ShapeMeasurementFisherFormalism Documentation

Release 0.2

Ismael Mendoza

Nov 29, 2018

Contents

Fast simulations and analysis for the Weak Lensing Working Group of the LSST Dark Energy Science Collaboration.

This software was primarily developed to study the effects of overlapping sources on pixel-noise bias estimation using the Fisher Formalism.

The code is hosted on github. Please use the issue tracker to let us know about any issues you have with installing or running this code, or to request new features.

This work was conducted by Ismael Mendoza, in collaboration with Pat Burchat and Josh Meyers, at Stanford University in 2015-2017. Work was supported by the National Science Foundation and by LSST.

CHAPTER 1

Installation

1.1 Install with GIT

The code is hosted on github so the easiest method to perform an initial installation is with git:

```
git clone https://github.com/ismael2395/ShapeMeasurementFisherFormalism.git
```

This will create a new subdirectory *ShapeMeasurementFisherFormalism* containing the latest stable version of the complete package.

Experts who already have a correctly configured github account might prefer this alternative:

```
git clone git@github.com:ismael2395/ShapeMeasurementFisherFormalism.git
```

1.2 Update with GIT

You can update your local copy of the package at any time using:

```
cd ShapeMeasurementFisherFormalism
git update
```

1.3 Getting Started

Programs can be run directly from the top-level directory as long as you have the required packages already installed, e.g.:

```
cd ShapeMeasurementFisherFormalism
python generate.py --help
```

For an introduction to the available programs, see *here* and for examples of running these programs see *here*.

1.4 Required Packages

The following python packages are required by this package:

- numpy (version >= 1.9)
- galsim (version >= 1.2)
- lmfit (version >= 0.8.3)

Note that numpy is available in recent anaconda or enthought canopy distributions. Installing GalSim is a more involved process, but well worth the effort. The *lmfit* package is only required if you will be running your own fits using the *fitting*.

CHAPTER 2

Examples

2.1 Command-line Options

Print out usage info for command-line options:

```
python generate.py --help
python fitting.py --help
```

2.2 Quick Simulation Demo

The first thing to do is to generate a desired galaxy model (with an optional PSF) using the generate.py file:

```
python generate.py -p project -gal 1 --galaxy-model gaussian --psf_model psf_gaussian␣
↪ --g1 0.2 --g2 0.2 --y0 0. --x0 0. --flux 1. --psf_flux 1. --hlr 0.5 --psf_fwhm 0.7␣
↪--snr 20.0
```

Display partial derivatives and save it to the project folder just created with generate.py:

```
python display.py -p project --partials --snr 20.
```

Display a fisher matrix elements without showing it when the command is executed:

```
python display.py -p project --fisher --snr 20. --hide
```

It is important to always specify the project that is being run as well as the signal to noise ratio to be used. . Finally one can also save all the possible outputs to the current project folder with:

```
python display.py -p project --all --snr 20.
```

This command hides the output by default and saves all files in a .pdf format.

2.3 More complicated examples

It is not necessary to use the *display* module to display the images (and it can be a little limiting in terms of formatting the images to your liking). It is also possible to access the images directly from the `analysis.fisher.fisher` objects and use them. Please refer to the *tutorial notebooks* for tutorials on how to do this. The notebooks also contain other more complicated examples of how the package can be used.

CHAPTER 3

Programs

All available programs are contained in the top-level directory where this package is installed. Programs are written in python and use the '.py' file extension.

All programs are configured by passing command-line options. Pass the *–help* option to view documentation for these options, e.g.:

```
python generate.py --help
```

3.1 Executable Programs

3.1.1 generate

The *generate* program writes to a CSV file that can be later be read by the `analysis.galfun` or by `analysis.fisher` modules to generate galaxy images and their analysis. Fast image rendering with galsim.

