final_q** (*Array*) – Array of axis ratios
- **final_a** (*Array*) – Array of Gaussian Weights
- **final_var** (*Array*) – Array of Gassian widths, note this the variance so sig^2

Returns **erf_model** – Array representing each rendered component

Return type array

get_hybrid_stack(*x0*, *y0*, *final_q*, *final_a*, *final_var*)

Function used to calculate render model using the hybrid method, which uses erf where neccesary to ensure accurate integration and gauss otherwise. Also set everything >5 sigma away to 0.

Parameters

- **x0** (*float*) – x position of center
- **y0** (*float*) – y position of the center
- **final_q** (*Array*) – Array of axis ratios
- **final_a** (*Array*) – Array of Gaussian Weights
- **final_var** (*Array*) – Array of Gassian widths, note this the variance so sig^2

Returns **erf_model** – Array representing each rendered component

Return type 3D array

make_model(*param*, *return_stack=False*)
Function to generate model image based on given paramters array. This version assumaes the gaussian weights are given in linear scale

Parameters **param** (*array*) – 1-D array containing all the Parameters

Returns **model_image** – Generated model image as the sum of all components plus sky, if included

Return type 2D Array

get_sky_model_flat(*args*)
Function used to calculate flat sky model

Parameters

args: (a,) (float,)

Returns **sky_model** – Model for sky background based on given parameters, same shape as 'shape'

Return type 2D Array

get_sky_model_tp(*args*)
Function used to calculate tilted-plane sky model

Parameters

- **args** (*(a,b,c) (float,float,float)*) –
- **- overall normalization** (*a*) –
- **- slope in x direction** (*b*) –
- **- slope in y direction** (*c*) –

Returns **sky_model** – Model for sky background based on given parameters, same shape as 'shape'

Return type 2D Array

imcascade.mgm.**rot_im_jax_exp**(*img*, *phi*, *x0*, *y0*)
Experimental, do not use yet!

imcascade.mgm.**rot_im**(*img*, *phi*, *x0*, *y0*)
Function to rotate image around a given point

Parameters

- **img** (*2D array*) – Image to be rotated
- **phi** (*Float*) – angle to rotate image
- **x0** (*Float*) – x coordinate to rotate image around

- **y0** (*Float*) – y coordinate to rotate image around

Returns rotated image

Return type 2D array

imcascade.mgm.**get_ellip_conv_params**(*var_all*, *q*, *phi*, *psf_var_all*, *psf_q*, *psf_phi*)

Function used to derrive the observed Gaussian Parameters for a non-circular PSF

Parameters

- **var** (*array*) – Variances of Gaussian components
- **q** (*Float*) – Axis ratio of Galaxy
- **phi** (*Float*) – PA of galaxy
- **psf_var_all** (*array*) – Variances of PSF gaussian decomposition
- **psf_q** (*float*) – Axis ratio of PSF
- **psf_phi** (*PA of PSF*) –

Returns

- **obs_var** (*array*) – Array of variances for the components of the convolved gaussian model
- **obs_phi** (*array*) – Array of position angles for the components of the convolved gaussian model
- **obs_q** (*array*) – Array of axis ratios for the components of the convolved gaussian model

imcascade.mgm.**_erf_approx**(*x*)

Approximate erf function for use with numba

Parameters **x** (*scalar*) – value

Returns

Return type Approximation of erf(x)

imcascade.mgm.**_get_hybrid_stack**(*x0*, *y0*, *final_q*, *final_a*, *final_var*, *im_args*)

Wrapper Function used to calculate render model using the hybrid method

Parameters

- **x0** (*float*) – x position of center
- **y0** (*float*) – y position of the center
- **final_q** (*Array*) – Array of axis ratios
- **final_a** – Array of Gaussian Weights
- **final_var** – Array of Gassian widths, note this the variance so sig^2
- **return_stack** (*Bool, optional*) – If True returns an image for each individual gaussian

Returns **erf_model** – Array representing the model image, same shape as 'shape'

Return type array

`imcascade.psf_fitter`

Module Contents

Classes

PSFFitter	A Class used to fit Gaussian models to a PSF image

class imcascade.psf_fitter.**PSFFitter**(*psf_img*, *oversamp=1.0*)

A Class used to fit Gaussian models to a PSF image

Parameters

- **psf_img** (*str or 2D array*) – PSF data to be fit. If a string is given will assume it is a fits file and load the Data in the first HDU. If it is an array then will use as the PSF image. Either way it is assumed the PSF is centered and image is square.
- **oversamp** (*Float, optional*) – Factor by which the PSF image is oversampled. Default is 1.

psf_data

pixelized PSF data to be fit

Type 2D array

intens

1D sbp of PSF

Type 1D array

radii

Radii corresponding to `intens`

Type 1D array

calc_profile()

