pyrad library reference for users

Release 0.5.0

meteoswiss-mdr

Dec 09, 2020

CONTENTS

Contents:

CHAPTER

ONE

PROCESSING FLOW CONTROL (PYRAD.FLOW)

Functions to control the Pyrad data processing flow

main(cfgfile[, starttime, endtime, ...])	Main flow control.
main_rt(cfgfile_list[, starttime, endtime, ...])	main flow control.

pyrad.flow.**main**(*cfgfile*, *starttime=None*, *endtime=None*, *trajfile=''*, *trajtype='plane'*, *flashnr=0*, *infostr=''*, *MULTIPROCESSING_DSET=False*, *MULTIPROCESSING_PROD=False*, *PROFILE_MULTIPROCESSING=False*, *USE_CHILD_PROCESS=False*)

Main flow control. Processes radar data off-line over a period of time given either by the user, a trajectory file, or determined by the last volume processed and the current time. Multiple radars can be processed simultaneously

Parameters

cfgfile [str] path of the main config file

starttime, endtime [datetime object] start and end time of the data to be processed

trajfile [str] path to file describing the trajectory

trajtype [str] type of trajectory file. Can be either 'plane', 'lightning' or 'proc_periods'

flashnr [int] If larger than 0 will select a flash in a lightning trajectory file. If 0 the data corresponding to the trajectory of all flashes will be plotted

infostr [str] Information string about the actual data processing (e.g. 'RUN57'). This string is added to product files.

MULTIPROCESSING_DSET [Bool] If true the generation of datasets at the same processing level will be parallelized

MULTIPROCESSING_PROD [Bool] If true the generation of products from each dataset will be parallelized

PROFILE_MULTIPROCESSING [Bool] If true and code parallelized the multiprocessing is profiled

USE_CHILD_PROCESS [Bool] If true the reading and processing of the data will be performed by a child process controlled by dask. This is done to make sure all memory used is released.

pyrad.flow.**main_rt**(*cfgfile_list*, *starttime=None*, *endtime=None*, *infostr_list=None*, *proc_period=60*, *proc_finish=None*)

main flow control. Processes radar data in real time. The start and end processing times can be determined by the user. This function is inteded for a single radar

Parameters

cfgfile_list [list of str] path of the main config files

starttime, endtime [datetime object] start and end time of the data to be processed

infostr_list [list of str] Information string about the actual data processing (e.g. 'RUN57'). This string is added to product files.

proc_period [int] period of time before starting a new processing round (seconds)

proc_finish [int or None] if set to a value the program will be forced to shut down after the value (in seconds) from start time has been exceeded

Returns

end_proc [Boolean] If true the program has ended successfully

CHAPTER

TWO

DATASET PROCESSING (PYRAD.PROC)

Initiate the dataset processing.

2.1 Auxiliary functions

get_process_func(dataset_type, dsname)	Maps the dataset type into its processing function and data set format associated.
process_raw(procstatus, dscfg[, radar_list])	Dummy function that returns the initial input data set
process_save_radar(procstatus, dscfg[, . . .])	Dummy function that allows to save the entire radar object
process_fixed_rng(procstatus, dscfg[, . . .])	Obtains radar data at a fixed range
process_fixed_rng_span(procstatus, dscfg[, . . .])	For each azimuth-elevation gets the data within a fixed range span and computes a user-defined statistic: mean, min, max, mode, median
process_roi(procstatus, dscfg[, radar_list])	Obtains the radar data at a region of interest defined by a TRT file or by the user.
process_azimuthal_average(procstatus, dscfg)	Averages radar data in azimuth obtaining and RHI as a result
process_radar_resampling(procstatus, dscfg)	Resamples the radar data to mimic another radar with different geometry and antenna pattern

2.2 Gridded data functions

process_raw_grid(procstatus, dscfg[, radar_list])	Dummy function that returns the initial input data set
process_grid(procstatus, dscfg[, radar_list])	Puts the radar data in a regular grid
process_grid_point(procstatus, dscfg[, . . .])	Obtains the grid data at a point location.
process_grid_time_stats(procstatus, dscfg[, . . .])	computes the temporal statistics of a field
process_grid_time_stats2(procstatus, dscfg)	computes temporal statistics of a field
process_grid_texture(procstatus, dscfg[, . . .])	Computes the 2D texture of a gridded field
process_grid_fields_diff(procstatus, dscfg)	Computes grid field differences
process_grid_mask(procstatus, dscfg[, . . .])	Mask data.
process_normalize_luminosity(procstatus, dscfg)	Normalize the data by the sinus of the sun elevation.
process_pixel_filter(procstatus, dscfg[, . . .])	Masks all pixels that are not of the class specified in keyword pixel_type

2.3 Spectral data functions

process_raw_spectra(procstatus, dscfg[, ...])	Dummy function that returns the initial input data set
process_spectra_point(procstatus, dscfg[, ...])	Obtains the spectra or IQ data at a point location.
process_filter_0Doppler(procstatus, dscfg[, ...])	Function to filter the 0-Doppler line bin and neighbours of the Doppler spectra
process_filter_spectra_noise(procstatus, dscfg)	Filter the noise of the Doppler spectra by clipping any data below the noise level plus a margin
process_filter_srhohv(procstatus, dscfg[, ...])	Filter Doppler spectra as a function of spectral RhoHV
process_spectra_ang_avg(procstatus, dscfg[, ...])	Function to average the spectra over the rays.
process_spectral_power(procstatus, dscfg[, ...])	Computes the spectral power
process_spectral_noise(procstatus, dscfg[, ...])	Computes the spectral noise
process_spectral_phase(procstatus, dscfg[, ...])	Computes the spectral phase
process_spectral_reflectivity(procstatus, dscfg)	Computes spectral reflectivity
process_spectral_differential_reflectivity(...)	Computes spectral differential reflectivity
process_spectral_differential_phase(...[, ...])	Computes the spectral differential phase
process_spectral_rhohv(procstatus, dscfg[, ...])	Computes the spectral RhoHV
process_pol_variables(procstatus, dscfg[, ...])	Computes the polarimetric variables from the complex spectra
process_noise_power(procstatus, dscfg[, ...])	Computes the noise power from the spectra
process_reflectivity(procstatus, dscfg[, ...])	Computes reflectivity from the spectral reflectivity
process_differential_reflectivity(...[, ...])	Computes differential reflectivity from the horizontal and vertical spectral reflectivity
process_differential_phase(procstatus, dscfg)	Computes the differential phase from the spectral differential phase and the spectral reflectivity
process_rhohv(procstatus, dscfg[, radar_list])	Computes RhoHV from the complex spectras
process_Doppler_velocity(procstatus, dscfg)	Compute the Doppler velocity from the spectral reflectivity
process_Doppler_width(procstatus, dscfg[, ...])	Compute the Doppler spectrum width from the spectral reflectivity
process_ifft(procstatus, dscfg[, radar_list])	Compute the Doppler spectrum width from the spectral reflectivity

2.4 IQ data functions

process_raw_iq(procstatus, dscfg[, radar_list])	Dummy function that returns the initial input data set
process_pol_variables_iq(procstatus, dscfg)	Computes the polarimetric variables from the IQ data
process_reflectivity_iq(procstatus, dscfg[, ...])	Computes reflectivity from the IQ data

Continued on next page

Table 4 – continued from previous page

process_st1_iq(procstatus, dscfg[, radar_list])	Computes the statistical test one lag fluctuation from the horizontal or vertical IQ data
process_st2_iq(procstatus, dscfg[, radar_list])	Computes the statistical test two lag fluctuation from the horizontal or vertical IQ data
process_wbn_iq(procstatus, dscfg[, radar_list])	Computes the wide band noise from the horizontal or vertical IQ data
process_differential_reflectivity_iq(. . . [, . . .])	Computes differential reflectivity from the horizontal and vertical IQ data
process_mean_phase_iq(procstatus, dscfg[, . . .])	Computes the mean phase from the horizontal or vertical IQ data
process_differential_phase_iq(procstatus, dscfg)	Computes the differential phase from the horizontal and vertical IQ data
process_rhohv_iq(procstatus, dscfg[, radar_list])	Computes RhoHV from the horizontal and vertical IQ data
process_Doppler_velocity_iq(procstatus, dscfg)	Compute the Doppler velocity from the spectral reflectivity
process_Doppler_width_iq(procstatus, dscfg)	Compute the Doppler spectrum width from the spectral reflectivity
process_fft(procstatus, dscfg[, radar_list])	Compute the Doppler spectra form the IQ data with a Fourier transform

2.5 Echo classification and filtering

process_echo_id(procstatus, dscfg[, radar_list])	identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Precipitation
process_birds_id(procstatus, dscfg[, radar_list])	identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Birds
process_clt_to_echo_id(procstatus, dscfg[, . . .])	Converts clutter exit code from rad4alp into pyrad echo ID
process_echo_filter(procstatus, dscfg[, . . .])	Masks all echo types that are not of the class specified in keyword echo_type
process_cdf(procstatus, dscfg[, radar_list])	Collects the fields necessary to compute the Cumulative Distribution Function
process_filter_snr(procstatus, dscfg[, . . .])	filters out low SNR echoes
process_filter_visibility(procstatus, dscfg)	filters out rays gates with low visibility and corrects the reflectivity
process_outlier_filter(procstatus, dscfg[, . . .])	filters out gates which are outliers respect to the surrounding
process_hydroclass(procstatus, dscfg[, . . .])	Classifies precipitation echoes
process_melting_layer(procstatus, dscfg[, . . .])	Detects the melting layer
process_filter_vel_diff(procstatus, dscfg[, . . .])	filters out range gates that could not be used for Doppler velocity estimation
process_zdr_column(procstatus, dscfg[, . . .])	Detects ZDR columns

2.6 Phase processing and attenuation correction

process_correct_phidp0(procstatus, dscfg[, ...])	corrects phidp of the system phase
process_smooth_phidp_single_window(...[, ...])	corrects phidp of the system phase and smoothes it using one window
process_smooth_phidp_double_window(...[, ...])	corrects phidp of the system phase and smoothes it using one window
process_kdp_leastsquare_single_window(...[, ...])	Computes specific differential phase using a piecewise least square method
process_kdp_leastsquare_double_window(...[, ...])	Computes specific differential phase using a piecewise least square method
process_phidp_kdp_Vulpiani(procstatus, dscfg)	Computes specific differential phase and differential phase using the method developed by Vulpiani et al.
process_phidp_kdp_Kalman(procstatus, dscfg)	Computes specific differential phase and differential phase using the Kalman filter as proposed by Schneebeli et al.
process_phidp_kdp_Maesaka(procstatus, dscfg)	Estimates PhiDP and KDP using the method by Maesaka.
process_phidp_kdp_lp(procstatus, dscfg[, ...])	Estimates PhiDP and KDP using a linear programming algorithm.
process_attenuation(procstatus, dscfg[, ...])	Computes specific attenuation and specific differential attenuation using the Z-Phi method and corrects reflectivity and differential reflectivity

2.7 Monitoring, calibration and noise correction

process_correct_bias(procstatus, dscfg[, ...])	Corrects a bias on the data
process_correct_noise_rhohv(procstatus, dscfg)	identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Precipitation
process_rhohv_rain(procstatus, dscfg[, ...])	Keeps only suitable data to evaluate the 80 percentile of RhoHV in rain
process_zdr_precip(procstatus, dscfg[, ...])	Keeps only suitable data to evaluate the differential reflectivity in moderate rain or precipitation (for vertical scans)
process_zdr_snow(procstatus, dscfg[, radar_list])	Keeps only suitable data to evaluate the differential reflectivity in snow
process_estimate_phidp0(procstatus, dscfg[, ...])	estimates the system differential phase offset at each ray
process_sun_hits(procstatus, dscfg[, radar_list])	monitoring of the radar using sun hits
process_selfconsistency_kdp_phidp(...[, ...])	Computes specific differential phase and differential phase in rain using the selfconsistency between Zdr, Zh and KDP
process_selfconsistency_bias(procstatus, dscfg)	Estimates the reflectivity bias by means of the selfconsistency algorithm by Gourley
process_selfconsistency_bias2(procstatus, dscfg)	Estimates the reflectivity bias by means of the selfconsistency algorithm by Gourley
process_time_avg_std(procstatus, dscfg[, ...])	computes the average and standard deviation of data.
process_occurrence(procstatus, dscfg[, ...])	computes the frequency of occurrence of data.
process_occurrence_period(procstatus, dscfg)	computes the frequency of occurrence over a long period of time by adding together shorter periods
process_monitoring(procstatus, dscfg[, ...])	computes monitoring statistics

Continued on next page

Table 7 – continued from previous page

process_gc_monitoring(procstatus, dscfg[, …])	computes ground clutter monitoring statistics
process_time_avg(procstatus, dscfg[, radar_list])	computes the temporal mean of a field
process_weighted_time_avg(procstatus, dscfg)	computes the temporal mean of a field weighted by the reflectivity
process_time_avg_flag(procstatus, dscfg[, …])	computes a flag field describing the conditions of the data used while averaging
process_time_stats(procstatus, dscfg[, …])	computes the temporal statistics of a field
process_time_stats2(procstatus, dscfg[, …])	computes the temporal mean of a field
process_colocated_gates(procstatus, dscfg[, …])	Find colocated gates within two radars
process_intercomp(procstatus, dscfg[, …])	intercomparison between two radars
process_intercomp_time_avg(procstatus, dscfg)	intercomparison between the average reflectivity of two radars
process_fields_diff(procstatus, dscfg[, …])	Computes the field difference between RADAR001 and radar002, i.e.
process_intercomp_fields(procstatus, dscfg)	intercomparison between two radars

2.8 Retrievals

process_ccor(procstatus, dscfg[, radar_list])	Computes the Clutter Correction Ratio, i.e.
process_signal_power(procstatus, dscfg[, …])	Computes the signal power in dBm
process_rcs(procstatus, dscfg[, radar_list])	Computes the radar cross-section (assuming a point target) from radar reflectivity.
process_rcs_pr(procstatus, dscfg[, radar_list])	Computes the radar cross-section (assuming a point target) from radar reflectivity by first computing the received power and then the RCS from it.
process_radial_noise_hs(procstatus, dscfg[, …])	Computes the radial noise from the signal power using the Hildebrand and Sekhon 1974 method
process_radial_noise_ivic(procstatus, dscfg)	Computes the radial noise from the signal power using the Ivic 2013 method
process_snr(procstatus, dscfg[, radar_list])	Computes SNR
process_l(procstatus, dscfg[, radar_list])	Computes L parameter
process_cdr(procstatus, dscfg[, radar_list])	Computes Circular Depolarization Ratio
process_rainrate(procstatus, dscfg[, radar_list])	Estimates rainfall rate from polarimetric moments
process_rainfall_accumulation(procstatus, dscfg)	Computes rainfall accumulation fields
process_vol_refl(procstatus, dscfg[, radar_list])	Computes the volumetric reflectivity in 10log10(cm^2 km^-3)
process_bird_density(procstatus, dscfg[, …])	Computes the bird density from the volumetric reflectivity

2.9 Doppler processing

process_turbulence(procstatus, dscfg[, …])	Computes turbulence from the Doppler spectrum width and reflectivity using the PyTDA package
process_dealias_fourdd(procstatus, dscfg[, …])	Dealiases the Doppler velocity field using the 4DD technique from Curtis and Houze, 2001

Continued on next page

Table 9 – continued from previous page

process_dealias_region_based(procstatus, dscfg)	Dealiases the Doppler velocity field using a region based algorithm
process_dealias_unwrap_phase(procstatus, dscfg)	Dealiases the Doppler velocity field using multi-dimensional phase unwrapping
process_radial_velocity(procstatus, dscfg[, . . .])	Estimates the radial velocity respect to the radar from the wind velocity
process_wind_vel(procstatus, dscfg[, radar_list])	Estimates the horizontal or vertical component of the wind from the radial velocity
process_windshear(procstatus, dscfg[, . . .])	Estimates the wind shear from the wind velocity
process_vad(procstatus, dscfg[, radar_list])	Estimates vertical wind profile using the VAD (velocity Azimuth Display) technique

2.10 Time series functions

process_point_measurement(procstatus, dscfg)	Obtains the radar data at a point location.
process_qvp(procstatus, dscfg[, radar_list])	Computes quasi vertical profiles, by averaging over height levels PPI data.
process_rqvp(procstatus, dscfg[, radar_list])	Computes range defined quasi vertical profiles, by averaging over height levels PPI data.
process_svp(procstatus, dscfg[, radar_list])	Computes slanted vertical profiles, by averaging over height levels PPI data.
process_evp(procstatus, dscfg[, radar_list])	Computes enhanced vertical profiles, by averaging over height levels PPI data.
process_time_height(procstatus, dscfg[, . . .])	Produces time height radar objects at a point of interest defined by latitude and longitude.
process_ts_along_coord(procstatus, dscfg[, . . .])	Produces time series along a particular antenna coordinate

2.11 Trajectory functions

process_trajectory(procstatus, dscfg[, . . .])	Return trajectory
process_traj_atplane(procstatus, dscfg[, . . .])	Return time series according to trajectory
process_traj_antenna_pattern(procstatus, dscfg)	Process a new array of data volumes considering a plane trajectory.
process_traj_lightning(procstatus, dscfg[, . . .])	Return time series according to lightning trajectory
process_traj_trt(procstatus, dscfg[, . . .])	Processes data according to TRT trajectory
process_traj_trt_contour(procstatus, dscfg)	Gets the TRT cell contour corresponding to each radar volume

2.12 COSMO data

process_cosmo(procstatus, dscfg[, radar_list])	Gets COSMO data and put it in radar coordinates
process_cosmo_lookup_table(procstatus, dscfg)	Gets COSMO data and put it in radar coordinates using look up tables computed or loaded when initializing

Continued on next page

Table 12 – continued from previous page

process_cosmo_coord(procstatus, dscfg[, …])	Gets the COSMO indices corresponding to each cosmo coordinates
process_hzt(procstatus, dscfg[, radar_list])	Gets iso0 degree data in HZT format and put it in radar coordinates
process_hzt_lookup_table(procstatus, dscfg)	Gets HZT data and put it in radar coordinates using look up tables computed or loaded when initializing
process_hzt_coord(procstatus, dscfg[, …])	Gets the HZT indices corresponding to each HZT coordinates
process_cosmo_to_radar(procstatus, dscfg[, …])	Gets COSMO data and put it in radar coordinates using look up tables

2.13 DEM data

process_dem(procstatus, dscfg[, radar_list])	Gets DEM data and put it in radar coordinates
process_visibility(procstatus, dscfg[, …])	Gets the visibility in percentage from the minimum visible elevation.

pyrad.proc.**get_process_func**(*dataset_type*, *dsname*)

Maps the dataset type into its processing function and data set format associated.

Parameters

dataset_type [str] The following is a list of data set types ordered by type of output dataset with the function they call. For details of what they do check the function documentation:

'VOL' format output: 'ATTENUATION': process_attenuation 'AZI_AVG': process_azimuthal_average 'BIAS_CORRECTION': process_correct_bias 'BIRDS_ID': process_birds_id 'BIRD_DENSITY': process_bird_density 'CCOR': process_ccor 'CDF': process_cdf 'CDR': process_cdr 'CLT_TO_SAN': process_clt_to_echo_id 'COSMO': process_cosmo 'COSMO_LOOKUP': process_cosmo_lookup_table 'DEM': process_dem 'DEALIAS_FOURDD': process_dealias_fourdd 'DEALIAS_REGION': process_dealias_region_based 'DEALIAS_UNWRAP': process_dealias_unwrap_phase 'DOPPLER_VELOCITY': process_Doppler_velocity 'DOPPLER_VELOCITY_IQ': process_Doppler_velocity_iq 'DOPPLER_WIDTH': process_Doppler_width 'DOPPLER_WIDTH_IQ': process_Doppler_width_iq 'ECHO_FILTER': process_echo_filter 'FIELDS_DIFF': process_fields_diff 'FIXED_RNG': process_fixed_rng 'FIXED_RNG_SPAN': process_fixed_rng_span 'HYDROCLASS': process_hydroclass 'HZT': process_hzt 'HZT_LOOKUP': process_hzt_lookup_table 'KDP_LEASTSQUARE_1W': process_kdp_leastsquare_single_window 'KDP_LEASTSQUARE_2W': process_kdp_leastsquare_double_window 'L': process_l 'MEAN_PHASE_IQ': process_mean_phase_iq 'NCVOL': process_save_radar 'NOISE_POWER': process_noise_power 'OUTLIER_FILTER': process_outlier_filter 'PhiDP': process_differential_phase 'PHIDP0_CORRECTION': process_correct_phidp0 'PHIDP0_ESTIMATE': process_estimate_phidp0 'PhiDP_IQ': process_differential_phase_iq 'PHIDP_KDP_KALMAN': process_phidp_kdp_Kalman 'PHIDP_KDP_LP': process_phidp_kdp_lp 'PHIDP_KDP_VULPIANI': process_phidp_kdp_Vulpiani 'PHIDP_SMOOTH_1W': process_smooth_phidp_single_window 'PHIDP_SMOOTH_2W': process_smooth_phidp_double_window 'POL_VARIABLES': process_pol_variables 'POL_VARIABLES_IQ': process_pol_variables_iq 'PWR': process_signal_power 'RADAR_RESAMPLING': process_radar_resampling 'RADIAL_NOISE_HS': process_radial_noise_hs 'RADIAL_NOISE_IVIC': process_radial_noise_ivic 'RADIAL_VELOCITY': process_radial_velocity 'RAINRATE': process_rainrate 'RAW': process_raw 'REFLECTIVITY': process_reflectivity 'REFLECTIVITY_IQ': process_reflectivity_iq 'RCS': process_rcs 'RCS_PR': process_rcs_pr 'RhoHV': process_rhohv 'RhoHV_IQ': process_rhohv_iq 'RHOHV_CORRECTION': process_correct_noise_rhohv 'RHOHV_RAIN': process_rhohv_rain 'ROI': process_roi 'SAN': process_echo_id 'SELFCONSISTENCY_BIAS': process_selfconsistency_bias 'SELFCONSISTENCY_BIAS2': process_selfconsistency_bias2 'SELFCONSISTENCY_KDP_PHIDP': process_selfconsistency_kdp_phidp 'SNR': process_snr 'SNR_FILTER': process_filter_snr 'ST1_IQ': process_st1_iq 'ST2_IQ': process_st2_iq 'TRAJ_TRT' : process_traj_trt 'TRAJ_TRT_CONTOUR' : process_traj_trt_contour 'TURBULENCE': process_turbulence 'VAD': process_vad 'VEL_FILTER': process_filter_vel_diff 'VIS': process_visibility 'VIS_FILTER': process_filter_visibility 'VOL_REFL': process_vol_refl 'WBN': process_wbn_iq 'WIND_VEL': process_wind_vel 'WINDSHEAR': process_windshear 'ZDR': process_differential_reflectivity 'ZDR_IQ': process_differential_reflectivity_iq 'ZDR_PREC': process_zdr_precip 'ZDR_SNOW': process_zdr_snow

'SPECTRA' format output: 'FFT': process_fft 'FILTER_0DOPPLER': process_filter_0Doppler 'FILTER_SPECTRA_NOISE': process_filter_spectra_noise 'IFFT': process_ifft 'RAW_IQ': process_raw_iq 'RAW_SPECTRA': process_raw_spectra 'SPECTRA_ANGULAR_AVERAGE': process_spectra_ang_avg 'SPECTRA_POINT': process_spectra_point 'SPECTRAL_NOISE': process_spectral_noise 'SPECTRAL_PHASE': process_spectral_phase 'SPECTRAL_POWER': process_spectral_power 'SPECTRAL_REFLECTIVITY': process_spectral_reflectivity 'sPhiDP': process_spectral_differential_phase 'sRhoHV': process_spectral_RhoHV 'SRHOHV_FILTER': process_filter_srhohv 'sZDR': process_spectral_differential_reflectivity

'COLOCATED_GATES' format output: 'COLOCATED_GATES': process_colocated_gates

'COSMO_COORD' format output: 'COSMO_COORD': process_cosmo_coord 'HZT_COORD': process_hzt_coord

'COSMO2RADAR' format output: 'COSMO2RADAR': process_cosmo_to_radar

'GRID' format output: 'RAW_GRID': process_raw_grid 'GRID': process_grid 'GRID_FIELDS_DIFF': process_grid_fields_diff 'GRID_MASK': process_grid_mask 'GRID_TEXTURE': process_grid_texture 'NORMALIZE_LUMINOSITY': process_normalize_luminosity 'PIXEL_FILTER': process_pixel_filter

'GRID_TIMEAVG' format output: 'GRID_TIME_STATS': process_grid_time_stats 'GRID_TIME_STATS2': process_grid_time_stats2

'INTERCOMP' format output: 'INTERCOMP': process_intercomp 'INTERCOMP_FIELDS': process_intercomp_fields 'INTERCOMP_TIME_AVG': process_intercomp_time_avg

'ML' format output: 'ML_DETECTION': process_melting_layer

'MONITORING' format output: 'GC_MONITORING': process_gc_monitoring 'MONITORING': process_monitoring

'OCCURRENCE' format output: 'OCCURRENCE': process_occurrence 'OCCURRENCE_PERIOD': process_occurrence_period 'TIMEAVG_STD': process_time_avg_std

'QVP' format output: 'EVP': process_evp 'QVP': process_qvp 'rQVP': process_rqvp 'SVP': process_svp 'TIME_HEIGHT': process_time_height 'TIME_ALONG_COORD': process_ts_along_coord

'SPARSE_GRID' format output: 'ZDR_COLUMN': process_zdr_column

'SUN_HITS' format output: 'SUN_HITS': process_sun_hits

'TIMEAVG' format output: 'FLAG_TIME_AVG': process_time_avg_flag 'TIME_AVG': process_time_avg 'WEIGHTED_TIME_AVG': process_weighted_time_avg 'TIME_STATS': process_time_stats 'TIME_STATS2': process_time_stats2 'RAIN_ACCU': process_rainfall_accumulation

'TIMESERIES' format output: 'GRID_POINT_MEASUREMENT': process_grid_point 'POINT_MEASUREMENT': 'process_point_measurement' 'TRAJ_ANTENNA_PATTERN': process_traj_antenna_pattern 'TRAJ_ATPLANE': process_traj_atplane 'TRAJ_LIGHTNING': process_traj_lightning

'TRAJ_ONLY' format output: 'TRAJ': process_trajectory

dsname [str] Name of dataset

Returns

func_name [str or processing function] pyrad function used to process the data set type

dsformat [str] data set format, i.e.: 'VOL', etc.

`pyrad.proc.`**`process_Doppler_velocity`**(*procstatus*, *dscfg*, *radar_list=None*)

Compute the Doppler velocity from the spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_Doppler_velocity_iq`**(*procstatus*, *dscfg*, *radar_list=None*)

Compute the Doppler velocity from the spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

direction [str] The convention used in the Doppler mean field. Can be negative_away or negative_towards

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_Doppler_width`**(*procstatus*, *dscfg*, *radar_list=None*)

Compute the Doppler spectrum width from the spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_Doppler_width_iq`**(*procstatus*, *dscfg*, *radar_list=None*)

Compute the Doppler spectrum width from the spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signals

lag [int] Time lag used in the denominator of the computation

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_attenuation`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes specific attenuation and specific differential attenuation using the Z-Phi method and corrects reflectivity and differential reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

ATT_METHOD [float. Dataset keyword] The attenuation estimation method used. One of the following: ZPhi, Philin

fzl [float. Dataset keyword] The default freezing level height. It will be used if no temperature field name is specified or the temperature field is not in the radar object. Default 2000.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_azimuthal_average`**(*procstatus*, *dscfg*, *radar_list=None*)

Averages radar data in azimuth obtaining and RHI as a result

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

angle [float or None. Dataset keyword] The center angle to average. If not set or set to -1 all available azimuth angles will be used

delta_azi [float. Dataset keyword] The angle span to average. If not set or set to -1 all the available azimuth angles will be used

avg_type [str. Dataset keyword] Average type. Can be mean or median

nvalid_min [int. Dataset keyword] the (minimum) radius of the region of interest in m. Default half the largest resolution

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the gridded data

ind_rad [int] radar index

`pyrad.proc.`**`process_bird_density`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the bird density from the volumetric reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

sigma_bird [float. Dataset keyword] The bird radar cross section

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_birds_id`**(*procstatus*, *dscfg*, *radar_list=None*)

identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Birds

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_ccor`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the Clutter Correction Ratio, i.e. the ratio between the signal without Doppler filtering and the signal with Doppler filtering

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_cdf`**(*procstatus*, *dscfg*, *radar_list=None*)

Collects the fields necessary to compute the Cumulative Distribution Function

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_cdr`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes Circular Depolarization Ratio

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_clt_to_echo_id`**(*procstatus*, *dscfg*, *radar_list=None*)

Converts clutter exit code from rad4alp into pyrad echo ID

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_colocated_gates`**(*procstatus*, *dscfg*, *radar_list=None*)

Find colocated gates within two radars

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

h_tol [float. Dataset keyword] Tolerance in altitude difference between radar gates [m]. Default 100.

latlon_tol [float. Dataset keyword] Tolerance in latitude and longitude position between radar gates [deg]. Default 0.0005

vol_d_tol [float. Dataset keyword] Tolerance in pulse volume diameter [m]. Default 100.

vismin [float. Dataset keyword] Minimum visibility [percent]. Default None.

hmin [float. Dataset keyword] Minimum altitude [m MSL]. Default None.

hmax [float. Dataset keyword] Maximum altitude [m MSL]. Default None.

rmin [float. Dataset keyword] Minimum range [m]. Default None.

rmax [float. Dataset keyword] Maximum range [m]. Default None.

elmin [float. Dataset keyword] Minimum elevation angle [deg]. Default None.

elmax [float. Dataset keyword] Maximum elevation angle [deg]. Default None.

azrad1min [float. Dataset keyword] Minimum azimuth angle [deg] for radar 1. Default None.

azrad1max [float. Dataset keyword] Maximum azimuth angle [deg] for radar 1. Default None.

azrad2min [float. Dataset keyword] Minimum azimuth angle [deg] for radar 2. Default None.

azrad2max [float. Dataset keyword] Maximum azimuth angle [deg] for radar 2. Default None.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [radar object] radar object containing the flag field

ind_rad [int] radar index

`pyrad.proc.`**`process_correct_bias`**(*procstatus*, *dscfg*, *radar_list=None*)

Corrects a bias on the data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type to correct for bias

bias [float. Dataset keyword] The bias to be corrected [dB]. Default 0

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_correct_noise_rhohv`**(*procstatus*, *dscfg*, *radar_list=None*)

identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Precipitation

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The data types used in the correction

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_correct_phidp0`**(*procstatus*, *dscfg*, *radar_list=None*)

corrects phidp of the system phase

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_cosmo`**(*procstatus*, *dscfg*, *radar_list=None*)

Gets COSMO data and put it in radar coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

keep_in_memory [int. Dataset keyword] if set keeps the COSMO data dict, the COSMO coordinates dict and the COSMO field in radar coordinates in memory

regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and there is no need to compute a new COSMO field if the COSMO data has not changed

cosmo_type [str. Dataset keyword] name of the COSMO field to process. Default TEMP

cosmo_variables [list of strings. Dataset keyword] Py-art name of the COSMO fields. Default temperature

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_cosmo_coord`**(*procstatus*, *dscfg*, *radar_list=None*)

Gets the COSMO indices corresponding to each cosmo coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

cosmopath [string. General keyword] path where to store the look up table

model [string. Dataset keyword] The COSMO model to use. Can be cosmo-1, cosmo-1e, cosmo-2, cosmo-7

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_cosmo_lookup_table`**(*procstatus*, *dscfg*, *radar_list=None*)

Gets COSMO data and put it in radar coordinates using look up tables computed or loaded when initializing

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

lookup_table [int. Dataset keyword] if set a pre-computed look up table for the COSMO coordinates is loaded. Otherwise the look up table is computed taking the first radar object as reference

regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and therefore there is no need to interpolate the COSMO field in memory to the current radar grid

cosmo_type [str. Dataset keyword] name of the COSMO field to process. Default TEMP

cosmo_variables [list of strings. Dataset keyword] Py-art name of the COSMO fields. Default temperature

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_cosmo_to_radar**(*procstatus*, *dscfg*, *radar_list=None*)

Gets COSMO data and put it in radar coordinates using look up tables

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

cosmo_type [str. Dataset keyword] name of the COSMO field to process. Default TEMP

cosmo_variables [list of strings. Dataset keyword] Py-art name of the COSMO fields. Default temperature

cosmo_time_index_min, cosmo_time_index_max [int] minimum and maximum indices of the COSMO data to retrieve. If a value is provided only data corresponding to the time indices within the interval will be used. If None all data will be used. Default None

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_dealias_fourdd**(*procstatus*, *dscfg*, *radar_list=None*)

Dealiases the Doppler velocity field using the 4DD technique from Curtis and Houze, 2001

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

filt [int. Dataset keyword] Flag controlling Bergen and Albers filter, 1 = yes, 0 = no.

sign [int. Dataset keyword] Sign convention which the radial velocities in the volume created from the sounding data will will. This should match the convention used in the radar data. A value of 1 represents when positive values velocities are towards the radar, -1 represents when negative velocities are towards the radar.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_dealias_region_based(procstatus, dscfg, radar_list=None)`

Dealiases the Doppler velocity field using a region based algorithm

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

interval_splits [int, optional] Number of segments to split the nyquist interval into when finding regions of similar velocity. More splits creates a larger number of initial regions which takes longer to process but may result in better dealiasing. The default value of 3 seems to be a good compromise between performance and artifact free dealiasing. This value is not used if the interval_limits parameter is not None.

skip_between_rays, skip_along_ray [int, optional] Maximum number of filtered gates to skip over when joining regions, gaps between region larger than this will not be connected. Parameters specify the maximum number of filtered gates between and along a ray. Set these parameters to 0 to disable unfolding across filtered gates.

centered [bool, optional] True to apply centering to each sweep after the dealiasing algorithm so that the average number of unfolding is near 0. False does not apply centering which may results in individual sweeps under or over folded by the nyquist interval.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_dealias_unwrap_phase(procstatus, dscfg, radar_list=None)`

Dealiases the Doppler velocity field using multi-dimensional phase unwrapping

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

unwrap_unit [{'ray', 'sweep', 'volume'}, optional] Unit to unwrap independently. 'ray' will unwrap each ray individually, 'sweep' each sweep, and 'volume' will unwrap the entire volume in a single pass. 'sweep', the default, often gives superior results when the lower sweeps of the radar volume are contaminated by clutter. 'ray' does not use the gatefilter parameter and rays where gates ared masked will result in poor dealiasing for that ray.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_dem(procstatus, dscfg, radar_list=None)`

Gets DEM data and put it in radar coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

keep_in_memory [int. Dataset keyword] if set keeps the COSMO data dict, the COSMO coordinates dict and the COSMO field in radar coordinates in memory. Default False

regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and there is no need to compute a new COSMO field if the COSMO data has not changed. Default False

dem_field [str. Dataset keyword] name of the DEM field to process

demfile [str. Dataset keyword] Name of the file containing the DEM data

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_differential_phase`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the differential phase from the spectral differential phase and the spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_differential_phase_iq`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the differential phase from the horizontal and vertical IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

phase_offset [float. Dataset keyword] The system differential phase offset to remove

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_differential_reflectivity`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes differential reflectivity from the horizontal and vertical spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_differential_reflectivity_iq**(*procstatus*, *dscfg*, *radar_list=None*)
Computes differential reflectivity from the horizontal and vertical IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal

lag [int] The time lag to use in the estimators

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_echo_filter**(*procstatus*, *dscfg*, *radar_list=None*)
Masks all echo types that are not of the class specified in keyword echo_type

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

echo_type [int or list of ints] The type of echoes to keep: 1 noise, 2 clutter, 3 precipitation. Default 3

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_echo_id**(*procstatus*, *dscfg*, *radar_list=None*)
identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Precipitation

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_estimate_phidp0`**(*procstatus*, *dscfg*, *radar_list=None*)

estimates the system differential phase offset at each ray

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_evp`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes enhanced vertical profiles, by averaging over height levels PPI data.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

lat, lon [float] latitude and longitude of the point of interest [deg]

latlon_tol [float] tolerance in latitude and longitude in deg. Default 0.0005

delta_rng, delta_azi [float] maximum range distance [m] and azimuth distance [degree] from the central point of the evp containing data to average. Default 5000. and 10.

