pysteps Reference

Release 0.2

Seppo Pulkkinen, Daniele Nerini and Loris Foresti

Dec 22, 2018

CONTENTS

Release
0.2

Date
Dec 22, 2018

CHAPTER

ONE

PYSTEPS REFERENCE MANUAL

This reference manual gives a detailed description of the modules, functions and objects included in pysteps.

1.1 Nowcasting methods (`pysteps.nowcasts`)

Implementations of nowcasting methods. Currently the module contains a deterministic advection extrapolation method and STEPS.

1.1.1 pysteps.nowcasts.interface

`pysteps.nowcasts.interface.`**`get_method`**`(`*name*`)`

Return a callable function for computing deterministic or ensemble precipitation nowcasts.

Implemented methods:

Name	Description
eulerian	this approach simply keeps the last observation frozen (Eulerian persistence)
lagrangian or extrapolation	this approach extrapolates the last observation following the motion field (Lagrangian persistence)
steps	implementation of the STEPS stochastic nowcasting method as described in [11], [1] and [12]

steps produces stochastic nowcasts, and the other methods are deterministic.

1.1.2 pysteps.nowcasts.extrapolation

Implementations of deterministic nowcasting methods.

`pysteps.nowcasts.extrapolation.`**`forecast`**`(`*R*, *V*, *num_timesteps*, *extrap_method='semilagrangian'*, *extrap_kwargs={}*`)`

Generate a nowcast by applying a simple advection-based extrapolation to the given precipitation field.

Parameters

R

[array-like] Two-dimensional array of shape (m,n) containing the input precipitation field.

V

[array-like] Array of shape (2,m,n) containing the x- and y-components of the advection field. The velocities are assumed to represent one time step.

num_timesteps
[int] Number of time steps to forecast.

Returns

out
[ndarray] Three-dimensional array of shape (num_timesteps,m,n) containing a time series of nowcast precipitation fields.

Other Parameters

extrap_method
[{'semilagrangian'}] Name of the extrapolation method to use. See the documentation of pysteps.extrapolation.interface.

extrap_kwargs
[dict] Optional dictionary that is supplied as keyword arguments to the extrapolation method.

See also:

`pysteps.extrapolation.interface`

1.1.3 pysteps.nowcasts.steps

Implementation of the STEPS method.

`pysteps.nowcasts.steps.`**`forecast`**(*R, V, n_timesteps, n_ens_members=24, n_cascade_levels=6, R_thr=None, kmperpixel=None, timestep=None, extrap_method='semilagrangian', decomp_method='fft', bandpass_filter_method='gaussian', noise_method='nonparametric', noise_stddev_adj=False, ar_order=2, vel_pert_method=None, conditional=False, use_precip_mask=True, use_probmatching=True, mask_method='incremental', callback=None, return_output=True, seed=None, num_workers=None, fft_method='numpy', extrap_kwargs={}, filter_kwargs={}, noise_kwargs={}, vel_pert_kwargs={}*)

Generate a nowcast ensemble by using the Short-Term Ensemble Prediction System (STEPS) method.

Parameters

R
[array-like] Array of shape (ar_order+1,m,n) containing the input precipitation fields ordered by timestamp from oldest to newest. The time steps between the inputs are assumed to be regular, and the inputs are required to have finite values.

V
[array-like] Array of shape (2,m,n) containing the x- and y-components of the advection field. The velocities are assumed to represent one time step between the inputs. All values are required to be finite.

n_timesteps
[int] Number of time steps to forecast.

n_ens_members
[int] The number of ensemble members to generate.

n_cascade_levels
[int] The number of cascade levels to use.

Returns

out
[ndarray] If return_output is True, a four-dimensional array of shape (n_ens_members,n_timesteps,m,n) containing a time series of forecast precipitation fields for each ensemble member. Otherwise, a None value is returned.

Other Parameters

R_thr
[float] Specifies the threshold value for minimum observable precipitation intensity. Must be set if use_probmatching is True or conditional is True.

kmperpixel
[float] Spatial resolution of the input data (kilometers/pixel). Required if vel_pert_method is not None or mask_method is 'incremental'.

timestep
[float] Time step of the motion vectors (minutes). Required if vel_pert_method is not None or mask_method is 'incremental'.

extrap_method
[{'semilagrangian'}] Name of the extrapolation method to use. See the documentation of pysteps.extrapolation.interface.

decomp_method
[{'fft'}] Name of the cascade decomposition method to use. See the documentation of pysteps.cascade.interface.

bandpass_filter_method
[{'gaussian', 'uniform'}] Name of the bandpass filter method to use with the cascade decomposition. See the documentation of pysteps.cascade.interface.

noise_method
[{'parametric','nonparametric','ssft','nested'}] Name of the noise generator to use for perturbating the precipitation field. See the documentation of pysteps.noise.interface.

noise_stddev_adj
[bool] Optional adjustment for the standard deviations of the noise fields added to each cascade level. See pysteps.noise.utils.compute_noise_stddev_adjs.

ar_order
[int] The order of the autoregressive model to use. Must be >= 1.

vel_pert_method
[{'bps'}] Name of the noise generator to use for perturbing the velocity field. See the documentation of pysteps.noise.interface.

conditional
[bool] If set to True, compute the statistics of the precipitation field conditionally by excluding the areas where the values are below the threshold R_thr.

use_precip_mask
[bool] If True, set pixels outside precipitation areas to the minimum value of the observed field.

mask_method
[{'obs', 'sprog', 'incremental'}] The precipitation/no precipitation method to use with mask: 'obs' = apply R_thr to the most recently observed precipitation intensity field, 'sprog' = use the smoothed forecast field from S-PROG, where the AR(p) model has been applied, 'incremental' = iteratively buffer the mask with a certain rate (currently it is 1 km/min)

use_probmatching
[bool] If True, apply probability matching to the forecast field in order to preserve the distribution of the most recently observed precipitation field.

callback
[function] Optional function that is called after computation of each time step of the nowcast. The function takes one argument: a three-dimensional array of shape (n_ens_members,h,w), where h and w are the height and width of the input field R, respectively. This can be used, for instance, writing the outputs into files.

return_output
[bool] Set to False to disable returning the outputs as numpy arrays. This can save memory if the intermediate results are written to output files using the callback function.

seed
[int] Optional seed number for the random generators.

num_workers
[int] The number of workers to use for parallel computation. Set to None to use all available CPUs. Applicable if dask is enabled.

fft_method
[str or tuple] A string or a (function,kwargs) tuple defining the FFT method to use (see utils.fft.get_method). Defaults to "numpy".

extrap_kwargs
[dict] Optional dictionary that is supplied as keyword arguments to the extrapolation method.

filter_kwargs
[dict] Optional dictionary that is supplied as keyword arguments to the filter method.

noise_kwargs
[dict] Optional dictionary that is supplied as keyword arguments to the initializer of the noise generator.

vel_pert_kwargs
[dict] Optional dictionary that is supplied as keyword arguments to the initializer of the velocity perturbator.

See also:

`pysteps.extrapolation.interface`, `pysteps.cascade.interface`, `pysteps.noise.interface`, `pysteps.noise.utils.compute_noise_stddev_adjs`

References

[11], [1], [12]

1.2 Input/output routines (`pysteps.io`)

Methods for browsing data archives, reading 2d precipitation fields and writing forecasts into files.

1.2.1 pysteps.io.interface

`pysteps.io.interface.get_method(name, type)`
Return a callable function for the method corresponding to the given name.

Parameters

name
[str] Name of the method. The available options are:

Importers:

Name	Description
bom_rf3	NetCDF files in the Bureau of Meterology (BoM) archive containing precipitation intensity composites
fmi_pgm	PGM files in the Finnish Meteorological Institute (FMI) archive containing reflectivity composites (dBZ)
mch_gif	GIF files in the MeteoSwiss archive containing precipitation composites
mch_hdf5	HDF5 file format used by MeteoSiss
mch_metranet	metranet files in the MeteoSwiss archive containing precipitation composites
odim_hdf5	ODIM HDF5 file format used by Eumetnet/OPERA

Exporters:

Name	Description
netcdf	NetCDF files conforming to the CF 1.7 specification

type
[str] Type of the method. The available options are 'importer' and 'exporter'.

1.2.2 pysteps.io.archive

Utilities for finding archived files that match the given criteria.

`pysteps.io.archive.`**`find_by_date`**(*date*, *root_path*, *path_fmt*, *fn_pattern*, *fn_ext*, *timestep*, *num_prev_files=0*, *num_next_files=0*)
List input files whose timestamp matches the given date.

Parameters

date
[datetime.datetime] The given date.

root_path
[str] The root path to search the input files.

path_fmt
[str] Path format. It may consist of directory names separated by '/' and date/time specifiers beginning with '%' (e.g. %Y/%m/%d).

fn_pattern
[str] The name pattern of the input files without extension. The pattern can contain time specifiers (e.g. %H, %M and %S).

fn_ext
[str] Extension of the input files.

timestep
[float] Time step between consecutive input files (minutes).

num_prev_files
[int] Optional, number of previous files to find before the given timestamp.

num_next_files
[int] Optional, number of future files to find after the given timestamp.

Returns

out
[tuple] If num_prev_files=0 and num_next_files=0, return a pair containing the found file name and the corresponding timestamp as a datetime.datetime object. Otherwise, return a tuple of two lists, the first one for the file names and the second one for the corresponding timestemps. The lists are sorted in ascending order with respect to timestamp. A None value is assigned if a file name corresponding to a given timestamp is not found.

1.2.3 pysteps.io.importers

`import_bom_rf3`(filename, **kwargs)	Import a NetCDF radar rainfall product from the BoM Rainfields3.
`import_fmi_pgm`(filename, **kwargs)	Import a 8-bit PGM radar reflectivity composite from the FMI archive.
`import_mch_gif`(filename, **kwargs)	Import a 8-bit gif radar reflectivity composite from the MeteoSwiss archive.
`import_odim_hdf5`(filename, **kwargs)	Read a precipitation field (and optionally the quality field) from a HDF5 file conforming to the ODIM specification.

Methods for importing files containing 2d precipitation fields.

The methods in this module implement the following interface:

import_xxx(filename, optional arguments)

where xxx is the name (or abbreviation) of the file format and filename is the name of the input file.

The output of each method is a three-element tuple containing the two-dimensional precipitation field, the corresponding quality field and a metadata dictionary. If the file contains no quality information, the quality field is set to None. Pixels containing missing data are set to nan.

The metadata dictionary contains the following mandatory key-value pairs:

Key	Value
projection	PROJ.4-compatible projection definition
x1	x-coordinate of the lower-left corner of the data raster (meters)
y1	y-coordinate of the lower-left corner of the data raster (meters)
x2	x-coordinate of the upper-right corner of the data raster (meters)
y2	y-coordinate of the upper-right corner of the data raster (meters)
xpixelsize	grid resolution in x-direction (meters)
ypixelsize	grid resolution in y-direction (meters)
yorigin	a string specifying the location of the first element in the data raster w.r.t. y-axis: ‘upper’ = upper border ‘lower’ = lower border
institution	name of the institution who provides the data
timestep	time step of the input data (minutes)
unit	the physical unit of the data: ‘mm/h’, ‘mm’ or ‘dBZ’
transform	the transformation of the data: None, ‘dB’, ‘Box-Cox’ or others
accutime	the accumulation time in minutes of the data, float
threshold	the rain/no rain threshold with the same unit, transformation and accutime of the data.
zerovalue	the value assigned to the no rain pixels with the same unit, transformation and accutime of the data.

`pysteps.io.importers.`**`import_bom_rf3`**(*filename*, ***kwargs*)
Import a NetCDF radar rainfall product from the BoM Rainfields3.

Parameters

filename
[str] Name of the file to import.

Returns

out
[tuple] A three-element tuple containing the rainfall field in mm/h imported from the Bureau RF3 netcdf, the quality field and the metadata. The quality field is currently set to None.

`pysteps.io.importers.`**`import_fmi_pgm`**(*filename*, ***kwargs*)
Import a 8-bit PGM radar reflectivity composite from the FMI archive.

Parameters

filename
[str] Name of the file to import.

Returns

out
[tuple] A three-element tuple containing the reflectivity composite in dBZ and the associated quality field and metadata. The quality field is currently set to None.

Other Parameters

gzipped
[bool] If True, the input file is treated as a compressed gzip file.

`pysteps.io.importers.`**`import_mch_gif`**(*filename*, ***kwargs*)
Import a 8-bit gif radar reflectivity composite from the MeteoSwiss archive.

Parameters

filename
[str] Name of the file to import.

Returns

out
[tuple] A three-element tuple containing the precipitation field in mm/h imported from a MeteoSwiss gif file and the associated quality field and metadata. The quality field is currently set to None.

Other Parameters

product
[string] The name of the MeteoSwiss QPE product:

Name	Product
AQC	Acquire
CPC	CombiPrecip
RZC	Precip

unit
[string] the physical unit of the data: 'mm/h', 'mm' or 'dBZ'

accutime
[float] the accumulation time in minutes of the data

`pysteps.io.importers.`**`import_mch_hdf5`**(*filename*, ***kwargs*)
Read a precipitation field (and optionally the quality field) from a HDF5 file conforming to the ODIM specification.

