tropy Documentation

Release v0.4_devel01

Fabian Senf

Jul 12, 2021

API Reference:

tropy contains a set of tools which were developed at TROPOS to support scientific work.

Main task are:

- input of datasets
- regridding & filtering
- segmentation & optical flow applications
- calculation of derived variables

Tutorial can be found here: <https://tropy-tutorials.readthedocs.io/en/latest/>

CHAPTER 1

Functions Overview

1.1 tropy.analysis_tools

1.1.1 tropy.analysis_tools.derived_variables

calc_lwp	Calculates the liquid water path.

1.1.2 tropy.analysis_tools.grid_and_interpolation

create_interpolation_index	Given georeference of two grids in lon and lat, an index is built to map a field on grid 1 to grid 2.
curve_flow_filter	Smoothing filter depending on isoline curvature.
cutout_cluster	Makes a cutout of a categorial field c for an object of class / number nc.
cutout_field4box	Cuts out a field based on center pix index and box size.
cutout_fields	Cuts out a field or a list of similar fields given row and column slices.
get_index	Given a point p the nearest grid point is returned.
i2iset	Returns an index set for a n-dim numpy array given an index i which points to a position within the same but flattened array.
interpolate_field_to_hor_pos	Given a sequence of points (or just one) a field (2d or 3d) given a grid is interpolated to the new positions.
ldiff	Calculates difference of a field at adjacent levels.
ll2xy	Transformation between longitude and latitude and local Cartesian coordinates in west-east direction (x) and south-north direction (y).
ll2xyc	Applies a centered version of ll2xy.
lmean	Calculates the layer mean of a field.

Continued on next page

Table 2 – continued from previous page

make_add_edge	Adds egde region to 2d data field.
make_index_set	This simple routine makes an 2d index set for given row and column number of a 2d field.
make_vert_cut	Makes a horizontal - vertical cut through a 3d field.
mid2edge_gridvector	Converts an mid-point based grid vector into an edge-based grid vector.
region2slice	Given a region of ((lon1,lon2),(lat1, lat2)) the routine provides a slicing ((row1, row2), (col1, col2)) than accomodates the region.
remap_field	Do nearest nearbor remapping on a simple equi-distant local cartesian coordiante grid.
simple_pixel_area	Approximates the grid box area of a lon - lat grid.
spline_smooting	Smoothes a curve with smooting spline.
tube_cutout4box	Performs tube cutout with given index set.
xy2ll	Transformation between local Cartesian coordinates in west-east direction (x) and south-north direction (y) and longitude and latitude.

1.1.3 tropy.analysis_tools.optical_flow

displacement_from_opt_flow	Derives displacement vector between to fields f1 and f2 using optical flow method (either farnebaeck, or tvl1).
displacement_from_opt_flow_farneback	Derives displacement vector between to fields f1 and f2 using optical flow method after Farneback (2003).
displacement_from_opt_flow_tvl1	Derives displacement vector between to fields f1 and f2 using optical flow method after Zach et al (2007).
displacement_vector	Calculates displacement vector for several times.
flow_velocity	Calculates flow field for Lagrangian displacements of tracers in the field.
Lagrangian_change	Calculates Lagrangian change of a field using possibly another field to generate optical flow.
morph_trans_opt_flow	Applies morphological transformation of field f given a displacement field.

1.1.4 tropy.analysis_tools.segmentation

clustering	General interface to aplly either connectivity or watershed clustering.
combine_object_stack	Sequentially combines object label fields by searching for new objects that have no predecessor in the higher hierarchy level.
connected_sequences2pairlists	The routine gets a set of paired numbers which show connection between numbers and collects all connections in a list of pairs.
connectivity_clustering	Applies connectivity clustering to a field.
markers_from_iterative_shrinking	The routine tries to find core parts of objects via iterative shrinking.
multithreshold_clustering	Performs sequential segmentation and returns the combine result only keeping the predecessor object alive.

Continued on next page

Table 4 – continued from previous page

percentiles_from_cluster	Calculates percentiles of cells in an segmented (labeled) field.
remove_clustersize_outside	Counts numbers of pixels per cluster cell and removes clusters outside a size range defined by a minimum and maximum size.
remove_clustersize_outside_slow	OLD VERSION –> much slower . . .
remove_small_clusters	Counts numbers of pixels per cluster cell and removes clusters smaller than minimum size.
sequential_segmentation	Performs sequential segmentation.
set_connectivity_footprint	Return connectivity footprint for either 4-connectivity or 8-connectivity.
sort_clusters	It sorts clusters by size (number of pixels) in decreasing order.
watershed_clustering	Applies watershed clustering to a field.
watershed_merge_clustering	Applies watershed clustering to a field and then tries to merge them again.

1.1.5 tropy.analysis_tools.statistics

autocorr	Calculates autocorrelation function of masked array.
cond_perc_simple	This is a simple routine to emulate the Rosenfeld T-Re plots.
correlation_bootstrap	Calculates percentile values of possible correlation using a bootstrap approach.
crosscorr	Calculates time-lagged crosscorrelation function of masked array.
cumsum_data_fraction	The function uses iso-lines of equal density and maps the fraction of data enclosed by these lines onto it.
draw_from_empirical_1ddist	Draw random number from an empirical distribution.
draw_from_empirical_dist	Draw random number from an empirical distribution.
fdistrib_mapping	Performs a transformation of field fin to have the same distribution as fmap.
get_outlier_from_residuals	Calculate outliers from residuals using percentile-based standardization.
KStest	Calculates Kolmogorov Smirnov test for the distributions of two samples in a standard way, but allows for input of effective degrees of freedom.
normalize_field	Normalization of a field is applied.
odrfit_with_outlier_removal	Calculates fitting parameter object using a orthogonal distance regression.
rank_transformation	The routine performs rank transformation of a field, meaning that the ranks of the individual field values are returned.

1.1.6 tropy.analysis_tools.thermodynamic_variables

absolute_humidity	Calculates absolute humidity.
both_mass_fractions2mixing_ratios	Converts water and the vapor mass fraction to condensed water and vapor mixing ratio.

Continued on next page

Table 6 – continued from previous page

dew_point	Calculates dew point temperature after Markowski book p 13 eq (2.25)
dry_air_density	Calculates dry air density.
dry_air_potential_temperature	Calculates dry air potential temperature.
equivalent_potential_temperature	Calculates equivalent potential temperature.
H2r	Converts relative humidity H into mixing ratio r.
lifting_condensation_level_temperature	Calculates lifting condensation level temperature.
moist_gas_constant	Calculates gas constant of moist air.
moist_potential_temperature	Calculates moist air potential temperature.
moist_specific_heat	Calculates specific heat at constant pressure of moist air.
relative_humidity	Calculates relative humidity.
saturation_over_ice	Calculates saturation water vapor pressure over ice.
saturation_pressure	Calculates saturation water vapor pressure after Bolton (1980) MWR 108, p.1046, eq.
specific_humidity	Calculates specific humidity.
thermodynamic_constants	Sets some thermodynamic constants.
total_density	Calculates total density of moist air with hydrometeors.
watermass_fraction2rw	Converts water mass fraction to condensed water mixing ratio.
water_vapor_pressure	Calculates water vapor pressure.

1.2 tropy.io_tools

1.2.1 tropy.io_tools.data_collection

Dataset	Description:
DataCollection	

CHAPTER 2

Subpackages

2.1 tropy.analysis_tools

The *tropy.analysis* package is a collection of tools for dedicated analysis of scientific data. It has been especially used to explore atmospheric data, but the application to other types of data might be rather straight forward and meaningful.

2.1.1 tropy.analysis_tools.derived_variables

This module contains / will contain variables derived from some basic input fields. The name "basic" refers to the typical set of variables stored in climate or weather model output.

tropy.analysis_tools.derived_variables.**calc_lwp**(*p*, *T*, *qv*, *qc*, *g=9.8*, *axis=-1*)

Calculates the liquid water path.

Parameters

- **p** (*numpy array*) – atmospheric pressure [hPa]
- **T** (*numpy array*) – temperature [K]
- **qv** (*numpy array*) – water vapor mixing ratio [kg / kg]
- **qc** (*numpy array*) – liquid water mixing ratio [kg / kg]

Returns **lwp** – liquid water path [g / m**2]

Return type numpy array

Notes

- hydrostatic approximation is used to convert vertical integration from pressure to height levels
- impact of condesate mass on air density is ignored.

2.1.2 tropy.analysis_tools.grid_and_interpolation

tropy.analysis_tools.grid_and_interpolation.**COSMO_mean_vertical_levels**(*Nlev*)
Outputs the mean vertical levels of COSMO-DE taken from the model description of COSMO (see reference below).

Parameters **Nlev** (*int*) – number of vertical levels, i.e.

Nlev = 51 for intermediate half-levels (e.g. for w) Nlev = 50 for main levels (e.g. for u)

Returns **lev** – mean vertical COSMO-DE levels

Return type numpy array (1-dim)

References

M. Baldauf, J. Foerstner, S. Klink, T. Reinhardt, C. Schraff, A. Seifert und K. Stephan "Kurze Beschreibung des Lokal-Modells Kuerzestfrist COSMO-DE (LMK) und seiner Datenbanken auf dem Datenserver des DWD", Version 1.6, Stand: 31. 03. 2011

Tabelle 1, Seite 29

tropy.analysis_tools.grid_and_interpolation.**create_interpolation_index**(*lon1*, *lat1*, *lon2*, *lat2*, *xy=False*)
Given georeference of two grids in lon and lat, an index is built to map a field on grid 1 to grid 2.

Parameters

- **lon1** (*numpy array (2-dim)*) – longitude of input grid 1
- **lat1** (*numpy array (2-dim)*) – latitude of input grid 1
- **lon2** (*numpy array (2-dim)*) – longitude of target grid 2
- **lat2** (*numpy array (2-dim)*) – latitude of target grid 2
- **xy** (*bool, optional, default = False*) – switch if georef field are interpreted as local Cartesian coordinates: Then no internal transformation is used.

Returns

- **ir** (*numpy array, 2-dim, dtype = int*) – index field for transforming rows
- **ic** (*numpy array, 2-dim, dtype = int*) – index field for transforming columns

tropy.analysis_tools.grid_and_interpolation.**curve_flow_filter**(*f*, *numberOfIterations=5*)
Smoothing filter depending on isoline curvature. Interface for curvature flow filter from simpleITK toolkit.

Parameters

- **f** (*numpy array (2-dim)*) – 2d field to be filtered (smoothed)
- **numberOfIterations** (*int, optional, default = 5*) – number of iterations, increases smooting effect

Returns **f_sm** – smoothed 2d field

Return type numpy array (2-dim)

Notes

Only works if SimpleITK is installed !!!

tropy.analysis_tools.grid_and_interpolation.**cutout_cluster**(*c*, *nc*, *nedge=30*)
Makes a cutout of a categorial field c for an object of class / number nc.

Parameters

- **c** (*numpy array, 2-dim, dtype = int*) – 2d categorial field
- **nc** (*int*) – category number / class which is chosen
- **nedge** (*int, optional, default = 30*) – number of edge pixels added / included

Returns **ccut** – cutout of categorial field c

Return type numpy array, 2-dim, dtype = int

tropy.analysis_tools.grid_and_interpolation.**cutout_field4box**(*f*, *ind*, *bsize*, ***kwargs*)
Cuts out a field based on center pix index and box size.

