aotools Documentation

Release 1.0.6+0.geaa86dd.dirty

Andrew Reeves

Mar 16, 2022

Contents

1 Introduction **1**
- 1.1 Installation . . . 1

2 Contents **3**
- 2.1 Introduction . . . 3
- 2.2 Atmospheric Turbulence . . . 3
- 2.3 Image Processing . . . 13
- 2.4 Circular Functions . . . 15
- 2.5 Telescope Pupils . . . 20
- 2.6 Wavefront Sensors . . . 21
- 2.7 Optical Propagation . . . 22
- 2.8 Astronomical Functions . . . 23
- 2.9 Normalised Fourier Transforms . . . 24
- 2.10 Interpolation . . . 26

3 Indices and tables **27**

Python Module Index **29**

Index **31**

CHAPTER 1

Introduction

AOtools is an attempt to gather together many common tools, equations and functions for Adaptive Optics. The idea is to provide an alternative to the current model, where a new AO researcher must write their own library of tools, many of which implement theory and ideas that have been in existance for over 20 years. AOtools will hopefully provide a common place to put these tools, which through use and bug fixes, should become reliable and well documented.

1.1 Installation

AOtools uses mainly standard library functions, and all attempts have been made to avoid adding unneccessary dependencies. Currently, the library requires only

- numpy
- scipy
- astropy
- matplotlib

CHAPTER 2

Contents

2.1 Introduction

AOtools is an attempt to gather together many common tools, equations and functions for Adaptive Optics. The idea is to provide an alternative to the current model, where a new AO researcher must write their own library of tools, many of which implement theory and ideas that have been in existance for over 20 years. AOtools will hopefully provide a common place to put these tools, which through use and bug fixes, should become reliable and well documented.

2.1.1 Installation

AOtools uses mainly standard library functions, and all attempts have been made to avoid adding unneccessary dependencies. Currently, the library requires only

- numpy
- scipy
- astropy
- matplotlib

2.2 Atmospheric Turbulence

2.2.1 Finite Phase Screens

Creation of phase screens of a defined size with Von Karmen Statistics.

`aotools.turbulence.phasescreen.`**`ft_phase_screen`**(*r0*, *N*, *delta*, *L0*, *l0*, *FFT=None*, *seed=None*)

Creates a random phase screen with Von Karmen statistics. (Schmidt 2010)

Parameters

- **r0** (*float*) – r0 parameter of scrn in metres
- **N** (*int*) – Size of phase scrn in pxls
- **delta** (*float*) – size in Metres of each pxl
- **L0** (*float*) – Size of outer-scale in metres
- **l0** (*float*) – inner scale in metres

Note: The phase screen is returned as a 2d array, with each element representing the phase change in **radians**. This means that to obtain the physical phase distortion in nanometres, it must be multiplied by (wavelength / (2*pi)), (where *wavellength* here is the same wavelength in which r0 is given in the function arguments)

Returns numpy array representing phase screen in radians

Return type ndarray

aotools.turbulence.phasescreen.**ft_sh_phase_screen**(*r0*, *N*, *delta*, *L0*, *l0*, *FFT=None*, *seed=None*)

Creates a random phase screen with Von Karmen statistics with added sub-harmonics to augment tip-tilt modes. (Schmidt 2010)

Note: The phase screen is returned as a 2d array, with each element representing the phase change in **radians**. This means that to obtain the physical phase distortion in nanometres, it must be multiplied by (wavelength / (2*pi)), (where *wavellength* here is the same wavelength in which r0 is given in the function arguments)

Parameters

- **r0** (*float*) – r0 parameter of scrn in metres
- **N** (*int*) – Size of phase scrn in pxls
- **delta** (*float*) – size in Metres of each pxl
- **L0** (*float*) – Size of outer-scale in metres
- **l0** (*float*) – inner scale in metres

Returns numpy array representing phase screen in radians

Return type ndarray

2.2.2 Infinite Phase Screens

An implementation of the "infinite phase screen", as deduced by Francois Assemat and Richard W. Wilson, 2006.

class aotools.turbulence.infinitephasescreen.**PhaseScreenVonKarman**(*nx_size*, *pixel_scale*, *r0*, *L0*, *random_seed=None*, *n_columns=2*)

Bases: aotools.turbulence.infinitephasescreen.PhaseScreen

A "Phase Screen" for use in AO simulation with Von Karmon statistics.

This represents the phase addition light experiences when passing through atmospheric turbulence. Unlike other phase screen generation techniques that translate a large static screen, this method keeps a small section of phase,

and extends it as necessary for as many steps as required. This can significantly reduce memory consumption at the expense of more processing power required.

The technique is described in a paper by Assemat and Wilson, 2006. It essentially assumes that there are two matrices, "A" and "B", that can be used to extend an existing phase screen. A single row or column of new phase can be represented by

X = A.Z + B.b

where X is the new phase vector, Z is some number of columns of the existing screen, and b is a random vector with gaussian statistics.

This object calculates the A and B matrices using an expression of the phase covariance when it is initialised. Calculating A is straightforward through the relationship:

A = Cov_xz . (Cov_zz)^(-1).

B is less trivial.

BB^t = Cov_xx - A.Cov_zx

(where B^t is the transpose of B) is a symmetric matrix, hence B can be expressed as

B = UL,

where U and L are obtained from the svd for BB^t

U, w, U^t = svd(BB^t)

L is a diagonal matrix where the diagonal elements are w^(1/2).

On initialisation an initial phase screen is calculated using an FFT based method. When `add_row` is called, a new vector of phase is added to the phase screen using *nCols* columns of previous phase. Assemat & Wilson claim that two columns are adequate for good atmospheric statistics. The phase in the screen data is always accessed as `<phasescreen>.scrn` and is in radians.

The phase screen is returned on each iteration as a 2d array, with each element representing the phase change in **radians**. This means that to obtain the physical phase distortion in nanometres, it must be multiplied by (wavelength / (2*pi)), (where *wavellength* here is the same wavelength in which r0 is given in the function arguments)

Parameters

- **nx_size** (*int*) – Size of phase screen (NxN)
- **pixel_scale** (*float*) – Size of each phase pixel in metres
- **r0** (*float*) – fried parameter (metres)
- **L0** (*float*) – Outer scale (metres)
- **n_columns** (*int, optional*) – Number of columns to use to continue screen, default is 2

add_row()
Adds a new row to the phase screen and removes old ones.

scrn
The current phase map held in the PhaseScreen object in radians.

class aotools.turbulence.infinitephasescreen.**PhaseScreenKolmogorov**(*nx_size*, *pixel_scale*, *r0*, *L0*, *random_seed=None*, *stencil_length_factor=4*)

Bases: aotools.turbulence.infinitephasescreen.PhaseScreen

A "Phase Screen" for use in AO simulation using the Fried method for Kolmogorov turbulence.

This represents the phase addition light experiences when passing through atmospheric turbulence. Unlike other phase screen generation techniques that translate a large static screen, this method keeps a small section of phase, and extends it as neccessary for as many steps as required. This can significantly reduce memory consuption at the expense of more processing power required.