The user specifies the arguments of the galaxy and the PSF in the following way:

```
python generate.py -p PROJECT_NAME -gal 1 --galaxy-model MODEL_NAME --snr VALUE --
↪PARAM1_NAME PARAM1_VALUE ...
```

For a given galaxy model at least the following parameters should always be specified:

- x0
- y0
- flux
- measure of size (sigma,hlr,fwhm)
- measure of ellipticity ((g1,g2),(e1,e2))

You can look up what the different parameters mean exactly in this *section*. The project name, the galaxy model, the snr value, and the galaxy ID (in the case when you draw 1 you only need one galaxy ID) should also always be specified. One can optionally add a PSF for the galaxy to be convolved with:

```
python generate.py -p PROJECT_NAME -gal 1 --galaxy-model MODEL_NAME --snr VALUE --
↪PARAM1_NAME PARAM1_VALUE ... --psf_model PSF_MODEL_NAME --PSF_PARAM1 PSF_PARAM1_
↪VALUE ...
```

A PSF always needs a model name, a measure of size, and a flux (which should be 1). The PSF and the galaxy are treated differently in the package. If you want to generate two galaxies just use the procedure above changing the id:

```
python generate.py -p PROJECT_NAME -gal 2 --galaxy-model MODEL_NAME --snr VALUE --
↪PARAM1_NAME PARAM1_VALUE ... --psf_model PSF_MODEL_NAME --PSF_PARAM1 PSF_PARAM1_
↪VALUE ...
```

Then the .csv file should contain information about both galaxies (one row per galaxy). It is recommended you use specify the same PSF for both galaxies.

3.1.2 display

The *display.py* program displays simulated images and analysis results stored in the CSV file generated by the *generate.py* program. It will display the images when the program is executed produced unless the *–hide* argument is used when executed. Please run:

```
python display.py --help
```

For the different images that can be produced and some explanations for them. More information on how to use the module can also be found in *here*.

3.1.3 fitting

The *fitting* program runs fits using lmfit. as an alternative way of estimating the noise bias and error in order to compare it to the Fisher Formalism.

In order to run a certain number N of fits, make sure you already created a galaxy with the *generate* program and then run:

```
fitting.py -p EXISTING_PROJECT_NAME -rf -n N --snr SNR_VALUE
```

If you are Stanford SLAC affiliated and want send N batch jobs to SLAC, then log in to SLAC and then run:

```
fitting.py -p EXISTING_PROJECT_NAME -rfs short -n N --snr SNR_VALUE
```

This *fitting* module internally calls another module named *runfits* *N* times which produces the results from a single fits and then writes them to a .csv file. The correct way to read the data produced from this process is to use the `analysis.galfun.read_results()` function which returns the results in a convenient format. For more specific information about the output of this function please consult the docstring. To see how these process can be used to produce interesting fits and their plots please take a look at the fits_anlaysis *tutorial notebooks*.

3.2 Other Modules

3.2.1 models

The `analysis.models` module contains the galaxy and PSF models that can be produced by *generate* and analyzed by `analysis.fisher`. It specifies the commands that galsim should use to produce them.

3.2.2 fisher

The *analysis.fisher* module contains the analysis by the Fisher Formalism which is made on the galaxy that *generate* produced. Please refer to the tutorials in *tutorial notebooks* for specific instructions on how to use this module efficiently in conjunction with *analysis.galfun* module.

3.2.3 galfun

The *analysis.galfun* is a multipurpose module that contains important functions ranging from managing parameters of generated galaxies to extracting information from relevant files.

3.2.4 defaults

The *analysis.defaults* module stores defaults parameter values for different parts of the package.

3.2.5 runfits

The *runfits* module runs `lmfit` to do the fits on several noisy instantiations of a given galaxy model and produces results in the form of .csv files inside the "results" folder that will be inside your corresponding project folder (depends on what you decided to name it).

CHAPTER 4

Models and Parameters

This document describes the galaxy parameters and models that are implemented in the *generate.py* program. For more detailed information take a look into *analysis.models* and the galsim documentation.

4.1 Galaxy Models

Name	Description
gaussian	Gaussian galaxy profile, `galsim.Gaussian`.
exponential	Exponential galaxy profile, `galsim.Exponential`.
deVaucouleurs	DeVaucouleurs galaxy profile, `galsim.DeVaucouleurs`.
bulgeDisk	Combined exponential (disk) and deVaucouleurs (bulge) profile.