Parameters

filename
[str] Name of the file to import.

Returns

out
[tuple] A three-element tuple containing the OPERA product for the requested quantity and the associated quality field and metadata. The quality field is read from the file if it contains a dataset whose quantity identifier is 'QIND'.

Other Parameters

qty
[{'RATE', 'ACRR', 'DBZH'}] The quantity to read from the file. The currently supported identitiers are: 'RATE'=instantaneous rain rate (mm/h), 'ACRR'=hourly rainfall accumulation (mm) and 'DBZH'=max-reflectivity (dBZ). The default value is 'RATE'.

`pysteps.io.importers.`**`import_mch_metranet`**(*filename*, ***kwargs*)
Import a 8-bit bin radar reflectivity composite from the MeteoSwiss archive.

Parameters

filename
[str] Name of the file to import.

Returns

out
[tuple] A three-element tuple containing the precipitation field in mm/h imported from a MeteoSwiss gif file and the associated quality field and metadata. The quality field is currently set to None.

Other Parameters

product
[string] The name of the MeteoSwiss QPE product:

Name	Product
AQC	Acquire
CPC	CombiPrecip
RZC	Precip

unit
[string] the physical unit of the data: 'mm/h', 'mm' or 'dBZ'

accutime
[float] the accumulation time in minutes of the data

`pysteps.io.importers.`**`import_odim_hdf5`**(*filename*, ***kwargs*)
Read a precipitation field (and optionally the quality field) from a HDF5 file conforming to the ODIM specification.

Parameters

filename
[str] Name of the file to import.

Returns

out
[tuple] A three-element tuple containing the OPERA product for the requested quantity and the associated quality field and metadata. The quality field is read from the file if it contains a dataset whose quantity identifier is 'QIND'.

Other Parameters

qty
[{'RATE', 'ACRR', 'DBZH'}] The quantity to read from the file. The currently supported identitiers are: 'RATE'=instantaneous rain rate (mm/h), 'ACRR'=hourly rainfall accumulation (mm) and 'DBZH'=max-reflectivity (dBZ). The default value is 'RATE'.

1.2.4 pysteps.io.readers

Methods for reading files.

`pysteps.io.readers.`**`read_timeseries`**(*inputfns*, *importer*, ***kwargs*)
Read a time series of input files using the methods implemented in the importers module and stack them into a 3d array of shape (num_timesteps, height, width).

Parameters

inputfns
[tuple] Input files returned by a function implemented in the archive module.

importer
[function] A function implemented in the importers module.

kwargs
[dict] Optional keyword arguments for the importer.

Returns

out
[tuple] A three-element tuple containing the read data and quality rasters and associated metadata. If an input file name is None, the corresponding precipitation and quality fields are filled with nan values. If all input file names are None or if the length of the file name list is zero, a three-element tuple containing None values is returned.

1.2.5 pysteps.io.exporters

`initialize_forecast_exporter_netcdf`(. . . [, . . .])	Initialize a netCDF forecast exporter.
`export_forecast_dataset`(F, exporter)	Write a forecast array into a file.
`close_forecast_file`(exporter)	Finish writing forecasts and close the file associated with a forecast exporter.

Methods for writing forecasts of 2d precipitation fields into various file formats.

Each exporter method in this module has its own initialization function that implements the following interface:

initialize_forecast_exporter_xxx(filename, startdate, timestep, num_timesteps, shape, num_ens_members, metadata, incremental=None)

where xxx describes the file format. This function creates the file and writes the metadata. The datasets are written by calling export_forecast_dataset, and the file is closed by calling close_forecast_file.

The arguments in the above are defined as follows:

Argument	Type/values	Description
filename	str	name of the output file
startdate	datetime.datetime	start date of the forecast
timestep	int	time step of the forecast (minutes)
n_timesteps	int	number of time steps in the forecast this argument is ignored if incremental is set to 'timestep'.
shape	tuple	two-element tuple defining the shape (height,width) of the forecast grids
n_ens_members	int	number of ensemble members in the forecast this argument is ignored if incremental is set to 'member'
metadata	dict	metadata dictionary containing the projection,x1,x2,y1,y2 and unit attributes described in the documentation of pysteps.io.importers
incremental	{'timestep', 'member'}	Allow incremental writing of datasets into the netCDF file the available options are: 'timestep' = write a forecast or a forecast ensemble for a given time step 'member' = write a forecast sequence for a given ensemble member

The return value is a dictionary containing an exporter object. This can be used with export_forecast_dataset to write datasets into the netCDF file.

`pysteps.io.exporters.`**`close_forecast_file`**(*exporter*)

Finish writing forecasts and close the file associated with a forecast exporter.

Parameters

exporter
[dict] An exporter object created with any initialization method implemented in this module.

`pysteps.io.exporters.`**`export_forecast_dataset`**(*F*, *exporter*)

Write a forecast array into a file. The written dataset has dimensions (num_ens_members,num_timesteps,shape[0],shape[1]), where shape refers to the shape of the two-dimensional forecast grids. If the exporter was initialized with incremental!=None, the array is appended to the existing dataset either along the ensemble member or time axis.

Parameters

exporter
[dict] An exporter object created with any initialization method implemented in this module.

F
[array_like] The array to write. The required shape depends on the choice of the 'incremental' parameter the exporter was initialized with:

incremental	required shape
None	(num_ens_members,num_timesteps,shape[0],shape[1])
'timestep'	(num_ens_members,shape[0],shape[1])
'member'	(num_timesteps,shape[0],shape[1])

`pysteps.io.exporters.`**`initialize_forecast_exporter_netcdf`**(*filename*, *startdate*, *timestep*, *n_timesteps*, *shape*, *n_ens_members*, *metadata*, *incremental=None*)

Initialize a netCDF forecast exporter.

`pysteps.io.exporters.`**`initialize_forecast_exporter_odimhdf5`**(*filename*, *startdate*, *timestep*, *n_timesteps*, *shape*, *n_ens_members*, *metadata*, *incremental=None*)

Initialize a ODIM HDF5 forecast exporter.

1.2.6 pysteps.io.nowcast_importers

Methods for importing nowcast files.

The methods in this module implement the following interface:

import_xxx(filename, optional arguments)

where xxx is the name (or abbreviation) of the file format and filename is the name of the input file.

The output of each method is a two-element tuple containing the nowcast array and a metadata dictionary.

The metadata dictionary contains the following mandatory key-value pairs:

Key	Value
projection	PROJ.4-compatible projection definition
x1	x-coordinate of the lower-left corner of the data raster (meters)
y1	y-coordinate of the lower-left corner of the data raster (meters)
x2	x-coordinate of the upper-right corner of the data raster (meters)
y2	y-coordinate of the upper-right corner of the data raster (meters)
xpixelsize	grid resolution in x-direction (meters)
ypixelsize	grid resolution in y-direction (meters)
yorigin	a string specifying the location of the first element in the data raster w.r.t. y-axis: 'upper' = upper border 'lower' = lower border
institution	name of the institution who provides the data
timestep	time step of the input data (minutes)
unit	the physical unit of the data: 'mm/h', 'mm' or 'dBZ'
transform	the transformation of the data: None, 'dB', 'Box-Cox' or others
accutime	the accumulation time in minutes of the data, float
threshold	the rain/no rain threshold with the same unit, transformation and accutime of the data.
zerovalue	it is the value assigned to the no rain pixels with the same unit, transformation and accutime of the data.

`pysteps.io.nowcast_importers.`**`import_netcdf_pysteps`**(*filename*, ***kwargs*)

Read a nowcast or a nowcast ensemble from a NetCDF file conforming to the CF 1.7 specification.

1.3 Optical flow methods (`pysteps.motion`)

Implementations of optical flow methods.

1.3.1 pysteps.motion.interface

`pysteps.motion.interface.get_method(name)`

Return a callable function for the optical flow method corresponding to the given name. The available options are:

Python-based implementations	
Name	Description
None	Returns a zero motion field
lucaskanade	OpenCV implementation of the Lucas-Kanade method with interpolated motion vectors for areas with no precipitation.
darts	Implementation of the DARTS method of Ruzanski et al.
vet	Implementation of the VET method of Laroche and Zawadzki (1995) and Germann and Zawadzki (2002)

Methods implemented in C (these require separate compilation and linkage)	
Name	Description
brox	implementation of the variational method of Brox et al. (2004) from IPOL (http://www.ipol.im/pub/art/2013/21)
clg	implementation of the Combined Local-Global (CLG) method of Bruhn et al., 2005 from IPOL (http://www.ipol.im/pub/art/2015/44)

1.3.2 pysteps.motion.darts

Implementation of the DARTS algorithm.

`pysteps.motion.darts.DARTS(Z, **kwargs)`

Compute the advection field from a sequence of input images by using the DARTS method.

Parameters

Z
[array-like] Array of shape (T,m,n) containing a sequence of T two-dimensional input images of shape (m,n).

Returns

out
[ndarray] Three-dimensional array (2,H,W) containing the dense x- and y-components of the motion field.

Other Parameters

N_x
[int] Number of DFT coefficients to use for the input images, x-axis (default=50).

N_y
[int] Number of DFT coefficients to use for the input images, y-axis (default=50).

N_t
[int] Number of DFT coefficients to use for the input images, time axis (default=4). N_t must be strictly smaller than T.

M_x
[int] Number of DFT coefficients to compute for the output advection field, x-axis (default=2).

M_y
[int] Number of DFT coefficients to compute for the output advection field, y-axis (default=2).

print_info
[bool] If True, print information messages.

lsq_method
[{1, 2}] The method to use for solving the linear equations in the least squares sense: 1=numpy.linalg.lstsq, 2=explicit computation of the Moore-Penrose pseudoinverse and SVD.

verbose
[bool] if set to True, it prints information about the program

References

[10]

1.3.3 pysteps.motion.lucaskanade

dense_lucaskanade(R, **kwargs)	OpenCV implementation of the Lucas-Kanade method with interpolated motion vectors for areas with no precipitation.

OpenCV implementation of the Lucas-Kanade method with interpolated motion vectors for areas with no precipitation.

`pysteps.motion.lucaskanade.`**`dense_lucaskanade`**`(`*R*, ***kwargs*`)`
OpenCV implementation of the Lucas-Kanade method with interpolated motion vectors for areas with no precipitation.

Parameters

R
[array-like, shape (t,m,n)] array containing the input precipitation fields, no missing values are accepted

Returns

out
[ndarray, shape (2,m,n)] three-dimensional array containing the dense x- and y-components of the motion field. Return an empty UV array when any sparse vectors is left to be interpolated.

Other Parameters

max_corners_ST
[int] maximum number of corners to return. If there are more corners than are found, the strongest of them is returned

quality_level_ST
[float] parameter characterizing the minimal accepted quality of image corners. See original documentation for more details (https://docs.opencv.org)

min_distance_ST
[int] minimum possible Euclidean distance between the returned corners [px]

block_size_ST
[int] size of an average block for computing a derivative covariation matrix over each pixel neighborhood

winsize_LK
[int] size of the search window at each pyramid level. Small windows (e.g. 10) lead to unrealistic motion

nr_levels_LK
[int] 0-based maximal pyramid level number. Not very sensitive parameter

nr_IQR_outlier
[int] nr of IQR above median to consider the velocity vector as outlier and discard it

size_opening
[int] the structuring element size for the filtering of isolated pixels [px]

decl_grid
[int] size of the declustering grid [px]

min_nr_samples
[int] the minimum number of samples for computing the median within given declustering cell

function
[string] the radial basis function, based on the Euclidian norm d, used in the interpolation of the sparse vectors. default : inverse available : nearest, inverse, gaussian

k
[int or "all"] the number of nearest neighbors used to speed-up the interpolation If set equal to "all", it employs all the sparse vectors default : 20

epsilon
[float] adjustable constant for gaussian or inverse functions default : median distance between sparse vectors

nchunks
[int] split the grid points in n chunks to limit the memory usage during the interpolation default : 5

extra_vectors
[array-like] additional sparse motion vectors as 2d array (columns: x,y,u,v; rows: nbr. of vectors) to be integrated with the sparse vectors from the Lucas-Kanade local tracking. x and y must be in pixel coordinates, with (0,0) being the upper-left corner of the field R. u and v must be in pixel units

verbose
[bool] if set to True, it prints information about the program

1.3.4 pysteps.motion.vet

vet(input_images[, sectors, smooth_gain, . . .])	Variational Echo Tracking Algorithm presented in Laroche and Zawadzki (1995) and used in the McGill Algorithm for Prediction by Lagrangian Extrapolation (MAPLE) described in Germann and Zawadzki (2002).

Variational Echo Tracking (VET) Module

This module implements the VET algorithm presented by Laroche and Zawadzki (1995) and used in the McGill

Algorithm for Prediction by Lagrangian Extrapolation (MAPLE) described in Germann and Zawadzki (2002).

