Py-ART documentation

Release 1.0.0.dev-814d4ee

Py-ART developers

January 09, 2014

Contents

Welcome this is the documentation for the Python ARM Radar Toolkit (Py-ART).

CHAPTER 1

Latest Release

1.1 User Reference Manual

Release 1.0.dev

Date January 09, 2014

This guide provides details on all public functions, modules and classes included in Py-ART which a typical user will use on a regular basis.

For a more detailed listing of all functions, module and classes (both public and private) aimed at developers please see the *Developer Reference Manual*.

1.1.1 Input and output (`pyart.io`)

Py-ART has modules, classes and functions which are able to read data from and write data to a number of file formats.

`read`(filename[, use_rsl])	Read a radar file and return a radar object.
`read_rsl`	
`read_mdv`(filename[, field_names, ...])	Read a MDV file.
`read_sigmet`(filename[, field_names, ...])	Read a Sigmet (IRIS) product file.
`read_cfradial`(filename[, field_names, ...])	Read a Cfradial netCDF file.
`read_nexrad_archive`(filename[, field_names, ...])	Read a NEXRAD Level 2 Archive file.
`read_nexrad_cdm`(filename[, field_names, ...])	Read a Common Data Model (CDM) NEXRAD Level 2 file.
`read_gamic`	
`write_cfradial`(filename, radar[, format, ...])	Write a Radar object to a CF/Radial compliant netCDF file.
`read_grid`(filename)	Read a netCDF grid file
`write_grid`(filename, grid[, format])	Write a Grid object to a CF-1.5 and ARM standard netcdf file
`is_vpt`(radar[, offset])	Determine if a Radar appears to be a vertical pointing scan.
`to_vpt`(radar[, single_scan])	Convert an existing Radar object to represent a vertical pointing scan.
`Radar`(time, _range, fields, metadata, ...[, ...])	A class for storing antenna coordinate radar data.
`Grid`(fields, axes, metadata)	An object for holding gridded Radar data.

pyart.io.read

`pyart.io.`**`read`**(*filename*, *use_rsl=False*, ***kwargs*)

Read a radar file and return a radar object.

Additional parameters are passed to the underlying read_* function.

Parameters **filename** : str

Name of radar file to read

use_rsl : bool

True to use the TRMM RSL library for reading if RSL is installed.

Returns **radar** : Radar

Radar object. A TypeError is raised if the format cannot be determined.

Other Parameters **field_names** : dict, optional

Dictionary mapping file data type names to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the metadata configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the metadata configuration file will be used.

file_field_names : bool, optional

True to use the file data type names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

pyart.io.read_mdv

pyart.io.**read_mdv**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*)

Read a MDV file.

Parameters **filename** : str

Name of MDV file to read or file-like object pointing to the beginning of such a file.

field_names : dict, optional

Dictionary mapping MDV data type names to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the Py-ART configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the Py-ART configuration file will be used.

file_field_names : bool, optional

True to use the MDV data type names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

Returns **radar** : Radar

Radar object containing data from MDV file.

Notes

Currently this function can only read polar MDV files which are gzipped. Support for cartesian and non-gzipped file are planned.

pyart.io.read_sigmet

pyart.io.**read_sigmet**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *time_ordered='none'*, *debug=False*)

Read a Sigmet (IRIS) product file.

Parameters **filename** : str

Name of Sigmet (IRIS) product file to read or file-like object pointing to the beginning of such a file.

field_names : dict, optional

Dictionary mapping Sigmet data type names to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the metadata configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the metadata configuration file will be used.

file_field_names : bool, optional

True to use the Sigmet data type names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

time_ordered : 'full', 'none' or 'roll'.

Parameter controlling the time ordering of the data. The default, 'none' keep the data ordered in the same manner as it appears in the Sigmet file. 'roll' will attempt to time order the data within each sweep by rolling the earliest collected ray to be the beginning.

Sequential ordering of the rays is maintained but strict time increasing order is not guaranteed. 'full' will place data within each sweep in a strictly time increasing order, but the rays will likely become non-sequential. The 'full' option is not recommended unless strict time increasing order is required.

debug : bool, optional

Returns **radar** : Radar

Radar object

pyart.io.read_cfradial

pyart.io.**read_cfradial**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*)

Read a Cfradial netCDF file.

Parameters **filename** : str

Name of CF/Radial netCDF file to read data from.

field_names : dict, optional

Dictionary mapping field names in the file names to radar field names. Unlike other read functions, fields not in this dictionary or having a value of None are still included in the radar.fields dictionary, to exclude them use the *exclude_fields* parameter. Fields which are mapped by this dictionary will be renamed from key to value.

additional_metadata : dict of dicts, optional

This parameter is not used, it is included for uniformity.

file_field_names : bool, optional

True to force the use of the field names from the file in which case the *field_names* parameter is ignored. False will use to *field_names* parameter to rename fields.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

Returns **radar** : Radar

Radar object.

Notes

This function has not been tested on "stream" Cfradial files.

pyart.io.read_nexrad_archive

pyart.io.**read_nexrad_archive**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *bzip=None*)

Read a NEXRAD Level 2 Archive file.

Parameters **filename** : str

Filename of NEXRAD Level 2 Archive file. The files hosted by at the NOAA National Climate Data Center [R16] as well as on the UCAR THREDDS Data Server [R17] have been tested. Other NEXRAD Level 2 Archive files may or may not work. Message type 1 file at not yet supported, only message type 31.

field_names : dict, optional

Dictionary mapping NEXRAD moments to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the metadata configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the metadata configuration file will be used.

file_field_names : bool, optional

True to use the NEXRAD field names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

bzip : bool or None

True if the file is compressed as a bzip2 file, False otherwise. None will examine the filename for a bzip extension.

Returns **radar** : Radar

Radar object containing all moments and sweeps/cuts in the volume. Gates not collected are masked in the field data.

References

[R16], [R17]

pyart.io.read_nexrad_cdm

pyart.io.**read_nexrad_cdm**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*)

Read a Common Data Model (CDM) NEXRAD Level 2 file.