4.2 Galaxy Parameters

Name	Type	Description
Program Parameters		
project	string	Pixel offset of left edge of bounding box relative to left edge of survey image.
gal	int32	Pixel offset of right edge of bounding box relative to left edge of survey image.
ymin	int32	Pixel offset of bottom edge of bounding box relative to bottom edge of survey image.
ymax	int32	Pixel offset of top edge of bounding box relative to bottom edge of survey image.
Source Properties		
f_disk	float32	Fraction of total galaxy flux to due a Sersic n=1 disk component.
f_bulge	float32	Fraction of total galaxy flux to due a Sersic n=4 bulge component.
flux	float32	Total detected flux in electrons.
x	float32	Source centroid in x relative to image center in arcseconds.
y	float32	Source centroid in y relative to image center in arcseconds.
e1	float32	Real part (+) of galaxy ellipticity spinor (Q11-Q22+2*Q12)/(Q11+Q22).
e2	float32	Imaginary part (x) of galaxy ellipticity spinor 2*Q12/(Q11+Q22).
g1	float32	Real part (+) of galaxy ellipticity spinor (Q11-Q22)/(Q11+Q22+2|Q|**0.5).
g2	float32	Imaginary part (x) of galaxy ellipticity spinor (2*Q12)/(Q11+Q22+2|Q|**0.5).
eta1	float32	Conformal shear with a/b = exp(eta).
eta2	float32	Conformal shear with a/b = exp(eta).
sigma	float32	Galaxy size arcseconds calculated as |Q|**0.25.
sigma_p	float32	Galaxy size in arcseconds calculated as (0.5*trQ)**0.5.
hlr	float32	The half-light-radius of the profile.
beta	float32	Position angle of second-moment ellipse in radians, or zero when a = b.
q	float32	b/a ratio of semi-minor axis to semi-major axis.
fwhm	float32	Full-width-half-max of the profile, defined as radius at half the max peak...
snr	float32	Signal to noise ration as defined in...
hlr_b	float32	hlr for the bulge component of the *analysis.models.bulgeDisk* model.
flux_d	float32	flux for the disk component of the *analysis.models.bulgeDisk* model.
flux_b	float32	flux for the bulge component of the *analysis.models.bulgeDisk* model.
R_r	float32	hlr for the bulge component of the *analysis.models.bulgeDisk* model.

4.3 PSF Models

Name	Description
psf_gaussian	Gaussian galaxy profile, `galsim.Gaussian`.
psf_moffat	Moffat galaxy profile, `galsim.Moffat`.

4.4 PSF Parameters

For the PSF, the parameters have the same significance as in the table of parameters as above.

CHAPTER 5

IPython Notebooks

The following iPython notebooks are included in the *notebooks/* directory and can be viewed online:

- Individual_Fisher_Analysis: Tutorial of how to use the package to analyze single galaxies using the Fisher Formalism.
- Group_Fisher_Analysis: Tutorial of how to use the package to analyze 2 (or more) galaxies using the Fisher Formalism.
- Fit_Analysis: Tutorial on how to use the *fitting* package to produce noise fits and compare results to the Fisher Formalism results.
- how_to_add_models: Tutorial on how to add new galaxy models to be analyzed in the package.
- Ring_Test: Tutorial on the ring test.

You can also edit and run these notebooks locally if you have ipython installed. First start the server in its own window (where it log messages will appear):

```
ipython notebook
```

This should open up a notebook browser where you can click on any of the notebooks to launch it.

CHAPTER 6

To Do List

- Add ability to change the ellipticity of the PSF.

6.1 Longer-Term Projects

-

CHAPTER 7

Information for Developers

7.1 Build the documentation

To build a local copy of the HTML documentation, use:

```
cd .../docs
make html
```

To view the most recent build of the HTML documentation, point your browser to *.../docs/_build/html/index.html*

7.2 Add a new package module

Create a new file *analysis/xxx.py* for module *analysis.xxx* with an initial descriptive docstring.

Add the line *import xxx* to *analysis/__init__.py*.

Add the line *analysis.xxx* to the list of submodules in *docs/src/analysis.rst*.

Create a new documentation file *docs/src/analysis.xxx.rst* containing (replace *xxx* in two places):

```
analysis.xxx module
=================

.. automodule:: analysis.xxx
    :members:
    :undoc-members:
    :show-inheritance:
```

7.3 Update the version

Update the *version* and *release* values in the sphinx configuration file *docs/conf.py*, then commit, tag, and push the new version, e.g.:

```
git tag -l v0.2
```

CHAPTER 8

Revision History

8.1 v0.1

- Download from here.

CHAPTER 9

Modules API Reference

9.1 analysis package

9.1.1 Submodules

analysis.fisher module

This module contains functions necessary to produce statistical results of the fisher formalism from a given galaxy.

class analysis.fisher.**Fisher**(*g_parameters*, *image_renderer*, *snr*, *var_noise=None*)
Bases: object

Produce fisher object (containing fisher analysis) for a given set of galaxy parameters.