The technique is described in a paper by Assemat and Wilson, 2006 and expanded upon by Fried, 2008. It essentially assumes that there are two matrices, "A" and "B", that can be used to extend an existing phase screen. A single row or column of new phase can be represented by

X = A.Z + B.b

where X is the new phase vector, Z is some data from the existing screen, and b is a random vector with gaussian statistics.

This object calculates the A and B matrices using an expression of the phase covariance when it is initialised. Calculating A is straightforward through the relationship:

A = Cov_xz . (Cov_zz)^(-1).

B is less trivial.

BB^t = Cov_xx - A.Cov_zx

(where B^t is the transpose of B) is a symmetric matrix, hence B can be expressed as

B = UL,

where U and L are obtained from the svd for BB^t

U, w, U^t = svd(BB^t)

L is a diagonal matrix where the diagonal elements are w^(1/2).

The Z data is taken from points in a "stencil" defined by Fried that samples the entire screen. An additional "reference point" is also considered, that is picked from a point separate from teh stencil and applied on each iteration such that the new phase equation becomes:

On initialisation an initial phase screen is calculated using an FFT based method. When `add_row` is called, a new vector of phase is added to the phase screen. The phase in the screen data is always accessed as `<phasescreen>.scrn` and is in radians.

Note: The phase screen is returned on each iteration as a 2d array, with each element representing the phase change in **radians**. This means that to obtain the physical phase distortion in nanometres, it must be multiplied by (wavelength / (2*pi)), (where *wavellength* here is the same wavelength in which r0 is given in the function arguments)

Parameters

- **nx_size** (*int*) – Size of phase screen (NxN)
- **pixel_scale** (*float*) – Size of each phase pixel in metres

- **r0** (*float*) – fried parameter (metres)
- **L0** (*float*) – Outer scale (metres)
- **random_seed** (*int, optional*) – seed for the random number generator
- **stencil_length_factor** (*int, optional*) – How much longer is the stencil than the desired phase? default is 4

add_row()
Adds a new row to the phase screen and removes old ones.

scrn
The current phase map held in the PhaseScreen object in radians.

2.2.3 Slope Covariance Matrix Generation

Slope covariance matrix routines for AO systems observing through Von Karmon turbulence. Such matrices have a variety of uses, though they are especially useful for creating 'tomographic reconstructors' that can reconstruct some 'psuedo' WFS measurements in a required direction (where there might be an interesting science target but no guide stars), given some actual measurements in other directions (where the some suitable guide stars are).

Warning: This code has been tested qualitatively and seems OK, but needs more rigorous testing.

class aotools.turbulence.slopecovariance.**CovarianceMatrix**(*n_wfs*, *pupil_masks*, *telescope_diameter*, *subap_diameters*, *gs_altitudes*, *gs_positions*, *wfs_wavelengths*, *n_layers*, *layer_altitudes*, *layer_r0s*, *layer_L0s*, *threads=1*)

Bases: object

A creator of slope covariance matrices in Von Karmon turbulence, based on the paper by Martin et al, SPIE, 2012.

Given a list of paramters describing an AO WFS system and the atmosphere above the telescope, this class can compute the covariance matrix between all the WFS measurements. This can support LGS sources that exist at a finite altitude. When computing the covariance matrix, Python's multiprocessing module is used to spread the work between different processes and processing cores.

On initialisation, this class performs some initial calculations and parameter sorting. To create the covariance matrix run the make_covariace_matrix method. This may take some time depending on your paramters...

Parameters

- **n_wfs** (*int*) – Number of wavefront sensors present.
- **pupil_masks** (*ndarray*) – A map of the pupil for each WFS which is nx_subaps by ny_subaps. 1 if subap active, 0 if not.
- **telescope_diameter** (*float*) – Diameter of the telescope

- **subap_diameters** (*ndarray*) – The diameter of the sub-apertures for each WFS in metres
- **gs_altitudes** (*ndarray*) – Reciprocal (1/metres) of the Guide star alitude for each WFS
- **gs_positions** (*ndarray*) – X,Y position of each WFS in arcsecs. Array shape (Wfs, 2)
- **wfs_wavelengths** (*ndarray*) – Wavelength each WFS observes
- **n_layers** (*int*) – The number of atmospheric turbulence layers
- **layer_altitudes** (*ndarray*) – The altitude of each turbulence layer in meters
- **layer_r0s** (*ndarray*) – The Fried parameter of each turbulence layer
- **layer_L0s** (*ndarray*) – The outer-scale of each layer in metres
- **threads** (*int, optional*) – Number of processes to use for calculation. default is 1

make_covariance_matrix()
Calculate and build the covariance matrix

Returns Covariance Matrix

Return type ndarray

make_tomographic_reconstructor(*svd_conditioning=0*)
Creats a tomohraphic reconstructor from the covariance matrix as in Vidal, 2010. See the documentation for the function *create_tomographic_covariance_reconstructor* in this module. Assumes the 1st WFS given is the one for which reconstruction is required to.

Parameters **svd_conditioning** (*float*) – Conditioning for the SVD used in inversion.

Returns A tomohraphic reconstructor.

Return type ndarray

aotools.turbulence.slopecovariance.**calculate_structure_function**(*phase*, *nbOfPoint=None*, *step=None*)
Compute the structure function of an 2D array, along the first dimension. SF defined as sf[j]= < (phase - phase_shifted_by_j)^2 > Translated from a YAO function.

Parameters

- **phase** (*ndarray, 2d*) – 2d-array
- **nbOfPoint** (*int*) – final size of the structure function vector. Default is phase.shape[1] / 4
- **step** (*int*) – step in pixel when computing the sf. (step * sampling_phase) gives the sf sampling in meters. Default is 1

Returns values for the structure function of the data.

Return type ndarray, float

aotools.turbulence.slopecovariance.**calculate_wfs_seperations**(*n_subaps1*, *n_subaps2*, *wfs1_positions*, *wfs2_positions*)
Calculates the seperation between all sub-apertures in two WFSs

Parameters

- **n_subaps1** (*int*) – Number of sub-apertures in WFS 1
- **n_subaps2** (*int*) – Number of sub-apertures in WFS 2
- **wfs1_positions** (*ndarray*) – Array of the X, Y positions of the centre of each sub-aperture with respect to the centre of the telescope pupil
- **wfs2_positions** (*ndarray*) – Array of the X, Y positions of the centre of each sub-aperture with respect to the centre of the telescope pupil

Returns 2-D Array of sub-aperture seperations

Return type ndarray

aotools.turbulence.slopecovariance.**create_tomographic_covariance_reconstructor**(*covariance_matr*
n_onaxis_subaps
svd_conditioning

Calculates a tomographic reconstructor using the method of Vidal, JOSA A, 2010.

Uses a slope covariance matrix to generate a reconstruction matrix that will convert a collection of measurements from WFSs observing in a variety of directions (the "off-axis" directions) to the measurements that would have been observed by a WFS observing in a different direction (the "on-axis" direction. The given covariance matrix must include the covariance between all WFS measurements, including the "psuedo" WFS pointing in the on-axis direction. It is assumed that the covariance of measurements from this on-axis direction are first in the covariance matrix.