Calculates the 1-D PSF profile in 1 pixel steps assuming it is circular

Returns

- **intens** (*1D array*) – Intensity profile.
- **radius** (*1D array*) – radii at which the intensity measuremnts are made.

multi_gauss_1d(*r*, **params*, *mu=0*)

Function used to evaluate a 1-D multi Gaussian model with any number of components

Parameters

- **params** (*1D array*) – List of parameters to define model. The length should be twice the number of components in the following pattern:[a_1, sig_1, a_2,sig_2, ...] where a_i is the weight of the i'th component and sig_i is the width of the i'th component.
- **r** (*float, array*) – Radii at which the profile is to be evaluated
- **mu** (*float, optional*) – The centre of the gaussian distribution, default is 0

Returns The multi-gaussian profile evaluated at 'r'

Return type 1D array

multi_gauss_1d_ls(**params*, *x_data=np.zeros(10)*, *y_data=np.zeros(10)*, *mu=0*)

Wrapper for multi_gauss_1d function, to be used in fitting the profile

Parameters

- `params` (*1D Array*) – List of parameters to define model. The length should be twice the number of components in the following pattern: [a_1, sig_1, a_2,sig_2, ….]. Here a_i is the weight of the i'th component and sig_i is the width of the i'th component.
- `x_data` (*1-D array*) – x_data to be fit, generally self.radius.
- `y_data` (*1-D array*) – y_data to be fit, genrally self.intens
- `mu` (*float, optional*) – The centre of the gaussian distribution, default is 0.

Returns **resid** – log scale residuals between the models specified by 'params' and the given y_data

Return type 1D array

`fit_N`(*N*, *frac_cutoff=0.0001*, *plot=False*)

'Fully' Fit a Multi Gaussian Model with a given number of gaussians to the psf profile. Start with 1D to find the best fit widths and then us evaluate chi2 in 2D

Parameters

- `N` (*Int*) – Number of gaussian to us in fit
- `frac_cutoff` (*float, optional*) – Fraction of max, below which to not fit. This is done to focus on the center of the PSF and not the edges. Important because we using the log-residuals
- `plot` (*bool*) – Whether or not to show summary plot

Returns

- **a_fit** (*1-D array*) – Best fit Weights, corrected for oversamping
- **sig_fit** (*1-D array*) – Best fit widths, corrected for oversamping
- **Chi2** (*Float*) – The overall chi squared of the fit, computed using the best fit 2D model

`fit_1D`(*N*, *init_guess=None*, *frac_cutoff=0.0001*)

Fit a 1-D Multi Gaussian Model to the psf profile

Parameters

- `N` (*Int*) – Number of gaussian to us in fit
- `guess` (*Init*) – Initial guess at parameters, if None will set Default based on N
- `frac_cutoff` (*float*) – Fraction of max, below which to not fit. This is done to focus on the center of the PSF and not the edges. Important because we using the log-residuals

Returns

- **a_fit** (*1-D array*) – Best fit Weights, Not corrected for oversampling
- **sig_fit** (*1-D array*) – Best fit widths, Not corrected for oversampling
- **Chi2** (*Float*) – The overall chi squared of the fit

`auto_fit`(*N_max=5*, *frac_cutoff=0.0001*, *norm_a=True*)

Function used for automatic fitting of PSF. First using a 1-D fit to find the smallest acceptable number of Gaussians and the corresponding widths, then using these widths to fit in 2D and find the weights.

Parameters

- `N` (*Int*) – Number of gaussian to us in fit

- **frac_cutoff** (*float*) – Fraction of max, below which to not fit. This is done to focus on the center of the PSF and not the edges. Important because we using the log-residuals
- **norm_a** (*Bool*) – Wheter or not to normize the resulting weight so that the sum is unity

Returns

- **a_fit** (*1-D array*) – Best fit Weights
- **sig_fit** (*1-D array*) – Best fit widths

calc_fwhm()

Function to estimate the FHWM of the PSF, by interpolating the measured profile

Returns **FWHM** – Estimate of the FWHM in pixels

Return type float

`imcascade.results`

Module Contents

Classes

ImcascadeResults	A class used for collating imcascade results and performing analysis
MultiResults	A Class to analyze and combine multiple ImcascadeResults classes using evidence weighting

Functions

calc_flux_input(weights, sig[, cutoff])	
r_root_func(r, f_L, weights, sig, cutoff)	

imcascade.results.**calc_flux_input**(*weights*, *sig*, *cutoff=None*)

imcascade.results.**r_root_func**(*r*, *f_L*, *weights*, *sig*, *cutoff*)

class imcascade.results.**ImcascadeResults**(*Obj*, *thin_posterior=1*)

