VIS Python Package Documentation

Release 1.1

Sami-Matias Niemi

October 31, 2013

Contents

Author Sami-Matias Niemi

Contact s.niemi@ucl.ac.uk

issue 1.0

version 1.2

date June 13, 2013

This Python package VIS-PP provides subpackages and methods related to generating mock data and reducing it, the main consideration being the visible instrument (VIS) that is being developed for the Euclid telescope. The subpackages include methods to e.g. generate object catalogues, simulate VIS images, study radiation damage effects and fit new trap species, reduce and analyse data, and to include instrumental characteristics such as readout noise and CTI to "pristine" images generated with e.g. the GREAT10 photon shooting code. In addition, an algorithm to measure ellipticities of galaxies is also provided. Thus, this package tries to provide an end-to-end simulation chain for the VIS instrument.

The documentation is also available in PDF format, please download the VIS Python Package Manual.

CHAPTER 1

Installation

The VIS Python Package (VIS-PP) is held in a GIT repository. You can download or fork the repository here. The package contains a mixture of classes and scripts divided in subpackages based on the usage. Unfortunately, there is no official or preferred installation instructions yet. To get most scripts working you should place the path to the root directory of the package to your PYTHONPATH environment variable. In addition, it is useful to compile the Fortran code available in the fortran subdirectory with the following command:

```
f2py -c -m cdm03 cdm03.f90
```

and then copy the .so file to the CTI directory. Please note that f2py is available in the NumPy package, but you still need for example gFortran compiler to actually compile the Fortran source.

1.1 Dependencies

The VIS Python package depends heavily on other Python packages such as NumPy, SciPy, PyFITS, and matplotlib. Thus it is recommended that one installs a Python distribution like Enthought Python, which installs all dependencies at once.

CHAPTER 2

Instrument Model

The *support* subpackage contains functions that define the VIS instrument model. This model contains information about noise, dark, cosmic rays, radiation damage parameters, sky background, and pixel scale. For the Python documentation, please see:

2.1 VIS Instrument Model

The file provides a function that returns VIS related information such as pixel size, dark current, gain, zeropoint, and sky background.

requires NumPy

requires numexpr

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

version 0.7

`support.VISinstrumentModel.`**`CCDnonLinearityModel`**(*data*)

This function provides a non-linearity model for a VIS CCD273.

The non-linearity is modelled based on the results presented in MSSL/Euclid/TR/12001 issue 2. Especially Fig. 5.6, 5.7, 5.9 and 5.10 were used as an input data. The shape of the non-linearity is assumed to follow a parabola (although this parabola has a break, see the note below). The MSSL report indicates that the residual non-linearity is on the level of +/-25 DN or about +/- 0.04 per cent over the measured range. This function tries to duplicate this effect.

Note: There is a break in the model around 22000e. This is because the non-linearity measurements performed thus far are not extremely reliable below 10ke (< 0.5s exposure). However, the assumption is that at low counts the number of excess electrons appearing due to non-linearity should not be more than a few.

Parameters **data** (*ndarray*) – data to which the non-linearity model is being applied to

Returns input data after conversion with the non-linearity model

Return type float or ndarray

support.VISinstrumentModel.**CCDnonLinearityModelSinusoidal**(*data*, *amplitude*, *phase=0.49*, *multi=1.5*)

This function provides a theoretical non-linearity model based on sinusoidal error with a given amplitude, phase and number of waves (multi).

Parameters

- **data** (*ndarray*) – data to which the non-linearity model is being applied to
- **amplitude** (*float*) – amplitude of the sinusoidal wave
- **phase** (*float*) – phase of the sinusoidal wave
- **multi** (*float*) – the number of waves to have over the dynamical range of the CCD

Returns input data after conversion with the non-linearity model

Return type ndarray

support.VISinstrumentModel.**VISinformation**()

Returns a dictionary describing VIS. The following information is provided (id: value - reference):

```
apCorrection: 0.925969 - derived using VIS system PSF (see EUCL-MSS-RP-6-001)
aperture_size: 132.73228961416876 - derived (radiometric_model_reference_phase4_JA1
beta: 0.6 - CDM03 (Short et al. 2010)
bias: 1000.0 - ROE Requirements Specification (EUCL-MSS-RD-6-009)
cosmic_bkgd: 0.172 - derived  (radiometric_model_reference_phase4_JA110415_2_MSSL_v
dark: 0.001 - CCD spec EUCL-EST-RS-6-002
diameter: 1.3 - radiometric_model_reference_phase4_JA110415_2_MSSL_version
dob: 0 - CDM03 (Short et al. 2010)
e_adu: 3.1 - ROE Requirements Specification (EUCL-MSS-RD-6-009)
fullwellcapacity: 200000 - CCD spec (for simulator)
fwc: 200000 - CCD spec EUCL-EST-RS-6-002 (for CDM03)
gain: 3.1 - ROE Requirements Specification (EUCL-MSS-RD-6-009)
galaxy_fraction: 0.836 - radiometric_model_reference_phase4_JA110415_2_MSSL_version
magzero: 15861729325.3279 - derived, see belowCDM (VIS ETC)
ovrscanx: 20 - ROE Requirements Specification (EUCL-MSS-RD-6-009) (req: CalCD-B)
peak_fraction: 0.261179 - derived
pixel_size: 0.1 - CCD spec EUCL-EST-RS-6-002
prescanx: 50 - CCD spec EUCL-EST-RS-6-002 (also in CalCD-B)
rdose: 30000000000.0 - derived (above the PLM requirement)
readnoise: 4.5 - WL requirement (PERD R-VIS-P-021)
readout: 4.5 - WL requirement (PERD R-VIS-P-021)
readtime: 88.0 - derived; ROE Requirements Specification (EUCL-MSS-RD-6-009)
sfwc: 730000.0 - CDM03 (Short et al. 2010), see also the CCD spec EUCL-EST-RS-6-002
sky_background: 22.3203 - radiometric_model_reference_phase4_JA110415_2_MSSL_versio
sky_high: 21.7206 - radiometric_model_reference_phase4_JA110415_2_MSSL_version
sky_low: 22.9207 - radiometric_model_reference_phase4_JA110415_2_MSSL_version
st: 5e-06 - CDM03 (Short et al. 2010)
star_fraction: 0.928243 -  derived using VIS system PSF (see EUCL-MSS-RP-6-001)
svg: 1e-10 - CDM03 (Short et al. 2010)
t: 0.01024 - CDM03 (Short et al. 2010)
trapfile: cdm_euclid.dat - CDM03 (derived, refitted to CCD204 data)
vg: 6e-11 - CDM03 (Short et al. 2010)
vth: 11680000.0 - CDM03 (Short et al. 2010)
```

```
xsize: 2048 - CCD spec EUCL-EST-RS-6-002
ysize: 2066 - CCD spec EUCL-EST-RS-6-002
zeropoint: 25.50087633632 - VIS ETC
zeropointNoObscuration: 25.57991044453 - radiometric_model_reference_phase4_JA11041
zodiacal: 22.3203 - VIS ETC
```

The magzero was calculated as follows:

```
1./10**(-0.4*(25.45338546114)) = 15182880871.225231
```

The throughput input values are derived from two Excel Spreadsheets namely:

1.110413_EUC_TN_00051_SYS_PERF_REF_iss4.xlsx

2.radiometric_model_reference_phase4_JA110415_2_MSSL_version

Returns instrument model parameters

Return type dict

support.VISinstrumentModel.**testNonLinearity**()
A simple test to plot the current non-linearity model.

2.2 MTF and PSF

These functions can be used to address the CCD requirements, which are written for an MTF while PERD requirements are for a PSF.

Note: The frequency nu_0 is the Nyquist limit for the CCD, which is defined as: nu_0 = 1 / (2p) , where p is the pixel pitch in mm. Hence, for VIS the nu_0 is about 41.666.

Some links: http://www.dspguide.com/CH25.PDF http://home.fnal.gov/~neilsen/notebook/astroPSF/astroPSF.html#sec-5 http://mathworld.wolfram.com/FourierTransformGaussian.html https://github.com/GalSim-developers/GalSim/wiki/Optics-Module-usage http://www.e2v.com/e2v/assets/File/documents/imaging-space-and-scientific-sensors/Papers/ccdtn105.pdf http://aberrator.astronomy.net/html/mtf.html

sandbox.MTF.**FWHM**(*sigma*)
Calculates FWHM from sigma assuming a Gaussian profile.

Parameters **sigma** – standard deviation

Returns FWHM

sandbox.MTF.**GaussianAnimation**(*array_shape=(512, 512)*, *frames=15*)
Animation showing how MTF changes as the size of the Gaussian PSF grows.

Parameters

- **array_shape** – size of the simulation array
- **frames** – number of frames in the animation

Returns None

sandbox.MTF.**MTF**(*wf*)
Derives an MTF from pupil image.

Parameters **wf** –

Returns MTF

sandbox.MTF.**PSF**(*wf*, *array_shape=(512, 512)*, *flux=1.0*, *dx=1.0*)
Derives a PSF from pupil image.

Parameters

- **wf** –
- **array_shape** –
- **flux** –
- **dx** –

Returns

sandbox.MTF.**circular2DGaussian**(*array_size*, *sigma*)
Create a circular symmetric Gaussian centered on x, y.

Parameters **sigma** (*float*) – standard deviation of the Gaussian, note that sigma_x = sigma_y = sigma

Returns circular Gaussian 2D

Return type ndarray

sandbox.MTF.**compareAnalytical**(*array_shape=(256, 256)*, *nyq=16.0*)
Compares an analytical derivation of FWHM - MTF relation to numerical solution. This is only valid for Gaussian PSFs.

Parameters

- **array_shape** –
- **nyq** – cutout frequency (Nyquist = 4?)

Returns None

sandbox.MTF.**kxky**(*array_shape=(256, 256)*)
Return the tuple kx, ky corresponding to the DFT of a unit integer-sampled array of input shape.

Uses the SBProfile conventions for Fourier space, so k varies in approximate range (-pi, pi]. Uses the most common DFT element ordering conventions (and those of FFTW), so that *(0, 0)* array element corresponds to *(kx, ky) = (0, 0)*.

See also the docstring for np.fftfreq, which uses the same DFT convention, and is called here, but misses a factor of pi.

Adopts Numpy array index ordering so that the trailing axis corresponds to kx, rather than the leading axis as would be expected in IDL/Fortran. See docstring for numpy.meshgrid which also uses this convention.

@param array_shape the Numpy array shape desired for *kx, ky*.

sandbox.MTF.**pupilImage**(*array_shape=(512, 512)*, *size=1.0*, *dx=1.0*)
Generates a pupil image.

Parameters

- **array_shape** –

- **size** –
- **dx** –

Returns

`sandbox.MTF.`**`requirement`**(*alpha=0.2*, *w=12.9*)

Plots the requirements, both for PSF and MTF and compares them.

A fudge factor is required... should be pi / 2., but power laws are scale invariant so it doesn't matter.

Parameters

- **alpha** – power law slope
- **w** – fudge factor (wavenumber)

Returns None

`sandbox.MTF.`**`roll2d`**(*image*, *(iroll*, *jroll)*)

Perform a 2D roll (circular shift) on a supplied 2D numpy array, conveniently.

@param image the numpy array to be circular shifted. @param (iroll, jroll) the roll in the i and j dimensions, respectively.

@returns the rolled image.

2.3 Instrument Characteristics

The *postproc* subpackage contains methods related to either generating a CCD mosaics from simulated data that is in quadrants like the VIS reference simulator produces or including instrument characteristics to simulated images that contain only Poisson noise and background. For more detailed documentation of the Python classes, please see:

2.3.1 Postprocessing Tools

Inserting instrument characteristics

This file provides a class to insert instrument specific features to a simulated image. The VIS instrument model is taken from support.VISinstrumentModel.VISinformation function.

The class supports multiprocessing.

Note: The output images will be compressed with gzip to save disk space.

Warning: The logging module used does not work well with multiprocessing, but starts to write multiple entries after a while. This should be fixed.

requires PyFITS

requires NumPy

requires CDM03 (FORTRAN code, f2py -c -m cdm03 cdm03.f90)

author Sami-Matias Niemi

contact smn2@mssl.ucl.ac.uk

version 0.9

class postproc.postprocessing.**PostProcessing**(*values*, *work_queue*, *result_queue*, *seed*)

Euclid Visible Instrument postprocessing class. This class allows to add radiation damage (as defined by the CDM03 model) and add readout noise to a simulated image.

applyLinearCorrection(*image*)

Applies a linear correction after one forward readout through the CDM03 model.

Bristow & Alexov (2003) algorithm further developed for HST data processing by Massey, Rhodes et al.

Parameters **image** (*ndarray*) – radiation damaged image

Returns corrected image after single forward readout

Return type ndarray

applyRadiationDamage(*data*, *iquadrant=0*)

Apply radian damage based on FORTRAN CDM03 model. The method assumes that input data covers only a single quadrant defined by the iquadrant integer.

Parameters

- **data** (*ndarray*) – imaging data to which the CDM03 model will be applied to.
- **iquandrant** (*int*) – number of the quadrant to process

cdm03 - Function signature: sout = cdm03(sinp,iflip,jflip,dob,rdose,in_nt,in_sigma,in_tr,[xdim,ydim,zdim

Required arguments: sinp : input rank-2 array('f') with bounds (xdim,ydim) iflip : input int jflip : input int dob : input float rdose : input float in_nt : input rank-1 array('d') with bounds (zdim) in_sigma : input rank-1 array('d') with bounds (zdim) in_tr : input rank-1 array('d') with bounds (zdim)

Optional arguments: xdim := shape(sinp,0) input int ydim := shape(sinp,1) input int zdim := len(in_nt) input int

Return objects: sout : rank-2 array('f') with bounds (xdim,ydim)

Note: Because Python/NumPy arrays are different row/column based, one needs to be extra careful here. NumPy.asfortranarray will be called to get an array laid out in Fortran order in memory. Before returning the array will be laid out in memory in C-style (row-major order).

Returns image that has been run through the CDM03 model

Return type ndarray

applyReadoutNoise(*data*)

Applies readout noise. The noise is drawn from a Normal (Gaussian) distribution. Mean = 0.0, and std = readout.

Parameters **data** (*ndarray*) – input data to which the readout noise will be added to

Returns updated data, noise image

Return type dict

compressAndRemoveFile (*filename*)
This method compresses the given file using gzip and removes the parent from the file system.