Parameters **filename** : str

File name or URL of a Common Data Model (CDM) NEXRAD Level 2 file. File of in this format can be created using the NetCDF Java Library tools [R18]. A URL of a OPeNDAP file on the UCAR THREDDS Data Server [R19] is also accepted the netCDF4 library has been compiled with OPeNDAP support.

field_names : dict, optional

Dictionary mapping NEXRAD moments to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the metadata configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the metadata configuration file will be used.

file_field_names : bool, optional

True to use the NEXRAD field names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

Returns **radar** : Radar

Radar object containing all moments and sweeps/cuts in the volume. Gates not collected are masked in the field data.

References

[R18], [R19]

pyart.io.write_cfradial

pyart.io.**write_cfradial**(*filename*, *radar*, *format='NETCDF4'*, *time_reference=False*)

Write a Radar object to a CF/Radial compliant netCDF file.

Parameters **filename** : str

Filename to create.

radar : Radar

Radar object.

format : str, optional

NetCDF format, one of 'NETCDF4', 'NETCDF4_CLASSIC', 'NETCDF3_CLASSIC' or 'NETCDF3_64BIT'. See netCDF4 documentation for details.

time_reference : bool

True to include a time_reference variable, False will not include this variable.

pyart.io.read_grid

pyart.io.**read_grid**(*filename*)

Read a netCDF grid file

Parameters **filename** : str

Filename of netCDF grid file to read

Returns **grid** : Grid

Grid object containing Grid data.

pyart.io.write_grid

pyart.io.**write_grid**(*filename*, *grid*, *format='NETCDF4'*)

Write a Grid object to a CF-1.5 and ARM standard netcdf file

Parameters **filename** : str

Filename to save grid to.

grid : Grid

Grid object to write.

format : str, optional

NetCDF format, one of 'NETCDF4', 'NETCDF4_CLASSIC', 'NETCDF3_CLASSIC' or 'NETCDF3_64BIT'. See netCDF4 documentation for details.

pyart.io.is_vpt

pyart.io.**is_vpt**(*radar*, *offset=0.5*)

Determine if a Radar appears to be a vertical pointing scan.

This function only verifies that the object is a vertical pointing scan, use the to_vpt function to convert the radar to a vpt scan if this function returns True.

Parameters **radar** : Radar

Radar object to determine if

offset : float

Maximum offset of the elevation from 90 degrees to still consider to be vertically pointing.

Returns **flag** : bool

True if the radar appear to be verticle pointing, False if not.

pyart.io.to_vpt

pyart.io.**to_vpt**(*radar*, *single_scan=True*)

Convert an existing Radar object to represent a vertical pointing scan.

This function does not verify that the Radar object contains a vertical pointing scan. To perform such a check use is_vpt.

Parameters **radar** : Radar

Mislabeled vertical pointing scan Radar object to convert to be properly labeled. This object is converted in place, no copy of the existing data is made.

single_scan : bool, optional

True to convert the volume to a single scan, any azimuth angle data is lost. False will convert the scan to contain the same number of scans as rays, azimuth angles are retained.

1.1.2 Radar Corrections (`pyart.correct`)

Py-ART has the ability to perform a number of common corrections on radar moments and data.

`dealias_fourdd`	
`calculate_attenuation`(radar, z_offset[, ...])	Calculate the attenuation from a polarimetric radar using Z-PHI method.
`phase_proc_lp`(radar, offset[, debug, ...])	Phase process using a LP method [1].
`find_time_in_interp_sonde`	

pyart.correct.calculate_attenuation

pyart.correct.**calculate_attenuation**(*radar*, *z_offset*, *debug=False*, *doc=15*, *fzl=4000.0*, *rhv_min=0.8*, *ncp_min=0.5*, *a_coef=0.06*, *beta=0.8*, *refl_field=None*, *ncp_field=None*, *rhv_field=None*, *phidp_field=None*, *spec_at_field=None*, *corr_refl_field=None*)

Calculate the attenuation from a polarimetric radar using Z-PHI method.

Parameters **radar** : Radar

Radar object to use for attenuation calculations. Must have copol_coeff, norm_coherent_power, proc_dp_phase_shift, reflectivity_horizontal fields.

z_offset : float

Horizontal reflectivity offset in dBZ.

debug : bool

True to print debugging information, False supressed this printing.

Returns **spec_at** : dict

Field dictionary containing the specific attenuation.

cor_z : dict

Field dictionary containing the corrected reflectivity.

Other Parameters **doc** : float

Number of gates at the end of each ray to to remove from the calculation.

fzl : float

Freezing layer, gates above this point are not included in the correction.

rhv_min : float

Minimum copol_coeff value to consider valid.

ncp_min : float

Minimum norm_coherent_power to consider valid.

a_coef : float

A coefficient in attenuation calculation.

beta : float

Beta parameter in attenuation calculation.

refl_field, ncp_field, rhv_field, phidp_field : str

Field names within the radar object which represent the horizonal reflectivity, normal coherent power, the copolar coefficient, and the differential phase shift. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

spec_at_field, corr_refl_field : str

Names of the specific attenuation and the corrected reflectivity fields that will be used to fill in the metadata for the returned fields. A value of None for any of these parameters will use the default field names as defined in the Py-ART configuration file.

References

Gu et al. Polarimetric Attenuation Correction in Heavy Rain at C Band, JAMC, 2011, 50, 39-58.

pyart.correct.phase_proc_lp

pyart.correct.**phase_proc_lp**(*radar*, *offset*, *debug=False*, *self_const=60000.0*, *low_z=10.0*, *high_z=53.0*, *min_phidp=0.01*, *min_ncp=0.5*, *min_rhv=0.8*, *fzl=4000.0*, *sys_phase=0.0*, *overide_sys_phase=False*, *nowrap=None*, *really_verbose=False*, *LP_solver='pyglpk'*, *refl_field=None*, *ncp_field=None*, *rhv_field=None*, *phidp_field=None*, *kdp_field=None*, *unf_field=None*, *window_len=35*, *proc=1*)

Phase process using a LP method [1].