Given a galaxy image and the appropiate parameters that describe it, (in the form of *analysis.galfun.GParameters* object) will produce a fisher object that contains the analysis of it using the Fisher Formalism.

NOTE: The matrices are in dictionary form, use the function matrixToNumpyArray() to change them to a matrix that is ordered according to param_names.

Args:

g_parameters(GParameters): String point to the directory specified by the user.

image_renderer(ImageRenderer): Object used to render image of galaxy. snr(float): Value S/N ratio to use in the analysis.

Attributes: image_renderer_partials(*analysis.galfun.ImageRenderer*): Object used to render images of partial derivatives. image(Galsim.Image): Dictionary whose keys are the ids of each of the

galaxies specified in galaxies.csv, and that map to another dictionary that can be taken in by *analysis.galfun.getGalaxyModel()*

var_noise(float): Variance of noise of given S/N . steps(dict): Dictionary containing the step size used when

calculating partial derivatives.

param_names(list): A list containing the keys of fit_params in a desirable order.

num_params(int): Number of parameters specified for the galaxy. num_galaxies(int): Number of galaxies specified. derivatives_images(dict): Dictionary containing np.array(s) that represent the

derivative of the galaxy(ies) with respect to each parameter.

second_derivatives_images(dict): Dictionary containing np.array(s) that represent the second derivatives of the galaxy(ies) with respect to its parameters.

fisher_matrix_images(dict): Dictionary containing np.array(s) that represent the fisher matrix images of the galaxy(ies) with respect to its parameters.

fisher_matrix(dict): Dictionary containing fisher matrix elements. covariance_matrix(dict): Dictionary containing covariance matrix elements. correlation_matrix(dict): Dictionary containing correlation matrix elements. bias_matrix_images(dict): Dictionary containing bias matrix image elements. bias_matrix(dict): Dictionary containing bias matrix elements. bias_images(dict): Dictionary containing bias images elements. biases(dict): Dictionary containing biases

biasImages()
Construct the bias of each parameter per pixel.

biasMatrix()
Return bias matrix from the images of the bias matrix

biasMatrixImages()
Produce images of each element of the bias matrix.

correlationMatrix()
Calculate correlation matrix from the covariance matrix.

covarianceMatrix()
Calculate the covariance matrix by inverting fisher matrix.

derivativesImages()
Return images of the partial derivatives of the galaxy.

The partial differentiation includes each of the different parameters that describe the galaxy.

fisherConditionNumber()
The condition number will give a sense of how singular the matrix tends to be.

fisherMatrix()
Calculate the actual values of the fisher matrix.

fisherMatrixImages()
Produce images of fisher matrix).

getBiases()
Return the value of the bias of each parameter in vector form.

matrixToNumpyArray(*matrix*)
Convert matrix dictionary to a numpy array.

numpyArrayToMatrix(*array*)
Convert numpy array to matrix dictionary.

secondDerivativesImages()
Return the images for the second derivatives of the given galaxy.

analysis.fisher.**getSNR**(*img*, *var_noise*)

analysis.galfun module

Multipurpose module that contains important functions ranging from managing parameters of generated galaxies to extracing information from relevant files.

class analysis.galfun.**GParameters**(*project=None*, *id_params=None*, *omit={}*)
Bases: object

Class that manages galaxies parameters obtained from galaxies.csv

This class reads a galaxies.csv file located in the specified project directory and extracts the parameters of each galaxy contained in it.

Args: project(str): String point to the directory specified by the user. id_params(dict): See Attributes for details. Makes it possible to create a *GParameters* object without galaxies.csv file.

Attributes:

omit_fit(dict): Dictionary defined in containing the parameters that should not be included in the analysis for a particular galaxy model but could be ecessary for drawing the galaxy.

id_params(dict): Dictionary whose keys are the ids of each of the galaxies specified in galaxies.csv, and that map to another dictionary that can be taken in by *getGalaxyModel()*

params(dict): Dictionary that encodes the same information as id_params but in a different form. Combines each of the dictionaries contained in id_params into a single dictionary that contains all parameters but in the form param_#.

fit_params(dict): Dictionary similar to the params attribute but without the parameters specified in omit_fit.

nfit_params(dict): Dictionary that contains all the parameters not contained in fit_params. Usually used for kwargs in conjunction with fit_params to draw Galaxies.

ordered_fit_names(list): A list containing the keys of fit_params in a desirable order.

num_galaxies(int): Number of galaxies specified.

static convertId_Params(*id_params*, *omit_fit={}*)
Converts id_params to the format of params.