A class used for collating imcascade results and performing analysis

Parameters

- **Obj** (*imcascade.fitter.Fitter class, dictionary or str*) – Object which contains the data to be analyzed. Can be a Fitter object once the run_(ls_min,dynesty, emcee) has been ran. If it is a dictionay needs to contain, at bare minmum the variables sig, Ndof, Ndof_sky, Ndof_gauss, log_weight_scale and either min_param or posterior. If a string is passed it will be interreted as a file locations with an ASDF file containing the neccesary information.
- **thin_posterior** (*int (optional)*) – Factor by which to thin the posterior distribution by. While one wants to ensure the posterior is large enough, some of this analysis can take time if you have >10^6 samples so this is one way to speed up this task but use with caution.

calc_flux(*cutoff=None*)
Calculate flux of given results

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **Flux** – Total flux of best fit model

Return type float or Array

_min_calc_rX(*X*, *cutoff=None*)
Old and slow Function to calculate the radius containing X percent of the light

Parameters

- **X** (*float*) – Fractional radius of intrest to calculate. if X < 1 will take as a fraction, else will interpret as percent and divide X by 100. i.e. to calculate the radius containing 20% of the light, once can either pass X = 20 or 0.2
- **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width used

Returns **r_X** – The radius containg X percent of the light

Return type float or Array

calc_rX(*X*, *cutoff=None*)
Function to calculate the radius containing X percent of the light

Parameters

- **X** (*float*) – Fractional radius of intrest to calculate. if X < 1 will take as a fraction, else will interpret as percent and divide X by 100. i.e. to calculate the radius containing 20% of the light, once can either pass X = 20 or 0.2
- **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width used

Returns **r_X** – The radius containg X percent of the light

Return type float or Array

calc_r90(*cutoff=None*)
Wrapper function to calculate the radius containing 90% of the light

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **r_90** – The radius containg 90 percent of the light

Return type float or Array

calc_r80(*cutoff=None*)
Wrapper function to calculate the radius containing 80% of the light

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **r_80** – The radius containg 80 percent of the light

Return type float or Array

calc_r50(*cutoff=None*)
Wrapper function to calculate the radius containing 50% of the light, or the effective radius

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **r_50** – The radius containg 50 percent of the light

Return type float or Array

`calc_r20`(*cutoff=None*)
Wrapper function to calculate the radius containing 20% of the light

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **r_20** – The radius containg 20 percent of the light

Return type float or Array

`calc_sbp`(*r*, *return_ind=False*)
Function to calculate surface brightness profiles for the given results

Parameters

- **r** (*float or array*) – Radii (in pixels) at which to evaluate the surface brightness profile
- **return_ind** (*bool (optional)*) – If False will only return the sum of all gaussian, i.e. the best fit profile. If true will return an array with +1 dimensions containing the profiles of each individual gaussian component

Returns **SBP** – Surface brightness profiles evaluated at 'r'. If 'return_ind = True', returns the profile of each individual gaussian component

Return type array

`calc_obs_sbp`(*r*, *return_ind=False*)
Function to calculate the observed surface brightness profiles, i.e. convolved with the PSF for the given results

Parameters

- **r** (*float or array*) – Radii (in pixels) at which to evaluate the surface brightness profile
- **return_ind** (*bool (optional)*) – If False will only return the sum of all gaussian, i.e. the best fit profile. If true will return an array with +1 dimensions containing the profiles of each individual gaussian component

Returns **obsereved SBP** – Observed surface brightness profiles evaluated at 'r'. If 'return_ind = True', returns the profile of each individual gaussian component

Return type array

`calc_cog`(*r*, *return_ind=False*, *norm=False*, *cutoff=None*)
Function to calculate curves-of-growth for the given results

Parameters

- **r** (*float or array*) – Radii (in pixels) at which to evaluate the surface brightness profile
- **return_ind** (*bool (optional)*) – If False will only return the sum of all gaussian, i.e. the best fit profile. If true will return an array with +1 dimensions containing the profiles of each individual gaussian component
- **norm** (*Bool (optional)*) – Wether to normalize curves-of-growth to total flux, calculated using 'self.calc_flux'. Does nothing if 'return_ind = True'
- **cutoff** (*Float (optional)*) – Cutoff radius used in 'self.calc_flux', only is used if 'norm' is True

Returns **COG** – curves-of-growth evaluated at 'r'. If 'return_ind = True', returns the profile of each individual gaussian component

Return type array

`calc_obs_cog`(*r*, *return_ind=False*, *norm=False*, *cutoff=None*)
Function to calculate the observed curve of growth, i.e. convolved with the PSF for the given results

Parameters

- `r` (*float or array*) – Radii (in pixels) at which to evaluate the surface brightness profile
- `return_ind` (*bool (optional)*) – If False will only return the sum of all gaussian, i.e. the best fit profile. If true will return an array with +1 dimensions containing the profiles of each individual gaussian component
- `norm` (*Bool (optional)*) – Wether to normalize curves-of-growth to total flux, calculated using 'self.calc_flux'. Does nothing if 'return_ind = True'
- `cutoff` (*Float (optional)*) – Cutoff radius used in 'self.calc_flux', only is used if 'norm' is True

