SHESHA Documentation

Release 602

COMPASS team

October 28, 2015

CONTENTS

Please, read the README file to install the Shesha module:

CHAPTER

ONE

CONTENTS:

1.1 Atmos

1.1.1 atmos module

class atmos.**Atmos**

Bases: object

add_screen()

Add a screen to the atmos object.

Parameters size: (float) : dimension of the screen (size x size)

amplitude: (float) : frac

altitude: (float) : altitude of the screen in meters

windspeed: (float) : windspeed of the screen [m/s]

winddir: (float) : wind direction (°)

deltax: (float) : extrude deltax pixels in the x-direction at each iteration

deltay: (float) : extrude deltay pixels in the y-direction at each iteration

device: (int) : device number

del_screen()

Delete a screen from the atmos object

Parameters **alt** – (float) : altitude of the screen to delete

disp()

Display the screen phase at a given altitude

Parameters **alt** – (float) : altitude of the screen to display

get_screen()

Return a numpy array containing the turbulence at a given altitude

Parameters **alt** – (float) :altitude of the screen to get

list_alt()

Display the list of the screens altitude

move_atmos()

Move the turbulence in the atmos screen following previous loaded paramters such as windspeed and wind direction

atmos.**atmos_init**()
Create and initialise an atmos object

Parameters c: (naga_context) : context

tel: (Param_tel) : telescope settings

geom: (Param_geom) : geometry settings

loop: (Param_loop) : loop settings

wfss: (list of Param_wfs) : (optional) wfs settings

target: (Param_target) : (optional) target_settings

1.2 Target

1.2.1 target module

class target.**Target**
Bases: object

Lambda
observation wavelength for each target

add_layer()
Add a phase screen dm or atmos as layers of turbulence

Parameters n: (int) : index of the target l_type: (str) : "atmos" or "dm"

alt: (float) : altitude

xoff: (float) : x-offset

yoff: (float) : y-offset

apod
boolean for apodizer

atmos_trace()
Raytracing of the target through the atmosphere

Parameters int: (nTarget) : index of the target

atm: (atmos) : atmos to get through

dmtrace()

get_amplipup()
Return the complex amplitude in the pupil plane of the target.

Parameters **nTarget** – (int) : index of the target

get_image()
Return the image from the target (or long exposure image according to the requested type) :parameters:

nTarget: (int) : index of the target

type_im: (str) : type of the image to get ("se" or "le")

puponly: (int) : if 1, image computed from phase on the pupil only

get_phase()
Return the phase's screen of the target

Parameters **nTarget** – (int) : index of the target

get_phasetele()
Return the telemetry phase of the target

Parameters **nTarget** – (int) : index of the target

Return data (np.ndarray(ndim=2,np.float32)) : phase screen

get_strehl()
Return the target's strehl

Parameters **nTarget** – (int) : index of the target

init_strehlmeter()
Initialise target's strehl

Parameters **nTarget** – (int) : index of the target

mag
magnitude for each target

ntargets
number of targets

xpos
x positions on sky (in arcsec) for each target

ypos
y positions on sky (in arcsec) for each target

target.**target_init**()
Create a cython target from parametres structures

Parameters ctxt: (naga_context) :

atm: (Param_atmos) : atmos settings

geom: (Param_geom) : geom settings

wfs: (Param_wfs) : wfs settings

dm: (Param_dm) : dm settings

1.3 Sensors

1.3.1 sensors module

class sensors.**Sensors**
Bases: object

Constructor: Sensors(nsensors,type_data,npup,nxsub,nvalid,nphase,pdiam,npix,nrebin,nfft,nftota,nphot,lgs,odevice,comm_size,ra

Parameters nsensors: (int) :

type_data: list of strings) :

npup: (np.ndarray[ndim=1,dtype=np.int64_t]) :

nxsub: (np.ndarray[ndim=1,dtype=np.int64_t]) :

nvalid: (np.ndarray[ndim=1,dtype=np.int64_t]) :

nphase: (np.ndarray[ndim=1,dtype=np.int64_t]) :

pdiam: (np.ndarray[ndim=1,dtype=np.float32_t) :

npix: (np.ndarray[ndim=1,dtype=np.int64_t]) :

nrebin: (np.ndarray[ndim=1,dtype=np.int64_t]) :

nfft: (np.ndarray[ndim=1,dtype=np.int64_t]) :

ntota: (np.ndarray[ndim=1,dtype=np.int64_t]) :

nphot: (np.ndarray[ndim=1,dtype=np.float32_t]) :

lgs: (np.ndarray[ndim=1,dtype=np.int32_t]) :

odevice: (int) :

comm_size: (int) : MPI communicator size

rank: (int) : process rank

get_amplifoc()
Return the 'amplifoc' array of a given wfs

Parameters **n** – (int) : number of the wfs to get the 'amplifoc' from

get_bincube()
Return the 'bincube' array of a given wfs

Parameters **n** – (int) : number of the wfs to get the 'bincube' from

get_binimg()
Return the 'binimg' array of a given wfs

Parameters **n** – (int) :number of the wfs to get the 'binimg' from

get_camplipup()
Return the 'camplipup' array of a given wfs

Parameters **n** – (int) : number of the wfs to get the 'camplipup' from

get_imgtele()
Return the 'image_telemetry' array of a given wfs

Parameters **n** – (int) : number of the wfs to get the 'image_telemetry' from

get_offsets()
Return the 'offset' array of a given wfs

Parameters **n** – (int) : number of the wfs to get the 'offset' from

get_phase()
Return the phase array of a given wfs

Parameters **n** – (int) : number of the wfs to get the phase from

get_pyrimg()
Return the image of a pyr wfs

Parameters **n** – (int) : number of the wfs to get the image from

get_rank()
Return the rank of one of the sensors wfs

Parameters **n** – (int) : index of the wfs to get the rank for

get_slopes()

Return the 'slopes' array of a given wfs

Parameters **n** – (int) : number of the wfs to get the 'slopes' from

sensors_addlayer()

Call function add_layer from the sutra_source of a sutra_wfs of the Sensors

Parameters i: (int) :

type_dm: (string) :

alt: (float) :

xoff: (float) :

yoff: (float) :

sensors_compimg()

TODO doc

Parameters **n** – (in) : index of the wfs

sensors_initarr()

Call the function wfs_initarrays from a sutra_wfs of the Sensors

Parameters

n: (int) : index of the wfs

wfs: (Param_wfs) :

geom: (Param_geom) :

sensors_initgs()

Call the function sensors_initgs

Parameters xpos: (np.ndarray[ndim=1,dtype=np.float32_t]) :

ypos: (np.ndarray[ndim=1,dtype=np.float32_t]) :

Lambda: (np.ndarray[ndim=1,dtype=np.float32_t]) :

mag: (np.ndarray[ndim=1,dtype=np.float32_t]) :

zerop: (float) :

size: (np.ndarray[ndim=1,dtype=np.int64_t]) :

noise: (np.ndarray[ndim=1,dtype=np.float32_t]) :

seed: (np.ndarray[ndim=1,dtype=np.int64_t]) :

sensors_trace()