Parameters **filename** (*str*) – name of the file to be compressed

Returns None

cutoutRegion (*data*)
Cuts out a region from the imaging data. The cutout region is specified by xstart/stop and ystart/stop that are read out from the self.values dictionary. Also checks if there are values that are above the given cutoff value and sets those pixels to a max value (default=33e3).

Parameters

- **data** (*ndarray*) – image array
- **max** (*int or float*) – maximum allowed value [default = 33e3]

Returns cut out image from the original data

Return type ndarray

discretisetoADUs (*data*)
Convert floating point arrays to integer arrays and convert to ADUs. Adds bias level after converting to ADUs.

Parameters **data** (*ndarray*) – data to be discretised to.

Returns discretised array in ADUs

Return type ndarray

generateCTImap (*CTIed*, *originalData*)
Calculates a map showing the CTI effect. This map is being generated by dividing radiation damaged image with the original data.

Parameters

- **CTIed** (*ndarray*) – Radiation damaged image
- **originalData** (*ndarray*) – Original image before any radiation damage

Returns CTI map (ratio of radiation damaged image and original data)

Return type ndarray

loadFITS (*filename*, *ext=0*)
Loads data from a given FITS file and extension.

Parameters

- **filename** (*str*) – name of the FITS file
- **ext** (*int*) – FITS header extension [default=0]

Returns data, FITS header, xsize, ysize

Return type dict

radiateFullCCD(*fullCCD*, *quads=(0, 1, 2, 3)*, *xsize=2048*, *ysize=2066*)
This routine allows the whole CCD to be run through a radiation damage mode. The routine takes into account the fact that the amplifiers are in the corners of the CCD. The routine assumes that the CCD is using four amplifiers.

Parameters

- **fullCCD** (*ndarray*) – image of containing the whole CCD
- **quads** (*list*) – quadrants, numbered from lower left

Returns radiation damaged image

Return type ndarray

run()
This is the method that will be called when multiprocessing.

writeFITSfile(*data*, *output*, *unsigned16bit=True*)
Write out FITS files using PyFITS.

Parameters

- **data** (*ndarray*) – data to write to a FITS file
- **output** (*string*) – name of the output file
- **unsigned16bit** (*bool*) – whether to scale the data using bzero=32768

Returns None

Generating a CCD mosaic

This file contains a class to create a single VIS CCD image from separate files one for each quadrant.

requires NumPy

requires PyFITS

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

To execute:

```
python tileCCD.py -f 'Q*science.fits' -e 1
```

where -f argument defines the input files to be tiled and the -e argument marks the FITS extension from which the imaging data are being read.

version 0.5

Todo

1. Does not deal properly with multiple WCSs coming in the different quadrants (should recalculate the centre of the CCD and modify the WCS accordingly).
2. Improve the history section.

---

class postproc.tileCCD.**tileCCD**(*inputs*, *log*)
Class to create a single VIS CCD image from separate quadrants files.

readData()
Reads in data from all the input files and the header from the first file. Input files are taken from the input dictionary given when class was initiated.

Subtracts the pre- and overscan regions if these were simulated. Takes into account which quadrant is being processed so that the extra regions are subtracted correctly.

runAll()
Wrapper to perform all class methods.

tileCCD(*xsize=2048*, *ysize=2066*)
Tiles quadrants to form a single CCD image.

Assume that the input file naming convention is Qx_CCDX_CCDY_name.fits.

Parameters

- **xsize** (*int*) – length of a quadrant in column direction
- **ysize** (*int*) – length of a quadrant in row direction

Returns image array of size (ysize*2, xsize*2)

Return type ndnarray

writeFITSfile(*data=None*, *unsigned16bit=True*)
Write out FITS files using PyFITS.

Parameters

- **data** (*ndarray*) – data to write to a FITS file, if None use self.data
- **unsigned16bit** (*bool*) – whether to scale the data using bzero=32768

Returns None

Generating an FPA mosaic

This file contains a class to create a single VIS FPA image from separate files one for each CCD.

requires NumPy

requires PyFITS

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

To execute:

```
python tileFPA.py -f 'CCD*science.fits' -e 1
```

where -f argument defines the input files to be tiled and the -e argument marks the FITS extension from which the imaging data are being read.

version 0.1

class postproc.tileFPA.**tileFPA**(*inputs*, *log*)
Class to create a single VIS FPA image from separate CCD files.

readData()
Reads in data from all the input files and the header from the first file. Input files are taken from the input dictionary given when class was initiated.

Subtracts the pre- and overscan regions if these were simulated. Takes into account which quadrant is being processed so that the extra regions are subtracted correctly.

runAll()
Wrapper to perform all class methods.

tileFPA(*xgap=1.643*, *ygap=8.116*)
Tiles quadrants to form a single CCD image.

Assume that the input file naming convention is Qx_CCDX_CCDY_name.fits.

Parameters

- **xgap** (*float*) – length of the gap between in mm two CCDs in column direction [default=1.643]
- **ygap** (*float*) – length of the gap between in mm two CCDs in row direction [default=8.116]

Returns FPA image array

Return type ndnarray

writeFITSfile(*data=None*, *unsigned16bit=True*, *ra=299.8680417*, *dec=40.73388889*)
Write out FITS files using PyFITS.

Parameters

- **data** (*ndarray*) – data to write to a FITS file, if None use self.data
- **unsigned16bit** (*bool*) – whether to scale the data using bzero=32768
- **ra** (*float*) – Right Ascension of the centre of the FPA [default = Cygnus A]
- **dec** (*float*) – Declination of the centre of the FPA [default = Cygnus A]

Returns None

CHAPTER 3

Exposure Times

The package provides a simple exposure time calculator (ETC) that allows to estimate a signal-to-noise ratio of an average galaxy or star with a given number of VIS exposures. The ETC also allows to calculate limiting magnitude or an exposure time for a given magnitude.

For the Python documentation, please see:

3.1 Exposure Time Calculator

3.1.1 Calculating Exposure Times and Limiting Magnitude

This file provides a simple functions to calculate exposure times or limiting magnitudes.

requires NumPy

requires SciPy

requires matplotlib

version 0.4

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

ETC.ETC.**SNR**(*info*, *magnitude=24.5*, *exptime=565.0*, *exposures=3*, *galaxy=True*, *background=True*, *diginoise=False*)

Calculates the signal-to-noise ratio for an object of a given magnitude in a given exposure time and a number of exposures.

Parameters

- **info** (*dict*) – instrumental information such as zeropoint and background
- **magnitude** (*float or ndarray*) – input magnitude of an object(s)
- **exptime** (*float*) – exposure time [seconds]
- **exposures** (*int*) – number of exposures [default = 3]

- **galaxy** (*boolean*) – whether the exposure time should be calculated for an average galaxy or a star. If galaxy=True then the fraction of flux within an aperture is lower than in case of a point source.
- **background** (*boolean*) – whether to include background from sky, instrument, and dark current [default=True]
- **diginoise** (*boolean*) – if the readout noise is undersampled or poorly resolved then the effective readout noise should be used [default = False]

Returns signal-to-noise ratio

Return type float or ndarray

ETC.ETC.**SNRproptoPeak**(*info*, *exptime=565.0*, *exposures=1*, *diginoise=False*, *server=False*)

Calculates the relation between the signal-to-noise ratio and the electrons in the peak pixel.

Parameters

- **info** (*dict*) – instrumental information such as zeropoint and background
- **exptime** (*float*) – exposure time [seconds]
- **exposures** (*int*) – number of exposures [default = 1]
- **diginoise** (*bool*) – if the readout noise is undersampled or poorly resolved then the effective readout noise should be used [default = False]
- **server** (*bool*) – whether to save the figure in a PNG or PDF format (the former can be used together with the WWW server)

Returns signal-to-noise ratio

Return type float or ndarray

ETC.ETC.**exposureTime**(*info*, *magnitude*, *snr=10.0*, *exposures=3*, *fudge=0.7*, *galaxy=True*, *diginoise=False*)

Returns the exposure time for a given magnitude.

Parameters

- **info** (*dict*) – information describing the instrument
- **magnitude** (*float or ndarray*) – the magnitude of the object
- **snr** (*float*) – signal-to-noise ratio required [default = 10.0].
- **exposures** (*int*) – number of exposures that the object is present in [default = 3]
- **fudge** (*float*) – the fudge parameter to which to use to scale the snr to SExtractor required [default = 0.7]
- **galaxy** (*boolean*) – whether the exposure time should be calculated for an average galaxy or a star. If galaxy=True then the fraction of flux within an aperture is lower than in case of a point source.
- **diginoise** (*boolean*) – if the readout noise is undersampled or poorly resolved then the effective readout noise should be used [default = False]

Returns exposure time [seconds]

Return type float or ndarray

ETC.ETC.**galaxyDetection**(*info*, *magnitude=24.5*, *exposures=3*, *exptime=565*)
Can be used to study the ghost contribution to the galaxy detection.

Parameters

- **info** –
- **magnitude** – object detection magnitude limit
- **exposures** – number of exposures to be combined
- **exptime** – individual exposure time

Returns number of electrons tolerated in the ghost

ETC.ETC.**limitingMagnitude**(*info*, *exp=565*, *snr=10.0*, *exposures=3*, *fudge=0.7*, *galaxy=True*, *diginoise=False*)
Calculates the limiting magnitude for a given exposure time, number of exposures and minimum signal-to-noise level to be reached.

Parameters

- **info** (*dict*) – instrumental information such as zeropoint and background
- **exp** (*float or ndarray*) – exposure time [seconds]
- **snr** (*float or ndarray*) – the minimum signal-to-noise ratio to be reached. .. Note:: This is couple to the fudge parameter: snr_use = snr / fudge
- **exposures** (*int*) – number of exposures [default = 3]
- **fudge** (*float*) – a fudge factor to divide the given signal-to-noise ratio with to reach to the required snr. This is mostly due to the fact that SExtractor may require a higher snr than what calculated otherwise.
- **galaxy** (*boolean*) – whether the exposure time should be calculated for an average galaxy or a star. If galaxy=True then the fraction of flux within an aperture is lower than in case of a point source.
- **diginoise** (*boolean*) – if the readout noise is undersampled or poorly resolved then the effective readout noise should be used [default = False]

Returns limiting magnitude given the input information

Return type float or ndarray

CHAPTER 4

Creating Object Catalogs

The *sources* subpackage contains a script to generate object catalogs with random x and y positions for stars and galaxies. The magnitudes of stars and galaxies are drawn from input distributions that are based on observations. As the number of stars depends on the galactic latitude, the script allows the user to use three different (30, 60, 90 degrees) angles when generating the magnitude distribution for stars (see the example plot below).

For the Python code documentation, please see:

4.1 Generating Object Catalogue

These simple functions can be used to generate different types of object catalogues that can then be used as an input for the VIS simulator.

Please note that the script requires files from the data folder. Thus, you should place the script to an empty directory and either copy or link to the data directory.

requires NumPy

requires SciPy

requires matplotlib

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

`sources.createObjectCatalogue.`**`drawFromCumulativeDistributionFunction`**(*cpdf*, *x*, *number*)

Draw a number of random x values from a cumulative distribution function.

Parameters

- **cpdf** (*numpy array*) – cumulative distribution function
- **x** (*numpy array*) – values of the abscissa
- **number** (*int*) – number of draws

Returns randomly drawn x value

Return type ndarray

sources.createObjectCatalogue.**generateCatalog**(***kwargs*)
Generate a catalogue of stars and galaxies that follow realistic number density functions.

Parameters **deg** – galactic latitude, either 30, 60, 90

sources.createObjectCatalogue.**plotCatalog**(*catalog*)
Plot the number of objects in the generated catalog. Will generate both cumulative and normal distributions for galaxies and stars separately.

Parameters **catalog** (*str*) – name of the catalogue file

Returns None

sources.createObjectCatalogue.**plotDistributionFunction**(*datax*, *datay*, *fitx*, *fity*, *output*)
Generates a simple plot showing the observed data points and the fit that was generated based on these data.

sources.createObjectCatalogue.**starCatalog**(*stars=400*, *xmax=2048*, *ymax=2066*, *magmin=23*, *magmax=26*)
Generate a catalog with stars at random positions.

sources.createObjectCatalogue.**starCatalogFixedMagnitude**(*stars=400*, *xmax=2048*, *ymax=2066*, *mag=18*, *random=True*, *pergrid=51*, *out='starsSameMag.dat'*)
Generate a catalog with stars of a given magnitude either at random positions or in a rectangular grid.

4.2 Generating Postage Stamp Images

This simple script can be used to generate postage stamp images from a larger mosaic. These images can then be used in the VIS simulator.

requires NumPy

requires PyFITS

requires VIS-PP

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

version 0.1

sources.generatePostageStamps.**generaPostageStamps**(*filename*, *catalog*, *maglimit=22.0*, *output='galaxy'*)
Generates postage stamp images from an input file given the input catalog position. The output

files are saved to FITS files.