Parameters

- **f** (*2d or 3d numpy array*) – the field to be cut out
- **ind** (*tuple or list*) – center index of horizontal cutout
- **bsize** (*int*) – size of the quadratic cutout box

Returns **fcut** – resulting quadratic cutout

Return type 2d or 3d numpy array

tropy.analysis_tools.grid_and_interpolation.**cutout_fields**(*fin*, *slices*, *vaxis=0*)
Cuts out a field or a list of similar fields given row and column slices.

Parameters

- **fin** (*2d or 3d numpy array or list of fields*) – the fields to be cut out
- **slices** (*tuple or list*) – region slice such that the cutout is done as ((row1, row2), (col1, col2))
- **vaxis** (*int, optional, default = 0*) – for 3d fields, axis which is still varying

Returns **fout** – resulting cutout (possibly a list of cutted fields)

Return type 2d or 3d numpy array or list of fields

tropy.analysis_tools.grid_and_interpolation.**get_index**(*p*, *lon*, *lat*)
Given a point p the nearest grid point is returned.

Parameters

- **p** (*numpy array*) – point in the 2d grid
- **lon** (*numpy array (2-dim)*) – grid values of the 1st dimension (can be longitude, x, etc.)
- **lat** (*numpy array (2-dim)*) – grid values of the 2nd dimension (can be latitude, y, etc.)

Returns

- **ir** (*numpy array (2-dim)*) – row index (the 1st)
- **ic** (*numpy array (2-dim)*) – column index (the 2nd)

tropy.analysis_tools.grid_and_interpolation.**i2iset**(*v*, *i*)

Returns an index set for a n-dim numpy array given an index i which points to a position within the same but flattened array.

It is assumed that the array was flattened in C-mode.

Parameters

- **v** (*numpy array (n-dim)*) – field which determines the rank and shape
- **i** (*int*) – input index which points to the position in v.flatten()

Returns **iset** – index set which points to the position in v

Return type tuple

Notes

This routine helps you to locate a maximum in a n-dimensional array using e.g. the `np.argmax` function.

tropy.analysis_tools.grid_and_interpolation.**interpolate_field_to_hor_pos**(*pos*, *lon*, *lat*, *field*, ***kwargs*)

Given a sequence of points (or just one) a field (2d or 3d) given a grid is interpolated to the new positions.

Parameters

- **pos** (*numpy array (2-dim)*) – sequence of points given as np.array((x,y)) where x and y can be n-dimensional
- **lon** (*numpy array (2-dim)*) – grid values of the 1st dimension (can be longitude, x, etc.)
- **lat** (*numpy array (2-dim)*) – grid values of the 2nd dimension (can be latitude, y, etc.)
- **field** (*numpy array*) – 2d or 3d field to be interpolated to pos
- ****kwargs** (*dict content*) – keyword arguments for the griddata routine of the scipy.interpolate package can be passed through

Returns **f** – field interpolated to the position sequence pos, f has the dimension n x Nlev, where n is the dimension of pos, Nlev the vertical dimension

Return type numpy array

tropy.analysis_tools.grid_and_interpolation.**ldiff**(*var*, *axis=-1*)

Calculates difference of a field at adjacent levels. Now, wrapper for `numpy.diff` with default on last axis.

Parameters

- **var** (*numpy array (n-dim where n = [1,2,3])*) – field; 1d, 2d or 3d array
- **axis** (*int, optional, default = -1*) – axis used for layer avering

Returns **dvar** – level difference

Return type numpy array (n-dim where n = [1,2,3])

tropy.analysis_tools.grid_and_interpolation.**ll2xy**(*lon*, *lat*, *lon0=10.0*, *lat0=50.0*, *R=6380.0*)

Transformation between longitude and latitude and local Cartesian coordinates in west-east direction (x) and south-north direction (y). The local projection is called "sinusoidal".

Parameters

- **lon** (*numpy array*) – longitude
- **lat** (*numpy array*) – latitude
- **lon0** (*float, optional, default = 10.*) – arbitrary initial longitude for x = 0
- **lat0** (*float, optional, default = 50.*) – arbitrary initial latitude for y = 0
- **R** (*float, optional, default = 6380.*) – approximation for the radius of Earth

Returns

- **x** (*numpy array*) – Cartesian coordinate in west-east direction (increasing towards the East)
- **y** (*numpy array*) – Cartesian coordinate in south-north direction (increasing towards the North)

Notes

The unit of Earth's radius determines the units of x and y. Default (km).

References

See https://en.wikipedia.org/wiki/Sinusoidal_projection for further info.

tropy.analysis_tools.grid_and_interpolation.**ll2xyc**(*lon*, *lat*, *mlon=None*, *mlat=None*, *lon0=0*, *lat0=0.0*)

Applies a centered version of ll2xy. The centering is around the mean lon/lat value and lon0, lat0 only define the x/y offset.

Parameters

- **lon** (*numpy array*) – longitude
- **lat** (*numpy array*) – latitude
- **mlon** (*float, optional, default = None*) – longitude center point of projection

  if `None` then average of `lon` is used
- **mlat** (*float, optional, defaulft = None*) – latitutde center point of projection

  if `None` then average of `lat` is used
- **lon0** (*float, optional, default = 0.*) – zero longitude (for x-offset)
- **lat0** (*float, optional, default = 0.*) – zero latitude (for y-offset)

Returns

- **x** (*numpy array*) – x-coordinate
- **y** (*numpy array*) – y-coordinate

tropy.analysis_tools.grid_and_interpolation.**lmean**(*var*, *axis=-1*)

Calculates the layer mean of a field.

Parameters

- **var** (*numpy array (n-dim where n = [1,2,3])*) – field; 1d, 2d or 3d array

- **axis** (*int, optional, default = -1*) – axis used for layer avering

Returns **varm** – layer mean

Return type numpy array (n-dim where n = [1,2,3])

tropy.analysis_tools.grid_and_interpolation.**low2hres_region**(*region*)

Based on low-res. region of MSG channels an correspoding HRV region is calculated.

Parameters **region** (*tuple of two tubles*) – region slice for cuuting a field ((row1, row2), (col1, col2))

Returns **hrv_region**

Return type corresponding HRV region as ((row1, row2), (col1, col2))

Notes

Calculation

It is assumed that the hrv region is a dependent variable which depends on the low res region attribute That means: Given the low res region cutout hrv region is determined as corresponding cutout which

(i) exactly fits with the low res region and

(ii) refers to the artifical hrv full disk of 11136 x 11136

low res pixel with the index (I,J) = (0,0) has a high res pixel with index (i_m, j_m) = (2,2) in the middle and (i_u, j_u) = (1,1) in the upper corner see doc "MSG Level 1.5 Image Data Format Description", Fig.8 which then leads to the relation between the low res pixel (I,J) and its corresponding upper corner high res pixel (i_u, j_u): (i_u, j_u) = 3 * (I, J) + 1

tropy.analysis_tools.grid_and_interpolation.**make_add_edge**(*var*)

Adds egde region to 2d data field. Values in added left and right column, and lower and upper row are linearily extrapolated from their neighbors.

Parameters **var** (*numpy array (2-dim)*) – 2d variable field (Nrow, Ncols)

Returns **var_new** – 2d variable field with added edge values (Nrows + 2, Ncols + 2)

Return type numpy array (2-dim)

tropy.analysis_tools.grid_and_interpolation.**make_hrv_upscaling**(*v*)

A 2d field is upscaled to 3-fold higher resolution using linear interpolation. Edge values are determined by linear extrapolation.

Parameters **v** (*numpy array (2-dim)*) – 2d variable field (Nrow, Ncols)

Returns **vhigh** – variable field with 3-times higher resolution (3*Nrows, 3*Ncols)

Return type numpy array (2-dim)

tropy.analysis_tools.grid_and_interpolation.**make_index_set**(*nrows*, *ncols*)

This simple routine makes an 2d index set for given row and column number of a 2d field.

Parameters

- **nrows** (*int*) – number of rows
- **ncols** (*int*) – number of columns

Returns

- **ii** (*numpy array, dtype = int*) – 2d field of rows indices
- **jj** (*numpy array, dtype = int*) – 2d field of column indices

tropy.analysis_tools.grid_and_interpolation.**make_vert_cut**(*p1*, *p2*, *lon*, *lat*, *vg*, ***kwargs*)

Makes a horizontal - vertical cut through a 3d field.

Parameters

- **p1** (*list or numpy array*) – 1st end of the cutting line (lon, lat)
- **p2** (*list or numpy array*) – 2nd end of the cutting line (lon, lat)
- **lon** (*numpy array (2-dim)*) – longitude
- **lat** (*numpy array (2-dim)*) – latitude
- **vg** (*numpy array (3-dim)*) – 3d field to be interpolated
- ****kwargs** (*dict content*) – keyword arguments for the griddata routine of the scipy.interpolate package can be passed through

Returns

- **s** (*numpy array (1-dim)*) – distance of km from one to the other end of the cutting line
- **lo** (*numpy array (1-dim)*) – longitude along the cut
- **la** (*numpy array (1-dim)*) – latitude along the cut
- **v** (*numpy array (2-dim)*) – field interpolated to the cutting line

  Vertical levels are kept.

tropy.analysis_tools.grid_and_interpolation.**mid2edge_gridvector**(*x_mid*, *x_edge0*)

Converts an mid-point based grid vector into an edge-based grid vector.

Parameters

- **x_mid** (*numpy array (1-dim)*) – mid-point-based grid vector (length N)
- **x_edge0** (*float*) – left edge

Returns **x_edge** – edge-based grid vector (length N + 1)

Return type numpy array (1-dim)

tropy.analysis_tools.grid_and_interpolation.**region2slice**(*lon*, *lat*, *region*)

Given a region of ((lon1,lon2),(lat1, lat2)) the routine provides a slicing ((row1, row2), (col1, col2)) than accomodates the region.

Parameters

- **lon** (*numpy array (2-dim)*) – longitude
- **lat** (*numpy array (2-dim)*) – latitude
- **region** (*tuple of two tubles*) – ((lon1, lon2), (lat1, lat2))

Returns **slice** – region slice for cuuting a field ((row1, row2), (col1, col2))

Return type tuple of two tubles

tropy.analysis_tools.grid_and_interpolation.**remap_field**(*lon*, *lat*, *f*, *dr=0.5*)

Do nearest nearbor remapping on a simple equi-distant local cartesian coordiante grid.

Uses curvature flow filter for smoothing and pixel edge correction.

Parameters

- **lon** (*numpy array, (2-dim)*) – longitude

- **lat** (*numpy array, (2-dim)*) – latitude
- **f** (*numpy array, (2-dim)*) – 2d field to be interpolated
- **dr** (*float, optional, default = 0.5*) – grid spacing in km

Returns

- **xnew** (*numpy array, (2-dim)*) – regridded, equi-distant x-coordinate in km
- **ynew** (*numpy array, (2-dim)*) – regridded, equi-distant y-coordinate in km
- **fint** (*numpy array, (2-dim)*) – remapped field

tropy.analysis_tools.grid_and_interpolation.**simple_pixel_area**(*lon*, *lat*, *xy=False*, *uncertainty=False*)

Approximates the grid box area of a lon - lat grid.