To create the tomographic reconstructor it is neccessary to invert the covariance matrix between off-axis WFS measurements. An SVD based psueod inversion is used for this (*numpy.linalg.pinv*). A conditioning to this SVD may be required to filter potentially unwanted modes.

Parameters

- **covariance_matrix** (*ndarray*) – A covariance matrix between WFSs
- **n_onaxis_subaps** (*int*) – Number of sub-aperture in on-axis WFS
- **svd_conditioning** (*float, optional*) – Conditioning for SVD inversion (default is 0)

Returns:

aotools.turbulence.slopecovariance.**mirror_covariance_matrix**(*cov_mat*)

Mirrors a covariance matrix around the axis of the diagonal.

Parameters

- **cov_mat** (*ndarray*) – The covariance matrix to mirror
- **n_subaps** (*ndarray*) – Number of sub-aperture in each WFS

aotools.turbulence.slopecovariance.**structure_function_kolmogorov**(*separation*, *r0*)

Compute the Kolmogorov phase structure function

Parameters

- **separation** (*ndarray, float*) – float or array of data representing separations between points
- **r0** (*float*) – Fried parameter for atmosphere

Returns Structure function for separation(s)

Return type ndarray, float

aotools.turbulence.slopecovariance.**structure_function_vk**(*seperation*, *r0*, *L0*)
Computes the Von Karmon structure function of atmospheric turbulence

Parameters

- **seperation** (*ndarray, float*) – float or array of data representing seperations between points
- **r0** (*float*) – Fried parameter for atmosphere
- **L0** (*float*) – Outer scale of turbulence

Returns Structure function for seperation(s)

Return type ndarray, float

aotools.turbulence.slopecovariance.**wfs_covariance**(*n_subaps1*, *n_subaps2*, *wfs1_positions*, *wfs2_positions*, *wfs1_diam*, *wfs2_diam*, *r0*, *L0*)
Calculates the covariance between 2 WFSs

Parameters

- **n_subaps1** (*int*) – number of sub-apertures in WFS 1
- **n_subaps2** (*int*) – number of sub-apertures in WFS 1
- **wfs1_positions** (*ndarray*) – Central position of each sub-apeture from telescope centre for WFS 1
- **wfs2_positions** (*ndarray*) – Central position of each sub-apeture from telescope centre for WFS 2
- **wfs1_diam** – Diameter of WFS 1 sub-apertures
- **wfs2_diam** – Diameter of WFS 2 sub-apertures
- **r0** – Fried parameter of turbulence
- **L0** – Outer scale of turbulence

Returns slope covariance of X with X , slope covariance of Y with Y, slope covariance of X with Y

aotools.turbulence.slopecovariance.**wfs_covariance_mpwrap**(*args*)

2.2.4 Temporal Power Spectra

Turbulence gradient temporal power spectra calculation and plotting

author Andrew Reeves

date September 2016

aotools.turbulence.temporal_ps.**calc_slope_temporalps**(*slope_data*)
Calculates the temporal power spectra of the loaded centroid data.

Calculates the Fourier transform over the number frames of the centroid data, then returns the square of the mean of all sub-apertures, for x and y. This is a temporal power spectra of the slopes, and should adhere to a -11/3 power law for the slopes in the wind direction, and -14/3 in the direction tranverse to the wind direction. See Conan, 1995 for more.

The phase slope data should be split into X and Y components, with all X data first, then Y (or visa-versa).

Parameters **slope_data** (*ndarray*) – 2-d array of shape (..., nFrames, nCentroids)

Returns The temporal power spectra of the data for X and Y components.

Return type ndarray

aotools.turbulence.temporal_ps.**fit_tps**(*tps*, *t_axis*, *D*, *V_guess=10*, *f_noise_guess=20*, *A_guess=9*, *tps_err=None*, *plot=False*)

Runs minimization routines to get t0.

Parameters

- **tps** (*ndarray*) – The temporal power spectrum to fit
- **axis** (*str*) – fit parallel ('par') or perpendicular ('per') to wind direction
- **D** (*float*) – (Sub-)Aperture diameter

Returns:

aotools.turbulence.temporal_ps.**get_tps_time_axis**(*frame_rate*, *n_frames*)

Parameters

- **frame_rate** (*float*) – Frame rate of detector observing slope gradients (Hz)
- **n_frames** – (int): Number of frames used for temporal power spectrum

Returns Time values for temporal power spectra plots

Return type ndarray

aotools.turbulence.temporal_ps.**plot_tps**(*slope_data*, *frame_rate*)

Generates a plot of the temporal power spectrum/a for a data set of phase gradients

Parameters

- **slope_data** (*ndarray*) – 2-d array of shape (..., nFrames, nCentroids)
- **frame_rate** (*float*) – Frame rate of detector observing slope gradients (Hz)

Returns The computed temporal power spectrum/a, and the time axis data

Return type tuple

2.2.5 Atmospheric Parameter Conversions

Functions for converting between different atmospheric parameters,

aotools.turbulence.atmos_conversions.**cn2_to_r0**(*cn2*, *lamda=5e-07*)

Calculates r0 from the integrated Cn2 value

Parameters

- **cn2** (*float*) – integrated Cn2 value in m^2/3
- **lamda** – wavelength

Returns r0 in m

aotools.turbulence.atmos_conversions.**cn2_to_seeing**(*cn2*, *lamda=5e-07*)

Calculates the seeing angle from the integrated Cn2 value

Parameters

- **cn2** (*float*) – integrated Cn2 value in m^2/3
- **lamda** – wavelength

Returns seeing angle in arcseconds

aotools.turbulence.atmos_conversions.**coherenceTime**(*cn2*, *v*, *lamda=5e-07*)
Calculates the coherence time from profiles of the Cn2 and wind velocity

Parameters

- **cn2** (*array*) – Cn2 profile in m^2/3
- **v** (*array*) – profile of wind velocity, same altitude scale as cn2
- **lamda** – wavelength

Returns coherence time in seconds

aotools.turbulence.atmos_conversions.**isoplanaticAngle**(*cn2*, *h*, *lamda=5e-07*)
Calculates the isoplanatic angle from the Cn2 profile

Parameters

- **cn2** (*array*) – Cn2 profile in m^2/3
- **h** (*Array*) – Altitude levels of cn2 profile in m
- **lamda** – wavelength

Returns isoplanatic angle in arcseconds

aotools.turbulence.atmos_conversions.**r0_from_slopes**(*slopes*, *wavelength*, *subapDiam*)
Measures the value of R0 from a set of WFS slopes.

Uses the equation in Saint Jaques, 1998, PhD Thesis, Appendix A to calculate the value of atmospheric seeing parameter, r0, that would result in the variance of the given slopes.

Parameters

- **slopes** (*ndarray*) – A 3-d set of slopes in radians, of shape (dimension, nSubaps, nFrames)
- **wavelength** (*float*) – The wavelegnth of the light observed
- **subapDiam** (*float*) –

Returns An estimate of r0 for that dataset.