The morphing and the cost functions are implemented in Cython and parallelized for performance.

`pysteps.motion.vet.`**`ceil_int`**(*scalar*)

Round number to nearest integer. Returns and integer value.

`pysteps.motion.vet.`**`get_padding`**(*dimension_size*, *sectors*)

Get the padding at each side of the one dimensions of the image so the new image dimensions are divided evenly in the number of *sectors* specified.

Parameters

dimension_size
[int] Actual dimension size.

sectors
[int] number of sectors over which the the image will be divided.

`pysteps.motion.vet.`**`morph`**(*image*, *displacement*, *gradient=False*)

Morph image by applying a displacement field (Warping).

The new image is created by selecting for each position the values of the input image at the positions given by the x and y displacements. The routine works in a backward sense. The displacement vectors have to refer to their destination.

For more information in Morphing functions see Section 3 in Beezley and Mandel (2008).

Beezley, J. D., & Mandel, J. (2008). Morphing ensemble Kalman filters. Tellus A, 60(1), 131-140.

The displacement field in x and y directions and the image must have the same dimensions.

The morphing is executed in parallel over x axis.

The value of displaced pixels that fall outside the limits takes the value of the nearest edge. Those pixels are indicated by values greater than 1 in the output mask.

Parameters

image
[ndarray (ndim = 2)] Image to morph

displacement
[ndarray (ndim = 3)] Displacement field to be applied (Warping). The first dimension corresponds to the coordinate to displace.

The dimensions are: displacement [i/x (0) or j/y (1) ,

i index of pixel, j index of pixel]

gradient
[bool, optional] If True, the gradient of the morphing function is returned.

Returns

image
[ndarray (float64 ,ndim = 2)] Morphed image.

mask
[ndarray (int8 ,ndim = 2)] Invalid values mask. Points outside the boundaries are masked. Values greater than 1, indicate masked values.

gradient_values
[ndarray (float64 ,ndim = 3), optional] If gradient keyword is True, the gradient of the function is also returned.

`pysteps.motion.vet.round_int(scalar)`

Round number to nearest integer. Returns and integer value.

`pysteps.motion.vet.vet(input_images, sectors=((32, 16, 4, 2), (32, 16, 4, 2)), smooth_gain=1000000.0, first_guess=None, intermediate_steps=False, verbose=True, indexing='yx', padding=0, options=None)`

Variational Echo Tracking Algorithm presented in Laroche and Zawadzki (1995) and used in the McGill Algorithm for Prediction by Lagrangian Extrapolation (MAPLE) described in Germann and Zawadzki (2002).

This algorithm computes the displacement field between two images (the input_image with respect to the template image). The displacement is sought by minimizing sum of the residuals of the squared differences of the images pixels and the contribution of a smoothness constrain.

In order to find the minimum an scaling guess procedure is applied, from larger scales to a finer scale. This reduces the changes that the minimization procedure converges to a local minimum. The scaling guess is defined by the scaling sectors (see **sectors** keyword).

The smoothness of the returned displacement field is controlled by the smoothness constrain gain (**smooth_gain** keyword).

If a first guess is not given, zero displacements are used as first guess.

To minimize the cost function, the **'scipy minimization'_** function is used with the 'CG' method. This method proved to give the best results under any different conditions and is the most similar one to the original VET implementation in Laroche and Zawadzki (1995).

The method CG uses a nonlinear conjugate gradient algorithm by Polak and Ribiere, a variant of the Fletcher-Reeves method described in Nocedal and Wright (2006), pp. 120-122.

Parameters

input_images
[ndarray or MaskedArray] Input images, sequence of 2D arrays, or 3D arrays. The first dimension represents the images time dimension.

The template_image (first element in first dimensions) denotes the reference image used to obtain the displacement (2D array). The second is the target image.

The expected dimensions are (2,ni,nj).

sectors
[list or array, optional] The number of sectors for each dimension used in the scaling procedure. If dimension is 1, the same sectors will be used both image dimensions (x and y). If is 2D, the each row determines the sectors of the each dimension.

smooth_gain
[float, optional] Smooth gain factor

first_guess
[ndarray, **optional_**] The shape of the first guess should have the same shape as the initial sectors shapes used in the scaling procedure. If first_guess is not present zeros are used as first guess.

E.g.:
If the first sector shape in the scaling procedure is (ni,nj), then the first_guess should have (2, ni, nj) shape.

intermediate_steps
[bool, optional] If True, also return a list with the first guesses obtained during the scaling procedure. False, by default.

verbose
[bool, optional] Verbosity enabled if True (default).

indexing
[str, optional] Input indexing order.'ij' and 'xy' indicates that the dimensions of the input are (time, longitude, latitude), while 'yx' indicates (time, latitude, longitude). The displacement field dimensions are ordered accordingly in a way that the first dimension indicates the displacement along x (0) or y (1). That is, UV displacements are always returned.

padding
[int] Padding width in grid points. A border is added to the input array to reduce the effects of the minimization at the border.

options
[dict, optional] A dictionary of solver options. See **'scipy minimization'_** function for more details.

.. _'scipy minimization'
[https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html]

Returns

displacement_field
[ndarray] Displacement Field (2D array representing the transformation) that warps the template image into the input image. The dimensions are (2,ni,nj), where the first dimension indicates the displacement along x (0) or y (1).

intermediate_steps
[list of ndarray] List with the first guesses obtained during the scaling procedure.

References

Laroche, S., and I. Zawadzki, 1995: Retrievals of horizontal winds from single-Doppler clear-air data by methods of cross-correlation and variational analysis. J. Atmos. Oceanic Technol., 12, 721–738. doi: http://dx.doi.org/10.1175/1520-0426(1995)012<0721:ROHWFS>2.0.CO;2

Germann, U. and I. Zawadzki, 2002: Scale-Dependence of the Predictability of Precipitation from Continental Radar Images. Part I: Description of the Methodology. Mon. Wea. Rev., 130, 2859–2873, doi: 10.1175/1520-0493(2002)130<2859:SDOTPO>2.0.CO;2.

Nocedal, J, and S J Wright. 2006. Numerical Optimization. Springer New York.

`pysteps.motion.vet.`**`vet_cost_function`**(*sector_displacement_1d*, *input_images*, *blocks_shape*, *mask*, *smooth_gain*, *debug=False*, *gradient=False*)

Variational Echo Tracking Cost Function.

This function is designed to be used with the scipy.optimize.minimize

The function first argument is the variable to be used in the minimization procedure.

The sector displacement must be a flat array compatible with the dimensions of the input image and sectors shape (see parameters section below for more details).

Parameters

sector_displacement_1d
[ndarray] Array of displacements to apply to each sector. The dimensions are: sector_displacement_2d [x (0) or y (1) displacement, i index of sector, j index of sector]. The shape of the sector displacements must be compatible with the input image and the block shape. The shape should be (2, mx, my) where mx and my are the numbers of sectors in the x and the y dimension.

input_images
[ndarray] Input images, sequence of 2D arrays, or 3D arrays. The first dimension represents the images time dimension.

The template_image (first element in first dimensions) denotes the reference image used to obtain the displacement (2D array). The second is the target image.

The expected dimensions are (2,nx,ny). Be aware the the 2D images dimensions correspond to (lon,lat) or (x,y).

blocks_shape
[ndarray (ndim=2)] Number of sectors in each dimension (x and y). blocks_shape.shape = (mx,my)

mask
[ndarray (ndim=2)] Data mask. If is True, the data is marked as not valid and is not used in the computations.

smooth_gain
[float] Smoothness constrain gain

debug
[bool, optional] If True, print debugging information.

gradient
[bool, optional] If True, the gradient of the morphing function is returned.

Returns

penalty or gradient values.

penalty
[float] Value of the cost function

gradient_values
[ndarray (float64 ,ndim = 3), optional] If gradient keyword is True, the gradient of the function is also returned.

1.4 Advection-based extrapolation (`pysteps.extrapolation`)

Methods for advection-based extrapolation of precipitation fields. Currently the module contains an implementation of the semi-Lagrangian method described in [5].

1.4.1 pysteps.extrapolation.interface

`pysteps.extrapolation.interface.get_method(name)`
Return a callable function for the extrapolation method corresponding to the given name. The available options are:

Name	Description
None	returns None
eulerian	this methods does not apply any advection to the input precipitation field (Eulerian persistence)
semilagrangian	implementation of the semi-Lagrangian method of Germann et al. (2002)

1.4.2 pysteps.extrapolation.semilagrangian

Implementation of the semi-Lagrangian method of Germann et al (2002).

`pysteps.extrapolation.semilagrangian.`**`extrapolate`**(*R*, *V*, *num_timesteps*, *outval=nan*, ***kwargs*)

Apply semi-Lagrangian extrapolation to a two-dimensional precipitation field.

Parameters

R
[array-like] Array of shape (m,n) containing the input precipitation field. All values are required to be finite.

V
[array-like] Array of shape (2,m,n) containing the x- and y-components of the m*n advection field. All values are required to be finite.

num_timesteps
[int] Number of time steps to extrapolate.

outval
[float] Optional argument for specifying the value for pixels advected from outside the domain. If outval is set to 'min', the value is taken as the minimum value of R. Default : np.nan

Returns

out
[array or tuple] If return_displacement=False, return a time series extrapolated fields of shape (num_timesteps,m,n). Otherwise, return a tuple containing the extrapolated fields and the total displacement along the advection trajectory.

Other Parameters

D_prev
[array-like] Optional initial displacement vector field of shape (2,m,n) for the extrapolation. Default : None

n_iter
[int] Number of inner iterations in the semi-Lagrangian scheme. Default : 3

inverse
[bool] If True, the extrapolation trajectory is computed backward along the flow (default), forward otherwise. Default : True

return_displacement
[bool] If True, return the total advection velocity (displacement) between the initial input field and the advected one integrated along the trajectory. Default : False

References

[5]

1.5 Scale-based decomposition of precipitation fields (`pysteps.cascade`)

Methods for constructing bandpass filters and decomposing 2d precipitation fields into different spatial scales.

1.5.1 pysteps.cascade.interface

`pysteps.cascade.interface.get_method(name)`

Return a callable function for the bandpass filter or decomposition method corresponding to the given name.

Filter methods:

Name	Description
gaussian	implementation of a bandpass filter using Gaussian weights
uniform	implementation of a filter where all weights are set to one

Decomposition methods:

Name	Description
fft	decomposition based on Fast Fourier Transform (FFT) and a bandpass filter

1.5.2 pysteps.cascade.bandpass_filters

filter_uniform(shape, n)	A dummy filter with one frequency band covering the whole domain.
filter_gaussian(shape, n[, l_0, …])	Implements a set of Gaussian bandpass filters in logarithmic frequency scale.

Bandpass filters for separating different spatial scales from two-dimensional images in the frequency domain.

The methods in this module implement the following interface:

filter_xxx(shape, n, optional arguments)

where shape is the shape of the input field, respectively, and n is the number of frequency bands to use.

The output of each filter function is a dictionary containing the following key-value pairs:

Key	Value
weights_1d	2d array of shape (n, r) containing 1d filter weights for each frequency band k=1,2,…,n
weights_2d	3d array of shape (n, M, int(N/2)+1) containing the 2d filter weights for each frequency band k=1,2,…,n
central_freqs	1d array of shape n containing the central frequencies of the filters

where r = int(max(N, M)/2)+1

The filter weights are assumed to be normalized so that for any Fourier wavenumber they sum to one.

`pysteps.cascade.bandpass_filters.filter_gaussian(shape, n, l_0=3, gauss_scale=0.5, gauss_scale_0=0.5)`

Implements a set of Gaussian bandpass filters in logarithmic frequency scale.

Parameters

shape
[int or tuple] The dimensions (height, width) of the input field. If shape is an int, the domain is assumed to have square shape.

n
[int] The number of frequency bands to use. Must be greater than 2.

l_0
[int] Central frequency of the second band (the first band is always centered at zero).

gauss_scale
[float] Optional scaling prameter. Proportional to the standard deviation of the Gaussian weight functions.

gauss_scale_0
[float] Optional scaling parameter for the Gaussian function corresponding to the first frequency band.

Returns

out
[dict] A dictionary containing the bandpass filters corresponding to the specified frequency bands.

References

[8]

`pysteps.cascade.bandpass_filters.`**`filter_uniform`**(*shape*, *n*)
A dummy filter with one frequency band covering the whole domain. The weights are set to one.

Parameters

shape
[int or tuple] The dimensions (height, width) of the input field. If shape is an int, the domain is assumed to have square shape.

n
[int] Not used. Needed for compatibility with the filter interface.

1.5.3 pysteps.cascade.decomposition

Methods for decomposing two-dimensional images into multiple spatial scales.