Returns **observed COG** – curves-of-growth evaluated at 'r'. If 'return_ind = True', returns the profile of each individual gaussian component

Return type array

`run_basic_analysis`(*zpt=None*, *cutoff=None*, *errp_lo=16*, *errp_hi=84*, *save_results=False*, *save_file='./imcascade_results.asdf'*)
Function to calculate a set of common variables and save the save the results

Parameters

- `zpt` (*float (optional)*) – photometric zeropoint for the data. if not 'None', will also calculate magnitude
- `cutoff` (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use
- `errp_(lo,hi)` (*float (optional)*) – percentiles to be used to calculate the lower and upper error bars from the posterior distribution. Default is 16 and 84, corresponding to 1-sigma for a guassian distribtuion
- `save_results` (*bool (optional)*) – If true will save results to file. If input is a file, will add to given file, else will save to file denoted by 'save_file' (see below)
- `save_file` (*str*) – String to describe where to save file, only applicaple if the input is not a file.

Returns **res_dict** – Dictionary contining the results of the analysis

Return type dictionary

`calc_iso_r`(*I*, *zpt=None*, *pix_scale=None*)
Function to calculate the isophotal radius

Parameters

- `I` (*float*) – Surface brightness target to define the isophotal radii. By defualt this shoud be in image units unless both zpt and pix_scale are given, then I is interpreted as mag per arcsec^2.
- `zpt` (*float (optional)*) – zeropoint magnitude of image, used to convert I to mag per arcsec^2

- **`pix_scale`** (*`float (optional)`*) – pixel scale in units of arcseconds/pixel, used to convert I to mag per arcsec^2

Returns **r_I** – The radius, in pixel units, where the surface brightness profile matches I

Return type float or Array

`calc_petro_r`(*P_ratio=0.2*, *r_fac_min=0.8*, *r_fac_max=1.25*)
Function to calculate the petrosian radii of a galaxy

Parameters

- **`P_ratio`** (*`float (optional)`*) – The Petrosian ratio which defines the Petrosian radii, default is 0.2
- **`r_fac_min`** (*`float (optional)`*) – lower multiplicative factor which is used to integrate flux, default 0.8
- **`r_fac_max`** (*`float (optional)`*) – higher multiplicative factor which is used to inegrate flux, default 1.25

Returns **r_I** – The radius, in pixel units, where the surface brightness profile matches I

Return type float or Array

`make_diagnostic_fig`()
Function which generates a diagnostic figure to assess fit

Returns **fig** – matplotlib figure object

Return type matplotlib figure

`class imcascade.results.`**`MultiResults`**(*lofr*)
A Class to analyze and combine multiple ImcascadeResults classes using evidence weighting

`calc_cog`(*r*, *num=1000*)

`calc_obs_cog`(*r*, *num=1000*)

`calc_sbp`(*r*, *num=1000*)

`calc_obs_sbp`(*r*, *num=1000*)

`calc_flux`(*cutoff=None*, *num=1000*)

`calc_rX`(*X*, *cutoff=None*, *num=1000*)

`imcascade.utils`

Module Contents

Functions

`guess_weights`(sig, re, flux)	Method to guess the weights of gaussian componenets given an re and flux.
`expand_mask`(mask[, radius, threshold])	Expands mask by convolving it with a Gaussians
`asinh_scale`(start, end, num)	Simple wrapper to generate list of numbers equally spaced in asinh space
`log_scale`(start, end, num)	Simple wrapper to generate list of numbers equally spaced in logspace

continues on next page

Table 9 – continued from previous page

dict_add(dict_use, key, obj)	Simple wrapper to add obj to dictionary if it doesn't exist. Used in fitter.Fitter when defining defaults
get_med_errors(arr[, lo, hi])	Simple function to find percentiles from distribution
b(n)	Simple function to approximate b(n) when evaluating a Sersic profile
sersic(r, n, re, Ltot)	Calculates the surface brightness profile for a Sersic profile
min_diff_array(arr)	Function used to calculate the minimum difference between any two elements

Attributes

vars_to_use

imcascade.utils.**vars_to_use = ['img', 'weight', 'mask', 'sig', 'Ndof', 'Ndof_sky', 'Ndof_gauss', 'has_psf', 'psf_a',...**

imcascade.utils.**guess_weights**(*sig*, *re*, *flux*)

Method to guess the weights of gaussian componenets given an re and flux. Based on a polynomial fit to the exp fits of Hogg & Lang 2013

Parameters

- **sig** (*array*) – List of gaussian widths for imcascade model
- **re** (*Float*) – Estimate of effective radius
- **flux** – Estimate of flux

Returns **a_i** – Inital estimate of weights based on re and flux

Return type Array

imcascade.utils.**expand_mask**(*mask*, *radius=5*, *threshold=0.001*)