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 250.

avg_type [str] The type of averaging to perform. Can be either "mean" or "median" Default "mean"

nvalid_min [int] Minimum number of valid points to consider the data valid when performing the averaging. Default 1

interp_kind [str] type of interpolation when projecting to vertical grid: 'none', or 'nearest', etc. Default 'none'. 'none' will select from all data points within the regular grid height bin the closest to the center of the bin. 'nearest' will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the EVP and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.`**`process_fft`**(*procstatus*, *dscfg*, *radar_list=None*)

Compute the Doppler spectra form the IQ data with a Fourier transform

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

window [list of str] Parameters of the window used to obtain the spectra. The parameters are the ones corresponding to function scipy.signal.windows.get_window. It can also be [‘None’].

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_fields_diff`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the field difference between RADAR001 and radar002, i.e. RADAR001-RADAR002. Assumes both radars have the same geometry

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing a radar object containing the field differences

ind_rad [int] radar index

`pyrad.proc.`**`process_filter_0Doppler`**(*procstatus*, *dscfg*, *radar_list=None*)

Function to filter the 0-Doppler line bin and neighbours of the Doppler spectra

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

filter_width [float] The Doppler filter width. Default 0.

filter_units [str] Can be ‘m/s’ or ‘Hz’. Default ‘m/s’

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_filter_snr**(*procstatus*, *dscfg*, *radar_list=None*)

filters out low SNR echoes

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

SNRmin [float. Dataset keyword] The minimum SNR to keep the data.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_filter_spectra_noise**(*procstatus*, *dscfg*, *radar_list=None*)

Filter the noise of the Doppler spectra by clipping any data below the noise level plus a margin

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

clipping_level [float] The clipping level [dB above noise level]. Default 10.

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_filter_srhohv**(*procstatus*, *dscfg*, *radar_list=None*)

Filter Doppler spectra as a function of spectral RhoHV

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

sRhoHV_threshold [float] Data with sRhoHV module above this threshold will be filtered. Default 1.

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_filter_vel_diff`**(*procstatus*, *dscfg*, *radar_list=None*)

filters out range gates that could not be used for Doppler velocity estimation

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_filter_visibility`**(*procstatus*, *dscfg*, *radar_list=None*)

filters out rays gates with low visibility and corrects the reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

VISmin [float. Dataset keyword] The minimum visibility to keep the data.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_fixed_rng`**(*procstatus*, *dscfg*, *radar_list=None*)

Obtains radar data at a fixed range

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of strings. Dataset keyword] The fields we want to extract

rng [float. Dataset keyword] The fixed range [m]

RngTol [float. Dataset keyword] The tolerance between the nominal range and the radar range

ele_min, ele_max, azi_min, azi_max [floats. Dataset keyword] The azimuth and elevation limits of the data [deg]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and metadata at the point of interest

ind_rad [int] radar index

pyrad.proc.**process_fixed_rng_span**(*procstatus*, *dscfg*, *radar_list=None*)
For each azimuth-elevation gets the data within a fixed range span and computes a user-defined statistic: mean, min, max, mode, median

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of strings. Dataset keyword] The fields we want to extract

rmin, rmax [float. Dataset keyword] The range limits [m]

ele_min, ele_max, azi_min, azi_max [floats. Dataset keyword] The azimuth and elevation limits of the data [deg]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and metadata at the point of interest

ind_rad [int] radar index

pyrad.proc.**process_gc_monitoring**(*procstatus*, *dscfg*, *radar_list=None*)
computes ground clutter monitoring statistics

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

excessgatespath [str. Config keyword] The path to the gates in excess of quantile location

excessgates_fname [str. Dataset keyword] The name of the gates in excess of quantile file

datatype [list of string. Dataset keyword] The input data types

step [float. Dataset keyword] The width of the histogram bin. Default is None. In that case the default step in function get_histogram_bins is used

regular_grid [Boolean. Dataset keyword] Whether the radar has a Boolean grid or not. Default False

val_min [Float. Dataset keyword] Minimum value to consider that the gate has signal. Default None

filter_prec [str. Dataset keyword] Give which type of volume should be filtered. None, no filtering; keep_wet, keep wet volumes; keep_dry, keep dry volumes.

rmax_prec [float. Dataset keyword] Maximum range to consider when looking for wet gates [m]

percent_prec_max [float. Dataset keyword] Maxim percentage of wet gates to consider the volume dry

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [Radar] radar object containing histogram data

ind_rad [int] radar index

pyrad.proc.**process_grid**(*procstatus*, *dscfg*, *radar_list=None*)
Puts the radar data in a regular grid

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

gridconfig [dictionary. Dataset keyword] Dictionary containing some or all of this keywords: xmin, xmax, ymin, ymax, zmin, zmax : floats

minimum and maximum horizontal distance from grid origin [km] and minimum and maximum vertical distance from grid origin [m] Defaults -40, 40, -40, 40, 0., 10000.

hres, vres [floats] horizontal and vertical grid resolution [m] Defaults 1000., 500.

latorig, lonorig, altorig [floats] latitude and longitude of grid origin [deg] and altitude of grid origin [m MSL] Defaults the latitude, longitude and altitude of the radar

wfunc [str. Dataset keyword] the weighting function used to combine the radar gates close to a grid point. Possible values BARNES, BARNES2, CRESSMAN, NEAREST Default NEAREST

roif_func [str. Dataset keyword] the function used to compute the region of interest. Possible values: dist_beam, constant

roi [float. Dataset keyword] the (minimum) radius of the region of interest in m. Default half the largest resolution

beamwidth [float. Dataset keyword] the radar antenna beamwidth [deg]. If None that of the key radar_beam_width_h in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present a default 1 deg value will be used

beam_spacing [float. Dataset keyword] the beam spacing, i.e. the ray angle resolution [deg]. If None, that of the attribute ray_angle_res of the radar object will be used. If the attribute is None a default 1 deg value will be used

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the gridded data

ind_rad [int] radar index

`pyrad.proc.`**`process_grid_fields_diff`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes grid field differences

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing a radar object containing the field differences

ind_rad [int] radar index

pyrad.proc.**process_grid_mask**(*procstatus*, *dscfg*, *radar_list=None*)

Mask data. Puts True if data is above a certain threshold and false otherwise.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of Radar objects] Optional. list of radar objects

threshold [float] Threshold used for the mask. Values below threshold are set to False. Above threshold are set to True. Default 0.

x_dir_ext, y_dir_ext [int] Number of pixels by which to extend the mask on each side of the west-east direction and south-north direction

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_grid_point**(*procstatus*, *dscfg*, *radar_list=None*)

Obtains the grid data at a point location.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

latlon [boolean. Dataset keyword] if True position is obtained from latitude, longitude information, otherwise position is obtained from grid index (iz, iy, ix).

lon [float. Dataset keyword] the longitude [deg]. Use when latlon is True.

lat [float. Dataset keyword] the latitude [deg]. Use when latlon is True.

alt [float. Dataset keyword] altitude [m MSL]. Use when latlon is True.

iz, iy, ix [int. Dataset keyword] The grid indices. Use when latlon is False

latlonTol [float. Dataset keyword] latitude-longitude tolerance to determine which grid point to use [deg]

altTol [float. Dataset keyword] Altitude tolerance to determine which grid point to use [deg]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and metadata at the point of interest

ind_rad [int] radar index

pyrad.proc.**process_grid_texture**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the 2D texture of a gridded field

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

xwind, ywind [int] The size of the local window in the x and y axis. Default 7

fill_value [float] The value with which to fill masked data. Default np.NaN

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing a radar object containing the field differences

ind_rad [int] radar index

`pyrad.proc.`**`process_grid_time_stats`**(*procstatus*, *dscfg*, *radar_list=None*)
computes the temporal statistics of a field

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. If -1 the statistics are going to be performed over the entire data. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

lin_trans: int. Dataset keyword If 1 apply linear transformation before averaging

use_nan [bool. Dataset keyword] If true non valid data will be used

nan_value [float. Dataset keyword] The value of the non valid data. Default 0

stat: string. Dataset keyword Statistic to compute: Can be mean, std, cov, min, max. Default mean

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_grid_time_stats2`**(*procstatus*, *dscfg*, *radar_list=None*)
computes temporal statistics of a field

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. If -1 the statistics are going to be performed over the entire data. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

stat: string. Dataset keyword Statistic to compute: Can be median, mode, percentileXX

use_nan [bool. Dataset keyword] If true non valid data will be used

nan_value [float. Dataset keyword] The value of the non valid data. Default 0

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_hydroclass`**(*procstatus*, *dscfg*, *radar_list=None*)

Classifies precipitation echoes

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

HYDRO_METHOD [string. Dataset keyword] The hydrometeor classification method. One of the following: SEMISUPERVISED

RADARCENTROIDS [string. Dataset keyword] Used with HYDRO_METHOD SEMISUPERVISED. The name of the radar of which the derived centroids will be used. One of the following: A Albis, L Lema, P Plaine Morte, DX50

compute_entropy [bool. Dataset keyword] If true the entropy is computed and the field hydroclass_entropy is output

output_distances [bool. Dataset keyword] If true the de-mixing algorithm based on the distances to the centroids is computed and the field proportions of each hydrometeor in the radar range gate is output

vectorize [bool. Dataset keyword] If true a vectorized version of the algorithm is used

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_hzt`**(*procstatus*, *dscfg*, *radar_list=None*)

Gets iso0 degree data in HZT format and put it in radar coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

metranet_read_lib [str. Global keyword] Type of METRANET reader library used to read the data. Can be 'C' or 'python'

datatype [string. Dataset keyword] arbitrary data type

keep_in_memory [int. Dataset keyword] if set keeps the COSMO data dict, the COSMO coordinates dict and the COSMO field in radar coordinates in memory

regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and there is no need to compute a new COSMO field if the COSMO data has not changed

cosmo_type [str. Dataset keyword] name of the COSMO field to process. Default TEMP

cosmo_variables [list of strings. Dataset keyword] Py-art name of the COSMO fields. Default temperature

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_hzt_coord`(*procstatus*, *dscfg*, *radar_list=None*)

Gets the HZT indices corresponding to each HZT coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

metranet_read_lib [str. Global keyword] Type of METRANET reader library used to read the data. Can be 'C' or 'python'

datatype [string. Dataset keyword] arbitrary data type

cosmopath [string. General keyword] path where to store the look up table

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_hzt_lookup_table`(*procstatus*, *dscfg*, *radar_list=None*)

Gets HZT data and put it in radar coordinates using look up tables computed or loaded when initializing

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

metranet_read_lib [str. Global keyword] Type of METRANET reader library used to read the data. Can be 'C' or 'python'

datatype [string. Dataset keyword] arbitrary data type

lookup_table [int. Dataset keyword] if set a pre-computed look up table for the COSMO coordinates is loaded. Otherwise the look up table is computed taking the first radar object as reference

regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and therefore there is no need to interpolate the COSMO field in memory to the current radar grid

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_ifft`(*procstatus*, *dscfg*, *radar_list=None*)

Compute the Doppler spectrum width from the spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_intercomp**(*procstatus*, *dscfg*, *radar_list=None*)

intercomparison between two radars

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

coloc_data_dir [string. Dataset keyword] name of the directory containing the csv file with colocated data

coloc_radars_name [string. Dataset keyword] string identifying the radar names

azi_tol [float. Dataset keyword] azimuth tolerance between the two radars. Default 0.5 deg

ele_tol [float. Dataset keyword] elevation tolerance between the two radars. Default 0.5 deg

rng_tol [float. Dataset keyword] range tolerance between the two radars. Default 50 m

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing a dictionary with intercomparison data and the key "final" which contains a boolean that is true when all volumes have been processed

ind_rad [int] radar index

pyrad.proc.**process_intercomp_fields**(*procstatus*, *dscfg*, *radar_list=None*)

intercomparison between two radars

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing a dictionary with intercomparison data

ind_rad [int] radar index

pyrad.proc.**process_intercomp_time_avg**(*procstatus*, *dscfg*, *radar_list=None*)

intercomparison between the average reflectivity of two radars

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

coloc_data_dir [string. Dataset keyword] name of the directory containing the csv file with colocated data

coloc_radars_name [string. Dataset keyword] string identifying the radar names

azi_tol [float. Dataset keyword] azimuth tolerance between the two radars. Default 0.5 deg

ele_tol [float. Dataset keyword] elevation tolerance between the two radars. Default 0.5 deg

rng_tol [float. Dataset keyword] range tolerance between the two radars. Default 50 m

clt_max [int. Dataset keyword] maximum number of samples that can be clutter contaminated. Default 100 i.e. all

phi_excess_max [int. Dataset keyword] maximum number of samples that can have excess instantaneous PhiDP. Default 100 i.e. all

non_rain_max [int. Dataset keyword] maximum number of samples that can be no rain. Default 100 i.e. all

phi_avg_max [float. Dataset keyword] maximum average PhiDP allowed. Default 600 deg i.e. any

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing a dictionary with intercomparison data and the key "final" which contains a boolean that is true when all volumes have been processed

ind_rad [int] radar index

`pyrad.proc.`**`process_kdp_leastsquare_double_window`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes specific differential phase using a piecewise least square method

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rwinds [float. Dataset keyword] The length of the short segment for the least square method [m]

rwindl [float. Dataset keyword] The length of the long segment for the least square method [m]

Zthr [float. Dataset keyword] The threshold defining which estimated data to use [dBZ]

vectorize [Bool. Dataset keyword] Whether to vectorize the KDP processing. Default false

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_kdp_leastsquare_single_window**(*procstatus*, *dscfg*, *radar_list=None*)
Computes specific differential phase using a piecewise least square method

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rwind [float. Dataset keyword] The length of the segment for the least square method [m]

vectorize [bool. Dataset keyword] Whether to vectorize the KDP processing. Default false

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_l**(*procstatus*, *dscfg*, *radar_list=None*)
Computes L parameter

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_mean_phase_iq**(*procstatus*, *dscfg*, *radar_list=None*)
Computes the mean phase from the horizontal or vertical IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_melting_layer**(*procstatus*, *dscfg*, *radar_list=None*)
Detects the melting layer

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_monitoring`**(*procstatus*, *dscfg*, *radar_list=None*)
computes monitoring statistics

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

step [float. Dataset keyword] The width of the histogram bin. Default is None. In that case the default step in function get_histogram_bins is used

max_rays [int. Dataset keyword] The maximum number of rays per sweep used when computing the histogram. If set above 0 the number of rays per sweep will be checked and if above max_rays the last rays of the sweep will be removed

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [Radar] radar object containing histogram data

ind_rad [int] radar index

`pyrad.proc.`**`process_noise_power`**(*procstatus*, *dscfg*, *radar_list=None*)
Computes the noise power from the spectra

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

units [str] The units of the returned signal. Can be 'ADU', 'dBADU' or 'dBm'

navg [int] Number of spectra averaged

rmin [int] Range from which the data is used to estimate the noise

nnoise_min [int] Minimum number of samples to consider the estimated noise power valid

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_normalize_luminosity`**(*procstatus*, *dscfg*, *radar_list=None*)
Normalize the data by the sinus of the sun elevation. The sun elevation is computed at the central pixel.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_occurrence`**(*procstatus*, *dscfg*, *radar_list=None*)

computes the frequency of occurrence of data. It looks only for gates where data is present.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

regular_grid [Boolean. Dataset keyword] Whether the radar has a Boolean grid or not. Default False

rmin, rmax [float. Dataset keyword] minimum and maximum ranges where the computation takes place. If -1 the whole range is considered. Default is -1

val_min [Float. Dataset keyword] Minimum value to consider that the gate has signal. Default None

filter_prec [str. Dataset keyword] Give which type of volume should be filtered. None, no filtering; keep_wet, keep wet volumes; keep_dry, keep dry volumes.

rmax_prec [float. Dataset keyword] Maximum range to consider when looking for wet gates [m]

percent_prec_max [float. Dataset keyword] Maxim percentage of wet gates to consider the volume dry

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_occurrence_period`**(*procstatus*, *dscfg*, *radar_list=None*)

computes the frequency of occurrence over a long period of time by adding together shorter periods

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

regular_grid [Boolean. Dataset keyword] Whether the radar has a Boolean grid or not. Default False

rmin, rmax [float. Dataset keyword] minimum and maximum ranges where the computation takes place. If -1 the whole range is considered. Default is -1

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_outlier_filter`(*procstatus*, *dscfg*, *radar_list=None*)

filters out gates which are outliers respect to the surrounding

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

threshold [float. Dataset keyword] The distance between the value of the examined range gate and the median of the surrounding gates to consider the gate an outlier

nb [int. Dataset keyword] The number of neighbours (to one side) to analyse. i.e. 2 would correspond to 24 gates

nb_min [int. Dataset keyword] Minimum number of neighbouring gates to consider the examined gate valid

percentile_min, percentile_max [float. Dataset keyword] gates below (above) these percentiles (computed over the sweep) are considered potential outliers and further examined

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phidp_kdp_Kalman`(*procstatus*, *dscfg*, *radar_list=None*)

Computes specific differential phase and differential phase using the Kalman filter as proposed by Schneebeli et al. The data is assumed to be clutter free and continous

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

parallel [boolean. Dataset keyword] if set use parallel computing

get_phidp [boolean. Datset keyword] if set the PhiDP computed by integrating the resultant KDP is added to the radar field

frequency [float. Dataset keyword] the radar frequency [Hz]. If None that of the key frequency in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present it will be assumed that the radar is C band

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_phidp_kdp_Maesaka**(*procstatus*, *dscfg*, *radar_list=None*)

Estimates PhiDP and KDP using the method by Maesaka. This method only retrieves data in rain (i.e. below the melting layer)

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

fzl [float. Dataset keyword] The freezing level height [m]. Default 2000.

ml_thickness [float. Dataset keyword] The melting layer thickness in meters. Default 700.

beamwidth [float. Dataset keyword] the antenna beamwidth [deg]. If None that of the keys radar_beam_width_h or radar_beam_width_v in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the beamwidth will be set to None

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_phidp_kdp_Vulpiani**(*procstatus*, *dscfg*, *radar_list=None*)

Computes specific differential phase and differential phase using the method developed by Vulpiani et al. The data is assumed to be clutter free and monotonous

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rwind [float. Dataset keyword] The length of the segment [m]

n_iter [int. Dataset keyword] number of iterations

interp [boolean. Dataset keyword] if set non valid values are interpolated using neighbouring valid values

parallel [boolean. Dataset keyword] if set use parallel computing

get_phidp [boolean. Datset keyword] if set the PhiDP computed by integrating the resultant KDP is added to the radar field

frequency [float. Dataset keyword] the radar frequency [Hz]. If None that of the key frequency in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present it will be assumed that the radar is C band

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_phidp_kdp_lp`**(*procstatus*, *dscfg*, *radar_list=None*)

Estimates PhiDP and KDP using a linear programming algorithm. This method only retrieves data in rain (i.e. below the melting layer)

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

fzl [float. Dataset keyword] The freezing level height [m]. Default 2000.

ml_thickness [float. Dataset keyword] The melting layer thickness in meters. Default 700.

beamwidth [float. Dataset keyword] the antenna beamwidth [deg]. If None that of the keys radar_beam_width_h or radar_beam_width_v in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the beamwidth will be set to None

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_pixel_filter`**(*procstatus*, *dscfg*, *radar_list=None*)

Masks all pixels that are not of the class specified in keyword pixel_type

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

pixel_type [int or list of ints] The type of pixels to keep: 0 No data, 1 Below threshold, 2 Above threshold. Default 2

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_point_measurement`**(*procstatus*, *dscfg*, *radar_list=None*)

Obtains the radar data at a point location.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

single_point [boolean. Dataset keyword] if True only one gate per radar volume is going to be kept. Otherwise all gates within the azimuth and elevation tolerance are going to be kept. This is useful to extract all data from fixed pointing scans. Default True

latlon [boolean. Dataset keyword] if True position is obtained from latitude, longitude information, otherwise position is obtained from antenna coordinates (range, azimuth, elevation).

truealt [boolean. Dataset keyword] if True the user input altitude is used to determine the point of interest. if False use the altitude at a given radar elevation ele over the point of interest.

lon [float. Dataset keyword] the longitude [deg]. Use when latlon is True.

lat [float. Dataset keyword] the latitude [deg]. Use when latlon is True.

alt [float. Dataset keyword] altitude [m MSL]. Use when latlon is True.

ele [float. Dataset keyword] radar elevation [deg]. Use when latlon is False or when latlon is True and truealt is False

azi [float. Dataset keyword] radar azimuth [deg]. Use when latlon is False

rng [float. Dataset keyword] range from radar [m]. Use when latlon is False

AziTol [float. Dataset keyword] azimuthal tolerance to determine which radar azimuth to use [deg]

EleTol [float. Dataset keyword] elevation tolerance to determine which radar elevation to use [deg]

RngTol [float. Dataset keyword] range tolerance to determine which radar bin to use [m]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and metadata at the point of interest

ind_rad [int] radar index

`pyrad.proc.`**`process_pol_variables`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the polarimetric variables from the complex spectra

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal. Default False

smooth_window [int or None] Size of the moving Gaussian smoothing window. If none no smoothing will be applied. Default None

variables [list of str] list of variables to compute. Default dBZ

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_pol_variables_iq**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the polarimetric variables from the IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal

lag [int] The time lag to use in the estimators

direction [str] The convention used in the Doppler mean field. Can be negative_away or negative_towards

variables [list of str] list of variables to compute. Default dBZ

phase_offset [float. Dataset keyword] The system differential phase offset to remove

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_qvp**(*procstatus*, *dscfg*, *radar_list=None*)

Computes quasi vertical profiles, by averaging over height levels PPI data.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

angle [int or float] If the radar object contains a PPI volume, the sweep number to use, if it contains an RHI volume the elevation angle. Default 0.

ang_tol [float] If the radar object contains an RHI volume, the tolerance in the elevation angle for the conversion into PPI

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 50

avg_type [str] The type of averaging to perform. Can be either "mean" or "median" Default "mean"

nvalid_min [int] Minimum number of valid points to accept average. Default 30.

interp_kind [str] type of interpolation when projecting to vertical grid: 'none', or 'nearest', etc. Default 'none' 'none' will select from all data points within the regular grid height bin the closest to the center of the bin. 'nearest' will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the QVP and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.`**`process_radar_resampling`**(*procstatus*, *dscfg*, *radar_list=None*)

Resamples the radar data to mimic another radar with different geometry and antenna pattern

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries]

datatype [list of string. Dataset keyword] The input data types

antennaType [str. Dataset keyword] Type of antenna of the radar we want to get the view from. Can be AZIMUTH, ELEVATION, LOWBEAM, HIGHBEAM

par_azimuth_antenna [dict. Global keyword] Dictionary containing the parameters of the PAR azimuth antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

par_elevation_antenna [dict. Global keyword] Dictionary containing the parameters of the PAR elevation antenna, i.e. name of the file with the antenna azimuth pattern and fixed antenna angle

asr_lowbeam_antenna [dict. Global keyword] Dictionary containing the parameters of the ASR low beam antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

asr_highbeam_antenna [dict. Global keyword] Dictionary containing the parameters of the ASR high beam antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

target_radar_pos [dict. Global keyword] Dictionary containing the latitude, longitude and altitude of the radar we want to get the view from. If not specifying it will assume the radar is collocated

change_antenna_pattern [Bool. Dataset keyword] If true the target radar has a different antenna pattern than the observations radar. Default True

rhi_resolution [Bool. Dataset keyword] Resolution of the synthetic RHI used to compute the data as viewed from the synthetic radar [deg]. Default 0.5

max_altitude [float. Dataset keyword] Max altitude of the data to use when computing the view from the synthetic radar [m MSL]. Default 12000.

latlon_tol [float. Dataset keyword] The tolerance in latitude and longitude to determine which synthetic radar gates are co-located with real radar gates [deg]. Default 0.04

alt_tol [float. Dataset keyword] The tolerance in altitude to determine which synthetic radar gates are co-located with real radar gates [m]. Default 1000.

distance_upper_bound [float. Dataset keyword] The maximum distance where to look for a neighbour when determining which synthetic radar gates are co-located with real radar gates [m]. Default 1000.

use_cKDTree [Bool. Dataset keyword] Which function to use to find co-located real radar gates with the synthetic radar. If True a function using cKDTree from scipy.spatial is used. This function uses parameter distance_upper_bound. If False a native implementation is used that takes as parameters latlon_tol and alt_tol. Default True.

pattern_thres [float. Dataset keyword] The minimum of the sum of the weights given to each value in order to consider the weighted quantile valid. It is related to the number of valid data points

data_is_log [dict. Dataset keyword] Dictionary specifying for each field if it is in log (True) or linear units (False). Default False

use_nans [dict. Dataset keyword] Dictionary specifying whether the nans have to be used in the computation of the statistics for each field. Default False

nan_value [dict. Dataset keyword] Dictionary with the value to use to substitute the NaN values when computing the statistics of each field. Default 0

moving_angle_min, moving_angle_max: float. Dataset keyword The minimum and maximum azimuth angle (deg) of the target radar. Default 0, 360.

ray_res: float Ray resolution (deg). Default 1 deg.

rng_min, rng_max: The minimum and maximum range of the target radar (m). Default 0, 100000

rng_res [float] The target radar range resolution (m). Default 100.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the new radar

ind_rad [int] radar index

`pyrad.proc.`**`process_radial_noise_hs`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the radial noise from the signal power using the Hildebrand and Sekhon 1974 method

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

rmin [float. Dataset keyword] The minimum range from which to start the computation

nbins_min [int. Dataset keyword] The minimum number of noisy gates to consider the estimation valid

max_std_pwr [float. Dataset keyword] The maximum standard deviation of the noise power to consider the estimation valid

get_noise_pos [bool. Dataset keyword] If True a field flagging the position of the noisy gets will be returned

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_radial_noise_ivic`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the radial noise from the signal power using the Ivic 2013 method

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

npulses_ray [int] Default number of pulses used in the computation of the ray. If the number of pulses is not in radar.instrument_parameters this will be used instead. Default 30

ngates_min: int minimum number of gates with noise to consider the retrieval valid. Default 800

iterations: int number of iterations in step 7. Default 10.

get_noise_pos [bool] If true an additional field with gates containing noise according to the algorithm is produced

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_radial_velocity`**(*procstatus*, *dscfg*, *radar_list=None*)

Estimates the radial velocity respect to the radar from the wind velocity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

latitude, longitude [float] arbitrary coordinates [deg] from where to compute the radial velocity. If any of them is None it will be the radar position

altitude [float] arbitrary altitude [m MSL] from where to compute the radial velocity. If None it will be the radar altitude

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_rainfall_accumulation`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes rainfall accumulation fields

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. If -1 the statistics are going to be performed over the entire data. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

use_nan [bool. Dataset keyword] If true non valid data will be used

nan_value [float. Dataset keyword] The value of the non valid data. Default 0

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_rainrate**(*procstatus*, *dscfg*, *radar_list=None*)

Estimates rainfall rate from polarimetric moments

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

RR_METHOD [string. Dataset keyword] The rainfall rate estimation method. One of the following: Z, ZPoly, KDP, A, ZKDP, ZA, hydro

alpha, beta [float] factor and exponent of the R-Var power law R = alpha*Var^Beta. Default value depending on RR_METHOD. Z (0.0376, 0.6112), KDP (None, None), A (None, None)

alphaz, betaz [float] factor and exponent of the R-Z power law R = alpha*Z^Beta. Default value (0.0376, 0.6112)

alphazr, betazr [float] factor and exponent of the R-Z power law R = alpha*Z^Beta applied to rain in method hydro. Default value (0.0376, 0.6112)

alphazs, betazs [float] factor and exponent of the R-Z power law R = alpha*Z^Beta applied to solid precipitation in method hydro. Default value (0.1, 0.5)

alphakdp, betakdp [float] factor and exponent of the R-KDP power law R = alpha*KDP^Beta. Default value (None, None)

alphaa, betaa [float] factor and exponent of the R-Ah power law R = alpha*Ah^Beta. Default value (None, None)

thresh [float] In hybrid methods, Rainfall rate threshold at which the retrieval method used changes [mm/h]. Default value depending on RR_METHOD. ZKDP 10, ZA 10, hydro 10

mp_factor [float] Factor by which the Z-R relation is multiplied in the melting layer in method hydro. Default 0.6

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_raw**(*procstatus*, *dscfg*, *radar_list=None*)

Dummy function that returns the initial input data set

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_raw_grid**(*procstatus*, *dscfg*, *radar_list=None*)
Dummy function that returns the initial input data set

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_raw_iq**(*procstatus*, *dscfg*, *radar_list=None*)
Dummy function that returns the initial input data set

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_raw_spectra**(*procstatus*, *dscfg*, *radar_list=None*)
Dummy function that returns the initial input data set

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_rcs**(*procstatus*, *dscfg*, *radar_list=None*)
Computes the radar cross-section (assuming a point target) from radar reflectivity.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

kw2 [float. Dataset keyowrd] The water constant

pulse_width [float. Dataset keyowrd] The pulse width [s]

beamwidthv [float. Global keyword] The vertical polarization antenna beamwidth [deg]. Used if input is vertical reflectivity

beamwidthh [float. Global keyword] The horizontal polarization antenna beamwidth [deg]. Used if input is horizontal reflectivity

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_rcs_pr**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the radar cross-section (assuming a point target) from radar reflectivity by first computing the received power and then the RCS from it.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

AntennaGainH, AntennaGainV [float. Dataset keyword] The horizontal (vertical) polarization antenna gain [dB]. If None it will be obtained from the attribute instrument_parameters of the radar object

txpwrh, txpwrv [float. Dataset keyword] The transmitted power of the horizontal (vertical) channel [dBm]. If None it will be obtained from the attribute radar_calibration of the radar object

mflossh, mflossv [float. Dataset keyword] The matching filter losses of the horizontal (vertical) channel [dB]. If None it will be obtained from the attribute radar_calibration of the radar object. Defaults to 0

radconsth, radconstv [float. Dataset keyword] The horizontal (vertical) channel radar constant. If None it will be obtained from the attribute radar_calibration of the radar object

lrxh, lrxv [float. Global keyword] The horizontal (vertical) receiver losses from the antenna feed to the reference point. [dB] positive value. Default 0

ltxh, ltxv [float. Global keyword] The horizontal (vertical) transmitter losses from the output of the high power amplifier to the antenna feed. [dB] positive value. Default 0

lradomeh, lradomev [float. Global keyword] The 1-way dry radome horizontal (vertical) channel losses. [dB] positive value. Default 0.

attg [float. Dataset keyword] The gas attenuation [dB/km]. If none it will be obtained from the attribute radar_calibration of the radar object or assigned according to the radar frequency. Defaults to 0.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_reflectivity`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes reflectivity from the spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_reflectivity_iq`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes reflectivity from the IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_rhohv`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes RhoHV from the complex spectras

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_rhohv_iq`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes RhoHV from the horizontal and vertical IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal

lag [int] Time lag used in the computation

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_rhohv_rain**(*procstatus*, *dscfg*, *radar_list=None*)

Keeps only suitable data to evaluate the 80 percentile of RhoHV in rain

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] minimum range where to look for rain [m]. Default 1000.

rmax [float. Dataset keyword] maximum range where to look for rain [m]. Default 50000.