Does the raytracing for the wfs phase screen in sutra_wfs

Parameters n: (int) :

type_trace: (str) ["all"][raytracing across atmos and dms seen] "dm" : raytracing across dms seen only "atmos" : raytracing across atmos only

atmos: (Atmos) :(optional) Atmos object

dms: (Dms) : (optional) Dms object

rst: (int) : (optional) reset before raytracing if rst = 1

slopes_geom()

Compute the geometric slopes in a sutra_wfs object

Parameters nsensor: (int) : wfs number

param t (int) : method (0 or 1)

`sensors.bin2d()`

Returns the input 2D array "array", binned with the binning factor "binfact". The input array X and/or Y dimensions needs not to be a multiple of "binfact"; The final/edge pixels are in effect replicated if needed. This routine prepares the parameters and calls the C routine _bin2d. The input array can be of type long, float or double. Last modified: Dec 15, 2003. Author: F.Rigaut SEE ALSO: _bin2d

Parmeters data_in: (np.ndarray) : data to binned

binfact: (int) : binning factor

`sensors.fft_goodsize()`

find best size for a fft from size s

Parameters s: (long) size

`sensors.init_wfs_geom()`

Compute the geometry of WFSs: valid subaps, positions of the subaps, flux per subap, etc...

Parameters wfs: (Param_wfs) : wfs settings

wfs0: (Param_wfs) : reference wfs settings

n: (int) : index of the wfs (diplay information purpose only)

atmos: (Param_atmos) : atmos settings

tel: (Param_tel) : telescope settings

geom: (Param_geom) : geom settings

target: (Param_target) : target settings

loop: (Param_loop) : loop settings

init: (int) : (optional)

verbose: (int) : (optional) display informations if 0

`sensors.make_lgs_prof1d()`

same as prep_lgs_prof but cpu only. original routine from rico

Parameters p_tel: (Param_tel) : telescope settings

prof: (np.ndarray[dtype=np.float32]) : Na profile intensity, in arbitrary units

h: (np.ndarray[dtype=np.float32]) : altitude, in meters. h MUST be an array with EQUALLY spaced elements.

beam: (float) : size in arcsec of the laser beam

center: (string) : either "image" or "fourier" depending on where the centre should be.

`sensors.noise_cov()`

Compute the diagonal of the noise covariance matrix for a SH WFS (arcsec²) Photon noise: (pi²/2)*(1/Nphotons)*(d/r0)² / (2*pi*d/lambda)² Electronic noise: (pi²/3)*(wfs.noise²/N²photons)*wfs.npix²*(wfs.npix*wfs.pixsize*d/lambda)² / (2*pi*d/lambda)²

Parameters nw: wfs number p_wfs: (Param_wfs) : wfs settings p_atmos: (Param_atmos) : atmos settings p_tel: (Param_tel) : telescope settings

Returns cov : (np.ndarray(ndim=1,dtype=np.float64)) : noise covariance diagonal `sensors.prep_lgs_prof()`

The function returns an image array(double,n,n) of a laser beacon elongated by perpective effect. It is obtaind by convolution of a gaussian of width "lgsWidth" arcseconds, with the line of the sodium profile "prof". The altitude of the profile is the array "h".

parameters nsensors: (int) : wfs index

p_tel: (Param_tel) : telescope settings

prof: (np.ndarray[dtype=np.float32]) : Na profile intensity, in arbitrary units

h: (np.ndarray[dtype=np.float32]) : altitude, in meters. h MUST be an array with EQUALLY spaced elements.

beam: (float) : size in arcsec of the laser beam

center: (string) : either "image" or "fourier" depending on where the centre should be.

Computation of LGS spot from the sodium profile: Everything is done here in 1D, because the Na profile is the result of the convolution of a function P(x,y) = profile(x) . dirac(y) by a gaussian function, for which variables x and y can be split : exp(-(x^2+y^2)/2.s^2) = exp(-x^2/2.s^2) * exp(-y^2/2.s^2) The convolution is (symbol $ denotes integral) C(X,Y) = $$ exp(-x^2/2.s^2) * exp(-y^2/2.s^2) * profile(x-X) * dirac(y-Y) dx dy First one performs the integration along y C(X,Y) = exp(-Y^2/2.s^2) $ exp(-x^2/2.s^2) * profile(x-X) dx which shows that the profile can be computed by - convolving the 1-D profile - multiplying it in the 2nd dimension by a gaussian function

If one has to undersample the inital profile, then some structures may be "lost". In this case, it's better to try to "save" those structures by re-sampling the integral of the profile, and then derivating it afterwards. Now, if the initial profile is a coarse one, and that one has to oversample it, then a simple re-sampling of the profile is adequate.

`sensors.type_present()`

Check the present types in a list

Parameters liste: (list of str) : list of types

pyr: (int) : set to 1 if the list contains "pyr" (0 else)

roof: (int): set to 1 if the list contains "roof" (0 else)

sh: (int) : set to 1 if the list contains "sh" (0 else)

geo: (int) : set to 1 if the list contains "geo" (0 else)

return 1 if the wfs type is present (0 else)

`sensors.wfs_init()`

Create and initialise a Sensors object

Parameters wfs: (list of Param_wfs) : wfs settings

p_atmos: (Param_atmos) : atmos settings

p_tel: (Param_tel) : telescope settings

p_geom: (Param_geom) : geom settings

p_target: (Param_target) : target settings

p_loop: (Param_loop) : loop settings

comm_size: (int) : communicator size

rank: (int) : process rank

dm: (list of Param_dm) : (optional) dms settings

`sensors.wheremax()`
return the index of the maximum value of the list

Parameters **liste** – (list of values) : values to get the index of the maximum from

1.4 RTC

1.4.1 rtc module

class `rtc.Rtc`
Bases: `object`

add_Controller()
Add a controller in the sutra_controller vector of the RTC on the GPU

Parameters nactu: (int) : number of actuators

delay: (float) : loop delay

type_control: (str) : controller's type

dms: (Dms) : sutra_dms object (GPU)

type_dmseen: (char**) : dms indices controled by the controller

alt: (np.ndarray[ndim=1,dtype=np.float32_t]) : altitudes of the dms seen

ndm: (int) : number of dms controled

Nphi: (long) : number of pixels in the pupil (used in geo controler case only)

add_centroider()
Add a centroider in the sutra_centroiders vector of the RTC on the GPU

Parameters sensor: (Sensors) : sutra_sensors object (GPU)

nwfs : (long) : number of wfs

nvalid: (long) : number of valid subaps

type_centro: (str) : centroider's type

offset: (float) :

scale: (float) :

applycontrol()
Compute the DMs shapes from the commands computed in a sutra_controller_object. From the command vector, it computes the voltage command (adding pertrubation voltages, taking delay into account) and then apply it to the dms