Expands mask by convolving it with a Gaussians

Parameters

- **Mask** (*2D array*) – inital mask with masked pixels equal to 1
- **radius** (*Float*) – width of gaussian used to convolve mask. default 5, set larger for more aggresive masking
- **threshold** (*Float*) – threshold to generate new mask from convolved mask. Default is 1e-3, set lower for more aggresive mask

Returns **new_mask** – New, expanded mask

Return type 2D-Array

imcascade.utils.**asinh_scale**(*start*, *end*, *num*)

Simple wrapper to generate list of numbers equally spaced in asinh space

Parameters

- **start** (*floar*) – Inital number
- **end** (*Float*) – Final number

- **num** (*Float*) – Number of number in the list

Returns list – List of number spanning start to end, equally space in asinh space

Return type 1d array

`imcascade.utils.log_scale`(*start*, *end*, *num*)

Simple wrapper to generate list of numbers equally spaced in logspace

Parameters

- **start** (*floar*) – Inital number
- **end** (*Float*) – Final number
- **num** (*Float*) – Number of number in the list

Returns list – List of number spanning start to end, equally space in log space

Return type 1d array

`imcascade.utils.dict_add`(*dict_use*, *key*, *obj*)

Simple wrapper to add obj to dictionary if it doesn't exist. Used in fitter.Fitter when defining defaults

Parameters

- **dict_use** (*Dictionary*) – dictionary to be, possibly, updated
- **key** (*str*) – key to update, only updated if the key doesn't exist in dict_use already
- **obj** (*Object*) – Object to be added to dict_use under key

Returns dict_add – updated dictionary

Return type Dictionary

`imcascade.utils.get_med_errors`(*arr*, *lo=16*, *hi=84*)

Simple function to find percentiles from distribution

Parameters

- **arr** (*array*) – Array containing in the distribution of intrest
- **lo** (*float (optional)*) – percentile to define lower error bar, Default 16
- **hi** (*float (optional)*) – percentile to define upper error bar, Default 84

Returns (med,err_lo,err_hi) – Array containg the median and errorbars of the distiribution

Return type array

`imcascade.utils.b`(*n*)

Simple function to approximate b(n) when evaluating a Sersic profile following Capaccioli (1989). Valid for 0.5 < n < 10

Parameters n (*float or array*) – Sersic index

Returns b(n) – Approximation to Gamma(2n) = 2 gamma(2n,b(n))

Return type float or array

`imcascade.utils.sersic`(*r*, *n*, *re*, *Ltot*)

Calculates the surface brightness profile for a Sersic profile

Parameters

- **r** (*array*) – Radii at which to evaluate surface brightness profile
- **n** (*float*) – Sersic index of profile

- **re** (*float*) – Half-light radius of profile
- **Ltot** (*float*) – Total flux of Sersic profile

Returns Surface brightness profile evaluate along the semi-major axis at 'r'

Return type float or array

`imcascade.utils.`**`min_diff_array`**(*arr*)

Function used to calculate the minimum difference between any two elements in a given array_like :param arr: Array to be searched :type arr: 1-D array

Returns **min_diff** – The minimum difference between any two elements of the given array

Return type Float

5.1.2 Package Contents

Classes

Fitter	A Class used fit images with MultiGaussModel
ImcascadeResults	A class used for collating imcascade results and performing analysis

Attributes

__version__

`imcascade.`**`__version__`** **= '1.0'**

class `imcascade.`**`Fitter`**(*img*, *sig*, *psf_sig*, *psf_a*, *weight=None*, *mask=None*, *sky_model=True*, *sky_type='tilted-plane'*, *render_mode='hybrid'*, *log_weight_scale=True*, *verbose=True*, *psf_shape=None*, *init_dict={}*, *bounds_dict={}*, *log_file=None*)

Bases: *imcascade.mgm.MultiGaussModel*

A Class used fit images with MultiGaussModel

This is the main class used to fit `imcascade` models

Parameters

- **img** (*2D Array*) – Data to be fit, it is assumed to be a cutout with the object of interest in the center of the image
- **sig** (*1D Array*) – Widths of Gaussians to be used in MultiGaussModel
- **psf_sig** (*1D array, None*) – Width of Gaussians used to approximate psf
- **psf_a** (*1D array, None*) – Weights of Gaussians used to approximate psf If both psf_sig and psf_a are None then will run in Non-psf mode
- **weight** (*2D Array, optional*) – Array of pixel by pixel weights to be used in fitting. Must be same shape as 'img' If None, all the weights will be set to 1.
- **mask** (*2D Array, optional*) – Array with the same shape as 'img' denoting which, if any, pixels to mask during fitting process. Values of '1' or 'True' values for the pixels to be

masked. If set to 'None' then will not mask any pixels. In practice, the weights of masked pixels is set to '0'.