Zmin [float. Dataset keyword] minimum reflectivity to consider the bin as precipitation [dBZ]. Default 20.

Zmax [float. Dataset keyword] maximum reflectivity to consider the bin as precipitation [dBZ] Default 40.

ml_thickness [float. Dataset keyword] assumed thickness of the melting layer. Default 700.

fzl [float. Dataset keyword] The default freezing level height. It will be used if no temperature field name is specified or the temperature field is not in the radar object. Default 2000.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_roi**(*procstatus*, *dscfg*, *radar_list=None*)

Obtains the radar data at a region of interest defined by a TRT file or by the user.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

trtfile [str. Dataset keyword] TRT file from which to extract the region of interest

lon_roi, lat_roi [float array. Dataset keyword] latitude and longitude positions defining a region of interest

alt_min, alt_max [float. Dataset keyword] Minimum and maximum altitude of the region of interest. Can be None

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and metadata at the point of interest

ind_rad [int] radar index

`pyrad.proc.`**`process_rqvp`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes range defined quasi vertical profiles, by averaging over height levels PPI data.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 2.

avg_type [str] The type of averaging to perform. Can be either "mean" or "median" Default "mean"

nvalid_min [int] Minimum number of valid points to accept average. Default 30.

interp_kind [str] type of interpolation when projecting to vertical grid: 'none', or 'nearest', etc. Default 'nearest' 'none' will select from all data points within the regular grid height bin the closest to the center of the bin. 'nearest' will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

rmax [float] ground range up to which the data is intended for use [m]. Default 50000.

weight_power [float] Power p of the weighting function 1/abs(grng-(rmax-1))**p given to the data outside the desired range. -1 will set the weight to 0. Default 2.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the QVP and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.`**`process_save_radar`**(*procstatus*, *dscfg*, *radar_list=None*)

Dummy function that allows to save the entire radar object

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_selfconsistency_bias**(*procstatus*, *dscfg*, *radar_list=None*)
Estimates the reflectivity bias by means of the selfconsistency algorithm by Gourley

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

parametrization [str] The type of parametrization for the self-consistency curves. Can be 'None', 'Gourley', 'Wolfensberger', 'Louf', 'Gorgucci' or 'Vaccarono' 'None' will use tables from config files. Default 'None'.

fzl [float. Dataset keyword] Default freezing level height. Default 2000.

rsmooth [float. Dataset keyword] length of the smoothing window [m]. Default 2000.

min_rhohv [float. Dataset keyword] minimum valid RhoHV. Default 0.92

filter_rain [Bool. Dataset keyword] If True the hydrometeor classification is used to filter out gates that are not rain. Default True

max_phidp [float. Dataset keyword] maximum valid PhiDP [deg]. Default 20.

ml_thickness [float. Dataset keyword] Melting layer thickness [m]. Default 700.

rcell [float. Dataset keyword] length of continuous precipitation to consider the precipitation cell a valid phidp segment [m]. Default 15000.

dphidp_min [float. Dataset keyword] minimum phase shift [deg]. Default 2.

dphidp_max [float. Dataset keyword] maximum phase shift [deg]. Default 16.

frequency [float. Dataset keyword] the radar frequency [Hz]. If None that of the key frequency in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the selfconsistency will not be computed

check_wet_radome [Bool. Dataset keyword] if True the average reflectivity of the closest gates to the radar is going to be check to find out whether there is rain over the radome. If there is rain no bias will be computed. Default True.

wet_radome_refl [Float. Dataset keyword] Average reflectivity [dBZ] of the gates close to the radar to consider the radome as wet. Default 25.

wet_radome_rng_min, wet_radome_rng_max [Float. Dataset keyword] Min and max range [m] of the disk around the radar used to compute the average reflectivity to determine whether the radome is wet. Default 2000 and 4000.

wet_radome_ngates_min [int] Minimum number of valid gates to consider that the radome is wet. Default 180

valid_gates_only [Bool] If True the reflectivity bias obtained for each valid ray is going to be assigned only to gates of the segment used. That will give more weight to longer segments when computing the total bias. Default False

keep_points [Bool] If True the ZDR, ZH and KDP of the gates used in the self- consistency algorithm are going to be stored for further analysis. Default False

rkdp [float] The length of the window used to compute KDP with the single window least square method [m]. Default 6000.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_selfconsistency_bias2**(*procstatus*, *dscfg*, *radar_list=None*)

Estimates the reflectivity bias by means of the selfconsistency algorithm by Gourley

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

parametrization [str] The type of parametrization for the self-consistency curves. Can be 'None', 'Gourley', 'Wolfensberger', 'Louf', 'Gorgucci' or 'Vaccarono' 'None' will use tables from config files. Default 'None'.

fzl [float. Dataset keyword] Default freezing level height. Default 2000.

rsmooth [float. Dataset keyword] length of the smoothing window [m]. Default 2000.

min_rhohv [float. Dataset keyword] minimum valid RhoHV. Default 0.92

filter_rain [Bool. Dataset keyword] If True the hydrometeor classification is used to filter out gates that are not rain. Default True

max_phidp [float. Dataset keyword] maximum valid PhiDP [deg]. Default 20.

ml_thickness [float. Dataset keyword] Melting layer thickness [m]. Default 700.

rcell [float. Dataset keyword] length of continuous precipitation to consider the precipitation cell a valid phidp segment [m]. Default 15000.

frequency [float. Dataset keyword] the radar frequency [Hz]. If None that of the key frequency in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the selfconsistency will not be computed

check_wet_radome [Bool. Dataset keyword] if True the average reflectivity of the closest gates to the radar is going to be check to find out whether there is rain over the radome. If there is rain no bias will be computed. Default True.

wet_radome_refl [Float. Dataset keyword] Average reflectivity [dBZ] of the gates close to the radar to consider the radome as wet. Default 25.

wet_radome_rng_min, wet_radome_rng_max [Float. Dataset keyword] Min and max range [m] of the disk around the radar used to compute the average reflectivity to determine whether the radome is wet. Default 2000 and 4000.

wet_radome_ngates_min [int] Minimum number of valid gates to consider that the radome is wet. Default 180

keep_points [Bool] If True the ZDR, ZH and KDP of the gates used in the self- consistency algorithm are going to be stored for further analysis. Default False

bias_per_gate [Bool] If True the bias per gate will be computed

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_selfconsistency_kdp_phidp`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes specific differential phase and differential phase in rain using the selfconsistency between Zdr, Zh and KDP

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of strings. Dataset keyword] The input data types

parametrization [str] The type of parametrization for the self-consistency curves. Can be 'None', 'Gourley', 'Wolfensberger', 'Louf', 'Gorgucci' or 'Vaccarono' 'None' will use tables from config files. Default 'None'.

rsmooth [float. Dataset keyword] length of the smoothing window [m]. Default 2000.

min_rhohv [float. Dataset keyword] minimum valid RhoHV. Default 0.92

filter_rain [Bool. Dataset keyword] If True the hydrometeor classification is used to filter out gates that are not rain. Default True

max_phidp [float. Dataset keyword] maximum valid PhiDP [deg]. Default 20.

ml_thickness [float. Dataset keyword] assumed melting layer thickness [m]. Default 700.

fzl [float. Dataset keyword] The default freezing level height. It will be used if no temperature field name is specified or the temperature field is not in the radar object. Default 2000.

frequency [float. Dataset keyword] the radar frequency [Hz]. If None that of the key frequency in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the selfconsistency will not be computed

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_signal_power`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the signal power in dBm

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

mflossh, mflossv [float. Dataset keyword] The matching filter losses of the horizontal (vertical) channel [dB]. If None it will be obtained from the attribute radar_calibration of the radar object. Defaults to 0

radconsth, radconstv [float. Dataset keyword] The horizontal (vertical) channel radar constant. If None it will be obtained from the attribute radar_calibration of the radar object

lrxh, lrxv [float. Global keyword] The horizontal (vertical) receiver losses from the antenna feed to the reference point. [dB] positive value. Default 0

lradomeh, lradomev [float. Global keyword] The 1-way dry radome horizontal (vertical) channel losses. [dB] positive value. Default 0.

attg [float. Dataset keyword] The gas attenuation [dB/km]. If none it will be obtained from the attribute radar_calibration of the radar object or assigned according to the radar frequency. Defaults to 0.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_smooth_phidp_double_window**(*procstatus*, *dscfg*, *radar_list=None*)
corrects phidp of the system phase and smoothes it using one window

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

rwinds [float. Dataset keyword] The length of the short smoothing window [m]

rwindl [float. Dataset keyword] The length of the long smoothing window [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

Zthr [float. Dataset keyword] The threshold defining wich smoothed data to used [dBZ]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_smooth_phidp_single_window**(*procstatus*, *dscfg*, *radar_list=None*)
corrects phidp of the system phase and smoothes it using one window

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

rwind [float. Dataset keyword] The length of the smoothing window [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_snr`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes SNR

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

output_type [string. Dataset keyword] The output data type. Either SNRh or SNRv

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_spectra_ang_avg`**(*procstatus*, *dscfg*, *radar_list=None*)

Function to average the spectra over the rays. This function is intended mainly for vertically pointing scans. The function assumes the volume is composed of a single sweep, it averages over the number of rays specified by the user and produces a single ray output.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

navg [int] Number of spectra to average. If -1 all spectra will be averaged. Default -1.

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_spectra_point`**(*procstatus*, *dscfg*, *radar_list=None*)

Obtains the spectra or IQ data at a point location.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

single_point [boolean. Dataset keyword] if True only one gate per radar volume is going to be kept. Otherwise all gates within the azimuth and elevation tolerance are going to be kept. This is useful to extract all data from fixed pointing scans. Default True

latlon [boolean. Dataset keyword] if True position is obtained from latitude, longitude information, otherwise position is obtained from antenna coordinates (range, azimuth, elevation). Default False

truealt [boolean. Dataset keyword] if True the user input altitude is used to determine the point of interest. if False use the altitude at a given radar elevation ele over the point of interest. Default True

lon [float. Dataset keyword] the longitude [deg]. Use when latlon is True.

lat [float. Dataset keyword] the latitude [deg]. Use when latlon is True.

alt [float. Dataset keyword] altitude [m MSL]. Use when latlon is True. Default 0.

ele [float. Dataset keyword] radar elevation [deg]. Use when latlon is False or when latlon is True and truealt is False

azi [float. Dataset keyword] radar azimuth [deg]. Use when latlon is False

rng [float. Dataset keyword] range from radar [m]. Use when latlon is False

AziTol [float. Dataset keyword] azimuthal tolerance to determine which radar azimuth to use [deg]. Default 0.5

EleTol [float. Dataset keyword] elevation tolerance to determine which radar elevation to use [deg]. Default 0.5

RngTol [float. Dataset keyword] range tolerance to determine which radar bin to use [m]. Default 50.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and metadata at the point of interest

ind_rad [int] radar index

`pyrad.proc.`**`process_spectral_differential_phase`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the spectral differential phase

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_spectral_differential_reflectivity`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes spectral differential reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal

smooth_window [int or None] Size of the moving Gaussian smoothing window. If none no smoothing will be applied

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_spectral_noise`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the spectral noise

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

units [str] The units of the returned signal. Can be 'ADU', 'dBADU' or 'dBm'

navg [int] Number of spectra averaged

rmin [int] Range from which the data is used to estimate the noise

nnoise_min [int] Minimum number of samples to consider the estimated noise power valid

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_spectral_phase`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the spectral phase

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_spectral_power`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the spectral power

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

units [str] The units of the returned signal. Can be 'ADU', 'dBADU' or 'dBm'

subtract_noise [Bool] If True noise will be subtracted from the signal

smooth_window [int or None] Size of the moving Gaussian smoothing window. If none no smoothing will be applied

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_spectral_reflectivity`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes spectral reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal

smooth_window [int or None] Size of the moving Gaussian smoothing window. If none no smoothing will be applied

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_spectral_rhohv`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the spectral RhoHV

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

subtract_noise [Bool] If True noise will be subtracted from the signal

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_st1_iq`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the statistical test one lag fluctuation from the horizontal or vertical IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_st2_iq`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes the statistical test two lag fluctuation from the horizontal or vertical IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_sun_hits`**(*procstatus*, *dscfg*, *radar_list=None*)

monitoring of the radar using sun hits

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

delev_max [float. Dataset keyword] maximum elevation distance from nominal radar elevation where to look for a sun hit signal [deg]. Default 1.5

dazim_max [float. Dataset keyword] maximum azimuth distance from nominal radar elevation where to look for a sun hit signal [deg]. Default 1.5

elmin [float. Dataset keyword] minimum radar elevation where to look for sun hits [deg]. Default 1.

attg [float. Dataset keyword] gaseous attenuation. Default None

sun_position [string. Datset keyword] The function to compute the sun position to use. Can be 'MF' or 'pysolar'

sun_hit_method [str. Dataset keyword] Method used to estimate the power of the sun hit. Can be HS (Hildebrand and Sekhon 1974) or Ivic (Ivic 2013)

rmin [float. Dataset keyword] minimum range where to look for a sun hit signal [m]. Used in HS method. Default 50000.

hmin [float. Dataset keyword] minimum altitude where to look for a sun hit signal [m MSL]. Default 10000. The actual range from which a sun hit signal will be search will be the minimum between rmin and the range from which the altitude is higher than hmin. Used in HS method. Default 10000.

nbins_min [int. Dataset keyword.] minimum number of range bins that have to contain signal to consider the ray a potential sun hit. Default 20 for HS and 8000 for Ivic.

npulses_ray [int] Default number of pulses used in the computation of the ray. If the number of pulses is not in radar.instrument_parameters this will be used instead. Used in Ivic method. Default 30

iterations: int number of iterations in step 7 of Ivic method. Default 10.

max_std_pwr [float. Dataset keyword] maximum standard deviation of the signal power to consider the data a sun hit [dB]. Default 2. Used in HS method

max_std_zdr [float. Dataset keyword] maximum standard deviation of the ZDR to consider the data a sun hit [dB]. Default 2.

az_width_co [float. Dataset keyword] co-polar antenna azimuth width (convoluted with sun width) [deg]. Default None

el_width_co [float. Dataset keyword] co-polar antenna elevation width (convoluted with sun width) [deg]. Default None

az_width_cross [float. Dataset keyword] cross-polar antenna azimuth width (convoluted with sun width) [deg]. Default None

el_width_cross [float. Dataset keyword] cross-polar antenna elevation width (convoluted with sun width) [deg]. Default None

ndays [int. Dataset keyword] number of days used in sun retrieval. Default 1

coeff_band [float. Dataset keyword] multiplicate coefficient to transform pulse width into receiver bandwidth

frequency [float. Dataset keyword] the radar frequency [Hz]. If None that of the key frequency in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present frequency dependent parameters will not be computed

beamwidth [float. Dataset keyword] the antenna beamwidth [deg]. If None that of the keys radar_beam_width_h or radar_beam_width_v in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the beamwidth dependent parameters will not be computed

pulse_width [float. Dataset keyword] the pulse width [s]. If None that of the key pulse_width in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the pulse width dependent parameters will not be computed

ray_angle_res [float. Dataset keyword] the ray angle resolution [deg]. If None that of the key ray_angle_res in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the ray angle resolution parameters will not be computed

AntennaGainH, AntennaGainV [float. Dataset keyword] the horizontal (vertical) polarization antenna gain [dB]. If None that of the attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the ray angle resolution parameters will not be computed

radar_list [list of Radar objects] Optional. list of radar objects

Returns

sun_hits_dict [dict] dictionary containing a radar object, a sun_hits dict and a sun_retrieval dictionary

ind_rad [int] radar index

`pyrad.proc.process_svp(procstatus, dscfg, radar_list=None)`

Computes slanted vertical profiles, by averaging over height levels PPI data.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

angle [int or float] If the radar object contains a PPI volume, the sweep number to use, if it contains an RHI volume the elevation angle. Default 0.

ang_tol [float] If the radar object contains an RHI volume, the tolerance in the elevation angle for the conversion into PPI. Default 1.

lat, lon [float] latitude and longitude of the point of interest [deg]

latlon_tol [float] tolerance in latitude and longitude in deg. Default 0.0005

delta_rng, delta_azi [float] maximum range distance [m] and azimuth distance [degree] from the central point of the svp containing data to average. Default 5000. and 10.

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 250.

avg_type [str] The type of averaging to perform. Can be either "mean" or "median" Default "mean"

nvalid_min [int] Minimum number of valid points to consider the data valid when performing the averaging. Default 1

interp_kind [str] type of interpolation when projecting to vertical grid: 'none', or 'nearest', etc. Default 'none' 'none' will select from all data points within the regular grid height bin the closest to the center of the bin. 'nearest' will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the svp and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.process_time_avg(procstatus, dscfg, radar_list=None)`

computes the temporal mean of a field

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

lin_trans: int. Dataset keyword If 1 apply linear transformation before averaging

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_time_avg_flag`**(*procstatus*, *dscfg*, *radar_list=None*)

computes a flag field describing the conditions of the data used while averaging

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

phidpmax: float. Dataset keyword maximum PhiDP

beamwidth [float. Dataset keyword] the antenna beamwidth [deg]. If None that of the keys radar_beam_width_h or radar_beam_width_v in attribute instrument_parameters of the radar object will be used. If the key or the attribute are not present the beamwidth will be set to None

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [Radar] radar object

ind_rad [int] radar index

`pyrad.proc.`**`process_time_avg_std`**(*procstatus*, *dscfg*, *radar_list=None*)

computes the average and standard deviation of data. It looks only for gates where data is present.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

regular_grid [Boolean. Dataset keyword] Whether the radar has a Boolean grid or not. Default False

rmin, rmax [float. Dataset keyword] minimum and maximum ranges where the computation takes place. If -1 the whole range is considered. Default is -1

val_min [Float. Dataset keyword] Minimum reflectivity value to consider that the gate has signal. Default None

filter_prec [str. Dataset keyword] Give which type of volume should be filtered. None, no filtering; keep_wet, keep wet volumes; keep_dry, keep dry volumes.

rmax_prec [float. Dataset keyword] Maximum range to consider when looking for wet gates [m]

percent_prec_max [float. Dataset keyword] Maxim percentage of wet gates to consider the volume dry

lin_trans [Boolean. Dataset keyword] If True the data will be transformed into linear units. Default False

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_time_height`**(*procstatus*, *dscfg*, *radar_list=None*)

Produces time height radar objects at a point of interest defined by latitude and longitude. A time-height contains the evolution of the vertical structure of radar measurements above the location of interest.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

lat, lon [float] latitude and longitude of the point of interest [deg]

latlon_tol [float] tolerance in latitude and longitude in deg. Default 0.0005

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 50

interp_kind [str] type of interpolation when projecting to vertical grid: 'none', or 'nearest', etc. Default 'none' 'none' will select from all data points within the regular grid height bin the closest to the center of the bin. 'nearest' will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the QVP and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.`**`process_time_stats`**(*procstatus*, *dscfg*, *radar_list=None*)

computes the temporal statistics of a field

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. If -1 the statistics are going to be performed over the entire data. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

lin_trans: int. Dataset keyword If 1 apply linear transformation before averaging

use_nan [bool. Dataset keyword] If true non valid data will be used

nan_value [float. Dataset keyword] The value of the non valid data. Default 0

stat: string. Dataset keyword Statistic to compute: Can be mean, std, cov, min, max. Default mean

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_time_stats2**(*procstatus*, *dscfg*, *radar_list=None*)

computes the temporal mean of a field

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. If -1 the statistics are going to be performed over the entire data. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

stat: string. Dataset keyword Statistic to compute: Can be median, mode, percentileXX

use_nan [bool. Dataset keyword] If true non valid data will be used

nan_value [float. Dataset keyword] The value of the non valid data. Default 0

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_traj_antenna_pattern**(*procstatus*, *dscfg*, *radar_list=None*, *trajectory=None*)

Process a new array of data volumes considering a plane trajectory. As result a timeseries with the values transposed for a given antenna pattern is created. The result is created when the LAST flag is set.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries]

datatype [list of string. Dataset keyword] The input data types

antennaType [str. Dataset keyword] Type of antenna of the radar we want to get the view from. Can be AZIMUTH, ELEVATION, LOWBEAM, HIGHBEAM

par_azimuth_antenna [dict. Global ekyword] Dictionary containing the parameters of the PAR azimuth antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

par_elevation_antenna [dict. Global keyword] Dictionary containing the parameters of the PAR elevation antenna, i.e. name of the file with the antenna azimuth pattern and fixed antenna angle

asr_lowbeam_antenna [dict. Global keyword] Dictionary containing the parameters of the ASR low beam antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

asr_highbeam_antenna [dict. Global keyword] Dictionary containing the parameters of the ASR high beam antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

target_radar_pos [dict. Global keyword] Dictionary containing the latitude, longitude and altitude of the radar we want to get the view from. If not specifying it will assume the radar is collocated

range_all [Bool. Dataset keyword] If the real radar and the synthetic radar are co-located and this parameter is true the statistics are going to be computed using all the data from range 0 to the position of the plane. Default False

rhi_resolution [Bool. Dataset keyword] Resolution of the synthetic RHI used to compute the data as viewed from the synthetic radar [deg]. Default 0.5

max_altitude [float. Dataset keyword] Max altitude of the data to use when computing the view from the synthetic radar [m MSL]. Default 12000.

latlon_tol [float. Dataset keyword] The tolerance in latitude and longitude to determine which synthetic radar gates are co-located with real radar gates [deg]. Default 0.04

alt_tol [float. Datset keyword] The tolerance in altitude to determine which synthetic radar gates are co-located with real radar gates [m]. Default 1000.

distance_upper_bound [float. Dataset keyword] The maximum distance where to look for a neighbour when determining which synthetic radar gates are co-located with real radar gates [m]. Default 1000.

use_cKDTree [Bool. Dataset keyword] Which function to use to find co-located real radar gates with the synthetic radar. If True a function using cKDTree from scipy.spatial is used. This function uses parameter distance_upper_bound. If False a native implementation is used that takes as parameters latlon_tol and alt_tol. Default True.

pattern_thres [float. Dataset keyword] The minimum of the sum of the weights given to each value in order to consider the weighted quantile valid. It is related to the number of valid data points

data_is_log [dict. Dataset keyword] Dictionary specifying for each field if it is in log (True) or linear units (False). Default False

use_nans [dict. Dataset keyword] Dictionary specyfing whether the nans have to be used in the computation of the statistics for each field. Default False

nan_value [dict. Dataset keyword] Dictionary with the value to use to substitute the NaN values when computing the statistics of each field. Default 0

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

trajectory [Trajectory object] Object holding time series

ind_rad [int] radar index

pyrad.proc.**process_traj_atplane**(*procstatus*, *dscfg*, *radar_list=None*, *trajectory=None*)
Return time series according to trajectory

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

data_is_log [dict. Dataset keyword] Dictionary specifying for each field if it is in log (True) or linear units (False). Default False

ang_tol [float. Dataset keyword] Factor that multiplies the angle resolution. Used when determining the neighbouring rays. Default 1.2

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

trajectory [Trajectory object] Object holding time series

ind_rad [int] radar index

pyrad.proc.**process_traj_lightning**(*procstatus*, *dscfg*, *radar_list=None*, *trajectory=None*)
Return time series according to lightning trajectory

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

data_is_log [dict. Dataset keyword] Dictionary specifying for each field if it is in log (True) or linear units (False). Default False

ang_tol [float. Dataset keyword] Factor that multiplies the angle resolution. Used when determining the neighbouring rays. Default 1.2

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

trajectory [Trajectory object] Object holding time series

ind_rad [int] radar index

pyrad.proc.**process_traj_trt**(*procstatus*, *dscfg*, *radar_list=None*, *trajectory=None*)
Processes data according to TRT trajectory

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

time_tol [float. Dataset keyword] tolerance between reference time of the radar volume and that of the TRT cell [s]. Default 100.

alt_min, alt_max [float. Dataset keyword] Minimum and maximum altitude of the data inside the TRT cell to retrieve [m MSL]. Default None

cell_center [Bool. Dataset keyword] If True only the range gate closest to the center of the cell is extracted. Default False

latlon_tol [Float. Dataset keyword] Tolerance in lat/lon when extracting data only from the center of the TRT cell. Default 0.01

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

new_dataset [dictionary] Dictionary containing radar_out, a radar object containing only data from inside the TRT cell

ind_rad [int] radar index

`pyrad.proc.`**`process_traj_trt_contour`**(*procstatus*, *dscfg*, *radar_list=None*, *trajectory=None*)

Gets the TRT cell contour corresponding to each radar volume

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

time_tol [float. Dataset keyword] tolerance between reference time of the radar volume and that of the TRT cell [s]. Default 100.

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

new_dataset [dict] Dictionary containing radar_out and roi_dict. Radar out is the current radar object. roi_dict contains the positions defining the TRT cell contour

ind_rad [int] radar index

`pyrad.proc.`**`process_trajectory`**(*procstatus*, *dscfg*, *radar_list=None*, *trajectory=None*)

Return trajectory

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

new_dataset [Trajectory object] radar object

ind_rad [int] None

`pyrad.proc.`**`process_ts_along_coord`**(*procstatus*, *dscfg*, *radar_list=None*)

Produces time series along a particular antenna coordinate

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the time series

mode [str] coordinate to extract data along. Can be ALONG_AZI, ALONG_ELE or ALONG_RNG

fixed_range, fixed_azimuth, fixed_elevation [float] The fixed range [m], azimuth [deg] or elevation [deg] to extract. In each mode two of these parameters have to be defined. If they are not defined they default to 0.

ang_tol, rng_tol [float] The angle tolerance [deg] and range tolerance [m] around the fixed range or azimuth/elevation

value_start, value_stop [float] The minimum and maximum value at which the data along a coordinate start and stop

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.`**`process_turbulence`**(*procstatus*, *dscfg*, *radar_list=None*)

Computes turbulence from the Doppler spectrum width and reflectivity using the PyTDA package

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

radius [float. Dataset keyword] Search radius for calculating Eddy Dissipation Rate (EDR). Default 2

split_cut [Bool. Dataset keyword] Set to True for split-cut volumes. Default False

max_split_cut [Int. Dataset keyword] Total number of tilts that are affected by split cuts. Only relevant if split_cut=True. Default 2

xran, yran [float array. Dataset keyword] Spatial range in X,Y to consider. Default [-100, 100] for both X and Y

beamwidth [Float. Dataset keyword] Radar beamwidth. Default None. If None it will be obtained from the radar object metadata. If cannot be obtained defaults to 1 deg.

compute_gate_pos [Bool. Dataset keyword] If True the gate position is going to be computed in PyTDA. Otherwise the position from the radar object is used. Default False

verbose [Bool. Dataset keyword] True for verbose output. Default False

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_vad(procstatus, dscfg, radar_list=None)`

Estimates vertical wind profile using the VAD (velocity Azimuth Display) technique

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_visibility(procstatus, dscfg, radar_list=None)`

Gets the visibility in percentage from the minimum visible elevation. Anything with elevation lower than the minimum visible elevation plus and offset is set to 0 while above is set to 100.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

offset [float. Dataset keyword] The offset above the minimum visibility that must be filtered

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_vol_refl(procstatus, dscfg, radar_list=None)`

Computes the volumetric reflectivity in 10log10(cm^2 km^-3)

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

freq [float. Dataset keyword] The radar frequency

kw [float. Dataset keyword] The water constant

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_wbn_iq(procstatus, dscfg, radar_list=None)`

Computes the wide band noise from the horizontal or vertical IQ data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted configuration keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of spectra objects] Optional. list of spectra objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_weighted_time_avg**(*procstatus*, *dscfg*, *radar_list=None*)
computes the temporal mean of a field weighted by the reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [Radar] radar object

ind_rad [int] radar index

pyrad.proc.**process_wind_vel**(*procstatus*, *dscfg*, *radar_list=None*)
Estimates the horizontal or vertical component of the wind from the radial velocity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

vert_proj [Boolean] If true the vertical projection is computed. Otherwise the horizontal projection is computed

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_windshear**(*procstatus*, *dscfg*, *radar_list=None*)
Estimates the wind shear from the wind velocity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

az_tol [float] The tolerance in azimuth when looking for gates on top of the gate when computation is performed

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_zdr_column`**(*procstatus*, *dscfg*, *radar_list=None*)

Detects ZDR columns

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.`**`process_zdr_precip`**(*procstatus*, *dscfg*, *radar_list=None*)

Keeps only suitable data to evaluate the differential reflectivity in moderate rain or precipitation (for vertical scans)

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

ml_filter [boolean. Dataset keyword] indicates if a filter on data in and above the melting layer is applied. Default True.

rmin [float. Dataset keyword] minimum range where to look for rain [m]. Default 1000.

rmax [float. Dataset keyword] maximum range where to look for rain [m]. Default 50000.

Zmin [float. Dataset keyword] minimum reflectivity to consider the bin as precipitation [dBZ]. Default 20.

Zmax [float. Dataset keyword] maximum reflectivity to consider the bin as precipitation [dBZ] Default 22.

RhoHVmin [float. Dataset keyword] minimum RhoHV to consider the bin as precipitation Default 0.97

PhiDPmax [float. Dataset keyword] maximum PhiDP to consider the bin as precipitation [deg] Default 10.

elmax [float. Dataset keyword] maximum elevation angle where to look for precipitation [deg] Default None.

ml_thickness [float. Dataset keyword] assumed thickness of the melting layer. Default 700.

fzl [float. Dataset keyword] The default freezing level height. It will be used if no temperature field name is specified or the temperature field is not in the radar object. Default 2000.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

pyrad.proc.**process_zdr_snow**(*procstatus*, *dscfg*, *radar_list=None*)

Keeps only suitable data to evaluate the differential reflectivity in snow

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] minimum range where to look for rain [m]. Default 1000.

rmax [float. Dataset keyword] maximum range where to look for rain [m]. Default 50000.

Zmin [float. Dataset keyword] minimum reflectivity to consider the bin as snow [dBZ]. Default 0.

Zmax [float. Dataset keyword] maximum reflectivity to consider the bin as snow [dBZ] Default 30.

SNRmin [float. Dataset keyword] minimum SNR to consider the bin as snow [dB]. Default 10.

SNRmax [float. Dataset keyword] maximum SNR to consider the bin as snow [dB] Default 50.

RhoHVmin [float. Dataset keyword] minimum RhoHV to consider the bin as snow Default 0.97

PhiDPmax [float. Dataset keyword] maximum PhiDP to consider the bin as snow [deg] Default 10.

elmax [float. Dataset keyword] maximum elevation angle where to look for snow [deg] Default None.

KDPmax [float. Dataset keyword] maximum KDP to consider the bin as snow [deg] Default None

TEMPmin [float. Dataset keyword] minimum temperature to consider the bin as snow [deg C]. Default None

TEMPmax [float. Dataset keyword] maximum temperature to consider the bin as snow [deg C] Default None

hydroclass [list of ints. Dataset keyword] list of hydrometeor classes to keep for the analysis Default [2] (dry snow)

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

CHAPTER

THREE

PRODUCTS GENERATION (PYRAD.PROD)

Initiate the products generation.

3.1 Auxiliary functions

`get_dsformat_func`

3.2 Product generation

generate_occurrence_products(dataset, prdcfg)	generates occurrence products. Accepted product types:
generate_cosmo_coord_products(dataset, prdcfg)	generates COSMO coordinates products. Accepted product types:
generate_cosmo_to_radar_products(dataset, prdcfg)	generates COSMO data in radar coordinates products.
generate_sun_hits_products(dataset, prdcfg)	generates sun hits products. Accepted product types:
generate_intercomp_products(dataset, prdcfg)	Generates radar intercomparison products. Accepted product types:
generate_colocated_gates_products(dataset, ...)	Generates colocated gates products. Accepted product types:
generate_time_avg_products(dataset, prdcfg)	generates time average products. Accepted product types:
generate_qvp_products(dataset, prdcfg)	Generates quasi vertical profile-like products.
generate_vol_products(dataset, prdcfg)	Generates radar volume products. Accepted product types:
generate_timeseries_products(dataset, prdcfg)	Generates time series products. Accepted product types:
generate_monitoring_products(dataset, prdcfg)	generates a monitoring product.
generate_spectra_products(dataset, prdcfg)	generates spectra products. Accepted product types:
generate_grid_products(dataset, prdcfg)	generates grid products. Accepted product types:
generate_grid_time_avg_products(dataset, prdcfg)	generates time average products. Accepted product types:
generate_traj_product(traj, prdcfg)	Generates trajectory products. Accepted product types:
generate_ml_products(dataset, prdcfg)	Generates melting layer products. Accepted product types:

`pyrad.prod.`**`generate_colocated_gates_products`**(*dataset*, *prdcfg*)

Generates colocated gates products. Accepted product types:

'WRITE_COLOCATED_GATES': Writes the position of the co-located gates in a csv file

All the products of the 'VOL' dataset group

Parameters

dataset [tuple] radar objects and colocated gates dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.`**`generate_cosmo_coord_products`**(*dataset*, *prdcfg*)

generates COSMO coordinates products. Accepted product types:

'SAVEVOL': Save an object containing the index of the COSMO model grid that corresponds to each radar gate in a C/F radial file. User defined parameters:

file_type: str The type of file used to save the data. Can be 'nc' or 'h5'. Default 'nc'

physical: Bool If True the data will be saved in physical units (floats). Otherwise it will be quantized and saved as binary

compression: str For ODIM file formats, the type of compression. Can be any of the allowed compression types for hdf5 files. Default gzip

compression_opts: any The compression options allowed by the hdf5. Depends on the type of compression. Default 6 (The gzip compression level).