Parameters **radar** : Radar

Input radar.

offset : float

Reflectivity offset in dBz.

debug : bool, optional

True to print debugging information.

self_const : float, optional

Self consistency factor.

low_z : float

Low limit for reflectivity. Reflectivity below this value is set to this limit.

high_z : float

High limit for reflectivity. Reflectivity above this value is set to this limit.

min_phidp : float

Minimum Phi differential phase.

min_ncp : float

Minimum normal coherent power.

min_rhv : float

Minimum copolar coefficient.

fzl : :

Maximum altitude.

sys_phase : float

System phase in degrees.

overide_sys_phase: bool. :

True to use *sys_phase* as the system phase. False will calculate a value automatically.

nowrap : int or None.

Gate number to begin phase unwrapping. None will unwrap all phases.

really_verbose : bool

True to print LPX messaging. False to suppress.

LP_solver : 'pyglpk' or 'cvxopt', 'cylp', or 'cylp_mp'

Module to use to solve LP problem.

refl_field, ncp_field, rhv_field, phidp_field, kdp_field: str :

Name of field in radar which contains the horizonal reflectivity, normal coherent power, copolar coefficient, differential phase shift, and differential phase. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

unf_field : str

Name of field which will be added to the radar object which will contain the unfolded differential phase. Metadata for this field will be taken from the phidp_field. A value of None will use the default field name as defined in the Py-ART configuration file.

window_len : int

Length of Sobel window applied to PhiDP field when prior to calculating KDP.

proc : int

Number of worker processes, only used when *LP_solver* is 'cylp_mp'.

Returns **reproc_phase** : dict

Field dictionary containing processed differential phase shifts.

sob_kdp : dict

Field dictionary containing recalculated differential phases.

References

[1] Giangrande, S.E., R. McGraw, and L. Lei. An Application of Linear Programming to Polarimetric Radar Differential Phase Processing. J. Atmos. and Oceanic Tech, 2013, 30, 1716.

1.1.3 Graphing (`pyart.graph`)

RadarDisplay(radar[, shift])	A display object for creating plots from data in a radar object.
GridMapDisplay	
MdvDisplay(mdvfile)	A display object for creating plots from data in a MdvFile objects.
RslDisplay	
CFRadialDisplay(dataset[, shift])	A display object for creating plots from data in NetCDF4 Dataset objects.

1.1.4 Mapping (pyart.map)

Py-ART has a robust function for mapping radar data from the collected radar coordinates to Cartesian coordinates.

grid_from_radars(radars, grid_shape, ...)	Map one or more radars to a Cartesian grid returning a Grid object.
map_to_grid(radars, grid_shape, grid_limits)	Map one or more radars to a Cartesian grid.
example_roi_func_constant(zg, yg, xg)	Example RoI function which returns a constant radius.
example_roi_func_dist(zg, yg, xg)	Example RoI function which returns a radius which grows with distance.
example_roi_func_dist_beam(zg, yg, xg)	Example RoI function which returns a radius which grows with distance

pyart.map.grid_from_radars

pyart.map.**grid_from_radars**(*radars*, *grid_shape*, *grid_limits*, ***kwargs*)

Map one or more radars to a Cartesian grid returning a Grid object.

Additional arguments are passed to map_to_grid

Parameters **radars** : tuple of Radar objects.

Radar objects which will be mapped to the Cartesian grid.

grid_shape : 3-tuple of floats

Number of points in the grid (z, y, x).

grid_limits : 3-tuple of 2-tuples

Minimum and maximum grid location (inclusive) in meters for the z, x, y coordinates.

Returns **grid** : Grid

A pyart.io.Grid object containing the gridded radar data.

See also:

map_to_grid Map to grid and return a dictionary of radar fields

pyart.map.map_to_grid

pyart.map.**map_to_grid**(*radars*, *grid_shape*, *grid_limits*, *grid_origin=None*, *fields=None*, *refl_filter_flag=True*, *refl_field=None*, *max_refl=None*, *map_roi=True*, *weighting_function='Barnes'*, *toa=17000.0*, *copy_field_data=True*, *algorithm='kd_tree'*, *leafsize=10.0*, *roi_func='dist_beam'*, *constant_roi=500.0*, *z_factor=0.05*, *xy_factor=0.02*, *min_radius=500.0*, *h_factor=1.0*, *nb=1.5*, *bsp=1.0*)

Map one or more radars to a Cartesian grid.

Generate a Cartesian grid of points for the requested fields from the collected points from one or more radars. The field value for a grid point is found by interpolating from the collected points within a given radius of

influence and weighting these nearby points according to their distance from the grid points. Collected points are filtered according to a number of criteria so that undesired points are not included in the interpolation.

Parameters **radars** : tuple of Radar objects.

Radar objects which will be mapped to the Cartesian grid.

grid_shape : 3-tuple of floats

Number of points in the grid (z, y, x).

grid_limits : 3-tuple of 2-tuples

Minimum and maximum grid location (inclusive) in meters for the z, y, x coordinates.

grid_origin : (float, float) or None

Latitude and longitude of grid origin. None sets the origin to the location of the first radar.

fields : list or None

List of fields within the radar objects which will be mapped to the cartesian grid. None, the default, will map the fields which are present in all the radar objects.

refl_filter_flag : bool

True to filter the collected points based on the reflectivity field. False to perform no filtering. Gates where the reflectivity field, specified by the *refl_field* parameter, is not-finited, masked or has a value above the *max_refl* parameter are excluded from the grid interpolation.

refl_field : str

Name of the field which will be used to filter the collected points. A value of None will use the default field name as defined in the Py-ART configuration file.

max_refl : float

Maximum allowable reflectivity. Points in the *refl_field* which are above is value are not included in the interpolation. None will include skip this filtering.

roi_func : str or function

Radius of influence function. A functions which takes an z, y, x grid location, in meters, and returns a radius (in meters) within which all collected points will be included in the weighting for that grid points. Examples can be found in the `example_roi_func_constant`, `example_roi_func_dist`, and `example_roi_func_dist_beam`. Alternatively the following strings can use to specify a built in radius of influence function:

- constant: constant radius of influence.
- dist: radius grows with the distance from each radar.
- dist_beam: radius grows with the distance from each radar and parameter are based of virtual beam sizes.