Parameters ncontro: (int) : controller index

buildcmat()
Compute the command matrix in a sutra_controller_ls object

Parameters ncontro: (int) : controller index

nfilt: (int) : number of modes to filter

filt_tt: (int) : (optional) flag to filter TT

buildcmatmv()
Compute the command matrix in a sutra_controller_mv object

Parameters ncontro: (int) : controller index

cond: (float) : conditioning factor for the Cmm inversion

docentroids()

Compute the centroids with sutra_controller #ncontrol object

Parameters ncontrol: (optional) controller's index

docentroids_geom()

Compute the geometric centroids with sutra_controller #ncontrol object

Parameters ncontrol: (optional) controller's index

docontrol()

Compute the command to apply on the DMs on a sutra_controller object

Parameters ncontro: (int) : controller index

docontrol_geo()

Compute the command to apply on the DMs on a sutra_controller_geo object

Parameters ncontro: (int) : controller index

doimat()

Compute the interaction matrix

Parameters ncontro: (int) : controller index

g_dms: (Dms) : Dms object

doimat_geom()

Compute the interaction matrix by using a geometric centroiding method

Parameters ncontro: (int) : controller index

g_dms: (Dms) : Dms object

geom: (int) : type of geometric method (0 or 1)

getCenbuff()

Return the centroids buffer from a sutra_controller_ls object. This buffer contains centroids from iteration i-delay to current iteration :parameters:

ncontro: (int) : controller index

Returns data : (np.ndarray[ndim=2,dtype=np.float32_t]) : centroids buffer

getCentroids()

Return the centroids computed by the sutra_rtc object

Parameters ncontrol: (int) : controller's index

Returns data : (np.ndarray[ndim=1,dtype=np.float32_t]) : centroids (arcsec)

getCmmEigenvals()

Return the eigen values of the Cmm decomposition in a sutra_controller_mv object :parameters:

ncontro: (int) : controller index

Returns eigenvals : (np.ndarray[ndim=1,dtype=np.float32_t]) : eigenvalues

getCom()

Return the command vector from a sutra_controller object

Parameters ncontro: (int) : controller index

Returns data : (np.ndarray[ndim=1,dtype=np.float32_t]) : command vector

getEigenvals()
Return the eigen values of the imat decomposition in a sutra_controller object

Parameters ncontro: (int) : controller index

Returns eigenvals : (np.ndarray[ndim=1,dtype=np.float32_t]) : eigenvalues

getErr()
Return the command increment (cmat*slopes) from a sutra_controller_ls object

Parameters ncontro: (int) : controller index

Returns data : (np.ndarray[ndim=1,dtype=np.float32_t]) : command increment

getU()
Return the eigen modes matrix of the imat decomposition from a sutra_controller_ls object

Parameters ncontro: (int) : controller index

Returns U : (np.ndarray[ndim=2,dtype=np.float32_t]) : eigen modes matrix

getVoltage()
Return the voltage vector that will be effectively applied to the DMs

Parameters ncontro: (int) : controller index

Returns data : (np.ndarray[ndim=1,dtype=np.float32_t]) : voltage vector

get_cmat()
Return the command matrix from a sutra_controller object

Parameters ncontro: (int) : controller index

Returns cmat : (np.ndarray[ndim=2,dtype=np.float32_t]) : command matrix

get_cmm()
Return the Cmm matrix from a sutra_controller_mv object

Parameters ncontro: (int) : controller index

Returns cmm : (np.ndarray[ndim=2,dtype=np.float32_t]) : Cmm matrix

get_cphim()
Return the Cphim matrix from a sutra_controller_mv object

:parameters; ncontro: (int) : controller index

Returns cphim : (np.ndarray[ndim=2,dtype=np.float32_t]) : Cphim matrix

get_imat()
Return the interaction matrix of a sutra_controller object

Parameters ncontro: (int) : controller index

Returns imat : (np.ndarray[ndim=2,dtype=np.float32_t]) : interaction matrix

get_mgain()
Return modal gains from sutra_controller

Parameters ncontro: (int) : controller index

Returns mgain : (np.ndarray[ndim=1,dtype=np.float32_t]) : modal gains

getcentroids()

Return the centroids computed by the sutra_rtc object If ncontrol <= d_control.size, return rtc.d_centroids Else, compute centroids from wfs[nwfs] with centroider[ncontrol]

Parameters ncontrol: (int) : controller's index

g_wfs: (Sensors) : (optional) sutra_sensors object

nwfs: (int) : (optional) number of wfs

Returns data : (np.ndarray[ndim=1,dtype=np.float32_t]) : centroids (arcsec)

getolmeas()

Return the reconstructed open-loop measurement from a sutra_controller_mv object

Parameters ncontro: (int) : controller index

Returns data : (np.ndarray[ndim=1,dtype=np.float32_t]) : reconstructed open-loop

imat_svd()

Compute the singular value decomposition of the interaction matrix

Parameters **ncontro** – controller index

init_modalOpti()

Initialize the modal optimization controller : compute the slopes-to-modes matrix and the transfer functions

Parameters ncontro: (int) : controller index

nmodes: (int) : number of modes

nrec: (int) : number of recorded open slopes measurements

M2V: (np.ndarray[ndim=2,dtype=np.float32_t]) : modes-to-volt matrix

gmin: (float) : minimum gain for modal optimization

gmax: (float) : maximum gain for modal optimization

ngain: (int) : Number of tested gains

Fs: (float) : sampling frequency

init_proj()

Initialize the projection matrix for sutra_controller_geo object. The projection matrix is (IFt.IF)**(-1) * IFt where IF is the DMs influence functions matrix

Parameters ncontro: (int) : controller index

dms: (Dms) : Dms object

indx_dm: (np.ndarray[ndim=1,dtype=np.int32_t]) : indices of where(pup) on DM screen

unitpervolt: (np.ndarray[ndim=1,dtype=np.float32_t]) : unitpervolt DM parameter

indx_pup: (np.ndarray[ndim=1,dtype=np.int32_t]) : indices of where(pup) on ipupil screen

loadOpenLoop()

Load an array of recoded open-loop measurements for modal optimization

Parameters ncontro: (int) : controller index

ol_slopes: (np.ndarray[ndim=2, dtype=np.float32_t]) : open-loop slopes

loadnoisemat()

Load the noise vector on a sutra_controller_mv object

Parameters ncontro: (int) : controller index

N: (np.ndarray[ndim=1,dtype=np.float32_t]) : noise vector

modalControlOptimization()

Compute the command matrix with modal control optimization

Parameter ncontro: controller index

rmcontrol()

Remove a controller

sensors_compslopes()

Compute the slopes in a sutra_wfs object. This function is equivalent to docentroids() but the centroids are stored in the sutra_wfs object instead of the sutra_rtc object

Parameters ncentro: (int) : centroider index

sensors_initbcube()

Initialize npix in the sutra_centroider_corr object (useless ?)