Parameters

dataset [tuple] radar object containing the COSMO coordinates

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.`**`generate_cosmo_to_radar_products`**(*dataset*, *prdcfg*)

generates COSMO data in radar coordinates products. Accepted product types:

'SAVEVOL': Save an object containing the COSMO data in radar

coordinatesin a C/F radial or ODIM file. User defined parameters: file_type: str

The type of file used to save the data. Can be 'nc' or 'h5'. Default 'nc'

physical: Bool If True the data will be saved in physical units (floats). Otherwise it will be quantized and saved as binary

compression: str For ODIM file formats, the type of compression. Can be any of the allowed compression types for hdf5 files. Default gzip

compression_opts: any The compression options allowed by the hdf5. Depends on the type of compression. Default 6 (The gzip compression level).

All the products of the 'VOL' dataset group

Parameters

dataset [tuple] radar object containing the COSMO coordinates

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

pyrad.prod.**generate_grid_products**(*dataset*, *prdcfg*)

generates grid products. Accepted product types:

'CROSS_SECTION': Plots a cross-section of gridded data

User defined parameters:

coord1, coord2: dict The two lat-lon coordinates marking the limits. They have the keywords 'lat' and 'lon' [degree]. The altitude limits are defined by the parameters in 'rhiImageConfig' in the 'loc' configuration file

'HISTOGRAM': Computes a histogram of the radar volum data

User defined parameters:

step: float or None the data quantization step. If none it will be obtained from the Py-ART configuration file

vmin, vmax: float or None The minimum and maximum values. If None they will be obtained from the Py-ART configuration file

mask_val: float or None A value to mask.

write_data: Bool If true the histogram data is written in a csv file

'LATITUDE_SLICE': Plots a cross-section of gridded data over a constant latitude. User defined parameters:

lon, lat: floats The starting point of the cross-section. The ending point is defined by the parameters in 'rhiImageConfig' in the 'loc' configuration file

'LONGITUDE_SLICE': Plots a cross-ection of gridded data over a constant longitude. User defined parameters:

lon, lat: floats The starting point of the cross-section. The ending point is defined by the parameters in 'rhiImageConfig' in the 'loc' configuration file

'SAVEALL': Saves a gridded data object including all or a list of user-defined fields in a netcdf file User defined parameters:

datatypes: list of str or None The list of data types to save. If it is None, all fields in the radar object will be saved

'SAVEVOL': Saves on field of a gridded data object in a netcdf file. 'STATS': Computes statistics over the whole images and stores them in

a file. User defined parameters:

stat: str The statistic used. Can be mean, median, min, max

'SURFACE_IMAGE': Plots a surface image of gridded data.

User defined parameters:

level: int The altitude level to plot. The rest of the parameters are defined by the parameters in 'ppiImageConfig' and 'ppiMapImageConfig' in the 'loc' configuration file

'SURFACE_CONTOUR': Plots a surface image of contour gridded data.

User defined parameters:

level: int The altitude level to plot. The rest of the parameters are defined by the parameters in 'ppiImageConfig' and 'ppiMapImageConfig' in the 'loc' configuration file

contour_values [float array or None] The contour values. If None the values are taken from the 'boundaries' keyword in the field description in the Py-ART config file. If 'boundaries' is not set the countours are 10 values linearly distributed from vmin to vmax

linewidths [float] width of the contour lines

colors [color string or sequence of colors] The contour colours

SURFACE_CONTOUR_OVERPLOT: Plots a surface image of gridded data with a contour overplotted. User defined parameters:

level: int The altitude level to plot. The rest of the parameters are defined by the parameters in 'ppiImageConfig' and 'ppiMapImageConfig' in the 'loc' configuration file

contour_values [float array or None] The contour values. If None the values are taken from the 'boundaries' keyword in the field description in the Py-ART config file. If 'boundaries' is not set the countours are 10 values linearly distributed from vmin to vmax

linewidths [float] width of the contour lines

colors [color string or sequence of colors] The contour colours

SURFACE_OVERPLOT: Plots on the same surface two images, one on top of the other. User defined parameters:

level: int The altitude level to plot. The rest of the parameters are defined by the parameters in 'ppiImageConfig' and 'ppiMapImageConfig' in the 'loc' configuration file

contour_values [float array or None] The contour values. If None the values are taken from the 'boundaries' keyword in the field description in the Py-ART config file. If 'boundaries' is not set the countours are 10 values linearly distributed from vmin to vmax

Parameters

dataset [grid] grid object

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

None or name of generated files

`pyrad.prod.`**`generate_grid_time_avg_products`**(*dataset*, *prdcfg*)

generates time average products. Accepted product types: All the products of the 'VOL' dataset group

Parameters

dataset [tuple] radar objects and colocated gates dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise pyrad.prod.**generate_intercomp_products**(*dataset*, *prdcfg*)

Generates radar intercomparison products. Accepted product types:

'PLOT_AND_WRITE_INTERCOMP_TS': Writes statistics of radar intercomparison in a file and plots the time series of the statistics. User defined parameters:

'add_date_in_fname': Bool If true adds the year in the csv file containing the statistics. Default False

'sort_by_date': Bool If true sorts the statistics by date when reading the csv file containing the statistics. Default False

'rewrite': Bool If true rewrites the csv file containing the statistics. Default False

'npoints_min': int The minimum number of points to consider the statistics valid and therefore use the data point in the plotting. Default 0

'corr_min': float The minimum correlation to consider the statistics valid and therefore use the data point in the plotting. Default 0.

'PLOT_SCATTER_INTERCOMP': Plots a density plot with the points of radar 1 versus the points of radar 2 User defined parameters:

'step': float The quantization step of the data. If none it will be computed using the Py-ART config file. Default None

'scatter_type': str Type of scatter plot. Can be a plot for each radar volume ('instant') or at the end of the processing period ('cumulative'). Default is 'cumulative'

'WRITE_INTERCOMP': Writes the instantaneously intercompared data (gate positions, values, etc.) in a csv file.

'WRITE_INTERCOMP_TIME_AVG': Writes the time-averaged intercompared data (gate positions, values, etc.) in a csv file.

Parameters

dataset [tuple] values of colocated gates dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

pyrad.prod.**generate_ml_products**(*dataset*, *prdcfg*)

Generates melting layer products. Accepted product types:

'ML_TS': Plots and writes a time series of the melting layer, i.e. the evolution of the average and standard deviation of the melting layer top and thickness and the the number of rays used in the retrieval. User defined parameters:

dpi: int The pixel density of the plot. Default 72

'SAVE_ML': Saves an object containing the melting layer retrieval information in a C/F radial file

All the products of the 'VOL' dataset group

Parameters

dataset [dict] dictionary containing the radar object and a keyword stating the status of the processing

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

pyrad.prod.**generate_monitoring_products**(*dataset*, *prdcfg*)

generates a monitoring product. With the parameter 'hist_type' the user may define if the product is computed for each radar volume ('instant') or at the end of the processing period ('cumulative'). Default is 'cumulative'. Accepted product types:

'ANGULAR_DENSITY': For a specified elevation angle, plots a 2D histogram with the azimuth angle in the X-axis and the data values in the Y-axis. The reference values and the user defined quantiles are also plot on the same figure User defined parameters:

anglenr: int The elevation angle number to plot

quantiles: list of floats The quantiles to plot. Default 25., 50., 75.

ref_value: float The reference value

vmin, vmax [floats or None] The minimum and maximum values of the data points. If not specified they are obtained from the Py-ART config file

'CUMUL_VOL_TS': Plots time series of the average of instantaneous quantiles stored in a csv file. User defined parameters:

quantiles: list of 3 floats the quantiles to compute. Default 25., 50., 75.

ref_value: float The reference value. Default 0

sort_by_date: Bool If true when reading the csv file containing the statistics the data is sorted by date. Default False

rewrite: Bool If true the csv file containing the statistics is rewritten

add_data_in_fname: Bool If true and the data used is cumulative the year is written in the csv file name and the plot file name

npoints_min: int Minimum number of points to use the data point in the plotting and to send an alarm. Default 0

vmin, vmax: float or None Limits of the Y-axis (data value). If None the limits are obtained from the Py-ART config file

alarm: Bool If true an alarm is sent

tol_abs: float Margin of tolerance from the reference value. If the current value is above this margin an alarm is sent. If the margin is not specified it is not possible to send any alarm

tol_trend: float Margin of tolerance from the reference value. If the trend of the last X events is above this margin an alarm is sent. If the margin is not specified it is not possible to send any alarm

nevents_min: int Minimum number of events with sufficient points to send an alarm related to the trend. If not specified it is not possible to send any alarm

sender: str The mail of the alarm sender. If not specified it is not possible to send any alarm

receiver_list: list of str The list of emails of the people that will receive the alarm.. If not specified it is not possible to send any alarm

'PPI_HISTOGRAM': Plots a histogram of data at a particular elevation angle. User defined parameters:

anglenr: int The elevation angle number to plot

'SAVEVOL': Saves the monitoring data in a C/F radar file. The data field contains histograms of data for each pair of azimuth and elevation angles

'VOL_HISTOGRAM': Plots a histogram of data collected from all the radar volume. User defined parameters:

write_data: bool If true the resultant histogram is also saved in a csv file. Default True.

'VOL_TS': Computes statistics of the gathered data and writes them in a csv file and plots a time series of those statistics. User defined parameters:

quantiles: list of 3 floats the quantiles to compute. Default 25., 50., 75.

ref_value: float The reference value. Default 0

sort_by_date: Bool If true when reading the csv file containing the statistics the data is sorted by date. Default False

rewrite: Bool If true the csv file containing the statistics is rewritten

add_data_in_fname: Bool If true and the data used is cumulative the year is written in the csv file name and the plot file name

npoints_min: int Minimum number of points to use the data point in the plotting and to send an alarm. Default 0

vmin, vmax: float or None Limits of the Y-axis (data value). If None the limits are obtained from the Py-ART config file

alarm: Bool If true an alarm is sent

tol_abs: float Margin of tolerance from the reference value. If the current value is above this margin an alarm is sent. If the margin is not specified it is not possible to send any alarm

tol_trend: float Margin of tolerance from the reference value. If the trend of the last X events is above this margin an alarm is sent. If the margin is not specified it is not possible to send any alarm

nevents_min: int Minimum number of events with sufficient points to send an alarm related to the trend. If not specified it is not possible to send any alarm

sender: str The mail of the alarm sender. If not specified it is not possible to send any alarm

receiver_list: list of str The list of emails of the people that will receive the alarm.. If not specified it is not possible to send any alarm

Parameters

dataset [dictionary] dictionary containing a histogram object and some metadata

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.`**`generate_occurrence_products`**(*dataset*, *prdcfg*)

generates occurrence products. Accepted product types:

'WRITE_EXCESS_GATES': Write the data that identifies radar gates with clutter that has a frequency of occurrence above a certain threshold. User defined parameters:

quant_min: float Minimum frequency of occurrence in percentage to keep the gate as valid. Default 95.

All the products of the 'VOL' dataset group

Parameters

dataset [tuple] radar object and metadata dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.`**`generate_qvp_products`**(*dataset*, *prdcfg*)

Generates quasi vertical profile-like products. Quasi vertical profiles come from azimuthal averaging of polarimetric radar data. With the variable 'qvp_type' the user decides if the product has to be generated at the end of the processing period ('final') or instantaneously ('instant') Accepted product types:

All the products of the 'VOL' dataset group

Parameters

dataset [dict] dictionary containing the radar object and a keyword stating the status of the processing

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.`**`generate_spectra_products`**(*dataset*, *prdcfg*)

generates spectra products. Accepted product types:

'AMPLITUDE_PHASE_ANGLE_DOPPLER': Makes an angle Doppler plot of complex spectra or IQ data. The plot can be along azimuth or along range. It is plotted separately the module and the phase of the signal. User defined parameters:

along_azi [bool] If true the plot is performed along azimuth, otherwise along elevation. Default true

ang [float] The fixed angle (deg). Default 0.

rng [float] The fixed range (m). Default 0.

ang_tol [float] The fixed angle tolerance (deg). Default 1.

rng_tol [float] The fixed range tolerance (m). Default 50.

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

ampli_vmin, ampli_vmax, phase_vmin, phase_vmax [float or None] Minimum and maximum of the color scale for the module and phase

'AMPLITUDE_PHASE_DOPPLER': Plots a complex Doppler spectrum or IQ data making two separate plots for the module and phase of the signal User defined parameters:

azi, ele, rng [float] azimuth and elevation (deg) and range (m) of the ray to plot

azi_to, ele_tol, rng_tol [float] azimuth and elevation (deg) and range (m) tolerance respect to nominal position to plot. Default 1, 1, 50.

ind_ray, ind_rng [int] index of the ray and range to plot. Alternative to defining its antenna coordinates

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

ampli_vmin, ampli_vmax, phase_vmin, phase_vmax [float or None] Minimum and maximum of the color scale for the module and phase

'AMPLITUDE_PHASE_RANGE_DOPPLER': Plots a complex spectra or IQ data range-Doppler making two separate plots for the module and phase of the signal User defined parameters:

azi, ele [float] azimuth and elevation (deg) of the ray to plot

azi_to, ele_tol [float] azimuth and elevation (deg) tolerance respect to nominal position to plot. Default 1, 1.

ind_ray [int] index of the ray to plot. Alternative to defining its antenna coordinates

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

ampli_vmin, ampli_vmax, phase_vmin, phase_vmax [float or None] Minimum and maximum of the color scale for the module and phase

'AMPLITUDE_PHASE_TIME_DOPPLER': Plots a complex spectra or IQ data time-Doppler making two separate plots for the module and phase of the signal User defined parameters:

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity' or 'Doppler frequency'

ampli_vmin, ampli_vmax, phase_vmin, phase_vmax [float or None] Minimum and maximum of the color scale for the module and phase

plot_type [str] Can be 'final' or 'temporal'. If final the data is only plotted at the end of the processing

'ANGLE_DOPPLER': Makes an angle Doppler plot. The plot can be along azimuth or along range User defined parameters:

along_azi [bool] If true the plot is performed along azimuth, otherwise along elevation. Default true

ang [float] The fixed angle (deg). Default 0.

rng [float] The fixed range (m). Default 0.

ang_tol [float] The fixed angle tolerance (deg). Default 1.

rng_tol [float] The fixed range tolerance (m). Default 50.

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

vmin, vmax [float or None] Minimum and maximum of the color scale

'COMPLEX_ANGLE_DOPPLER': Makes an angle Doppler plot of complex spectra or IQ data. The plot can be along azimuth or along range. The real and imaginary parts are plotted separately User defined parameters:

along_azi [bool] If true the plot is performed along azimuth, otherwise along elevation. Default true

ang [float] The fixed angle (deg). Default 0.

rng [float] The fixed range (m). Default 0.

ang_tol [float] The fixed angle tolerance (deg). Default 1.

rng_tol [float] The fixed range tolerance (m). Default 50.

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

vmin, vmax [float or None] Minimum and maximum of the color scale

'COMPLEX_DOPPLER': Plots a complex Doppler spectrum or IQ data making two separate plots for the real and imaginary parts User defined parameters:

azi, ele, rng [float] azimuth and elevation (deg) and range (m) of the ray to plot

azi_to, ele_tol, rng_tol [float] azimuth and elevation (deg) and range (m) tolerance respect to nominal position to plot. Default 1, 1, 50.

ind_ray, ind_rng [int] index of the ray and range to plot. Alternative to defining its antenna coordinates

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

vmin, vmax [float or None] Minimum and maximum of the color scale

'COMPLEX_RANGE_DOPPLER': Plots the complex spectra or IQ data range-Doppler making two separate plots for the real and imaginary parts User defined parameters:

azi, ele [float] azimuth and elevation (deg) of the ray to plot

azi_to, ele_tol [float] azimuth and elevation (deg) tolerance respect to nominal position to plot. Default 1, 1.

ind_ray [int] index of the ray to plot. Alternative to defining its antenna coordinates

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

vmin, vmax [float or None] Minimum and maximum of the color scale

'COMPLEX_TIME_DOPPLER': Plots the complex spectra or IQ data time-Doppler making two separate plots for the real and imaginary parts User defined parameters:

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity' or 'Doppler frequency'

vmin, vmax [float or None] Minimum and maximum of the color scale

plot_type [str] Can be 'final' or 'temporal'. If final the data is only plotted at the end of the processing

'DOPPLER': Plots a Doppler spectrum variable or IQ data variable

User defined parameters:

azi, ele, rng [float] azimuth and elevation (deg) and range (m) of the ray to plot

azi_to, ele_tol, rng_tol [float] azimuth and elevation (deg) and range (m) tolerance respect to nominal position to plot. Default 1, 1, 50.

ind_ray, ind_rng [int] index of the ray and range to plot. Alternative to defining its antenna coordinates

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

vmin, vmax [float or None] Minimum and maximum of the color scale

'RANGE_DOPPLER': Makes a range-Doppler plot of spectral or IQ data

User defined parameters:

azi, ele [float] azimuth and elevation (deg) of the ray to plot

azi_to, ele_tol [float] azimuth and elevation (deg) tolerance respect to nominal position to plot. Default 1, 1.

ind_ray [int] index of the ray to plot. Alternative to defining its antenna coordinates

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

vmin, vmax [float or None] Minimum and maximum of the color scale

'SAVEALL': Saves radar spectra or IQ volume data including all or a list of userdefined fields in a netcdf file User defined parameters:

datatypes: list of str or None The list of data types to save. If it is None, all fields in the radar object will be saved

physical: Bool If True the data will be saved in physical units (floats). Otherwise it will be quantized and saved as binary

'SAVEVOL': Saves one field of a radar spectra or IQ volume data in a netcdf file User defined parameters:

physical: Bool If True the data will be saved in physical units (floats). Otherwise it will be quantized and saved as binary

'TIME_DOPPLER': Makes a time-Doppler plot of spectral or IQ data at a point of interest. User defined parameters:

xaxis_info [str] The xaxis type. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

vmin, vmax [float or None] Minimum and maximum of the color scale

plot_type [str] Can be 'final' or 'temporal'. If final the data is only plotted at the end of the processing

Parameters

dataset [spectra] spectra object

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

None or name of generated files

pyrad.prod.**generate_sun_hits_products**(*dataset*, *prdcfg*)

generates sun hits products. Accepted product types:

'PLOT_SUN_HITS': Plots in a sun-radar azimuth difference-sun-radar elevation difference grid the values of all sun hits obtained during the processing period

'PLOT_SUN_RETRIEVAL': Plots in a sun-radar azimuth difference-sun- radar elevation difference grid the retrieved sun pattern

'PLOT_SUN_RETRIEVAL_TS': Plots time series of the retrieved sun pattern parameters User defined parameters:

dpi: int The pixel density of the plot. Default 72

add_date_in_fname: Bool If true the year is added in the plot file name

'WRITE_SUN_HITS': Writes the information concerning possible sun hits in a csv file

'WRITE_SUN_RETRIEVAL': Writes the retrieved sun pattern parameters in a csv file. User defined parameters:

add_date_in_fname: Bool If true the year is added in the csv file name

All the products of the 'VOL' dataset group

Parameters

dataset [tuple] radar object and sun hits dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

pyrad.prod.**generate_time_avg_products**(*dataset*, *prdcfg*)

generates time average products. Accepted product types: All the products of the 'VOL' dataset group

Parameters

dataset [tuple] radar objects and colocated gates dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

pyrad.prod.**generate_timeseries_products**(*dataset*, *prdcfg*)

Generates time series products. Accepted product types:

'COMPARE_CUMULATIVE_POINT': Plots in the same graph 2 time series of data accumulation (tipically rainfall rate). One time series is a point measurement of radar data while the other is from a co-located instrument (rain gauge or disdrometer) User defined parameters:

dpi: int The pixel density of the plot. Default 72

vmin, vmax: float The limits of the Y-axis. If none they will be obtained from the Py-ART config file.

sensor: str The sensor type. Can be 'rgage' or 'disdro'

sensorid: str The sensor ID.

location: str A string identifying the location of the disdrometer

freq: float The frequency used to retrieve the polarimetric variables of a disdrometer

ele: float The elevation angle used to retrieve the polarimetric variables of a disdrometer

ScanPeriod: float The scaning period of the radar in seconds. This parameter is defined in the 'loc' config file

'COMPARE_POINT': Plots in the same graph 2 time series of data . One time series is a point measurement of radar data while the other is from a co-located instrument (rain gauge or disdrometer) User defined parameters:

dpi: int The pixel density of the plot. Default 72

vmin, vmax: float The limits of the Y-axis. If none they will be obtained from the Py-ART config file.

sensor: str The sensor type. Can be 'rgage' or 'disdro'

sensorid: str The sensor ID.

location: str A string identifying the location of the disdrometer

freq: float The frequency used to retrieve the polarimetric variables of a disdrometer

ele: float The elevation angle used to retrieve the polarimetric variables of a disdrometer

'COMPARE_TIME_AVG': Creates a scatter plot of average radar data versus average sensor data. User defined parameters:

dpi: int The pixel density of the plot. Default 72

sensor: str The sensor type. Can be 'rgage' or 'disdro'

sensorid: str The sensor ID.

location: str A string identifying the location of the disdrometer

freq: float The frequency used to retrieve the polarimetric variables of a disdrometer

ele: float The elevation angle used to retrieve the polarimetric variables of a disdrometer

cum_time: float Data accumulation time [s]. Default 3600.

base_time: float Starting moment of the accumulation [s from midnight]. Default 0.

'PLOT_AND_WRITE': Writes and plots a trajectory time series.

User defined parameters:

ymin, ymax: float The minimum and maximum value of the Y-axis. If none it will be obtained from the Py-ART config file.

'PLOT_AND_WRITE_POINT': Plots and writes a time series of radar data at a particular point User defined parameters:

dpi: int The pixel density of the plot. Default 72

vmin, vmax: float The limits of the Y-axis. If none they will be obtained from the Py-ART config file.

'PLOT_CUMULATIVE_POINT': Plots a time series of radar data accumulation at a particular point. User defined parameters:

dpi: int The pixel density of the plot. Default 72

vmin, vmax: float The limits of the Y-axis. If none they will be obtained from the Py-ART config file.

ScanPeriod: float The scaning period of the radar in seconds. This parameter is defined in the 'loc' config file

'PLOT_HIST': plots and writes a histogram of all the data gathered during the trajectory processing User defined parameters:

step: float or None The quantization step of the data. If None it will be obtained from the Py-ART config file

'TRAJ_CAPPI_IMAGE': Creates a CAPPI image with the trajectory position overplot on it. User defined parameters:

color_ref: str The meaning of the color code with which the trajectory is plotted. Can be 'None', 'altitude' (the absolute altitude), 'rel_altitude' (altitude relative to the CAPPI altitude), 'time' (trajectory time respect of the start of the radar scan leading to the CAPPI)

altitude: float The CAPPI altitude [m]

wfunc: str Function used in the gridding of the radar data. The function types are defined in pyart.map.grid_from_radars. Default 'NEAREST'

res: float The CAPPI resolution [m]. Default 500.

Parameters

dataset [dictionary] radar object

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

no return

`pyrad.prod.generate_traj_product(traj, prdcfg)`

Generates trajectory products. Accepted product types:

'TRAJ_MAP': Plots the trajectory on a lat-lon map with the altitude color coded

'TRAJ_PLOT': Plots time series of the trajectory respect to the radar elevation, azimuth or range User defined parameters:

'datatype': str The type of parameter: 'EL', 'AZ', or 'RANGE'

'TRAJ_TEXT': Writes the trajectory information in a csv file

Parameters

traj [Trajectory object]

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

None

`pyrad.prod.generate_vol_products(dataset, prdcfg)`

Generates radar volume products. Accepted product types:

'CDF': plots and writes the cumulative density function of data

User defined parameters:

quantiles: list of floats The quantiles to compute in percent. Default None

sector: dict dictionary defining the sector where to compute the CDF. Default is None and the CDF is computed over all the data May contain:

rmin, rmax: float min and max range [m]

azmin, azmax: float min and max azimuth angle [deg]

elmin, elmax: float min and max elevation angle [deg]

hmin, hmax: float min and max altitude [m MSL]

vismin: float The minimum visibility to use the data. Default None

absolute: Bool If true the absolute values of the data will be used. Default False

use_nans: Bool If true NaN values will be used. Default False

nan_value: Bool The value by which the NaNs are substituted if NaN values are to be used in the computation

filterclt: Bool If True the gates containing clutter are filtered

filterprec: list of ints The hydrometeor types that are filtered from the analysis. Default empty list.

'BSCOPE_IMAGE': Creates a B-scope image (azimuth, range)

User defined parameters:

anglenr [int] The elevation angle number to use

ray_dim [str] the ray dimension. Can be 'ang' or 'time'. Default 'ang'

xaxis_rng [bool] if True the range will be in the x-axis. Otherwise it will be in the y-axis. Default True

vmin, vmax: float or None The minimum and maximum values of the color scale. If None the scale is going to be set according to the Py-ART config file

'CAPPI_IMAGE': Creates a CAPPI image

User defined parameters:

altitude: flt CAPPI altitude [m MSL]

wfunc: str The function used to produce the CAPPI as defined in pyart.map.grid_from_radars. Default 'NEAREST'

cappi_res: float The CAPPI resolution [m]. Default 500.

'FIELD_COVERAGE': Gets the field coverage over a certain sector

User defined parameters:

threshold: float or None Minimum value to consider the data valid. Default None

nvalid_min: float Minimum number of valid gates in the ray to consider it valid. Default 5

ele_res, azi_res: float Elevation and azimuth resolution of the sectors [deg]. Default 1. and 2.

ele_min, ele_max: float Min and max elevation angle defining the sector [deg]. Default 0. and 30.

ele_step: float Elevation step [deg]. Default 5.

ele_sect_start, ele_sect_stop: float or None start and stop angles of the sector coverage. Default None

quantiles: list of floats The quantiles to compute in the sector. Default 10. to 90. by steps of 10.

AngTol: float The tolerance in elevation angle when putting the data in a fixed grid

'FIXED_RNG_IMAGE': Plots a fixed range image

User defined parameters:

AngTol [float] The tolerance between the nominal angles and the actual radar angles. Default 1.

ele_res, azi_res: float or None The resolution of the fixed grid [deg]. If None it will be obtained from the separation between angles

vmin, vmax [float or None] Min and Max values of the color scale. If None the values are taken from the Py-ART config file

'FIXED_RNG_SPAN_IMAGE': Plots a user-defined statistic over a fixed range image User defined parameters:

AngTol [float] The tolerance between the nominal angles and the actual radar angles. Default 1.

ele_res, azi_res: float or None The resolution of the fixed grid [deg]. If None it will be obtained from the separation between angles

stat [str] The statistic to compute. Can be 'min', 'max', 'mean', 'mode'. Default 'max'

'HISTOGRAM': Computes a histogram of the radar volum data

User defined parameters:

step: float or None the data quantization step. If none it will be obtained from the Py-ART configuration file

write_data: Bool If true the histogram data is written in a csv file

'PLOT_ALONG_COORD': Plots the radar volume data along a particular coordinate User defined parameters:

colors: list of str or None The colors of each ploted line

mode: str Ploting mode. Can be 'ALONG_RNG', 'ALONG_AZI' or 'ALONG_ELE'

value_start, value_stop: float The starting and ending points of the data to plot. According to the mode it may refer to the range, azimuth or elevation. If not specified the minimum and maximum possible values are used

fix_elevations, fix_azimuths, fix_ranges: list of floats The elevations, azimuths or ranges to plot for each mode. 'ALONG_RNG' would use fix_elevations and fix_azimuths 'ALONG_AZI' fix_ranges and fix_elevations 'ALONG_ELE' fix_ranges and fix_azimuths

AngTol: float The tolerance to match the radar angle to the fixed angles Default 1.

RngTol: float The tolerance to match the radar range to the fixed ranges Default 50.

'PPI_CONTOUR': Plots a PPI countour plot

User defined parameters:

contour_values: list of floats or None The list of contour values to plot. If None the contour values are going to be obtained from the Py-ART config file either with the dictionary key 'contour_values' or from the minimum and maximum values of the field with an assumed division of 10 levels.

anglenr: float The elevation angle number

'PPI_CONTOUR_OVERPLOT': Plots a PPI of a field with another field overplotted as a contour plot. User defined parameters:

contour_values: list of floats or None The list of contour values to plot. If None the contour values are going to be obtained from the Py-ART config file either with the dictionary key 'contour_values' or from the minimum and maximum values of the field with an assumed division of 10 levels.

anglenr: float The elevation angle number

'PPI_IMAGE': Plots a PPI image. It can also plot the histogram and the quantiles of the data in the PPI. User defined parameters:

anglenr: float The elevation angle number

plot_type: str The type of plot to perform. Can be 'PPI', 'QUANTILES' or 'HISTOGRAM'

step: float or None If the plot type is 'HISTOGRAM', the width of the histogram bin. If None it will be obtained from the Py-ART config file

quantiles: list of float or None If the plot type is 'QUANTILES', the list of quantiles to compute. If None a default list of quantiles will be computed

vmin, vmax: float or None The minimum and maximum values of the color scale. If None the scale is going to be set according to the Py-ART config file

'PPI_MAP': Plots a PPI image over a map. The map resolution and the type of maps used are defined in the variables 'mapres' and 'maps' in 'ppiMapImageConfig' in the loc config file. User defined parameters:

anglenr: float The elevation angle number

'PPIMAP_ROI_OVERPLOT': Over plots a polygon delimiting a region of interest on a PPI map. The map resolution and the type of maps used are defined in the variables 'mapres' and 'maps' in 'ppiMapImageConfig' in the loc config file. User defined parameters:

anglenr: float The elevation angle number

'PROFILE_STATS': Computes and plots a vertical profile statistics. The statistics are saved in a csv file User defined parameters:

heightResolution: float The height resolution of the profile [m]. Default 100.

heightMin, heightMax: float or None The minimum and maximum altitude of the profile [m MSL]. If None the values will be obtained from the minimum and maximum gate altitude.

quantity: str The type of statistics to plot. Can be 'quantiles', 'mode', 'reqgression_mean' or 'mean'.

quantiles: list of floats If quantity type is 'quantiles' the list of quantiles to compute. Default 25., 50., 75.

nvalid_min: int The minimum number of valid points to consider the statistic valid. Default 4

make_linear: Bool If true the data is converted from log to linear before computing the stats

include_nans: Bool If true NaN values are included in the statistics

fixed_span: Bool If true the profile plot has a fix X-axis

vmin, vmax: float or None If fixed_span is set, the minimum and maximum values of the X-axis. If None, they are obtained from the Py-ART config file

'PSEUDOPPI_CONTOUR': Plots a pseudo-PPI countour plot

User defined parameters:

contour_values: list of floats or None The list of contour values to plot. If None the contour values are going to be obtained from the Py-ART config file either with the dictionary key 'contour_values' or from the minimum and maximum values of the field with an assumed division of 10 levels.

angle: float The elevation angle at which compute the PPI

EleTol: float The tolerance between the actual radar elevation angle and the nominal pseudo-PPI elevation angle.

'PSEUDOPPI_CONTOUR_OVERPLOT': Plots a pseudo-PPI of a field with another field overplotted as a contour plot User defined parameters:

contour_values: list of floats or None The list of contour values to plot. If None the contour values are going to be obtained from the Py-ART config file either with the dictionary key 'contour_values' or from the minimum and maximum values of the field with an assumed division of 10 levels.

angle: float The elevation angle at which compute the PPI

EleTol: float The tolerance between the actual radar elevation angle and the nominal pseudo-PPI elevation angle.

'PSEUDOPPI_IMAGE': Plots a pseudo-PPI image. It can also plot the histogram and the quantiles of the data in the pseudo-PPI. User defined parameters:

angle: float The elevation angle of the pseudo-PPI

EleTol: float The tolerance between the actual radar elevation angle and the nominal pseudo-PPI elevation angle.

plot_type: str The type of plot to perform. Can be 'PPI', 'QUANTILES' or 'HISTOGRAM'

step: float or None If the plot type is 'HISTOGRAM', the width of the histogram bin. If None it will be obtained from the Py-ART config file

quantiles: list of float or None If the plot type is 'QUANTILES', the list of quantiles to compute. If None a default list of quantiles will be computed

vmin, vmax [float or None] Min and Max values of the color scale. If None the values are taken from the Py-ART config file

'PSEUDOPPI_MAP': Plots a pseudo-PPI image over a map. The map resolution and the type of maps used are defined in the variables 'mapres' and 'maps' in 'ppiMapImageConfig' in the loc config file. User defined parameters:

angle: float The elevation angle of the pseudo-PPI

EleTol: float The tolerance between the actual radar elevation angle and the nominal pseudo-PPI elevation angle.