The parameters which control these functions are listed in the *Other Parameters* section below.

map_roi : bool

True to include a radius of influence field in the returned dictionary under the 'ROI' key. This is the value of roi_func at all grid points.

weighting_function : 'Barnes' or 'Cressman'

Functions used to weight nearby collected points when interpolating a grid point.

toa : float

Top of atmosphere in meters. Collected points above this height are not included in the interpolation.

Returns **grids** : dict

Dictionary of mapped fields. The keysof the dictionary are given by parameter fields. Each elements is a *grid_size* float64 array containing the interpolated grid for that field.

Other Parameters **constant_roi** : float

Radius of influence parameter for the built in 'constant' function. This parameter is the constant radius in meter for all grid points. This parameter is only used when *roi_func* is *constant*.

z_factor, xy_factor, min_radius : float

Radius of influence parameters for the built in 'dist' function. The parameter correspond to the radius size increase, in meters, per meter increase in the z-dimension from the nearest radar, the same foreach meteter in the xy-distance from the nearest radar, and the minimum radius of influence in meters. These parameters are only used when *roi_func* is 'dist'.

h_factor, nb, bsp, min_radius : float

Radius of influence parameters for the built in 'dist_beam' function. The parameter correspond to the height scaling, virtual beam width, virtual beam spacing, and minimum radius of influence. These parameters are only used when *roi_func* is 'dist_mean'.

copy_field_data : bool

True to copy the data within the radar fields for faster gridding, the dtype for all fields in the grid will be float64. False will not copy the data which preserves the dtype of the fields in the grid, may use less memory but results in significantly slower gridding times. When False gates which are masked in a particular field but are not masked in the *refl_field* field will still be included in the interpolation. This can be prevented by setting this parameter to True or by gridding each field individually setting the *refl_field* parameter and the *fields* parameter to the field in question. It is recommended to set this parameter to True.

algorithm : 'kd_tree' or 'ball_tree'

Algorithms to use for finding the nearest neighbors. 'kd_tree' tends to be faster. This value should only effects the speed of the gridding, not the results.

leafsize : int

Leaf size passed to the neighbor lookup tree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. This value should only effect the speed of the gridding, not the results.

See also:

`grid_from_radars` Map to grid and return a Grid object.

pyart.map.example_roi_func_constant

pyart.map.**example_roi_func_constant**(*zg*, *yg*, *xg*)
Example RoI function which returns a constant radius.

Parameters **zg, yg, xg** : float

Distance from the grid center in meters for the x, y and z axes.

Returns **roi** : float

Radius of influence in meters

pyart.map.example_roi_func_dist

pyart.map.**example_roi_func_dist**(*zg*, *yg*, *xg*)
Example RoI function which returns a radius which grows with distance.

Parameters **zg, yg, xg** : float

Distance from the grid center in meters for the x, y and z axes.

Returns **roi** : float

pyart.map.example_roi_func_dist_beam

pyart.map.**example_roi_func_dist_beam**(*zg*, *yg*, *xg*)
Example RoI function which returns a radius which grows with distance and whose parameters are based on virtual beam size.

Parameters **zg, yg, xg** : float

Distance from the grid center in meters for the x, y and z axes.

Returns **roi** : float

1.1.5 Testing Utilities (pyart.testing)

Py-ART comes with a number of utilities helpful when writing and running unit tests.

make_empty_ppi_radar(ngates, rays_per_sweep, ...)	Return an Radar object, representing a PPI scan.
make_target_radar()	Return a PPI radar with a target like reflectivity field.
make_single_ray_radar()	Return a PPI radar with a single ray taken from a ARM C-SAPR Radar
make_velocity_aliased_radar()	Return a PPI radar with a target like reflectivity field.
make_empty_grid(grid_shape, grid_limits)	Make an empty grid object without any fields or metadata.
make_target_grid()	Make a sample Grid with a rectangular target.

pyart.testing.make_empty_ppi_radar

pyart.testing.**make_empty_ppi_radar**(*ngates*, *rays_per_sweep*, *nsweeps*)
Return an Radar object, representing a PPI scan.

Parameters **ngates** : int

Number of gates per ray.

rays_per_sweep : int

Number of rays in each PPI sweep.

nsweeps : int

Number of sweeps.

Returns **radar** : Radar

Radar object with no fields, other parameters are set to default values.

pyart.testing.make_target_radar

pyart.testing.**make_target_radar**()
Return a PPI radar with a target like reflectivity field.

pyart.testing.make_single_ray_radar

pyart.testing.**make_single_ray_radar**()
Return a PPI radar with a single ray taken from a ARM C-SAPR Radar

Radar object returned has 'reflectivity_horizontal', 'norm_coherent_power', 'copol_coeff', 'dp_phase_shift', and 'diff_phase' fields with no metadata but a 'data' key. This radar is used for unit tests in correct modules.

pyart.testing.make_velocity_aliased_radar

pyart.testing.**make_velocity_aliased_radar**()
Return a PPI radar with a target like reflectivity field.

pyart.testing.make_empty_grid

pyart.testing.**make_empty_grid**(*grid_shape*, *grid_limits*)
Make an empty grid object without any fields or metadata.

Parameters **grid_shape** : 3-tuple of floats

Number of points in the grid (x, y, z).

grid_limits : 3-tuple of 2-tuples

Minimum and maximum grid location (inclusive) in meters for the x, y, z coordinates.