The grid is treated as if it is an equidistant grid in local Cartesian coordinates on Earth's surface (i.e. in x and y). The grid point is placed at each of the for corners of the grid box and the corresponding average area ist calculated.

Parameters

- **lon** (*numpy array (2-dim)*) – longitude
- **lat** (*numpy array (2-dim)*) – latitude
- **xy** (*bool, optional, default = False*) – switch if `lon` & `lat` are interpreted as local Cartesian coordinates
- **uncertainty** (*bool, optional, default = False*) – switch if std of pixel area is output

  Deviation are caused by effects of the not-rectangular grid.

Returns

- **a** (*numpy array (2-dim)*) – grid box area
- **da** (*numpy array (2-dim), optional*) – standard deviation of grid box area, if `uncertainty == True`

tropy.analysis_tools.grid_and_interpolation.**spline_smooting**(*x*, *y*, *s=0.1*, *fixed_endpoints=True*)

Smoothes a curve with smooting spline.

Parameters

- **x** (*numpy array (1-dim)*) – abscissa values
- **y** (*numpy array (1-dim)*) – ordinate values
- **s** (*float, optional, default = 0.1*) – smooting parameter
- **fixed_endpoints** (*nool, optional, default = True*) – switch, if endpoinds should be hold fixed

Returns **y_smooth** – smoothed version of y

Return type numpy array (1-dim)

Notes

Uses the function `scipy.interpolate.UnivariateSpline`.

`tropy.analysis_tools.grid_and_interpolation.`**`tube_cutout4box`**(*v3d*, *i0*, *boxsize*)

Performs tube cutout with given index set. The center index might not be constant.

Parameters

- **v3d** (*numpy array (3-dim)*) – 3d field, 1st dimension is time (or variable)
- **i0** (*list or numpy array (2-dim)*) – center indices (irvec, icvec) index tuple, (row index vector, co index vector)
- **boxsize** (*int*) – size of the quadratic cutout box

Returns **vtube** – 3d tube, quadratic cutout in the last two dimensions

Return type numpy array (3-dim)

Notes

This function can be used top make vertical cutouts, but a transpose command has to be applied before and after.

`tropy.analysis_tools.grid_and_interpolation.`**`xy2ll`**(*x*, *y*, *lon0=10.0*, *lat0=50.0*, *R=6380*)

Transformation between local Cartesian coordinates in west-east direction (x) and south-north direction (y) and longitude and latitude. This assumes a sinusoidal projection. Inversion of the function `ll2xy`.

Parameters

- **x** (*numpy array*) – Cartesian coordinate in west-east direction (increasing towards the East)
- **y** (*numpy array*) – Cartesian coordinate in south-north direction (increasing towards the North)
- **lon0** (*float, optional, default = 10.*) – arbitrary initial longitude for x = 0
- **lat0** (*float, optional, default = 50.*) – arbitrary initial latitude for y = 0
- **R** (*float, optional, default = 6380.*) – approximation for the radius of Earth

Returns

- **lon** (*numpy array*) – longitude
- **lat** (*numpy array*) – latitude

Notes

The unit of Earth's radius determines the units of x and y. Default (km).

References

See https://en.wikipedia.org/wiki/Sinusoidal_projection for further info.

2.1.3 tropy.analysis_tools.optical_flow

tropy.analysis_tools.optical_flow.**Lagrangian_change**(*f3d*, *tracer_field=None*, *gauss_sigma=1.0*, *flow_input=None*, *vmin=0*, *vmax=1.0*)

Calculates Lagrangian change of a field using possibly another field to generate optical flow.

Parameters

- **f3d** (*numpy array*) – the field (time on 1st axis) for which Lagrangian change is calculated
- **tracer_field** (*numpy array, optional, default = None*) – the field (time on 1st axis) for which displacement is derived if None: displacement is derievd from `f3d`
- **sigma_gauss** (*float, optional, default = 1.*) – sigma for Gaussian smoothing for Lagrangian change fields
- **flow_input** (*numpy array, optional, default = None*) – predefined flow field for u/v calculation

  if None: flow is derived from either `f3d` or `tracer_field`
- **vmin** (*float, optional, default = 0.*) – lower bound of fields for rescaling needed for flow calculation
- **vmax** (*float, optional, default = 1.*) – upper bound of fields for rescaling needed for flow calculation

Returns **df** – Lagrangian change of field `f3d`

Return type numpy array

Notes

Lagrangian changes are derived from displacements between subsequent time slots and hence, time dimension is ntimes - 1.

tropy.analysis_tools.optical_flow.**displacement_from_opt_flow**(*f1*, *f2*, *method='farneback'*, ***kwargs*)

Derives displacement vector between to fields f1 and f2 using optical flow method (either farnebaeck, or tvl1).

Parameters

- **f1** (*numpy array, 2dim*) – field for past time step
- **f2** (*numpy array, 2dim*) – field for actual time step
- **method** (*str, optional, default = 'farneback'*) – method selected for optical flow calculations possible options: 'farneback' and 'tvl1'
- **kwargs** (*dict*) – parameters for opencv optical flow algorithm if not given, default is taken from _farnebaeck_default_parameters

Returns **flow** – displacement vector as index shift 1st component is related to u, i.e. displacement along row, 2nd to v, i.e. along column

Return type numpy array, 3dim

tropy.analysis_tools.optical_flow.**displacement_from_opt_flow_farneback**(*f1*, *f2*, *vmin=None*, *vmax=None*, ***kwargs*)

Derives displacement vector between to fields f1 and f2 using optical flow method after Farneback (2003).

Parameters

- **f1** (*numpy array, 2dim*) – field for past time step
- **f2** (*numpy array, 2dim*) – field for actual time step
- **vmin** (*float, optional, default = None*) – lower bound of fields for rescaling
- **vmax** (*float, optional, default = None*) – upper bound of fields for rescaling
- **kwargs** (*dict*) – parameters for opencv optical flow algorithm if not given, default is taken from _farnebaeck_default_parameters

Returns **flow** – displacement vector as index shift 1st component is related to u, i.e. displacement along row, 2nd to v, i.e. along column

Return type numpy array, 3dim

tropy.analysis_tools.optical_flow.**displacement_from_opt_flow_tvl1**(*f1*, *f2*, *vmin=None*, *vmax=None*, ***kwargs*)

Derives displacement vector between to fields f1 and f2 using optical flow method after Zach et al (2007).

Parameters

- **f1** (*numpy array, 2dim*) – field for past time step
- **f2** (*numpy array, 2dim*) – field for actual time step
- **vmin** (*float, optional, default = None*) – lower bound of fields for rescaling
- **vmax** (*float, optional, default = None*) – upper bound of fields for rescaling
- **kwargs** (*dict*) – parameters for opencv optical flow algorithm if not given, default is taken from _farnebaeck_default_parameters

Returns **flow** – displacement vector as index shift 1st component is related to u, i.e. displacement along row, 2nd to v, i.e. along column

Return type numpy array, 3dim

tropy.analysis_tools.optical_flow.**displacement_vector**(*tracer_field*, *vmin=0*, *vmax=1.0*)

Calculates displacement vector for several times.

Parameters

- **tracer_field** (*numpy array*) – 3d fields (time on 1st axis)
- **vmin** (*float, optional, default = 0.*) – lower bound of fields for rescaling needed for flow calculation
- **vmax** (*float, optional, default = 1.*) – upper bound of fields for rescaling needed for flow calculation

Returns **flow** – displacement vector

Return type numpy array

tropy.analysis_tools.optical_flow.**flow_velocity**(*lon*, *lat*, *f3d*, *dt=5.0*, *flow_input=None*, *vmin=0*, *vmax=1.0*)

Calculates flow field for Lagrangian displacements of tracers in the field.

Parameters

- **lon** (*numpy array*) – longitude
- **lat** (*numpy array*) – latitude
- **f3d** (*numpy array*) – 3d fields (time on 1st axis)
- **dt** (*float, optional, default = 5.*) – time interval in minutes
- **flow_input** (*numpy array, optional, default = None*) – predefined flow field for u/v calculation not use if None
- **vmin** (*float, optional, default = 0.*) – lower bound of fields for rescaling needed for flow calculation
- **vmax** (*float, optional, default = 1.*) – upper bound of fields for rescaling needed for flow calculation

Returns

- **u** (*numpy array*) – zonal velocity,
- **v** (*numpy array*) – meridional velocity

Notes

Velocities are derived from displacements between subsequent time slots and hence, time dimension is ntimes - 1.

tropy.analysis_tools.optical_flow.**morph_trans_opt_flow**(*f*, *flow*, *method='forward'*)

Applies morphological transformation of field f given a displacement field.

Parameters

- **f** (*numpy array*) – 2d field that is transformed (source)
- **flow** (*numpy array*) – displacement vector between source and target stage

Returns **ftrans** – transformed field

Return type numpy array

2.1.4 tropy.analysis_tools.parallax

2.1.5 tropy.analysis_tools.segmentation

Package to perform segmentation of fields.

The special emphasis is on threshold-based segmentation of 2-dim atmoshperic fields for which object-based analysis techniques should be applied.

Notes

- the package has variants of connected compund analsis & watershedding
- some split-and-merge rules exists
- most of the method might be also applied to 3-dim fields

tropy.analysis_tools.segmentation.**clustering**(*f*, *thresh*, *cluster_method='watershed'*, *min_size=20*, ***kwargs*)

General interface to aplly either connectivity or watershed clustering.

Parameters

- **f** (*np.array*) – field to be segmented
- **thresh** (*float*) – threshold for masking, set foreground to f > thresh
- **cluster_method** (*{'watershed', 'watershed_merge', 'connect'}, optional*) – switch between watershed or connectivity clustering
- **min_size** (*int, optional, default = 20*) – minimum cluster size (px) left in the cluster field
- ****kwargs** (*dict, optional*) – keyword argument for selected cluster algorithm

Returns **c** – categorial cluster field

Return type np.array

tropy.analysis_tools.segmentation.**combine_object_stack**(*c3d*)

Sequentially combines object label fields by searching for new objects that have no predecessor in the higher hierarchy level.

Parameters **c3d** (*numpy array, 3dim or 4dim*) – labeled object data stack

Returns **c** – combined label field

Return type numpy array, 2dim or 3dim

tropy.analysis_tools.segmentation.**connected_sequences2pairlists**(*connected_pairs_list*)

The routine gets a set of paired numbers which show connection between numbers and collects all connections in a list of pairs.

Parameters **connected_pairs_list** (*list*) – list of number pairs

Returns **pair_list** – dictionary that contains pair lists

Return type dict

tropy.analysis_tools.segmentation.**connectivity_clustering**(*f*, *thresh*, *filter_method='curve'*, *numberOfIterations=5*, *ctype=4*, *cluster_masking=True*, *siggauss=1.0*, ***kwargs*)

Applies connectivity clustering to a field.