Return type float

aotools.turbulence.atmos_conversions.**r0_to_cn2**(*r0*, *lamda=5e-07*)
Calculates integrated Cn2 value from r0

Parameters

- **r0** (*float*) – r0 in cm
- **lamda** – wavelength

Returns integrated Cn2 value in m^2/3

Return type cn2 (float)

aotools.turbulence.atmos_conversions.**r0_to_seeing**(*r0*, *lamda=5e-07*)
Calculates the seeing angle from r0

Parameters

- **r0** (*float*) – Freid's parameter in cm
- **lamda** – wavelength

Returns seeing angle in arcseconds

aotools.turbulence.atmos_conversions.**seeing_to_cn2**(*seeing*, *lamda=5e-07*)
Calculates the integrated Cn2 value from the seeing

Parameters

- **seeing** (*float*) – seeing in arcseconds
- **lamda** – wavelength

Returns integrated Cn2 value in m^2/3

aotools.turbulence.atmos_conversions.**seeing_to_r0**(*seeing*, *lamda=5e-07*)
Calculates r0 from seeing

Parameters

- **seeing** (*float*) – seeing angle in arcseconds
- **lamda** – wavelength

Returns Freid's parameter in cm

Return type r0 (float)

aotools.turbulence.atmos_conversions.**slope_variance_from_r0**(*r0*, *wavelength*, *subapDiam*)
Uses the equation in Saint Jaques, 1998, PhD Thesis, Appendix A to calculate the slope variance resulting from a value of r0.

Parameters

- **r0** (*float*) – Fried papamerter of turubulence in metres
- **wavelength** (*float*) – Wavelength of light in metres (where 1e-9 is 1nm)
- **subapDiam** (*float*) – Diameter of the aperture in metres

Returns The expected slope variance for a given r0 ValueError

2.3 Image Processing

2.3.1 Centroiders

Functions for centroiding images.

aotools.image_processing.centroiders.**brightest_pixel**(*img*, *threshold*, ***kwargs*)
Centroids using brightest Pixel Algorithm (A. G. Basden et al, MNRAS, 2011)

Finds the nPxlsth brightest pixel, subtracts that value from frame, sets anything below 0 to 0, and finally takes centroid.

Parameters

- **img** (*ndarray*) – 2d or greater rank array of imgs to centroid
- **threshold** (*float*) – Fraction of pixels to use for centroid

Returns Array of centroid values

Return type ndarray

aotools.image_processing.centroiders.**centre_of_gravity**(*img*, *threshold=0*, *min_threshold=0*, ***kwargs*)

Centroids an image, or an array of images. Centroids over the last 2 dimensions. Sets all values under "threshold*max_value" to zero before centroiding Origin at 0,0 index of img.

Parameters

- **img** (*ndarray*) – ([n,]y, x) 2d or greater rank array of imgs to centroid
- **threshold** (*float*) – Percentage of max value under which pixels set to 0

Returns Array of centroid values (2[, n])

Return type ndarray

aotools.image_processing.centroiders.**correlation_centroid**(*im*, *ref*, *threshold=0.0*, *padding=1*)

Correlation Centroider, currently only works for 3d im shape. Performs a simple thresholded COM on the correlation.

Parameters

- **im** – sub-aperture images (t, y, x)
- **ref** – reference image (y, x)
- **threshold** – fractional threshold for COM (0=all pixels, 1=brightest pixel)
- **padding** – factor to zero-pad arrays in Fourier transforms

Returns centroids of im (2, t), given in order x, y

Return type ndarray

aotools.image_processing.centroiders.**cross_correlate**(*x*, *y*, *padding=1*)

2D convolution using FFT, use to generate cross-correlations.

Parameters

- **x** (*array*) – subap image
- **y** (*array*) – reference image
- **padding** (*int*) – Factor to zero-pad arrays in Fourier transforms

Returns cross-correlation of x and y

Return type ndarray

aotools.image_processing.centroiders.**quadCell**(*img*, ***kwargs*)

Centroider to be used for 2x2 images.

Parameters **img** – 2d or greater rank array, where centroiding performed over last 2 dimensions

Returns Array of centroid values

Return type ndarray

2.3.2 Contrast

Functions for calculating the contrast of an image.

aotools.image_processing.contrast.**image_contrast**(*image*)

Calculates the 'Michelson' contrast.

Uses a method by Michelson (Michelson, A. (1927). Studies in Optics. U. of Chicago Press.), to calculate the contrast ratio
(img_max - img_min)/(img_max + img_min)

Parameters **image** (*ndarray*) – Image array

Returns Contrast value

Return type float

aotools.image_processing.contrast.**rms_contrast**(*image*)
Calculates the RMS contrast - basically the standard deviation of the image

Parameters **image** (*ndarray*) – Image array

Returns Contrast value

Return type float

2.3.3 PSF Analysis

Functions for analysing PSFs.

aotools.image_processing.psf.**azimuthal_average**(*data*)
Measure the azimuthal average of a 2d array

Parameters **data** (*ndarray*) – A 2-d array of data

Returns A 1-d vector of the azimuthal average

Return type ndarray

aotools.image_processing.psf.**encircled_energy**(*data*, *fraction=0.5*, *center=None*, *eeDiameter=True*)
Return the encircled energy diameter for a given fraction (default is ee50d). Can also return the encircled energy function. Translated and extended from YAO.

Parameters

- **data** – 2-d array
- **fraction** – energy fraction for diameter calculation default = 0.5
- **center** – default = center of image
- **eeDiameter** – toggle option for return. If False returns two vectors: (x, ee(x)) Default = True

Returns Encircled energy diameter or 2 vectors: diameters and encircled energies

2.4 Circular Functions

2.4.1 Zernike Modes

aotools.functions.zernike.**makegammas**(*nzrad*)
Make "Gamma" matrices which can be used to determine first derivative of Zernike matrices (Noll 1976).

Parameters **nzrad** – Number of Zernike radial orders to calculate Gamma matrices for

Returns Array with x, then y gamma matrices

Return type ndarray

aotools.functions.zernike.**phaseFromZernikes**(*zCoeffs*, *size*, *norm='noll'*)
Creates an array of the sum of zernike polynomials with specified coefficeints

Parameters

- **zCoeffs** (*list*) – zernike Coefficients
- **size** (*int*) – Diameter of returned array
- **norm** (*string, optional*) – The normalisation of Zernike modes. Can be "noll", "p2v" (peak to valley), or "rms". default is "noll".

Returns a *size* x *size* array of summed Zernike polynomials

Return type ndarray

aotools.functions.zernike.**zernIndex**(*j*)
Find the [n,m] list giving the radial order n and azimuthal order of the Zernike polynomial of Noll index j.

Parameters **j** (*int*) – The Noll index for Zernike polynomials

Returns n, m values

Return type list

aotools.functions.zernike.**zernikeArray**(*J*, *N*, *norm='noll'*)
Creates an array of Zernike Polynomials

Parameters

- **maxJ** (*int or list*) – Max Zernike polynomial to create, or list of zernikes J indices to create
- **N** (*int*) – size of created arrays
- **norm** (*string, optional*) – The normalisation of Zernike modes. Can be "noll", "p2v" (peak to valley), or "rms". default is "noll".