'PSEUDORHI_CONTOUR': Plots a pseudo-RHI countour plot

User defined parameters:

contour_values: list of floats or None The list of contour values to plot. If None the contour values are going to be obtained from the Py-ART config file either with the dictionary key 'contour_values' or from the minimum and maximum values of the field with an assumed division of 10 levels.

angle: float The azimuth angle at which to compute the RPI

AziTol: float The tolerance between the actual radar azimuth angle and the nominal pseudo-RHI azimuth angle.

'PSEUDORHI_CONTOUR_OVERPLOT': Plots a pseudo-RHI of a field with another field over-plotted as a contour plot User defined parameters:

contour_values: list of floats or None The list of contour values to plot. If None the contour values are going to be obtained from the Py-ART config file either with the dictionary key 'contour_values' or from the minimum and maximum values of the field with an assumed division of 10 levels.

angle: float The azimuth angle at which to compute the RPI

AziTol: float The tolerance between the actual radar azimuth angle and the nominal pseudo-RHI azimuth angle.

'PSEUDORHI_IMAGE': Plots a pseudo-RHI image. It can also plot the histogram and the quantiles of the data in the pseudo-RHI. User defined parameters:

angle: float The azimuth angle at which to compute the RPI

AziTol: float The tolerance between the actual radar azimuth angle and the nominal pseudo-RHI azimuth angle.

plot_type: str The type of plot to perform. Can be 'RHI', 'QUANTILES' or 'HISTOGRAM'

step: float or None If the plot type is 'HISTOGRAM', the width of the histogram bin. If None it will be obtained from the Py-ART config file

quantiles: list of float or None If the plot type is 'QUANTILES', the list of quantiles to compute. If None a default list of quantiles will be computed

vmin, vmax [float or None] Min and Max values of the color scale. If None the values are taken from the Py-ART config file

'QUANTILES': Plots and writes the quantiles of a radar volume

User defined parameters:

quantiles: list of floats or None the list of quantiles to compute. If None a default list of quantiles will be computed.

write_data: Bool If True the computed data will be also written in a csv file

fixed_span: Bool If true the quantile plot has a fix Y-axis

vmin, vmax: float or None If fixed_span is set, the minimum and maximum values of the Y-axis. If None, they are obtained from the Py-ART config file

'RHI_CONTOUR': Plots an RHI countour plot

User defined parameters:

contour_values: list of floats or None The list of contour values to plot. If None the contour values are going to be obtained from the Py-ART config file either with the dictionary key 'contour_values' or from the minimum and maximum values of the field with an assumed division of 10 levels.

anglenr: int The azimuth angle number

'RHI_CONTOUR_OVERPLOT': Plots an RHI of a field with another field over-plotted as a contour plot User defined parameters:

contour_values: list of floats or None The list of contour values to plot. If None the contour values are going to be obtained from the Py-ART config file either with the dictionary key 'contour_values' or from the minimum and maximum values of the field with an assumed division of 10 levels.

anglenr: int The azimuth angle number

'RHI_IMAGE': Plots an RHI image. It can also plot the histogram and the quantiles of the data in the RHI. User defined parameters:

anglenr: int The azimuth angle number

plot_type: str The type of plot to perform. Can be 'RHI', 'QUANTILES' or 'HISTOGRAM'

step: float or None If the plot type is 'HISTOGRAM', the width of the histogram bin. If None it will be obtained from the Py-ART config file

quantiles: list of float or None If the plot type is 'QUANTILES', the list of quantiles to compute. If None a default list of quantiles will be computed

vmin, vmax: float or None The minimum and maximum values of the color scale. If None the scale is going to be set according to the Py-ART config file

'RHI_PROFILE': Computes and plots a vertical profile statistics out of an RHI. The statistics are saved in a csv file User defined parameters:

rangeStart, rangeStop: float The range start and stop of the data to extract from the RHI to compute the statistics [m]. Default 0., 25000.

heightResolution: float The height resolution of the profile [m]. Default 100.

heightMin, heightMax: float or None The minimum and maximum altitude of the profile [m MSL]. If None the values will be obtained from the minimum and maximum gate altitude.

quantity: str The type of statistics to plot. Can be 'quantiles', 'mode', 'reqgression_mean' or 'mean'.

quantiles: list of floats If quantity type is 'quantiles' the list of quantiles to compute. Default 25., 50., 75.

nvalid_min: int The minimum number of valid points to consider the statistic valid. Default 4

make_linear: Bool If true the data is converted from log to linear before computing the stats

include_nans: Bool If true NaN values are included in the statistics

fixed_span: Bool If true the profile plot has a fix X-axis

vmin, vmax: float or None If fixed_span is set, the minimum and maximum values of the X-axis. If None, they are obtained from the Py-ART config file

'SAVEALL': Saves radar volume data including all or a list of user- defined fields in a C/F radial or ODIM file User defined parameters:

file_type: str The type of file used to save the data. Can be 'nc' or 'h5'. Default 'nc'

datatypes: list of str or None The list of data types to save. If it is None, all fields in the radar object will be saved

physical: Bool If True the data will be saved in physical units (floats). Otherwise it will be quantized and saved as binary

compression: str For ODIM file formats, the type of compression. Can be any of the allowed compression types for hdf5 files. Default gzip

compression_opts: any The compression options allowed by the hdf5. Depends on the type of compression. Default 6 (The gzip compression level).

'SAVESTATE': Saves the last processed data in a file. Used for real- time data processing

'SAVEVOL': Saves one field of a radar volume data in a C/F radial or ODIM file User defined parameters:

file_type: str The type of file used to save the data. Can be 'nc' or 'h5'. Default 'nc'

physical: Bool If True the data will be saved in physical units (floats). Otherwise it will be quantized and saved as binary

compression: str For ODIM file formats, the type of compression. Can be any of the allowed compression types for hdf5 files. Default gzip

compression_opts: any The compression options allowed by the hdf5. Depends on the type of compression. Default 6 (The gzip compression level).

'SAVE_FIXED_ANGLE': Saves the position of the first fix angle in a csv file

'SELFCONSISTENCY': Plots a ZDR versus KDP/ZH histogram of data.

User defined parameters:

retrieve_relation [bool] If True plots also the retrieved relationship. Default True

plot_theoretical [bool] If True plots also the theoretical relationship. Default True

normalize [bool] If True the occurrence density of ZK/KDP for each ZDR bin is going to be represented. Otherwise it will show the number of gates at each bin. Default True

'SELFCONSISTENCY2': Plots a ZH measured versus ZH inferred from a self-consistency relation histogram of data. User defined parameters:

normalize [bool] If True the occurrence density of ZK/KDP for each ZDR bin is going to be represented. Otherwise it will show the number of gates at each bin. Default True

'TIME_RANGE': Plots a time-range/azimuth/elevation plot

User defined parameters:

anglenr: float The number of the fixed angle to plot

vmin, vmax: float or None The minimum and maximum values of the color scale. If None the scale is going to be set according to the Py-ART config file

'VOL_TS': Writes and plots a value corresponding to a time series. Meant primarily for writing and plotting the results of the SELFCONSISTENCY2 algorithm User defined parameters:

ref_value: float The reference value. Default 0

sort_by_date: Bool If true when reading the csv file containing the statistics the data is sorted by date. Default False

rewrite: Bool If true the csv file containing the statistics is rewritten

add_data_in_fname: Bool If true and the data used is cumulative the year is written in the csv file name and the plot file name

npoints_min: int Minimum number of points to use the data point in the plotting and to send an alarm. Default 0

vmin, vmax: float or None Limits of the Y-axis (data value). If None the limits are obtained from the Py-ART config file

alarm: Bool If true an alarm is sent

tol_abs: float Margin of tolerance from the reference value. If the current value is above this margin an alarm is sent. If the margin is not specified it is not possible to send any alarm

tol_trend: float Margin of tolerance from the reference value. If the trend of the last X events is above this margin an alarm is sent. If the margin is not specified it is not possible to send any alarm

nevents_min: int Minimum number of events with sufficient points to send an alarm related to the trend. If not specified it is not possible to send any alarm

sender: str The mail of the alarm sender. If not specified it is not possible to send any alarm

receiver_list: list of str The list of emails of the people that will receive the alarm.. If not specified it is not possible to send any alarm

'WIND_PROFILE': Plots vertical profile of wind data (U, V, W components and wind velocity and direction) out of a radar volume containing the retrieved U,V and W components of the wind, the standard deviation of the retrieval and the velocity difference between the estimated radial velocity (assuming the wind to be uniform) and the actual measured radial velocity. User defined parameters:

heightResolution: float The height resolution of the profile [m]. Default 100.

heightMin, heightMax: float or None The minimum and maximum altitude of the profile [m MSL]. If None the values will be obtained from the minimum and maximum gate altitude.

min_ele: float The minimum elevation to be used in the computation of the vertical velocities. Default 5.

max_ele: float The maximum elevation to be used in the computation of the horizontal velocities. Default 85.

fixed_span: Bool If true the profile plot has a fix X-axis

vmin, vmax: float or None If fixed_span is set, the minimum and maximum values of the X-axis. If None, they are obtained from the span of the U component defined in the Py-ART config file

Parameters

dataset [dict] dictionary with key radar_out containing a radar object

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

The list of created fields or None

`pyrad.prod.`**`get_prodgen_func`**(*dsformat*, *dsname*, *dstype*)

maps the dataset format into its processing function

Parameters

dsformat [str] dataset group. The following is a list of dataset groups with the function that is called to generate their products. For details about what the functions do check the function documentation:

'VOL': generate_vol_products 'COLOCATED_GATES': generate_colocated_gates_products 'COSMO_COORD': generate_cosmo_coord_products 'COSMO2RADAR': generate_cosmo_to_radar_products 'GRID': generate_grid_products 'SPECTRA': generate_spectra_products 'GRID_TIMEAVG': generate_grid_time_avg_products 'INTERCOMP': generate_intercomp_products 'ML': generate_ml_products 'MONITORING': generate_monitoring_products 'OCCURRENCE': generate_occurrence_products 'QVP': generate_qvp_products 'SPARSE_GRID': generate_sparse_grid_products 'SUN_HITS': generate_sun_hits_products 'TIMEAVG': generate_time_avg_products 'TIMESERIES': generate_timeseries_products 'TRAJ_ONLY': generate_traj_product

Returns

func [function] pyrad function used to generate the products

CHAPTER

FOUR

INPUT AND OUTPUT (PYRAD.IO)

Functions to read and write data and configuration files.

4.1 Reading configuration files

read_config(fname[, cfg])	Read a pyrad config file.

4.2 Reading radar data

get_data(voltime, datatypesdescr, cfg)	Reads pyrad input data.

4.3 Reading cosmo data

cosmo2radar_data(radar, cosmo_coord, cosmo_data)	get the COSMO value corresponding to each radar gate using nearest neighbour interpolation
cosmo2radar_coord(radar, cosmo_coord[, ...])	Given the radar coordinates find the nearest COSMO model pixel
hzt2radar_data(radar, hzt_coord, hzt_data[, ...])	get the HZT value corresponding to each radar gate using nearest neighbour interpolation
hzt2radar_coord(radar, hzt_coord[, ...])	Given the radar coordinates find the nearest HZT pixel
get_cosmo_fields(cosmo_data, cosmo_ind[, ...])	Get the COSMO data corresponding to each radar gate using a precomputed look up table of the nearest neighbour
get_iso0_field(hzt_data, hzt_ind, z_radar[, ...])	Get the height over iso0 data corresponding to each radar gate using a precomputed look up table of the nearest neighbour
read_cosmo_data(fname[, field_names, celsius])	Reads COSMO data from a netcdf file
read_cosmo_coord(fname[, zmin])	Reads COSMO coordinates from a netcdf file
read_hzt_data(fname[, chy0, chx0, read_lib])	Reads iso-0 degree data from an HZT file

4.4 Reading DEM data

dem2radar_data	
dem2radar_coord	
read_idrisi_data(fname, field_name[, fill_value])	Reads DEM data from an IDRISI .rst file
read_idrisi_metadata(fname)	Reads DEM metadata from a IDRISI .rdc file

4.5 Reading other data

read_proc_periods(fname)	Reads a file containing the start and stop times of periods to process
read_last_state(fname)	Reads a file containing the date of acquisition of the last volume processed
read_status(voltime, cfg[, ind_rad])	Reads rad4alp xml status file.
read_rad4alp_cosmo(fname, datatype[, ngates])	Reads rad4alp COSMO data binary file.
read_rad4alp_vis(fname, datatype)	Reads rad4alp visibility data binary file.
read_excess_gates(fname)	Reads a csv files containing the position of gates exceeding a given percentile of frequency of occurrence
read_colocated_gates(fname)	Reads a csv files containing the position of colocated gates
read_colocated_data(fname)	Reads a csv files containing colocated data
read_timeseries(fname)	Reads a time series contained in a csv file
read_ts_cum(fname)	Reads a time series of precipitation accumulation contained in a csv file
read_monitoring_ts(fname[, sort_by_date])	Reads a monitoring time series contained in a csv file
read_intercomp_scores_ts(fname[, sort_by_date])	Reads a radar intercomparison scores csv file
get_sensor_data(date, datatype, cfg)	Gets data from a point measurement sensor (rain gauge or disdrometer)
read_smn(fname)	Reads SwissMetNet data contained in a csv file
read_smn2(fname)	Reads SwissMetNet data contained in a csv file with format station,time,value
read_disdro_scattering(fname)	Reads scattering parameters computed from disdrometer data contained in a text file
read_sun_hits(fname)	Reads sun hits data contained in a csv file
read_sun_hits_multiple_days(cfg, time_ref[, ...])	Reads sun hits data from multiple file sources
read_sun_retrieval(fname)	Reads sun retrieval data contained in a csv file
read_solar_flux(fname)	Reads solar flux data from the DRAO observatory in Canada
read_selfconsistency(fname)	Reads a self-consistency table with Zdr, Kdp/Zh columns
read_antenna_pattern(fname[, linear, twoway])	Read antenna pattern from file
read_meteorage(fname)	Reads METEORAGE lightning data contained in a text file.
read_lightning(fname[, filter_data])	Reads lightning data contained in a text file.
read_lightning_traj(fname)	Reads lightning trajectory data contained in a csv file.
read_lightning_all(fname[, labels])	Reads a file containing lightning data and co-located polarimetric data.
read_trt_scores(fname)	Reads the TRT scores contained in a text file.

Continued on next page

Table 5 – continued from previous page

read_trt_data(fname)	Reads the TRT data contained in a text file.
read_trt_traj_data(fname)	Reads the TRT cell data contained in a text file.
read_trt_thundertracking_traj_data(fname)	Reads the TRT cell data contained in a text file.
read_trt_cell_lightning(fname)	Reads the lightning data of a TRT cell.
read_trt_info_all(info_path)	Reads all the TRT info files
read_trt_info_all2(info_path)	Reads all the TRT info files
read_trt_info(fname)	Reads the TRT info used for thundertracking and contained in a text file.
read_trt_info2(fname)	Reads the TRT info used for thundertracking and contained in a text file.
read_thundertracking_info(fname)	Reads the TRT info used for thundertracking
read_rhi_profile(fname[, labels])	Reads a monitoring time series contained in a csv file
read_histogram(fname)	Reads a histogram contained in a csv file
read_quantiles(fname)	Reads quantiles contained in a csv file
read_profile_ts(fname_list, labels[, hres, . . .])	Reads a colection of profile data file and creates a time series
read_histogram_ts(fname_list, datatype[, t_res])	Reads a colection of histogram data file and creates a time series
read_quantiles_ts(fname_list[, step, qmin, . . .])	Reads a colection of quantiles data file and creates a time series
read_ml_ts(fname)	Reads a melting layer time series contained in a csv file
read_windmills_data(fname)	Read the wind mills data csv file

4.6 Writing data

write_proc_periods(start_times, end_times, fname)	writes an output file containing start and stop times of periods to process
write_ts_lightning(flashnr, time_data, . . .)	writes the LMA sources data and the value of the colocated polarimetric variables
send_msg(sender, receiver_list, subject, fname)	sends the content of a text file by email
write_alarm_msg(radar_name, param_name_unit, . . .)	writes an alarm file
write_last_state(datetime_last, fname)	writes SwissMetNet data in format datetime,avg_value, std_value
write_smn(datetime_vec, value_avg_vec, . . .)	writes SwissMetNet data in format datetime,avg_value, std_value
write_trt_info(ids, max_rank, nscans, . . .)	writes TRT info of the thundertracking
write_trt_thundertracking_data(traj_ID, . . .)	writes TRT cell data of the thundertracking scan
write_trt_cell_data(traj_ID, yyyymmddHHMM, . . .)	writes TRT cell data
write_trt_cell_scores(traj_ID, . . .)	writes TRT cells scores
write_trt_cell_lightning(cell_ID, cell_time, . . .)	writes the lightning data for each TRT cell
write_trt_rpc(cell_ID, cell_time, lon, lat, . . .)	writes the rimed particles column data for a TRT cell
write_rhi_profile(hvec, data, nvalid_vec, . . .)	writes the values of an RHI profile in a text file
write_field_coverage(quantiles, values, . . .)	writes the quantiles of the coverage on a particular sector
write_cdf(quantiles, values, ntot, nnan, . . .)	writes a cumulative distribution function

Continued on next page

Table 6 – continued from previous page

write_histogram(bin_edges, values, fname[, ...])	writes a histogram
write_quantiles(quantiles, values, fname[, ...])	writes quantiles
write_ts_polar_data(dataset, fname)	writes time series of data
write_ts_grid_data(dataset, fname)	writes time series of data
write_ts_cum(dataset, fname)	writes time series accumulation of data
write_ts_stats(dt, value, fname[, stat])	writes time series of statistics
write_monitoring_ts(start_time, np_t, ...[, ...])	writes time series of data
write_excess_gates(excess_dict, fname)	Writes the position and values of gates that have a frequency of occurrence higher than a particular threshold
write_intercomp_scores_ts(start_time, stats, ...)	writes time series of radar intercomparison scores
write_colocated_gates(coloc_gates, fname)	Writes the position of gates colocated with two radars
write_colocated_data(coloc_data, fname)	Writes the data of gates colocated with two radars
write_colocated_data_time_avg(coloc_data, fname)	Writes the time averaged data of gates colocated with two radars
write_sun_hits(sun_hits, fname)	Writes sun hits data.
write_sun_retrieval(sun_retrieval, fname)	Writes sun retrieval data.
write_fixed_angle(time_data, fixed_angle, ...)	writes an output file with the fixed angle data

4.7 Auxiliary functions

get_rad4alp_prod_fname(datatype)	Given a datatype find the corresponding start and termination of the METRANET product file name
map_hydro(hydro_data_op)	maps the operational hydrometeor classification identifiers to the ones used by Py-ART
map_Doppler(Doppler_data_bin, Nyquist_vel)	maps the binary METRANET Doppler data to actual Doppler velocity
get_save_dir(basepath, procname, dsname, prdname)	obtains the path to a product directory and eventually creates it
make_filename(prdtype, dstype, dsname, ext_list)	creates a product file name
generate_field_name_str(datatype)	Generates a field name in a nice to read format.
get_fieldname_pyart(datatype)	maps the config file radar data type name into the corresponding rainbow Py-ART field name
get_fieldname_cosmo(field_name)	maps the Py-ART field name into the corresponding COSMO variable name
get_field_unit(datatype)	Return unit of datatype.
get_file_list(datadescriptor, starttimes, ...)	gets the list of files with a time period
get_rad4alp_dir(basepath, voltime[, ...])	gets the directory where rad4alp data is stored
get_rad4alp_grid_dir(basepath, voltime, ...)	gets the directory where rad4alp grid data is stored
get_trtfile_list(basepath, starttime, endtime)	gets the list of TRT files with a time period
get_new_rainbow_file_name(master_fname, ...)	get the rainbow file name containing datatype from a master file name and data type
get_datatype_fields(datadescriptor)	splits the data type descriptor and provides each individual member
get_dataset_fields(datasetdescr)	splits the dataset type descriptor and provides each individual member
get_datetime(fname, datadescriptor)	Given a data descriptor gets date and time from file name

Continued on next page

Table 7 – continued from previous page

`find_raw_cosmo_file`(voltime, datatype, cfg)	Search a COSMO file in netcdf format
`find_hzt_file`(voltime, cfg[, ind_rad])	Search an ISO-0 degree file in HZT format
`_get_datetime`(fname, datagroup[, ftime_format])	Given a data group gets date and time from file name

4.8 Trajectory

`Trajectory`(filename[, starttime, endtime, . . .])	A class for reading and handling trajectory data from a file.

4.9 TimeSeries

`TimeSeries`(desc[, timevec, timeformat, . . .])	Holding timeseries data and metadata.

class `pyrad.io.`**`TimeSeries`**(*desc*, *timevec=None*, *timeformat=None*, *maxlength=None*, *datatype=''*)

Bases: `object`

Holding timeseries data and metadata.

Attributes

description [array of str] Description of the data of the time series.

time_vector [array of datetime objects]

timeformat [how to print the time (default:] 'Date, UTC [seconds since midnight]'

dataseries [List of _dataSeries object holding the] data

Methods

`add_dataseries`(label, unit_name, unit[, . . .])	Add a new data series to the timeseries object.
`add_timesample`(dt, values)	Add a new sample to the time series.
`plot`(fname[, ymin, ymax])	Make a figure of a time series
`plot_hist`(fname[, step])	Make histograms of time series
`write`(fname)	Write time series output

`__class__`
alias of `builtins.type`

`__delattr__`(*name*, */*)
Implement delattr(self, name).

`__dict__ = mappingproxy({'__module__': 'pyrad.io.timeseries', '__doc__': "\n Holding`

`__dir__`(*/*)
Default dir() implementation.

`__eq__`(*value*, */*)
Return self==value.

`__format__`(*format_spec*, */*)
Default object formatter.

`__ge__`(*value*, /)
Return self>=value.

`__getattribute__`(*name*, /)
Return getattr(self, name).

`__gt__`(*value*, /)
Return self>value.

`__hash__`(/)
Return hash(self).

`__init__`(*desc*, *timevec=None*, *timeformat=None*, *maxlength=None*, *datatype=''*)
Initalize the object.

Parameters

desc [array of str]

timevec [array of datetime]

timeformat [specifies time format]

maxlength [Maximal length of the time series]

num_el [Number of values in the time series]

`__init_subclass__`()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

`__le__`(*value*, /)
Return self<=value.

`__lt__`(*value*, /)
Return self<value.

`__module__ = 'pyrad.io.timeseries'`

`__ne__`(*value*, /)
Return self!=value.

`__new__`(**args*, ***kwargs*)
Create and return a new object. See help(type) for accurate signature.

`__reduce__`(/)
Helper for pickle.

`__reduce_ex__`(*protocol*, /)
Helper for pickle.

`__repr__`(/)
Return repr(self).

`__setattr__`(*name*, *value*, /)
Implement setattr(self, name, value).

`__sizeof__`(/)
Size of object in memory, in bytes.

`__str__`(/)
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

add_dataseries(*label*, *unit_name*, *unit*, *dataseries=None*, *plot=True*, *color=None*, *linestyle=None*)
Add a new data series to the timeseries object. The length of the data vector must be the same as the length of the time vector.

add_timesample(*dt*, *values*)
Add a new sample to the time series.

plot(*fname*, *ymin=None*, *ymax=None*)
Make a figure of a time series

plot_hist(*fname*, *step=None*)
Make histograms of time series

write(*fname*)
Write time series output

class pyrad.io.**Trajectory**(*filename*, *starttime=None*, *endtime=None*, *trajtype='plane'*, *flashnr=0*)
Bases: object

A class for reading and handling trajectory data from a file.

Attributes

filename [str] Path and name of the trajectory definition file

starttime [datetime] Start time of trajectory processing.

endtime [datetime] End time of trajectory processing.

trajtype [str]

Type of trajectory. Can be 'plane' or 'lightning'

time_vector [Array of datetime objects] Array containing the trajectory time samples

wgs84_lat_deg [Array of floats] WGS84 latitude samples in radian

wgs84_lon_deg [Array of floats] WGS84 longitude samples in radian

wgs84_alt_m [Array of floats] WGS84 altitude samples in m

nsamples [int]

Number of samples in the trajectory

_swiss_grid_done [Bool] Indicates that convertion to Swiss coordinates has been performed

swiss_chy, swiss_chx, swiss_chh [Array of floats] Swiss coordinates in m

radar_list [list] List of radars for which trajectories are going to be computed

flashnr [int] For 'lightning' only. Number of flash for which trajectory data is going to be computed. If 0 all all flashes are going to be considered.

time_in_flash [array of floats] For 'lightning' only. Time within flash (sec)

flashnr_vec [array of ints] For 'lightning' only. Flash number of each data sample

dBm [array of floats] For 'lightning' only. Lightning power (dBm)

Methods

add_radar(radar)	Add the coordinates (WGS84 longitude, latitude and non WGS84 altitude) of a radar to the radar_list.
calculate_velocities(radar)	Calculate velocities.
get_end_time()	Get time of last trajectory sample.
get_samples_in_period([start, end])	”
get_start_time()	Get time of first trajectory sample.

__class__
alias of `builtins.type`

__delattr__(*name*, /)
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyrad.io.trajectory', '__doc__': "\n A class

__dir__(/)
Default dir() implementation.

__eq__(*value*, /)
Return self==value.

__format__(*format_spec*, /)
Default object formatter.

__ge__(*value*, /)
Return self>=value.

__getattribute__(*name*, /)
Return getattr(self, name).

__gt__(*value*, /)
Return self>value.

__hash__(/)
Return hash(self).

__init__(*filename*, *starttime=None*, *endtime=None*, *trajtype='plane'*, *flashnr=0*)
Initalize the object.

Parameters

filename [str] Filename containing the trajectory samples.

starttime [datetime] Start time of trajectory processing. If not given, use the time of the first trajectory sample.

endtime [datetime] End time of trajectory processing. If not given, use the time of the last trajectory sample.

trajtype [str] type of trajectory. Can be plane or lightning

flashnr [int] If type of trajectory is lightning, the flash number to check the trajectory. 0 means all flash numbers included

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__(*value*, /)
Return self<=value.

__lt__(*value*, /)
Return self<value.

__module__ = 'pyrad.io.trajectory'

__ne__(*value*, /)
Return self!=value.

__new__(**args*, ***kwargs*)
Create and return a new object. See help(type) for accurate signature.

__reduce__(/)
Helper for pickle.

__reduce_ex__(*protocol*, /)
Helper for pickle.

__repr__(/)
Return repr(self).

__setattr__(*name*, *value*, /)
Implement setattr(self, name, value).

__sizeof__(/)
Size of object in memory, in bytes.

__str__(/)
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_convert_traj_to_swissgrid()
Convert trajectory samples from WGS84 to Swiss CH1903 coordinates

_get_total_seconds(*x*)
Return total seconds of timedelta object

_read_traj()
Read trajectory from file

_read_traj_lightning(*flashnr=0*)
Read trajectory from lightning file

Parameters

flashnr [int] the flash number to keep. If 0 data from all flashes will be kept

_read_traj_trt()
Read trajectory from TRT file

add_radar(*radar*)
Add the coordinates (WGS84 longitude, latitude and non WGS84 altitude) of a radar to the radar_list.

Parameters

radar [pyart radar object] containing the radar coordinates

calculate_velocities(*radar*)
Calculate velocities.

get_end_time()
Get time of last trajectory sample.

get_samples_in_period(*start=None*, *end=None*)
" Get indices of samples of the trajectory within given time period.

get_start_time()
Get time of first trajectory sample.

pyrad.io.**add_field**(*radar_dest*, *radar_orig*)
adds the fields from orig radar into dest radar. If they are not in the same grid, interpolates them to dest grid

Parameters

radar_dest [radar object] the destination radar

radar_orig [radar object] the radar object containing the original field

Returns

field_dest [dict] interpolated field and metadata

pyrad.io.**cosmo2radar_coord**(*radar*, *cosmo_coord*, *slice_xy=True*, *slice_z=False*, *field_name=None*)
Given the radar coordinates find the nearest COSMO model pixel

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

cosmo_coord [dict] dictionary containing the COSMO coordinates

slice_xy [boolean] if true the horizontal plane of the COSMO field is cut to the dimensions of the radar field

slice_z [boolean] if true the vertical plane of the COSMO field is cut to the dimensions of the radar field

field_name [str] name of the field

Returns

cosmo_ind_field [dict] dictionary containing a field of COSMO indices and metadata

pyrad.io.**cosmo2radar_data**(*radar, cosmo_coord, cosmo_data, time_index=0, slice_xy=True, slice_z=False, field_names=['temperature'], dtype=<class 'numpy.float32'>*)
get the COSMO value corresponding to each radar gate using nearest neighbour interpolation

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

cosmo_coord [dict] dictionary containing the COSMO coordinates

cosmo_data [dict] dictionary containing the COSMO data

time_index [int] index of the forecasted data

slice_xy [boolean] if true the horizontal plane of the COSMO field is cut to the dimensions of the radar field

slice_z [boolean] if true the vertical plane of the COSMO field is cut to the dimensions of the radar field

field_names [str] names of COSMO fields to convert (default temperature)

dtype [numpy data type object] the data type of the output data

Returns

cosmo_fields [list of dict] list of dictionary with the COSMO fields and metadata

`pyrad.io.`**`find_hzt_file`**(*voltime*, *cfg*, *ind_rad=0*)

Search an ISO-0 degree file in HZT format

Parameters

voltime [datetime object] volume scan time

cfg [dictionary of dictionaries] configuration info to figure out where the data is

ind_rad [int] radar index

Returns

fname [str] Name of HZT file if it exists. None otherwise

`pyrad.io.`**`find_raw_cosmo_file`**(*voltime*, *datatype*, *cfg*, *ind_rad=0*)

Search a COSMO file in netcdf format

Parameters

voltime [datetime object] volume scan time

datatype [str] type of COSMO data to look for

cfg [dictionary of dictionaries] configuration info to figure out where the data is

ind_rad [int] radar index

Returns

fname [str] Name of COSMO file if it exists. None otherwise

`pyrad.io.`**`generate_field_name_str`**(*datatype*)

Generates a field name in a nice to read format.

Parameters

datatype [str] The data type

Returns

field_str [str] The field name

`pyrad.io.`**`get_cosmo_fields`**(*cosmo_data, cosmo_ind, time_index=0, field_names=['temperature']*)

Get the COSMO data corresponding to each radar gate using a precomputed look up table of the nearest neighbour

Parameters

cosmo_data [dict] dictionary containing the COSMO data and metadata

cosmo_ind [dict] dictionary containing a field of COSMO indices and metadata

time_index [int] index of the forecasted data

field_names [str] names of COSMO parameters (default temperature)

Returns

cosmo_fields [list of dict] dictionary with the COSMO fields and metadata

`pyrad.io.get_data(voltime, datatypesdescr, cfg)`

Reads pyrad input data.

Parameters

voltime [datetime object] volume scan time

datatypesdescr [list] list of radar field types to read. Format : [radarnr]:[datagroup]:[datatype],[dataset],[product] 'dataset' is only specified for data groups 'ODIM', 'CFRADIAL', 'CFRADIAL2', 'CF1', 'ODIMPYRAD' 'PYRADGRID' and 'NETCDFSPECTRA'. 'product' is only specified for data groups 'CFRADIAL', 'ODIMPYRAD', 'PYRADGRID' and 'NETCDFSPECTRA' The data group specifies the type file from which data is extracted. It can be:

'RAINBOW': Propietary Leonardo format 'COSMO': COSMO model data saved in Rainbow file format 'DEM': Visibility data saved in Rainbow file format 'PSR': Reads PSR data file to extract range gate information

(Noise and transmitted power)

'RAD4ALP': METRANET format used for the operational MeteoSwiss data. To find out which datatype to use to match a particular METRANET field name check the function 'get_datatype_metranet' in pyrad/io/io_aux.py

'RAD4ALPCOSMO': COSMO model data saved in a binary file format. Used by operational MeteoSwiss radars

'RAD4ALPDEM': Visibility data saved in a binary format used by operational MeteoSwiss radars

'RAD4ALPHYDRO': Used to read the MeteoSwiss operational hydrometeor classification

'RAD4ALPDOPPLER': Used to read the MeteoSwiss operational dealiased Doppler velocity

'ODIM': Generic ODIM file format. For such types 'dataset' specifies the directory and file name date convention. Example: ODIM:dBZ,D{%Y-%m-%d}-F{%Y%m%d%H%M%S}. To find out which datatype to use to match a particular ODIM field name check the function 'get_datatype_odim' in pyrad/io/io_aux.py

'NEXRADII': Nexrad-level II file format.