Parameters

- **filename** (*str*) – name of the FITS file from which the postage stamps are extracted
- **catalog** (*str*) – name of the catalogue with x and y positions and magnitudes
- **maglimit** (*float*) – brighter galaxies than the given magnitude limit are extracted
- **output** (*str*) – name of the postage stamp prefix (will add a running number to this)

Returns None

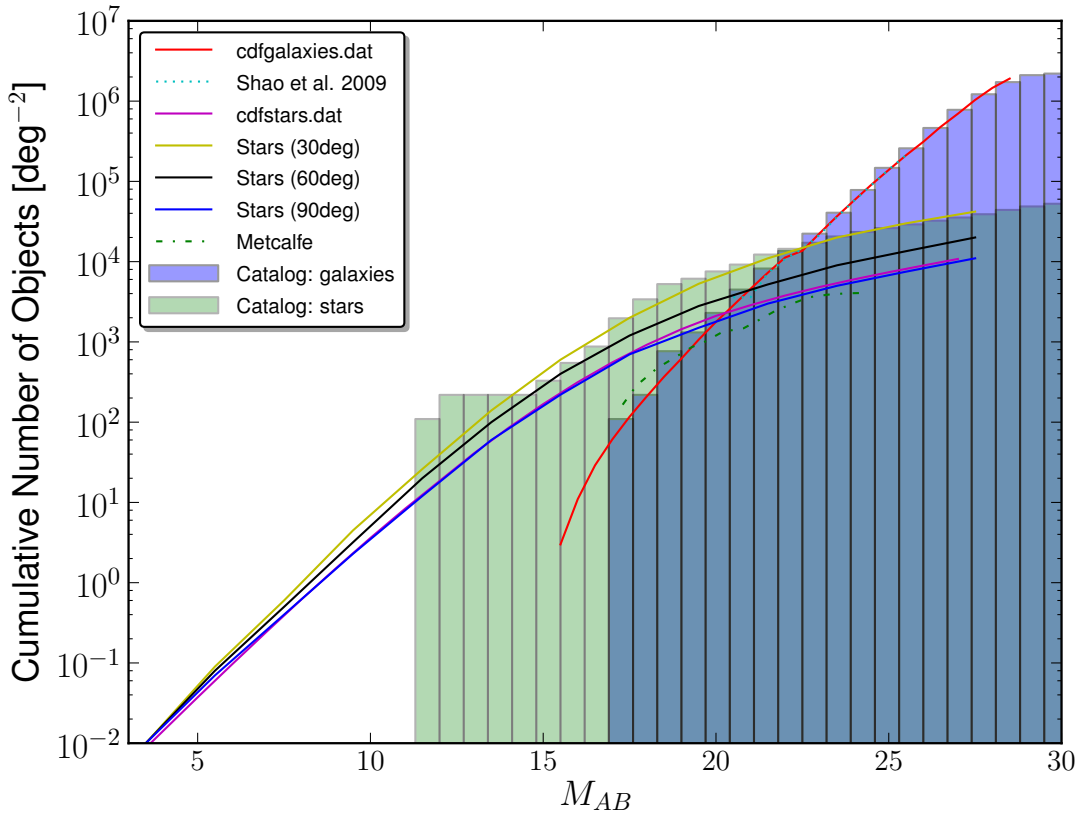

Figure 4.1: An example showing star and galaxy number counts in a source catalog suitable for VIS simulator. The solid lines show observations while the histograms show the distributions in the output catalogues.

Another way of creating a source catalog is to use *generateGalaxies* script in the *simulator* subpackage. This script depends on IRAF and uses the makeart IRAF package. There are many more options in this script, which basically just calls the makeart's gallist and starlist. For options, see IRAF's documentation.

CHAPTER 5

Creating Simulated Mock Images

The *simulator* subpackage contains scripts to generate simulated VIS images. Two different methods of generating mock images is provided. One which takes observed images (say from HST) as an input and another in which analytical profiles are used for galaxies. The former code is custom made while the latter relies hevily on IRAF's artdata package and mkobjects task.

The VIS reference simulator is the custom made with real observed galaxies as an input. The IRAF based simulator can be used, for example, to train algorithms to derive elliptiticy of an object. For more detailed documentation, please see:

5.1 Simulation tools

5.1.1 The Euclid Visible Instrument Image Simulator

This file contains an image simulator for the Euclid VISible instrument.

The approximate sequence of events in the simulator is as follows:

1. Read in a configuration file, which defines for example, detector characteristics (bias, dark and readout noise, gain, plate scale and pixel scale, oversampling factor, exposure time etc.).
2. Read in another file containing charge trap definitions (for CTI modelling).
3. Read in a file defining the cosmic rays (trail lengths and cumulative distributions).
4. Read in CCD offset information, displace the image, and modify the output file name to contain the CCD and quadrant information (note that VIS has a focal plane of 6 x 6 detectors).
5. Read in a source list and determine the number of different object types.
6. Read in a file which assigns data to a given object index.
7. Load the PSF model (a 2D map with a given over sampling or field dependent maps).
8. Generate a finemap (oversampled image) for each object type. If an object is a 2D image then calculate the shape tensor to be used for size scaling. Each type of an object is then placed onto its own finely sampled finemap.
9. Loop over the number of exposures to co-add and for each object in the object catalog:

- determine the number of electrons an object should have by scaling the object's magnitude with the given zeropoint and exposure time.
- determine whether the object lands on to the detector or not and if it is a star or an extended source (i.e. a galaxy).
- if object is extended determine the size (using a size-magnitude relation) and scale counts, convolve with the PSF, and finally overlay onto the detector according to its position.
- if object is a star, scale counts according to the derived scaling (first step), and finally overlay onto the detector according to its position.
- add a ghost of image of the object (scaled to the peak pixel of the object) [optional].

10. Apply calibration unit flux to mimic flat field exposures [optional].
11. Apply a multiplicative flat-field map to emulate pixel-to-pixel non-uniformity [optional].
12. Add a charge injection line (horizontal and/or vertical) [optional].
13. Add cosmic ray tracks onto the CCD with random positions but known distribution [optional].
14. Apply detector charge bleeding in column direction [optional].
15. Add constant dark current and background light from Zodiacal light [optional].
16. Include spatially uniform scattered light to the pixel grid [optional].
17. Add photon (Poisson) noise [optional]
18. Add cosmetic defects from an input file [optional].
19. Add pre- and overscan regions in the serial direction [optional].
20. Apply the CDM03 radiation damage model [optional].
21. Apply CCD273 non-linearity model to the pixel data [optional].
22. Add readout noise selected from a Gaussian distribution [optional].
23. Convert from electrons to ADUs using a given gain factor.
24. Add a given bias level and discretise the counts (the output is going to be in 16bit unsigned integers).
25. Finally the simulated image is converted to a FITS file, a WCS is assigned and the output is saved to the current working directory.

Warning: The code is still work in progress and new features are being added. The code has been tested, but nevertheless bugs may be lurking in corners, so please report any weird or inconsistent simulations to the author.

Dependencies

This script depends on the following packages:

requires PyFITS (tested with 3.0.6)

requires NumPy (tested with 1.6.1 and 1.7.1)

requires numexpr (tested with 2.0.1)

requires SciPy (tested with 0.10.1 and 0.12)

requires vissim-python package

Note: This class is not Python 3 compatible. For example, xrange does not exist in Python 3 (but is used here for speed and memory consumption improvements). In addition, at least the string formatting should be changed if moved to Python 3.x.

Note: CUDA acceleration requires an NVIDIA GPU that supports CUDA and PyFFT and PyCUDA packages. Note that the CUDA acceleration does not speed up the computations unless oversampled PSF is being used. If > 2GB of memory is available on the GPU, speed up up to a factor of 50 is possible.

Testing

Before trying to run the code, please make sure that you have compiled the cdm03bidir.f90 Fortran code using f2py (f2py -c -m cdm03bidir cdm03bidir.f90) and the the .so is present in the CTI folder. For testing, please run the unittest as follows:

```
python simulator.py -t
```

This will run the unittest and compare the result to a previously calculated image. Please inspect the standard output for results.

Running the test will produce an image representing VIS lower left (0th) quadrant of the CCD (x, y) = (0, 0). Because noise and cosmic rays are randomised one cannot directly compare the science outputs but we must rely on the outputs that are free from random effects. The data subdirectory contains a file named "nonoisenocrQ0_00_00testscience.fits", which is the comparison image without any noise or cosmic rays.

Benchmarking

A minimal benchmarking has been performed using the TESTSCIENCE1X section of the test.config input file:

```
Galaxy: 26753/26753 magnitude=26.710577 intscale=177.489159281 FWHM=0.117285374813 arc s
7091 objects were place on the detector

real        1m40.464s
user        1m38.389s
sys         0m1.749s
```

These numbers have been obtained with my desktop (3.4 GHz Intel Core i7 with 32GB 1600MHz DDR3) with 64-bit Python 2.7.3 installation. Further speed testing can be performed using the cProfile module as follows:

```
python -m cProfile -o vissim.profile simulator.py -c data/test.config -s TESTSCIENCE3X
```

and then analysing the results with e.g. snakeviz or RunSnakeRun.

The result above was obtained with nominally sampled PSF, however, that is only good for testing purposes. If instead one uses say four times over sampled PSF (TESTSCIENCE4x) then the execution time may increases substantially. This is mostly due to the fact that convolution becomes rather expensive

when done in the finely sampled PSF domain. If the four times oversampled case is run on CPU using SciPy.signal.fftconvolve for the convolution the run time is:

```
real        22m48.456s
user        21m58.730s
sys         0m50.171s
```

Instead, if we use an NVIDIA GPU for the convolution (and code that has not been optimised), the run time is:

```
real        12m7.745s
user        11m55.047s
sys         0m9.535s
```

Change Log

version 1.30

Version and change logs:

```
0.1: pre-development backbone.
0.4: first version with most pieces together.
0.5: this version has all the basic features present, but not fully tested.
0.6: implemented pre/overscan, fixed a bug when an object was getting close to the upper
     image it was not overlaid correctly. Included multiplicative flat fielding effect
0.7: implemented bleeding.
0.8: cleaned up the code and improved documentation. Fixed a bug related to checking if
     Improved the information that is being written to the FITS header.
0.9: fixed a problem with the CTI model swapping Q1 with Q2. Fixed a bug that caused the
     be identical for each quadrant even though Q1 and 3 needs the regions to be mirrore
1.0: First release. The code can now take an over sampled PSF and use that for convoluti
     to the header.
1.05: included an option to add flux from the calibration unit to allow flat field expos
      Now scaled the number of cosmic rays with the exposure time so that 10s flats have
      of cosmic ray tracks.
1.06: changed how stars are laid down on the CCD. Now the PSF is interpolated to a new o
      oversampled frame after which it is downsampled to the CCD grid. This should incre
      accuracy.
1.07: included an option to apply non-linearity model. Cleaned the documentation.
1.08: optimised some of the operations with numexpr (only a minor improvement).
1.1: Fixed a bug related to adding the system readout noise. In previous versions the re
     being underestimated due to the fact that it was included as a variance not standar
1.2: Included a spatially uniform scattered light. Changed how the image pixel values ar
     deriving the Poisson noise. Included focal plane CCD gaps. Included a unittest.
1.21: included an option to exclude cosmic background; separated dark current from backg
1.25: changed to a bidirectional CDM03 model. This allows different CTI parameters to be
      and serial directions.
1.26: an option to include ghosts from the dichroic. The ghost model is simple and does
      the fact that the ghost depends on the focal plane position. Fixed an issue with i
      (zero indexing). Now input catalogue values agree with DS9 peak pixel locations.
1.27: Convolution can now be performed using a GPU using CUDA if the hardware is availab
      is now controlled using a single parameter. Change from 'full' to 'same' as full p
      over 'same'. In principle the 'valid' mode would give all valid information, but i
      truncated convolved galaxy images if the image and the kernel are of similar size.
1.28: Moved the cosmic ray event generation to a separate class for easier management. U
      generate more realistic looking cosmic rays. Included a charge diffusion smoothing
      to mimic the spreading of charge within the CCD. This is closer to reality, but pr
```

```
      given geometric arguments (charge diffusion kernels are measured using light comir
      the CCD, while cosmic rays can come from any direction and penetrate to any depth)
1.29: Fixed a bug in the object pixel coordinates for simulations other than the 0, 0 CC
      were incorrectly taken into account (forcing the objects to be about 100 pixels of
1.30: now nocti files contain ADC offset and readnoise, the same as the true output if C
```

Future Work

Todo

1. objects.dat is now hard coded into the code, this should be read from the config file
2. implement spatially variable PSF and ghost model
3. test that the WCS is correctly implemented and allows CCD offsets
4. charge injection line positions are now hardcoded to the code, read from the config file
5. include rotation in metrology
6. implement optional dithered offsets
7. CCD273 has 4 pixel row gap between the top and bottom half, this is not taken into account in coordinate shifts

---

Contact Information

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

class `simulator.simulator.`**`VISsimulator`**(*opts*)

Euclid Visible Instrument Image Simulator

The image that is being build is in:

`self.image`

Parameters **opts** (*OptionParser instance*) – OptionParser instance

`addChargeInjection`()

Add either horizontal or vertical charge injection line to the image.

`addCosmicRays`()

Add cosmic rays to the arrays based on a power-law intensity distribution for tracks. Cosmic ray properties (such as location and angle) are chosen from random Uniform distribution. For details, see the documentation for the cosmicrays class in the support package.

`addLampFlux`()

Include flux from the calibration source.

`addObjects`()

Add objects from the object list to the CCD image (self.image).

Scale the object's brightness in electrons and size using the input catalog magnitude. The size-magnitude scaling relation is taken to be the equation B1 from Miller et al. 2012 (1210.8201v1; Appendix "prior distributions"). The spread is estimated from Figure 1 to be around 0".1 (1 sigma). A random draw from a Gaussian distribution with spread of 0".1 arc sec is performed so that galaxies of the same brightness would not be exactly the same size.

> **Warning:** If random Gaussian dispersion is added to the scale-magnitude relation, then one cannot simulate several dithers. The random dispersion can be turned off by setting random=no in the configuration file so that dithers can be simulated and co-added correctly.

addObjectsAndGhosts()

Add objects from the object list and associated ghost images to the CCD image (self.image).

Scale the object's brightness in electrons and size using the input catalog magnitude. The size-magnitude scaling relation is taken to be the equation B1 from Miller et al. 2012 (1210.8201v1; Appendix "prior distributions"). The spread is estimated from Figure 1 to be around 0".1 (1 sigma). A random draw from a Gaussian distribution with spread of 0".1 arc sec is performed so that galaxies of the same brightness would not be exactly the same size.

> **Warning:** If random Gaussian dispersion is added to the scale-magnitude relation, then one cannot simulate several dithers. The random dispersion can be turned off by setting random=no in the configuration file so that dithers can be simulated and co-added correctly.

addPreOverScans()

Add pre- and overscan regions to the self.image. These areas are added only in the serial direction. Because the 1st and 3rd quadrant are read out in to a different serial direction than the nominal orientation, in these images the regions are mirrored.

The size of prescan and overscan regions are defined by the prescanx and overscanx keywords, respectively.

applyBias()

Adds a bias level to the image being constructed.

The value of bias is read from the configure file and stored in the information dictionary (key bias).

applyBleeding()

Apply bleeding along the CCD columns if the number of electrons in a pixel exceeds the full-well capacity.

Bleeding is modelled in the parallel direction only, because the CCD273s are assumed not to bleed in serial direction.

Returns None

applyCosmetics()

Apply cosmetic defects described in the input file.

> **Warning:** This method does not work if the input file has exactly one line.

applyCosmicBackground()
Apply dark the cosmic background. Scales the background with the exposure time.

Additionally saves the image without noise to a FITS file.

applyDarkCurrent()
Apply dark current. Scales the dark with the exposure time.

Additionally saves the image without noise to a FITS file.

applyFlatfield()
Applies multiplicative flat field to emulate pixel-to-pixel non-uniformity.