Parameters

- **f** (*np.array*) – field to be segmented
- **thresh** (*float*) – threshold for masking, set foreground to f > thresh
- **filter_method** (*{'gauss', 'curve'}, optional*) – filter method used to smooth the input field
  - 'gauss' for 2d-Gaussian
  - 'curve' for curvature flow filter
- **numberOfIterations** (*int, optional*) – number of repeated application of curvature flow filter the larger then smoother if selected
- **ctype** (*{4, 8}, optional*) – set either 4- or 8-connectivity (edge vs. edges+corners)

- **cluster_masking** (*{True, False}, optional*) – if original (True) or smoothed (False) threshold mask is used.
- **siggauss** (*float, optional*) – sigma for Gaussian filter if selected
- ****kwargs** (*dict, optional*) – set of additional, but not used keywords

Returns **c** – categorial cluster field

Return type np.array

tropy.analysis_tools.segmentation.**markers_from_iterative_shrinking**(*mask*, *marker_shrinkage_Niter=3*, *marker_shrinkage_dmin=0.1*, ***kwargs*)

The routine tries to find core parts of objects via iterative shrinking.

Each object is normalized by its maximum distance-to-background which makes the algorithm scale-invariant.

Parameters **mask** (*np.array*) – binary mask

marker_shrinkage_Niter [int, optional] option for marker creation, how often shrinking is applied

marker_shrinkage_dmin [float, optional] option for marker creation

How much of the edge is taken away. Distance is measured relative to the maximum distance, thus 0 < dmin < 1.

Returns **markers** – markers field with increasing index per marker region

Return type np.array

tropy.analysis_tools.segmentation.**multithreshold_clustering**(*f*, *thresh_min*, *thresh_max*, *nthresh=10*, *use_percentile_threshold=False*, ***kws*)

Performs sequential segmentation and returns the combine result only keeping the predecessor object alive.

Parameters

- **f** (*numpy array, 2dim or 3dim*) – input field
- **thresh_min** (*float*) – minimum threshold value
- **thresh_max** (*float*) – maximum threshold value
- **nthresh** (*int, optional*) – number of threshold values
- **use_percentile_threshold** (*{True, False}, optional*) – if threshold list is derived from field percentiles
- ****kws** (*dict*) – keywords passed to segmentation routine

Returns **c** – combined label field

Return type numpy array, same shape as f

tropy.analysis_tools.segmentation.**percentiles_from_cluster**(*f, c, p=[25, 50, 75], index=None*)

Calculates percentiles of cells in an segmented (labeled) field.

Functionality missing in scipy.ndimage.measurements.

Parameters

- **f** (*np.array*) – the field as basis for percentile calculation
- **c** (*np.array*) – categorial cluster field
- **p** (*list or np.array, optional*) – percentiles array
- **index** (list or *None*, optional) – list of object labels for which percentiles are calculated

Returns **pc** – percentiles per cell (including background (set to zero))

Return type np.array

tropy.analysis_tools.segmentation.**remove_clustersize_outside**(*c*, *min_size=6*, *max_size=inf*)

Counts numbers of pixels per cluster cell and removes clusters outside a size range defined by a minimum and maximum size.

In addition, it sorts clusters by size in decreasing order.

Parameters

- **c** (*np.array*) – categorial cluster field
- **min_size** (*int, optional*) – minimum cluster size (px) left in the cluster field
- **max_size** (*int, optional*) – maximum cluster size (px) left in the cluster field

Returns

c_re – categorial cluster field with sorted objects (from large to small)

- objects smaller than or equal *min_size* are removed
- objects larger than or equal *max_size* are removed

Return type np.array

tropy.analysis_tools.segmentation.**remove_clustersize_outside_slow**(*c*, *min_size=6*, *max_size=inf*, *sort=True*)

OLD VERSION –> much slower . . .

Counts numbers of pixels per cluster cell and removes clusters outside a size range defined by a minimum and maximum size.

In addition, it sorts clusters by size in decreasing order.

Parameters

- **c** (*np.array*) – categorial cluster field
- **min_size** (*int, optional*) – minimum cluster size (px) left in the cluster field
- **max_size** (*int, optional*) – maximum cluster size (px) left in the cluster field
- **sort** (*{True, False}, optional*) – if objects are sorted by size

Returns

c_re – categorial cluster field with sorted objects (from large to small)

- objects smaller than or equal *min_size* are removed
- objects larger than or equal *max_size* are removed

Return type np.array

tropy.analysis_tools.segmentation.**remove_small_clusters**(*c*, *min_size=6*)

Counts numbers of pixels per cluster cell and removes clusters smaller than minimum size.

In addition, it sorts clusters by size in decreasing order.

Parameters

- **c** (*np.array*) – categorial cluster field
- **min_size** (*int, optional*) – minimum cluster size (px) left in the cluster field

Returns

c_re – categorial cluster field with sorted objects (from large to small)

objects smaller than or equal *min_size* are removed

Return type np.array

tropy.analysis_tools.segmentation.**sequential_segmentation**(*f*, *thresh_min*, *thresh_max*, *nthresh=10*, *use_percentile_threshold=False*, ***kws*)

Performs sequential segmentation.

Parameters

- **f** (*numpy array, 2dim or 3dim*) – input field
- **thresh_min** (*float*) – minimum threshold value
- **thresh_max** (*float*) – maximum threshold value
- **nthresh** (*int*) – number of threshold values
- **use_percentile_threshold** (*{True, False}, optional*) – if threshold list is derived from field percentiles
- ****kws** (*dict*) – keywords passed to segmentation routine

Returns **c3d** – stack of labeled field, from max. thresh to min. thresh

Return type numpy array, one dimension added

tropy.analysis_tools.segmentation.**set_connectivity_footprint**(*ctype*, *ndim*)

Return connectivity footprint for either 4-connectivity or 8-connectivity.

4-connectivity only includes neigboring sides, 8-connectivity also includes neighboring diagonal values.

Parameters

- **ctype** (*{4, 8}*) – set either 4- or 8-connectivity (edge vs. edges+corners)
- **ndim** (*{2,3}*) – dimension of the field (either 2-dim or 3-dim)

Returns **footprint** – ndim-dimensional connectivity footprint

Return type np.array

tropy.analysis_tools.segmentation.**sort_clusters**(*c*)

It sorts clusters by size (number of pixels) in decreasing order.

Parameters **c** (*np.array*) – categorial cluster field

Returns **c_re** – categorial cluster field with sorted objects (from large to small)

Return type np.array

tropy.analysis_tools.segmentation.**watershed_clustering**(*f*, *thresh*, *dradius=3*, *ctype=4*, *marker_field='dist'*, *filter_method='curve'*, *numberOfIterations=5*, *cluster_masking=True*, *exclude_border=False*, *marker_method='mahotas_regmax'*, *siggauss=1.0*, *marker_shrinkage_Niter=3*, *marker_shrinkage_dmin=0.1*, ***kwargs*)

Applies watershed clustering to a field.

Parameters

- **f** (*np.array*) – field to be segmented
- **thresh** (*float*) – threshold for masking, set foreground to f > thresh
- **dradius** (*int, optional*) – minimal distance for neighboring maxima
- **ctype** (*{4, 8}, optional*) – set either 4- or 8-connectivity (edge vs. edges+corners)
- **marker_field** (*str, optional*) – switch if input field *f* or a distance transform as used to set the watershed markers

  if 'dist', then max. in Euclidean distance to background is used to set initial markers, else the max. in field f are taken
- **filter_method** (*{'gauss', 'curve'}, optional*) – filter method used to smooth the input field
  - 'gauss' for 2d-Gaussian
  - 'curve' for curvature flow filter
- **numberOfIterations** (*int, optional*) – number of repeated application of curvature flow filter the larger then smoother if selected
- **cluster_masking** (*{True, False}, optional*) – if original (True) or smoothed (False) threshold mask is used.
- **exclude_border** (*{True, False}, optional*) – if clusters that touch domain borders are excluded (set to zero)
- **marker_method** (*str, optional*) – defines a method for marker assignment
  - 'brute_force_local_max' : just uses scipy.ndimage.maximum_filter
  - 'skimage_peak_local_max' : uses skimage local maximum routine
  - 'mahotas_regmax' : uses mahotas regional maximum detection
  - 'iterative_shrinking' : uses iterative object shrinking depending on relative distance-to-background
- **siggauss** (*float, optional*) – sigma for Gaussian filter if selected
- **marker_shrinkage_Niter** (*int, optional*) – option for marker creation, how often shrinking is applied
- **marker_shrinkage_dmin** (*float, optional*) – option for marker creation

How much of the edge is taken away. Distance is measured relative to the maximum distance, thus 0 < dmin < 1.

Returns **c** – categorial cluster field

Return type np.array

tropy.analysis_tools.segmentation.**watershed_merge_clustering**(*f*, *thresh*, *min_size=20*, *merge_ratio=0.5*, ***kwargs*)

Applies watershed clustering to a field and then tries to merge them again.

The merging is based on the ratio between maximum distances-to-background within the subclusters and on the cluster broders.

Parameters

- **f** (*np.array*) – field to be segmented
- **thresh** (*float*) – threshold for masking, set foreground to f > thresh
- **min_size** (*int, optional, default = 20*) – optional, minimum cluster size (px) left in the cluster field
- **merge_ratio** (*float, optional*) – essentially the ratio between interface length and cluster size

  if the interface is longer than *merge_ratio* * cluster_size than to two connected objects are merged.
- ****kwargs** (*dict, optional*) – keywords passed to *watershed_clustering* routine

Returns **c_update** – categorial cluster field (updated for merges)

Return type np.array

2.1.6 tropy.analysis_tools.statistics

Package for some statistical utility functions.

Special Highlights are:

- conditioned percentiles
- rank transformation / distribution mapping
- drawing from an empirical 1d / 2d distribution

tropy.analysis_tools.statistics.**KStest**(*x*, *y*, *Nx_eff=None*, *Ny_eff=None*)

Calculates Kolmogorov Smirnov test for the distributions of two samples in a standard way, but allows for input of effective degrees of freedom.

Nullhypothesis that the two samples share the same distribution is rejected if p-value is smaller than a threshold (typically 0.05 or 0.01).

Parameters

- **x** (*numpy array*) – sample of 1st variable
- **y** (*numpy array*) – sample of 2nd variable
- **Nx_eff** (*int, optional, default = None*) – number of effective degrees of freedom of variable x set to size of x if Nx_eff == None

- **Ny_eff** (*int, optional, default = None*) – number of effective degrees of freedom of variable y set to size of y if Ny_eff == None

Returns

- **D** (*float*) – KS test statistic
- **pval** (*float*) – p-value

tropy.analysis_tools.statistics.**autocorr**(*x*, *Nmax=None*)

Calculates autocorrelation function of masked array.

Parameters

- **x** (*numpy array*) – sample of 1st variable
- **Nmax** (*int, optional, default = None*) – maximum lag Nmax is set to len(x) - 1 if Nmax == None

Returns **c** – autocorrelation function for lag 0 . . . Nmax,

Return type numpy array

tropy.analysis_tools.statistics.**cond_perc_simple**(*v1, v2, x1, p=[10, 25, 50, 75, 90], sig=0, medsize=0*)

This is a simple routine to emulate the Rosenfeld T-Re plots.

Input are v1 (T) and v2 (Re) and percentiles of v2 conditioned on intervals of v1 given by x1 are calculated.