Parameters ncentro: (int) : centroider's index

sensors_initcorr()

Initialize sutra_centroider_corr oblect

Parameters ncentro: (int) : centroider's index

w: (np.ndarray[ndim=1,dtype=np.float32_t]) : weight

corr_norm: (np.ndarray[ndim=2,dtype=np.float32_t]) :

sizex: (int) :

sizey: (int) :

interpmat: ([ndim=2,dtype=np.float32_t]) :

sensors_initweights()

Load the weight array in sutra_centroider_wcog object

Parameters ncentro: (int) : centroider's index

w: (np.ndarray[ndim=2, dtype=np.float32_t]) : weight

setCom()

Set the command vector of a sutra_controller object to comvec

Parameters ncontro: (int) : controller index

setEigenvals()

Set the eigen values of the imat decomposition in a sutra_controller_ls object

Parameters ncontro: (int) : controller index

eigenvals: (np.ndarray[ndim=1,dtype=np.float32_t]) : eigen values

setU()

Set the eigen modes matrix of the imat decomposition in a sutra_controller_ls object

Parameters ncontro: (int) : controller index

U: (np.ndarray[ndim=2,dtype=np.float32_t]) : eigen modes matrix

set_cmat()

Set the command matrix on a sutra_controller object

Parameters ncontro: (int) : controller index

data: (np.ndarray[ndim=2,dtype=np.float32_t]) : command matrix to use

set_cmm()
Set the Cmm matrix on a sutra_controller_mv object

Parameters ncontro: (int) : controller index

data: (np.ndarray[ndim=2,dtype=np.float32_t]) : Cmm matrix

set_decayFactor()
Set the decay factor on a sutra_controller_generic object

Parameters ncontro: (int) : controller index

decay: (np.ndarray[ndim=1,dtype=np.float32_t]) : ask to Rico

set_gain()
Set the loop gain in sutra_controller object

Parameters ncontro: (int) : controller index

gain: (float) : loop gain

set_imat()
Set the interaction matrix on a sutra_controller object

Parameters ncontro: (int) : controller index

data: (np.ndarray[ndim=2,dtype=np.float32_t]) : interaction matrix to use

set_matE()
Set the matrix E on a sutra_controller_generic object

Parameters ncontro: (int) : controller index

matE: (np.ndarray[ndim=2,dtype=np.float32_t]) : ask to Rico

set_mgain()
Set modal gains in sutra_controller object

Parameters ncontro: (int) : controller index

mgain: (np.ndarray[ndim=1,dtype=np.float32_t]) : modal gains

setnmax()
set the number of brightest pixels to consider for bpcog centroider

Parameters ncentro: (int) : centroider's index

nmax: (int) : number of brightest pixels

setthresh()
set threshold for the centroider #ncentro

Parameters ncentro: (int) : centroider's index

thresh: (float) : threshold

rtc.**cmat_init**()
Compute the command matrix on the GPU

Parameters ncontro: (int) : g_rtc: (Rtc) : p_rtc: (Param_rtc) : rtc settings p_wfs: (list of Param_wfs) : wfs settings p_tel : (Param_tel) : telescope settings clean: (int) : (optional) clean datafiles (imat, U, eigenv) simul_name: (str) : (optional) simulation's name, use for data files' path

`rtc.`**`compute_KL2V`**`()`
Compute the Karhunen-Loeve to Volt matrix (transfer matrix between the KL space and volt space for a pzt dm)

Parameters p_dms: (list of Param_dm) : dms settings

controller: (Param_controller) : controller settings

`rtc.`**`correct_dm`**`()`
Correct the geometry of the DMs using the imat (filter unseen actuators)

Parameters p_dms: (list of Param_dm) : dms settings

g_dms: (Dms) : Dms object

p_control: (Param_controller) : controller settings

p_geom: (Param_geom) : geom settings

imat: (np.ndarray) : interaction matrix

simul_name: (str) : simulation's name, use for data files' path

`rtc.`**`create_interp_mat`**`()`
TODO doc

Parameters dimx: (int) :

dimy: (int) :

`rtc.`**`create_nact_geom`**`()`
Compute the DM coupling matrix :param:

p_dms : (list of Param_dm) : dms parameters ndm : (int) : dm number

Returns Nact : (np.array(dtype=np.float64)) : the DM coupling matrix

`rtc.`**`create_piston_filter`**`()`

`rtc.`**`doTomoMatrices`**`()`
Compute Cmm and Cphim matrices for the MV controller on GPU

Parameters g_wfs: (Sensors) :

p_wfs: (list of Param_wfs) : wfs settings

g_dms: (Dms) :

p_dms: (list of Param_dms) : dms settings

p_geom: (Param_geom) : geom settings

p_atmos: (Param_atmos) : atmos settings

g_atmos: (Atmos) :

p_tel: (Param_tel) : telescope settings

`rtc.`**`get_r0`**`()`
Compute r0 at lambda2 from r0 value at lambda1

Parameters r0_at_lambda1: (float) : r0 value at lambda1

lambda1: (float) : lambda1

lambda2: (float) : lambda2

`rtc.`**`imat_geom`**`()`
Compute the interaction matrix with a geometric method

Parameters g_wfs: (Sensors) : Sensors object

p_wfs: (list of Param_wfs) : wfs settings

p_control: (Param_controller) : controller settings

g_dms: (Dms) : Dms object

p_dms: (list of Param_dm) : dms settings

meth: (int) : (optional) method type (0 or 1)

`rtc.`**`imat_init`**`()`

Initialize and compute the interaction matrix on the GPU

Parameters ncontro: (int) : controller's index

g_rtc: (Rtc) : Rtc object

p_rtc: (Param_rtc) : rtc settings

g_dms: (Dms) : Dms object

g_wfs: (Sensors) : Sensors object

p_wfs: (list of Param_wfs) : wfs settings

p_tel: (Param_tel) : telescope settings

clean: (int) : (optional) : clean datafiles (imat, U, eigenv)

simul_name: (str) : (optional) simulation's name, use for data files' path

`rtc.`**`manual_imat`**`()`

Compute the interaction matrix 'manually', ie without sutra_rtc doimat method

Parameters g_rtc: (Rtc) : Rtc object

g_wfs: (Sensors) : Sensors object

g_dms: (Dms) : Dms object

p_dms: (list of Param_dm) : dm settings

`rtc.`**`openLoopSlp`**`()`

Return a set of recorded open-loop slopes, usefull for modal control optimization

Parameters g_atm: (Atmos) : Atmos object

g_rtc: (Rtc) : Rtc object

nrec: (int) : number of samples to record

ncontro: (int) : controller's index

g_wfs: (Sensors) : Sensors object

p_wfs: (list of Param_wfs) : wfs settings

p_tar: (Param_target) : target settings

g_tar: (Target) : Target object

`rtc.`**`rtc_init`**`()`

Initialize all the sutra_rtc objects : centroiders and controllers

Parameters g_wfs: (Sensors) : Sensors object

p_wfs: (list of Param_wfs) : wfs settings g_dms: (Dms) : Dms object

p_dms: (list of Param_dms) : dms settings

p_geom: (Param_geom) : geom settings

p_atmos: (Param_atmos) : atmos settings

g_atmos: (Atmos) : Atmos object

p_tel: (Param_tel) : telescope settings

p_loop: (Param_loop) : loop settings

p_tar: (Param_target) : (optional) target settings

clean: (int) : (optional) clean datafiles (imat, U, eigenv)

brama: (int) : (optional) not implemented yet

doimat: (int) : (optional) force imat computation

simul_name: (str) : (optional) simulation's name, use for path to save data (imat, U...)