Because the pixel-to-pixel non-uniformity effect (i.e. multiplicative) flat fielding takes place before CTI and other effects, the flat field file must be the same size as the pixels that see the sky. Thus, in case of a single quadrant (x, y) = (2048, 2066).

applyNonlinearity()
Applies a CCD273 non-linearity model to the image being constructed.

applyPoissonNoise()
Add Poisson noise to the image.

applyRadiationDamage()
Applies CDM03 radiation model to the image being constructed.

See Also:

Class :*CDM03*

applyReadoutNoise()
Applies readout noise to the image being constructed.

The noise is drawn from a Normal (Gaussian) distribution with average=0.0 and std=readout noise.

applyScatteredLight()
Adds spatially uniform scattered light to the image.

configure()
Configures the simulator with input information and creates and empty array to which the final image will be build on.

discretise(*max=65535*)
Converts a floating point image array (self.image) to an integer array with max values defined by the argument max.

Parameters **max** (*float*) – maximum value the the integer array may contain [default 65k]

Returns None

electrons2ADU()
Convert from electrons to ADUs using the value read from the configuration file.

generateFinemaps()
Generates finely sampled images of the input data.

objectOnDetector(*object*)
Tests if the object falls on the detector area being simulated.

Parameters **object** (*list*) – object to be placed to the self.image being simulated.

Returns whether the object falls on the detector or not

Return type bool

overlayToCCD(*data*, *obj*)
Overlay data from a source object onto the self.image.

Parameters

- **data** (*ndarray*) – ndarray of data to be overlaid on to self.image
- **obj** (*list*) – object information such as x,y position

processConfigs()
Processes configuration information and save the information to a dictionary self.information.

The configuration file may look as follows:

```
[TEST]
quadrant = 0
CCDx = 0
CCDy = 0
CCDxgap = 1.643
CCDygap = 8.116
xsize = 2048
ysize = 2066
prescanx = 50
ovrscanx = 20
fullwellcapacity = 200000
dark = 0.001
readout = 4.5
bias = 1000.0
cosmic_bkgd = 0.182758225257
e_ADU = 3.1
injection = 150000.0
magzero = 15182880871.225231
exposures = 1
exptime = 565.0
rdose = 8.0e9
RA = 145.95
DEC = -38.16
sourcelist = data/source_test.dat
PSFfile = data/interpolated_psf.fits
parallelTrapfile = data/cdm_euclid_parallel.dat
serialTrapfil e= data/cdm_euclid_serial.dat
cosmeticsFile = data/cosmetics.dat
flatfieldfile = data/VISFlatField2percent.fits
output = test.fits
addSources = yes
noise = yes
cosmetics = no
chargeInjectionx = no
chargeInjectiony = no
radiationDamage = yes
cosmicRays = yes
overscans = yes
bleeding = yes
flatfieldM = yes
random = yes
```

```
background = yes
ghosts = no
```

For explanation of each field, see /data/test.config. Note that if an input field does not exist, then the values are taken from the default instrument model as described in support.VISinstrumentModel.VISinformation(). Any of the defaults can be overwritten by providing a config file with a correct field name.

readConfigs()
Reads the config file information using configParser.

readCosmicRayInformation()
Reads in the cosmic ray track information from two input files.

Stores the information to a dictionary called cr.

readObjectlist()
Reads object list using numpy.loadtxt, determines the number of object types, and finds the file that corresponds to a given object type.

The input catalog is assumed to contain the following columns:

1.x coordinate

2.y coordinate

3.apparent magnitude of the object

4.type of the object [0=star, number=type defined in the objects.dat]

5.rotation [0 for stars, [0, 360] for galaxies]

This method also displaces the object coordinates based on the quadrant and the CCD to be simulated.

Note: If even a single object type does not have a corresponding input then this method forces the program to exit.

---

readPSFs()
Reads in a PSF from a FITS file.

Note: at the moment this method supports only a single PSF file.

---

simulate()
Create a single simulated image of a quadrant defined by the configuration file. Will do all steps defined in the config file sequentially.

Returns None

smoothingWithChargeDiffusion(*image*, *sigma=(0.32*, *0.32)*)
Smooths a given image with a gaussian kernel with widths given as sigmas. This smoothing can be used to mimic charge diffusion within the CCD.

The default values are from Table 8-2 of CCD_273_Euclid_secification_1.0.130812.pdf converted to sigmas (FWHM / (2sqrt(2ln2)) and rounded up to the second decimal.

Note: This method should not be called for the full image if the charge spreading has already been taken into account in the system PSF to avoid double counting.

Parameters

- **image** (*ndarray*) – image array which is smoothed with the kernel
- **sigma** – widths of the gaussian kernel that approximates the charge diffusion [0.32, 0.32].
- **sigma** – tuple

Returns smoothed image array

Return type ndarray

writeFITSfile(*data*, *filename*, *unsigned16bit=False*)
Writes out a simple FITS file.

Parameters

- **data** (*ndarray*) – data to be written
- **filename** (*str*) – name of the output file
- **unsigned16bit** (*bool*) – whether to scale the data using bzero=32768

Returns None

writeOutputs()
Writes out a FITS file using PyFITS and converts the image array to 16bit unsigned integer as appropriate for VIS.

Updates header with the input values and flags used during simulation.

5.1.2 Generating Mock Objects with IRAF

This script provides a class that can be used to generate objects such as galaxies using IRAF.

requires PyRAF

requires PyFITS

requires NumPy

author Sami-Matias Niemi

contact smn2@mssl.ucl.ac.uk

version 0.1

class simulator.generateGalaxies.**generateFakeData**(*log*, ***kwargs*)
Generates an image frame with stars and galaxies using IRAF's artdata.

addObjects(*inputlist='galaxies.dat'*)
Add object(s) from inputlist to the output image.

Parameters **inputlist** (*str*) – name of the input list

createGalaxylist (*ngalaxies=150*, *output='galaxies.dat'*)
Generates an ascii file with uniform random x and y positions. The magnitudes of galaxies are taken from an isotropic and homogeneous power-law distribution.

The output ascii file contains the following columns: xc yc magnitude model radius ar pa <save>

Parameters

- **ngalaxies** (*int*) – number of galaxies to include
- **output** (*str*) – name of the output ascii file

createStarlist (*nstars=20*, *output='stars.dat'*)
Generates an ascii file with uniform random x and y positions. The magnitudes of stars are taken from an isotropic and homogeneous power-law distribution.

The output ascii file contains the following columns: xc yc magnitude

Parameters

- **nstars** (*int*) – number of stars to include
- **output** (*str*) – name of the output ascii file

maskCrazyValues (*filename=None*)
For some reason mkobjects sometimes adds crazy values to an image. This method tries to remove those values and set them to more reasonable ones. The values > 65k are set to the median of the image.

Parameters **filename** (*str*) – name of the input file to modify [default = self.settings['output']]

Returns None

runAll (*nostars=True*)
Run all methods sequentially.

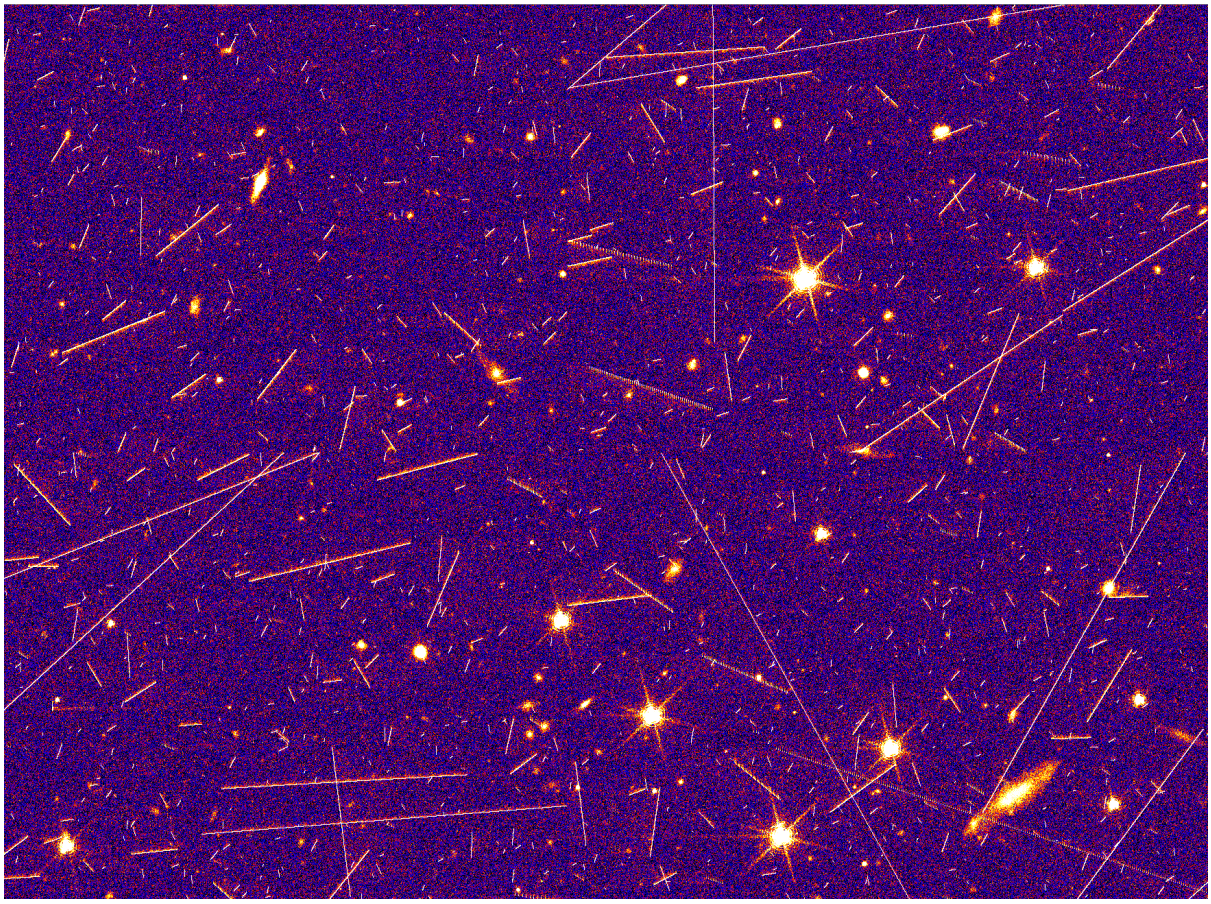

Figure 5.1: An example image generated with the VIS simulator. The image shows a part of a single CCD. The image is on a logarithmic stretch.

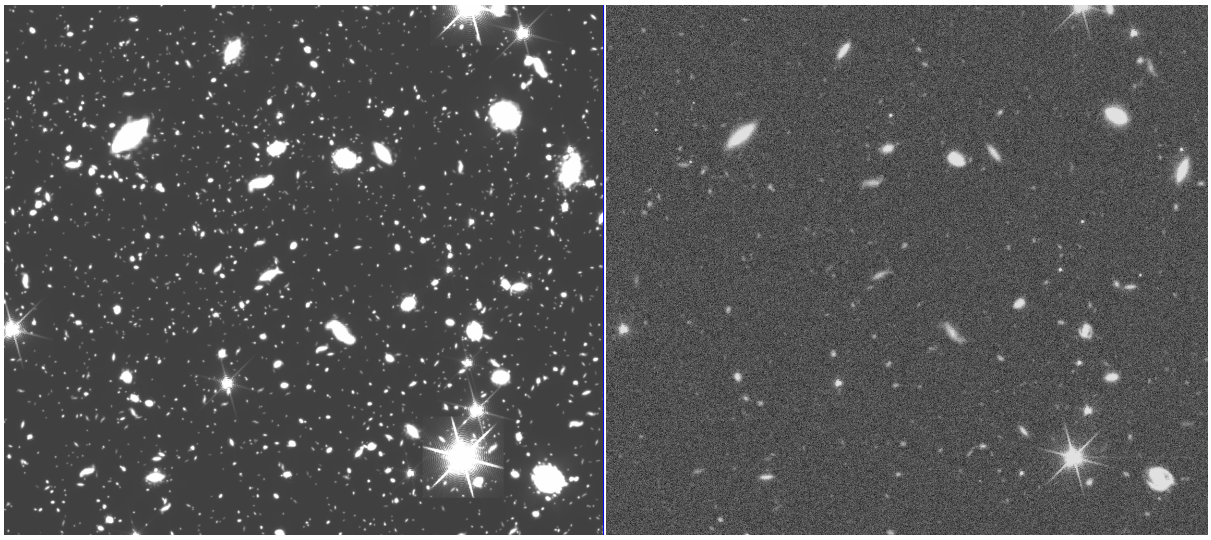

Figure 5.2: A simulated image with no noise (left) and with noise (right). The simulated image with noise includes: Zodiacal background, scattered light, readout and Poisson noise, CTI, etc. Both images are on logarithmic stretch.