The methods in this module implement the following interface:

decomposition_xxx(X, filter, **kwargs)

where X is the input field and filter is a dictionary returned by a filter method implemented in bandpass_filters.py. Optional parameters can be passed in the keyword arguments. The output of each method is a dictionary with the following key-value pairs:

Key	Value
cascade_levels	three-dimensional array of shape (k,m,n), where k is the number of cascade levels and the input fields have shape (m,n)
means	list of mean values for each cascade level
stds	list of standard deviations for each cascade level

`pysteps.cascade.decomposition.`**`decomposition_fft`**(*X*, *filter*, ***kwargs*)
Decompose a 2d input field into multiple spatial scales by using the Fast Fourier Transform (FFT) and a bandpass filter.

Parameters

X
[array_like] Two-dimensional array containing the input field. All values are required to be finite.

filter
[dict] A filter returned by a method implemented in bandpass_filters.py.

Returns

out
[ndarray] A dictionary described in the module documentation. The number of cascade levels is determined from the filter (see bandpass_filters.py).

Other Parameters

fft_method
[str or tuple] A string or a (function,kwargs) tuple defining the FFT method to use (see utils.fft.get_method). Defaults to "numpy".

MASK
[array_like] Optional mask to use for computing the statistics for the cascade levels. Pixels with MASK==False are excluded from the computations.

1.6 Noise generators (`pysteps.noise`)

Methods for generating stochastic perturbations of 2d precipitation and velocity fields.

1.6.1 pysteps.noise.interface

`pysteps.noise.interface.get_method(name)`

Return two callable functions to initialize and generate 2d perturbations of precipitation or velocity fields.

Methods for precipitation fields:

Name	Description
parametric	this global generator uses parametric Fourier filering (power-law model)
nonparametric	this global generator uses nonparametric Fourier filering
ssft	this local generator uses the short-space Fourier filtering
nested	this local generator uses a nested Fourier filtering

Methods for velocity fields:

Name	Description
bps	The method described in [1], where time-dependent velocity perturbations are sampled from the exponential distribution

1.6.2 pysteps.noise.fftgenerators

initialize_nonparam_2d_fft_filter(X, **kwargs)	Takes one ore more 2d input fields and produces one non-paramtric, global and anasotropic fourier filter.
initialize_param_2d_fft_filter(X, **kwargs)	Takes one ore more 2d input fields, fits two spectral slopes, beta1 and beta2, to produce one parametric, global and isotropic fourier filter.
generate_noise_2d_fft_filter(F[, randstate, . . .])	Produces a field of correlated noise using global Fourier filtering.
initialize_nonparam_2d_ssft_filter(X, **kwargs)	Function to compute the local Fourier filters using the Short-Space Fourier filtering approach.

Continued on next page

Table 6 – continued from previous page

initialize_nonparam_2d_nested_filter(X[, ...])	Function to compute the local Fourier filters using a nested approach.
generate_noise_2d_ssft_filter(F[, ...])	Function to compute the locally correlated noise using a nested approach.
build_2D_tapering_function(win_size[, win_type])	Produces two-dimensional tapering function for rectangular fields.

Methods for noise generators based on FFT filtering of white noise.

The methods in this module implement the following interface for filter initialization depending on their parametric or nonparametric nature:

> initialize_param_2d_xxx_filter(X, **kwargs)
>
> or
>
> initialize_nonparam_2d_xxx_filter(X, **kwargs)

where X is an array of shape (m, n) that defines the target field and optional parameters are supplied as keyword arguments.

The output of each initialization method is a dictionary containing the keys F and input_shape. The first is a two-dimensional array of shape (m, int(n/2)+1) that defines the filter. The second one is the shape of the input field for the filter.

The methods in this module implement the following interface for the generation of correlated noise:

> generate_noise_2d_xxx_filter(F, randstate=np.random, seed=None, **kwargs)

where F (m, n) is a filter returned from the correspondign initialization method, and randstate and seed can be used to set the random generator and its seed. Additional keyword arguments can be included as a dictionary.

The output of each generator method is a two-dimensional array containing the field of correlated noise cN of shape (m, n).

`pysteps.noise.fftgenerators.`**`build_2D_tapering_function`**(*win_size*, *win_type='flat-hanning'*)

Produces two-dimensional tapering function for rectangular fields.

Parameters

win_size
[tuple of int] Size of the tapering window as two-element tuple of integers.

win_type
[str] Name of the tapering window type (hanning, flat-hanning)

Returns

w2d
[array-like] A two-dimensional numpy array containing the 2D tapering function.

`pysteps.noise.fftgenerators.`**`generate_noise_2d_fft_filter`**(*F*, *randstate=<module 'numpy.random' from '/top/college/academic/ECE/spulkkin/home/ar packages/numpy/random/__init__.py'>*, *seed=None*, *fft_method=None*)

Produces a field of correlated noise using global Fourier filtering.

Parameters

F
[dict] A filter object returned by initialize_param_2d_fft_filter or initialize_nonparam_2d_fft_filter. All values in the filter array are required to be finite.

randstate
[mtrand.RandomState] Optional random generator to use. If set to None, use numpy.random.

seed
[int] Value to set a seed for the generator. None will not set the seed.

fft_method
[tuple] A string or a (function,kwargs) tuple defining the FFT method to use (see utils.fft.get_method). Defaults to "numpy".

Returns

N
[array-like] A two-dimensional numpy array of stationary correlated noise.

`pysteps.noise.fftgenerators.`**`generate_noise_2d_ssft_filter`**(*F*, *randstate=<module 'numpy.random' from '/top/college/academic/ECE/spulkkin/home/*… *packages/numpy/random/__init__.py'>*, *seed=None*, ***kwargs*)

Function to compute the locally correlated noise using a nested approach.

Parameters

F
[array-like] A filter object returned by initialize_nonparam_2d_nested_filter or initialize_nonparam_2d_ssft_filter. The filter is a four-dimensional array containing the 2d fourier filters distributed over a 2d spatial grid.

randstate
[mtrand.RandomState] Optional random generator to use. If set to None, use numpy.random.

seed
[int] Value to set a seed for the generator. None will not set the seed.

Returns

N
[array-like] A two-dimensional numpy array of non-stationary correlated noise.

Other Parameters

overlap
[float] Percentage overlap [0-1] between successive windows. Default : 0.2

win_type
[string ['hanning', 'flat-hanning']] Type of window used for localization. Default : flat-hanning

fft_method
[tuple] A string or a (function,kwargs) tuple defining the FFT method to use (see utils.fft.get_method). Defaults to "numpy".

`pysteps.noise.fftgenerators.`**`initialize_nonparam_2d_fft_filter`**(*X*, ***kwargs*)

Takes one ore more 2d input fields and produces one non-paramtric, global and anasotropic fourier filter.

Parameters

X
[array-like] Two- or three-dimensional array containing one or more input fields. All values are required to be finite. If more than one field are passed, the average fourier filter is returned. It assumes that fields are stacked by the first axis: [nr_fields, y, x].

Returns

F
[array-like] A two-dimensional array containing the non-parametric filter. It can be passed to generate_noise_2d_fft_filter().

Other Parameters

win_type
[string] Optional tapering function to be applied to X. Default : flat-hanning

donorm
[bool] Option to normalize the real and imaginary parts. Default : False

rm_rdisc
[bool] Whether or not to remove the rain/no-rain disconituity. It assumes no-rain pixels are assigned with lowest value.

fft_method
[tuple] A string or a (function,kwargs) tuple defining the FFT method to use (see utils.fft.get_method). Defaults to "numpy".

`pysteps.noise.fftgenerators.`**`initialize_nonparam_2d_nested_filter`**(*X*, *gridres=1.0*, ***kwargs*)

Function to compute the local Fourier filters using a nested approach.

Parameters

X
[array-like] Two- or three-dimensional array containing one or more input fields. All values are required to be finite. If more than one field are passed, the average fourier filter is returned. It assumes that fields are stacked by the first axis: [nr_fields, y, x].

gridres
[float] Grid resolution in km.

Returns

F
[array-like] Four-dimensional array containing the 2d fourier filters distributed over a 2d spatial grid.

Other Parameters

max_level
[int] Localization parameter. 0: global noise, >0: increasing degree of localization. Default : 3

win_type
[string ['hanning', 'flat-hanning']] Type of window used for localization. Default : flat-hanning

war_thr
[float [0;1]] Threshold for the minimum fraction of rain needed for computing the FFT. Default : 0.1

rm_rdisc
[bool] Whether or not to remove the rain/no-rain disconituity. It assumes no-rain pixels are assigned with lowest value.

fft_method
[tuple] A string or a (function,kwargs) tuple defining the FFT method to use (see utils.fft.get_method). Defaults to "numpy".

`pysteps.noise.fftgenerators.`**`initialize_nonparam_2d_ssft_filter`**(*X*, ***kwargs*)

Function to compute the local Fourier filters using the Short-Space Fourier filtering approach.

Parameters

X
[array-like] Two- or three-dimensional array containing one or more input fields. All values are required to be finite. If more than one field are passed, the average fourier filter is returned. It assumes that fields are stacked by the first axis: [nr_fields, y, x].

Returns

F
[array-like] Four-dimensional array containing the 2d fourier filters distributed over a 2d spatial grid.

Other Parameters

win_size
[int or two-element tuple of ints] Size-length of the window to compute the SSFT. Default : (128, 128)

win_type
[string ['hanning', 'flat-hanning']] Type of window used for localization. Default : flat-hanning

overlap
[float [0,1[] The proportion of overlap to be applied between successive windows. Default : 0.3

war_thr
[float [0,1]] Threshold for the minimum fraction of rain needed for computing the FFT. Default : 0.1

rm_rdisc
[bool] Whether or not to remove the rain/no-rain disconituity. It assumes no-rain pixels are assigned with lowest value.

fft_method
[tuple] A string or a (function,kwargs) tuple defining the FFT method to use (see utils.fft.get_method). Defaults to "numpy".

References

[7]

`pysteps.noise.fftgenerators.`**`initialize_param_2d_fft_filter`**(*X*, ***kwargs*)

Takes one ore more 2d input fields, fits two spectral slopes, beta1 and beta2, to produce one parametric, global and isotropic fourier filter.

Parameters

X
[array-like] Two- or three-dimensional array containing one or more input fields. All values are required to be finite. If more than one field are passed, the average fourier filter is returned. It assumes that fields are stacked by the first axis: [nr_fields, y, x].

Returns

F
[array-like] A two-dimensional array containing the parametric filter. It can be passed to generate_noise_2d_fft_filter().

Other Parameters

win_type
[string] Optional tapering function to be applied to X. Default : flat-hanning

model
[string] The parametric model to be used to fit the power spectrum of X. Default : power-law

weighted
[bool] Whether or not to apply the sqrt(power) as weight in the polyfit() function. Default : True

rm_rdisc
[bool] Whether or not to remove the rain/no-rain disconituity. It assumes no-rain pixels are assigned with lowest value. Default : True

doplot
[bool] Plot the fit. Default : False

fft_method
[tuple] A string or a (function,kwargs) tuple defining the FFT method to use (see utils.fft.get_method). Defaults to “numpy”.

1.6.3 pysteps.noise.motion

initialize_bps(V, pixelsperkm, timestep[, …])	Initialize the motion field perturbator described in [1].
generate_bps(perturbator, t)	Generate a motion perturbation field by using the method described in [1].

Methods for generating perturbations of two-dimensional motion fields.

The methods in this module implement the following interface for initialization:

inizialize_xxx(V, pixelsperkm, timestep, optional arguments)

where V (2,m,n) is the motion field and pixelsperkm and timestep describe the spatial and temporal resolution of the motion vectors. The output of each initialization method is a dictionary containing the perturbator that can be supplied to generate_xxx.

The methods in this module implement the following interface for the generation of a motion perturbation field:

generate_xxx(perturbator, t, randstate=np.random, seed=None)

where perturbator is a dictionary returned by an initialize_xxx method. Optional random generator can be specified with the randstate and seed arguments, respectively. The output of each generator method is an array of shape (2,m,n) containing the x- and y-components of the motion vector perturbations, where m and n are determined from the perturbator.

`pysteps.noise.motion.`**`generate_bps`**(*perturbator*, *t*)

Generate a motion perturbation field by using the method described in [1].

Parameters

perturbator
[dict] A dictionary returned by initialize_motion_perturbations_bps.

t
[float] Lead time for the perturbation field (minutes).

Returns

out
[ndarray] Array of shape (2,m,n) containing the x- and y-components of the motion vector perturbations, where m and n are determined from the perturbator.

See also:

pysteps.noise.motion.initialize_bps

`pysteps.noise.motion.`**`get_default_params_bps_par`**()

Return a tuple containing the default velocity perturbation parameters given in [1] for the parallel component.

`pysteps.noise.motion.`**`get_default_params_bps_perp`**()

Return a tuple containing the default velocity perturbation parameters given in [1] for the perpendicular component.

`pysteps.noise.motion.`**`initialize_bps`**(*V*, *pixelsperkm*, *timestep*, *p_pert_par=None*, *p_pert_perp=None*, *randstate=<module 'numpy.random' from '/top/college/academic/ECE/spulkkin/home/anaconda3/lib/python3.7/site-packages/numpy/random/__init__.py'>*, *seed=None*)

Initialize the motion field perturbator described in [1]. For simplicity, the bias adjustment procedure described there has not been implemented. The perturbator generates a constant field whose magnitude depends on lead time.