Parameters

- **v1** (*numpy array*) – sample of 1st variable of bi-variate distribution
- **v2** (*numpy array*) – sample of 2nd variable of bi-variate distribution
- **x1** (*numpy array*) – intervals at which 1st variable is gathered
- **p** (*list, optional, default = [10, 25, 50, 75, 90]*) – percentiles of 2nd variables which will be calculated for each interval [x1[i], x1[i + 1])
- **sig** (*float, optional, default = 0.*) – sigma for Gaussian filter to smooth percentile curves (not applied for sig == 0)
- **medsize** (*int, optional, default = 0*) – footprint size of median filter to smooth percentile curves (not applied for sig != 0 OR medsize == 0)

Returns **v2p** – percentile statistics of variable `v2` conditioned on different `x1` intervals

Return type numpy array

tropy.analysis_tools.statistics.**correlation_bootstrap**(*x, y, perc=[2.5, 50, 97.5], xerr=0, yerr=0, Nsample=1000, corr=<function pearsonr>*)

Calculates percentile values of possible correlation using a bootstrap approach.

(x,y) are interpreted as mean values and error bars as standard deviations of 2d Gaussian distribution per point and the fit is repeated for random draws out of these. After Nsampe fits, the range of possible fits is analyzed for their percentiles.

Parameters

- **x** (*numpy array, 1-dim*) – independent values
- **y** (*numpy array, 1-dim*) – dependent values

- **perc** (*list or numpy array (1-dim), optional, default = [2.5, 50, 97.5]*) – list of percentiles for which correlation is estimated
- **xerr** (*float or numpy array (1-dim), optional, default = 0.*) – standard deviation of x
- **yerr** (*float or numpy array (1-dim), optional, default = 0.*) – standard deviation of y
- **Nsample** (*int, optional, default = 1000*) – size of bottstrap sample
- **corr** (func, optional, default = `scipy.stats.pearsonr`) – function object used for correlation calculation

Returns **corr** – percentiles of possible correlation obtained from bootstrapping

Return type numpy array, 1-dim

tropy.analysis_tools.statistics.**crosscorr**(*x*, *y*, *Nmax=None*)

Calculates time-lagged crosscorrelation function of masked array.

Parameters

- **x** (*numpy array*) – 1st masked input array, evaluated at t + tau
- **y** (*numpy array*) – 2nd masked input array
- **Nmax** (*int, optional, default = None*) – maximum lag Nmax is set to len(x) - 1 if Nmax == None

Returns **c** – autocorrelation function for lag 0 . . . Nmax,

Return type numpy array

tropy.analysis_tools.statistics.**cumsum_data_fraction**(*h*, *divide_by_total_sum=True*)

The function uses iso-lines of equal density and maps the fraction of data enclosed by these lines onto it.

Parameters

- **h** (*numpy array (n-dim)*) – histogram / density or other variable for which cumsum makes sense
- **divide_by_total_sum** (*bool, optional, default = True*) – switch if cumsum field should be divided by total sum this generates a relative field with range (0, 1)

Returns **f** – data fraction field

Return type numpy array (n-dim)

tropy.analysis_tools.statistics.**draw_from_empirical_1ddist**(*bins*, *hist*, *Nsamp=100*)

Draw random number from an empirical distribution. Implemented for 1d histograms.

Parameters

- **bins** (*numpy array*) – bins edges (1dim)
- **hist** (*numpy array (1-dim)*) – histogram values (either absolute frequencies or relative)
- **Nsamp** (*int, optional, default = 100*) – number of random samples that are drawn

Returns **rvalues** – random values that are distributed like hist

Return type numpy array

tropy.analysis_tools.statistics.**draw_from_empirical_dist**(*bins*, *hist*, *Nsamp=100*, *discrete_version=False*)

Draw random number from an empirical distribution. Implemented for 2d histograms.

Parameters

- **bins** (*list of two numpy arrays*) – bins edges (2dim)
- **hist** (*numpy array (2-dim)*) – histogram values (either absolute frequencies or relative)
- **Nsamp** (*int, optional, default = 100*) – number of random samples that are drawn
- **discrete_version** (*bool, optional, default = False*) – if True: samples can only be placed at bin midpoints if False: samples are uniformly distributed within each bin

Returns **rvalues** – random values (2d) that are distributed like hist

Return type list of two numpy arrays

tropy.analysis_tools.statistics.**fdistrib_mapping**(*fin*, *fmap*, *keep_range=False*)

Performs a transformation of field fin to have the same distribution as fmap.

Parameters

- **fin** (*numpy array (n-dim)*) – input field
- **fmap** (*numpy array (n-dim)*) – field from which distribution is taken
- **keep_range** (*bool, optional, default = False*) – switch if output field is scaled to input range (with min-max transformation)

Returns **fout** – transformed field fin

Return type numpy array (n-dim)

tropy.analysis_tools.statistics.**get_outlier_from_residuals**(*e*, *thresh=3.0*)

Calculate outliers from residuals using percentile-based standardization.

Parameters

- **e** (*numpy array*) – input residuals
- **thresh** (*float, optional, default = 3.*) – value of residual (standardized) at which outliers are identified

Returns **m** – outlier mask

Return type numpy array

tropy.analysis_tools.statistics.**normalize_field**(*f*, *vmin=None*, *vmax=None*)

Normalization of a field is applied.

Parameters

- **f** (*numpy array, 2dim*) – 2dim field
- **vmin** (*float, optional, default = None*) – lower bound of fields for rescaling
- **vmax** (*float, optional, default = None*) – upper bound of fields for rescaling

Returns **f_norm** – 2dim field

Return type numpy array, 2dim

tropy.analysis_tools.statistics.**odrfit_with_outlier_removal**(*func*, *x*, *y*, *xerr=0*, *yerr=0*, *n_iter=10*, *outlier_thresh=3.0*, *p0=None*)

Calculates fitting parameter object using a orthogonal distance regression.

(x,y) are interpreted as mean values and error bars as standard deviations of 2d Gaussian distribution per point. The range of possible fits is analyzed for their percentiles.

Parameters

- **func** (*function*) – fitting function func(args, x), args: parameters of the function ! ARGS HAVE BEEN TURNED !
- **x** (*numpy array, 1-dim*) – independent values
- **y** (*numpy array, 1-dim*) – dependent values
- **xerr** (*float or numpy array (1-dim), optional, default = 0.*) – standard deviation of x
- **yerr** (*float or numpy array (1-dim), optional, default = 0.*) – standard deviation of y
- **n_iter** (*int, optional, default = 10*) – numbers of iteration used for outlier removal
- **outlier_thresh** (*float, optional, default = 3.*) – threshold for outlier removal (standardized fit residuals)
- **p0** (*list or tuple, optional, default = None*) – first guess of fitted parameters (for increased convergence)

Returns **fit_result** – result of the ODR fit

Return type ‘‘ scipy.odr.ODR‘‘

tropy.analysis_tools.statistics.**rank_transformation**(*f*, *normalize=True*, *gamma=1.0*)

The routine performs rank transformation of a field, meaning that the ranks of the individual field values are returned.

Parameters

- **f** (*numpy array (n-dim)*) – field to be transformed
- **normalize** (*bool, optional, default = True*) – option if the rank field should be normalized to one
- **gamma** (*float, optional, default = 1.*) – gamma correction factor

Returns **f** – field ranks

Return type numpy array (n-dim)

2.1.7 tropy.analysis_tools.thermodynamic_variables

Package for the calculation of several thermodynamical properties of moist air (including condensate).

ToDo

- Some final tests of the routines should be done.
- There was an issue with mixing ratio vs. specific humidity, but might be resolved . . .

tropy.analysis_tools.thermodynamic_variables.**H2r**(*hrel*, *p*, *T*)
Converts relative humidity H into mixing ratio r.

Parameters

- **hrel** (*numpy array*) – relative humidity in [%]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **es** – water vapor pressure in [Pa]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**absolute_humidity**(*r*, *p*, *T*)
Calculates absolute humidity.

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **rho_v** – absolute humidity / m**3]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**both_mass_fractions2mixing_ratios**(*qv*, *qw*)
Converts water and the vapor mass fraction to condensed water and vapor mixing ratio.

Parameters

- **qv** (*numpy array*) – water vapor mass fraction [kg / kg]
- **qw** (*numpy array*) – condensed water mass fraction [kg / kg]

Returns

- **r** (*numpy array*) – water vapor mixing ratio [kg / kg]
- **rw** (*numpy array*) – condensed water mixing ratio [kg / kg]

tropy.analysis_tools.thermodynamic_variables.**dew_point**(*r*, *p*, *T*)
Calculates dew point temperature after Markowski book p 13 eq (2.25)

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **Td** – dew point temperature in [K]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**dry_air_density**(*r*, *p*, *T*)
Calculates dry air density.

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]

- **T** (*numpy array*) – temperature [K]

Returns **rho_d** – dry air density in [kg / m**3]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**dry_air_potential_temperature**(*r*, *p*, *T*)

Calculates dry air potential temperature.

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **theta_d** – dry air potential temperature (K)

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**equivalent_potential_temperature**(*r*, *p*, *T*, *eq=15*)

Calculates equivalent potential temperature. after Bolton (1980) MWR 108, p.1046, eq. (43)

Davies-Jones (2009) MWR137, eq. (6.3)

also using either one of (15), (21) or (22)

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]
- **eq** (*int, optional, default = 15*) – which eq. of Bolton (1980) to be chosen for lifting condensation level temperature eq in [15, 21, 22]

Returns **theta_E** – equivalent potential temperature

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**lifting_condensation_level_temperature**(*r*, *p*, *T*, *eq=15*)

Calculates lifting condensation level temperature. after Bolton (1980) MWR 108, p.1046, eq. (15), (21) or (22)

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]
- **eq** (*int, optional, default = 15*) – which eq. of Bolton (1980) to be chosen eq in [15, 21, 22]

Returns **TL** – lifting condensation level temperature in [K]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**moist_gas_constant**(*r*)
Calculates gas constant of moist air.

Parameters **r** (*float or numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]

Returns **R_m** – gas constant of moist air [J / kg / K]

Return type float or numpy array

tropy.analysis_tools.thermodynamic_variables.**moist_potential_temperature**(*r*, *p*, *T*)
Calculates moist air potential temperature.

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **theta_m** – moist air potential temperature (K)

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**moist_specific_heat**(*r*)
Calculates specific heat at constant pressure of moist air.

Parameters **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]

Returns **c_pm** – specific heat at constant pressure of moist air

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**relative_humidity**(*r*, *p*, *T*)
Calculates relative humidity.

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **H** – relative humidity in [%]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**saturation_over_ice**(*T*, *method='PK'*)
Calculates saturation water vapor pressure over ice.