'CFRADIAL2': CFRADIAL2 file format. For such types 'dataset' specifies the directory and file name date convention. Example: ODIM:dBZ,D{%Y-%m-%d}-F{%Y%m%d%H%M%S}. To find out which datatype to use to match a particular ODIM field name check the function 'get_datatype_odim' in pyrad/io/io_aux.py

'CF1': CF1 file format. For such types 'dataset' specifies the directory and file name date convention. Example: ODIM:dBZ,D{%Y-%m-%d}-F{%Y%m%d%H%M%S}. To find out which datatype to use to match a particular ODIM field name check the function 'get_datatype_odim' in pyrad/io/io_aux.py

'MXPOL': MXPOL (EPFL) data written in a netcdf file

'CFRADIAL': CFRadial format with the naming convention and directory structure in which Pyrad saves the data. For such datatypes 'dataset' specifies the directory where the dataset is stored and 'product' specifies the directroy where the product is stored. Example: CFRADIAL:dBZc,Att_ZPhi,SAVEVOL_dBZc

'CFRADIALCOSMO': COSMO data in radar coordinates in a CFRadial file format.

'ODIMPYRAD': ODIM file format with the naming convention and directory structure in which Pyrad saves the data. For such datatypes 'dataset' specifies the directory where the dataset is stored and 'product' specifies the directroy where the product is stored. Example: ODIMPYRAD:dBZc,Att_ZPhi,SAVEVOL_dBZc

'RAD4ALPGRID': METRANET format used for the operational MeteoSwiss Cartesian products.

'RAD4ALPGIF': Format used for operational MeteoSwiss Cartesian products stored as gif files

'PYRADGRID': Pyrad generated Cartesian grid products. For such datatypes 'dataset' specifies the directory where the dataset is stored and 'product' specifies the directroy where the product is stored. Example: ODIMPYRAD:RR,RZC,SAVEVOL

'SATGRID': CF Netcdf from used for the MeteoSat satellite data in the CCS4 (Radar composite) grid.

'PSRSPECTRA': Format used to store Rainbow power spectra recordings.

'NETCDFSPECTRA': Format analogous to CFRadial and used to store Doppler spectral

'RAD4ALPIQ': Format used to store rad4alp IQ data

'RAINBOW', 'RAD4ALP', 'ODIM' 'CFRADIAL2', 'CF1' and 'MXPOL' are primary data file sources and they cannot be mixed for the same radar. It is also the case for their complementary data files, i.e. 'COSMO' and 'RAD4ALPCOSMO', etc. 'CFRADIAL' and 'ODIMPYRAD' are secondary data file sources and they can be combined with any other datagroup type. For a list of accepted datatypes and how they map to the Py-ART name convention check function 'get_field_name_pyart' in pyrad/io/io_aux.py

cfg: dictionary of dictionaries configuration info to figure out where the data is

Returns

radar [Radar] radar object

`pyrad.io.`**`get_dataset_fields`**(*datasetdescr*)
splits the dataset type descriptor and provides each individual member

Parameters

datasetdescr [str] dataset type. Format : [processing level]:[dataset type]

Returns

proclevel [str] dataset processing level

dataset [str] dataset type, i.e. dBZ, ZDR, ISO0, …

`pyrad.io.`**`get_datatype_fields`**(*datadescriptor*)
splits the data type descriptor and provides each individual member

Parameters

datadescriptor [str] radar field type. Format : [radar file type]:[datatype]

Returns

radarnr [str] radar number, i.e. RADAR1, RADAR2, ...

datagroup [str] data type group, i.e. RAINBOW, RAD4ALP, ODIM, CFRADIAL, COSMO, MXPOL ...

datatype [str] data type, i.e. dBZ, ZDR, ISO0, ...

dataset [str] dataset type (for saved data only)

product [str] product type (for saved data only)

`pyrad.io.`**`get_datetime`**(*fname*, *datadescriptor*)

Given a data descriptor gets date and time from file name

Parameters

fname [str] file name

datadescriptor [str] radar field type. Format : [radar file type]:[datatype]

Returns

fdatetime [datetime object] date and time in file name

`pyrad.io.`**`get_field_unit`**(*datatype*)

Return unit of datatype.

Parameters

datatype [str] The data type

Returns

unit [str] The unit

`pyrad.io.`**`get_fieldname_cosmo`**(*field_name*)

maps the Py-ART field name into the corresponding COSMO variable name

Parameters

field_name [str] Py-ART field name

Returns

cosmo_name [str] Py-ART variable name

`pyrad.io.`**`get_fieldname_pyart`**(*datatype*)

maps the config file radar data type name into the corresponding rainbow Py-ART field name

Parameters

datatype [str] config file radar data type name

Returns

field_name [str] Py-ART field name

`pyrad.io.`**`get_file_list`**(*datadescriptor*, *starttimes*, *endtimes*, *cfg*, *scan=None*)

gets the list of files with a time period

Parameters

datadescriptor [str] radar field type. Format : [radar file type]:[datatype]

startimes [array of datetime objects] start of time periods

endtimes [array of datetime object] end of time periods

cfg: dictionary of dictionaries configuration info to figure out where the data is

scan [str] scan name

Returns

filelist [list of strings] list of files within the time period

`pyrad.io.`**`get_iso0_field`**(*hzt_data*, *hzt_ind*, *z_radar*, *field_name='height_over_iso0'*)

Get the height over iso0 data corresponding to each radar gate using a precomputed look up table of the nearest neighbour

Parameters

hzt_data [dict] dictionary containing the HZT data and metadata

hzt_ind [dict] dictionary containing a field of HZT indices and metadata

z_radar [ndarray] gates altitude [m MSL]

field_name [str] names of HZT parameters (default height_over_iso0)

Returns

iso0_field [list of dict] dictionary with the height over iso0 field and metadata

`pyrad.io.`**`get_new_rainbow_file_name`**(*master_fname*, *master_datadescriptor*, *datatype*)

get the rainbow file name containing datatype from a master file name and data type

Parameters

master_fname [str] the master file name

master_datadescriptor [str] the master data type descriptor

datatype [str] the data type of the new file name to be created

Returns

new_fname [str] the new file name

`pyrad.io.`**`get_rad4alp_dir`**(*basepath*, *voltime*, *radar_name='A'*, *radar_res='L'*, *scan='001'*, *path_convention='MCH'*)

gets the directory where rad4alp data is stored

Parameters

basepath [str] base path

voltime [datetime object] nominal time

radar_name [str] radar name (A, D, L, P, W)

radar_res [str] radar resolution (H, L)

scan [str] scan

path_convention [str] The path convention. Can be 'LTE', 'MCH' or 'RT'

Returns

datapath [str] The data path

basename [str] The base name. ex: PHA17213

pyrad.io.**get_rad4alp_grid_dir**(*basepath*, *voltime*, *datatype*, *acronym*, *path_convention='MCH'*)
gets the directory where rad4alp grid data is stored

Parameters

basepath [str] base path

voltime [datetime object] nominal time

datatype [str] data type

acronym [str] acronym identifying the data type

path_convention [str] The path convention. Can be 'LTE', 'MCH' or 'RT'

Returns

datapath [str] The data path

pyrad.io.**get_rad4alp_prod_fname**(*datatype*)
Given a datatype find the corresponding start and termination of the METRANET product file name

Parameters

datatype [str] the data type

Returns

acronym [str] The start of the METRANET file name

termination [str] The end of the METRANET file name

pyrad.io.**get_save_dir**(*basepath*, *procname*, *dsname*, *prdname*, *timeinfo=None*, *timeformat='%Y-%m-%d'*, *create_dir=True*)
obtains the path to a product directory and eventually creates it

Parameters

basepath [str] product base path

procname [str] name of processing space

dsname [str] data set name

prdname [str] product name

timeinfo [datetime] time info to generate the date directory. If None there is no time format in the path

timeformat [str] Optional. The time format.

create_dir [boolean] If True creates the directory

Returns

savedir [str] path to product

pyrad.io.**get_sensor_data**(*date*, *datatype*, *cfg*)
Gets data from a point measurement sensor (rain gauge or disdrometer)

Parameters

date [datetime object] measurement date

datatype [str] name of the data type to read

cfg [dictionary] dictionary containing sensor information

Returns

sensordate , sensorvalue, label, period [tupple] date, value, type of sensor and measurement period

pyrad.io.**get_trtfile_list**(*basepath*, *starttime*, *endtime*)
gets the list of TRT files with a time period

Parameters

datapath [str] directory where to look for data

starttime [datetime object] start of time period

endtime [datetime object] end of time period

Returns

filelist [list of strings] list of files within the time period

pyrad.io.**hzt2radar_coord**(*radar*, *hzt_coord*, *slice_xy=True*, *field_name=None*)
Given the radar coordinates find the nearest HZT pixel

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

hzt_coord [dict] dictionary containing the HZT coordinates

slice_xy [boolean] if true the horizontal plane of the HZT field is cut to the dimensions of the radar field

field_name [str] name of the field

Returns

hzt_ind_field [dict] dictionary containing a field of HZT indices and metadata

pyrad.io.**hzt2radar_data**(*radar*, *hzt_coord*, *hzt_data*, *slice_xy=True*, *field_name='height_over_iso0'*)
get the HZT value corresponding to each radar gate using nearest neighbour interpolation

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

hzt_coord [dict] dictionary containing the HZT coordinates

hzt_data [dict] dictionary containing the HZT data

slice_xy [boolean] if true the horizontal plane of the COSMO field is cut to the dimensions of the radar field

field_name [str] name of HZT fields to convert (default height_over_iso0)

Returns

hzt_fields [list of dict] list of dictionary with the HZT fields and metadata

pyrad.io.**interpol_field**(*radar_dest*, *radar_orig*, *field_name*, *fill_value=None*, *ang_tol=0.5*)
interpolates field field_name contained in radar_orig to the grid in radar_dest

Parameters

radar_dest [radar object] the destination radar

radar_orig [radar object] the radar object containing the original field

field_name: str name of the field to interpolate

fill_value: float The fill value

ang_tol [float] angle tolerance to determine whether the radar origin sweep is the radar destination sweep

Returns

field_dest [dict] interpolated field and metadata

`pyrad.io.`**`make_filename`**(*prdtype*, *dstype*, *dsname*, *ext_list*, *prdcfginfo=None*, *timeinfo=None*, *timeformat='%Y%m%d%H%M%S'*, *runinfo=None*)

creates a product file name

Parameters

timeinfo [datetime] time info to generate the date directory

prdtype [str] product type, i.e. 'ppi', etc.

dstype [str] data set type, i.e. 'raw', etc.

dsname [str] data set name

ext_list [list of str] file name extensions, i.e. 'png'

prdcfginfo [str] Optional. string to add product configuration information, i.e. 'el0.4'

timeformat [str] Optional. The time format

runinfo [str] Optional. Additional information about the test (e.g. 'RUN01', 'TS011')

Returns

fname_list [list of str] list of file names (as many as extensions)

`pyrad.io.`**`map_Doppler`**(*Doppler_data_bin*, *Nyquist_vel*)

maps the binary METRANET Doppler data to actual Doppler velocity

Parameters

Doppler_data_bin [numpy array] The binary METRANET data

Returns

Doppler_data [numpy array] The Doppler veloctiy in [m/s]

`pyrad.io.`**`map_hydro`**(*hydro_data_op*)

maps the operational hydrometeor classification identifiers to the ones used by Py-ART

Parameters

hydro_data_op [numpy array] The operational hydrometeor classification data

Returns

hydro_data_py [numpy array] The pyart hydrometeor classification data

`pyrad.io.`**`read_antenna_pattern`**(*fname*, *linear=False*, *twoway=False*)

Read antenna pattern from file

Parameters

fname [str] path of the antenna pattern file

linear [boolean] if true the antenna pattern is given in linear units

twoway [boolean] if true the attenuation is two-way

Returns

pattern [dict] dictionary with the fields angle and attenuation

`pyrad.io.read_colocated_data(fname)`
Reads a csv files containing colocated data

Parameters

fname [str] path of time series file

Returns

rad1_time, rad1_ray_ind, rad1_rng_ind, rad1_ele, rad1_azi, rad1_rng,

rad1_val, rad2_time, rad2_ray_ind, rad2_rng_ind, rad2_ele, rad2_azi,

rad2_rng, rad2_val [tupple] A tupple with the data read. None otherwise

`pyrad.io.read_colocated_gates(fname)`
Reads a csv files containing the position of colocated gates

Parameters

fname [str] path of time series file

Returns

rad1_ray_ind, rad1_rng_ind, rad1_ele, rad1_azi, rad1_rng,

rad2_ray_ind, rad2_rng_ind, rad2_ele, rad2_azi, rad2_rng [tupple] A tupple with the data read. None otherwise

`pyrad.io.read_config(fname, cfg=None)`
Read a pyrad config file.

Parameters

fname [str] Name of the configuration file to read.

cfg [dict of dicts, optional] dictionary of dictionaries containing configuration parameters where the new parameters will be placed

Returns

cfg [dict of dicts] dictionary of dictionaries containing the configuration parameters

`pyrad.io.read_cosmo_coord(fname, zmin=None)`
Reads COSMO coordinates from a netcdf file

Parameters

fname [str] name of the file to read

Returns

cosmo_coord [dictionary] dictionary with the data and metadata

`pyrad.io.read_cosmo_data(fname, field_names=['temperature'], celsius=True)`
Reads COSMO data from a netcdf file

Parameters

fname [str] name of the file to read

field_names [str] name of the variable to read

celsius [Boolean] if True and variable temperature converts data from Kelvin to Centigrade

Returns

cosmo_data [dictionary] dictionary with the data and metadata `pyrad.io.`**`read_disdro_scattering`**(*fname*)

Reads scattering parameters computed from disdrometer data contained in a text file

Parameters

fname [str] path of time series file

Returns

date, preciptype, lwc, rr, zh, zv, zdr, ldr, ah, av, adiff, kdp, deltaco,

rhohv [tupple] The read values

`pyrad.io.`**`read_excess_gates`**(*fname*)

Reads a csv files containing the position of gates exceeding a given percentile of frequency of occurrence

Parameters

fname [str] path of time series file

Returns

rad1_ray_ind, rad1_rng_ind, rad1_ele, rad1_azi, rad1_rng,

rad2_ray_ind, rad2_rng_ind, rad2_ele, rad2_azi, rad2_rng [tupple] A tupple with the data read. None otherwise

`pyrad.io.`**`read_histogram`**(*fname*)

Reads a histogram contained in a csv file

Parameters

fname [str] path of time series file

Returns

hist , bin_edges [tupple] The read data. None otherwise

`pyrad.io.`**`read_histogram_ts`**(*fname_list*, *datatype*, *t_res=300.0*)

Reads a colection of histogram data file and creates a time series

Parameters

fname_list [str] list of files to read

datatype [str] The data type (dBZ, ZDR, etc.)

t_res [float] time resolution [s]. If None the time resolution is taken as the median

Returns

tbin_edges, bin_edges, data_ma, datetime_arr [tupple] The read data. None otherwise

`pyrad.io.`**`read_hzt_data`**(*fname*, *chy0=255.0*, *chx0=-160.0*, *read_lib='C'*)

Reads iso-0 degree data from an HZT file

Parameters

fname [str] name of the file to read

chy0, chx0: float south west point of grid in Swiss coordinates [km]

read_lib [str] Type of METRANET read library used. Can be 'C' or 'python'

Returns

hzt_data [dictionary] dictionary with the data and metadata

`pyrad.io.`**`read_idrisi_data`**(*fname*, *field_name*, *fill_value=-99.0*)
Reads DEM data from an IDRISI .rst file

Parameters

fname [str] name of the file to read

field_name [str] name of the readed variable

fill_value [float] The fill value

Returns

dem_data [dictionary] dictionary with the data and metadata

`pyrad.io.`**`read_idrisi_metadata`**(*fname*)
Reads DEM metadata from a IDRISI .rdc file

Parameters

fname [str] name of the file to read

Returns

metadata [dictionary] dictionary with the metadata

`pyrad.io.`**`read_intercomp_scores_ts`**(*fname*, *sort_by_date=False*)
Reads a radar intercomparison scores csv file

Parameters

fname [str] path of time series file

sort_by_date [bool] if True, the read data is sorted by date prior to exit

Returns

date_vec, np_vec, meanbias_vec, medianbias_vec, quant25bias_vec,

quant75bias_vec, modebias_vec, corr_vec, slope_vec, intercep_vec,

intercep_slope1_vec [tupple] The read data. None otherwise

`pyrad.io.`**`read_last_state`**(*fname*)
Reads a file containing the date of acquisition of the last volume processed

Parameters

fname [str] name of the file to read

Returns

last_state [datetime object] the date

`pyrad.io.`**`read_lightning`**(*fname*, *filter_data=True*)
Reads lightning data contained in a text file. The file has the following fields:

flashnr: (0 is for noise) UTC seconds of the day Time within flash (in seconds) Latitude (decimal degrees) Longitude (decimal degrees) Altitude (m MSL) Power (dBm)

Parameters

fname [str] path of time series file

filter_data [Boolean] if True filter noise (flashnr = 0)

Returns

flashnr, time_data, time_in_flash, lat, lon, alt, dBm [tupple] A tupple containing the read values. None otherwise

`pyrad.io.`**`read_lightning_all`**(*fname, labels=['hydro [-]', 'KDPc [deg/Km]', 'dBZc [dBZ]', 'RhoHVc [-]', 'TEMP [deg C]', 'ZDRc [dB]']*)

Reads a file containing lightning data and co-located polarimetric data. fields:

flashnr time data Time within flash (in seconds) Latitude (decimal degrees) Longitude (decimal degrees) Altitude (m MSL) Power (dBm) Polarimetric values at flash position

Parameters

fname [str] path of time series file

labels [list of str] The polarimetric variables labels

Returns

flashnr, time_data, time_in_flash, lat, lon, alt, dBm,

pol_vals_dict [tupple] A tupple containing the read values. None otherwise

`pyrad.io.`**`read_lightning_traj`**(*fname*)

Reads lightning trajectory data contained in a csv file. The file has the following fields:

Date UTC [seconds since midnight] # Flash Flash Power (dBm) Value at flash Mean value in a 3x3x3 polar box Min value in a 3x3x3 polar box Max value in a 3x3x3 polar box # valid values in the polar box

Parameters

fname [str] path of time series file

Returns

time_flash, flashnr, dBm, val_at_flash, val_mean, val_min, val_max,

nval [tupple] A tupple containing the read values. None otherwise

`pyrad.io.`**`read_meteorage`**(*fname*)

Reads METEORAGE lightning data contained in a text file. The file has the following fields:

date: date + time + time zone lon: longitude [degree] lat: latitude [degree] intens: amplitude [kilo amperes] ns: number of strokes of the flash mode: kind of localization [0,15] intra: 1 = intra-cloud , 0 = cloud-to-ground ax: length of the semi-major axis of the ellipse [km] ki2: standard deviation on the localization computation (Ki^2) ecc: eccentricity (major-axis / minor-axis) incl: ellipse inclination (angle with respect to the North, +90° is

East) [degrees]

sind: stroke index within the flash

Parameters

fname [str] path of time series file

Returns

stroke_time, lon, lat, intens, ns, mode, intra, ax, ki2, ecc, incl,

sind [tupple] A tupple containing the read values. None otherwise

pyrad.io.**read_ml_ts**(*fname*)
Reads a melting layer time series contained in a csv file

Parameters

fname [str] path of time series file

Returns

dt_ml, ml_top_avg, ml_top_std, thick_avg, thick_std, nrays_valid,

nrays_total [tupple] The read data. None otherwise

pyrad.io.**read_monitoring_ts**(*fname*, *sort_by_date=False*)
Reads a monitoring time series contained in a csv file

Parameters

fname [str] path of time series file

sort_by_date [bool] if True, the read data is sorted by date prior to exit

Returns

date , np_t, central_quantile, low_quantile, high_quantile [tupple] The read data. None otherwise

pyrad.io.**read_proc_periods**(*fname*)
Reads a file containing the start and stop times of periods to process

Parameters

fname [str] name of the file to read

Returns

starttimes, endtimes [array of datetime objects or None] The start and end times of the periods to process if the reading has been successful

pyrad.io.**read_profile_ts**(*fname_list*, *labels*, *hres=None*, *label_nr=0*, *t_res=300.0*)
Reads a colection of profile data file and creates a time series

Parameters

fname_list [str] list of files to read

labels [list of str] The data labels

hres [float] Height resolution

label_nr [int] the label nr of the data that will be used in the time series

t_res [float] time resolution [s]. If None the time resolution is taken as the median

Returns

tbin_edges, hbin_edges, np_ma, data_ma, datetime_arr [tupple] The read data. None otherwise

pyrad.io.**read_quantiles**(*fname*)
Reads quantiles contained in a csv file

Parameters

fname [str] path of time series file

Returns

quantiles, values [tupple] The read data. None otherwise

`pyrad.io.`**`read_quantiles_ts`**(*fname_list*, *step=5.0*, *qmin=0.0*, *qmax=100.0*, *t_res=300.0*)
Reads a colection of quantiles data file and creates a time series

Parameters

fname_list [str] list of files to read

step, qmin, qmax [float] The minimum, maximum and step quantiles

t_res [float] time resolution [s]. If None the time resolution is taken as the median

Returns

tbin_edges, qbin_edges, data_ma, datetime_arr [tupple] The read data. None otherwise

`pyrad.io.`**`read_rad4alp_cosmo`**(*fname*, *datatype*, *ngates=0*)
Reads rad4alp COSMO data binary file.

Parameters

fname [str] name of the file to read

datatype [str] name of the data type

ngates [int] maximum number of range gates per ray. If larger than 0 the radar field will be cut accordingly.

Returns

field [dictionary] The data field

`pyrad.io.`**`read_rad4alp_vis`**(*fname*, *datatype*)
Reads rad4alp visibility data binary file.

Parameters

fname [str] name of the file to read

datatype [str] name of the data type

Returns

field_list [list of dictionaries] A data field. Each element of the list corresponds to one elevation

`pyrad.io.`**`read_rhi_profile`**(*fname, labels=['50.0-percentile', '25.0-percentile', '75.0-percentile']*)
Reads a monitoring time series contained in a csv file

Parameters

fname [str] path of time series file

labels [list of str] The data labels

Returns

height, np_t, vals [tupple] The read data. None otherwise

`pyrad.io.`**`read_selfconsistency`**(*fname*)
Reads a self-consistency table with Zdr, Kdp/Zh columns

Parameters

fname [str] path of time series file

Returns

zdr, kdpzh [arrays] The read values

`pyrad.io.read_smn(fname)`
Reads SwissMetNet data contained in a csv file

Parameters

fname [str] path of time series file

Returns

smn_id, date , pressure, temp, rh, precip, wspeed, wdir [tupple] The read values

`pyrad.io.read_smn2(fname)`
Reads SwissMetNet data contained in a csv file with format station,time,value

Parameters

fname [str] path of time series file

Returns

smn_id, date , value [tupple] The read values

`pyrad.io.read_solar_flux(fname)`
Reads solar flux data from the DRAO observatory in Canada

Parameters

fname [str] path of time series file

Returns

flux_datetime [datetime array] the date and time of the solar flux retrievals

flux_value [array] the observed solar flux

`pyrad.io.read_status(voltime, cfg, ind_rad=0)`
Reads rad4alp xml status file.

Parameters

voltime [datetime object] volume scan time

cfg: dictionary of dictionaries configuration info to figure out where the data is

ind_rad: int radar index

Returns

root [root element object] The information contained in the status file

`pyrad.io.read_sun_hits(fname)`
Reads sun hits data contained in a csv file

Parameters

fname [str] path of time series file

Returns

date, ray, nrng, rad_el, rad_az, sun_el, sun_az, ph, ph_std, nph, nvalh,

pv, pv_std, npv, nvalv, zdr, zdr_std, nzdr, nvalzdr [tupple] Each parameter is an array containing a time series of information on a variable

`pyrad.io.read_sun_hits_multiple_days(cfg, time_ref, nfiles=1)`
Reads sun hits data from multiple file sources

Parameters

cfg [dict] dictionary with configuration data to find out the right file

time_ref [datetime object] reference time

nfiles [int] number of files to read

Returns

date, ray, nrng, rad_el, rad_az, sun_el, sun_az, ph, ph_std, nph, nvalh,

pv, pv_std, npv, nvalv, zdr, zdr_std, nzdr, nvalzdr [tupple] Each parameter is an array containing a time series of information on a variable

`pyrad.io.`**`read_sun_retrieval`**(*fname*)

Reads sun retrieval data contained in a csv file

Parameters

fname [str] path of time series file

Returns

first_hit_time, last_hit_time, nhits_h, el_width_h, az_width_h, el_bias_h,

az_bias_h, dBm_sun_est, std_dBm_sun_est, sf_h,

nhits_v, el_width_v, az_width_v, el_bias_v, az_bias_v, dBmv_sun_est,

std_dBmv_sun_est, sf_v,

nhits_zdr, zdr_sun_est, std_zdr_sun_est,

sf_ref, ref_time [tupple] Each parameter is an array containing a time series of information on a variable

`pyrad.io.`**`read_thundertracking_info`**(*fname*)

Reads the TRT info used for thundertracking

Parameters

fname [str] Name of the file containing the info

Returns

A tupple containing the read values. None otherwise. The read values are

id, max_rank, nscans_Xband, time_start, time_end

`pyrad.io.`**`read_timeseries`**(*fname*)

Reads a time series contained in a csv file

Parameters

fname [str] path of time series file

Returns

date , value [tupple] A datetime object array containing the time and a numpy masked array containing the value. None otherwise

`pyrad.io.`**`read_trt_cell_lightning`**(*fname*)

Reads the lightning data of a TRT cell. The file has the following fields:

traj_ID yyyymmddHHMM lon lat area RANKr nflashes flash_dens

Parameters

fname [str] path of the TRT data file

Returns

A tupple containing the read values. None otherwise

`pyrad.io.`**`read_trt_data`**(*fname*)

Reads the TRT data contained in a text file. The file has the following fields:

traj_ID yyyymmddHHMM

Description of ellipsis: lon [deg] lat [deg] ell_L [km] long ell_S [km] short ell_or [deg] orientation area [km2]

Cell speed: vel_x [km/h] vel_y [km/h] det [dBZ]: detection threshold RANKr from 0 to 40 (int)

Lightning information: CG- number (int) CG+ number (int) CG number (int) %CG+ [%]

Echo top information: ET45 [km] echotop 45 max ET45m [km] echotop 45 median ET15 [km] echotop 15 max ET15m [km] echotop 15 median

VIL and max echo: VIL [kg/m2] vertical integrated liquid content maxH [km] height of maximum reflectivity (maximum on the cell) maxHm [km] height of maximum reflectivity (median per cell)

POH [%] RANK (deprecated)

standard deviation of the current time step cell velocity respect to the previous time: Dvel_x [km/h] Dvel_y [km/h]

cell_contour_lon-lat

Parameters

fname [str] path of the TRT data file

Returns

A tupple containing the read values. None otherwise

`pyrad.io.`**`read_trt_info`**(*fname*)

Reads the TRT info used for thundertracking and contained in a text file.

Parameters

fname [str] path of the TRT info file

Returns

A tupple containing the read values. None otherwise. The read values are

trt_time, id, rank, nscans, azi, rng, lat, lon, ell_l, ell_s, ell_or,

vel_x, vel_y, det

`pyrad.io.`**`read_trt_info2`**(*fname*)

Reads the TRT info used for thundertracking and contained in a text file.

Parameters

fname [str] path of the TRT info file

Returns

A tupple containing the read values. None otherwise. The read values are

trt_time, id, rank, scan_time, azi, rng, lat, lon, ell_l, ell_s, ell_or,

vel_x, vel_y, det

pyrad.io.**read_trt_info_all**(*info_path*)
Reads all the TRT info files

Parameters

info_path [str] directory where the files are stored

Returns

A tupple containing the read values. None otherwise. The read values are

trt_time, id, rank, nscans, azi, rng, lat, lon, ell_l, ell_s, ell_or,

vel_x, vel_y, det

pyrad.io.**read_trt_info_all2**(*info_path*)
Reads all the TRT info files

Parameters

info_path [str] directory where the files are stored

Returns

A tupple containing the read values. None otherwise. The read values are

trt_time, id, rank, scan_time, azi, rng, lat, lon, ell_l, ell_s, ell_or,

vel_x, vel_y, det

pyrad.io.**read_trt_scores**(*fname*)
Reads the TRT scores contained in a text file. The file has the following fields:

traj ID max flash density time max flash density rank max flash density max rank time max rank

Parameters

fname [str] path of the TRT data file

Returns

A tupple containing the read values. None otherwise

pyrad.io.**read_trt_thundertracking_traj_data**(*fname*)
Reads the TRT cell data contained in a text file. The file has the following fields:

traj_ID scan_ordered_time scan_time azi rng yyyymmddHHMM

lon [deg] lat [deg] ell_L [km] long ell_S [km] short ell_or [deg] orientation area [km2]

vel_x [km/h] cell speed vel_y [km/h] det [dBZ] detection threshold RANKr from 0 to 40 (int)

CG- number (int) CG+ number (int) CG number (int) %CG+ [%]

ET45 [km] echotop 45 max ET45m [km] echotop 45 median ET15 [km] echotop 15 max ET15m [km] echotop 15 median VIL [kg/m2] vertical integrated liquid content maxH [km] height of maximum reflectivity (maximum on the cell) maxHm [km] height of maximum reflectivity (median per cell) POH [%] RANK (deprecated)

Standard deviation of the current time step cell velocity respect to the previous time: Dvel_x [km/h] Dvel_y [km/h]

cell_contour_lon-lat

Parameters

fname [str] path of the TRT data file

Returns

A tupple containing the read values. None otherwise

`pyrad.io.`**`read_trt_traj_data`**(*fname*)

Reads the TRT cell data contained in a text file. The file has the following fields:

traj_ID yyyymmddHHMM

lon [deg] lat [deg] ell_L [km] long ell_S [km] short ell_or [deg] orientation area [km2]

vel_x [km/h] cell speed vel_y [km/h] det [dBZ] detection threshold RANKr from 0 to 40 (int)

CG- number (int) CG+ number (int) CG number (int) %CG+ [%]

ET45 [km] echotop 45 max ET45m [km] echotop 45 median ET15 [km] echotop 15 max ET15m [km] echotop 15 median VIL [kg/m2] vertical integrated liquid content maxH [km] height of maximum reflectivity (maximum on the cell) maxHm [km] height of maximum reflectivity (median per cell) POH [%] RANK (deprecated)

Standard deviation of the current time step cell velocity respect to the previous time: Dvel_x [km/h] Dvel_y [km/h]

cell_contour_lon-lat

Parameters

fname [str] path of the TRT data file

Returns

A tupple containing the read values. None otherwise

`pyrad.io.`**`read_ts_cum`**(*fname*)

Reads a time series of precipitation accumulation contained in a csv file

Parameters

fname [str] path of time series file

Returns

date, np_radar, radar_value, np_sensor, sensor_value [tupple] The data read

`pyrad.io.`**`read_windmills_data`**(*fname*)

Read the wind mills data csv file

Parameters

fname [str] path of the windmill data file

Returns

windmill_dict [dict] A dictionary containing all the parameters or None

`pyrad.io.`**`send_msg`**(*sender*, *receiver_list*, *subject*, *fname*)

sends the content of a text file by email

Parameters

sender [str] the email address of the sender

receiver_list [list of string] list with the email addresses of the receiver

subject [str] the subject of the email

fname [str] name of the file containing the content of the email message

Returns

fname [str] the name of the file containing the content

`pyrad.io.`**`write_alarm_msg`**(*radar_name*, *param_name_unit*, *date_last*, *target*, *tol_abs*, *np_trend*, *value_trend*, *tol_trend*, *nevents*, *np_last*, *value_last*, *fname*)

writes an alarm file

Parameters

radar_name [str] Name of the radar being controlled

param_name_unit [str] Parameter and units

date_last [datetime object] date of the current event

target, tol_abs [float] Target value and tolerance

np_trend [int] Total number of points in trend

value_trend, tol_trend [float] Trend value and tolerance

nevents: int Number of events in trend

np_last [int] Number of points in the current event

value_last [float] Value of the current event

fname [str] Name of file where to store the alarm information

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_cdf`**(*quantiles*, *values*, *ntot*, *nnan*, *nclut*, *nblocked*, *nprec_filter*, *noutliers*, *ncdf*, *fname*, *use_nans=False*, *nan_value=0.0*, *filterprec=[]*, *vismin=None*, *sector=None*, *datatype=None*, *timeinfo=None*)

writes a cumulative distribution function

Parameters

quantiles [datetime array] array containing the measurement time

values [float array] array containing the average value

fname [float array] array containing the standard deviation

sector [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_colocated_data`**(*coloc_data*, *fname*)

Writes the data of gates colocated with two radars

Parameters

coloc_data [dict] dictionary containing the colocated data parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_colocated_data_time_avg`**(*coloc_data*, *fname*)

Writes the time averaged data of gates colocated with two radars

Parameters

coloc_data [dict] dictionary containing the colocated data parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_colocated_gates(coloc_gates, fname)`

Writes the position of gates colocated with two radars

Parameters

coloc_gates [dict] dictionary containing the colocated gates parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_excess_gates(excess_dict, fname)`

Writes the position and values of gates that have a frequency of occurrence higher than a particular threshold

Parameters

excess_dict [dict] dictionary containing the gates parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_field_coverage(quantiles, values, ele_start, ele_stop, azi_start, azi_stop, threshold, nvalid_min, datatype, timeinfo, fname)`

writes the quantiles of the coverage on a particular sector

Parameters

quantiles [datetime array] array containing the quantiles computed

values [float array] quantile value

ele_start, ele_stop, azi_start, azi_stop [float] The limits of the sector

threshold [float] The minimum value to consider the data valid

nvalid_min [int] the minimum number of points to consider that there are values in a ray

datatype [str] data type and units

timeinfo [datetime object] the time stamp of the data

fname [str] name of the file where to write the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_fixed_angle(time_data, fixed_angle, rad_lat, rad_lon, rad_alt, fname)`

writes an output file with the fixed angle data

Parameters

time_data [datetime object] The scan time

fixed_angle [float] The first fixed angle in the scan

rad_lat, rad_lon, rad_alt [float] Latitude, longitude [deg] and altitude [m MSL] of the radar

fname [str] The name of the file where to write

Returns

fname [str] the name of the file containing the content

`pyrad.io.`**`write_histogram`**(*bin_edges*, *values*, *fname*, *datatype='undefined'*, *step=0*)
writes a histogram

Parameters

bin_edges [float array] array containing the histogram bin edges

values [int array] array containing the number of points in each bin

fname [str] file name

datatype :str The data type

step [str] The bin step

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_intercomp_scores_ts`**(*start_time*, *stats*, *field_name*, *fname*, *rad1_name='RADAR001'*, *rad2_name='RADAR002'*, *rewrite=False*)
writes time series of radar intercomparison scores

Parameters

start_time [datetime object or array of date time objects] the time of the intercomparison

stats [dict] dictionary containing the statistics

field_name [str] The name of the field

fname [str] file name where to store the data

rad1_name, rad2_name [str] Name of the radars intercompared

rewrite [bool] if True a new file is created

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_last_state`**(*datetime_last*, *fname*)
writes SwissMetNet data in format datetime,avg_value, std_value

Parameters

datetime_last [datetime object] date and time of the last state

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_monitoring_ts`**(*start_time*, *np_t*, *values*, *quantiles*, *datatype*, *fname*, *rewrite=False*)
writes time series of data

Parameters

start_time [datetime object or array of date time objects] the time of the monitoring

np_t [int or array of ints] the total number of points

values: float array with 3 elements of array of arrays the values at certain quantiles

quantiles: float array with 3 elements the quantiles computed

datatype [str] The data type

fname [str] file name where to store the data

rewrite [bool] if True a new file is created

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_proc_periods`**(*start_times*, *end_times*, *fname*)
writes an output file containing start and stop times of periods to process

Parameters

start_times, end_times [datetime object] The starting and ending times of the periods

fname [str] The name of the file where to write

Returns

fname [str] the name of the file containing the content

`pyrad.io.`**`write_quantiles`**(*quantiles*, *values*, *fname*, *datatype='undefined'*)
writes quantiles

Parameters

quantiles [float array] array containing the quantiles to write

values [float array] array containing the value of each quantile

fname [str] file name

datatype :str The data type

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_rhi_profile`**(*hvec*, *data*, *nvalid_vec*, *labels*, *fname*, *datatype=None*, *timeinfo=None*, *sector=None*)
writes the values of an RHI profile in a text file

Parameters

hvec [float array] array containing the alitude in m MSL

data [list of float array] the quantities at each altitude

nvalid_vec [int array] number of valid data points used to compute the quantiles

labels [list of strings] label specifying the quantitites in data

fname [str] file name where to store the data

datatype [str] the data type

timeinfo [datetime object] time of the rhi profile

sector [dict] dictionary specying the sector limits

Returns

fname [str] the name of the file where data has been written

pyrad.io.**write_smn**(*datetime_vec*, *value_avg_vec*, *value_std_vec*, *fname*)
writes SwissMetNet data in format datetime,avg_value, std_value

Parameters

datetime_vec [datetime array] array containing the measurement time

value_avg_vec [float array] array containing the average value

value_std_vec [float array] array containing the standard deviation

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_sun_hits**(*sun_hits*, *fname*)
Writes sun hits data.