Parameters

- **id_params** (*dict*) – Same as id_params
- **omit_fit** (*dict*) – Dictionary that has the same purpose as omit_fit

Returns A dictionary of params (dict).

static convertParams_Id(*params*)
Convert a dictionary params in the format of params to a dictionary in the format id_params

getNFitParams()
Extract nfit_params from params by noticing which parameters are in fit_params.

sortModelParamsNames()
Return the keys of params in an ordered specified by the class parameters. And when having more than one galaxy, all the parameters from one of the galaxies are ordered together.

class analysis.galfun.**ImageRenderer**(*pixel_scale=None*, *nx=None*, *ny=None*, *stamp=None*, *project=None*, *bounds=None*, *mask=None*)

Bases: object

Object used to produce the image of a galaxy.

Parameters

- **pixel_scale** (*float*) – Pixel_scale to use in the image (ratio of pixels to arcsecs)
- **nx** (*float*) – Width of the image in pixels.
- **ny** (*float*) – height of the image in pixels.
- **stamp** (*int*) – optional, galsim.Image of appropiate dimensions to draw the image. does not actually use whatever was originally in the stamp.
- **bounds** (*tuple*) – When drawn, the image will be clipped to these bounds.
- **mask** (np.array) – the pixels selected in this mask will be set to 0.

One of the the following must be specified: *stamp *nx,ny,pixel_scale

This object is made so it can be passsed in to a class *analysis.fisher.Fisher* object.

getImage(*galaxy*)

analysis.galfun.**addNoise**(*image*, *snr*, *noise_seed=0*)

Set gaussian noise to the given galsim.Image.

Parameters

- **image** (galsim.Image) – Galaxy image that noise is going to be added to.
- **snr** (*float*) – Signal to noise ratio.
- **noise_seed** (*int*) – Seed to set to galsim.BaseDeviate which will create the galsim.noise instance.

Returns A galsim.Image, variance_noise tuple. The image is the noisy version of the original image and variance_noise is the noise variance on each pixel due to the added noise.

analysis.galfun.**getGalaxiesModels**(*fit_params=None*, *id_params=None*, *g_parameters=None*, ***kwargs*)

Return the model of a set of galaxies.

One of the the following must be specified: fit_params (and nfit_params as **kwargs. Used specially in runfits.py). id_params g_parameters (from which id_params is extracted.)

This function generates each of the galaxies specified in id_params and then sums them together to get a final galaxy.

Parameters

- **fit_params** (*dict*) – Partial form of id_params that only includes the parameters to be used for the fit. For details, *GParameters*
- **id_params** (*dict*) – Dictionary containing each of the galaxies parameters. For details, *GParameters*
- **g_parameters** (*GParameters*) – An object containing different forms of the galaxy parameters.
- **image** (*bool*) – If :bool:True returns an galsim.Image otherwise it
- **a np.array** (*returns*) –

Returns A `galsim.GSObject`

`analysis.galfun.`**`getGalaxyModel`**(*params*)
Return the image of a single galaxy optionally drawn with a psf.

Look at *`analysis.models`* to figure out which galaxy models and psf models are supported as well as their corresponding implemented parameters. You can even add your own galaxies if desired.

Parameters

- **`params`** (*`dict`*) – Dictionary containing the information of a single
- **`where the keys is the name`** (*`galaxy`*) –
- **`are the values of the parametes.`** (*`values`*) –

Returns A `galsim.GSObject`

`analysis.galfun.`**`getOmitFit`**(*id_params*, *omit*)

`analysis.galfun.`**`read_results`**(*project*, *g_parameters*, *fish*, *limit=None*)

analysis.models module

This module contains the models that are used for galaxies and psfs for galsim, more information of how to add your own models for both galaxies and psfs can be found in the corresponding tutorial.

class `analysis.models.`**`bulgeDisk`**(*params=None*, *params_omit=None*)
Bases: *`analysis.models.model`*

`getProfile`(*params*)

`omit_general = ['delta_e', 'delta_theta', 'n_d', 'n_b']`

`parameters = ['x0', 'y0', 'flux_b', 'flux_d', 'flux_b/flux_total', 'hlr_d', 'hlr_b', '`

class `analysis.models.`**`bulgeDisk6`**(*params=None*, *params_omit=None*)
Bases: *`analysis.models.model`*