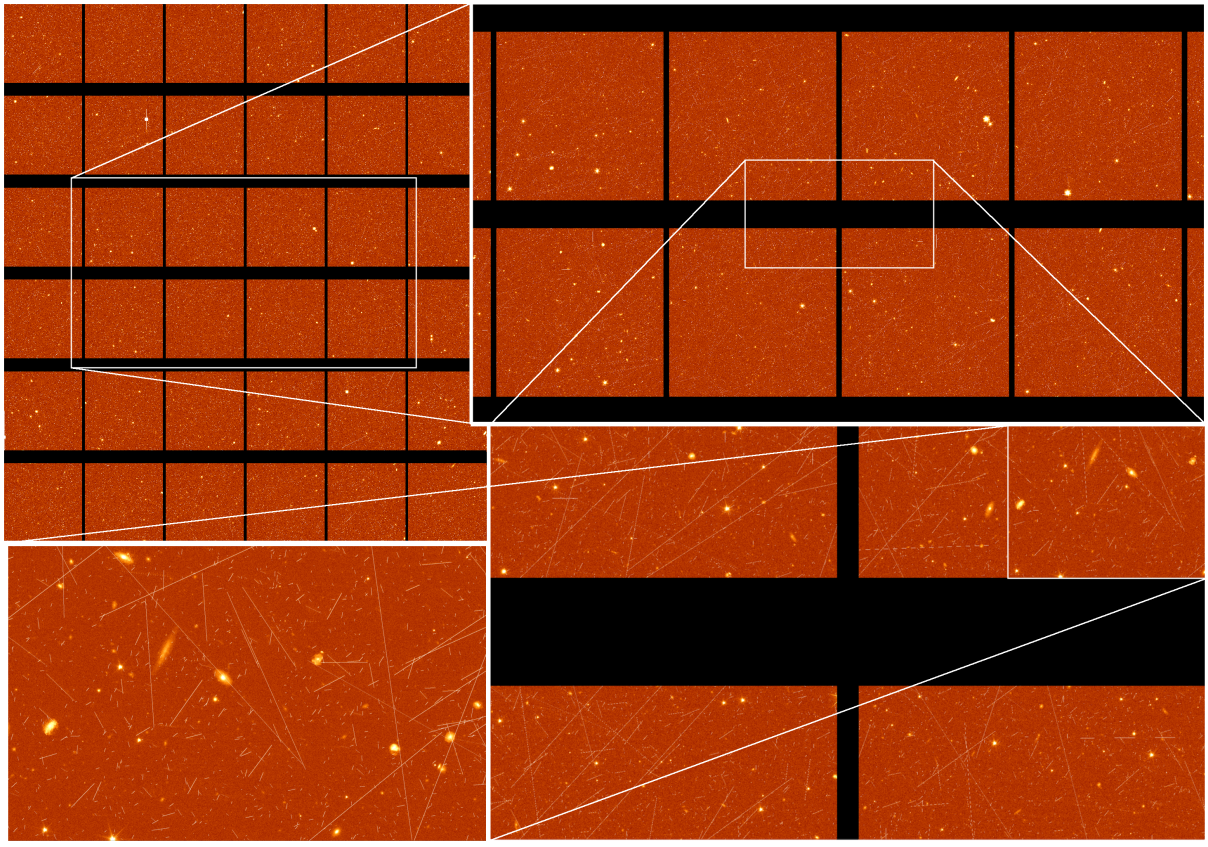

Figure 5.3: Full focal plane of VIS with zoom-in sections. The data were generated with the VIS simulator. The data are on log-scale.

CHAPTER 6

Data reduction

The *reduction* subpackage contains a simple script to reduce VIS data. For more detailed documentation of the classes, please see:

6.1 Data reduction tools

6.1.1 VIS Data Reduction and Processing

This simple script can be used to reduce (simulated) VIS data.

The script was initially written for reducing a single CCD data. However, since the version 0.5 the script tries to guess if the input is a single quadrant then reduce correctly.

The script performs the following data reduction steps:

```
1 Bias correction
2 CTI correction
3 Flat fielding (only if an input file is provided)
```

To Run:

```
python reduceVISdata.py -i VISCCD.fits -b superBiasVIS.fits -f SuperFlatField.fits
```

requires PyFITS

requires NumPy

requires CDM03 (FORTRAN code, f2py -c -m cdm03 cdm03.f90)

author Sami-Matias Niemi

contact smn2@mssl.ucl.ac.uk

version 0.5

Todo

1. implement background/sky subtraction

class reduction.reduceVISdata.**reduceVISdata**(*values*, *log*)
Simple class to reduce VIS data.

applyCTICorrection()
Applies a CTI correction in electrons using CDM03 CTI model. Converts the data to electrons using the gain value given in self.values. The number of forward reads is defined by self.values['order'] parameter.

Bristow & Alexov (2003) algorithm further developed for HST data processing by Massey, Rhodes et al.

There is probably an excess of .copy() calls here, but I had some problems when calling the Fortran code so I added them for now.

flatfield()
Take into account pixel-to-pixel non-uniformity through multiplicative flat fielding.

subtractBias()
Simply subtracts self.bias from the input data.

writeFITSfile()
Write out FITS files using PyFITS.

6.2 VIS data reduction studies

6.2.1 Bias Calibration

This simple script can be used to study the number of bias frames required to meet the VIS calibration requirements.

The following requirements related to the bias calibration has been taken from GDPRD.

R-GDP-CAL-052: The contribution of the residuals of VIS bias subtraction to the *error on the determination of each ellipticity component* of the local PSF shall not exceed 3x10-5 (one sigma).

R-GDP-CAL-062: The contribution of the residuals of VIS bias subtraction to the *relative error* sigma(R2)/R2 on the determination of the local PSF R2 shall not exceed 1x10-4 (one sigma).

requires PyFITS

requires NumPy

requires matplotlib

requires VISsim-Python

version 0.95

author Sami-Matias Niemi

contact smn2@mssl.ucl.ac.uk

analysis.biasCalibration.**addReadoutNoise**(*data*, *readnoise=4.5*, *gain=3.1*, *number=1*)
Add readout noise to the input data. The readout noise is the median of the number of frames.

Parameters

- **data** (*ndarray*) – input data to which the readout noise will be added to [ADUs]
- **readnoise** (*float*) – standard deviation of the read out noise [electrons]
- **gain** (*float*) – the gain factor that is used to convert electrons to ADUs
- **number** (*int*) – number of read outs to median combine before adding to the data [default=1]

Returns data + read out noise

Return type ndarray [same as input data]

analysis.biasCalibration.**findTolerableErrorPiston**(*log*, *file='data/psf12x.fits'*, *oversample=12.0*, *samples=12*, *psfs=4000*, *sigma=0.36*, *iterations=5*, *debug=False*)

Calculate ellipticity and size for PSFs of different scaling when there is a residual bias offset.

Parameters sigma (*float*) – 1sigma radius of the Gaussian weighting function for shape measurements

analysis.biasCalibration.**findTolerableErrorSlope**(*log*, *file='data/psf12x.fits'*, *oversample=12.0*, *samples=12*, *psfs=4000*, *sigma=0.36*, *iterations=5*, *pixels=60*)

Calculate ellipticity and size for PSFs of different scaling when there is a residual bias slope.

Parameters sigma (*float*) – 1sigma radius of the Gaussian weighting function for shape measurements

analysis.biasCalibration.**pistonKnowledge**(*log*, *file='data/psf2x.fits'*, *oversample=2.0*, *psfs=1000*, *sigma=0.36*, *iterations=4*, *debug=False*)

analysis.biasCalibration.**plotDeltaEs**(*deltae1*, *deltae2*, *deltae*, *output*, *title=''*, *ymax=8*, *req=3*)

Generates a simple plot showing the errors in the ellipticity components.

analysis.biasCalibration.**plotEs**(*deltae1*, *deltae2*, *deltae*, *output*, *title=''*)

Generates a simple plot showing the ellipticity components.

analysis.biasCalibration.**plotNumberOfFramesDelta**(*results*, *timeStamp=False*)

Creates a simple plot to combine and show the results for errors (delta).

Parameters

- **results** (*dict*) – results to be plotted
- **timeStamp** (*bool*) – whether to include a time stamp in the output image

analysis.biasCalibration.**plotNumberOfFramesSigma**(*results*, *reqe=3e-05*, *reqr2=0.0001*, *shift=0.1*, *timeStamp=False*)

Creates a simple plot to combine and show the results.

Parameters

- **results** (*dict*) – results to be plotted
- **req** (*float*) – the requirement
- **ymax** (*int or float*) – maximum value to show on the y-axis
- **shift** (*float*) – the amount to shift the e2 results on the abscissa (for clarity)
- **timeStamp** (*bool*) – whether to include a time stamp in the output image

analysis.biasCalibration.**simpleAnalytical**(*offset=1500*, *size=(50, 50)*, *readnoise=4.5*, *gain=3.1*, *req=0.6*)

A simple function to test the area of pixels needed (defined by size) to derive the pixel offset to the level of required number of electrons given the readout noise and the gain of the system.

Parameters

- **offset** (*int*) – the offset level in electrons [default = 1500]
- **size** (*tuple*) – area describing the number of pixels available [default = (50, 50)]
- **readnoise** (*float*) – readout noise of the full detection chain [default = 4.5]
- **gain** (*float*) – gain of the detection system [default = 3.1]
- **req** (*float*) – required level to reach in electrons [default = 0.1]

Returns none

analysis.biasCalibration.**testBiasCalibrationDelta**(*log*, *numdata=2066*, *floor=995*, *xsize=2048*, *ysize=2066*, *order=3*, *biases=15*, *surfaces=100*, *file='psf1x.fits'*, *psfs=500*, *psfscale=1000.0*, *debug=False*, *plots=False*)

Derive the PSF ellipticities for a given number of random surfaces with random PSF positions and a given number of biases median combined and compare to the nominal PSF ellipticity.

This function can be used to derive the error (delta) in determining ellipticity and size given a reference PSF.

Choices that need to be made and effect the results:

1.bias surface that is assumed (amplitude, complexity, etc.)

2.whether the order of the polynomial surface to be fitted is known or not

3.size of the Gaussian weighting function when calculating the ellipticity components

There are also other choices such as the number of PSFs and scaling and the random numbers generated for the surface that also affect the results, however, to a lesser degree.

Generates a set of plots that can be used to inspect the simulation.

analysis.biasCalibration.**testBiasCalibrationSigma**(*log*, *numdata=2066*, *floor=3500*, *xsize=2048*, *ysize=2066*, *order=3*, *biases=15*, *surfaces=100*, *file='psf1x.fits'*, *psfs=500*, *psfscale=100000.0*, *gain=3.1*, *debug=False*, *plots=True*)

Derive the PSF ellipticities for a given number of random surfaces with random PSF positions and a given number of biases median combined.

This function is to derive the the actual values so that the knowledge (variance) can be studied.

Choices that need to be made and effect the results:

1.bias surface that is assumed (amplitude, complexity, etc.)

2.whether the order of the polynomial surface to be fitted is known or not

3.size of the Gaussian weighting function when calculating the ellipticity components

There are also other choices such as the number of PSFs and scaling and the random numbers generated for the surface that also affect the results, however, to a lesser degree.

Generates a set of plots that can be used to inspect the simulation.

6.2.2 Flat Field Calibration

This simple script can be used to study the number of flat fields required to meet the VIS calibration requirements.

The following requirements related to the flat field calibration has been taken from GDPRD.

R-GDP-CAL-054: The contribution of the residuals of VIS flat-field correction to the error on the determination of each ellipticity component of the local PSF shall not exceed 3x10-5 (one sigma).

R-GDP-CAL-064: The contribution of the residuals of VIS flat-field correction to the relative error on the determination of the local PSF R2 shall not exceed 1x10-4 (one sigma).

Note: The amount of cosmic rays in the simulated input images might be too low, because the exposure was set to 10 seconds and cosmic rays were calculated based on this. However, in reality the readout takes about 80 seconds. Thus, the last row is effected by cosmic a lot more than by assuming a single 10 second exposure.

---

requires PyFITS

requires NumPy

requires SciPy

requires matplotlib

requires VISsim-Python

version 0.96

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

analysis.FlatfieldCalibration.**findTolerableError**(*log*, *file='data/psf4x.fits'*, *oversample=4.0*, *psfs=10000*, *iterations=7*, *sigma=0.75*)

Calculate ellipticity and size for PSFs of different scaling when there is a residual pixel-to-pixel variations.

analysis.FlatfieldCalibration.**generateResidualFlatField**(*files='Q0*flatfield*.fits'*, *combine=77*, *lampfile='data/VIScalibrationUnitflux.fit*, *reference='data/VISFlatField1percent.fi*, *gain=3.5*, *plots=False*, *debug=False*)

Generate a median combined flat field residual from given input files.

Randomly draws a given number (kw combine) of files from the file list identified using the files kw. Median combine all files before the lamp profile given by lampfile kw is being divided out. This will produce a derived flat field. This flat can be compared against the reference that was used to produce the initial data to derive a residual flat that describes the error in the flat field that was derived.

Parameters

- **files** (*str*) – wildcard flagged name identifier for the FITS files to be used for generating a flat
- **combine** (*int*) – number of files to median combine
- **lampfile** (*str*) – name of the calibration unit flux profile FITS file
- **reference** (*str*) – name of the reference pixel-to-pixel flat field FITS file
- **gain** (*float*) – gain factor [e/ADU]
- **plots** (*boolean*) – whether or not to generate plots
- **debug** (*boolean*) – whether or not to produce output FITS files

Warning: Remember to use an appropriate lamp and reference files so that the error in the derived flat field can be correctly calculated.

Returns residual flat field (difference between the generated flat and the reference)

Return type ndarray

analysis.FlatfieldCalibration.**plotNumberOfFrames**(*results*, *reqe=3e-05*, *reqr2=0.0001*, *shift=0.1*, *outdir='results'*, *timeStamp=False*)

Creates a simple plot to combine and show the results.

Parameters

- **res** (*list*) – results to be plotted [results dictionary, reference values]
- **reqe** (*float*) – the requirement for ellipticity [default=3e-5]
- **reqr2** (*float*) – the requirement for size R2 [default=1e-4]
- **shift** (*float*) – the amount to shift the e2 results on the abscissa (for clarity)
- **outdir** (*str*) – output directory to which the plots will be saved to
- **timeStamp** (*bool*) – whether or not to include a time stamp to the output image

Returns None

analysis.FlatfieldCalibration.**testFlatCalibration**(*log*, *flats*, *surfaces=10*, *file='data/psf1x.fits'*, *psfs=5000*, *sigma=0.75*, *iterations=7*, *weighting=True*, *plot=False*, *debug=False*)

Derive the PSF ellipticities for a given number of random surfaces with random PSF positions and a given number of flat fields median combined.

This function is to derive the the actual values so that the knowledge (variance) can be studied.

analysis.FlatfieldCalibration.**testNoFlatfieldingEffects**(*log*, *file='data/psf1x.fits'*, *oversample=1.0*, *psfs=500*)

Calculate ellipticity and size variance and error in case of no pixel-to-pixel flat field correction.

6.2.3 Non-linearity I: Detection Chain

This simple script can be used to study the error in the non-linearity correction that can be tolerated given the requirements.

The following requirements related to the non-linearity have been taken from GDPRD.

R-GDP-CAL-058: The contribution of the residuals of the non-linearity correction on the error on the determination of each ellipticity component of the local PSF shall not exceed 3x10**-5 (one sigma).

R-GDP-CAL-068: The contribution of the residuals of the non-linearity correction on the error on the relative error sigma(R**2)/R**2 on the determination of the local PSF R**2 shall not exceed 1x10**-4 (one sigma).

requires PyFITS

requires NumPy

requires SciPy

requires matplotlib

requires VISsim-Python

version 0.97

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

analysis.nonlinearityCalibration.**plotResults**(*results*, *reqe=3e-05*, *reqr2=0.0001*, *outdir='results'*, *timeStamp=False*)

Creates a simple plot to combine and show the results.