Parameters

sun_hits [dict] dictionary containing the sun hits parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_sun_retrieval**(*sun_retrieval*, *fname*)
Writes sun retrieval data.

Parameters

sun_retrieval [dict] dictionary containing the sun retrieval parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_trt_cell_data**(*traj_ID*, *yyyymmddHHMM*, *lon*, *lat*, *ell_L*, *ell_S*, *ell_or*, *area*, *vel_x*, *vel_y*, *det*, *RANKr*, *CG_n*, *CG_p*, *CG*, *CG_percent_p*, *ET45*, *ET45m*, *ET15*, *ET15m*, *VIL*, *maxH*, *maxHm*, *POH*, *RANK*, *Dvel_x*, *Dvel_y*, *cell_contour*, *fname*)
writes TRT cell data

Parameters

traj_ID, yyyymmddHHMM, lon, lat, ell_L, ell_S, ell_or, area,

vel_x, vel_y, det, RANKr, CG_n, CG_p, CG, CG_percent_p, ET45,

ET45m, ET15, ET15m, VIL, maxH, maxHm, POH, RANK, Dvel_x,

Dvel_y, cell_contour: the cell parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_trt_cell_lightning**(*cell_ID*, *cell_time*, *lon*, *lat*, *area*, *rank*, *nflash*, *flash_density*, *fname*, *timeformat='%Y%m%d%H%M'*)
writes the lightning data for each TRT cell

Parameters

cell_ID [array of ints] the cell ID

cell_time [array of datetime] the time step

lon, lat [array of floats] the latitude and longitude of the center of the cell

area [array of floats] the area of the cell

rank [array of floats] the rank of the cell

nflash [array of ints] the number of flashes/sources within the cell

flash_density [array of floats] the flash/source density

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_trt_cell_scores**(*traj_ID*, *flash_density_max_time*, *flash_density_max_rank*, *nflashes_max_list*, *area_flash_max_list*, *flash_density_max*, *rank_max_time*, *rank_max*, *fname*)

writes TRT cells scores

Parameters

traj_ID [array of ints] The ID of the cells

flash_density_max_time [array of date times] The time at which the maximum flash density was reached for each cell

flash_density_max_rank [array of floats] The rank when the maximum flash density was reached for each cell

nflashes_max_list [array of ints] the number of flashes when the max flash density was reached

area_flash_max_list [array of floats] The area when the max flash density was reached

flash_density_max [array of floats] The maximum flash density for each cell

rank_max_time [array of datetime] the time at wich the maximum rank of each cell was reached

rank_max [array of float] the rank when the maximum rank of each cell was reached

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_trt_info**(*ids*, *max_rank*, *nscans*, *time_start*, *time_end*, *fname*)

writes TRT info of the thundertracking

Parameters

ids, max_rank, nscans, time_start, time_end: array the cell parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_trt_rpc**(*cell_ID*, *cell_time*, *lon*, *lat*, *area*, *rank*, *hmin*, *hmax*, *freq*, *fname*, *timeformat='%Y%m%d%H%M'*)
writes the rimed particles column data for a TRT cell

Parameters

cell_ID [array of ints] the cell ID

cell_time [array of datetime] the time step

lon, lat [array of floats] the latitude and longitude of the center of the cell

area [array of floats] the area of the cell

rank [array of floats] the rank of the cell

hmin, hmax [array of floats] Minimum and maximum altitude of the rimed particle column

freq [array of floats] Frequency of the species constituting the rime particle column within the limits of it

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_trt_thundertracking_data**(*traj_ID*, *scan_ordered_time*, *scan_time*, *azi*, *rng*, *yyyymmddHHMM*, *lon*, *lat*, *ell_L*, *ell_S*, *ell_or*, *area*, *vel_x*, *vel_y*, *det*, *RANKr*, *CG_n*, *CG_p*, *CG*, *CG_percent_p*, *ET45*, *ET45m*, *ET15*, *ET15m*, *VIL*, *maxH*, *maxHm*, *POH*, *RANK*, *Dvel_x*, *Dvel_y*, *cell_contour*, *fname*)
writes TRT cell data of the thundertracking scan

Parameters

traj_ID, scan_ordered_time, scan_time, azi, rng, yyyymmddHHMM, lon, lat,

ell_L, ell_S, ell_or, area, vel_x, vel_y, det, RANKr, CG_n, CG_p, CG,

CG_percent_p, ET45, ET45m, ET15, ET15m, VIL, maxH, maxHm, POH, RANK,

Dvel_x, Dvel_y, cell_contour: the cell parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_ts_cum**(*dataset*, *fname*)
writes time series accumulation of data

Parameters

dataset [dict] dictionary containing the time series parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

pyrad.io.**write_ts_grid_data**(*dataset*, *fname*)
writes time series of data

Parameters

dataset [dict] dictionary containing the time series parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_ts_lightning`**(*flashnr*, *time_data*, *time_in_flash*, *lat*, *lon*, *alt*, *dBm*, *vals_list*, *fname*, *pol_vals_labels*)

writes the LMA sources data and the value of the colocated polarimetric variables

Parameters

flashnr [int] flash number

time_data [datetime object] flash source time

time_in_flash [float] seconds since start of flash

lat, lon, alt [float] latitude, longitude [deg] and altitude [m MSL] of the flash source

dBm [float] flash power

vals_list [list of arrays] List containing the data for each polarimetric variable

fname [str] the name of the file containing the content

pol_values_labels [list of strings] List containing strings identifying each polarimetric variable

Returns

fname [str] the name of the file containing the content

`pyrad.io.`**`write_ts_polar_data`**(*dataset*, *fname*)

writes time series of data

Parameters

dataset [dict] dictionary containing the time series parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.`**`write_ts_stats`**(*dt*, *value*, *fname*, *stat='mean'*)

writes time series of statistics

Parameters

dt [date time array] array of time steps

value: float arrays the average and the standard deviation of the melting layer top height

fname [str] file name where to store the data

stat [str] Statistic that is written

Returns

fname [str] the name of the file where data has written

CHAPTER FIVE

PLOTTING (PYRAD.GRAPH)

Functions to plot graphics.

5.1 Plots

`plot_ray`(radar, field_name, ind_ray, prdcfg, …)	plots a ray
`plot_surface`(grid, field_name, level, … [, …])	plots a surface from gridded data
`plot_latitude_slice`(grid, field_name, lon, …)	plots a latitude slice from gridded data
`plot_longitude_slice`(grid, field_name, lon, …)	plots a longitude slice from gridded data
`plot_latlon_slice`(grid, field_name, coord1, …)	plots a croos section crossing two points in the grid
`plot_ppi`(radar, field_name, ind_el, prdcfg, …)	plots a PPI
`plot_ppi_contour`(radar, field_name, ind_el, …)	plots contour data on a PPI
`plot_ppi_map`(radar, field_name, ind_el, …)	plots a PPI on a geographic map
`plot_rhi`(radar, field_name, ind_az, prdcfg, …)	plots an RHI
`plot_rhi_contour`(radar, field_name, ind_az, …)	plots contour data on an RHI
`plot_bscope`(radar, field_name, ind_sweep, …)	plots a B-Scope (angle-range representation)
`plot_fixed_rng`(radar, field_name, prdcfg, …)	plots a fixed range plot
`plot_fixed_rng_span`(radar, field_name, …)	plots a fixed range plot
`plot_time_range`(radar, field_name, … [, …])	plots a time-range plot
`plot_rhi_profile`(data_list, hvec, fname_list)	plots an RHI profile
`plot_along_coord`(xval_list, yval_list, …)	plots data along a certain radar coordinate
`plot_field_coverage`(xval_list, yval_list, …)	plots a time series
`plot_density`(hist_obj, hist_type, … [, …])	density plot (angle-values representation)
`plot_cappi`(radar, field_name, altitude, …)	plots a Constant Altitude Plan Position Indicator CAPPI
`plot_traj`(rng_traj, azi_traj, ele_traj, …)	plots a trajectory on a Cartesian surface
`plot_pos`(lat, lon, alt, fname_list[, ax, …])	plots a trajectory on a Cartesian surface
`plot_pos_map`(lat, lon, alt, fname_list[, …])	plots a trajectory on a map
`plot_quantiles`(quant, value, fname_list[, …])	plots quantiles
`plot_histogram`(bin_edges, values, fname_list)	computes and plots histogram
`plot_histogram2`(bin_centers, hist, fname_list)	plots histogram
`plot_antenna_pattern`(antpattern, fname_list)	plots an antenna pattern
`plot_selfconsistency`(zdrkdp_table, fname_list)	plots a ZDR-KDP/ZH selfconsistency in rain relation
`plot_selfconsistency_instrument`(zdr, kdp, …)	plots the ZDR-KDP/ZH relationship obtained by an instrument.
`plot_timeseries`(tvec, data_list, fname_list)	plots a time series

Continued on next page

Table 1 – continued from previous page

plot_timeseries_comp(date1, value1, date2, …)	plots 2 time series in the same graph
plot_monitoring_ts(date, np_t, cquant, …)	plots a time series of monitoring data
plot_scatter_comp(value1, value2, fname_list)	plots the scatter between two time series
plot_intercomp_scores_ts(date_vec, np_vec, …)	plots a time series of radar intercomparison scores
plot_ml_ts(dt_ml_arr, ml_top_avg_arr, …[, …])	plots a time series of melting layer data
plot_sun_hits(field, field_name, fname_list, …)	plots the sun hits
plot_sun_retrieval_ts(sun_retrieval, …[, …])	plots sun retrieval time series series
plot_Doppler(spectra, field_name, ray, rng, …)	Makes a Doppler plot
plot_complex_Doppler(spectra, field_name, …)	Makes a complex Doppler plot plotting separately the real and the imaginary parts
plot_amp_phase_Doppler(spectra, field_name, …)	Makes a complex Doppler plot plotting separately the module and the phase of the signal
plot_range_Doppler(spectra, field_name, ray, …)	Makes a range-Doppler plot
plot_complex_range_Doppler(spectra, …[, …])	Makes a complex range-Doppler plot.
plot_amp_phase_range_Doppler(spectra, …[, …])	Makes a complex range-Doppler plot plotting separately the module and the phase of the signal
plot_angle_Doppler(spectra, field_name, ang, …)	Makes an angle-Doppler plot
plot_complex_angle_Doppler(spectra, …[, …])	Makes an angle-Doppler plot of complex spectra
plot_amp_phase_angle_Doppler(spectra, …[, …])	Makes an angle-Doppler plot of complex spectra
plot_time_Doppler(spectra, field_name, …)	Makes a time-Doppler plot
plot_complex_time_Doppler(spectra, …[, …])	Makes a complex time-Doppler plot.
plot_amp_phase_time_Doppler(spectra, …[, …])	Makes a complex time-Doppler plot plotting separately the module and the phase of the signal
plot_roi_contour(roi_dict, prdcfg, fname_list)	plots the contour of a region of interest on a map
get_colobar_label(field_dict, field_name)	creates the colorbar label using field metadata
get_field_name(field_dict, field)	Return a nice field name for a particular field
_plot_time_range(rad_time, rad_range, …[, …])	plots a time-range plot

pyrad.graph.**get_colobar_label**(*field_dict*, *field_name*)
creates the colorbar label using field metadata

Parameters

field_dict [dict] dictionary containing field metadata

field_name [str] name of the field

Returns

label [str] colorbar label

pyrad.graph.**get_field_name**(*field_dict*, *field*)
Return a nice field name for a particular field

Parameters

field_dict [dict] dictionary containing field metadata

field [str] name of the field

Returns

field_name [str] the field name

pyrad.graph.**plot_Doppler**(*spectra*, *field_name*, *ray*, *rng*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *ylabel=None*, *titl=None*, *vmin=None*, *vmax=None*)

Makes a Doppler plot

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ray, rng [int] ray and rng index

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

ylabel [str or None] The label of the y-axis

titl [str or None] The plot title

vmin, vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_along_coord**(*xval_list*, *yval_list*, *fname_list*, *labelx='coord'*, *labely='Value'*, *labels=None*, *title='Plot along coordinate'*, *colors=None*, *linestyles=None*, *ymin=None*, *ymax=None*, *dpi=72*)

plots data along a certain radar coordinate

Parameters

xval_list [list of float arrays] the x values, range, azimuth or elevation

yval_list [list of float arrays] the y values. Parameter to plot

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

labels [array of str] The label of the legend

title [str] The figure title

colors [array of str] Specifies the colors of each line

linestyles [array of str] Specifies the line style of each line

ymin, ymax: float Lower/Upper limit of y axis

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_amp_phase_Doppler**(*spectra*, *field_name*, *ray*, *rng*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *titl=None*, *ampli_vmin=None*, *ampli_vmax=None*, *phase_vmin=None*, *phase_vmax=None*)

Makes a complex Doppler plot plotting separately the module and the phase of the signal

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ray, rng [int] ray and range index

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

titl [str or None] The plot title

ampli_vmin, ampli_vmax, phase_vmin, phase_vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_amp_phase_angle_Doppler**(*spectra*, *field_name*, *ang*, *ind_rays*, *ind_rng*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *yaxis_pos='centre'*, *along_azi=True*, *titl=None*, *ampli_vmin=None*, *ampli_vmax=None*, *phase_vmin=None*, *phase_vmax=None*)

Makes an angle-Doppler plot of complex spectra

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ang [float] The fixed angle

ind_rays [1D int array] The indices of the rays to plot

ind_rng [int] The index of the range to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

yaxis_pos [str] the position that the y point represents in the y-axis bin. Can be 'start', end' or 'centre'

along_azi [bool] If true the plot is performed along azimuth. If false it is performed along elevation

titl [str or None] The plot title

ampli_vmin, ampli_vmax, phase_vmin, phase_vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots pyrad.graph.**plot_amp_phase_range_Doppler**(*spectra*, *field_name*, *ray*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *titl=None*, *ampli_vmin=None*, *ampli_vmax=None*, *phase_vmin=None*, *phase_vmax=None*)

Makes a complex range-Doppler plot plotting separately the module and the phase of the signal

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ray [int] ray index

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

titl [str or None] The plot title

ampli_vmin, ampli_vmax, phase_vmin, phase_vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_amp_phase_time_Doppler**(*spectra*, *field_name*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *yaxis_pos='start'*, *titl=None*, *ampli_vmin=None*, *ampli_vmax=None*, *phase_vmin=None*, *phase_vmax=None*)

Makes a complex time-Doppler plot plotting separately the module and the phase of the signal

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

yaxis_pos [str] the position that the y point represents in the y-axis bin. Can be 'start', end' or 'centre'

titl [str or None] The plot title

ampli_vmin, ampli_vmax, phase_vmin, phase_vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_angle_Doppler**(*spectra*, *field_name*, *ang*, *ind_rays*, *ind_rng*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *yaxis_pos='centre'*, *along_azi=True*, *titl=None*, *clabel=None*, *vmin=None*, *vmax=None*)

Makes an angle-Doppler plot

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ang [float] The fixed angle

ind_rays [1D int array] The indices of the rays to plot

ind_rng [int] The index of the range to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

yaxis_pos [str] the position that the y point represents in the y-axis bin. Can be 'start', end' or 'centre'

along_azi [bool] If true the plot is performed along azimuth. If false it is performed along elevation

titl [str or None] The plot title

clabel [str or None] The color bar label

vmin, vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

`pyrad.graph.`**`plot_antenna_pattern`**(*antpattern*, *fname_list*, *labelx='Angle [Deg]'*, *linear=False*, *twoway=False*, *title='Antenna Pattern'*, *ymin=None*, *ymax=None*, *dpi=72*)

plots an antenna pattern

Parameters

antpattern [dict] dictionary with the angle and the attenuation

value [float array] values of the time series

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

linear [boolean] if true data is in linear units

linear [boolean] if true data represents the two way attenuation

titl [str] The figure title

ymin, ymax: float Lower/Upper limit of y axis

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.`**`plot_bscope`**(*radar*, *field_name*, *ind_sweep*, *prdcfg*, *fname_list*, *vmin=None*, *vmax=None*, *ray_dim='ang'*, *xaxis_rng=True*)

plots a B-Scope (angle-range representation)

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_sweep [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

vmin, vmax [float] Min and max values of the colorbar

ray_dim [str] the ray dimension. Can be 'ang' or 'time'

xaxis [bool] if true the range will be in the x-axis. Otherwise it will be in the y-axis.

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_cappi**(*radar*, *field_name*, *altitude*, *prdcfg*, *fname_list*, *beamwidth=1.0*, *beam_spacing=1.0*, *save_fig=True*)

plots a Constant Altitude Plan Position Indicator CAPPI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

altitude [float] the altitude [m MSL] to be plotted

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

beamwidth [float] The radar beamwidth

beam_spacing [float] the ray angle resolution

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

pyrad.graph.**plot_complex_Doppler**(*spectra*, *field_name*, *ray*, *rng*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *ylabel=None*, *titl=None*, *vmin=None*, *vmax=None*)

Makes a complex Doppler plot plotting separately the real and the imaginary parts

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ray, rng [int] ray and range index

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

ylabel [str or None] The label of the y-axis

titl [str or None] The plot title

vmin, vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_complex_angle_Doppler**(*spectra*, *field_name*, *ang*, *ind_rays*, *ind_rng*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *yaxis_pos='centre'*, *along_azi=True*, *titl=None*, *clabel=None*, *vmin=None*, *vmax=None*)

Makes an angle-Doppler plot of complex spectra

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ang [float] The fixed angle

ind_rays [1D int array] The indices of the rays to plot

ind_rng [int] The index of the range to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

yaxis_pos [str] the position that the y point represents in the y-axis bin. Can be 'start', end' or 'centre'

along_azi [bool] If true the plot is performed along azimuth. If false it is performed along elevation

titl [str or None] The plot title

clabel [str or None] The color bar label

vmin, vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_complex_range_Doppler**(*spectra*, *field_name*, *ray*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *titl=None*, *clabel=None*, *vmin=None*, *vmax=None*)

Makes a complex range-Doppler plot. Plotting separately the real and the imaginary part

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ray [int] ray index

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

titl [str or None] The plot title

clabel [str or None] The label of color bar

vmin, vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_complex_time_Doppler**(*spectra*, *field_name*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *yaxis_pos='start'*, *titl=None*, *clabel=None*, *vmin=None*, *vmax=None*)

Makes a complex time-Doppler plot. Plotting separately the real and the imaginary part

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

yaxis_pos [str] the position that the y point represents in the y-axis bin. Can be 'start', end' or 'centre'

titl [str or None] The plot title

clabel [str or None] The label of color bar

vmin, vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_density**(*hist_obj, hist_type, field_name, ind_sweep, prdcfg, fname_list, quantiles=[25.0, 50.0, 75.0], ref_value=0.0, vmin=None, vmax=None*)

density plot (angle-values representation)

Parameters

hist_obj [histogram object] object containing the histogram data to plot

hist_type [str] type of histogram (instantaneous data or cumulative)

field_name [str] name of the radar field to plot

ind_sweep [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

quantiles [array] the quantile lines to plot

ref_value [float] the reference value

vmin, vmax [float] Minim and maximum extend of the vertical axis

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_field_coverage**(*xval_list*, *yval_list*, *fname_list*, *labelx='Azimuth (deg)'*, *labely='Range extension [m]'*, *labels=None*, *title='Field coverage'*, *ymin=None*, *ymax=None*, *xmeanval=None*, *ymeanval=None*, *labelmeanval=None*, *dpi=72*)

plots a time series

Parameters

xval_list [list of float arrays] the x values, azimuth

yval_list [list of float arrays] the y values. Range extension

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

labels [array of str] The label of the legend

title [str] The figure title

ymin, ymax [float] Lower/Upper limit of y axis

xmeanval, ymeanval [float array] the x and y values of a mean along elevation

labelmeanval [str] the label of the mean

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_fixed_rng**(*radar*, *field_name*, *prdcfg*, *fname_list*, *azi_res=None*, *ele_res=None*, *ang_tol=1.0*, *vmin=None*, *vmax=None*)

plots a fixed range plot

Parameters

radar [radar object] The radar object containing the fixed range data

field_name [str] The name of the field to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

azi_res, ele_res [float] The nominal azimuth and elevation angle resolution [deg]

ang_tol [float] The tolerance between the nominal and the actual radar angle

vmin, vmax [float] Min and Max values of the color scale. If None it is going to be taken from the Py-ART config files

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_fixed_rng_span**(*radar*, *field_name*, *prdcfg*, *fname_list*, *azi_res=None*, *ele_res=None*, *ang_tol=1.0*, *stat='max'*)

plots a fixed range plot

Parameters

radar [radar object] The radar object containing the fixed range data

field_name [str] The name of the field to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

azi_res, ele_res [float] The nominal azimuth and elevation angle resolution [deg]

ang_tol [float] The tolerance between the nominal and the actual radar angle

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_histogram**(*bin_edges*, *values*, *fname_list*, *labelx='bins'*, *labely='Number of Samples'*, *titl='histogram'*, *dpi=72*)

computes and plots histogram

Parameters

bin_edges [array] histogram bin edges

values [array] data values

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

titl [str] The figure title

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_histogram2**(*bin_centers*, *hist*, *fname_list*, *width=None*, *labelx='bins'*, *labely='Number of Samples'*, *titl='histogram'*, *dpi=72*, *ax=None*, *fig=None*, *save_fig=True*, *color=None*, *alpha=None*, *invert_xaxis=False*)

plots histogram

Parameters

bin_centers [array] histogram bin centers

hist [array] values for each bin

fname_list [list of str] list of names of the files where to store the plot

width [scalar or array-like] the width(s) of the bars. If None it is going to be estimated from the distances between centers

labelx [str] The label of the X axis

labely [str] The label of the Y axis

titl [str] The figure title

dpi [int] dots per inch

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

color [str] color of the bars

alpha [float] parameter controling the transparency

invert_xaxis [bool] If true inverts the x axis

Returns

fname_list or fig, ax: list of str list of names of the created plots

pyrad.graph.**plot_intercomp_scores_ts**(*date_vec*, *np_vec*, *meanbias_vec*, *medianbias_vec*, *quant25bias_vec*, *quant75bias_vec*, *modebias_vec*, *corr_vec*, *slope_vec*, *intercep_vec*, *intercep_slope1_vec*, *fname_list*, *ref_value=0.0*, *np_min=0*, *corr_min=0.0*, *labelx='Time UTC'*, *titl='RADAR001-RADAR002 intercomparison'*, *dpi=72*)

plots a time series of radar intercomparison scores

Parameters

date_vec [datetime object] time of the time series

np_vec [int array] number of points

meanbias_vec, medianbias_vec, modebias_vec [float array] mean, median and mode bias

quant25bias_vec, quant75bias_vec: 25th and 75th percentile of the bias

corr_vec [float array] correlation

slope_vec, intercep_vec [float array] slope and intercep of a linear regression

intercep_slope1_vec [float] the intercep point of a inear regression of slope 1

ref_value [float] the reference value

np_min [int] The minimum number of points to consider the result valid

corr_min [float] The minimum correlation to consider the results valid

labelx [str] The label of the X axis

titl [str] The figure title

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_latitude_slice**(*grid*, *field_name*, *lon*, *lat*, *prdcfg*, *fname_list*)

plots a latitude slice from gridded data

Parameters

grid [Grid object] object containing the gridded data to plot

field_name [str] name of the radar field to plot

lon, lat [float] coordinates of the slice to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_latlon_slice**(*grid*, *field_name*, *coord1*, *coord2*, *prdcfg*, *fname_list*)

plots a croos section crossing two points in the grid

Parameters

grid [Grid object] object containing the gridded data to plot

field_name [str] name of the radar field to plot

coord1 [tupple of floats] lat, lon of the first point

coord2 [tupple of floats] lat, lon of the second point

fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.`**`plot_longitude_slice`**(*grid*, *field_name*, *lon*, *lat*, *prdcfg*, *fname_list*)

plots a longitude slice from gridded data

Parameters

grid [Grid object] object containing the gridded data to plot

field_name [str] name of the radar field to plot

lon, lat [float] coordinates of the slice to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.`**`plot_ml_ts`**(*dt_ml_arr*, *ml_top_avg_arr*, *ml_top_std_arr*, *thick_avg_arr*, *thick_std_arr*, *nrays_valid_arr*, *nrays_total_arr*, *fname_list*, *labelx='Time UTC'*, *titl='Melting layer time series'*, *dpi=72*)

plots a time series of melting layer data

Parameters

dt_ml_arr [datetime object] time of the time series

np_vec [int array] number of points

meanbias_vec, medianbias_vec, modebias_vec [float array] mean, median and mode bias

quant25bias_vec, quant75bias_vec: 25th and 75th percentile of the bias

corr_vec [float array] correlation

slope_vec, intercep_vec [float array] slope and intercep of a linear regression

intercep_slope1_vec [float] the intercep point of a inear regression of slope 1

ref_value [float] the reference value

np_min [int] The minimum number of points to consider the result valid

corr_min [float] The minimum correlation to consider the results valid

labelx [str] The label of the X axis

titl [str] The figure title

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_monitoring_ts**(*date*, *np_t*, *cquant*, *lquant*, *hquant*, *field_name*, *fname_list*, *ref_value=None*, *vmin=None*, *vmax=None*, *np_min=0*, *labelx='Time [UTC]'*, *labely='Value'*, *titl='Time Series'*, *dpi=72*)

plots a time series of monitoring data

Parameters

date [datetime object] time of the time series

np_t [int array] number of points

cquant, lquant, hquant [float array] values of the central, low and high quantiles

field_name [str] name of the field

fname_list [list of str] list of names of the files where to store the plot

ref_value [float] the reference value

vmin, vmax [float] The limits of the y axis

np_min [int] minimum number of points to consider the sample plotable

labelx [str] The label of the X axis

labely [str] The label of the Y axis

titl [str] The figure title

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_pos**(*lat*, *lon*, *alt*, *fname_list*, *ax=None*, *fig=None*, *save_fig=True*, *sort_altitude='No'*, *dpi=72*, *alpha=1.0*, *cb_label='height [m MSL]'*, *titl='Position'*, *xlabel='Lon [Deg]'*, *ylabel='Lat [Deg]'*, *limits=None*, *vmin=None*, *vmax=None*)

plots a trajectory on a Cartesian surface

Parameters

lat, lon, alt [float array] Points coordinates

fname_list [list of str] list of names of the files where to store the plot

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

sort_altitude [str] String indicating whether to sort the altitude data. Can be 'No', 'Lowest_on_top' or 'Highest_on_top'

dpi [int] Pixel density

alpha [float] Transparency

cb_label [str] Color bar label

titl [str] Plot title

xlabel, ylabel [str] The labels of the X and Y axis

limits [tupple or None] The limits of the field to plot

vmin, vmax [float] The limits of the color scale

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

pyrad.graph.**plot_pos_map**(*lat*, *lon*, *alt*, *fname_list*, *ax=None*, *fig=None*, *save_fig=True*, *sort_altitude='No'*, *dpi=72*, *alpha=1.0*, *cb_label='height [m MSL]'*, *titl='Position'*, *xlabel='Lon [Deg]'*, *ylabel='Lat [Deg]'*, *limits=None*, *vmin=None*, *vmax=None*, *lon_step=0.3*, *lat_step=0.1*, *background_zoom=8*)

plots a trajectory on a map

Parameters

lat, lon, alt [float array] Points coordinates

fname_list [list of str] list of names of the files where to store the plot

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

sort_altitude [str] String indicating whether to sort the altitude data. Can be 'No', 'Lowest_on_top' or 'Highest_on_top'

dpi [int] Pixel density

alpha [float] Transparency

cb_label [str] Color bar label

titl [str] Plot title

xlabel, ylabel [str] The labels of the X and Y axis

limits [tupple or None] The limits of the field to plot

vmin, vmax [float] The limits of the color scale

lon_step, lat_step [float] The step interval of the latitude, longitude lines to plot

background_zoom [int] The zoom of the background image. A higher number will give more level of detail at the expense of speed.