Returns **grid** : Grid

Empty Grid object, centered near the ARM SGP site (Oklahoma).

pyart.testing.make_target_grid

pyart.testing.**make_target_grid**()
Make a sample Grid with a rectangular target.

Packages level functions:

load_config([filename])	Load a Py-ART configuration from a config file.
test([verbose])	This would invoke the Py-ART test suite, but nose couldn't be

1.1.6 pyart.load_config

pyart.**load_config**(*filename=None*)

Load a Py-ART configuration from a config file.

The default values for a number of Py-ART parameters and metadata is controlled by a single Python configuration file. An self-descriping example of this file can be found in the Py-ART source directory named **default_config.py**. These defaults can modified by setting the environmental variable *PYART_CONFIG* to point to a new configuration file. If this variable is not set then the settings contained in the **default_config.py** file are used.

The code the configuration file is executed as-is with full permission, this may present a security issue, do not load un-trusted configuration files.

The recommended method for changing these defaults is for users to copy this file into their home directory, rename it to .pyart_config.py, make any changes, and adjust their login scripts to set the PYART_CONFIG environmental variable to point to .pyart_config.py in their home directory.

Py-ART's configuration can also be modified within a script or shell session using this function, the modification will last until a the end of the script/session or until a new configuration is loaded.

Parameters **filename** : str

Filename of configuration file. If None the default configuration file is loaded from the Py-ART source code directory.

1.1.7 pyart.test

pyart.**test**(*verbose=False*)

This would invoke the Py-ART test suite, but nose couldn't be imported so the test suite can not run.

1.2 Developer Reference Manual

Release 1.0.dev

Date January 09, 2014

The intended audience of this guide is developers who use Py-ART, for a more generate introduction to Py-ART aimed at users see the *User Reference Manual*.

This guide provides for all functions modules, and classes within Py-ART, both those in the public API and private functions, modules and classes. Documentation is broken down by directory and module.

1.2.1 pyart.io

Input/Output routines.

pyart.io.auto_read

Automatic reading of radar files by detecting format.

read(filename[, use_rsl])	Read a radar file and return a radar object.
determine_filetype(filename)	Return the filetype of a given file by examining the first few bytes.

pyart.io.auto_read.read

pyart.io.auto_read.**read**(*filename*, *use_rsl=False*, ***kwargs*)

Read a radar file and return a radar object.

Additional parameters are passed to the underlying read_* function.

Parameters **filename** : str

Name of radar file to read

use_rsl : bool

True to use the TRMM RSL library for reading if RSL is installed.

Returns **radar** : Radar

Radar object. A TypeError is raised if the format cannot be determined.

Other Parameters **field_names** : dict, optional

Dictionary mapping file data type names to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the metadata configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the metadata configuration file will be used.

file_field_names : bool, optional

True to use the file data type names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

pyart.io.auto_read.determine_filetype

pyart.io.auto_read.**determine_filetype**(*filename*)

Return the filetype of a given file by examining the first few bytes.

The following filetypes are detected:

- 'MDV'
- 'NETCDF3'
- 'NETCDF4'
- 'WSR88D'
- 'UF'
- 'HDF4'
- 'RSL'

- 'DORAD'
- 'SIGMET'
- 'BZ2'
- 'UNKNOWN'

Parameters **filename** : str

Name of file to examine.

Returns **filetype** : str

Type of file.

pyart.io.common

Input/output routines common to many file formats.

dms_to_d(dms)	Degrees, minutes, seconds to degrees
stringarray_to_chararray(arr[, numchars])	Convert an string array to a character array with one extra dimension.
radar_coords_to_cart(rng, az, ele[, debug])	Calculate Cartesian coordinate from radar coordinates
make_time_unit_str(dtobj)	Return a time unit string from a datetime object.

pyart.io.common.dms_to_d

pyart.io.common.**dms_to_d**(*dms*)
Degrees, minutes, seconds to degrees

pyart.io.common.stringarray_to_chararray

pyart.io.common.**stringarray_to_chararray**(*arr*, *numchars=None*)
Convert an string array to a character array with one extra dimension.

Parameters **arr** : array

Array with numpy dtype 'SN', where N is the number of characters in the string.

numchars : int

Number of characters used to represent the string. If numchar > N the results will be padded on the right with blanks. The default, None will use N.

Returns **chararr** : array

Array with dtype 'S1' and shape = arr.shape + (numchars,).

pyart.io.common.radar_coords_to_cart

pyart.io.common.**radar_coords_to_cart**(*rng*, *az*, *ele*, *debug=False*)
Calculate Cartesian coordinate from radar coordinates

Parameters **rng** : array

Distances to the center of the radar gates (bins) in kilometers.

az : array

Azimuth angle of the radar in degrees.

ele : array

Elevation angle of the radar in degrees.

Returns **x, y, z** : array

Cartesian coordinates in meters from the radar.

Notes

The calculation for Cartesian coordinate is adapted from equations 2.28(b) and 2.28(c) of Doviak and Zrnic [R2] assuming a standard atmosphere (4/3 Earth's radius model).

$$z = \sqrt{r^2 + R^2 + r * R * sin(\theta_e)} - R$$
$$s = R * arcsin(\frac{r * cos(\theta_e)}{R + z})$$
$$x = s * sin(\theta_a)$$
$$y = s * cos(\theta_a)$$

Where r is the distance from the radar to the center of the gate, θ_a is the azimuth angle, θ_e is the elevation angle, s is the arc length, and R is the effective radius of the earth, taken to be 4/3 the mean radius of earth (6371 km).