Parameters

V
[array_like] Array of shape (2,m,n) containing the x- and y-components of the m*n motion field to perturb.

p_pert_par
[tuple] Tuple containing the parameters a,b and c for the standard deviation of the perturbations in the direction parallel to the motion vectors. The standard deviations are modeled by the function f_par(t) = a*t**b+c, where t is lead time. The default values are taken from [1].

p_pert_perp
[tuple] Tuple containing the parameters a,b and c for the standard deviation of the perturbations in the direction perpendicular to the motion vectors. The standard deviations

are modeled by the function f_par(t) = a*t**b+c, where t is lead time. The default values are taken from [1].

pixelsperkm
[float] Spatial resolution of the motion field (pixels/kilometer).

timestep
[float] Time step for the motion vectors (minutes).

randstate
[mtrand.RandomState] Optional random generator to use. If set to None, use numpy.random.

seed
[int] Optional seed number for the random generator.

Returns

out
[dict] A dictionary containing the perturbator that can be supplied to generate_motion_perturbations_bps.

See also:

pysteps.noise.motion.generate_bps

1.6.4 pysteps.noise.utils

Miscellaneous utility functions related to generation of stochastic perturbations.

`pysteps.noise.utils.`**`compute_noise_stddev_adjs`**(*R, R_thr_1, R_thr_2, F, decomp_method, num_iter, conditional=True, num_workers=None*)

Apply a scale-dependent adjustment factor to the noise fields used in STEPS.

Simulates the effect of applying a precipitation mask to a Gaussian noise field obtained by the nonparametric filter method. The idea is to decompose the masked noise field into a cascade and compare the standard deviations of each level into those of the observed precipitation intensity field. This gives correction factors for the standard deviations [1]. The calculations are done for n realizations of the noise field, and the correction factors are calculated from the average values of the standard deviations.

Parameters

R
[array_like] The input precipitation field, assumed to be in logarithmic units (dBR or reflectivity).

R_thr_1
[float] Intensity threshold for precipitation/no precipitation.

R_thr_2
[float] Intensity values below R_thr_1 are set to this value.

F
[dict] A bandpass filter dictionary returned by a method defined in pysteps.cascade.bandpass_filters. This defines the filter to use and the number of cascade levels.

decomp_method
[function] A function defined in pysteps.cascade.decomposition. Specifies the method to use for decomposing the observed precipitation field and noise field into different spatial scales.

num_iter
[int] The number of noise fields to generate.

conditional
[bool] If set to True, compute the statistics conditionally by excluding areas of no precipitation.

num_workers
[int] The number of workers to use for parallel computation. Set to None to use all available CPUs. Applicable if dask is enabled.

Returns

out
[list] A list containing the standard deviation adjustment factor for each cascade level.

1.7 Post-processing of forecasts (`pysteps.postprocessing`)

Methods for post-processing of forecasts. Currently the module contains cumulative density function (CDF)-based matching between a forecast and the target distribution and computation of mean value and exceedance probabilities from forecast ensembles.

1.7.1 pysteps.postprocessing.ensemblestats

`mean`(X[, ignore_nan, X_thr])	Compute the mean value from a forecast ensemble field.
`excprob`(X, X_thr[, ignore_nan])	For a given forecast ensemble field, compute exceedance probabilities for the given intensity thresholds.

`pysteps.postprocessing.ensemblestats.`**`excprob`**(*X*, *X_thr*, *ignore_nan=False*)
For a given forecast ensemble field, compute exceedance probabilities for the given intensity thresholds.

Parameters

X
[array_like] Array of shape (k,m,n,...) containing an k-member ensemble of forecasts with shape (m,n,...).

X_thr
[float or a sequence of floats] Intensity threshold(s) for which the exceedance probabilities are computed.

ignore_nan
[bool] If True, ignore nan values.

Returns

out
[ndarray] Array of shape (len(X_thr),m,n) containing the exceedance probabilities for the given intensity thresholds. If len(X_thr)=1, the first dimension is dropped.

`pysteps.postprocessing.ensemblestats.`**`mean`**(*X*, *ignore_nan=False*, *X_thr=None*)
Compute the mean value from a forecast ensemble field.

Parameters

X
[array_like] Array of shape (n_members,m,n) containing an ensemble of forecast fields of shape (m,n).

ignore_nan
[bool] If True, ignore nan values.

X_thr
[float] Optional threshold for computing the ensemble mean. Values below X_thr are ignored.

Returns

out
[ndarray] Array of shape (m,n) containing the ensemble mean.

1.7.2 pysteps.postprocessing.probmatching

compute_empirical_cdf(bin_edges, hist)	Compute an empirical cumulative distribution function from the given histogram.
nonparam_match_empirical_cdf(R, R_trg)	Matches the empirical CDF of the initial array with the empirical CDF of a target array.
pmm_init(bin_edges_1, cdf_1, bin_edges_2, cdf_2)	Initialize a probability matching method (PMM) object from binned cumulative distribution functions (CDF).
pmm_compute(pmm, x)	For a given PMM object and x-coordinate, compute the probability matched value (i.e.

Methods for matching the empirical probability distribution of two data sets.

`pysteps.postprocessing.probmatching.`**`compute_empirical_cdf`**(*bin_edges*, *hist*)
Compute an empirical cumulative distribution function from the given histogram.

Parameters

bin_edges
[array_like] Coordinates of left edges of the histogram bins.

hist
[array_like] Histogram counts for each bin.

Returns

out
[ndarray] CDF values corresponding to the bin edges.

`pysteps.postprocessing.probmatching.`**`nonparam_match_empirical_cdf`**(*R*, *R_trg*)
Matches the empirical CDF of the initial array with the empirical CDF of a target array. Initial ranks are conserved, but empirical distribution matches the target one. Zero-pixels in initial array are conserved.

Parameters

R
[array_like] The initial array whose CDF is to be changed.

R_trg
[array_like] The target array whose CDF is to be matched.

Returns

out
[array_like] The new array.

`pysteps.postprocessing.probmatching.`**`pmm_compute`**(*pmm*, *x*)

For a given PMM object and x-coordinate, compute the probability matched value (i.e. the x-coordinate for which the target CDF has the same value as the source CDF).

Parameters

pmm
[dict] A PMM object returned by pmm_init.

x
[float] The coordinate for which to compute the probability matched value.

`pysteps.postprocessing.probmatching.`**`pmm_init`**(*bin_edges_1*, *cdf_1*, *bin_edges_2*, *cdf_2*)

Initialize a probability matching method (PMM) object from binned cumulative distribution functions (CDF).

Parameters

bin_edges_1
[array_like] Coordinates of the left bin edges of the source cdf.

cdf_1
[array_like] Values of the source CDF at the bin edges.

bin_edges_2
[array_like] Coordinates of the left bin edges of the target cdf.

cdf_2
[array_like] Values of the target CDF at the bin edges.

`pysteps.postprocessing.probmatching.`**`shift_scale`**(*R*, *f*, *rain_fraction_trg*, *second_moment_trg*, ***kwargs*)

Find shift and scale that is needed to return the required second_moment and rain area. The optimization is performed with the Nelder-Mead algorithm available in scipy. It ssumes a forward transformation ln_rain = ln(rain)-ln(min_rain) if rain > min_rain, else 0.

Parameters

R
[array_like] The initial array to be shift and scaled.

f
[function] The inverse transformation that is applied after the shift and scale.

rain_fraction_trg
[float] The required rain fraction to be matched by shifting.

second_moment_trg
[float] The required second moment to be matched by scaling. The second_moment is defined as second_moment = var + mean^2.

Returns

shift
[float] The shift value that produces the required rain fraction.

scale
[float] The scale value that produces the required second_moment.

R
[array_like] The shifted, scaled and back-transformed array.

Other Parameters

scale
[float] Optional initial value of the scale parameter for the Nelder-Mead optimisation. Typically, this would be the scale parameter estimated the previous time step. Default : 1.

max_iterations
[int] Maximum allowed number of iterations and function evaluations. More details: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html Deafult: 100.

tol
[float] Tolerance for termination. More details: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html Default: 0.05*second_moment_trg, i.e. terminate the search if the error is less than 5% since the second moment is a bit unstable.

1.8 Time series modeling and analysis (`pysteps.timeseries`)

Methods and models for time series analysis. Currently the module contains implementation of an autoregressive AR(p) model and methods for estimating the model parameters.

1.8.1 pysteps.timeseries.autoregression

adjust_lag2_corrcoef1(gamma_1, gamma_2)	A simple adjustment of lag-2 temporal autocorrelation coefficient to ensure that the resulting AR(2) process is stationary when the parameters are estimated from the Yule-Walker equations.
adjust_lag2_corrcoef2(gamma_1, gamma_2)	A more advanced adjustment of lag-2 temporal autocorrelation coefficient to ensure that the resulting AR(2) process is stationary when the parameters are estimated from the Yule-Walker equations.
estimate_ar_params_yw(gamma)	Estimate the parameters of an AR(p) model from the Yule-Walker equations using the given set of autocorrelation coefficients.
iterate_ar_model(X, phi[, EPS])	Apply an AR(p) model to a time-series of two-dimensional fields.

Methods related to autoregressive AR(p) models.

pysteps.timeseries.autoregression.**adjust_lag2_corrcoef1**(*gamma_1*, *gamma_2*)
A simple adjustment of lag-2 temporal autocorrelation coefficient to ensure that the resulting AR(2) process is stationary when the parameters are estimated from the Yule-Walker equations.

Parameters

gamma_1
[float] Lag-1 temporal autocorrelation coeffient.

gamma_2

[float] Lag-2 temporal autocorrelation coeffient.

Returns

out
[float] The adjusted lag-2 correlation coefficient.

`pysteps.timeseries.autoregression.`**`adjust_lag2_corrcoef2`**(*gamma_1*, *gamma_2*)
A more advanced adjustment of lag-2 temporal autocorrelation coefficient to ensure that the resulting AR(2) process is stationary when the parameters are estimated from the Yule-Walker equations.

Parameters

gamma_1
[float] Lag-1 temporal autocorrelation coeffient.

gamma_2
[float] Lag-2 temporal autocorrelation coeffient.

Returns

out
[float] The adjusted lag-2 correlation coefficient.

`pysteps.timeseries.autoregression.`**`estimate_ar_params_yw`**(*gamma*)
Estimate the parameters of an AR(p) model from the Yule-Walker equations using the given set of autocorrelation coefficients.

Parameters

gamma
[array_like] Array of length p containing the lag-l, l=1,2,. . . p, temporal autocorrelation coefficients. The correlation coefficients are assumed to be in ascending order with respect to time lag.

Returns

out
[ndarray] An array of shape (n,p+1) containing the AR(p) parameters for for the lag-p terms for each cascade level, and also the standard deviation of the innovation term.

`pysteps.timeseries.autoregression.`**`iterate_ar_model`**(*X*, *phi*, *EPS=None*)
Apply an AR(p) model to a time-series of two-dimensional fields.

Parameters

X
[array_like] Three-dimensional array of shape (p,w,h) containing a time series of p two-dimensional fields of shape (w,h). The fields are assumed to be in ascending order by time, and the timesteps are assumed to be regular.

phi
[array_like] Array of length p+1 specifying the parameters of the AR(p) model. The parameters are in ascending order by increasing time lag, and the last element is the parameter corresponding to the innovation term EPS.

EPS
[array_like] Optional perturbation field for the AR(p) process. If EPS is None, the innovation term is not added.

1.8.2 pysteps.timeseries.correlation

Methods for computing spatial and temporal correlation of time series of two-dimensional fields.

pysteps.timeseries.correlation.**temporal_autocorrelation**(*X*, *MASK=None*)

Compute lag-l autocorrelation coefficients gamma_l, l=1,2,. . . ,n-1, for a time series of n two-dimensional input fields.

Parameters

X

[array_like] Two-dimensional array of shape (n, w, h) containing a time series of n two-dimensional fields of shape (w, h). The input fields are assumed to be in increasing order with respect to time, and the time step is assumed to be regular (i.e. no missing data). X is required to have finite values.

MASK

[array_like] Optional mask to use for computing the correlation coefficients. Pixels with MASK==False are excluded from the computations.

Returns

out

[ndarray] Array of length n-1 containing the temporal autocorrelation coefficients for time lags l=1,2,. . . ,n-1.

1.9 Miscellaneous utility functions (`pysteps.utils`)

Utility functions for converting data values to/from different units, manipulating the dimensions of precipitation fields and computing the FFT.

1.9.1 pysteps.utils.conversion

to_rainrate(R, metadata[, a, b])	Convert to rain rate [mm/h].
to_raindepth(R, metadata[, a, b])	Convert to rain depth [mm].
to_reflectivity(R, metadata[, a, b])	Convert to reflectivity [dBZ].

Methods for converting physical units.

pysteps.utils.conversion.**to_raindepth**(*R*, *metadata*, *a=None*, *b=None*)

Convert to rain depth [mm].