Parameters

- **T** (*numpy array*) – temperature [K]
- **method** (*str, optional, default = 'PK'*) – use an equation which approximates the vapor pressure over ice method in ['PK', 'Murphy']

Returns **es** – water vapor pressure in [Pa]

Return type numpy array

Notes

'PK' refers to Pruppacher and Klett (1997) 'Murphy' refers to Murphy and Koop (2005) eq. (7)

References

Murphy, D. M., and T. Koop (2005), Review of the vapour pressures of ice and supercooled water for atmospheric applications, Quart. J. Roy. Meteor. Soc., 131(608), 1539-1565, doi:10.1256/qj.04.94.

tropy.analysis_tools.thermodynamic_variables.**saturation_pressure**(*T*)
Calculates saturation water vapor pressure after Bolton (1980) MWR 108, p.1046, eq. (10)

Parameters **T** (*numpy array*) – temperature [K]

Returns **es** – water vapor pressure in [Pa]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**specific_humidity**(*r*)
Calculates specific humidity.

Parameters **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]

Returns **q** – specific humidity i.e. vapor mass per moist air mass [kg / kg]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**thermodynamic_constants**()
Sets some thermodynamic constants.

Parameters **None** –

Returns **const** – set of thermodynamic constans saved in dictionary

Return type dict

tropy.analysis_tools.thermodynamic_variables.**total_density**(*r*, *r_w*, *p*, *T*)
Calculates total density of moist air with hydrometeors.

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **r_w** (*numpy array*) – mixing ratio of condensed water i.e. condensed water mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **rho_d** – dry air density in [kg / m**3]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**water_vapor_pressure**(*r*, *p*, *T*)
Calculates water vapor pressure.

Parameters

- **r** (*numpy array*) – mixing ratio i.e. vapor mass per dry air mass in [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **e** – water vapor pressure in [Pa]

Return type numpy array

tropy.analysis_tools.thermodynamic_variables.**watermass_fraction2rw**(*qw*, *r*)
Converts water mass fraction to condensed water mixing ratio.

Parameters

- **qw** (*numpy array*) – condensed water mass fraction [kg / kg]
- **r** (*numpy array*) – mixing ration of water vapor [kg / kg]
- **p** (*numpy array*) – total gas pressure [Pa]
- **T** (*numpy array*) – temperature [K]

Returns **rw** – condensed water mixing ratio [kg / kg]

Return type numpy array

2.2 tropy.io_tools package

2.2.1 Submodules

2.2.2 tropy.io_tools.HRIT module

tropy.io_tools.HRIT.**SEVIRI_channel_list**()

tropy.io_tools.HRIT.**bit_conversion**(*input_set*, *inbit=8*, *outbit=10*)
An integer array which is based on a n-bit representation is converted to an integer array based on a m-bit representation.

output_set = bit_conversion(input_set, inbit=8, outbit=10)

input_set : numpy integer array in a n-bit representation inbit : number of bits for input_set (n) outbit : number of bits for output_set (m)

output_set : numpy integer array in a m-bit representation

The size of input_set should be a multiple of inbit / outbit!

tropy.io_tools.HRIT.**channel_segment_sets**(*set_name*)
Outputs a predefined set of channels and segments'

chan_seg = channel_segment_sets(set_name)

set_name: name of a predefined set

chan_seg: Dictionary of channels with a list of segments.

tropy.io_tools.HRIT.**combine_segments**(*seg_dict*, *missing=-1*)
Combines individual segments of SEVIRI scans. Segments will be sorted by key.

combined = combine_two_segments(seg_list)

seg_list: dictionary of segment, sorted by key from bottom to top

combined: combination of both segments

tropy.io_tools.HRIT.**combine_two_segments**(*seg1*, *seg2*)
Combines two segments where seg1 is the upper and seg2 the lower segment.

seg = combine_two_segments(seg1,seg2)

seg1: upper segment seg2: lower segment

seg: combination of both segments

tropy.io_tools.HRIT.**divide_two_segments**(*seg*)

Divides one segment into two with half the line size.

seg1, seg2 = divide_two_segments(seg)

seg: combination of two segments

seg1: upper segment seg2: lower segment

tropy.io_tools.HRIT.**get_HRIT_from_arch**(*day, chan_seg={'IR_016': [1, 2, 3, 4, 5, 6, 7, 8], 'VIS006': [1, 2, 3, 4, 5, 6, 7, 8], 'VIS008': [1, 2, 3, 4, 5, 6, 7, 8]}, scan_type='pzs', tarname=None, arch_dir=None, out_path=None*)

Gets and decompresses the original HRIT files from archive. A typical set of channels and segments can be chosen.

file_list = get_HRIT_from_arch(day, chan_set = 'nc-full', scan_type = 'pzs' , arch_dir = None, out_path = None)

day: datetime object which include day and time of MSG time slot chan_set = name for a channel-segment set scan_type: which type of msg is used,

-) allowed keys are: 'pzs' and 'rss'

arch_dir = archive directory out_path = directory where the HRIT files are saved.

file_list : list of extracted HRIT files

tropy.io_tools.HRIT.**read_HRIT_data**(*day, chan_seg, calibrate=True, scan_type='pzs', arch_dir=None, standard_seg=(464, 3712), add_meta=False, **kwargs*)

Read HRIT data given a time slot, satellite type and channel-segment set

combined = read_HRIT_data(day, chan_seg, calibrate=True, scan_type = 'pzs', arch_dir = None, add_meta = False, **kwargs)

day = datetime object which include day and time of MSG time slot chan_seg = a channel-segment set, dictionary of channel names with lists of segments calibrate = flag, if slope / offset calibration should be done scan_type = which type of msg is used,

-) allowed keys are: 'pzs' and 'rss'

arch_dir = archive directory add_meta = flag, if meta data should be added out_path = directory where the HRIT files are saved.

combined = dictionary of retrieved channels with combined segments

tropy.io_tools.HRIT.**read_HRIT_seg_data**(*hrit_file*)

Reads directly the counts data for an HRIT segment.

seg_data = read_HRIT_seg_data(hrit_file)

hrit_file : HRIT file of the considered segment

seg_data: integer segment data field of radiance counts

tropy.io_tools.HRIT.**read_slope_offset_from_prolog**(*pro_file, NJUMP=387065*)

Reads directly slope and offset from HRIT PRO file.

slope, offset = read_slope_offset_from_prolog(pro_file)

pro_file : PROLOG file of the considered time slot

slope: slope of the data offset: offset of the data

Following transformation is applied:

DATA = SLOPE * COUNTS + OFFSET

if COUNTS are valid data points!

!! ATTENTION !! Jump in binary file is hard-coded!

tropy.io_tools.HRIT.**segments_for_region**(*region*, *Nrow=3712*, *Nseg=8*, *seg_size=464*)

tropy.io_tools.HRIT.**write_HRIT_seg_data**(*seg_data*, *hrit_file*)

Reads directly the counts data for an HRIT segment.

write_HRIT_seg_data(seg_data, hrit_file)

hrit_file : HRIT file of the considered segment seg_data: integer segment data field of radiance counts

2.2.3 tropy.io_tools.bit_conversion module

2.2.4 tropy.io_tools.data_collection module

class tropy.io_tools.data_collection.**DataCollection**

Bases: dict

add(*vname*, *array*, *setting='bt'*, *subpath=None*)

list()

save(*fname*)

class tropy.io_tools.data_collection.**Dataset**(*name*, *data=None*, *setting={'_FillValue': 0, 'add_offset': 0, 'dtype': '\u2', 'longname': 'Brightness Temperature', 'scale_factor': 0.01, 'unit': 'K'}*)

Bases: object

The Dataset class is a building block of the DataCollection class which will be used to easy input and output dataset sets.

add(*array*)

array()

attrs()

load(*fname*, *subpath=None*)

save(*fname*, *subpath=None*, *mode='overwrite'*, *time_out=300.0*)

2.2.5 tropy.io_tools.data_settings module

tropy.io_tools.data_settings.**Dge_settings**()

tropy.io_tools.data_settings.**bt_settings**()

tropy.io_tools.data_settings.**histogram_settings**()

tropy.io_tools.data_settings.**refl_settings**()

tropy.io_tools.data_settings.**settings**(*vartype='bt'*)

tropy.io_tools.data_settings.**standard_int_setting**(*longname='Unknown'*, *unit='Unknown'*)

tropy.io_tools.data_settings.**standard_real_setting**(*longname='Unknown'*, *unit='Unknown'*)

2.2.6 tropy.io_tools.file_status module

tropy.io_tools.file_status.**test**(*fname*)

Returns status of a file:

True: if open False: if not open

2.2.7 tropy.io_tools.find_latest_slot module

2.2.8 tropy.io_tools.hdf module

tropy.io_tools.hdf.**dict_merge**(*a*, *b*)

Recursively merges dict's. not just simple a['key'] = b['key'], if both a and bhave a key who's value is a dict then dict_merge is called on both values and the result stored in the returned dictionary.

from https://www.xormedia.com/recursively-merge-dictionaries-in-python/

tropy.io_tools.hdf.**get_seviri_chan**(*chan_list*, *day*, *scan_type='rss'*, *fname=None*, *calibrate=True*, *add_meta=False*, *add_geo=False*, *arch_dir=None*)

Reads MSG - SEVIRI radiance of a given channel for a given date.

var = get_seviri_chan(chan_list, day, scan_type = 'rss', calibrate=True, add_meta = False)

chan_list: name of channel, e.g. ir_108 or IR_108 (case does not matter) or list of channel names

day: datetime object which include day and time of MSG time slot

scan_type: sets of scanning modus, i.e. 'rss' or 'pzs' (DEFAULT: 'rss')

calibrate : optional, decides if output is radiance (True) or counts (False)

add_meta: meta data of the MSG-SEVIRI hdf file

var: dictionary including radiance/counts of channel <ch_name> at date <day> and if chosen meta data

tropy.io_tools.hdf.**list_hdf_groups**(*fname*)

Makes a list of hdf groups.

Only the 1st level is implemented, TDB: recursive listing.

glist = list_hdf_groups(fname)

fname: filename of hdf file

glist: list of group names

tropy.io_tools.hdf.**read_dict_cont**(*f*, *d*)

Recursively read nested dictionaries.

tropy.io_tools.hdf.**read_dict_from_hdf**(*fname*)

The content of an hdf file with arbitrary depth of subgroups is saved in nested dictionaries.

d = read_dict_from_hdf(fname)

fname: filename of output hdf file

d: nested dictionary which contains the data

tropy.io_tools.hdf.**read_var_from_hdf**(*fname*, *vname*, *subpath=None*)
A specific variable is read from an hdf file. Group and subgroups can be specified.

v = read_var_from_hdf(fname, vname, subpath = None)

fname: filename of input hdf file vname: variable name as given in hdf file

subpath: group or subgroup path with /

v: variable as numpy array

tropy.io_tools.hdf.**save_dict2hdf**(*fname*, *d*, *mode='w'*)
The content of nested dictionaries of arbritrary depth is saved into an hdf file.

The key, subkeys, etc. are mapped into the hdf-groups directory structure.

save_dict2hdf(fname, d)

fname: filename of output hdf file d: dictionary which contains the data

None

tropy.io_tools.hdf.**save_dict_cont**(*f*, *d*)
Recursively saves nested dictionaries.

tropy.io_tools.hdf.**update_dict_in_hdf**(*fname*, *din*)
The content of nested dictionaries of arbritrary depth is updated in an hdf file.