- **sky_model** (*bool, optional*) – If True will incorperate a tilted plane sky model. Reccomended to be set to True
- **sky_type** (*str, 'tilted-plane' or 'flat'*) – Function used to model sky. Default is tilted plane with 3 parameters, const bkg and slopes in each directin. 'flat' uses constant background model with 1 parameter.
- **render_mode** (*'hybrid', 'erf' or 'gauss'*) – Option to decide how to render models. 'erf' analytically computes the integral over the pixel of each profile therefore is more accurate but more computationally intensive. 'gauss' assumes the center of a pixel provides a reasonble estimate of the average flux in that pixel. 'gauss' is faster but far less accurate for objects which vary on O(pixel size), so use with caution. 'hybrid' is the defualt, uses 'erf' for components with width < 5 to ensure accuracy and uses 'gauss' otherwise as it is accurate enough and faster. Also assumes all flux > 5 sigma for components is 0.
- **log_weight_scale** (*bool, optional*) – Wether to treat weights as log scale, Default True
- **verbose** (*bool, optional*) – If true will log and print out errors
- **psf_shape** (*dict, Optional*) – Dictionary containg at 'q' and 'phi' that define the shape of the PSF. Note that this slows down model rendering significantly so only reccomended if neccesary.
- **init_dict** (*dict, Optional*) – Dictionary specifying initial guesses for least_squares fitting. The code is desigined to make 'intelligent' guesses if none are provided
- **bounds_dict** (*dict, Optional*) – Dictionary specifying boundss for least_squares fitting and priors. The code is desigined to make 'intelligent' guesses if none are provided

resid_1d(*params*)

Given a set of parameters returns the 1-D flattened residuals when compared to the Data, to be used in run_ls_min Function

Parameters **params** (*Array*) – List of parameters to define model

Returns **resid_flatten** – 1-D array of the flattened residuals

Return type array

run_ls_min(*ls_kwargs={}*)

Function to run a least_squares minimization routine using pre-determined inital guesses and bounds.

Utilizes the scipy least_squares routine (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html)

Parameters **ls_kwargs** (*dict, optional*) – Optional list of arguments to be passes to least_squares routine

Returns **min_param** – Returns a 1D array containing the optimized parameters that describe the best fit model.

Return type 1D array

set_up_express_run(*set_params=None*)

Function to set up 'express' run using pre-rendered images with a fixed x0,y0, phi and q. Sets class attribute 'express_gauss_arr' which is needed to run dynesty or emcee in express mode

Parameters `set_params` (*len(4) array-like, optional*) – Parameters (x0,y0,q,phi) to use to per-render images. If None will call `run_ls_min()` or use stored `min_res` to find parameters.

Returns **express_gauss_arr** – Returns a 3-D with pre-rendered images based on input parameters

Return type array (shape[0],shape[1], Ndof_gauss)

`make_express_model`(*exp_params*)

Function to generate a model for a given set of paramters, specifically using the pre-renedered model for the 'express' mode

Parameters `exo_params` (*Array*) – List of parameters to define model. Length is Ndof_gauss + Ndof_sky since the structural parameters (x0,y0,q, PA) are set

Returns **model** – Model image based on input parameters

Return type 2D-array

`chi_sq`(*params*)

Function to calculate chi_sq for a given set of paramters

Parameters `params` (*Array*) – List of parameters to define model

Returns **chi^2** – Chi squared statistic for the given set of parameters

Return type float

`log_like`(*params*)

Function to calculate the log likeliehood for a given set of paramters

Parameters `params` (*Array*) – List of parameters to define model

Returns **log likeliehood** – log likeliehood for a given set of paramters, defined as -0.5*chi^2

Return type float

`ptform`(*u*)

Prior transformation function to be used in dynesty 'full' mode

Parameters `u` (*array*) – array of random numbers from 0 to 1

Returns **x** – array containing distribution of parameters from prior

Return type array

`log_like_exp`(*exp_params*)

Function to calculate the log likeliehood for a given set of paramters, specifically using the pre-renedered model for the 'express' mode

Parameters `exo_params` (*Array*) – List of parameters to define model. Length is Ndof_gauss + Ndof_sky since the structural parameters (x0,y0,q, PA) are set

Returns **log likeliehood** – log likeliehood for a given set of paramters, defined as -0.5*chi^2

Return type float

`ptform_exp_ls`(*u*)

Prior transformation function to be used in dynesty 'express' mode using gaussian priors defined by the results of the least_squares minimization

Parameters `u` (*array*) – array of random numbers from 0 to 1

Returns **x** – array containing distribution of parameters from prior

Return type array

ptform_exp_lin_cauchy(*u*)

Prior transformation function to be used in dynesty 'express' mode using gaussian priors defined by the results of the least_squares minimization

Parameters **u** (*array*) – array of random numbers from 0 to 1

Returns **x** – array containing distribution of parameters from prior

Return type array

ptform_exp_unif(*u*)

Prior transformation function to be used in dynesty 'express' mode using unifrom priors defined by self.lb and self.ub

Parameters **u** (*array*) – array of random numbers from 0 to 1

Returns **x** – array containing distribution of parameters from prior

Return type array

run_dynesty(*method='full'*, *sampler_kwargs={}*, *run_nested_kwargs={}*, *prior='min_results'*)

Function to run dynesty to sample the posterior distribution using either the 'full' methods which explores all paramters, or the 'express' method which sets the structural parameters.