Parameters

R

[array-like] Array of any shape to be (back-)transformed.

metadata

[dict] The metadata dictionary contains all data-related information.

a,b

[float] Optional, the a and b coefficients of the Z-R relationship.

Returns

R

[array-like] Array of any shape containing the converted units.

metadata
[dict] The metadata with updated attributes.

`pysteps.utils.conversion.`**`to_rainrate`**(*R*, *metadata*, *a=None*, *b=None*)
Convert to rain rate [mm/h].

Parameters

R
[array-like] Array of any shape to be (back-)transformed.

metadata
[dict] The metadata dictionary contains all data-related information.

a,b
[float] Optional, the a and b coefficients of the Z-R relationship.

Returns

R
[array-like] Array of any shape containing the converted units.

metadata
[dict] The metadata with updated attributes.

`pysteps.utils.conversion.`**`to_reflectivity`**(*R*, *metadata*, *a=None*, *b=None*)
Convert to reflectivity [dBZ].

Parameters

R
[array-like] Array of any shape to be (back-)transformed.

metadata
[dict] The metadata dictionary contains all data-related information.

a,b
[float] Optional, the a and b coefficients of the Z-R relationship.

Returns

R
[array-like] Array of any shape containing the converted units.

metadata
[dict] The metadata with updated attributes.

1.9.2 pysteps.utils.dimension

aggregate_fields_time(R, metadata, …[, …])	Aggregate fields in time.
aggregate_fields(R, window_size[, axis, method])	Aggregate fields.
square_domain(R, metadata[, method, inverse])	Either pad or crop a field to obtain a square domain.

Functions to manipulate array dimensions.

`pysteps.utils.dimension.`**`aggregate_fields`**(*R*, *window_size*, *axis=0*, *method='mean'*)
Aggregate fields. It attemps to aggregate the given R axis in an integer number of sections of length = window_size. If such a aggregation is not possible, an error is raised.

Parameters

R
[array-like] Array of any shape containing the input fields.

window_size
[int] The length of the window that is used to aggregate the fields.

axis
[int] The axis where to perform the aggregation.

method
[string] Optional argument that specifies the operation to use to aggregate the values within the window. Default to mean operator.

Returns

outputarray
[array-like] The new aggregated array with shape[axis] = k, where k = R.shape[axis]/window_size

`pysteps.utils.dimension.`**`aggregate_fields_space`**(*R*, *metadata*, *space_window_m*, *ignore_nan=False*)

Upscale fields in space.

Parameters

R
[array-like] Array of shape (m,n), (t,m,n) or (l,t,m,n) containing a single field or a time series of (ensemble) input fields.

metadata
[dict] The metadata dictionary contains all data-related information. It requires the keys "xpixelsize", "ypixelsize" and "unit".

space_window_m
[float or None] The length in meters of the space window that is used to upscale the fields. The space spanned by the m and n dimensions of R must be a multiple of space_window_m. If set to None, it returns a copy of the original R and metadata.

ignore_nan
[bool] If True, ignore nan values.

Returns

outputarray
[array-like] The new array of aggregated fields of shape (k,j), (t,k,j) or (l,t,k,j), where k = m*delta/space_window_m and j = n*delta/space_window_m; delta is the grid size.

metadata
[dict] The metadata with updated attributes.

`pysteps.utils.dimension.`**`aggregate_fields_time`**(*R*, *metadata*, *time_window_min*, *ignore_nan=False*)

Aggregate fields in time.

Parameters

R
[array-like] Array of shape (t,m,n) or (l,t,m,n) containing a time series of (ensemble) input fields. They must be evenly spaced in time.

metadata
[dict] The metadata dictionary contains all data-related information. It requires the keys "timestamps" and "unit".

time_window_min
[float or None] The length in minutes of the time window that is used to aggregate the fields. The time spanned by the t dimension of R must be a multiple of time_window_min. If set to None, it returns a copy of the original R and metadata.

ignore_nan
[bool] If True, ignore nan values.

Returns

outputarray
[array-like] The new array of aggregated fields of shape (k,m,n) or (l,k,m,n), where k = t*delta/time_window_min and delta is the time interval between two successive timestamps.

metadata
[dict] The metadata with updated attributes.

`pysteps.utils.dimension.`**`clip_domain`**(*R*, *metadata*, *extent=None*)

Clip the field domain by geographical coordinates.

Parameters

R
[array-like] Array of shape (m,n) or (t,m,n) containing the input fields.

metadata
[dict] The metadata dictionary contains all data-related information.

extent
[scalars (left, right, bottom, top)] The extent of the bounding box in data coordinates to be used to clip the data. Note that the direction of the vertical axis and thus the default values for top and bottom depend on origin. We follow the same convention as in the imshow method of matplotlib: https://matplotlib.org/tutorials/intermediate/imshow_extent.html

Returns

R
[array-like] the clipped array

metadata
[dict] the metadata with updated attributes.

`pysteps.utils.dimension.`**`square_domain`**(*R*, *metadata*, *method='pad'*, *inverse=False*)

Either pad or crop a field to obtain a square domain.

Parameters

R
[array-like] Array of shape (m,n) or (t,m,n) containing the input fields.

metadata
[dict] The metadata dictionary contains all data-related information.

method
[string] Either pad or crop. If pad, an equal number of zeros is added to both ends of its shortest side in order to produce a square domain. If crop, an equal number of pixels is

removed to both ends of its longest side in order to produce a square domain. Note that the crop method involves a loss of data.

inverse
[bool] Perform the inverse method to recover the original domain shape. After a crop, the inverse is performed by padding the field with zeros.

Returns

R
[array-like] the reshape dataset

metadata
[dict] the metadata with updated attributes.

1.9.3 pysteps.utils.fft

`get_method`(name)	Return a callable function for the FFT method corresponding to the given name.

Interface module for different FFT methods.

`pysteps.utils.fft.get_method(name)`
Return a callable function for the FFT method corresponding to the given name.

Parameters

name
[str] The name of the method. The available options are 'numpy', 'scipy' and 'pyfftw'

Returns

out
[tuple] A two-element tuple containing the FFT module and a dictionary of default keyword arguments for calling the FFT method. Each module implements the numpy.fft interface.

1.9.4 pysteps.utils.interface

`pysteps.utils.interface.get_method(name)`
Return a callable function for the utility method corresponding to the given name. For the FFT methods, the return value is a two-element tuple containing the function and a dictionary of keyword arguments.

Conversion methods:

Name	Description
mm/h or rainrate	convert to rain rate [mm/h]
mm or raindepth	convert to rain depth [mm]
dbz or reflectivity	convert to reflectivity [dBZ]

Transformation methods:

Name	Description
boxcox or box-cox	one-parameter Box-Cox transform
db or decibel	transform to units of decibel
log	log transform
nqt	Normal Quantile Transform
sqrt	square-root transform

Dimension methods:

Name	Description
accumulate	aggregate fields in time
clip	resize the field domain by geographical coordinates
square	either pad or crop the data to get a square domain
upscale	upscale the field

FFT methods (wrappers to different implementations):

Name	Description
numpy_fft	numpy.fft
scipy_fft	scipy.fftpack
pyfftw_fft	pyfftw.interfaces.numpy_fft

1.9.5 pysteps.utils.transformation

`dB_transform`(R[, metadata, threshold, ...])	Methods to transform to/from dB units.
`boxcox_transform`(R[, metadata, Lambda, ...])	The one-parameter Box-Cox transformation.
`boxcox_transform_test_lambdas`(R[, Lambdas, ...])	Test and plot various lambdas for the Box-Cox transformation.

Methods for transforming data values.

`pysteps.utils.transformation.`**`NQ_transform`**(*R*, *metadata=None*, *inverse=False*, ***kwargs*)

The normal quantile transformation. Zero rain vales are set to zero in norm space.

Parameters

R
[array-like] Array of any shape to be transformed.

metadata
[dict] The metadata dictionary contains all data-related information.

inverse
[bool] Optional, if set to True, it performs the inverse transform

Returns

R
[array-like] Array of any shape containing the (back-)transformed units.

metadata
[dict] The metadata with updated attributes.

Other Parameters

a
[float] Optional offset fraction to be used; typically in (0,1). Default : 0., i.e. it spaces the points evenly in the uniform distribution

`pysteps.utils.transformation.`**`boxcox_transform`**(*R*, *metadata=None*, *Lambda=None*, *threshold=None*, *zerovalue=None*, *inverse=False*)

The one-parameter Box-Cox transformation. Default parameters will produce a log transform (i.e. Lambda=0).

Parameters

R
[array-like] Array of any shape to be transformed.

metadata
[dict] The metadata dictionary contains all data-related information.

Lambda
[float] Parameter lambda of the Box-Cox transformation. Default : 0

threshold
[float] Optional value that is used for thresholding with the same units as R. If None, the threshold contained in metadata is used.

zerovalue
[float] Optional value to be assigned to no rain pixels as defined by the threshold.

inverse
[bool] Optional, if set to True, it performs the inverse transform

Returns

R
[array-like] Array of any shape containing the (back-)transformed units.

metadata
[dict] The metadata with updated attributes.

`pysteps.utils.transformation.`**`boxcox_transform_test_lambdas`**(*R*, *Lambdas=None*, *threshold=0.1*)

Test and plot various lambdas for the Box-Cox transformation.

`pysteps.utils.transformation.`**`dB_transform`**(*R*, *metadata=None*, *threshold=None*, *zerovalue=None*, *inverse=False*)

Methods to transform to/from dB units.

Parameters

R
[array-like] Array of any shape to be (back-)transformed.

metadata
[dict] The metadata dictionary contains all data-related information.

threshold
[float] Optional value that is used for thresholding with the same units as R. If None, the threshold contained in metadata is used.

zerovalue
[float] Optional value to be assigned to no rain pixels as defined by the threshold.

inverse
[bool] Optional, if set to True, it performs the inverse transform

Returns

R
[array-like] Array of any shape containing the (back-)transformed units.

metadata
[dict] The metadata with updated attributes.

`pysteps.utils.transformation.`**`sqrt_transform`**(*R*, *metadata=None*, *inverse=False*, ***kwargs*)

Square-root transform.

Parameters

R
[array-like] Array of any shape to be transformed.

metadata
[dict] The metadata dictionary contains all data-related information.

inverse
[bool] Optional, if set to True, it performs the inverse transform

Returns

R
[array-like] Array of any shape containing the (back-)transformed units.

metadata
[dict] The metadata with updated attributes.

1.10 Forecast verification (`pysteps.verification`)

Methods for verification of deterministic and ensemble forecasts.

1.10.1 pysteps.verification.detcatscores

det_cat_fcst(pred, obs, thr, scores)	Calculate simple and skill scores for deterministic categorical forecasts.

Forecast evaluation and skill scores for deterministic categorial forecasts.

`pysteps.verification.detcatscores.`**`det_cat_fcst`**(*pred*, *obs*, *thr*, *scores*)

Calculate simple and skill scores for deterministic categorical forecasts.

Parameters

pred
[array_like] predictions

obs
[array_like] verifying observations

scores
[list] a list containing the names of the scores to be computed, the full list is:

Name	Description
ACC	accuracy (proportion correct)
BIAS	frequency bias
CSI	critical success index (threat score)
FA	false alarm rate (prob. of false detection)
FAR	false alarm ratio
GSS	Gilbert skill score (equitable threat score)
HK	Hanssen-Kuipers discriminant (Pierce skill score)
HSS	Heidke skill score
POD	probability of detection (hit rate)
SEDI	symmetric extremal dependency index

Returns

result
[list] the verification results

1.10.2 pysteps.verification.detcontscores

det_cont_fcst(pred, obs, scores, **kwargs)	Calculate simple and skill scores for deterministic continuous forecasts

Forecast evaluation and skill scores for deterministic continuous forecasts.

pysteps.verification.detcontscores.**det_cont_fcst**(*pred*, *obs*, *scores*, ***kwargs*)
Calculate simple and skill scores for deterministic continuous forecasts

Parameters

pred
[array_like] predictions

obs
[array_like] verifying observations

scores
[list] a list containing the names of the scores to be computed, the full list is:

Name	Description
beta	linear regression slope (conditional bias)
corr_p	pearson's correleation coefficien (linear correlation)
corr_s	spearman's correlation coefficient (rank correlation)
ME_add	mean error or bias of additive residuals
ME_mult	mean error or bias of multiplicative residuals
RMSE_add	root mean squared additive error
RMSE_mult	root mean squared multiplicative error
RV_add	reduction of variance (Brier Score, Nash-Sutcliffe Efficiency)
RV_mult	reduction of variance in multiplicative space
scatter	half the distance between the 16% and 84% percentiles of the error distribution

Returns

result
[list] list containing the verification results

Other Parameters

offset
[float] an offset that is added to both prediction and observation to avoid 0 division when computing multiplicative residuals. Default is 0.01.