References

[R2]

pyart.io.common.make_time_unit_str

pyart.io.common.**make_time_unit_str**(*dtobj*)
Return a time unit string from a datetime object.

pyart.io.cfradial

Utilities for reading CF/Radial files.

read_cfradial(filename[, field_names, ...])	Read a Cfradial netCDF file.
write_cfradial(filename, radar[, format, ...])	Write a Radar object to a CF/Radial compliant netCDF file.
_find_all_meta_group_vars(ncvars, ...)	Return a list of all variables which are in a given meta_group.
_ncvar_to_dict(ncvar)	Convert a NetCDF Dataset variable to a dictionary.
_stream_ncvar_to_dict(ncvar, sweeps, sweepe, ...)	Convert a Stream NetCDF Dataset variable to a dict.
_stream_to_2d(data, sweeps, sweepe, ray_len, ...)	Convert a 1D stream to a 2D array.
_create_ncvar(dic, dataset, name, dimensions)	Create and fill a Variable in a netCDF Dataset object.

pyart.io.cfradial.read_cfradial

pyart.io.cfradial.**read_cfradial**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*)

Read a Cfradial netCDF file.

Parameters **filename** : str

Name of CF/Radial netCDF file to read data from.

field_names : dict, optional

Dictionary mapping field names in the file names to radar field names. Unlike other read functions, fields not in this dictionary or having a value of None are still included in the radar.fields dictionary, to exclude them use the *exclude_fields* parameter. Fields which are mapped by this dictionary will be renamed from key to value.

additional_metadata : dict of dicts, optional

This parameter is not used, it is included for uniformity.

file_field_names : bool, optional

True to force the use of the field names from the file in which case the *field_names* parameter is ignored. False will use to *field_names* parameter to rename fields.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

Returns **radar** : Radar

Radar object.

Notes

This function has not been tested on "stream" Cfradial files.

pyart.io.cfradial.write_cfradial

pyart.io.cfradial.**write_cfradial**(*filename*, *radar*, *format='NETCDF4'*, *time_reference=False*)

Write a Radar object to a CF/Radial compliant netCDF file.

Parameters **filename** : str

Filename to create.

radar : Radar

Radar object.

format : str, optional

NetCDF format, one of 'NETCDF4', 'NETCDF4_CLASSIC', 'NETCDF3_CLASSIC' or 'NETCDF3_64BIT'. See netCDF4 documentation for details.

time_reference : bool

True to include a time_reference variable, False will not include this variable.

pyart.io.cfradial._find_all_meta_group_vars

pyart.io.cfradial.**_find_all_meta_group_vars**(*ncvars*, *meta_group_name*)
Return a list of all variables which are in a given meta_group.

pyart.io.cfradial._ncvar_to_dict

pyart.io.cfradial.**_ncvar_to_dict**(*ncvar*)
Convert a NetCDF Dataset variable to a dictionary.

pyart.io.cfradial._stream_ncvar_to_dict

pyart.io.cfradial.**_stream_ncvar_to_dict**(*ncvar*, *sweeps*, *sweepe*, *ray_len*, *maxgates*, *nrays*, *ray_start_index*)
Convert a Stream NetCDF Dataset variable to a dict.

pyart.io.cfradial._stream_to_2d

pyart.io.cfradial.**_stream_to_2d**(*data*, *sweeps*, *sweepe*, *ray_len*, *maxgates*, *nrays*, *ray_start_index*)
Convert a 1D stream to a 2D array.

pyart.io.cfradial._create_ncvar

pyart.io.cfradial.**_create_ncvar**(*dic*, *dataset*, *name*, *dimensions*)
Create and fill a Variable in a netCDF Dataset object.

Parameters **dic** : dict

Radar dictionary to containing variable data and meta-data

dataset : Dataset

NetCDF dataset to create variable in.

name : str

Name of variable to create.

dimension : tuple of str

Dimension of variable.

pyart.io.grid

Reading and writing Grid objects.

read_grid(filename)	Read a netCDF grid file
write_grid(filename, grid[, format])	Write a Grid object to a CF-1.5 and ARM standard netcdf file
_read_grid_cf(filename)	Read a CF compliant netCDF file containing a grid.
_read_grid_wrf(filename)	Read a WRF netCDF file containing a grid.

pyart.io.grid.read_grid

pyart.io.grid.**read_grid**(*filename*)
Read a netCDF grid file

Parameters **filename** : str

Filename of netCDF grid file to read

Returns **grid** : Grid

Grid object containing Grid data.

pyart.io.grid.write_grid

pyart.io.grid.**write_grid**(*filename*, *grid*, *format='NETCDF4'*)
Write a Grid object to a CF-1.5 and ARM standard netcdf file

Parameters **filename** : str

Filename to save grid to.

grid : Grid

Grid object to write.

format : str, optional

NetCDF format, one of 'NETCDF4', 'NETCDF4_CLASSIC', 'NETCDF3_CLASSIC' or 'NETCDF3_64BIT'. See netCDF4 documentation for details.

pyart.io.grid._read_grid_cf

pyart.io.grid.**_read_grid_cf**(*filename*)
Read a CF compliant netCDF file containing a grid.

Parameters **filename** : str

Filename of the netCDF file.

Returns **grid** : Grid

Frid object containing data.

Notes

This function does only the most basic variable checking. The resulting Grid object is most likely not writable.

pyart.io.grid._read_grid_wrf

pyart.io.grid.**_read_grid_wrf**(*filename*)
Read a WRF netCDF file containing a grid.

Parameters **filename** : str

Filename of the WRF netCDF file.

Returns **grid** : Grid

Grid object containing data.

Notes

This function does only the most basic variable checking. The resulting Grid object is most likely not writable.

`Grid`(fields, axes, metadata)	An object for holding gridded Radar data.

pyart.io.mdv

Utilities for reading of MDV files.

`MdvFile`(filename[, debug, read_fields])	A file object for MDV data.

`read_mdv`(filename[, field_names, ...])	Read a MDV file.

pyart.io.mdv.read_mdv

pyart.io.mdv.**read_mdv**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*)

Read a MDV file.

Parameters **filename** : str

Name of MDV file to read or file-like object pointing to the beginning of such a file.

field_names : dict, optional

Dictionary mapping MDV data type names to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the Py-ART configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the Py-ART configuration file will be used.

file_field_names : bool, optional

True to use the MDV data type names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

Returns **radar** : Radar

Radar object containing data from MDV file.