Returns array of Zernike Polynomials

Return type ndarray

aotools.functions.zernike.**zernikeRadialFunc**(*n*, *m*, *r*)
Fucntion to calculate the Zernike radial function

Parameters

- **n** (*int*) – Zernike radial order
- **m** (*int*) – Zernike azimuthal order
- **r** (*ndarray*) – 2-d array of radii from the centre the array

Returns The Zernike radial function

Return type ndarray

aotools.functions.zernike.**zernike_nm**(*n*, *m*, *N*)
Creates the Zernike polynomial with radial index, n, and azimuthal index, m.

Parameters

- **n** (*int*) – The radial order of the zernike mode
- **m** (*int*) – The azimuthal order of the zernike mode
- **N** (*int*) – The diameter of the zernike more in pixels

Returns The Zernike mode

Return type ndarray

aotools.functions.zernike.**zernike_noll**(*j*, *N*)

Creates the Zernike polynomial with mode index j, where j = 1 corresponds to piston.

Parameters

- **j** (*int*) – The noll j number of the zernike mode
- **N** (*int*) – The diameter of the zernike more in pixels

Returns The Zernike mode

Return type ndarray

2.4.2 Karhunen Loeve Modes

Collection of routines related to Karhunen-Loeve modes.

The theory is based on the paper "Optimal bases for wave-front simulation and reconstruction on annular apertures", Robert C. Cannon, 1996, JOSAA, 13, 4

The present implementation is based on the IDL package of R. Cannon (wavefront modelling and reconstruction). A closely similar implementation can also be find in Yorick in the YAO package.

Usage

Main routine is 'make_kl' to generate KL basis of dimension `[dim, dim, nmax]`.

For Kolmogorov statistics, e.g.

```
kl, _, _, _ = make_kl(150, 128, ri = 0.2, stf='kolmogorov')
```

Warning: make_kl with von Karman stf fails. It has been implemented but KL generation failed in 'while loop' of gkl_fcom...

date November 2017

aotools.functions.karhunenLoeve.**gkl_azimuthal**(*nord*, *npp*)

Compute the azimuthal function of the KL basis.

Parameters

- **nord** (*int*) – number of azimuthal orders
- **npp** (*int*) – grid of point sampling the azimuthal coordinate

Returns **gklazi** – azimuthal function of the KL basis.

Return type ndarray

aotools.functions.karhunenLoeve.**gkl_basis**(*ri=0.25*, *nr=40*, *npp=None*, *nfunc=500*, *stf='kolstf'*, *outerscale=None*)

Wrapper to create the radial and azimuthal K-L functions.

Parameters

- **ri** (*float*) – normalized internal radius
- **nr** (*int*) – number of radial resolution elements

- **np** (*int*) – number of azimuthal resolution elements
- **nfunc** (*int*) – number of generated K-L function
- **stf** (*string*) – structure function tag describing the atmospheric statistics

Returns **gklbasis** – dictionary containing the radial and azimuthal basis + other relevant information

Return type dic

aotools.functions.karhunenLoeve.**gkl_fcom**(*ri*, *kernels*, *nfunc*, *verbose=False*)

Computation of the radial eigenvectors of the KL basis.

Obtained by taking the eigenvectors from the matrix L^p. The final function corresponds to the 'nfunc' largest eigenvalues. See eq. 16-18 in Cannon 1996.

Parameters

- **ri** (*float*) – radius of central obscuration radius (normalized by D/2; <1)
- **kernels** (*ndarray*) – kernel L^p of dimension (nr, nr, nth), where nth is the azimuthal discretization (5 * nr)
- **nfunc** (*int*) – number of final KL functions

Returns

- **evals** (*1darray*) – eigenvalues
- **nord** (*int*) – resulting number of azimuthal orders
- **npo** (*int*)
- **ord** (*1darray*)
- **rabas** (*ndarray*) – radial eigenvectors of the KL basis

aotools.functions.karhunenLoeve.**gkl_kernel**(*ri*, *nr*, *rad*, *stfunc='kolmogorov'*, *outerscale=None*)

Calculation of the kernel L^p

The kernel constructed here should be simply a discretization of the continuous kernel. It needs rescaling before it is treated as a matrix for finding the eigen-values

Parameters

- **ri** (*float*) – radius of central obscuration radius (normalized by D/2; <1)
- **nr** (*int*) – number of resolution elements
- **rad** (*1d-array*) – grid of points where the kernel is evaluated
- **stfunc** (*string*) – string tag of the structure function on which the kernel are computed
- **outerscale** (*float*) – in unit of telescope diameter. Outer-scale for von Karman structure function.

Returns **L^p** – kernel L^p of dimension (nr, nr, nth), where nth is the azimuthal discretization (5 * nr)

Return type ndarray

aotools.functions.karhunenLoeve.**gkl_radii**(*ri*, *nr*)

Generate n points evenly spaced in r^2 between r_i^2 and 1

Parameters

- **nr** (*int*) – number of resolution elements

- **ri** (*float*) – radius of central obscuration radius (normalized; <1)

Returns **r** – grid of point to calculate the appropriate kernels. correspond to 'sqrt(s)' wrt the paper.

Return type 1d-array, float

aotools.functions.karhunenLoeve.**gkl_sfi**(*kl_basis*, *i*)
return the i'th function from the generalized KL basis 'bas'. 'bas' must be generated by 'gkl_basis'

aotools.functions.karhunenLoeve.**make_kl**(*nmax*, *dim*, *ri=0.0*, *nr=40*, *stf='kolmogorov'*, *outerscale=None*, *mask=True*)
Main routine to generatre a KL basis of dimension [nmax, dim, dim].

For Kolmogorov statistics, e.g.

```
kl, _, _, _ = make_kl(150, 128, ri = 0.2, stf='kolmogorov')
```

As a rule of thumb
nr x npp = 50 x 250 is fine up to 500 functions
60 x 300 for a thousand
80 x 400 for three thousands.

Parameters

- **nmax** (*int*) – number of KL function to generate
- **dim** (*int*) – size of the KL arrays
- **ri** (*float*) – radial central obscuration normalized by D/2
- **nr** (*int*) – number of point on radius. npp (number of azimuthal pts) is =2 pi * nr
- **stf** (*string*) – structure function tag. Default is 'kolmogorov'
- **outerscale** (*float*) – outer scale in units of telescope diameter. Releveant if von vonKarman stf. (not implemented yet)
- **mask** (*bool*) – pupil masking. Default is True

Returns

- **kl** (*ndarray (nmax, dim, dim)*) – KL basis in cartesian coordinates
- **varKL** (*1darray*) – associated variance
- **pupil** (*2darray*) – pupil
- **polar_base** (*dictionary*) – polar base dictionary used for the KL basis computation. as returned by 'gkl_basis'

See also:

gkl_basis(), *set_pctr()*, *pol2car()*

aotools.functions.karhunenLoeve.**pcgeom**(*nr*, *npp*, *ncp*, *ri*, *ncmar*)
This routine builds a geom dic.

Parameters

- **py** (*px,*) – the x, y coordinates of points in the polar arrays.
- **cp** (*cr,*) – the r, phi coordinates of points in the cartesian grids.