Parameters

- **method** (*str: 'full' or 'express'*) – Which method to use to run dynesty
- **sampler_kwargs** (*dict*) – set of keyword arguments to pass the the dynesty DynamicNestedSampler call, see: https://dynesty.readthedocs.io/en/latest/api.html#dynesty.dynesty.DynamicNestedSampler
- **run_nested_kwargs** (*dict*) – set of keyword arguments to pass the the dynesty run_nested call, see: https://dynesty.readthedocs.io/en/latest/api.html#dynesty.dynamicsampler.DynamicSampler.run_nested
- **prior** (*'min_results' or 'uniform'*) – Which of the two choices of priors to use. The *min_results* priors are Gaussian, with centers defined by the best fit paramters and variance equal to 4 times the variance estimated using the Hessian matrix from the run_ls_min() run. *uniform* is what it sounds like, uniform priors based on the the lower and upper bounds Defualt is *min_results*

Returns **Posterior** – posterior distribution derrived. If method is 'express', the first 4 columns, containg x0, y0, PA and q, are all the same and equal to values used to pre-render the images

Return type Array

save_results(*file_name*)

Function to save results after run_ls_min, run_dynesty and/or run_emcee is performed. Will be saved as an ASDF file.

Parameters **file_name** (*str*) – Str defining location of where to save data

Returns **dict_saved** – Dictionary containing all the save quantities

Return type dict

class imcascade.**ImcascadeResults**(*Obj*, *thin_posterior=1*)

A class used for collating imcascade results and performing analysis

Parameters

- **Obj** (*imcascade.fitter.Fitter class, dictionary or str*) – Object which contains the data to be analyzed. Can be a Fitter object once the run_(ls_min,dynesty, emcee) has been ran. If it is a dictionay needs to contain, at bare minmum the variables sig, Ndof,

Ndof_sky, Ndof_gauss, log_weight_scale and either min_param or posterior. If a string is passed it will be interreted as a file locations with an ASDF file containing the neccesary information.

- **thin_posterior** (*int (optional)*) – Factor by which to thin the posterior distribution by. While one wants to ensure the posterior is large enough, some of this analysis can take time if you have >10^6 samples so this is one way to speed up this task but use with caution.

calc_flux(*cutoff=None*)
Calculate flux of given results

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **Flux** – Total flux of best fit model

Return type float or Array

_min_calc_rX(*X*, *cutoff=None*)
Old and slow Function to calculate the radius containing X percent of the light

Parameters

- **X** (*float*) – Fractional radius of intrest to calculate. if X < 1 will take as a fraction, else will interpret as percent and divide X by 100. i.e. to calculate the radius containing 20% of the light, once can either pass X = 20 or 0.2
- **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width used

Returns **r_X** – The radius containg X percent of the light

Return type float or Array

calc_rX(*X*, *cutoff=None*)
Function to calculate the radius containing X percent of the light

Parameters

- **X** (*float*) – Fractional radius of intrest to calculate. if X < 1 will take as a fraction, else will interpret as percent and divide X by 100. i.e. to calculate the radius containing 20% of the light, once can either pass X = 20 or 0.2
- **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width used

Returns **r_X** – The radius containg X percent of the light

Return type float or Array

calc_r90(*cutoff=None*)
Wrapper function to calculate the radius containing 90% of the light

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **r_90** – The radius containg 90 percent of the light

Return type float or Array

calc_r80(*cutoff=None*)
Wrapper function to calculate the radius containing 80% of the light

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **r_80** – The radius containg 80 percent of the light

Return type float or Array

`calc_r50`(*cutoff=None*)

Wrapper function to calculate the radius containing 50% of the light, or the effective radius

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **r_50** – The radius containg 50 percent of the light

Return type float or Array

`calc_r20`(*cutoff=None*)

Wrapper function to calculate the radius containing 20% of the light

Parameters **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use

Returns **r_20** – The radius containg 20 percent of the light

Return type float or Array

`calc_sbp`(*r*, *return_ind=False*)

Function to calculate surface brightness profiles for the given results

Parameters

- **r** (*float or array*) – Radii (in pixels) at which to evaluate the surface brightness profile
- **return_ind** (*bool (optional)*) – If False will only return the sum of all gaussian, i.e. the best fit profile. If true will return an array with +1 dimensions containing the profiles of each individual gaussian component

Returns **SBP** – Surface brightness profiles evaluated at 'r'. If 'return_ind = True', returns the profile of each individual gaussian component