Returns Rtc : (Rtc) : Rtc object

`rtc.`**`selectDMforLayers`**`()`

For each atmos layer, select the DM which have to handle it in the Cphim computation for MV controller :parameters:

ncontro : (int) : controller number p_atmos : (Param_atmos) : atmos parameters p_rtc : (Param_rtc) : rtc parameters p_dms :(list of Param_dm) : dms parameters

Returns indlayersDM : (np.array(dtype=np.int32)) : for each atmos layer, the Dm number corresponding to it

1.5 Param

1.5.1 `param` module

class `param.`**`Param_atmos`**

Bases: `object`

`L0`

L0 per layers in meters.

`alt`

altitudes of each layer.

`deltax`

x translation speed (in pix / iteration) for each layer.

`deltay`

y translation speed (in pix / iteration) for each layer.

`dim_screens`

linear size of phase screens.

`frac`

fraction of r0 for each layer.

nscreens
number of turbulent layers.

pupixsize
pupil pixel size (in meters).

r0
global r0.

seeds

set_L0()
Set the L0 per layers

Parameters **l** – (lit of float) : L0 for each layers

set_alt()
Set the altitudes of each layer

Parameters **l** – (lit of float) : altitudes

set_deltax()
Set the translation speed on axis x for each layer

Parameters **l** – (lit of float) : translation speed

set_deltay()
Set the translation speed on axis y for each layer

Parameters **l** – (lit of float) : translation speed

set_dim_screens()
Set the size of the phase screens

Parameters **l** – (lit of float) : phase screens sizes

set_frac()
Set the fraction of r0 for each layers

Parameters **l** – (lit of float) : fraction of r0

set_nscreens()
Set the number of turbulent layers

Parameters **n** – (long) number of screens.

set_pupixsize()
Set the pupil pixel size

Parameters **xsize** – (float) : pupil pixel size

set_r0()
Set the global r0

Parameters **r** – (float) : global r0

set_seeds()
Set the seed for each layer

Parameters **l** – (lit of float) : seed

set_winddir()
Set the wind direction for each layer

Parameters **l** – (lit of float) : wind directions

set_windspeed()
Set the the wind speed for each layer

Parameters **l** – (lit of float) : wind speeds

winddir
wind directions of each layer.

windspeed
wind speeds of each layer.

class param.**Param_centroider**
Bases: object

interpmat
optional reference function(s) used for corr centroiding

nmax
number of brightest pixels

nwfs
index of wfs in y_wfs structure on which we want to do centroiding

set_nmax()
Set the number of brightest pixels to use for bpcog

Parameters n: (int) : number of brightest pixels

set_nwfs()
Set the index of wfs

Parameters **n** – (int) : index of wfs

set_sizex()
Set sizex parameters for corr centroider (interp_mat size)

Parameters s: (long) : x size

set_sizey()
Set sizey parameters for corr centroider (interp_mat size)

Parameters s: (long) : y size

set_thresh()
Set the threshold for tcog

Parameters t: (float) : threshold

set_type()
Set the centroider type :param t: (str) : centroider type

set_type_fct()
Set the type of ref function

Parameters **f** – (str) : type of ref function

set_weights()
Set the weights to use with wcog or corr

Parameters w: (np.ndarray[ndim=3 ,dtype=np.float32_t]) : weights

set_width()
Set the width of the Gaussian

Parameters **w** – (float) : width of the gaussian

sizex
x-size for inter mat (corr)

sizey
x-size for inter mat (corr)

thresh
Threshold

type_centro
type of centroiding cog, tcog, bpcog, wcog, corr

type_fct
type of ref function gauss, file, model

weights
optional reference function(s) used for centroiding

width
width of the Gaussian

class `param.`**Param_controller**
Bases: `object`

TTcond
tiptilt condition number for cmat filtering with mv controller

cmat
full control matrix

cured_ndivs
subdivision levels in cured

delay
loop delay [frames]

gain
loop gain

gmax
Maximum gain for modal optimization

gmin
Minimum gain for modal optimization

imat
full interaction matrix

maxcond
max condition number

modopti
Flag for modal optimization

nactu
number of controled actuator per dm

ndm
index of dms in controller

ngain
Number of tested gains

nkl

nmodes
Number of modes for M2V matrix (modal optimization)

nrec
Number of sample of open loop slopes for modal optimization computation

nvalid
number of valid subaps per wfs

nwfs
index of wfss in controller

set_TTcond()
Set the tiptilt condition number for cmat filtering with mv controller

:param : (float) : tiptilt condition number

set_cmat()
Set the full control matrix

Parameters **cmat** – (np.ndarray[ndim=2,dtype=np.float32_t]) : full control matrix

set_cured_ndivs()
Set the subdivision levels in cured

Parameters **c** – (long) : subdivision levels in cured

set_delay()
Set the loop delay expressed in frames

Parameters d: (float) :delay [frames]

set_gain()
Set the loop gain

Parameters g: (float) : loop gain

set_gmax()
Set the maximum gain for modal optimization

Parameters **g** – (flaot) : maximum gain for modal optimization

set_gmin()
Set the minimum gain for modal optimization

Parameters **g** – (float) : minimum gain for modal optimization

set_imat()
Set the full interaction matrix

Parameters **imat** – (np.ndarray[ndim=2,dtype=np.float32_t]) : full interaction matrix

set_maxcond()
Set the max condition number

:param : (float) : max condition number

set_modopti()
Set the flag for modal optimization

Parameters **m** – (int) : flag for modal optimization

set_nactu()
Set the number of controled actuator

Parameters **l** – (list of int) : number of controled actuator per dm

set_ndm()
Set the indices of dms

Parameters **l** – (list of int) : indices of dms

set_ngain()
Set the number of tested gains

Parameters **n** – (int) : number of tested gains

set_nkl()
Set the number of KL modes used for computation of covmat in case of minimum variance controller

Parameters **n** – (long) : number of KL modes

set_nmodes()
Set the number of modes for M2V matrix (modal optimization)

Parameters **n** – (int) : number of modes

set_nrec()
Set the number of sample of open loop slopes for modal optimization computation