- **ncmar** – allows the possibility that there is a margin of ncmar points in the cartesian arrays outside the region of interest.

aotools.functions.karhunenLoeve.**piston_orth**(*nr*)
Unitary matrix used to filter out piston term. Eq. 19 in Cannon 1996.

Parameters **nr** (*int*) – number of resolution elements

Returns **U** – Unitary matrix

Return type 2d array

aotools.functions.karhunenLoeve.**pol2car**(*cpgeom*, *pol*, *mask=False*)
Polar to cartesian conversion.

(points not in the aperture are treated as though they were at the first or last radial polar value...)

aotools.functions.karhunenLoeve.**polang**(*r*)
Generate an array with the same dimensions as r, but containing the azimuthal values for a polar coordinate system.

aotools.functions.karhunenLoeve.**radii**(*nr*, *npp*, *ri*)
Use to generate a polar coordinate system.

Generate an nr x npp array with npp copies of the radial coordinate array. Radial coordinate span the range from r=ri to r=1 with successive annuli having equal areas.

ie, the area between ri and 1 is divided into nr equal rings, and the points are positioned at the half-area mark on each ring; there are no points on the border

See also:

polang()

aotools.functions.karhunenLoeve.**rebin**(*a*, *newshape*)
Rebin an array to a new shape. See scipy cookbook. It is intended to be similar to 'rebin' of IDL

aotools.functions.karhunenLoeve.**set_pctr**(*bas*, *ncp=None*, *ncmar=None*)
call pcgeom to build the dic geom_struct with the right initializtions.

Parameters **bas** (*dic*) – gkl_basis dic built with the gkl_bas routine

aotools.functions.karhunenLoeve.**setpincs**(*ax*, *ay*, *px*, *py*, *ri*)
determine a set of squares for interpolating from cartesian to polar coordinates, using only those points with ri <= r <= 1

aotools.functions.karhunenLoeve.**stf_kolmogorov**(*r*)
Kolmogorov structure function with r = (D/r0)

aotools.functions.karhunenLoeve.**stf_vonKarman**(*r*, *L0*)
von Karman structure function with r = (D / r0) L0 is in unit of telescope diameter, typically a few (3; or 20m)

aotools.functions.karhunenLoeve.**stf_vonKarman_yao**(*r*, *L*)

2.5 Telescope Pupils

2.5.1 Pupil Maps

Functions for the creation of pupil maps and masks.

aotools.functions.pupil.**circle**(*radius*, *size*, *circle_centre=(0, 0)*, *origin='middle'*)

Create a 2-D array: elements equal 1 within a circle and 0 outside.

The default centre of the coordinate system is in the middle of the array: circle_centre=(0,0), origin="middle" This means: if size is odd : the centre is in the middle of the central pixel if size is even : centre is in the corner where the central 4 pixels meet

origin = "corner" is used e.g. by psfAnalysis:radialAvg()

Examples:

```
circle(1,5) circle(0,5) circle(2,5) circle(0,4) circle(0.8,4) circle(2,4)
  00000       00000       00100       0000        0000          0110
  00100       00000       01110       0000        0110          1111
  01110       00100       11111       0000        0110          1111
  00100       00000       01110       0000        0000          0110
  00000       00000       00100

circle(1,5,(0.5,0.5))   circle(1,4,(0.5,0.5))
   .-->+
   |  00000               0000
   |  00000               0010
  +V  00110               0111
      00110               0010
      00000
```

Parameters

- **radius** (*float*) – radius of the circle
- **size** (*int*) – size of the 2-D array in which the circle lies
- **circle_centre** (*tuple*) – coords of the centre of the circle
- **origin** (*str*) – where is the origin of the coordinate system in which circle_centre is given; allowed values: {"middle", "corner"}

Returns the circle array

Return type ndarray (float64)

2.6 Wavefront Sensors

aotools.wfs.**computeFillFactor**(*mask*, *subapPos*, *subapSpacing*)

Calculate the fill factor of a set of sub-aperture co-ordinates with a given pupil mask.

Parameters

- **mask** (*ndarray*) – Pupil mask
- **subapPos** (*ndarray*) – Set of n sub-aperture co-ordinates (n, 2)
- **subapSpacing** – Number of mask pixels between sub-apertures

Returns fill factor of sub-apertures

Return type list

aotools.wfs.**findActiveSubaps**(*subaps*, *mask*, *threshold*, *returnFill=False*)

Finds the subapertures which are "seen" be through the pupil function. Returns the coords of those subapertures

Parameters

- **subaps** (*int*) – The number of subaps in x (assumes square)
- **mask** (*ndarray*) – A pupil mask, where is transparent when 1, and opaque when 0
- **threshold** (*float*) – The mean value across a subap to make it "active"
- **returnFill** (*optional, bool*) – Return an array of fill-factors

Returns An array of active subap coords

Return type ndarray

aotools.wfs.**make_subaps_2d**(*data*, *mask*)

Fills in a pupil shape with 2-d sub-apertures

Parameters

- **data** (*ndarray*) – slope data, of shape, (frames, 2, nSubaps)
- **mask** (*ndarray*) – 2-d array of shape (nxSubaps, nxSubaps), where 1 indicates a valid subap position and 0 a masked subap

2.7 Optical Propagation

A library of useful optical propagation methods.

Many extracted from the book by Schmidt, 2010: Numerical Methods of optical proagation

aotools.opticalpropagation.**angularSpectrum**(*inputComplexAmp*, *wvl*, *inputSpacing*, *outputSpacing*, *z*)

Propogates light complex amplitude using an angular spectrum algorithm

Parameters

- **inputComplexAmp** (*ndarray*) – Complex array of input complex amplitude
- **wvl** (*float*) – Wavelength of light to propagate
- **inputSpacing** (*float*) – The spacing between points on the input array in metres
- **outputSpacing** (*float*) – The desired spacing between points on the output array in metres
- **z** (*float*) – Distance to propagate in metres

Returns propagated complex amplitude

Return type ndarray

aotools.opticalpropagation.**lensAgainst**(*Uin*, *wvl*, *d1*, *f*)

Propagates from the pupil plane to the focal plane for an object placed against (and just before) a lens.

Parameters

- **Uin** (*ndarray*) – Input complex amplitude
- **wvl** (*float*) – Wavelength of light in metres
- **d1** (*float*) – spacing of input plane
- **f** (*float*) – Focal length of lens

Returns Output complex amplitude

Return type ndarray

aotools.opticalpropagation.**oneStepFresnel**(*Uin*, *wvl*, *d1*, *z*)
Fresnel propagation using a one step Fresnel propagation method.

Parameters

- **Uin** (*ndarray*) – A 2-d, complex, input array of complex amplitude
- **wvl** (*float*) – Wavelength of propagated light in metres
- **d1** (*float*) – spacing of input plane
- **z** (*float*) – metres to propagate along optical axis

Returns Complex ampltitude after propagation

Return type ndarray

aotools.opticalpropagation.**twoStepFresnel**(*Uin*, *wvl*, *d1*, *d2*, *z*)
Fresnel propagation using a two step Fresnel propagation method.