Parameters

- **res** (*list*) – results to be plotted [reference values, results dictionary]
- **reqe** (*float*) – the requirement for ellipticity [default=3e-5]
- **reqr2** (*float*) – the requirement for size R2 [default=1e-4]
- **outdir** (*str*) – output directory to which the plots will be saved to

Returns None

analysis.nonlinearityCalibration.**testNonlinearity**(*log*, *file='data/psf12x.fits'*, *oversample=12.0*, *sigma=0.75*, *phs=0.98*, *phases=None*, *psfs=5000*, *amps=12*, *multiplier=1.5*, *minerror=-5.0*, *maxerror=-1*, *linspace=False*)

Function to study the error in the non-linearity correction on the knowledge of the PSF ellipticity and size.

The error has been assumed to follow a sinusoidal curve with random phase and a given number of angular frequencies (defined by the multiplier). The amplitudes being studied, i.e. the size of the maximum deviation, can be spaced either linearly or logarithmically.

Parameters

- **log** (*instance*) – logger instance
- **file** (*str*) – name of the PSF FITS files to use [default=data/psf12x.fits]
- **oversample** – the PSF oversampling factor, which needs to match the input file [default=12]

- **sigma** (*float*) – 1sigma radius of the Gaussian weighting function for shape measurements
- **phs** (*float*) – phase in case phases = None
- **phases** (*None or int*) – if None then a single fixed phase will be applied, if an int then a given number of random phases will be used
- **psfs** (*int*) – the number of PSFs to use.
- **amps** (*int*) – the number of individual samplings used when covering the error space
- **multiplier** (*int or float*) – the number of angular frequencies to be used
- **minerror** (*float*) – the minimum error to be covered, given in log10(min_error) [default=-5 i.e. 0.001%]
- **maxerror** (*float*) – the maximum error to be covered, given in log10(max_error) [default=-1 i.e. 10%]
- **linspace** (*boolean*) – whether the amplitudes of the error curves should be linearly or logarithmically spaced.

Returns reference value and results dictionaries

Return type list

6.2.4 Testing the CTI Correction Algorithm

This script can be used to test the CTI correction algorithm performance.

requires NumPy

requires PyFITS

requires matplotlib

version 0.3

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

analysis.testCTIcorrection.**cutoutRegions**(*files*, *xcen=1900*, *ycen=1900*, *side=140*)

Parameters

- **files** –
- **xcen** –
- **ycen** –
- **side** –

Returns

analysis.testCTIcorrection.**plotResults**(*results*)

Plot the CTI correction algorithm results.

Parameters **results** – CTI test results

Returns None

analysis.testCTIcorrection.**plotResultsNoNoise**(*inputfile*, *title*, *bins=10*)
Plot the CTI correction algorithm results.

Returns None

analysis.testCTIcorrection.**testCTIcorrection**(*log*, *files*, *sigma=0.75*, *iterations=4*, *xcen=1900*, *ycen=1900*, *side=20*)
Calculates PSF properties such as ellipticity and size from data without CTI and from CTI data.

Parameters

- **log** (*instance*) – python logger instance
- **files** (*list*) – a list of files to be processed
- **sigma** (*float*) – size of the Gaussian weighting function
- **iterations** (*int*) – the number of iterations for the moment based shape estimator
- **xcen** (*int*) – x-coordinate of the object centre
- **ycen** (*int*) – y-coordinate of the object centre
- **side** (*int*) – size of the cutout around the centre (+/- side)

Returns ellipticity and size

Return type dict

analysis.testCTIcorrection.**testCTIcorrectionNonoise**(*log*, *files*, *output*, *sigma=0.75*, *iterations=4*)
Calculates PSF properties such as ellipticity and size from data w/ and w/o CTI.

Parameters

- **log** (*instance*) – python logger instance
- **files** (*list*) – a list of files to be processed
- **sigma** (*float*) – size of the Gaussian weighting function
- **iterations** (*int*) – the number of iterations for the moment based shape estimator

Returns ellipticity and size

Return type dict

analysis.testCTIcorrection.**useThibautsData**(*log*, *output*, *bcgr=72.2*, *sigma=0.75*, *iterations=4*, *loc=1900*, *galaxies=1000*, *datadir='/Users/smn2/EUCLID/CTItesting/uniform/'*, *thibautCDM03=False*, *beta=False*, *serial=1*, *parallel=1*)
Test the impact of CTI in case of no noise and no correction.

Parameters

- **log** – logger instance
- **bcgr** – background in electrons for the CTI modelling
- **sigma** – size of the weighting function for the quadrupole moment
- **iterations** – number of iterations in the quadrupole moments estimation
- **loc** – location to which the galaxy will be placed [default=1900]
- **galaxies** – number of galaxies to use (< 10000)
- **datadir** – directory pointing to the galaxy images

Returns

6.2.5 Cosmic Ray Rejection

This simple script can be used to study the error in the cosmic ray rejection that can be tolerated given the requirements.

The following requirements related to the cosmic rays have been taken from CalCD-B.

Note that the analysis is for a single star. Thus, if we consider a single exposure we can relax the requirements given that there will be on average about 1850 stars in each field that are usable for PSF modelling. Furthermore it shuold be noted that the presence of cosmic rays will always increase the size. Because of this one cannot combine the contribution from cosmic rays with other effects by adding each individual contribution in quadrature. It is more appropriate to add the impact of cosmic rays to the size of the PSF linearly given the preferred direction.

requires PyFITS

requires NumPy

requires SciPy

requires matplotlib

requires VISsim-Python

version 0.4

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

`analysis.cosmicrayCalibration.`**`plotResults`**(*results*, *reqe=3e-05*, *reqr2=0.0001*, *outdir='results'*, *timeStamp=False*)

Creates a simple plot to combine and show the results.

Parameters

- **res** (*list*) – results to be plotted [reference values, results dictionary]
- **reqe** (*float*) – the requirement for ellipticity [default=3e-5]
- **reqr2** (*float*) – the requirement for size R2 [default=1e-4]
- **outdir** (*str*) – output directory to which the plots will be saved to

Returns None

analysis.cosmicrayCalibration.**testCosmicrayRejection**(*log*, *file='data/psf1x.fits'*, *oversample=1.0*, *sigma=0.75*, *psfs=20000*, *scale=1000.0*, *min=1e-05*, *max=50*, *levels=15*, *covering=1.4*, *single=False*)

This is for a single PSF.

Parameters

- **log** –
- **file** –
- **oversample** –
- **sigma** –
- **psfs** –
- **scale** –
- **min** –
- **max** –
- **levels** –
- **covering** –
- **single** –

Returns

analysis.cosmicrayCalibration.**testCosmicrayRejectionMultiPSF**(*log*, *file='data/psf1x.fits'*, *oversample=1.0*, *sigma=0.75*, *psfs=2000*, *scale=1000.0*, *min=0.0001*, *max=100.0*, *levels=9*, *covering=2.0*, *stars=1850*, *single=False*)

Parameters

- **log** –
- **file** –
- **oversample** –
- **sigma** –
- **psfs** –
- **scale** –
- **min** –
- **max** –
- **levels** –
- **covering** –
- **single** –

Returns

6.2.6 Impact of Ghost Images

This scripts can be used to study the impact of ghost images on the weak lensing measurements.

requires PyFITS

requires NumPy (1.7.0 or newer for numpy.pad)

requires SciPy

requires matplotlib

requires VISsim-Python

version 0.2

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

`analysis.analyseGhosts.`**`analyseInFocusImpact`**(*log*, *filename='data/psf4x.fits'*, *psfscale=100000*, *maxdistance=100*, *oversample=4.0*, *psfs=1000*, *iterations=6*, *sigma=0.75*)

Calculates PSF size and ellipticity when including another PSF scaled to a given level (requirement = 5e-5)

Parameters

- **log** –
- **filename** – name of the PSF file to analyse
- **psfscale** – level to which the original PSF is scaled to
- **maxdistance** – maximum distance the ghost image can be from the original PSF (centre to centre)
- **oversample** – oversampling factor

- **psfs** – number of PSFs to analyse (number of ghosts in random locations)
- **iterations** – number of iterations in the shape measurement
- **sigma** – size of the Gaussian weighting function

Returns results

Return type dict

analysis.analyseGhosts.**analyseOutofFocusImpact**(*log*, *filename='data/psf4x.fits'*, *psfscale=100000*, *maxdistance=100*, *inner=8*, *outer=60*, *oversample=4.0*, *psfs=5000*, *iterations=5*, *sigma=0.75*, *lowghost=-7*, *highghost=-2*, *samples=9*)

Calculates PSF size and ellipticity when including an out-of-focus doughnut of a given contrast level. The dougnut pixel values are all scaled to a given scaling value (requirement 5e-5).

Parameters

- **log** – logger instance
- **filename** – name of the PSF file to analyse
- **psfscale** – level to which the original PSF is scaled to
- **maxdistance** – maximum distance the ghost image can be from the original PSF (centre to centre)
- **inner** – inner radius of the out-of-focus doughnut
- **outer** – outer radius of the out-of-focus doughnut
- **oversample** – oversampling factor
- **psfs** – number of PSFs to analyse (number of ghosts in random locations)
- **iterations** – number of iterations in the shape measurement
- **sigma** – size of the Gaussian weighting function
- **lowghost** – log of the highest ghost contrast ratio to study
- **highghost** – log of the lowest ghost contrast ratio to study
- **samples** – number of points for the contrast ratio to study

Returns results

Return type dict

analysis.analyseGhosts.**deleteAndMove**(*dir*, *files='*.fits'*)

Parameters

- **dir** –
- **files** –

Returns

analysis.analyseGhosts.**drawDoughnut**(*inner*, *outer*, *oversample=1*, *plot=False*)
Draws a doughnut shape with a given inner and outer radius.

Parameters

- **inner** (*float*) – inner radius in pixels
- **outer** (*float*) – outer radius in pixels
- **oversample** (*int*) – oversampling factor [default=1]
- **plot** (*bool*) – whether to generate a plot showing the doughnut or not

Returns image, xcentre, ycentre

Return type list

analysis.analyseGhosts.**objectDetection**(*log*, *magnitude=24.5*, *exptime=565*, *exposures=3*, *fpeak=0.7*, *offset=0.5*, *covering=2550*, *ghostlevels=(6e-07*, *1e-06*, *4e-06*, *5e-06*, *1e-05*, *5e-05)*)
Derive area loss in case of object detection i.e. the SNR drops below the requirement. Assumes that a ghost covers the covering number of pixels and that the peak pixel of a point source contains fpeak fraction of the total counts. An offset between the V-band and VIS band is applied as VIS = V + offset.

Parameters

- **log** – logger instance
- **magnitude** – magnitude limit of the objects to be detected
- **exptime** – exposure time of an individual exposure in seconds
- **exposures** – number of exposures combined in the data analysis
- **fpeak** – PSF peak fraction
- **offset** – offset between the V-band and the VIS band, VIS = V + offset
- **covering** – covering fraction of a single ghost in pixels
- **ghostlevels** – a tuple of ghost levels to inspect
- **ghostlevels** – tuple

Returns None

analysis.analyseGhosts.**plotGhostContribution**(*res*, *title*, *output*, *title2*, *output2*)

analysis.analyseGhosts.**plotGhostContributionElectrons**(*log*, *res*, *title*, *output*, *title2*, *output2*, *req=0.001*)

analysis.analyseGhosts.**shapeMeasurement**(*log*)
Shape measurement bias as a result of ghosts.

CHAPTER 7

Data Analysis

The *analysis* subpackage contains classes and scripts related to data analysis. A simple source finder and shape measuring classes are provided together with a wrapper to analyse reduced VIS data. For more detailed documentation of the classes, please see:

7.1 VIS data analysis tools

7.1.1 Object finding and measuring ellipticity

This script provides a class that can be used to analyse VIS data. One can either choose to use a Python based source finding algorithm or give a SExtractor catalog as an input. If an input catalog is provided then the program assumes that X_IMAGE and Y_IMAGE columns are present in the input file.

requires PyFITS

requires NumPy

requires matplotlib

author Sami-Matias Niemi

contact smn2@mssl.ucl.ac.uk

version 0.2

class `analysis.analyse.`**`analyseVISdata`**(*filename*, *log*, ***kwargs*)

Simple class that can be used to find objects and measure their ellipticities.

One can either choose to use a Python based source finding algorithm or give a SExtractor catalog as an input. If an input catalog is provided then the program assumes that X_IMAGE and Y_IMAGE columns are present in the input file.

Parameters

- **filename** (*string*) – name of the FITS file to be analysed.
- **log** (*instance*) – logger
- **kwargs** (*dict*) – additional keyword arguments

Settings dictionary contains all parameter values needed for source finding and analysis.

doAll()
Run all class methods sequentially.

findSources()
Finds sources from data that has been read in when the class was initiated. Saves results such as x and y coordinates of the objects to self.sources. x and y coordinates are also available directly in self.x and self.y.

measureEllipticity()
Measures ellipticity for all objects with coordinates (self.x, self.y).

Ellipticity is measured using Guassian weighted quadrupole moments. See shape.py and especially the ShapeMeasurement class for more details.

plotEllipticityDistribution()
Creates a simple plot showing the derived ellipticity distribution.

readSources()
Reads in a list of sources from an external file. This method assumes that the input source file is in SExtractor format. Input catalog is saves to self.sources. x and y coordinates are also available directly in self.x and self.y.

writeResults()
Outputs results to an ascii file defined in self.settings. This ascii file is in SExtractor format and contains the following columns:

```
1. X coordinate
2. Y coordinate
3. ellipticity
4. R_{2}
```

7.1.2 Measuring a shape of an object

Simple class to measure quadrupole moments and ellipticity of an object.

Note: Double check that the e1 component is not flipped in sense that Qxx and Qyy would be reversed because NumPy arrays are column major.

---

requires NumPy

requres PyFITS

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

version 0.45

class analysis.shape.**shapeMeasurement**(*data*, *log*, ***kwargs*)
Provides methods to measure the shape of an object.

Parameters

- **data** (*ndarray*) – name of the FITS file to be analysed.
- **log** (*instance*) – logger
- **kwargs** (*dict*) – additional keyword arguments

Settings dictionary contains all parameter values needed.

Gaussian2D(*x*, *y*, *sigmax*, *sigmay*)
Create a two-dimensional Gaussian centered on x, y.