`getProfile`(*params*)

`omit_general = ['n_d', 'n_b']`

`parameters = ['x0', 'y0', 'flux', 'hlr', 'e1', 'e2', 'eta1', 'eta2', 'n_d', 'n_b']`

class `analysis.models.`**`deVaucouleurs`**(*params=None*, *params_omit=None*)
Bases: *`analysis.models.model`*

`getProfile`(*params*)

`omit_general = []`

`parameters = ['x0', 'y0', 'flux', 'hlr', 'fwhm', 'sigma', 'e1', 'e2', 'eta1', 'eta2']`

class `analysis.models.`**`exponential`**(*params=None*, *params_omit=None*)
Bases: *`analysis.models.model`*

`getProfile`(*params*)

`omit_general = []`

`parameters = ['x0', 'y0', 'flux', 'hlr', 'fwhm', 'sigma', 'e1', 'e2', 'g1', 'g2', 'eta`

class `analysis.models.`**`gaussian`**(*params=None*, *params_omit=None*)
Bases: *`analysis.models.model`*

`getProfile`(*params*)

omit_general = []

parameters = ['flux', 'x0', 'y0', 'hlr', 'fwhm', 'sigma', 'e1', 'e2', 'eta1', 'eta2',

analysis.models.**getAllModels**()
Used to display choices in generate.py

analysis.models.**getAllParameters**()

analysis.models.**getAllPsfModels**()
Used to display choices in generate.py

analysis.models.**getExtra**()

analysis.models.**getFieldnames**()

analysis.models.**getGalParameters**()

analysis.models.**getModelCls**(*model*)
Return the corresponding class to the model specified in params

analysis.models.**getPsfModelCls**(*model*)
Return the corresponding psf class specified in params.

analysis.models.**getPsfParameters**()

class analysis.models.**model**(*params=None*, *params_omit=None*)
Bases: object

getGal(*params*)

getOmitFit()

getProfile(*params*)

omit_general = []

parameters = []

setOmitSpecific(*params_omit*)

shear(*gal*, *params*)

shift(*gal*, *params*)

class analysis.models.**psf_gaussian**(*params=None*)
Bases: *analysis.models.psf_model*

getProfile(*params*)

parameters = ['psf_flux', 'psf_fwhm', 'psf_e1', 'psf_e2']

class analysis.models.**psf_model**(*params=None*)
Bases: object

getProfile(*params*)

parameters = []

shearPsf(*params*)

class analysis.models.**psf_moffat**(*params=None*)
Bases: *analysis.models.psf_model*

getProfile(*params*)

parameters = ['psf_flux', 'psf_fwhm', 'psf_hlr', 'psf_beta', 'psf_e1', 'psf_e2']

analysis.draw module

analysis.defaults module

Some of the defaults that are used in the overall program.

`analysis.defaults.`**`getInitialValuesFit`**(*g_parameters*)
Return a dictionary containing the initial values to be used in the in the fitting of the parameters.

The dictionary is of the form: 'parameter_name:initial_value'

Args: g_parameters(*analysis.galfun.GParameters*): An object containing different

forms of the galaxy parameters.

Returns A dict.

`analysis.defaults.`**`getMaximums`**(*g_parameters*, *gal_image*)
Return a dictionary containing the minimum values to be used in the in the fitting of the parameters.

The dictionary is of the form: 'parameter_name:maximum_value'

Args: g_parameters(`galfun.GParameters`): An object containing different

forms of the galaxy parameters.

Returns A dict.

`analysis.defaults.`**`getMinimums`**(*g_parameters*, *gal_image*)
Return a dictionary containing the minimum values to be used in the in the fitting of the parameters.

The dictionary is of the form: 'parameter_name:initial_value'

Args: g_parameters(*analysis.galfun.GParameters*): An object containing different

forms of the galaxy parameters.

Returns A dict.

`analysis.defaults.`**`getSteps`**(*g_parameters*, *image_renderer*)
Return a dictionary containing the steps to be used in the derivatives of each parameter.

The dictionary is of the form: 'parameter_name:value_of_step'

Some parameter variations were copied from David's code suggestions.

Args: g_parameters(*analysis.galfun.GParameters*): An object containing different

forms of the galaxy parameters.

Returns A dict.

9.1.2 Module contents

Python Module Index

a

Index

N

O

P

R

S