Parameters

- **Uin** (*ndarray*) – A 2-d, complex, input array of complex amplitude
- **wvl** (*float*) – Wavelength of propagated light in metres
- **d1** (*float*) – spacing of input plane
- **d2** (*float*) – desired output array spacing
- **z** (*float*) – metres to propagate along optical axis

Returns Complex ampltitude after propagation

Return type ndarray

2.8 Astronomical Functions

2.8.1 Magnitude and Flux Calculations

aotools.astronomy.**flux_to_magnitude**(*flux*, *waveband='V'*)
Converts incident flux of photons to the apparent magnitude

Parameters

- **flux** (*float*) – Number of photons received from an object per second per meter squared
- **waveband** (*string*) – Waveband of the measured flux, can be U, B, V, R, I, J, H, K, g, r, i, z

Returns Apparent magnitude

Return type float

aotools.astronomy.**magnitude_to_flux**(*magnitude*, *waveband='V'*)
Converts apparent magnitude to a flux of photons

Parameters

- **magnitude** (*float*) – Star apparent magnitude
- **waveband** (*string*) – Waveband of the stellar magnitude, can be U, B, V, R, I, J, H, K, g, r, i, z

Returns Number of photons emitted by the object per second per meter squared

Return type float

aotools.astronomy.**photons_per_band**(*mag*, *mask*, *pxlScale*, *expTime*, *waveband='V'*)
Calculates the photon flux for a given aperture, star magnitude and wavelength band

Parameters

- **mag** (*float*) – Star apparent magnitude
- **mask** (*ndarray*) – 2-d pupil mask array, 1 is transparent, 0 opaque
- **pxlScale** (*float*) – size in metres of each pixel in mask
- **expTime** (*float*) – Exposure time in seconds
- **waveband** (*string*) – Waveband

Returns number of photons

Return type float

aotools.astronomy.**photons_per_mag**(*mag*, *mask*, *pixel_scale*, *wvlBand*, *exposure_time*)
Calculates the photon flux for a given aperture, star magnitude and wavelength band

Parameters

- **mag** (*float*) – Star apparent magnitude
- **mask** (*ndarray*) – 2-d pupil mask array, 1 is transparent, 0 opaque
- **pixel_scale** (*float*) – size in metres of each pixel in mask
- **wvlBand** (*float*) – length of wavelength band in nanometres
- **exposure_time** (*float*) – Exposure time in seconds

Returns number of photons

Return type float

2.9 Normalised Fourier Transforms

Module containing useful FFT based function and classes

aotools.fouriertransform.**ft**(*data*, *delta*)
A properly scaled 1-D FFT

Parameters

- **data** (*ndarray*) – An array on which to perform the FFT
- **delta** (*float*) – Spacing between elements

Returns scaled FFT

Return type ndarray

aotools.fouriertransform.**ft2**(*data*, *delta*)
A properly scaled 2-D FFT

Parameters

- **data** (*ndarray*) – An array on which to perform the FFT
- **delta** (*float*) – Spacing between elements

Returns scaled FFT

Return type ndarray

aotools.fouriertransform.**ift**(*data*, *delta_f*)
Scaled inverse 1-D FFT

Parameters

- **data** (*ndarray*) – Data in Fourier Space to transform
- **delta_f** (*ndarray*) – Frequency spacing of grid

Returns Scaled data in real space

Return type ndarray

aotools.fouriertransform.**ift2**(*data*, *delta_f*)
Scaled inverse 2-D FFT

Parameters

- **data** (*ndarray*) – Data in Fourier Space to transform
- **delta_f** (*ndarray*) – Frequency spacing of grid

Returns Scaled data in real space

Return type ndarray

aotools.fouriertransform.**irft**(*data*, *delta_f*)
Scaled real inverse 1-D FFT

Parameters

- **data** (*ndarray*) – Data in Fourier Space to transform
- **delta_f** (*ndarray*) – Frequency spacing of grid

Returns Scaled data in real space

Return type ndarray

aotools.fouriertransform.**irft2**(*data*, *delta_f*)
Scaled inverse real 2-D FFT

Parameters

- **data** (*ndarray*) – Data in Fourier Space to transform
- **delta_f** (*ndarray*) – Frequency spacing of grid

Returns Scaled data in real space

Return type ndarray

aotools.fouriertransform.**rft**(*data*, *delta*)
A properly scaled real 1-D FFT

Parameters

- **data** (*ndarray*) – An array on which to perform the FFT
- **delta** (*float*) – Spacing between elements

Returns scaled FFT

Return type ndarray

aotools.fouriertransform.**rft2**(*data*, *delta*)
A properly scaled, real 2-D FFT

Parameters

- **data** (*ndarray*) – An array on which to perform the FFT
- **delta** (*float*) – Spacing between elements

Returns scaled FFT

Return type ndarray

2.10 Interpolation

aotools.interpolation.**binImgs**(*data*, *n*)

Bins one or more images down by the given factor bins. n must be a factor of data.shape, who knows what happens otherwise.....

aotools.interpolation.**zoom**(*array*, *newSize*, *order=3*)

A Class to zoom 2-dimensional arrays using interpolation

Uses the scipy *Interp2d* interpolation routine to zoom into an array. Can cope with real of complex data.

Parameters

- **array** (*ndarray*) – 2-dimensional array to zoom
- **newSize** (*tuple*) – the new size of the required array
- **order** (*int, optional*) – Order of interpolation to use (1, 3, 5). default is 3

Returns zoom array of new size.

Return type ndarray

aotools.interpolation.**zoom_rbs**(*array*, *newSize*, *order=3*)

A Class to zoom 2-dimensional arrays using RectBivariateSpline interpolation

Uses the scipy `RectBivariateSpline` interpolation routine to zoom into an array. Can cope with real of complex data. May be slower than above `zoom`, as RBS routine copies data.

Parameters

- **array** (*ndarray*) – 2-dimensional array to zoom
- **newSize** (*tuple*) – the new size of the required array
- **order** (*int, optional*) – Order of interpolation to use. default is 3

Returns zoom array of new size.

Return type ndarray

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

aotools.astronomy, 23
aotools.fouriertransform, 24
aotools.functions.karhunenLoeve, 17
aotools.functions.pupil, 20
aotools.functions.zernike, 15
aotools.image_processing.centroiders, 13
aotools.image_processing.contrast, 14
aotools.image_processing.psf, 15
aotools.interpolation, 26
aotools.opticalpropagation, 22
aotools.turbulence.atmos_conversions, 11
aotools.turbulence.infinitephasescreen, 4
aotools.turbulence.phasescreen, 3
aotools.turbulence.slopecovariance, 7
aotools.turbulence.temporal_ps, 10
aotools.wfs, 21

Index

A

add_row() (*aotools.turbulence.infinitephasescreen.PhaseScreenKolmogorov method*), 7
add_row() (*aotools.turbulence.infinitephasescreen.PhaseScreenVonKarman method*), 5
angularSpectrum() (*in module aotools.opticalpropagation*), 22
aotools.astronomy (*module*), 23
aotools.fouriertransform (*module*), 24
aotools.functions.karhunenLoeve (*module*), 17
aotools.functions.pupil (*module*), 20
aotools.functions.zernike (*module*), 15
aotools.image_processing.centroiders (*module*), 13
aotools.image_processing.contrast (*module*), 14
aotools.image_processing.psf (*module*), 15
aotools.interpolation (*module*), 26
aotools.opticalpropagation (*module*), 22
aotools.turbulence.atmos_conversions (*module*), 11
aotools.turbulence.infinitephasescreen (*module*), 4
aotools.turbulence.phasescreen (*module*), 3
aotools.turbulence.slopecovariance (*module*), 7
aotools.turbulence.temporal_ps (*module*), 10
aotools.wfs (*module*), 21
azimuthal_average() (*in module aotools.image_processing.psf*), 15