Parameters

- **x** (*float*) – x coordinate of the centre
- **y** (*float*) – y coordinate of the centre
- **sigmax** (*float*) – standard deviation of the Gaussian in x-direction
- **sigmay** (*float*) – standard deviation of the Gaussian in y-direction

Returns circular Gaussian 2D profile and x and y mesh grid

Return type dict

circular2DGaussian(*x*, *y*, *sigma*)
Create a circular symmetric Gaussian centered on x, y.

Parameters

- **x** (*float*) – x coordinate of the centre
- **y** (*float*) – y coordinate of the centre
- **sigma** (*float*) – standard deviation of the Gaussian, note that sigma_x = sigma_y = sigma

Returns circular Gaussian 2D profile and x and y mesh grid

Return type dict

measureRefinedEllipticity()
Derive a refined iterated polarisability/ellipticity measurement for a given object.

By default polarisability/ellipticity is defined in terms of the Gaussian weighted quadrupole moments. If self.settings['weighted'] is False then no weighting scheme is used.

The number of iterations is defined in self.settings['iterations'].

:return centroids [indexing stars from 1], ellipticity (including projected e1 and e2), and R2
:rtype: dict

quadrupoles(*img*)
Derive quadrupole moments and ellipticity from the input image.

Parameters **img** (*ndarray*) – input image data

Returns quadrupoles, centroid, and ellipticity (also the projected components e1, e2)

Return type dict

writeFITS(*data*, *output*)
Write out a FITS file using PyFITS.

Parameters

- **data** (*ndarray*) – data to write to a FITS file
- **output** (*string*) – name of the output file

Returns None

7.1.3 Source finding

Simple source finder that can be used to find objects from astronomical images.

reqiures NumPy

requires SciPy

requires matplotlib

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

version 0.6

class `analysis.sourceFinder.`**`sourceFinder`**(*image*, *log*, ***kwargs*)

This class provides methods for source finding.

Parameters

- **image** (*numpy.ndarray*) – 2D image array
- **log** (*instance*) – logger
- **kwargs** (*dictionary*) – additional keyword arguments

`cleanSample`()

Cleans up small connected components and large structures.

`doAperturePhotometry`(*pixel_based=True*)

Perform aperture photometry and calculate the shape of the object based on quadrupole moments. This method also calculates refined centroid for each object.

> **Warning:** Results are rather sensitive to the background subtraction, while the errors depend strongly on the noise estimate from the background. Thus, great care should be exercised when applying this method.

Parameters **pixel_based** (*boolean*) – whether to do a global or pixel based background subtraction

Returns photometry, error_in_photometry, ellipticity, refined_x_pos, refined_y_pos

Returns ndarray, ndarray, ndarray, ndarray, ndarray

`find`()

Find all pixels above the median pixel after smoothing with a Gaussian filter.

Note: maybe one should use mode instead of median?

> **Warning:** The background estimation is very crude and should not be trusted. One should probably write an iterative scheme were the background is recalculated after identifying the objects.

generateOutput()
Outputs the found positions to an ascii and a DS9 reg file.

Returns None

getCenterOfMass()
Finds the center-of-mass for all objects using numpy.ndimage.center_of_mass method.

Note: these positions are zero indexed!

Returns xposition, yposition, center-of-masses

Return type list

getContours()
Derive contours using the diskStructure function.

getFluxes()
Derive fluxes or counts.

getSizes()
Derives sizes for each object.

plot()
Generates a diagnostic plot.

Returns None

runAll()
Performs all steps of source finding at one go.

Returns source finding results such as positions, sizes, fluxes, etc.

Return type dictionary

writePhotometry()
Outputs the photometric results to an ascii file.

Returns None

class analysis.sourceFinder.**sourceFinder**(*image*, *log*, ***kwargs*)
This class provides methods for source finding.

Parameters

- **image** (*numpy.ndarray*) – 2D image array
- **log** (*instance*) – logger
- **kwargs** (*dictionary*) – additional keyword arguments

cleanSample()
Cleans up small connected components and large structures.

doAperturePhotometry(*pixel_based=True*)
Perform aperture photometry and calculate the shape of the object based on quadrupole moments. This method also calculates refined centroid for each object.

> **Warning:** Results are rather sensitive to the background subtraction, while the errors depend strongly on the noise estimate from the background. Thus, great care should be exercised when applying this method.

Parameters **pixel_based** (*boolean*) – whether to do a global or pixel based background subtraction

Returns photometry, error_in_photometry, ellipticity, refined_x_pos, refined_y_pos

Returns ndarray, ndarray, ndarray, ndarray, ndarray

find()
Find all pixels above the median pixel after smoothing with a Gaussian filter.

Note: maybe one should use mode instead of median?

> **Warning:** The background estimation is very crude and should not be trusted. One should probably write an iterative scheme were the background is recalculated after identifying the objects.

generateOutput()
Outputs the found positions to an ascii and a DS9 reg file.

Returns None

getCenterOfMass()
Finds the center-of-mass for all objects using numpy.ndimage.center_of_mass method.

Note: these positions are zero indexed!

Returns xposition, yposition, center-of-masses

Return type list

getContours()
Derive contours using the diskStructure function.

getFluxes()
Derive fluxes or counts.

getSizes()
Derives sizes for each object.

plot()
Generates a diagnostic plot.

Returns None

runAll()
Performs all steps of source finding at one go.

Returns source finding results such as positions, sizes, fluxes, etc.

Return type dictionary

writePhotometry()
Outputs the photometric results to an ascii file.

Returns None

7.1.4 CCD Spot Measurements

This scripts can be used to study CCD spot measurements.

requires PyFITS

requires NumPy

requires SciPy

requires matplotlib

requires VISsim-Python

version 0.1

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

analysis.analyseSpotMeasurements.**analyseSpotsDeconvolution**(*files*)
Analyse spot measurements using deconvolutions.

Note: does not really work... perhaps an issue with sizes.

Parameters **files** (*list*) – a list of input files

Returns None

analysis.analyseSpotMeasurements.**analyseSpotsFitting**(*files*, *gaussian=False*, *pixelvalues=False*, *bessel=True*, *maxfev=10000*)
Analyse spot measurements using different fitting methods.

Parameters

- **files** – names of the FITS files to analyse (should match the IDs)
- **gaussian** – whether or not to do a simple Gaussian fitting analysis
- **pixelvalues** – whether or not to plot pixel values on a grid
- **bessel** – whether or not to do a Bessel + Gaussian convolution analysis
- **maxfev** – maximum number of iterations in the least squares fitting

Returns None

analysis.analyseSpotMeasurements.**examples**()
Simple convolution-deconvolution examples.

Returns None

analysis.analyseSpotMeasurements.**fitf**(*height*, *center_x*, *center_y*, *width_x*, *width_y*)
Fitting function: 2D gaussian

analysis.analyseSpotMeasurements.**gaussianFit**(*ydata*, *xdata=None*, *initials=None*)

Fits a single Gaussian to a given data. Uses scipy.optimize.leastsq for fitting.

Parameters

- **ydata** – to which a Gaussian will be fitted to.
- **xdata** – if not given uses np.arange
- **initials** – initial guess for Gaussian parameters in order [amplitude, mean, sigma, floor]

Returns coefficients, best fit params, success

Return type dictionary

analysis.analyseSpotMeasurements.**generateBessel**(*radius=1.5*, *oversample=500*, *size=1000*, *cx=None*, *cy=None*, *debug=False*)

Generates a 2D Bessel function by taking a Fourier transform of a disk with a given radius. The real image and the subsequent power spectrum is oversampled with a given factor. The peak value of the generated Bessel function is normalized to unity.

Parameters

- **radius** – radius of the disc [default=1.5]
- **oversample** – oversampling factor [default=500]
- **size** – size of the output array
- **cx** – centre of the disc in x direction
- **cy** – centre of the disc in y direction
- **debug** – whether or not to generate FITS files

Returns

analysis.analyseSpotMeasurements.**plotGaussianResults**(*data, ids, output, vals=[0, 2]*)

Parameters

- **res** – results
- **ids** – file identifiers

Returns None

analysis.analyseSpotMeasurements.**readData**(*filename*, *crop=True*)

Parameters

- **filename** –
- **crop** –

Returns

analysis.analyseSpotMeasurements.**weinerFilter**(*data*, *kernel*, *regularization=0.01*, *normalize=True*)

Performs Wiener deconvolution on data using a given kernel. The input can be either 1D or 2D array.

For further information: http://en.wikipedia.org/wiki/Wiener_deconvolution

Parameters

- **data** – data to be deconvolved (either 1D or 2D array)
- **kernel** – kernel that is used for the deconvolution
- **regularization** – regularization parameter
- **normalize** – whether or not the peak value should be normalized to unity

Returns deconvolved data

7.1.5 Properties of the Point Spread Function

This script can be used to plot some PSF properties such as ellipticity and size as a function of the focal plane position.

requires PyFITS

requires NumPy

requires SciPy

requires matplotlib

requires VISsim-Python

author Sami-Matias Niemi

contact smn2@mssl.ucl.ac.uk

analysis.PSFproperties.**encircledEnergy**(*file='data/psf12x.fits'*)

Calculates the encircled energy from a PSF. The default input PSF is 12 times over-sampled with 1 micron pixel.

analysis.PSFproperties.**generatePlots**(*filedata*, *interactive=False*)

Generate a simple plot showing some results.

analysis.PSFproperties.**measureChars**(*data*, *info*, *log*)

Measure ellipticity, R2, FWHM etc.

analysis.PSFproperties.**parseName**(*file*)

Parse information from the input file name.

Example name:

```
detector_jitter-1_TOL05_MC_T0074_50arcmin2_grid_Nim=16384x16384_pixsize=1.000um_lbd
```

analysis.PSFproperties.**peakFraction**(*file='data/psf12x.fits'*, *radius=0.65*, *oversample=12*)

Calculates the fraction of energy in the peak pixel for a given PSF compared to an aperture of a given radius.

analysis.PSFproperties.**plotEncircledEnergy**(*radius*, *energy*, *scale=12*)

analysis.PSFproperties.**readData**(*file*)
Reads in the data from a given FITS file.

7.1.6 Generating PSF Basis Sets

This script can be used to derive a basis set for point spread functions.

requires Scikit-learn

requires PyFITS

requires NumPy

requires SciPy

requires matplotlib

requires VISsim-Python

version 0.3

author Sami-Matias Niemi

contact smn2@mssl.ucl.ac.uk

analysis.PSFbasisSets.**deriveBasisSetsICA**(*data*, *cut*, *outfolder*, *components=10*)
Derives a basis set from input data using Perform Fast Independent Component Analysis. Saves the basis sets to a FITS file for further processing.

http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.fastica.html#sklearn.decomposition.fastica

analysis.PSFbasisSets.**deriveBasisSetsPCA**(*data*, *cut*, *outfolder*, *components=10*, *whiten=False*)
Derives a basis set from input data using Principal component analysis (PCA). Saves the basis sets to a FITS file for further processing.

Information about PCA can be found from the scikit-learn website: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA

Parameters

- **data** (*ndarray*) – input data from which the basis set are derived from. The input data must be an array of arrays. Each array should describe an independent data set that has been flatted to 1D.
- **cut** (*int*) – size of the cutout region that has been used
- **outfolder** (*str*) – name of the output folder e.g. 'output'
- **components** (*int*) – the number of basis set function components to derive
- **whiten** (*bool*) – When True (False by default) the **components_** vectors are divided by n_samples times singular values to ensure uncorrelated outputs with unit component-wise variances.

Returns PCA components

analysis.PSFbasisSets.**deriveBasisSetsRandomizedPCA**(*data*, *cut*, *outfolder*, *components=10*, *whiten=False*)

Derives a basis set from input data using Randomized Principal component analysis (PCA). Saves the basis sets to a FITS file for further processing.

Information about PCA can be found from the scikit-learn website: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.RandomizedPCA.html#sklearn.decomposition.Randon

Parameters

- **data** (*ndarray*) – input data from which the basis set are derived from. The input data must be an array of arrays. Each array should describe an independent data set that has been flatted to 1D.
- **cut** (*int*) – size of the cutout region that has been used
- **outfolder** (*str*) – name of the output folder e.g. 'output'
- **components** (*int*) – the number of basis set function components to derive
- **whiten** (*bool*) – When True (False by default) the **components_** vectors are divided by n_samples times singular values to ensure uncorrelated outputs with unit component-wise variances.

Returns Randomized PCA components

analysis.PSFbasisSets.**generateMovie**(*files*, *output*, *outputfolder*, *title*)

Generates a movie from the basis set files.

Returns None

analysis.PSFbasisSets.**processArgs**(*printHelp=False*)

Processes command line arguments.

analysis.PSFbasisSets.**visualiseBasisSets**(*files*, *output*, *outputfolder*, *big3d=False*)

Generate visualisation of the basis sets.

Parameters **files** – a list of file names that should be visualised

Returns None

7.1.7 PSF Fitting

This script can be used to fit a set of basis functions to a point spread function.

requires Scikit-learn

requires PyFITS

requires NumPy

requires SciPy

requires matplotlib

requires VISsim-Python

version 0.2

author Sami-Matias Niemi

contact smn2@mssl.ucl.ac.uk

analysis.fitPSF.**BaysianPCAfitting**()
Perform PCA basis set fitting using Bayesian Markov Chain Monte Carlo fitting technique.

Requires pyMC

Returns None

analysis.fitPSF.**ICAleastSq**()
Perform least squares fitting using ICA basis set.

Returns None

analysis.fitPSF.**PCAleastSQ**()
Perform least squares fitting using PCA basis set.

Returns None

analysis.fitPSF.**leastSQfit**(*x*, *a0*, *a1*, *a2*, *a3*, *a4*, *a5*, *a6*, *a7*, *a8*, *a9*, *a10*, *a11*, *a12*, *a13*, *a14*, *a15*, *a16*, *a17*, *a18*, *a19*, *a20*)
A simple function returning a linear combination of 20 input parameters.