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

pyrad.graph.**plot_ppi**(*radar*, *field_name*, *ind_el*, *prdcfg*, *fname_list*, *plot_type='PPI'*, *titl=None*, *vmin=None*, *vmax=None*, *step=None*, *quantiles=None*, *save_fig=True*)

plots a PPI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_el [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

plot_type [str] type of plot (PPI, QUANTILES or HISTOGRAM)

titl [str] Plot title

vmin, vmax [float] The minimum and maximum value. If None the scale is going to be obtained from the Py-ART config file.

step [float] step for histogram plotting

quantiles [float array] quantiles to plot

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

`pyrad.graph.`**`plot_ppi_contour`**(*radar*, *field_name*, *ind_el*, *prdcfg*, *fname_list*, *contour_values=None*, *linewidths=1.5*, *ax=None*, *fig=None*, *save_fig=True*)

plots contour data on a PPI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_el [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

contour_values [float array] list of contours to plot

linewidths [float] width of the contour lines

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

`pyrad.graph.`**`plot_ppi_map`**(*radar*, *field_name*, *ind_el*, *prdcfg*, *fname_list*, *save_fig=True*)

plots a PPI on a geographic map

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_el [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax, display [tupple] list of names of the saved plots or handle of the figure an axes

pyrad.graph.**plot_quantiles**(*quant*, *value*, *fname_list*, *labelx='quantile'*, *labely='value'*, *titl='quantile'*, *vmin=None*, *vmax=None*, *dpi=72*)

plots quantiles

Parameters

quant [array] quantiles to be plotted

value [array] values of each quantile

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

titl [str] The figure title

vmin, vmax: float Lower/Upper limit of data values

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_range_Doppler**(*spectra*, *field_name*, *ray*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *titl=None*, *clabel=None*, *vmin=None*, *vmax=None*)

Makes a range-Doppler plot

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

ray [int] ray index

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

titl [str or None] The plot title

clabel [str or None] The color bar label

vmin, vmax [float or None] The value limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_ray**(*radar*, *field_name*, *ind_ray*, *prdcfg*, *fname_list*, *titl=None*, *vmin=None*, *vmax=None*, *save_fig=True*)

plots a ray

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_ray [int] ray index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

plot_type [str] type of plot (PPI, QUANTILES or HISTOGRAM)

titl [str] Plot title

vmin, vmax [float] min and max values of the y axis

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

pyrad.graph.**plot_rhi**(*radar*, *field_name*, *ind_az*, *prdcfg*, *fname_list*, *plot_type='RHI'*, *titl=None*, *vmin=None*, *vmax=None*, *step=None*, *quantiles=None*, *save_fig=True*)

plots an RHI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_az [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

plot_type [str] type of plot (PPI, QUANTILES or HISTOGRAM)

titl [str] Plot title

vmin, vmax [float] The minimum and maximum value. If None the scale is going to be obtained from the Py-ART config file.

step [float] step for histogram plotting

quantiles [float array] quantiles to plot

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

pyrad.graph.**plot_rhi_contour**(*radar*, *field_name*, *ind_az*, *prdcfg*, *fname_list*, *contour_values=None*, *linewidths=1.5*, *ax=None*, *fig=None*, *save_fig=True*)

plots contour data on an RHI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_az [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

contour_values [float array] list of contours to plot

linewidths [float] width of the contour lines

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

`pyrad.graph.`**`plot_rhi_profile`**(*data_list, hvec, fname_list, labelx='Value', labely='Height (m MSL)', labels=['Mean'], title='RHI profile', colors=None, linestyles=None, vmin=None, vmax=None, hmin=None, hmax=None, dpi=72*)

plots an RHI profile

Parameters

data_list [list of float array] values of the profile

hvec [float array] height points of the profile

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

labels [array of str] The label of the legend

title [str] The figure title

colors [array of str] Specifies the colors of each line

linestyles [array of str] Specifies the line style of each line

vmin, vmax: float Lower/Upper limit of data values

hmin, hmax: float Lower/Upper limit of altitude

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.`**`plot_roi_contour`**(*roi_dict, prdcfg, fname_list, plot_center=True, xlabel='Lon [Deg]', ylabel='Lat [Deg]', titl='TRT cell position', ax=None, fig=None, save_fig=True*)

plots the contour of a region of interest on a map

Parameters

roi_dict [dict] dictionary containing lon_roi, lat_roi, the points defining the contour

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

plot_center [bool] If True a marked with the center of the roi is plotted

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

`pyrad.graph.`**`plot_scatter`**(*bin_edges1*, *bin_edges2*, *hist_2d*, *field_name1*, *field_name2*, *fname_list*, *prdcfg*, *metadata=None*, *lin_regr=None*, *lin_regr_slope1=None*, *rad1_name='RADAR001'*, *rad2_name='RADAR002'*)

2D histogram

Parameters

bin_edges1, bin_edges2 [float array2] the bins of each field

hist_2d [ndarray 2D] the 2D histogram

field_name1, field_name2 [str] the names of each field

fname_list [list of str] list of names of the files where to store the plot

prdcfg [dict] product configuration dictionary

metadata [str] a string with metadata to write in the plot

lin_regr [tupple with 2 values] the coefficients for a linear regression

lin_regr_slope1 [float] the intercep point of a linear regression of slope 1

rad1_name, rad2_name [str] name of the radars which data is used

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.`**`plot_scatter_comp`**(*value1*, *value2*, *fname_list*, *labelx='Sensor 1'*, *labely='Sensor 2'*, *titl='Scatter'*, *axis=None*, *metadata=None*, *dpi=72*, *ax=None*, *fig=None*, *save_fig=True*, *point_format='bx'*)

plots the scatter between two time series

Parameters

value1 [float array] values of the first time series

value2 [float array] values of the second time series

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

titl [str] The figure title

axis [str] type of axis

metadata [string] a string containing metadata

dpi [int] dots per inch

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

point_format [str] format of the scatter point

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_selfconsistency**(*zdrkdp_table*, *fname_list*, *labelx='ZDR [dB]'*, *labely='KDP/Zh [(deg*m3)/(km*mm6)]'*, *title='Selfconsistency in rain'*, *ymin=None*, *ymax=None*, *dpi=72*, *save_fig=True*, *ax=None*, *fig=None*)

plots a ZDR-KDP/ZH selfconsistency in rain relation

Parameters

antpattern [dict] dictionary with the angle and the attenuation

value [float array] values of the time series

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

linear [boolean] if true data is in linear units

linear [boolean] if true data represents the two way attenuation

titl [str] The figure title

ymin, ymax: float Lower/Upper limit of y axis

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_selfconsistency_instrument**(*zdr*, *kdp*, *zh*, *fname_list*, *bins_zdr_step=0.05*, *bins_zdr_min=0.0*, *bins_zdr_max=6.0*, *bins_kdpzh_step=0.1*, *bins_kdpzh_min=-2.0*, *bins_kdpzh_max=20.0*, *normalize=True*, *vmin=0.0*, *vmax=0.01*, *parametrization='None'*, *zdr_kdpzh_dict=None*, *retrieve_relation=True*, *plot_theoretical=True*, *dpi=72*)

plots the ZDR-KDP/ZH relationship obtained by an instrument. The theoretical curve and the retrieved curve

Parameters

zdr, kdp, zh [1D ndarray] The valid values of ZDR [dB], KDP [deg/km] and Zh [mm6/m3] collected by the instrument

fname_list [list of str] list of names of the files where to store the plot

bins_zdr_step [float] The step of the ZDR axis of the histogram [dB]

bins_zdr_min, bins_zdr_max [float] The limits of the ZDR axis of the histogram (bins center) [dB]

bins_kdpzh_step [float] The step of the 1e5*KDP^a/ZH^b axis of the histogram [(deg*m3)/(km*mm6)]

bins_kdpzh_min, bins_kdpzh_max [float] The limits of the 1e5*KDP^a/ZH^b axis of the histogram (bins center) [(deg*m3)/(km*mm6)]

normalize [Bool] If True the occurrence density of ZH/KDP for each ZDR bin is going to be represented. Otherwise it will show the number of gates at each bin

vmin, vmax [float] min and max values of the colorbar

parametrization [str] The type of parametrization for the self-consistency curves. Can be 'None', 'Gourley', 'Wolfensberger', 'Louf', 'Gorgucci' or 'Vaccarono'. 'None' will use tables contained in zdr_kdpzh_dict. The parametrized curves are obtained from literature except for Wolfensberger that was derived from disdrometer data obtained by MeteoSwiss and EPFL. All parametrizations are valid for C-band only except that of Gourley.

zdr_kdpzh_dict [dict] dictionary containing a look up table relating ZDR with KDP/Zh for different elevations and the frequency band of the radar

retrieve_relation [boolean] if true a zdr-kdp/zh relationship is retrieved from the data

plot_theoretical [bool] if true the theoretical relationship is retrieved

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.`**`plot_sun_hits`**(*field*, *field_name*, *fname_list*, *prdcfg*)

plots the sun hits

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

altitude [float] the altitude [m MSL] to be plotted

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.`**`plot_sun_retrieval_ts`**(*sun_retrieval*, *data_type*, *fname_list*, *labelx='Date'*, *titl='Sun retrieval Time Series'*, *dpi=72*)

plots sun retrieval time series series

Parameters

sun_retrieval [tuple] tuple containing the retrieved parameters

data_type [str] parameter to be plotted

fname_list [list of str] list of names of the files where to store the plot

labelx [str] the x label

titl [str] the title of the plot

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_surface**(*grid*, *field_name*, *level*, *prdcfg*, *fname_list*, *titl=None*, *alpha=None*, *ax=None*, *fig=None*, *display=None*, *save_fig=True*, *use_basemap=False*)

plots a surface from gridded data

Parameters

grid [Grid object] object containing the gridded data to plot

field_name [str] name of the radar field to plot

level [int] level index

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

titl [str] Plot title

alpha [float or None] Set the alpha transparency of the grid plot. Useful for overplotting radar over other datasets.

ax [Axis] Axis to plot on. if fig is None a new axis will be created

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

display [GridMapDisplay object] The display used

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax, display [tupple] list of names of the saved plots or handle of the figure an axes

pyrad.graph.**plot_time_Doppler**(*spectra*, *field_name*, *prdcfg*, *fname_list*, *xaxis_info='Doppler_velocity'*, *yaxis_pos='start'*, *titl=None*, *clabel=None*, *vmin=None*, *vmax=None*, *xmin=None*, *xmax=None*, *ymin=None*, *ymax=None*)

Makes a time-Doppler plot

Parameters

spectra [radar spectra object] object containing the spectra or the IQ data to plot

field_name [str] name of the field to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

xaxis_info [str] Type of x-axis. Can be 'Doppler_velocity', 'Doppler_frequency' or 'pulse_number'

yaxis_pos [str] the position that the y point represents in the y-axis bin. Can be 'start', end' or 'centre'

titl [str or None] The plot title

clabel [str or None] The color bar label

vmin, vmax [float or None] The value limits

xmin, xmax, ymin, ymax [float or None] The axis limits

Returns

fname_list [list of str] list of names of the saved plots

pyrad.graph.**plot_time_range**(*radar*, *field_name*, *ind_sweep*, *prdcfg*, *fname_list*, *vmin=None*, *vmax=None*, *ylabel='range (Km)'*)

plots a time-range plot

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_sweep [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

vmin, vmax [float] Min and max values of the colorbar

ylabel [str] The y-axis label

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_timeseries**(*tvec, data_list, fname_list, labelx='Time [UTC]', labely='Value', labels=['Sensor'], title='Time Series', period=0, timeformat=None, colors=None, linestyles=None, markers=None, ymin=None, ymax=None, dpi=72*)

plots a time series

Parameters

tvec [datetime object] time of the time series

data_list [list of float array] values of the time series

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

labels [array of str] The label of the legend

title [str] The figure title

period [float] measurement period in seconds used to compute accumulation. If 0 no accumulation is computed

timeformat [str] Specifies the tvec and time format on the x axis

colors [array of str] Specifies the colors of each line

linestyles [array of str] Specifies the line style of each line

markers: array of str Specify the markers to be used for each line

ymin, ymax: float Lower/Upper limit of y axis

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_timeseries_comp**(*date1*, *value1*, *date2*, *value2*, *fname_list*, *labelx='Time [UTC]'*, *labely='Value'*, *label1='Sensor 1'*, *label2='Sensor 2'*, *titl='Time Series Comparison'*, *period1=0*, *period2=0*, *ymin=None*, *ymax=None*, *dpi=72*)

plots 2 time series in the same graph

Parameters

date1 [datetime object] time of the first time series

value1 [float array] values of the first time series

date2 [datetime object] time of the second time series

value2 [float array] values of the second time series

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

label1, label2 [str] legend label for each time series

titl [str]

The figure title

period1, period2 [float] measurement period in seconds used to compute accumulation. If 0 no accumulation is computed

dpi [int] dots per inch

ymin, ymax [float] The limits of the Y-axis. None will keep the default limit.

Returns

fname_list [list of str] list of names of the created plots

pyrad.graph.**plot_traj**(*rng_traj*, *azi_traj*, *ele_traj*, *time_traj*, *prdcfg*, *fname_list*, *rad_alt=None*, *rad_tstart=None*, *ax=None*, *fig=None*, *save_fig=True*)

plots a trajectory on a Cartesian surface

Parameters

rng_traj, azi_traj, ele_traj [float array] antenna coordinates of the trajectory [m and deg]

time_traj [datetime array] trajectory time

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

rad_alt [float or None] radar altitude [m MSL]

rad_tstart [datetime object or None] start time of the radar scan

surface_alt [float] surface altitude [m MSL]

color_ref [str] What the color code represents. Can be 'None', 'rel_altitude', 'altitude' or 'time'

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tupple] list of names of the saved plots or handle of the figure an axes

CHAPTER

SIX

UTILITIES (PYRAD.UTIL)

Functions to read and write data and configuration files.

6.1 Radar Utilities

get_data_along_rng(radar, field_name, ...[, ...])	Get data at particular (azimuths, elevations)
get_data_along_azi(radar, field_name, ...[, ...])	Get data at particular (ranges, elevations)
get_data_along_ele(radar, field_name, ...[, ...])	Get data at particular (ranges, azimuths)
get_ROI(radar, fieldname, sector)	filter out any data outside the region of interest defined by sector
rainfall_accumulation(t_in_vec, val_in_vec)	Computes the rainfall accumulation of a time series over a given period
time_series_statistics(t_in_vec, val_in_vec)	Computes statistics over a time-averaged series.
find_contiguous_times(times[, step])	Given and array of ordered times, find those contiguous according to a maximum time step
join_time_series(t1, val1, t2, val2[, dropnan])	joins time_series.
get_range_bins_to_avg(rad1_rng, rad2_rng)	Compares the resolution of two radars and determines if and which radar has to be averaged and the length of the averaging window
find_ray_index(ele_vec, azi_vec, ele, azi[, ...])	Find the ray index corresponding to a particular elevation and azimuth
find_rng_index(rng_vec, rng[, rng_tol])	Find the range index corresponding to a particular range
find_nearest_gate(radar, lat, lon[, latlon_tol])	Find the radar gate closest to a lat,lon point
find_neighbour_gates(radar, azi, rng[, ...])	Find the neighbouring gates within +-delta_azi and +-delta_rng
find_colocated_indexes(radar1, radar2, ...)	Given the theoretical elevation, azimuth and range of the co-located gates of two radars and a given tolerance returns the indices of the gates for the current radars
get_target_elevations(radar_in)	Gets RHI target elevations
get_fixed_rng_data(radar, field_names, fixed_rng)	Creates a 2D-grid with (azi, ele) data at a fixed range
time_avg_range(timeinfo, avg_starttime, ...)	finds the new start and end time of an averaging
get_closest_solar_flux(hit_datetime_list, ...)	finds the solar flux measurement closest to the sun hit

Continued on next page

Table 1 – continued from previous page

create_sun_hits_field(rad_el, rad_az, ...)	creates a sun hits field from the position and power of the sun hits
create_sun_retrieval_field(par, field_name, ...)	creates a sun retrieval field from the retrieval parameters
compute_quantiles(field[, quantiles])	computes quantiles
compute_quantiles_from_hist(bin_centers, hist)	computes quantiles from histograms
compute_quantiles_sweep(field, ray_start, ...)	computes quantiles of a particular sweep
compute_2d_hist(field1, field2, field_name1, ...)	computes a 2D histogram of the data
compute_1d_stats(field1, field2)	returns statistics of data
compute_2d_stats(field1, field2, ...[, ...])	computes a 2D histogram and statistics of the data
compute_histogram(field, field_name[, ...])	computes histogram of the data
compute_histogram_sweep(field, ray_start, ...)	computes histogram of the data in a particular sweep
belongs_roi_indices(lat, lon, roi)	Get the indices of points that belong to roi in a list of points
compute_profile_stats(field, gate_altitude, ...)	Compute statistics of vertical profile
compute_directional_stats(field[, avg_type, ...])	Computes the mean or the median along one of the axis (ray or range)
project_to_vertical(data_in, data_height, ...)	Projects radar data to a regular vertical grid
quantiles_weighted(values[, weight_vector, ...])	Given a set of values and weights, compute the weighted quantile(s) and average.
ratio_bootstrapping(nominator, denominator)	Computes a set of samples obtained as sum(nominator)/sum(denominator) where the nominator and the denominator are randomly sampled with replacement.

pyrad.util.**belongs_roi_indices**(*lat*, *lon*, *roi*)

Get the indices of points that belong to roi in a list of points

Parameters

lat, lon [float arrays] latitudes and longitudes to check

roi [dict] Dictionary describing the region of interest

Returns

inds [array of ints] list of indices of points belonging to ROI

is_roi [str] Whether the list of points is within the region of interest. Can be 'All', 'None', 'Some'

pyrad.util.**compute_1d_stats**(*field1*, *field2*)

returns statistics of data

Parameters

field1, field2 [ndarray 1D] the two fields to compare

Returns

stats [dict] a dictionary with statistics

pyrad.util.**compute_2d_hist**(*field1*, *field2*, *field_name1*, *field_name2*, *step1=None*, *step2=None*)

computes a 2D histogram of the data

Parameters

field1, field2 [ndarray 2D] the radar fields

field_name1, field_name2 [str] field names

step1, step2 [float] size of the bins

Returns

H [float array 2D] The bi-dimensional histogram of samples x and y

xedges, yedges [float array] the bin edges along each dimension

`pyrad.util.`**`compute_2d_stats`**(*field1*, *field2*, *field_name1*, *field_name2*, *step1=None*, *step2=None*)

computes a 2D histogram and statistics of the data

Parameters

field1, field2 [ndarray 2D] the two fields

field_name1, field_nam2: str the name of the fields

step1, step2 [float] size of bin

Returns

hist_2d [array] the histogram

bin_edges1, bin_edges2 [float array] The bin edges

stats [dict] a dictionary with statistics

`pyrad.util.`**`compute_directional_stats`**(*field*, *avg_type='mean'*, *nvalid_min=1*, *axis=0*)

Computes the mean or the median along one of the axis (ray or range)

Parameters

field [ndarray] the radar field

avg_type :str the type of average: 'mean' or 'median'

nvalid_min [int] the minimum number of points to consider the stats valid. Default 1

axis [int] the axis along which to compute (0=ray, 1=range)

Returns

values [ndarray 1D] The resultant statistics

nvalid [ndarray 1D] The number of valid points used in the computation

`pyrad.util.`**`compute_histogram`**(*field*, *field_name*, *bin_edges=None*, *step=None*, *vmin=None*, *vmax=None*)

computes histogram of the data

Parameters

field [ndarray 2D] the radar field

field_name: str or none name of the field

bins_edges :ndarray 1D the bin edges

step [float] size of bin

vmin, vmax [float] The minimum and maximum value of the histogram

Returns

bin_edges [float array] interval of each bin

values [float array] values at each bin

pyrad.util.**compute_histogram_sweep**(*field*, *ray_start*, *ray_end*, *field_name*, *step=None*, *vmin=None*, *vmax=None*)
computes histogram of the data in a particular sweep

Parameters

field [ndarray 2D] the radar field

ray_start, ray_end [int] starting and ending ray indexes

field_name: str name of the field

step [float] size of bin

vmin, vmax [float] minimum and maximum values

Returns

bin_edges [float array] interval of each bin

values [float array] values at each bin

pyrad.util.**compute_profile_stats**(*field*, *gate_altitude*, *h_vec*, *h_res*, *quantity='quantiles'*, *quantiles=array([0.25, 0.5, 0.75])*, *nvalid_min=4*, *std_field=None*, *np_field=None*, *make_linear=False*, *include_nans=False*)
Compute statistics of vertical profile

Parameters

field [ndarray] the radar field

gate_altitude: ndarray the altitude at each radar gate [m MSL]

h_vec [1D ndarray] height vector [m MSL]

h_res [float] heigh resolution [m]

quantity [str] The quantity to compute. Can be ['quantiles', 'mode', 'regression_mean', 'mean']. If 'mean', the min, max, and average is computed.

quantiles [1D ndarray] the quantiles to compute

nvalid_min [int] the minimum number of points to consider the stats valid

std_field [ndarray] the standard deviation of the regression at each range gate

np_field [ndarray] the number of points used to compute the regression at each range gate

make_linear [Boolean] If true the data is transformed into linear coordinates before taking the mean

include_nans [Boolean] If true NaN will be considered as zeros

Returns

vals [ndarray 2D] The resultant statistics

val_valid [ndarray 1D] The number of points to compute the stats used at each height level

pyrad.util.**compute_quantiles**(*field*, *quantiles=None*)
computes quantiles

Parameters

field [ndarray 2D] the radar field

ray_start, ray_end [int] starting and ending ray indexes

quantiles: float array list of quantiles to compute

Returns

quantiles [float array] list of quantiles

values [float array] values at each quantile

`pyrad.util.`**`compute_quantiles_from_hist`**(*bin_centers*, *hist*, *quantiles=None*)
computes quantiles from histograms

Parameters

bin_centers [ndarray 1D] the bins

hist [ndarray 1D] the histogram

quantiles: float array list of quantiles to compute

Returns

quantiles [float array] list of quantiles

values [float array] values at each quantile

`pyrad.util.`**`compute_quantiles_sweep`**(*field*, *ray_start*, *ray_end*, *quantiles=None*)
computes quantiles of a particular sweep

Parameters

field [ndarray 2D] the radar field

ray_start, ray_end [int] starting and ending ray indexes

quantiles: float array list of quantiles to compute

Returns

quantiles [float array] list of quantiles

values [float array] values at each quantile

`pyrad.util.`**`create_sun_hits_field`**(*rad_el*, *rad_az*, *sun_el*, *sun_az*, *data*, *imgcfg*)
creates a sun hits field from the position and power of the sun hits

Parameters

rad_el, rad_az, sun_el, sun_az [ndarray 1D] azimuth and elevation of the radar and the sun respectively in degree

data [masked ndarray 1D] the sun hit data

imgcfg: dict a dictionary specifying the ranges and resolution of the field to create

Returns

field [masked ndarray 2D] the sun hit field

`pyrad.util.`**`create_sun_retrieval_field`**(*par*, *field_name*, *imgcfg*, *lant=0.0*)
creates a sun retrieval field from the retrieval parameters

Parameters

par [ndarray 1D] the 5 retrieval parameters

imgcfg: dict a dictionary specifying the ranges and resolution of the field to create

Returns

field [masked ndarray 2D] the sun retrieval field `pyrad.util.`**`find_colocated_indexes`**(*radar1*, *radar2*, *rad1_ele*, *rad1_azi*, *rad1_rng*, *rad2_ele*, *rad2_azi*, *rad2_rng*, *ele_tol=0.5*, *azi_tol=0.5*, *rng_tol=50.0*)

Given the theoretical elevation, azimuth and range of the co-located gates of two radars and a given tolerance returns the indices of the gates for the current radars

Parameters

radar1, radar2 [radar objects] the two radar objects

rad1_ele, rad1_azi, rad1_rng [array of floats] the radar coordinates of the radar1 gates

rad2_ele, rad2_azi, rad2_rng [array of floats] the radar coordinates of the radar2 gates

ele_tol, azi_tol [floats] azimuth and elevation angle tolerance [deg]

rng_tol [float] range Tolerance [m]

Returns

ind_ray_rad1, ind_rng_rad1, ind_ray_rad2, ind_rng_rad2 [array of ints] the ray and range indexes of each radar gate

`pyrad.util.`**`find_contiguous_times`**(*times*, *step=600*)

Given and array of ordered times, find those contiguous according to a maximum time step

Parameters

times [array of datetimes] The array of times

step [float] The time step [s]

Returns

start_times, end_times [array of date times] The start and end of each consecutive time period

`pyrad.util.`**`find_nearest_gate`**(*radar*, *lat*, *lon*, *latlon_tol=0.0005*)

Find the radar gate closest to a lat,lon point

Parameters

radar [radar object] the radar object

lat, lon [float] The position of the point

latlon_tol [float] The tolerance around this point

Returns

ind_ray, ind_rng [int] The ray and range index

azi, rng [float] the range and azimuth position of the gate

`pyrad.util.`**`find_neighbour_gates`**(*radar*, *azi*, *rng*, *delta_azi=None*, *delta_rng=None*)

Find the neighbouring gates within +-delta_azi and +-delta_rng

Parameters

radar [radar object] the radar object

azi, rng [float] The azimuth [deg] and range [m] of the central gate

delta_azi, delta_rng [float] The extend where to look for

Returns

inds_ray_aux, ind_rng_aux [int] The indices (ray, rng) of the neighbouring gates

pyrad.util.**find_ray_index**(*ele_vec*, *azi_vec*, *ele*, *azi*, *ele_tol=0.0*, *azi_tol=0.0*, *nearest='azi'*)
Find the ray index corresponding to a particular elevation and azimuth

Parameters

ele_vec, azi_vec [float arrays] The elevation and azimuth data arrays where to look for

ele, azi [floats] The elevation and azimuth to search

ele_tol, azi_tol [floats] Tolerances [deg]

nearest [str] criteria to define wich ray to keep if multiple rays are within tolerance. azi: nearest azimuth, ele: nearest elevation

Returns

ind_ray [int] The ray index

pyrad.util.**find_rng_index**(*rng_vec*, *rng*, *rng_tol=0.0*)
Find the range index corresponding to a particular range

Parameters

rng_vec [float array] The range data array where to look for

rng [float] The range to search

rng_tol [float] Tolerance [m]

Returns

ind_rng [int] The range index

pyrad.util.**get_ROI**(*radar*, *fieldname*, *sector*)
filter out any data outside the region of interest defined by sector

Parameters

radar [radar object] the radar object where the data is

fieldname [str] name of the field to filter

sector [dict] a dictionary defining the region of interest

Returns

roi_flag [ndarray] a field array with ones in gates that are in the Region of Interest

pyrad.util.**get_closest_solar_flux**(*hit_datetime_list*, *flux_datetime_list*, *flux_value_list*)
finds the solar flux measurement closest to the sun hit

Parameters

hit_datetime_list [datetime array] the date and time of the sun hit

flux_datetime_list [datetime array] the date and time of the solar flux measurement

flux_value_list: ndarray 1D the solar flux values

Returns

flux_datetime_closest_list [datetime array] the date and time of the solar flux measurement closest to sun hit

flux_value_closest_list [ndarray 1D] the solar flux values closest to the sun hit time

pyrad.util.**get_data_along_azi**(*radar*, *field_name*, *fix_ranges*, *fix_elevations*, *rng_tol=50.0*, *ang_tol=1.0*, *azi_start=None*, *azi_stop=None*)
Get data at particular (ranges, elevations)

Parameters

radar [radar object] the radar object where the data is

field_name [str] name of the field to filter

fix_ranges, fix_elevations: list of floats List of ranges [m], elevations [deg] couples

rng_tol [float] Tolerance between the nominal range and the radar range [m]

ang_tol [float] Tolerance between the nominal angle and the radar angle [deg]

azi_start, azi_stop: float Start and stop azimuth angle of the data [deg]

Returns

xvals [list of float arrays] The ranges of each rng, ele pair

yvals [list of float arrays] The values

valid_rng, valid_ele [float arrays] The rng, ele pairs

`pyrad.util.`**`get_data_along_ele`**(*radar*, *field_name*, *fix_ranges*, *fix_azimuths*, *rng_tol=50.0*, *ang_tol=1.0*, *ele_min=None*, *ele_max=None*)

Get data at particular (ranges, azimuths)

Parameters

radar [radar object] the radar object where the data is

field_name [str] name of the field to filter

fix_ranges, fix_azimuths: list of floats List of ranges [m], azimuths [deg] couples

rng_tol [float] Tolerance between the nominal range and the radar range [m]

ang_tol [float] Tolerance between the nominal angle and the radar angle [deg]

ele_min, ele_max: float Min and max elevation angle [deg]

Returns

xvals [list of float arrays] The ranges of each rng, ele pair

yvals [list of float arrays] The values

valid_rng, valid_ele [float arrays] The rng, ele pairs

`pyrad.util.`**`get_data_along_rng`**(*radar*, *field_name*, *fix_elevations*, *fix_azimuths*, *ang_tol=1.0*, *rmin=None*, *rmax=None*)

Get data at particular (azimuths, elevations)

Parameters

radar [radar object] the radar object where the data is

field_name [str] name of the field to filter

fix_elevations, fix_azimuths: list of floats List of elevations, azimuths couples [deg]

ang_tol [float] Tolerance between the nominal angle and the radar angle [deg]

rmin, rmax: float Min and Max range of the obtained data [m]

Returns

xvals [list of float arrays] The ranges of each azi, ele pair

yvals [list of float arrays] The values

valid_azi, valid_ele [float arrays] The azi, ele pairs `pyrad.util.get_fixed_rng_data(radar, field_names, fixed_rng, rng_tol=50.0, ele_min=None, ele_max=None, azi_min=None, azi_max=None)`

Creates a 2D-grid with (azi, ele) data at a fixed range

Parameters

radar [radar object] The radar object containing the data

field_name [str] The field name

fixed_rng [float] The fixed range [m]

rng_tol [float] The tolerance between the nominal range and the actual radar range [m]

ele_min, ele_max, azi_min, azi_max [float or None] The limits of the grid [deg]. If None the limits will be the limits of the radar volume

Returns

radar [radar object] The radar object containing only the desired data

`pyrad.util.get_range_bins_to_avg(rad1_rng, rad2_rng)`

Compares the resolution of two radars and determines if and which radar has to be averaged and the length of the averaging window

Parameters

rad1_rng [array] the range of radar 1

rad2_rng [datetime] the range of radar 2

Returns

avg_rad1, avg_rad2 [Boolean] Booleans specifying if the radar data has to be average in range

avg_rad_lim [array with two elements] the limits to the average (centered on each range gate)

`pyrad.util.get_target_elevations(radar_in)`

Gets RHI target elevations

Parameters

radar_in [Radar object] current radar object

Returns

target_elevations [1D-array] Azimuth angles

el_tol [float] azimuth tolerance

`pyrad.util.join_time_series(t1, val1, t2, val2, dropnan=False)`

joins time_series. Only of package pandas is available otherwise returns None.

Parameters

t1 [datetime array] time of first series

val1 [float array] value of first series

t2 [datetime array] time of second series

val2 [float array] value of second series

dropnan [boolean] if True remove NaN from the time series

Returns

t_out_vec [datetime array] the resultant date time after joining the series

val1_out_vec [float array] value of first series

val2_out_vec [float array] value of second series

pyrad.util.**project_to_vertical**(*data_in*, *data_height*, *grid_height*, *interp_kind='none'*, *fill_value=-9999.0*)

Projects radar data to a regular vertical grid

Parameters

data_in [ndarray 1D] the radar data to project

data_height [ndarray 1D] the height of each radar point

grid_height [ndarray 1D] the regular vertical grid to project to

interp_kind [str] The type of interpolation to use: 'none' or 'nearest'

fill_value [float] The fill value used for interpolation

Returns

data_out [ndarray 1D] The projected data

pyrad.util.**quantiles_weighted**(*values*, *weight_vector=None*, *quantiles=array([0.5])*, *weight_threshold=None*, *data_is_log=False*, *nvalid_min=3*)

Given a set of values and weights, compute the weighted quantile(s) and average.

Parameters

values [array of floats] Array containing the values. Can be 2-dimensional

weight_vector [array of floats or None] array containing the weights to apply. If None it will be an array of ones (uniform weight). If values is a 2D array it will be repeated for the second dimension

quantiles [array of floats] The quantiles to be computed

weight_threshold [float or None] If weight_threshold is set quantiles will be computed only if the total weight (sum of the weights of valid data) exceeds this threshold

data_is_log [Bool] If true the values will be considered to be in logarithmic scale and transformed into linear scale before computing the quantiles and average

nvalid_min [int] Minimum number of valid points to consider the computation valid

Returns

avg [float] the weighted average

quants [array of floats] an array containing the weighted quantiles in the same order as the quantiles vector

nvalid [int] Number of valid points in the computation of the statistics

pyrad.util.**rainfall_accumulation**(*t_in_vec*, *val_in_vec*, *cum_time=3600.0*, *base_time=0.0*, *dropnan=False*)

Computes the rainfall accumulation of a time series over a given period

Parameters

t_in_vec [datetime array] the input date and time array

val_in_vec [float array] the input values array [mm/h]

cum_time [int] accumulation time [s]

base_time [int] base time [s]

dropnan [boolean] if True remove NaN from the time series

Returns

t_out_vec [datetime array] the output date and time array

val_out_vec [float array] the output values array

np_vec [int array] the number of samples at each period

`pyrad.util.`**`ratio_bootstrapping`**(*nominator*, *denominator*, *nsamples=1000*)

Computes a set of samples obtained as sum(nominator)/sum(denominator) where the nominator and the denominator are randomly sampled with replacement.

Parameters

nominator, denominator [1D array] The data points in the nominator and the denominator. Nominator and denominator are not independent, i.e. data point i in the nominator is linked to data point i in the denominator

nsamples [int] Number of iteration, i.e. number of samples desired

Returns

samples [1D array] the resultant samples

`pyrad.util.`**`time_avg_range`**(*timeinfo*, *avg_starttime*, *avg_endtime*, *period*)

finds the new start and end time of an averaging

Parameters

timeinfo [datetime] the current volume time

avg_starttime [datetime] the current average start time

avg_endtime: datetime the current average end time

period: float the averaging period

Returns

new_starttime [datetime] the new average start time

new_endtime [datetime] the new average end time

`pyrad.util.`**`time_series_statistics`**(*t_in_vec*, *val_in_vec*, *avg_time=3600*, *base_time=1800*, *method='mean'*, *dropnan=False*)

Computes statistics over a time-averaged series. Only of package pandas is available otherwise returns None

Parameters

t_in_vec [datetime array] the input date and time array

val_in_vec [float array] the input values array

avg_time [int] averaging time [s]

base_time [int] base time [s]

method [str] statistical method

dropnan [boolean] if True remove NaN from the time series

Returns

t_out_vec [datetime array] the output date and time array

val_out_vec [float array] the output values array

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p