Notes

Currently this function can only read polar MDV files which are gzipped. Support for cartesian and non-gzipped file are planned.

pyart.io.nexrad_archive

Functions for reading NEXRAD Level II Archive files.

read_nexrad_archive(filename[, field_names, ...])	Read a NEXRAD Level 2 Archive file.

pyart.io.nexrad_archive.read_nexrad_archive

pyart.io.nexrad_archive.**read_nexrad_archive**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *bzip=None*)

Read a NEXRAD Level 2 Archive file.

Parameters **filename** : str

Filename of NEXRAD Level 2 Archive file. The files hosted by at the NOAA National Climate Data Center [R3] as well as on the UCAR THREDDS Data Server [R4] have been tested. Other NEXRAD Level 2 Archive files may or may not work. Message type 1 file at not yet supported, only message type 31.

field_names : dict, optional

Dictionary mapping NEXRAD moments to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the metadata configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the metadata configuration file will be used.

file_field_names : bool, optional

True to use the NEXRAD field names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

bzip : bool or None

True if the file is compressed as a bzip2 file, False otherwise. None will examine the filename for a bzip extension.

Returns **radar** : Radar

Radar object containing all moments and sweeps/cuts in the volume. Gates not collected are masked in the field data.

References

[R3], [R4]

pyart.io.nexrad_cdm

Functions for accessing Common Data Model (CDM) NEXRAD Level 2 files.

read_nexrad_cdm(filename[, field_names, ...])	Read a Common Data Model (CDM) NEXRAD Level 2 file.
_scan_info(dvars)	Return a list of information on the scans in the volume.
_populate_scan_dic(scan_dic, time_var, ...)	Populate a dictionary in the scan_info list.
_get_moment_data(moment_var, index, ngates)	Retieve moment data for a given scan.

pyart.io.nexrad_cdm.read_nexrad_cdm

pyart.io.nexrad_cdm.**read_nexrad_cdm**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*)

Read a Common Data Model (CDM) NEXRAD Level 2 file.

Parameters **filename** : str

File name or URL of a Common Data Model (CDM) NEXRAD Level 2 file. File of in this format can be created using the NetCDF Java Library tools [R5]. A URL of a OPeNDAP file on the UCAR THREDDS Data Server [R6] is also accepted the netCDF4 library has been compiled with OPeNDAP support.

field_names : dict, optional

Dictionary mapping NEXRAD moments to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the metadata configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the metadata configuration file will be used.

file_field_names : bool, optional

True to use the NEXRAD field names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

Returns **radar** : Radar

Radar object containing all moments and sweeps/cuts in the volume. Gates not collected are masked in the field data.

References

[R5], [R6]

pyart.io.nexrad_cdm._scan_info

pyart.io.nexrad_cdm.**_scan_info**(*dvars*)
Return a list of information on the scans in the volume.

pyart.io.nexrad_cdm._populate_scan_dic

pyart.io.nexrad_cdm.**_populate_scan_dic**(*scan_dic*, *time_var*, *time_var_i*, *moment*, *dvars*)
Populate a dictionary in the scan_info list.

pyart.io.nexrad_cdm._get_moment_data

pyart.io.nexrad_cdm.**_get_moment_data**(*moment_var*, *index*, *ngates*)
Retieve moment data for a given scan.

pyart.io.nexrad_level2

NEXRADLevel2File(filename[, bzip])	Class for accessing data in a NEXRAD (WSR-88D) Level II file.

_decompress_records(file_handler)	Decompressed the records from an BZ2 compressed Archive 2 file.
_get_record_from_buf(buf, pos)	Retrieve and unpack a NEXRAD record from a buffer.
_get_msg31_data_block(buf, ptr)	Unpack a msg_31 data block into a dictionary.
_structure_size(structure)	Find the size of a structure in bytes.
_unpack_from_buf(buf, pos, structure)	Unpack a structure from a buffer.
_unpack_structure(string, structure)	Unpack a structure from a string

pyart.io.nexrad_level2._decompress_records

pyart.io.nexrad_level2.**_decompress_records**(*file_handler*)
Decompressed the records from an BZ2 compressed Archive 2 file.

pyart.io.nexrad_level2._get_record_from_buf

pyart.io.nexrad_level2.**_get_record_from_buf**(*buf*, *pos*)
Retrieve and unpack a NEXRAD record from a buffer.

pyart.io.nexrad_level2._get_msg31_data_block

pyart.io.nexrad_level2.**_get_msg31_data_block**(*buf*, *ptr*)
Unpack a msg_31 data block into a dictionary.

pyart.io.nexrad_level2._structure_size

pyart.io.nexrad_level2.**_structure_size**(*structure*)
Find the size of a structure in bytes.

pyart.io.nexrad_level2._unpack_from_buf

pyart.io.nexrad_level2.**_unpack_from_buf**(*buf*, *pos*, *structure*)
Unpack a structure from a buffer.

pyart.io.nexrad_level2._unpack_structure

pyart.io.nexrad_level2.**_unpack_structure**(*string*, *structure*)
Unpack a structure from a string

pyart.io.radar

A general central radial scanning (or dwelling) instrument class.

Radar(time, _range, fields, metadata, ...[, ...])	A class for storing antenna coordinate radar data.
join_radar(radar1, radar2)	
is_vpt(radar[, offset])	Determine if a Radar appears to be a vertical pointing scan.
to_vpt(radar[, single_scan])	Convert an existing Radar object to represent a vertical pointing scan.

pyart.io.radar.join_radar

pyart.io.radar.**join_radar**(*radar1*, *radar2*)

pyart.io.radar.is_vpt

pyart.io.radar.**is_vpt**(*radar*, *offset=0.5*)
Determine if a Radar appears to be a vertical pointing scan.

This function only verifies that the object is a vertical pointing scan, use the to_vpt function to convert the radar to a vpt scan if this function returns True.

Parameters **radar** : Radar

Radar object to determine if

offset : float

Maximum offset of the elevation from 90 degrees to still consider to be vertically pointing.