Parameters

- **x** (*mdarray*) – data
- **a0** – first fitting coefficient
- **a0** – float
- **a1** – second fitting coefficient
- **a1** – float
- **a2** – third fitting coefficient
- **a2** – float
- **a3** – fourth fitting coefficient
- **a3** – float
- **a4** – fifth fitting coefficient
- **a4** – float
- **a5** – sixth fitting coefficient
- **a5** – float
- **a6** – seventh fitting coefficient
- **a6** – float
- **a7** – eight fitting coefficient
- **a7** – float
- **a8** – ninth fitting coefficient
- **a8** – float
- **a9** – tenth fitting coefficient
- **a9** – float

- **a10** – eleventh fitting coefficient
- **a10** – float
- **a11** – twelfth fitting coefficient
- **a0** – float
- **a12** – thirteenth fitting coefficient
- **a0** – float
- **a13** – fourteenth fitting coefficient
- **a0** – float
- **a14** – fifteenth fitting coefficient
- **a0** – float
- **a15** – sixteenth fitting coefficient
- **a0** – float
- **a16** – seventieth fitting coefficient
- **a0** – float
- **a17** – eighteenth fitting coefficient
- **a0** – float
- **a18** – nineteenth fitting coefficient
- **a0** – float
- **a19** – twentieth fitting coefficient
- **a0** – float
- **a20** – twentyfirst fitting coefficient
- **a0** – float

Returns a linear combination of the data given the coefficients

Return type ndarray

`analysis.fitPSF.`**`test`**`()`

Simple test using both least squares and Bayesian MCMC fitting. Includes Poisson noise to the PSF prior fitting. The PSF is build randomly from the the mean PSF and the PCA components are being fitted.

Returns None

`analysis.fitPSF.`**`visualise`**`(`*popt*, *psf*, *modedata*, *output='PCA'*`)`

Plot the fitting results, residuals and coefficients

Parameters

- **popt** (*ndarray*) – fitting coefficients
- **psf** (*ndarray*) – data to which the modes were fitted to
- **modedata** (*ndarray*) – each of the basis set modes that were fit to the data

Returns None

7.1.8 Impact of CTI Trailing on Shear Power Spectrum

A simple script to study the effects of CTI trailing to weak lensing shear power spectrum.

requires NumPy

requires SciPy

requires matplotlib

version 0.1

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

analysis.CTIpower.**azimuthalAverage**(*image*, *center=None*, *stddev=False*, *returnradii=False*, *return_nr=False*, *binsize=0.5*, *weights=None*, *steps=False*, *interpnan=False*, *left=None*, *right=None*)

Calculates the azimuthally averaged radial profile.

Parameters

- **image** (*ndarray*) – image
- **center** (*list or None*) – The [x,y] pixel coordinates used as the centre. The default is None, which then uses the center of the image (including fractional pixels).
- **stddev** – if specified, return the azimuthal standard deviation instead of the average
- **returnradii** – if specified, return (radii_array, radial_profile)
- **return_nr** – if specified, return number of pixels per radius *and* radius
- **binsize** – size of the averaging bin. Can lead to strange results if non-binsize factors are used to specify the center and the binsize is too large.
- **weights** – can do a weighted average instead of a simple average if this keyword parameter is set. weights.shape must = image.shape. weighted stddev is undefined, so don't set weights and stddev.
- **steps** – if specified, will return a double-length bin array and radial profile
- **interpnan** – Interpolate over NAN values, i.e. bins where there is no data
- **left** – passed to interpnan to set the extrapolated values
- **right** – passed to interpnan to set the extrapolated values

analysis.CTIpower.**azimuthalAverageSimple**(*image*, *center=None*)

A simple algorithm to calculate the azimuthally averaged radial profile.

Parameters

- **image** (*ndarray*) – image
- **center** (*list or None*) – The [x,y] pixel coordinates used as the centre. The default is None, which then uses the center of the image (including fractional pixels).

Returns radial profile

analysis.CTIpower.**plot**(*data*, *fourier*, *radius*, *profile*, *xi*, *yi*, *output*, *title*, *log=False*)

Parameters

- **data** –
- **fourier** –
- **profile** –
- **output** –
- **title** –
- **log** –

Returns

analysis.CTIpower.**plotCTIeffect**(*data*)

This function plots the CTI impact on shear. The Data is assumed to be in the following format:

```
x [deg]     y [deg] g1_parallel   g1_serial    g1_total    g2_total
```

Parameters **data** –

Returns

analysis.CTIpower.**plotPower**(*data*)

Parameters **data** –

Returns

7.1.9 Non-linearity II: Model Transfer

Most PSF stars are bright while most weak lensing galaxies are faint, thus the PSF model needs to be transferable to an appropriate object luminosity (and colour, position, etc.). This simple script can be used to study the error in the model transfer due to non-linearity that can be tolerated given the requirements.

The following requirements related to the model transfer have been taken from CalCD-B.

requires PyFITS

requires NumPy

requires SciPy

requires matplotlib

requires VISsim-Python

version 0.2

author Sami-Matias Niemi

contact s.niemi@ucl.ac.uk

analysis.nonlinearityModelTransfer.**plotResults**(*results*, *reqe=3.5e-05*, *reqr2=0.00035*, *outdir='results'*, *timeStamp=False*)

Creates a simple plot to combine and show the results.

Parameters

- **res** (*list*) – results to be plotted [reference values, results dictionary]
- **reqe** (*float*) – the requirement for ellipticity [default=3.5e-5]
- **reqr2** (*float*) – the requirement for size R2 [default=3.5e-4]
- **outdir** (*str*) – output directory to which the plots will be saved to

Returns None

analysis.nonlinearityModelTransfer.**testNonlinearityModelTransfer**(*log*, *file='data/psf12x.fits'*, *oversample=12.0*, *sigma=0.75*, *psfs=5000*, *amps=12*, *multiplier=1.5*, *minerror=-5.0*, *maxerror=-1.0*, *lowflux=500*, *highflux=180000*, *linspace=False*)

Function to study the error in the non-linearity correction on the knowledge of the PSF ellipticity and size.

The error has been assumed to follow a sinusoidal curve with random phase and a given number of angular frequencies (defined by the multiplier). The amplitudes being studied, i.e. the size of the maximum deviation, can be spaced either linearly or logarithmically.

Parameters

- **log** (*instance*) – logger instance
- **file** (*str*) – name of the PSF FITS files to use [default=data/psf12x.fits]
- **oversample** – the PSF oversampling factor, which needs to match the input file [default=12]
- **sigma** (*float*) – 1sigma radius of the Gaussian weighting function for shape measurements
- **phs** (*float*) – phase in case phases = None

- **phases** (*None or int*) – if None then a single fixed phase will be applied, if an int then a given number of random phases will be used
- **psfs** (*int*) – the number of PSFs to use.
- **amps** (*int*) – the number of individual samplings used when covering the error space
- **multiplier** (*int or float*) – the number of angular frequencies to be used
- **minerror** (*float*) – the minimum error to be covered, given in log10(min_error) [default=-5 i.e. 0.001%]
- **maxerror** (*float*) – the maximum error to be covered, given in log10(max_error) [default=-1 i.e. 10%]
- **linspace** (*boolean*) – whether the amplitudes of the error curves should be linearly or logarithmically spaced.

Returns reference value and results dictionaries

Return type list

The *data* subfolder contains the supporting data, such as cosmic ray distributions, cosmetics maps, flat fielding files, PSFs, and an example configuration file.

CHAPTER 8

Charge Transfer Inefficiency

The *fitting* subpackage contains a simple script that can be used to fit trap species so that the Charge Transfer Inefficiency (CTI) trails forming behind charge injection lines agree with measured data.

8.1 Fortran code for CTI

The *fortran* folder contains a CDM03 CTI model Fortran code. For speed the CDM03 model has been written in Fortran because it contains several nested loops. One can use f2py to compile the code to a format that can be imported directly to Python.

CHAPTER 9

Supporting methods and files

9.1 Objects

A few postage stamps showing observed galaxies have been placed to the *objects* directory. These FITS files can be used for, e.g., testing the shape measurement code.

9.2 Code

The *support* subpackage contains some support classes and methods related to generating log files and read in data.

CHAPTER 10

Photometric Accuracy

The reference image simulator has been tested against photometric accuracy (without aperture correction). A simulated image was generated with the reference simulator (using three times over sampled PSF) after which the different quadrants were combiled to form a single CCD. These data were then reduced using the reduction script provided in the package. Finally, sources were identified from the reduced data and photometry performed using SExtractor, after which the extracted magnitudes were compared against the input catalog.

The following figures show that the photometric accuracy with realistic noise and the end-of-life radiation damage is about 0.08 mag. Please note, however, that the derived magnitudes are based on a single 565 second exposure, and therefore the scatter is large in case of fainter objects.

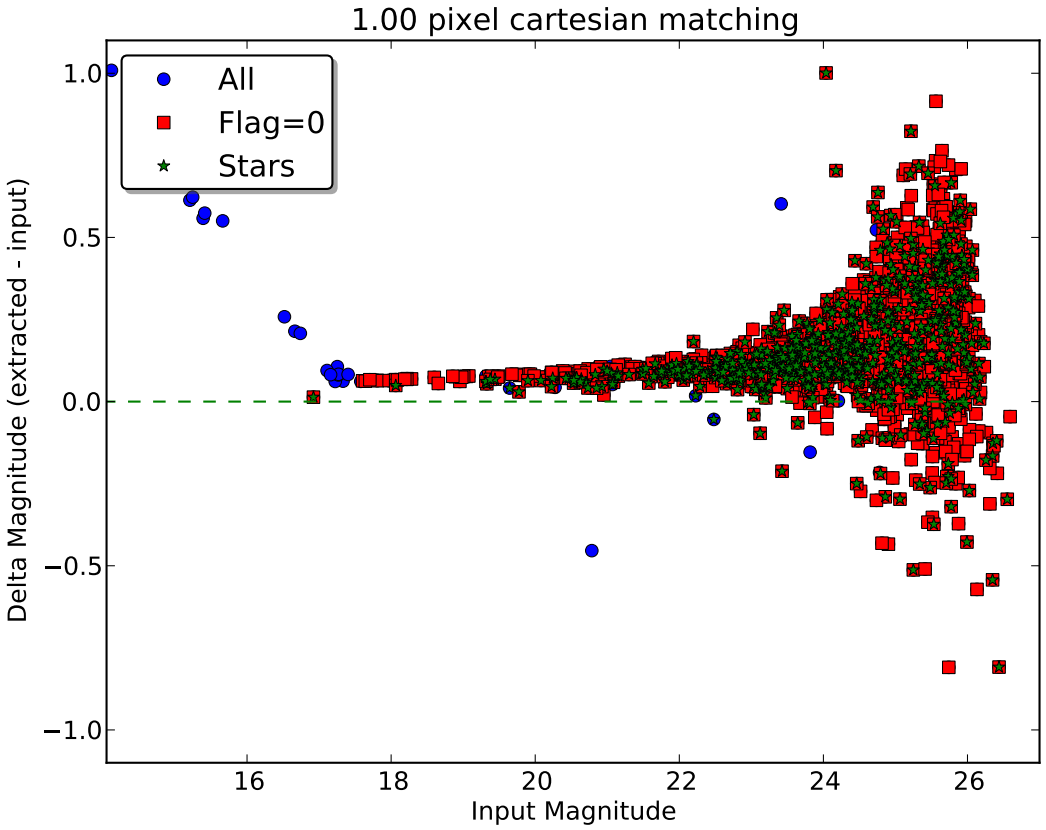

Figure 10.1: Example showing the recovered photometry from a reference simulator image with realistic noise, average background, and end-of-life radiation damage, but without aperture correction. The offset is about 0.08mag.

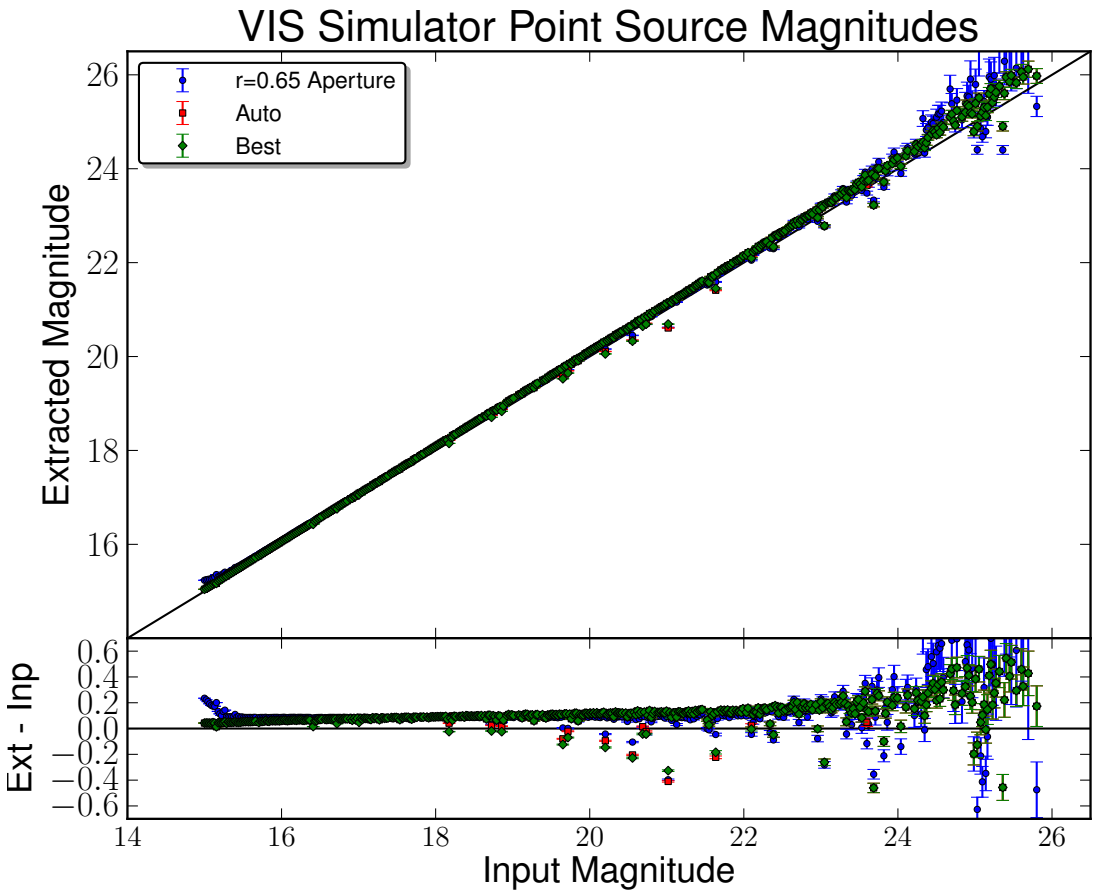

Figure 10.2: Example showing the recovered photometry for point sources from a reference simulator image with realistic noise and an average background, but without any radiation damage to the CCD.

CHAPTER 11

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

a

e

p

r

s

Index

A

B

C

D

E

F

Q

R

S

T

U

V

W