Parameters **n** – (int) : number of sample

set_nvalid()
Set the number of valid subaps

Parameters **l** – (list of int) : number of valid subaps per wfs

set_nwfs()
Set the indices of wfs

Parameters **l** – (list of int) : indices of wfs

set_type()

type_control
type of controller

class param.**Param_dm**
Bases: object

_com
current command

_i1

_influ
influence functions

_influkernel

_influpos

_influsize
total number of actuators

_influstart

_j1

_klbas
np.ndarray to a kl struct

_n1
position of leftmost pixel in largest support

_n2
position of rightmost pixel in largest support

_ninflu
Influence functions

_ntotact
total number of actuators

_pitch
inter-actuator space in pixels

_pupil
pupil mask for this dm

_puppixoffset

_xpos
x positions of influ functions

_ypos
y positions of influ functions

alt
conjugaison altitude (im m)

coupling
actuators coupling (<0.3)

hyst
actuators hysteresis (<1.)

margin

nact
number of actuators in the diameter

nkl
number of kl modes

pupoffset

2.

push4imat
nominal voltage for imat

set_alt()
set the conjugaison altitude

Parameters **a** – (float) : conjugaison altitude (im m)

set_coupling()
set the actuators coupling

Parameters **c** – (float) : actuators coupling (<0.3)

set_i1()
TODO doc

Parameters **i1** – (np.ndarray[ndim=1,dtype=np.int32_t]) :

set_influ()
Set the influence function

Parameters **influ** – (np.ndarray[ndim=3,dtype=np.float32_t]) : influence function

set_j1()
TODO doc

Parameters **j1** – (np.ndarray[ndim=1,dtype=np.int32_t]) :

set_nact()
set the number of actuator

Parameters **n** – (long) : number of actuators in the diameter

set_ntotact()
set the total number of actuators

Parameters **n** – (long) : total number of actuators

set_push4imat()
set the nominal voltage for imat

Parameters **p** – (float) : nominal voltage for imat

set_thresh()
set the threshold on response for selection

Parameters **t** – (float) : threshold on response for selection (<1)

set_type()
set the dm type

Parameters **t** – (str) : type of dm

set_unitpervolt()
set the Influence function sensitivity

Parameters **u** – (float) : Influence function sensitivity in unit/volt

set_xpos()
Set the x positions of influ functions

Parameters **xpos** – (np.ndarray[ndim=1,dtype=np.float32_t]) : x positions of influ functions

set_ypos()
Set the y positions of influ functions

Parameters **ypos** – (np.ndarray[ndim=1,dtype=np.float32_t]) : y positions of influ functions

thresh
threshold on response for selection (<1)

type_dm
type of dm

unitpervolt
Influence function sensitivity in unit/volt. Optional [0.01] Stackarray: mic/volt, Tip-tilt: arcsec/volt.

class param.**Param_geom**
Bases: object

_mpupil

_n

_n1

_n2

_p1

_p2

cent
central point of the simulation.

geom_init()
Initialize the system geometry

Parameters tel: (Param_tel) : telescope settings

pupdiam: (long) : linear size of total pupil

apod: (int) : apodizer

get_ipupil()
return the full pupil support

get_mpupil()
return the padded pupil

get_n()
Return the linear size of the medium pupil

get_n1()
Return the min(x,y) for valid points for the total pupil

get_n2()
Return the max(x,y) for valid points for the total pupil

get_p1()
Return the min(x,y) for valid points for the medium pupil

get_p2()
Return the max(x,y) for valid points for the medium pupil

get_spupil()
return the small pupil

pupdiam
linear size of total pupil (in pixels).

set_cent()
Set the central point of the simulation

Parameters **c** – (float) : central point of the simulation.

set_pupdiam()
Set the linear size of total pupil

Parameters **p** – (long) : linear size of total pupil (in pixels).

set_ssize()
Set linear size of full image

Parameters **s** – (long) : linear size of full image (in pixels).

set_zenithangle()
Set observations zenith angle

Parameters **z** – (float) : observations zenith angle (in deg).

ssize
linear size of full image (in pixels).

zenithangle
observations zenith angle (in deg).

class param.**Param_loop**
Bases: object

ittime
iteration time (in sec)

niter
number of iterations

set_ittime()
Set iteration time

Parameters t: (float) :iteration time

set_niter()
Set the number of iteration

Parameters n: (long) : number of iteration

class param.**Param_rtc**
Bases: object

centroiders

controllers

nwfs

set_centroiders()
Set the centroiders

Parameters **l** – (list of Param_centroider) : centroiders settings

set_controllers()
Set the controller

Parameters **l** – (list of Param_controller) : controllers settings

set_nwfs()
Set the number of wfs

Parameters **n** – (int) number of wfs

class param.**Param_target**
Bases: object

Lambda
observation wavelength for each target

apod
boolean for apodizer

dms_seen
index of dms seen by the target

mag
magnitude for each target

ntargets
number of targets

set_Lambda()
Set the observation wavelength

Parameters **l** – (list of float) : observation wavelength for each target

set_apod()
Tells if the apodizer is used

The apodizer is used if a is not 0 :param a: (int) boolean for apodizer

set_mag()
set the magnitude

Parameters **l** – (list of float) : magnitude for each target

set_nTargets()
Set the number of targets

Parameters **n** – (int) : number of targets

set_xpos()
Set the x positions on sky (in arcsec)

Parameters **l** – (list of float) : x positions on sky for each target

set_ypos()
Set the y positions on sky (in arcsec)

Parameters **l** – (list of float) : y positions on sky for each target

xpos
x positions on sky (in arcsec) for each target

ypos
y positions on sky (in arcsec) for each target

zerop
target flux for magnitude 0

class param.**Param_tel**
Bases: object

cobs
central obstruction ratio.

diam
telescope diameter (in meters).

nbrmissing
number of missing segments for EELT pupil (max is 20).

pupangle
rotation angle of pupil.

referr
std of reflectivity errors for EELT segments (fraction).

set_cobs()
set the central obstruction ratio

Parameters **c** – (float) : central obstruction ratio

set_diam()
set the telescope diameter

Parameters **d** – (float) : telescope diameter (in meters)

set_nbrmissing()
set the number of missing segments for EELT pupil

Parameters **nb** – (long) : number of missing segments for EELT pupil (max is 20)

set_pupangle()
set the rotation angle of pupil

Parameters **p** – (float) : rotation angle of pupil

set_referr()
set the std of reflectivity errors for EELT segments

Parameters **ref** – (float) : std of reflectivity errors for EELT segments (fraction)

set_spiders_type()
set the secondary supports type

Parameters **spider** – (str) : secondary supports type

set_t_spiders()
set the secondary supports ratio

Parameters **spider** – (float) : secondary supports ratio

set_type_ap()
set the EELT aperture type

Parameters **t** – (str) : EELT aperture type

spiders_type
secondary supports type: “four” or “six”.

t_spiders
secondary supports ratio.

type_ap
EELT aperture type: “Nominal”, “BP1”, “BP3”, “BP5” (for back-up plan with 1, 3, or 5 missing annulus).

class param.**Param_wfs**
Bases: object

Lambda
observation wavelength (in μm) for a subap.