1.10.3 pysteps.verification.ensscores

ensemble_skill(X_f, X_o, metric, **kwargs)	Compute mean ensemble skill for a given skill metric.
ensemble_spread(X_f, metric, **kwargs)	Compute mean ensemble spread for a given skill metric.
rankhist_init(num_ens_members, X_min)	Initialize a rank histogram object.
rankhist_accum(rankhist, X_f, X_o)	Accumulate forecast-observation pairs to the given rank histogram.
rankhist_compute(rankhist[, normalize])	Return the rank histogram counts and optionally normalize the histogram.

Evaluation and skill scores for ensemble forecasts.

`pysteps.verification.ensscores.`**`ensemble_skill`**(*X_f*, *X_o*, *metric*, ***kwargs*)
Compute mean ensemble skill for a given skill metric.

Parameters

X_f
[array-like] Array of shape (l,m,n) containing the forecast fields of shape (m,n) from l ensemble members.

X_o
[array_like] Array of shape (m,n) containing the observed field corresponding to the forecast.

metric
[str] The deterministic skill metric to be used (list available in `get_method()`)

Returns

out
[float] The mean skill of all ensemble members that is used as defintion of ensemble skill (as in Zacharov and Rezcova 2009 with the FSS).

Other Parameters

thr
[float] Intensity threshold for categorical scores.

scale
[int] The spatial scale to verify in px. In practice it represents the size of the moving window that it is used to compute the fraction of pixels above the threshold for the FSS.

References

[13]

`pysteps.verification.ensscores.`**`ensemble_spread`**(*X_f*, *metric*, ***kwargs*)
Compute mean ensemble spread for a given skill metric.

Parameters

X_f
[array-like] Array of shape (l,m,n) containing the forecast fields of shape (m,n) from l ensemble members.

metric
[str] The skill metric to be used, the list includes:

Returns

out
[float] The mean skill compted between all possible pairs of the ensemble members, which can be used as definition of mean ensemble spread (as in Zacharov and Rezcova 2009 with the FSS).

Other Parameters

thr
[float] Intensity threshold for categorical scores.

scale
[int] The spatial scale to verify in px. In practice it represents the size of the moving window that it is used to compute the fraction of pixels above the threshold for the FSS.

References

[13]

`pysteps.verification.ensscores.`**`rankhist_accum`**(*rankhist*, *X_f*, *X_o*)
Accumulate forecast-observation pairs to the given rank histogram.

Parameters

X_f
[array-like] Array of shape (k,m,n,...) containing the values from an ensemble forecast of k members with shape (m,n,...).

X_o
[array_like] Array of shape (m,n,...) containing the observed values corresponding to the forecast.

`pysteps.verification.ensscores.`**`rankhist_compute`**(*rankhist*, *normalize=True*)
Return the rank histogram counts and optionally normalize the histogram.

Parameters

rankhist
[dict] A rank histogram object created with rankhist_init.

normalize
[bool] If True, normalize the rank histogram so that the bin counts sum to one.

Returns

out
[array_like] The counts for the n+1 bins in the rank histogram, where n is the number of ensemble members.

`pysteps.verification.ensscores.`**`rankhist_init`**(*num_ens_members*, *X_min*)

Initialize a rank histogram object.

Parameters

num_ens_members
[int] Number ensemble members in the forecasts to accumulate into the rank histogram.

X_min
[float] Threshold for minimum intensity. Forecast-observation pairs, where all ensemble members and verifying observations are below X_min, are not counted in the rank histogram.

Returns

out
[dict] The rank histogram object.

1.10.4 pysteps.verification.plots

plot_rankhist(rankhist[, ax])	Plot a rank histogram.
plot_reldiag(reldiag[, ax])	Plot a reliability diagram.
plot_ROC(ROC[, ax, opt_prob_thr])	Plot a ROC curve.

`pysteps.verification.plots.`**`plot_ROC`**(*ROC*, *ax=None*, *opt_prob_thr=False*)

Plot a ROC curve.

Parameters

ROC
[dict] A ROC curve object created by probscores.ROC_curve_init.

ax
[axis handle] Axis handle for the figure. If set to None, the handle is taken from the current figure (matplotlib.pylab.gca()).

opt_prob_thr
[bool] If set to True, plot the optimal probability threshold that maximizes the difference between the hit rate (POD) and false alarm rate (POFD).

`pysteps.verification.plots.`**`plot_intensityscale`**(*iss*, *fig=None*, *vmin=-2*, *vmax=1*, *kmperpixel=None*, *unit=None*)

Plot a intensity-scale verification table with a color bar and axis labels.

Parameters

iss
[dict] An intensity-scale verification results dictionary returned by pysteps.verification.spatialscores.intensity_scale.

fig
[matplotlib.figure.Figure] Optional figure object to use for plotting. If not supplied, a new figure is created.

vmin
[float] Optional minimum value for the intensity-scale skill score in the plot. Defaults to -2.

vmax
[float] Optional maximum value for the intensity-scale skill score in the plot. Defaults

to 1.

kmperpixel
[float] Optional conversion factor from pixels to kilometers. If supplied, the unit of the shown spatial scales is km instead of pixels.

unit
[string] Optional unit of the intensity thresholds.

`pysteps.verification.plots.`**`plot_rankhist`**(*rankhist*, *ax=None*)
Plot a rank histogram.

Parameters

rankhist
[dict] A rank histogram object created by ensscores.rankhist_init.

ax
[axis handle] Axis handle for the figure. If set to None, the handle is taken from the current figure (matplotlib.pylab.gca()).

`pysteps.verification.plots.`**`plot_reldiag`**(*reldiag*, *ax=None*)
Plot a reliability diagram.

Parameters

reldiag
[dict] A ROC curve object created by probscores.reldiag_init.

ax
[axis handle] Axis handle for the figure. If set to None, the handle is taken from the current figure (matplotlib.pylab.gca()).

1.10.5 pysteps.verification.probscores

CRPS(X_f, X_o)	Compute the average continuous ranked probability score (CRPS) for a set of forecast ensembles and the corresponding observations.
reldiag_init(X_min[, n_bins, min_count])	Initialize a reliability diagram object.
reldiag_accum(reldiag, P_f, X_o)	Accumulate the given probability-observation pairs into the reliability diagram.
reldiag_compute(reldiag)	Compute the x- and y- coordinates of the points in the reliability diagram.
ROC_curve_init(X_min[, n_prob_thrs])	Initialize a ROC curve object.
ROC_curve_accum(ROC, P_f, X_o)	Accumulate the given probability-observation pairs into the given ROC object.
ROC_curve_compute(ROC[, compute_area])	Compute the ROC curve and its area from the given ROC object.

Evaluation and skill scores for probabilistic forecasts.

`pysteps.verification.probscores.`**`CRPS`**(*X_f*, *X_o*)
Compute the average continuous ranked probability score (CRPS) for a set of forecast ensembles and the corresponding observations.

Parameters

X_f
[array_like] Array of shape (n,m) containing n ensembles of forecast values with each

ensemble having m members.

X_o
[array_like] Array of n observed values.

Returns

out
[float] The continuous ranked probability score.

References

[6]

`pysteps.verification.probscores.`**`ROC_curve_accum`**(*ROC*, *P_f*, *X_o*)
Accumulate the given probability-observation pairs into the given ROC object.

Parameters

ROC
[dict] A ROC curve object created with ROC_curve_init.

P_f
[array_like] Forecasted probabilities for exceeding the threshold specified in the ROC object. Non-finite values are ignored.

X_o
[array_like] Observed values. Non-finite values are ignored.

`pysteps.verification.probscores.`**`ROC_curve_compute`**(*ROC*, *compute_area=False*)
Compute the ROC curve and its area from the given ROC object.

Parameters

ROC
[dict] A ROC curve object created with ROC_curve_init.

compute_area
[bool] If True, compute the area under the ROC curve (between 0.5 and 1).

Returns

out
[tuple] A two-element tuple containing the probability of detection (POD) and probability of false detection (POFD) for the probability thresholds specified in the ROC curve object. If compute_area is True, return the area under the ROC curve as the third element of the tuple.

`pysteps.verification.probscores.`**`ROC_curve_init`**(*X_min*, *n_prob_thrs=10*)
Initialize a ROC curve object.

Parameters

X_min
[float] Precipitation intensity threshold for yes/no prediction.

n_prob_thrs
[int] The number of probability thresholds to use. The interval [0,1] is divided into n_prob_thrs evenly spaced values.

Returns

out
[dict] The ROC curve object.

`pysteps.verification.probscores.`**`reldiag_accum`**(*reldiag*, *P_f*, *X_o*)
Accumulate the given probability-observation pairs into the reliability diagram.

Parameters

reldiag
[dict] A reliability diagram object created with reldiag_init.

P_f
[array-like] Forecast probabilities for exceeding the intensity threshold specified in the reliability diagram object.

X_o
[array-like] Observed values.

`pysteps.verification.probscores.`**`reldiag_compute`**(*reldiag*)
Compute the x- and y- coordinates of the points in the reliability diagram.

Parameters

reldiag
[dict] A reliability diagram object created with reldiag_init.

Returns

out
[tuple] Two-element tuple containing the x- and y-coordinates of the points in the reliability diagram.

`pysteps.verification.probscores.`**`reldiag_init`**(*X_min*, *n_bins=10*, *min_count=10*)
Initialize a reliability diagram object.

Parameters

X_min
[float] Precipitation intensity threshold for yes/no prediction.

n_bins
[int] Number of bins to use in the reliability diagram.

min_count
[int] Minimum number of samples required for each bin. A zero value is assigned if the number of samples in a bin is smaller than bin_count.

Returns

out
[dict] The reliability diagram object.

References

[2]

1.10.6 pysteps.verification.spatialscores

fss(X_f, X_o, thr, scale)	Compute the fractions skill score (FSS) for a deterministic forecast field and the corresponding observation.

Skill scores for spatial forecasts.

`pysteps.verification.spatialscores.binary_mse` (*X_f*, *X_o*, *thr*, *wavelet='haar'*)

Compute an intensity-scale verification as the MSE of the binary error. This method uses PyWavelets for decomposing the error field between the forecasts and observations into multiple spatial scales.

Parameters

X_f
[array_like] Array of shape (n,m) containing the forecast field.

X_o
[array_like] Array of shape (n,m) containing the verification observation field.

thr
[sequence] The intensity threshold for which to compute the verification.

wavelet
[str] The name of the wavelet function to use. Defaults to the Haar wavelet, as described in Casati et al. 2004. See the documentation of PyWavelets for a list of available options.

Returns

SS
[array] One-dimensional array containing the binary MSE for each spatial scale.

spatial_scale
[list]

References

[3]

`pysteps.verification.spatialscores.fss` (*X_f*, *X_o*, *thr*, *scale*)

Compute the fractions skill score (FSS) for a deterministic forecast field and the corresponding observation.

Parameters

X_f
[array_like] Array of shape (n,m) containing the forecast field.

X_o
[array_like] Array of shape (n,m) containing the reference field (observation).

thr
[float] Intensity threshold.

scale
[int] The spatial scale in px. In practice they represent the size of the moving window that it is used to compute the fraction of pixels above the threshold.

Returns

out
[float] The fractions skill score between 0 and 1.

References

[9], [4]

`pysteps.verification.spatialscores.intensity_scale_accum(iss, X_f, X_o)`

Compute and update the intensity-scale verification scores.

Parameters

iss
[dict] An intensity-scale object created with intensity_scale_init.

X_f
[array_like] Array of shape (n,m) containing the forecast field.

X_o
[array_like] Array of shape (n,m) containing the verification observation field.

Returns

iss
[dict] A dictionary with the following key-value pairs:

Key	Value
name	the name of the intensity-scale skill score
SS	two-dimensional array containing the intensity-scale skill scores for each spatial scale and intensity threshold
scales	the spatial scales, corresponds to the first index of SS
thrs	the used intensity thresholds in increasing order, corresponds to the second index of SS
n	the number of verified fct-obs pairs that were averaged
shape	the shape of the fct-obs fields

`pysteps.verification.spatialscores.intensity_scale_init(name, thrs, scales=None, wavelet='Haar')`

Initialize an intensty-scale verification object.

Parameters

score_names
[string] A string indicating the name of the spatial verification score to be used:

Name	Description
FSS	Fractions skill score
BMSE	Binary mean squared error

thrs
[sequence] A sequence of intensity thresholds for which to compute the verification.

scales
[sequence] A sequence of spatial scales in pixels to be used in the FSS.

wavelet
[str] The name of the wavelet function to use in the BMSE. Defaults to the Haar wavelet, as described in Casati et al. 2004. See the documentation of PyWavelets for a list of available options.

Returns

iss
[dict] The intensity-scale object.

1.11 Visualization (`pysteps.visualization`)

Methods for plotting precipitation and motion fields.

1.11.1 pysteps.visualization.animations

Functions to produce animations for pysteps.

`pysteps.visualization.animations.`**`animate`**(*R_obs*, *nloops=2*, *timestamps=None*, *R_fct=None*, *timestep_min=5*, *UV=None*, *motion_plot='quiver'*, *geodata=None*, *map=None*, *colorscale='MeteoSwiss'*, *units='mm/h'*, *colorbar=True*, *probmaps=False*, *probmap_thrs=None*, *ensmeans=False*, *plotanimation=True*, *savefig=False*, *fig_dpi=150*, *fig_format='png'*, *path_outputs="*, ***kwargs*)

Function to animate observations and forecasts in pysteps.