The key, subkeys, etc. are mapped into the hdf-groups directory structure.

update_dict_in_hdf(fname, d)

fname: filename of output hdf file d: dictionary which contains the data

None

2.2.9 tropy.io_tools.netcdf module

tropy.io_tools.netcdf.**read_icon_2d_data**(*fname*, *var_list*, *itime=0*)

tropy.io_tools.netcdf.**read_icon_4d_data**(*fname*, *var_list*, *itime=0*, *itime2=None*)
Read netcdf data.

dset = read_icon_4d_data(fname, var_list, itime = 0)

fname: netcdf filename var_list: variable name or list of variables itime (OPTIONAL): index of 1st dimension (assumed to be time) to be read only

if itime = None, the full field is read

dset: dictionary containing the fields

tropy.io_tools.netcdf.**read_icon_dimension**(*fname*, *dim_name*)

tropy.io_tools.netcdf.**read_icon_georef**(*fname*)

tropy.io_tools.netcdf.**read_icon_time**(*fname*, *itime=0*)

tropy.io_tools.netcdf.**roundTime**(*dt=None*, *roundTo=60*)

Round a datetime object to any time laps in seconds dt : datetime.datetime object, default now.

roundTo : Closest number of seconds to round to, default 1 minute. Author: Thierry Husson 2012 - Use it as you want but don't blame me.

tropy.io_tools.netcdf.**save_icon_georef**(*fname*, *geopath=None*)

tropy.io_tools.netcdf.**save_icon_time_reference**(*fname*, *outfile=None*)

2.2.10 tropy.io_tools.radolan module

2.2.11 Module contents

2.3 tropy.l15_msevi package

2.3.1 Submodules

2.3.2 tropy.l15_msevi.msevi module

2.3.3 tropy.l15_msevi.msevi_config module

2.3.4 tropy.l15_msevi.msevi_rgb module

2.3.5 Module contents

2.4 tropy.plotting_tools package

2.4.1 Submodules

2.4.2 tropy.plotting_tools.bmaps module

2.4.3 tropy.plotting_tools.colormaps module

tropy.plotting_tools.colormaps.**colorname_based_cmap**(*colname*, *start_col=None*, *final_col=None*, *reverse=False*)

A matplotlib colormap is constructed based on one color'

cmap = colorname_based_cmap(colname, start_col = None, final_col = None, reverse = False)

colname : supported colorname as basis for colormap start_col : color at one end of the colormap final_col : color at the other end of the colormap reverse: option if order is reversed

cmap: resulting colormap

tropy.plotting_tools.colormaps.**dwd_sfmap**()

tropy.plotting_tools.colormaps.**enhanced_colormap**(*vmin=200.0*, *vmed=240.0*, *vmax=300.0*)

tropy.plotting_tools.colormaps.**enhanced_wv62_cmap**(*vmin=200.0*, *vmed1=230.0*, *vmed2=240.0*, *vmax=260.0*)

tropy.plotting_tools.colormaps.**nice_cmaps**(*cmap_name*)

2.4.4 tropy.plotting_tools.compare module

tropy.plotting_tools.compare.**compare**(**args*, ***kwargs*)

The function compare can be used to fastly compare several 2d fields.

The coordinates of the x and y axis are needed. An arbitrary number of fields is plotted using the function shaded. Corresponding option keywords can be used.

tropy.plotting_tools.compare.**compare_shaded**(*x*, *y*, **args*, ***kwargs*)
The function compare can be used to fastly compare several 2d fields.

The coordinates of the x and y axis are needed. An arbitrary number of fields is plotted using the function shaded. Corresponding option keywords can be used.

2.4.5 tropy.plotting_tools.histograms module

Module for histogram calculations and plotting.

Histogram calculations include * partial normalization to get marginal distributions * conditional averages, standard deviations and percentiles

Histogram plotting is for * conditioned histograms with averages or percentiles

tropy.plotting_tools.histograms.**ave_sig_from_hist**(*xe*, *ye*, *h*)
Use output from the numpy 2d histogram routine to calculate y-mean and standard deviations along y direction.

Parameters

- **xe** (*np.array*) – x-part of histogram grid (edge values)
- **ye** (*np.array*) – y-part of histogram grid (edge values)
- **h** (*np.array*) – frequency of occurence

Returns

- **yave** (*np.array*) – mean of y weigthed by h along y-direction
- **ysig** (*np.array*) – standard deviation of y weigthed by h along y-direction

tropy.plotting_tools.histograms.**axis_average_from_hist3d**(*bins3d*, *h*, *axis=0*)
Calculates the average of a 3-dim histogram along a selected axis.

Parameters

- **bins3d** (*list of 3dim np.array*) – mesh of bin edges
- **h** (*np.array*) – absolute histogram counts
- **axis** (*{0, 1}, optional*) – axis along which the calculations are peformed

Returns **ave** – conditional average field

Return type np.array

tropy.plotting_tools.histograms.**conditioned_hist**(*h*, *axis=0*)
Make conditioned histogram.

Prob per bin (not PDF!!!).

Parameters **h** (*np.array*) – absolute frequency of occurence

Returns "normalized" frequency

Return type np.array

tropy.plotting_tools.histograms.**hist2d_scatter**(*x*, *y*, *bins=200*, ***kwargs*)
A 2d histogram is constructed and displayed as scatter plot.

Parameters

- **x** (*np.array*) – 1st data vector

- **y** (*np.array*) – 2nd data vector

Returns

- **hxy** (*np.array*) – frequency of occurence
- **xs** (*np.array*) – x-part of histogram grid (edge values)
- **ys** (*np.array*) – y-part of histogram grid (edge values)

tropy.plotting_tools.histograms.**hist3d**(*v1*, *v2*, *v3*, *bins*)

A wrapper for 3d histogram binning.

It additionally generates the average bin values with the same 3d shape as histogram itself.

Parameters

- **v1** (*np.array*) – 1st data vector
- **v2** (*np.array*) – 2nd data vector
- **v3** (*np.array*) – 3rd data vector
- **bins** (*list of 3 np.arrays*) – bins argument passed to the np.histogramdd function

Returns

- **bins3d** (*list of 3dim np.array*) – mesh of bin edges
- **h** (*np.array*) – absolute histogram counts

tropy.plotting_tools.histograms.**max_from_hist**(*xe*, *ye*, *h*)

Use output from the numpy 2d histogram routine to calculate y-positions where maximum occurences are located.

Parameters

- **xe** (*np.array*) – x-part of histogram grid (edge values)
- **ye** (*np.array*) – y-part of histogram grid (edge values)
- **h** (*np.array*) – frequency of occurence

Returns **ymax** – y-position where h is maximal along y-direction

Return type np.array

tropy.plotting_tools.histograms.**percentiles_from_hist**(*xe, ye, h, p=[25, 50, 75], axis=0, sig=0*)

Use output from the numpy 2d histogram routine to calculate percentiles of the y variable based on relative occurence rates.

Parameters

- **xe** (*np.array*) – x-part of histogram grid (edge values)
- **ye** (*np.array*) – y-part of histogram grid (edge values)
- **h** (*np.array*) – frequency of occurence
- **p** (*list, optional, default = [25, 50, 75]*) – list of percentile values to be calculated (in 100th)
- **axis** (*{0, 1}, optional*) – axis along which the calculations are peformed
- **sig** (*float, optional, default = 0*) – signa of a Gaussian filter apllied to the histogram in advance

  Should be non-negative.

Returns **yperc** – list of arrays containing the percentiles

Return type np.array

tropy.plotting_tools.histograms.**plot_cond_hist**(*xe*, *ye*, *h*, *ax*, *axis=0*, *logscale=True*, ***kwargs*)

Plot conditioned histogram (marginal distribution).

Parameters

- **xe** (*np.array*) – x-part of histogram grid (edge values)
- **ye** (*np.array*) – y-part of histogram grid (edge values)
- **h** (*np.array*) – frequency of occurence
- **ax** (*plt.axes instance*) – a current axes where the plot is placed in
- **axis** (*{0, 1}, optional*) – axis along which the calculations are peformed
- **logscale** (*{True, False}, optional*) – if histogram is plotted with logarithmic colorscale
- ****kwargs** (*dict*) – further optional keywords passed to plt.pcolormesh

Returns **pcm**

Return type plt.pcolormesh instance

tropy.plotting_tools.histograms.**plot_hist_median_and_intervals**(*xe*, *ye*, *h*, *ax*)

Plot histogram with median and IQR. (axis = 1, fixed).

Parameters

- **xe** (*np.array*) – x-part of histogram grid (edge values)
- **ye** (*np.array*) – y-part of histogram grid (edge values)
- **h** (*np.array*) – frequency of occurence
- **ax** (*plt.axes instance*) – a current axes where the plot is placed in

Returns **ax** – a current axes where the plot is placed in

Return type plt.axes instance

2.4.6 tropy.plotting_tools.meta2png module

Module designed to write an Author Name and the Source Filename into the Meta-Data of an PNG file saved with matplotlib.

tropy.plotting_tools.meta2png.**meta2png**(*fname*, *meta*)

Saves meta data into image file.

Parameters

- **fname** (*str*) – name of png file
- **meta** (*dict*) – collection of extra meta data to be stored in image file

class tropy.plotting_tools.meta2png.**pngsave**(**args*, ***kwargs*)

Bases: object

That class is designed to save pylab figures in png and add meta data.

Parameters

- ***args** (*list*) – other positional arguments passed to *plt.savefig*
- ****kwargs** (*dict*) – other optional arguments passed to *plt.savefig*‘

  ’author’ : Place yeur name into author keyword

  ’source’ [Specify your source filename if needed,] if not set, it tries to automatically find the filename

meta()
Set the meta data, esp. the Author name and Source file information.

2.4.7 tropy.plotting_tools.shaded module

A module to make shaded plots.

It contains a function for non-linear colormaps.

class tropy.plotting_tools.shaded.**nlcmap**(*cmap*, *levels*)
Bases: matplotlib.colors.LinearSegmentedColormap

A nonlinear colormap class.

Parameters

- **cmap** (*matplotlib.cmap*) – a matplotlib colormap, e.g. matplotlib.cm.jet
- **levels** (*list or np.array*) – list of values where different colors should appear

Notes

Derived from the matplotlib.colors.LinearSegmentedColormap

Copyright (c) 2006-2007, Robert Hetland <hetland@tamu.edu> Release under MIT license.

name = 'nlcmap'

tropy.plotting_tools.shaded.**set_levs**(*nmax*, *depth*, *largest=5*, *sym=True*, *sign=True*, *zmax='None'*)
Makes a non-linear level set based on pre-defined numbers.

Parameters

- **nmax** (*int*) – exponent of maximum number
- **depth** (*int*) – number of iterations used down to scales of 10**(nmax - depth)
- **largest** (*{5, 8}, optional*) – set the largest number in the base array either to 5 or 8
- **sym** (*{True, False}, optional*) – switch if levels are symmetric around origin
- **sign** (*{True, False}, optional*) – switches sign if negative
- **zmax** (*float, optional, default = 'none'*) – limiter for levels, **|levs|** > zmax are not allowed

Returns **levs** – set of non-linear levels

Return type np.array

tropy.plotting_tools.shaded.**shaded**(*x*, *y*, *z*, **args*, ***kwargs*)
The function shaded is a wrapper for the pylab function contourf.