_Nfft
array size for fft for a subap (in pixel)

_Ntot
total size of hr image for a subap (in pixel)

_altna
corresponding altitude

_azimuth
angles of rotation for each spot

_beam
1d beam function

_binmap
(int*) array of pixels transform from full FoV hr images to binned images

_fluxPerSub
fraction of nphotons per subap

_ftbeam
1d beam function fft

_halfxy
(float*) phase offset for 1/2 pixel shift in (x,y)

_hrmap
(int*) array of pixels transform from minimal FoV image to (in case type is sh or geo)

_istart
(int*) x start indexes for cutting phase screens

_isvalid
(int*) array of 0/1 for valid subaps

_jstart
(int*) y start indexes for cutting phase screens

_lgskern
lgs kernels for each subap

_nphotons
number of photons per subap

_nrebin
rebin factor from hr to binned image for a subap

_nvalid
number of valid subaps

_pdiam
pupil diam for a subap (in pixel)

_phasemap
(int*) array of pixels transform from phase screen into subaps phase screens

_prof1d
hr profile

_profcum
hr profile cumulated

_profna
sodium profile

_pyr_cx

_pyr_cy

_pyr_offsets

_qpixsize
quantum pixel size for the simulation

_subapd
subap diameter (m)

_submask
(float*) fieldstop for each subap

_validsubsx
(int*) indices of valid subaps along axis x

_validsubsy
(int*) indices of valid subaps along axis y

atmos_seen
1 if the WFS sees the atmosphere layers

beamsize
laser beam fwhm on-sky (in arcsec).

dms_seen
index of dms seen by the WFS

fracsub
minimal illumination fraction for valid subaps.

fssize
size of field stop in arcsec.

fstop
size of field stop in arcsec.

gsalt
altitude of guide star (in m) 0 if ngs.

gsmag
magnitude of guide star.

kernel

laserpower
laser power in W.

lgsreturnperwatt
return per watt factor (high season : 10 ph/cm2/s/W).

lltx
x position (in meters) of llt.

llty
y position (in meters) of llt.

noise
desired noise : < 0 = no noise / 0 = photon only / > 0 photon + ron.

npix
number of pixels per subap.

nxsub
linear number of subaps.

openloop
1 if in "open-loop" mode (i.e. does not see dm).

optthroughput
wfs global throughput.

pixsize
pixel size (in arcsec) for a subap.

proftype
type of sodium profile "gauss", "exp", etc ...

pyr_ampl
pyramid wfs modulation amplitude radius [arcsec].

pyr_loc
Location of modulation, before/after the field stop. valid value are "before" or "after" (default "after").

pyr_npts
total number of point along modulation circle [unitless].

pyrtype
Type of pyramid, either 0 for "Pyramid" or 1 for "RoofPrism".

set_Lambda()
Set the observation wavelength

Parameters **L** – (float) : observation wavelength (in μm) for a subap

set_altna()
Set the corresponding altitude

Parameters **a** – (np.ndarray[ndim=1,dtype=np.float32]) : corresponding altitude

set_atmos_seen()
Tells if the wfs sees the atmosphere layers

Parameters **i** – (int) :1 if the WFS sees the atmosphere layers

set_beamsize()
Set the laser beam fwhm on-sky

Parameters **b** – (float) : laser beam fwhm on-sky (in arcsec)

set_dms_seen()
Set the index of dms seen by the WFS

Parameters **dms_seen** – (np.ndarray[ndim=1,dtype=np.int32_t) : index of dms seen by the WFS

set_fracsub()
Set the minimal illumination fraction for valid subaps

Parameters **f** – (float) : minimal illumination fraction for valid subaps

set_fssize()
Set the size of field stop

Parameters **f** – (float) : size of field stop in arcsec

set_fstop()
Set the size of field stop

Parameters **f** – (str) : size of field stop in arcsec

set_gsalt()
Set the altitude of guide star

Parameters **g** – (float) : altitude of guide star (in m) 0 if ngs

set_gsmag()
Set the magnitude of guide star

Parameters **g** – (float) : magnitude of guide star

set_kernel()
Set the attribute kernel

Parameters **k** – (float) :

set_laserpower()
Set the laser power

Parameters **l** – (float) : laser power in W

set_lgsreturnperwatt()
Set the return per watt factor

Parameters **lpw** – (float) : return per watt factor (high season : 10 ph/cm2/s/W)

set_lltx()
Set the x position of llt

Parameters **l** – (float) : x position (in meters) of llt

set_llty()
Set the y position of llt

Parameters **l** – (float) : y position (in meters) of llt

set_noise()
Set the desired noise

Parameters **n** – (float) : desired noise : < 0 = no noise / 0 = photon only / > 0 photon + ron

set_npix()
Set the number of pixels per subap

Parameters **n** – (long) : number of pixels per subap

set_nxsub()
Set the linear number of subaps

Parameters **n** – (long) : linear number of subaps

set_openloop()
Set the loop state (open or closed)

Parameters **o** – (long) : 1 if in "open-loop" mode (i.e. does not see dm)

set_optthroughput()
Set the wfs global throughput

Parameters **o** – (float) : wfs global throughput

set_pixsize()
Set the pixel size

Parameters **p** – (float) : pixel size (in arcsec) for a subap

set_profna()
Set the sodium profile

Parameters **p** – (np.ndarray[ndim=1,dtype=np.float32]) : sodium profile

set_proftype()
Set the type of sodium profile

Parameters **p** – (str) : type of sodium profile "gauss", "exp", etc ...

set_pyr_ampl()
Set the pyramid wfs modulation amplitude radius

Parameters **p** – (float) : pyramid wfs modulation amplitude radius (in arsec)

set_pyr_loc()
Set the location of modulation

Parameters **p** – (str) : location of modulation, before/after the field stop. valid value are "before" or "after" (default "after")

set_pyr_npts()
Set the total number of point along modulation circle

Parameters **p** – (long) : total number of point along modulation circle

set_pyrtype()
Set the type of pyramid,

Parameters **p** – (str) : type of pyramid, either 0 for “Pyramid” or 1 for “RoofPrism”

set_type()
Set the type of wfs

Parameters **t** – (str) : type of wfs (“sh” or “pyr”)

set_xpos()
Set the guide star x position on sky

Parameters **x** – (float) : guide star x position on sky (in arcsec)

set_ypos()
Set the guide star y position on sky

Parameters **y** – (float) : guide star y position on sky (in arcsec)

set_zerop()
Set the detector zero point

Parameters **z** – (float) : detector zero point

type_wfs
type of wfs : “sh” or “pyr”.

xpos
guide star x position on sky (in arcsec).

ypos
guide star x position on sky (in arcsec).

zerop
detector zero point.

param.**indices**()
DOCUMENT indices(dim) Return a dimxdimx2 array. First plane is the X indices of the pixels in the dimxdim array. Second plane contains the Y indices. Inspired by the Python scipy routine of the same name. New (June 12 2002): dim can either be : - a single number N (e.g. 128) in which case the returned array are

square (NxN)

- a Yorick array size, e.g. [#dimension,N1,N2], in which case the returned array are N1xN2
- a vector [N1,N2], same result as previous case

F.Rigaut 2002/04/03 SEE ALSO: span

Parameters dim1: (int) : first dimension dim2: (int) : (optional) second dimension

param.**make_apodizer**()
TODO doc

Parameters (int) : im:

(int) : pupd:

(str) : filename:

(float) : angle:

param.**makegaussian**(*size*, *fwhm*, *xc*, *yc*)
Returns a centered gaussian of specified size and fwhm. norm returns normalized 2d gaussian

Parameters size: (int) :

fwhm: (float) :

xc: (int) : (optional) center position on x axis

yc: (int) : (optional) center position on y axis

norm: (int) : (optional) normalization

`param.rotate()`

Rotates an image of an angle "ang" (in DEGREES).