B

binImgs() (*in module aotools.interpolation*), 26
brightest_pixel() (*in module aotools.image_processing.centroiders*), 13

C

calc_temporalps() (*in module aotools.turbulence.temporal_ps*), 10
calculate_structure_function() (*in module aotools.turbulence.slopecovariance*), 8
calculate_wfs_seperations() (*in module aotools.turbulence.slopecovariance*), 8
centre_of_gravity() (*in module aotools.image_processing.centroiders*), 13
circle() (*in module aotools.functions.pupil*), 20
cn2_to_r0() (*in module aotools.turbulence.atmos_conversions*), 11
cn2_to_seeing() (*in module aotools.turbulence.atmos_conversions*), 11
coherenceTime() (*in module aotools.turbulence.atmos_conversions*), 11
computeFillFactor() (*in module aotools.wfs*), 21
correlation_centroid() (*in module aotools.image_processing.centroiders*), 14
CovarianceMatrix (*class in aotools.turbulence.slopecovariance*), 7
create_tomographic_covariance_reconstructor() (*in module aotools.turbulence.slopecovariance*), 9
cross_correlate() (*in module aotools.image_processing.centroiders*), 14

E

encircled_energy() (*in module aotools.image_processing.psf*), 15

F

findActiveSubaps() (*in module aotools.wfs*), 21
fit_tps() (*in module aotools.turbulence.temporal_ps*), 11
flux_to_magnitude() (*in module aotools.astronomy*), 23
ft() (*in module aotools.fouriertransform*), 24
ft2() (*in module aotools.fouriertransform*), 24

ft_phase_screen() (*in module aotools.turbulence.phasescreen*), 3
ft_sh_phase_screen() (*in module aotools.turbulence.phasescreen*), 4

G

get_tps_time_axis() (*in module aotools.turbulence.temporal_ps*), 11
gkl_azimuthal() (*in module aotools.functions.karhunenLoeve*), 17
gkl_basis() (*in module aotools.functions.karhunenLoeve*), 17
gkl_fcom() (*in module aotools.functions.karhunenLoeve*), 18
gkl_kernel() (*in module aotools.functions.karhunenLoeve*), 18
gkl_radii() (*in module aotools.functions.karhunenLoeve*), 18
gkl_sfi() (*in module aotools.functions.karhunenLoeve*), 19

I

ift() (*in module aotools.fouriertransform*), 25
ift2() (*in module aotools.fouriertransform*), 25
image_contrast() (*in module aotools.image_processing.contrast*), 14
irft() (*in module aotools.fouriertransform*), 25
irft2() (*in module aotools.fouriertransform*), 25
isoplanaticAngle() (*in module aotools.turbulence.atmos_conversions*), 12

L

lensAgainst() (*in module aotools.opticalpropagation*), 22

M

magnitude_to_flux() (*in module aotools.astronomy*), 23
make_covariance_matrix() (*aotools.turbulence.slopecovariance.CovarianceMatrix method*), 8
make_kl() (*in module aotools.functions.karhunenLoeve*), 19
make_subaps_2d() (*in module aotools.wfs*), 22
make_tomographic_reconstructor() (*aotools.turbulence.slopecovariance.CovarianceMatrix method*), 8
makegammas() (*in module aotools.functions.zernike*), 15
mirror_covariance_matrix() (*in module aotools.turbulence.slopecovariance*), 9

O

oneStepFresnel() (*in module aotools.opticalpropagation*), 22

P

pcgeom() (*in module aotools.functions.karhunenLoeve*), 19
phaseFromZernikes() (*in module aotools.functions.zernike*), 15
PhaseScreenKolmogorov (*class in aotools.turbulence.infinitephasescreen*), 5
PhaseScreenVonKarman (*class in aotools.turbulence.infinitephasescreen*), 4
photons_per_band() (*in module aotools.astronomy*), 24
photons_per_mag() (*in module aotools.astronomy*), 24
piston_orth() (*in module aotools.functions.karhunenLoeve*), 20
plot_tps() (*in module aotools.turbulence.temporal_ps*), 11
pol2car() (*in module aotools.functions.karhunenLoeve*), 20
polang() (*in module aotools.functions.karhunenLoeve*), 20

Q

quadCell() (*in module aotools.image_processing.centroiders*), 14

R

r0_from_slopes() (*in module aotools.turbulence.atmos_conversions*), 12
r0_to_cn2() (*in module aotools.turbulence.atmos_conversions*), 12
r0_to_seeing() (*in module aotools.turbulence.atmos_conversions*), 12
radii() (*in module aotools.functions.karhunenLoeve*), 20
rebin() (*in module aotools.functions.karhunenLoeve*), 20
rft() (*in module aotools.fouriertransform*), 25
rft2() (*in module aotools.fouriertransform*), 25
rms_contrast() (*in module aotools.image_processing.contrast*), 15

S

scrn (*aotools.turbulence.infinitephasescreen.PhaseScreenKolmogorov attribute*), 7
scrn (*aotools.turbulence.infinitephasescreen.PhaseScreenVonKarman attribute*), 5
seeing_to_cn2() (*in module aotools.turbulence.atmos_conversions*), 12

seeing_to_r0() (*in module aotools.turbulence.atmos_conversions*), 13
set_pctr() (*in module aotools.functions.karhunenLoeve*), 20
setpincs() (*in module aotools.functions.karhunenLoeve*), 20
slope_variance_from_r0() (*in module aotools.turbulence.atmos_conversions*), 13
stf_kolmogorov() (*in module aotools.functions.karhunenLoeve*), 20
stf_vonKarman() (*in module aotools.functions.karhunenLoeve*), 20
stf_vonKarman_yao() (*in module aotools.functions.karhunenLoeve*), 20
structure_function_kolmogorov() (*in module aotools.turbulence.slopecovariance*), 9
structure_function_vk() (*in module aotools.turbulence.slopecovariance*), 9

T

twoStepFresnel() (*in module aotools.opticalpropagation*), 23

W

wfs_covariance() (*in module aotools.turbulence.slopecovariance*), 10
wfs_covariance_mpwrap() (*in module aotools.turbulence.slopecovariance*), 10

Z

zernike_nm() (*in module aotools.functions.zernike*), 16
zernike_noll() (*in module aotools.functions.zernike*), 17
zernikeArray() (*in module aotools.functions.zernike*), 16
zernikeRadialFunc() (*in module aotools.functions.zernike*), 16
zernIndex() (*in module aotools.functions.zernike*), 16
zoom() (*in module aotools.interpolation*), 26
zoom_rbs() (*in module aotools.interpolation*), 26