In addition to contourf, shaded can plot filled contours for which a non-linear colormap is used.

Parameters

- **x** (*np.array*) – x-values passed to *plt.contourf*
- **y** (*np.array*) – y-values passed to *plt.contourf*
- **z** (*np.array*) – z-values passed to *plt.contourf* (color value)
- ***args** (*list*) – other positional arguments passed to *plt.contourf*
- ****kwargs** (*dict*) – other optional arguments passed to *plt.contourf*

  special keywords:

  – 'levels' : numpy array of color / contour levels

  – 'lev_depth' : gives the depth of an automatically generated level set i.e. an initial base (e.g. [1,2,3,5]) that contains the maximum value of the field (e.g. 4.3) is downscaled by 10**-1, . . . , 10**-lev_depth.

  Example: Given lev_depth = 2, for a positive field with maximum 4.3 a level set [0.01, 0.02, 0.03, 0.05, 0.1, 0.2, 0.3, 0.5, 1, 2, 3, 5] is generated.

Returns

Return type pl.contourf instance

CHAPTER 3

Submodules

3.1 tropy package

3.1.1 Submodules

tropy.MSGtools.**channel_segment_sets**(*set_name*)

Outputs a predefined set of channels and segments'

Parameters **set_name** (*str*) – name of a predefined set

Returns **chan_seg** – Dictionary of channels with a list of segments.

Return type dict

tropy.MSGtools.**get_HRIT_from_arch**(*day*, *chan_set='nc-full'*, *scan_type='pzs'*, *arch_dir=None*, *out_path=None*)

Gets and decompresses the original HRIT files from archive. A typical set of channels and segments can be chosen.

Parameters

- **day** (*datetime object*) – day and time of MSG time slot
- **chan_set** (*str, optional, default = 'nc-full'*) – name of a predefined channels set
- **scan_type** (*str, optional, default = 'pzs'*) – one of the two Meteosat scan types
- **arch_dir** (*str, optional, default = None*) – archive directory where NWC-SAF files are saved
- **out_path** (*str, optional, default = None*) – directory where the HRIT files are saved.

Returns **file_list** – list of extracted HRIT files

Return type list

tropy.MSGtools.**get_berendes_prod**(*prod*, *t*, *region='de'*, *scan_type='rss'*, *arch='none'*)

Reads product of Berendes Cloud type product for a given date.

Parameters

- **prod** (*str*) – name of Berendes product, either ccm or texture
- **t** (*datetime object*) – day and time of MSG time slot
- **region** (*str, optional, default = 'de'*) – string that defines regional cutout
- **scan_type** (*str, optional, default = 'rss'*) –
- **arch** (*str, optional, default = "none"*) – archive directory where NWC-SAF files are saved (OPTIONAL)

Returns **var** – Berendes product as field

Return type numpy array

tropy.MSGtools.**get_cmsaf_prod**(*prod*, *day*, *scan_type='rss'*, *region='eu'*, *arch='none'*, *calibrate=False*, *switch2new=True*)

Reads product of CM-SAF (KNMI retrieval) for a given date.

Parameters

- **prod** (*str*) – name of CMSAF product
- **day** (*datetime object*) – day and time of MSG time slot
- **region** (*str, optional, default = 'eu'*) – string that defines regional cutout
- **scan_type** (*str, optional, default = 'rss'*) – one of the two Meteosat scan types
- **arch** (*str, optional, default = "none"*) – archive directory where NWC-SAF files are saved
- **calibrate** (*bool, optional, default = False*) – switch for slope/offset calibration
- **switch2new** (*bool, optional, default = True*) – if the "new" set of CMSAF products should be selected "new" means collection two: "c2"

Returns

- **product** (*numpy array, optional*) – calibrated CMSAF product data (if calibrate == True)
- **counts** (*numpy array, optional*) – CMSAF product count data (if calibrate == False)
- **scal** (*float, optional*) – scale factor for CMSAF products (if calibrate == False)
- **offset** (*float, optional*) – offset for CMSAF products (if calibrate == False)

Notes

Data are saved as integer via: DATA = scal * count + offset

The following structures can be returned:

counts, scal, offset = get_cmsaf_prod(prod, day, calibrate = False)

OR

product = get_cmsaf_prod(prod, day, calibrate = True)

tropy.MSGtools.**get_cos_zen**(*day*, *scan_type='rss'*, *arch='none'*)
Reads sun zenith angle from the partical satellite hdf file and calculates the cosine of the sun zenith.

Parameters

- **day** (*datetime object*) – day and time of MSG time slot
- **scan_type** (*str, optional, default = 'rss'*) – one of the two Meteosat scan types

Returns **coszen** – cosine of sun zenith angle

Return type numpy array, optional

tropy.MSGtools.**get_highres_cma**(*t*, *region='de'*, *scan_type='rss'*, *arch='none'*)
Reads product of High-Res Cloud Mask product (Bley et al., 2013) for a given date.

Parameters

- **t** (*datetime object*) – day and time of MSG time slot
- **region** (*str, optional, default = 'de'*) – string that defines regional cutout
- **scan_type** (*str, optional, default = 'rss'*) –
- **arch** (*str, optional, default = "none"*) – archive directory where NWC-SAF files are saved (OPTIONAL)

Returns **var** – Cma product as field

Return type numpy array

tropy.MSGtools.**get_msg_lon_lat**(*region*, *scan_type='rss'*, *arch_dir='none'*)
Reads longitudes and latitudes of Meteosat8 pixels saved in an auxiliary file in the MSG archive.

Parameters

- **region** (*str*) – predefined cutout of MSG fulldisk, e.g. 'eu'
- **scan_type** (*str, optional, default = 'rss'*) – one of the two Meteosat scan types
- **arch_dir** (*str, optional, default = "none"*) – archive directory where NWC-SAF files are saved

Returns

- **lon** (*numpy array*) – longitudes of pixels
- **lat** (*numpy array*) – latitudes of pixels

tropy.MSGtools.**get_msg_lsm**(*region*, *scan_type='rss'*, *arch_dir='none'*)
Reads land sea mask of Meteosat8 pixels saved in an auxiliary file in the MSG archive.

Parameters

- **region** (*str*) – predefined cutout of MSG fulldisk, e.g. 'eu'
- **scan_type** (*str*) – which type of msg is used, *) allowed keys are: 'pzs','rss'
- **arch_dir** (*str, optional, default = "none"*) – archive directory where NWC-SAF files are saved

Returns **lsm** – land sea mask

Return type numpy array

tropy.MSGtools.**get_msg_sat_zen**(*day*, *sat_type='rss'*)
Reads satellite zenith angle from the partical satellite hdf file.

Parameters

- **day** (*datetime object*) – day and time of MSG time slot
- **sat_type** (*str, optional, default = 'rss'*) – one of the two Meteosat scan types

Returns **satzen** – satellite zenith angle

Return type numpy array

tropy.MSGtools.**get_nwcsaf_prod**(*prod*, *day*, *scan_type='rss'*, *arch='none'*, *region='eu'*, *calibrate=False*, *is_region_shifted=False*)
Reads product of NWC-SAF for a given date.

Parameters **prod** (*str*) – name of NWCSAF product prod in ['CMa', 'CMa_DUST', 'CT', 'CTTH_EFFECT', 'CTTH_HEIGHT', 'CTTH_PRESS', 'CTTH_QUALITY', 'CTTH_TEMPER', 'CT_PHASE']

day [datetime object] day and time of MSG time slot

region [str, optional, default = 'eu'] string that defines regional cutout

scan_type [str, optional, default = 'rss'] one of the two Meteosat scan types

arch [str, optional, default = "none"] archive directory where NWC-SAF files are saved

calibrate [bool, optional, default = False] switch for slope/offset calibration

is_region_shifted [bool, optional, default = False] switch if bugfix for (mistakingly) shifted regions has to be applied

Returns

- **product** (*numpy array, optional*) – calibrated NWCSAF product data (if calibrate == True)
- **counts** (*numpy array, optional*) – NWCSAF product count data (if calibrate == False)
- **scal** (*float, optional*) – scale factor for NWCSAF products (if calibrate == False)
- **offset** (*float, optional*) – offset for NWCSAF products (if calibrate == False)

Notes

Data are saved as integer via: DATA = scal * count + offset

The following structures can be returned:

counts, scal, offset = get_nwcsaf_prod(prod, day, calibrate = False)

OR

product = get_nwcsaf_prod(prod, day, calibrate = True)

tropy.MSGtools.**get_seviri_chan**(*ch_name*, *day*, *scan_type='rss'*, *arch='none'*, *calibrate=True*)
Reads MSG - SEVIRI radiance of a given channel for a given date.

Parameters

- **ch_name** (*str*) – name of MSG SEVIRI channel, e.g. ir_108 or IR_108 (case does not matter)

- **day** (*datetime object*) – day and time of MSG time slot
- **scan_type** (*str, optional, default = 'rss'*) – one of the two Meteosat scan types
- **arch** (*str, optional, default = "none"*) – archive directory where NWC-SAF files are saved
- **calibrate** (*bool, optional, default = False*) – switch for slope/offset calibration if True, radiances are output

Returns **out_field** – radiance/counts (depending on calibrate option) of channel <ch_name> at date <day>

Return type numpy array

info [dict] meta info list which includes calibration coefficients

tropy.MSGtools.**list_cmsaf_prod**(*day*, *arch='none'*)

Lists products contained in a specific CM-SAF (KNMI retrieval) file at a given date.

Parameters

- **day** (*datetime object*) – day and time of MSG time slot
- **arch** (*str, optional, default = "none"*) – archive directory where NWC-SAF files are saved

Returns **L** – list of cmsaf products

Return type list

tropy.MSGtools.**prod_reference**(*prod*)

Given a NWC-SAF product, the routine returns the correspondig substring to address the product file.

Parameters **prod** (*str*) – name of NWC-SAF product, e.g. CMa, CT, CRR

Returns **name** – substring in the NWC-SAF product file

Return type str

Notes

Further products with its corresponding substrings should be included in future.

tropy.MSGtools.**rad2bt**(*rad*, *info*)

Calculates brightness temperature of infrared channels given the radiance and info list containing calibration coefficients.

Parameters

- **rad** (*numpy array*) – radiance of an infrared channel
- **info** (*dict*) – dictionary which contains information about calibration coefficients.

Returns **bt** – brightness temperature

Return type numpy array

tropy.MSGtools.**rad2refl**(*rad*, *info*, *day*, *coszen*)

Conversion of satellite radiances in reflectances.

Parameters

- **rad** (*numpy array*) – radiance of a visible channel

- **info** (*dict*) – dictionary which contains information about calibration coefficients
- **day** (*datetime object*) – day and time of MSG time slot
- **coszen** (*numpy array*) – cosine of sun zenith angle

Returns **refl** – reflectance

Return type numpy array

tropy.MSGtools.**read_rad_from_hdf**(*hfile*)

Reads radiances from synthetic satellite calculation which are intermediately saved in an hdf file.

Parameters **hfile** (*str*) – file name of the hdf file where synthetic radiances are stored.

Returns

RAD – dictionary of radiances

the channel name, e.g. IR_108, is used as key to access the data array (e.g. saved on COSMO grid)

Return type dict

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

X