Parameters

R_obs
[array-like] Three-dimensional array containing the time series of observed precipitation fields.

Returns

ax
[fig axes] Figure axes. Needed if one wants to add e.g. text inside the plot.

Other Parameters

nloops
[int] Optional, the number of loops in the animation.

R_fct
[array-like] Optional, the three or four-dimensional (for ensembles) array containing the time series of forecasted precipitation field.

timestep_min
[float] The time resolution in minutes of the forecast.

UV
[array-like] Optional, the motion field used for the forecast.

motion_plot
[string] The method to plot the motion field.

geodata
[dictionary] Optional dictionary containing geographical information about the field. If geodata is not None, it must contain the following key-value pairs:

Key	Value
projection	PROJ.4-compatible projection definition
x1	x-coordinate of the lower-left corner of the data raster (meters)
y1	y-coordinate of the lower-left corner of the data raster (meters)
x2	x-coordinate of the upper-right corner of the data raster (meters)
y2	y-coordinate of the upper-right corner of the data raster (meters)
yorigin	a string specifying the location of the first element in the data raster w.r.t. y-axis: 'upper' = upper border, 'lower' = lower border

map
[str] Optional method for plotting a map. See pysteps.visualization.precipifields.plot_precip.field.

units
[str] Units of the input array (mm/h or dBZ)

colorscale
[str] Which colorscale to use.

title
[str] If not None, print the title on top of the plot.

colorbar
[bool] If set to True, add a colorbar on the right side of the plot.

probmaps
[bool] If True, compute and plot exceedance probability maps from the nowcast ensemble.

probmap_thrs
[a sequence of floats] Intensity thresholds for the exceedance probability maps. Applicable if probmaps is set to True.

ensmeans
[bool] If True, plot ensemble mean nowcasts.

plotanimation
[bool] If set to True, visualize the animation (useful when one is only interested in saving the individual frames).

savefig
[bool] If set to True, save the individual frames to path_outputs.

fig_dpi
[scalar > 0] Resolution of the output figures, see the documentation of matplotlib.pyplot.savefig. Applicable if savefig is True.

path_outputs
[string] Path to folder where to save the frames.

****kwargs**
[dict] Optional keyword arguments that are supplied to plot_precip_field.

1.11.2 pysteps.visualization.motionfields

quiver(UV[, ax, geodata])	Function to plot a motion field as arrows.
streamplot(UV[, ax, geodata])	Function to plot a motion field as streamlines.

Functions to plot motion fields.

`pysteps.visualization.motionfields.`**`quiver`**(*UV*, *ax=None*, *geodata=None*, ***kwargs*)

Function to plot a motion field as arrows.

Parameters

UV
[array-like] Array of shape (2,m,n) containing the input motion field.

ax
[axis object] Optional axis object to use for plotting.

geodata
[dictionary] Optional dictionary containing geographical information about the field. If geodata is not None, it must contain the following key-value pairs:

Key	Value
projection	PROJ.4-compatible projection definition
x1	x-coordinate of the lower-left corner of the data raster (meters)
y1	y-coordinate of the lower-left corner of the data raster (meters)
x2	x-coordinate of the upper-right corner of the data raster (meters)
y2	y-coordinate of the upper-right corner of the data raster (meters)
yorigin	a string specifying the location of the first element in the data raster w.r.t. y-axis: 'upper' = upper border, 'lower' = lower border

Returns

out
[axis object] Figure axes. Needed if one wants to add e.g. text inside the plot.

Other Parameters

step
[int] Optional resample step to control the density of the arrows. Default : 20

color
[string] Optional color of the arrows. This is a synonym for the PolyCollection facecolor kwarg in matplotlib.collections. Default : black

`pysteps.visualization.motionfields.`**`streamplot`**(*UV*, *ax=None*, *geodata=None*, ***kwargs*)

Function to plot a motion field as streamlines.

Parameters

UV
[array-like] Array of shape (2, m,n) containing the input motion field.

ax
[axis object] Optional axis object to use for plotting.

geodata
[dictionary] Optional dictionary containing geographical information about the field. If geodata is not None, it must contain the following key-value pairs:

Key	Value
projection	PROJ.4-compatible projection definition
x1	x-coordinate of the lower-left corner of the data raster (meters)
y1	y-coordinate of the lower-left corner of the data raster (meters)
x2	x-coordinate of the upper-right corner of the data raster (meters)
y2	y-coordinate of the upper-right corner of the data raster (meters)
yorigin	a string specifying the location of the first element in the data raster w.r.t. y-axis: 'upper' = upper border, 'lower' = lower border

Returns

out
[axis object] Figure axes. Needed if one wants to add e.g. text inside the plot.

Other Parameters

density
[float] Controls the closeness of streamlines. Default : 1.5

color
[string] Optional streamline color. This is a synonym for the PolyCollection facecolor kwarg in matplotlib.collections. Default : black

1.11.3 pysteps.visualization.precipfields

plot_precip_field(R[, type, map, geodata, ...])	Function to plot a precipitation intensity or probability field with a colorbar.
get_colormap(type[, units, colorscale])	Function to generate a colormap (cmap) and norm.

Methods for plotting precipitation fields.

`pysteps.visualization.precipfields.`**`get_colormap`**(*type*, *units='mm/h'*, *colorscale='MeteoSwiss'*)

Function to generate a colormap (cmap) and norm.

Parameters

type
[str] Type of the map to plot: 'intensity' = precipitation intensity field, 'prob' = exceedance probability field.

units
[str] Units of the input array (mm/h or dBZ).

colorscale
[str] Which colorscale to use (MeteoSwiss, STEPS-BE). Applicable if units is 'mm/h' or 'dBZ'.

Returns

cmap
[Colormap instance] colormap

norm
[colors.Normalize object] Colors norm

clevs: list(float)
List of precipitation values defining the color limits.

clevsStr: list(str)
List of precipitation values defining the color limits (with correct number of decimals).

`pysteps.visualization.precipfields.plot_precip_field`(*R*, *type='intensity'*, *map=None*, *geodata=None*, *units='mm/h'*, *colorscale='MeteoSwiss'*, *probthr=None*, *title=None*, *colorbar=True*, *drawlonlatlines=False*, *basemap_resolution='l'*, *cartopy_scale='50m'*)

Function to plot a precipitation intensity or probability field with a colorbar.

Parameters

R
[array-like] Two-dimensional array containing the input precipitation field or an exceedance probability map.

Returns

ax
[fig axes] Figure axes. Needed if one wants to add e.g. text inside the plot.

Other Parameters

type
[str] Type of the map to plot: 'intensity' = precipitation intensity field, 'prob' = exceedance probability field.

map
[str] Optional method for plotting a map: 'basemap' or 'cartopy'. The former uses mpl_toolkits.basemap (https://matplotlib.org/basemap), and the latter uses cartopy (https://scitools.org.uk/cartopy/docs/latest).

geodata
[dictionary] Optional dictionary containing geographical information about the field. If geodata is not None, it must contain the following key-value pairs:

Key	Value
projection	PROJ.4-compatible projection definition
x1	x-coordinate of the lower-left corner of the data raster (meters)
y1	y-coordinate of the lower-left corner of the data raster (meters)
x2	x-coordinate of the upper-right corner of the data raster (meters)
y2	y-coordinate of the upper-right corner of the data raster (meters)
yorigin	a string specifying the location of the first element in the data raster w.r.t. y-axis: 'upper' = upper border, 'lower' = lower border

units
[str] Units of the input array (mm/h or dBZ). If type is 'prob', this specifies the unit of the intensity threshold.

colorscale
[str] Which colorscale to use (MeteoSwiss, STEPS-BE). Applicable if units is 'mm/h' or 'dBZ'.

probthr
[float] Intensity threshold to show in the color bar of the exceedance probability map. Required if type is "prob" and colorbar is True.

title
[str] If not None, print the title on top of the plot.

colorbar
[bool] If set to True, add a colorbar on the right side of the plot.

drawlonlatlines
[bool] If set to True, draw longitude and latitude lines. Applicable if map is 'basemap' or 'cartopy'.

basemap_resolution
[str] The resolution of the basemap, see the documentation of mpl_toolkits.basemap. Applicable if map is 'basemap'.

cartopy_scale
[str] The scale (resolution) of the map. The available options are '10m', '50m', and '110m'. Applicable if map is 'cartopy'.

1.11.4 pysteps.visualization.utils

Miscellaneous utility functions.

`pysteps.visualization.utils.`**`parse_proj4_string`**(*proj4str*)
Construct a dictionary from a PROJ.4 projection string.

Parameters

proj4str
[str] A PROJ.4-compatible projection string.

Returns

out
[dict] Dictionary, where keys and values are parsed from the projection parameter tokens beginning with '+'.

`pysteps.visualization.utils.`**`proj4_to_basemap`**(*proj4str*)
Convert a PROJ.4 projection string into a dictionary that can be expanded as keyword arguments to mpl_toolkits.basemap.Basemap.__init__.

Parameters

proj4str
[str] A PROJ.4-compatible projection string.

Returns

out
[dict] The output dictionary.

`pysteps.visualization.utils.`**`proj4_to_cartopy`**(*proj4str*)
Convert a PROJ.4 projection string into a Cartopy coordinate reference system (crs) object.

Parameters

proj4str
[str] A PROJ.4-compatible projection string.

Returns

out
[object] Instance of a crs class defined in cartopy.crs.

BIBLIOGRAPHY

[1] N. E. Bowler, C. E. Pierce, and A. W. Seed. STEPS: a probabilistic precipitation forecasting scheme which merges an extrapolation nowcast with downscaled NWP. *Quarterly Journal of the Royal Meteorological Society*, 132(620):2127–2155, 2006. doi:10.1256/qj.04.100.

[2] J. Bröcker and L. A. Smith. Increasing the reliability of reliability diagrams. *Weather and Forecasting*, 22(3):651–661, 2007. doi:10.1175/WAF993.1.

[3] B. Casati, G. Ross, and D. B. Stephenson. A new intensity-scale approach for the verification of spatial precipitation forecasts. *Meteorological Applications*, 11(2):141—154, 2004. doi:10.1017/S1350482704001239.

[4] E. Ebert, L. Wilson, A. Weigel, M. Mittermaier, P. Nurmi, P. Gill, M. Göber, S. Joslyn, B. Brown, T. Fowler, and A. Watkins. Progress and challenges in forecast verification. *Meteorological Applications*, 20(2):130–139, 2013. doi:10.1002/met.1392.

[5] U. Germann and I. Zawadzki. Scale-dependence of the predictability of precipitation from continental radar images. Part I: description of the methodology. *Monthly Weather Review*, 130(12):2859–2873, 2002. doi:10.1175/1520-0493(2002)130<2859:SDOTPO>2.0.CO;2.

[6] H. Hersbach. Decomposition of the continuous ranked probability score for ensemble prediction systems. *Weather and Forecasting*, 15(5):559–570, 2000. doi:10.1175/1520-0434(2000)015<0559:DOTCRP>2.0.CO;2.

[7] D. Nerini, N. Besic, I. Sideris, U. Germann, and L. Foresti. A non-stationary stochastic ensemble generator for radar rainfall fields based on the short-space Fourier transform. *Hydrology and Earth System Sciences*, 21(6):2777–2797, 2017. doi:10.5194/hess-21-2777-2017.

[8] S. Pulkkinen, V. Chandrasekar, and A.-M. Harri. Nowcasting of precipitation in the high-resolution Dallas-Fort Worth (DFW) urban radar remote sensing network. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(8):2773–2787, 2018. doi:10.1109/JSTARS.2018.2840491.

[9] N. M. Roberts and H. W. Lean. Scale-selective verification of rainfall accumulations from high-resolution forecasts of convective events. *Monthly Weather Review*, 136(1):78–97, 2008. doi:10.1175/2007MWR2123.1.

[10] E. Ruzanski, V. Chandrasekar, and Y. Wang. The CASA nowcasting system. *Journal of Atmospheric and Oceanic Technology*, 28(5):640–655, 2011. doi:10.1175/2011JTECHA1496.1.

[11] A. W. Seed. A dynamic and spatial scaling approach to advection forecasting. *Journal of Applied Meteorology*, 42(3):381–388, 2003. doi:10.1175/1520-0450(2003)042<0381:ADASSA>2.0.CO;2.

[12] A. W. Seed, C. E. Pierce, and K. Norman. Formulation and evaluation of a scale decomposition-based stochastic precipitation nowcast scheme. *Water Resources Research*, 49(10):6624–6641, 2013. doi:10.1002/wrcr.20536.

[13] P. Zacharov and D. Rezacova. Using the fractions skill score to assess the relationship between an ensemble QPF spread and skill. *Atmospheric Research*, 94(4):684–693, 2009. doi:10.1016/j.atmosres.2009.03.004.