Return type array

`calc_obs_sbp`(*r*, *return_ind=False*)

Function to calculate the observed surface brightness profiles, i.e. convolved with the PSF for the given results

Parameters

- **r** (*float or array*) – Radii (in pixels) at which to evaluate the surface brightness profile
- **return_ind** (*bool (optional)*) – If False will only return the sum of all gaussian, i.e. the best fit profile. If true will return an array with +1 dimensions containing the profiles of each individual gaussian component

Returns **obsereved SBP** – Observed surface brightness profiles evaluated at 'r'. If 'return_ind = True', returns the profile of each individual gaussian component

Return type array

`calc_cog`(*r*, *return_ind=False*, *norm=False*, *cutoff=None*)

Function to calculate curves-of-growth for the given results

Parameters

- **r** (*float or array*) – Radii (in pixels) at which to evaluate the surface brightness profile

- **return_ind** (*bool (optional)*) – If False will only return the sum of all gaussian, i.e. the best fit profile. If true will return an array with +1 dimensions containing the profiles of each individual gaussian component
- **norm** (*Bool (optional)*) – Wether to normalize curves-of-growth to total flux, calculated using 'self.calc_flux'. Does nothing if 'return_ind = True'
- **cutoff** (*Float (optional)*) – Cutoff radius used in 'self.calc_flux', only is used if 'norm' is True

Returns **COG** – curves-of-growth evaluated at 'r'. If 'return_ind = True', returns the profile of each individual gaussian component

Return type array

`calc_obs_cog`(*r*, *return_ind=False*, *norm=False*, *cutoff=None*)

Function to calculate the observed curve of growth, i.e. convolved with the PSF for the given results

Parameters

- **r** (*float or array*) – Radii (in pixels) at which to evaluate the surface brightness profile
- **return_ind** (*bool (optional)*) – If False will only return the sum of all gaussian, i.e. the best fit profile. If true will return an array with +1 dimensions containing the profiles of each individual gaussian component
- **norm** (*Bool (optional)*) – Wether to normalize curves-of-growth to total flux, calculated using 'self.calc_flux'. Does nothing if 'return_ind = True'
- **cutoff** (*Float (optional)*) – Cutoff radius used in 'self.calc_flux', only is used if 'norm' is True

Returns **observed COG** – curves-of-growth evaluated at 'r'. If 'return_ind = True', returns the profile of each individual gaussian component

Return type array

`run_basic_analysis`(*zpt=None*, *cutoff=None*, *errp_lo=16*, *errp_hi=84*, *save_results=False*, *save_file='./imcascade_results.asdf'*)

Function to calculate a set of common variables and save the save the results

Parameters

- **zpt** (*float (optional)*) – photometric zeropoint for the data. if not 'None', will also calculate magnitude
- **cutoff** (*float (optional)*) – Radius out to which to consider the profile. Generally this should be around the half-width of the image or the largest gaussian width use
- **errp_(lo,hi)** (*float (optional)*) – percentiles to be used to calculate the lower and upper error bars from the posterior distribution. Default is 16 and 84, corresponding to 1-sigma for a guassian distribtuion
- **save_results** (*bool (optional)*) – If true will save results to file. If input is a file, will add to given file, else will save to file denoted by 'save_file' (see below)
- **save_file** (*str*) – String to describe where to save file, only applicaple if the input is not a file.

Returns **res_dict** – Dictionary contining the results of the analysis

Return type dictionary

`calc_iso_r`(*I*, *zpt=None*, *pix_scale=None*)

Function to calculate the isophotal radius

Parameters

- `I` (*float*) – Surface brightness target to define the isophotal radii. By defualt this shoud be in image units unless both zpt and pix_scale are given, then I is interpreted as mag per arcsec^2.
- `zpt` (*float (optional)*) – zeropoint magnitude of image, used to convert I to mag per arcsec^2
- `pix_scale` (*float (optional)*) – pixel scale in units of arcseconds/pixel, used to convert I to mag per arcsec^2

Returns **r_I** – The radius, in pixel units, where the surface brightness profile matches I

Return type float or Array

`calc_petro_r`(*P_ratio=0.2*, *r_fac_min=0.8*, *r_fac_max=1.25*)

Function to calculate the petrosian radii of a galaxy

Parameters

- `P_ratio` (*float (optional)*) – The Petrosian ratio which defines the Petrosian radii, default is 0.2
- `r_fac_min` (*float (optional)*) – lower multiplicative factor which is used to integrate flux, default 0.8
- `r_fac_max` (*float (optional)*) – higher multiplicative factor which is used to inegrate flux, default 1.25

Returns **r_I** – The radius, in pixel units, where the surface brightness profile matches I

Return type float or Array

`make_diagnostic_fig`()

Function which generates a diagnostic figure to assess fit

Returns **fig** – matplotlib figure object

Return type matplotlib figure

PYTHON MODULE INDEX

INDEX