The center of rotation is cx,cy. A zoom factor can be applied.

(cx,cy) can be omitted :one will assume one rotates around the center of the image. If zoom is not specified, the default value of 1.0 is taken.

Parameters im: (np.ndarray[ndim=3,dtype=np.float32_t]) : array to rotate

ang: (float) : rotation angle (in degrees)

cx: (int) : (optional) rotation center on x axis (default: image center)

cy: (int) : (optional) rotation center on x axis (default: image center)

zoom: (float) : (opional) zoom factor (default =1.0)

`param.rotate3d()`

Rotates an image of an angle "ang" (in DEGREES).

The center of rotation is cx,cy. A zoom factor can be applied.

(cx,cy) can be omitted :one will assume one rotates around the center of the image. If zoom is not specified, the default value of 1.0 is taken.

modif dg : allow to rotate a cube of images with one angle per image

Parameters im: (np.ndarray[ndim=3,dtype=np.float32_t]) : array to rotate

ang: (np.ndarray[ndim=1,dtype=np.float32_t]) : rotation angle (in degrees)

cx: (int) : (optional) rotation center on x axis (default: image center)

cy: (int) : (optional) rotation center on x axis (default: image center)

zoom: (float) : (opional) zoom factor (default =1.0)

1.6 Dms

1.6.1 `dms` module

class `dms.Dms`

Bases: `object`

`add_dm()`

`comp_oneactu()`

Compute the shape of the dm when pushing the nactu actuator

Parameters type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

nactu: (int) : actuator number pushed

ampli: (float) : amplitude

computeKLbasis()

Compute a Karhunen-Loeve basis for the dm:

- compute the phase covariance matrix on the actuators using Kolmogorov
- compute the geometric covariance matrix
- double diagonalisation to obtain KL basis

Parameters type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

xpos: (np.ndarray[ndim=1,dtype=np.float32_t]) : x-position of actuators

ypos: (np.ndarray[ndim=1,dtype=np.float32_t]) : y-position of actuators

indx_pup: (np.ndarray[ndim=1,dtype=np.int32_t]) : indices of where(pup)

dim: (long) : number of where(pup)

norm: (float) : normalization factor

ampli: (float) : amplitude

getComm()

Return the voltage command of the sutra_dm

Parameters type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

Returns data : (np.ndarray(dims=1,dtype=np.float32)) : voltage vector

getInflu()

Return the influence functions of the DM

Parameters type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

Returns data : (np.ndarray(dims=3,dtype=np.float32)) : influence functions

get_KLbasis()

Return the klbasis computed by computeKLbasis

Parameters type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

Returns KLbasis : (np.ndarray(dims=2,dtype=np.float32)) : the KL basis

get_dm()

Return the shape of the dm

Parameters type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

Returns data : (np.ndarray(dims=2,dtype=np.float32)) : DM shape

load_kl()

Load all the arrays computed during the initialization for a kl DM in a sutra_dms object

Parameters alt: (float) : dm conjugaison altitude

rabas: (np.ndarray[ndim=1,dtype=np.float32_t]) : TODO

azbas: (np.ndarray[ndim=1,dtype=np.float32_t]) :

ord: (np.ndarray[ndim=1,dtype=np.int32_t]) :

cr: (np.ndarray[ndim=1,dtype=np.float32_t]) :

cp: (np.ndarray[ndim=1,dtype=np.float32_t]) :

load_pzt()

Load all the arrays computed during the initialization for a pzt DM in a sutra_dms object

Parameters alt: (float) : dm conjugaison altitude

influ: (np.ndarray[ndim=3,dtype=np.float32_t]) : influence functions

influpos: (np.ndarray[ndim=1,dtype=np.int32_t]) : positions of the IF

npoints: (np.ndarray[ndim=1,dtype=np.int32_t]) [for each pixel on the DM screen,] the number of IF which impact on this pixel

istart: (np.ndarray[ndim=1,dtype=np.int32_t]) :

xoff: (np.ndarray[ndim=1,dtype=np.int32_t]) : x-offset

yoff: (np.ndarray[ndim=1,dtype=np.int32_t]) :y-offset

kern: (np.ndarray[ndim=1,dtype=np.float32_t]) : convoltuon kernel

load_tt()

Load all the arrays computed during the initialization for a tt DM in a sutra_dms object

Parameters alt: (float) : dm conjugaison altitude

influ: (np.ndarray[ndim=3,dtype=np.float32_t]) : influence functions

oneactu()

Push on on the nactu actuator of the DM with ampli amplitude and compute the corresponding shape

Parameters type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

nactu: (int) : actuator number

ampli: (float): amplitude

remove_dm()

Remove a dm from a Dms object

Parameters type_dm: (str) : dm type to remove

alt: (float) : dm conjugaison altitude to remove

resetdm()

Reset the shape of the DM to 0

Parameters type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

set_comm()

Set the voltage command on a sutra_dm

type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

comm: (np.ndarray[ndim=1,dtype=np.float32_t]) : voltage vector

`shape_dm()`

Compute the shape of the DM in a sutra_dm object

type_dm: (str) : dm type

alt: (float) : dm conjugaison altitude

`dms.comp_dmgeom()`

Compute the geometry of a DM : positions of actuators and influence functions

Parameters dm: (Param_dm) : dm settings

geom: (Param_geom) : geom settings

`dms.compute_klbasis()`

Compute a Karhunen-Loeve basis for the dm:

- compute the phase covariance matrix on the actuators using Kolmogorov
- compute the geometric covariance matrix
- double diagonalisation to obtain KL basis

Parameters g_dm: (Dms) : Dms object

p_dm: (Param_dm) : dm settings

p_geom: (Param_geom) : geom settings

p_atmos: (Param_atmos) : atmos settings

p_tel: (Param_tel) : telescope settings

`dms.dm_init()`

Create and initialize a Dms object on the gpu

Parameters p_dms: (list of Param_dms) : dms settings

p_wfs: (Param_wfs) : wfs settings

p_geom: (Param_geom) : geom settings

p_tel: (Param_tel) : telescope settings

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

Symbols

A

B

C

D

F

G

H

I

T

U

W

X

Y

Z