Returns **flag** : bool

True if the radar appear to be verticle pointing, False if not.

pyart.io.radar.to_vpt

pyart.io.radar.**to_vpt**(*radar*, *single_scan=True*)

Convert an existing Radar object to represent a vertical pointing scan.

This function does not verify that the Radar object contains a vertical pointing scan. To perform such a check use is_vpt.

Parameters **radar** : Radar

Mislabeled vertical pointing scan Radar object to convert to be properly labeled. This object is converted in place, no copy of the existing data is made.

single_scan : bool, optional

True to convert the volume to a single scan, any azimuth angle data is lost. False will convert the scan to contain the same number of scans as rays, azimuth angles are retained.

pyart.io.sigmet

Reading and writing of Sigmet (raw format) files

read_sigmet(filename[, field_names, ...])	Read a Sigmet (IRIS) product file.
ymds_time_to_datetime(ymds)	Return a datetime object from a Sigmet ymds_time dictionary.
_time_order_data_and_metadata_full(data, ...)	Put Sigmet data and metadata in time increasing order by sorting the
_time_order_data_and_metadata_roll(data, ...)	Put Sigmet data and metadata in time increasing order using a single

pyart.io.sigmet.read_sigmet

pyart.io.sigmet.**read_sigmet**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *time_ordered='none'*, *debug=False*)

Read a Sigmet (IRIS) product file.

Parameters **filename** : str

Name of Sigmet (IRIS) product file to read or file-like object pointing to the beginning of such a file.

field_names : dict, optional

Dictionary mapping Sigmet data type names to radar field names. If a data type found in the file does not appear in this dictionary or has a value of None it will not be placed in the radar.fields dictionary. A value of None, the default, will use the mapping defined in the metadata configuration file.

additional_metadata : dict of dicts, optional

Dictionary of dictionaries to retrieve metadata from during this read. This metadata is not used during any successive file reads unless explicitly included. A value of None, the default, will not introduct any addition metadata and the file specific or default metadata as specified by the metadata configuration file will be used.

file_field_names : bool, optional

True to use the Sigmet data type names for the field names. If this case the field_names parameter is ignored. The field dictionary will likely only have a 'data' key, unless the fields are defined in *additional_metadata*.

exclude_fields : list or None, optional

List of fields to exclude from the radar object. This is applied after the *file_field_names* and *field_names* parameters.

time_ordered : 'full', 'none' or 'roll'.

Parameter controlling the time ordering of the data. The default, 'none' keep the data ordered in the same manner as it appears in the Sigmet file. 'roll' will attempt to time order the data within each sweep by rolling the earliest collected ray to be the beginning. Sequential ordering of the rays is maintained but strict time increasing order is not guaranteed. 'full' will place data within each sweep in a strictly time increasing order, but the rays will likely become non-sequential. The 'full' option is not recommended unless strict time increasing order is required.

debug : bool, optional

Returns **radar** : Radar

Radar object

pyart.io.sigmet.ymds_time_to_datetime

pyart.io.sigmet.**ymds_time_to_datetime**(*ymds*)
Return a datetime object from a Sigmet ymds_time dictionary.

pyart.io.sigmet._time_order_data_and_metadata_full

pyart.io.sigmet.**_time_order_data_and_metadata_full**(*data*, *metadata*)
Put Sigmet data and metadata in time increasing order by sorting the time.

pyart.io.sigmet._time_order_data_and_metadata_roll

pyart.io.sigmet.**_time_order_data_and_metadata_roll**(*data*, *metadata*)
Put Sigmet data and metadata in time increasing order using a single roll.

pyart.io._sigmetfile

A class and supporting functions for reading Sigmet (raw format) files.

SigmetFile	A class for accessing data from Sigmet (IRIS) product files.
convert_sigmet_data	Convert sigmet data.
bin2_to_angle	Return an angle from Sigmet bin2 encoded value (or array).
bin4_to_angle	Return an angle from Sigmet bin4 encoded value (or array).
_data_types_from_mask	Return a list of the data types from the words in the data_type mask.
_is_bit_set	Return True if bit is set in number.
_parse_ray_headers	Parse the metadata from Sigmet ray headers.
_unpack_structure	Unpack a structure
_unpack_key	Unpack a key.

Continued on next page

Table 1.20 – continued from previous page

_unpack_ingest_data_headers	Unpack one or more ingest_data_header from a record.
_unpack_ingest_data_header	Unpack a single ingest_data_header from record.
_unpack_raw_prod_bhdr	Return a dict with the unpacked raw_prod_bhdr from a record.
_unpack_product_hdr	Return a dict with the unpacked product_hdr from the first record.
_unpack_ingest_header	Return a dict with the unpacked ingest_header from the second record.

pyart.io._sigmetfile.convert_sigmet_data

pyart.io._sigmetfile.**convert_sigmet_data**()
Convert sigmet data.

pyart.io._sigmetfile.bin2_to_angle

pyart.io._sigmetfile.**bin2_to_angle**()
Return an angle from Sigmet bin2 encoded value (or array).

pyart.io._sigmetfile.bin4_to_angle

pyart.io._sigmetfile.**bin4_to_angle**()
Return an angle from Sigmet bin4 encoded value (or array).

pyart.io._sigmetfile._data_types_from_mask

pyart.io._sigmetfile.**_data_types_from_mask**()
Return a list of the data types from the words in the data_type mask.

pyart.io._sigmetfile._is_bit_set

pyart.io._sigmetfile.**_is_bit_set**()
Return True if bit is set in number.

pyart.io._sigmetfile._parse_ray_headers

pyart.io._sigmetfile.**_parse_ray_headers**()
Parse the metadata from Sigmet ray headers.

Parameters **ray_headers** : array, shape=(..., 6)

Ray headers to parse.

Returns **az0** : array

Azimuth angles (in degrees) at beginning of the rays.

el0 : array

Elevation angles at the beginning of the rays.

az1 : array

Azimuth angles at the end of the rays.

el1 : array

Elevation angles at the end of the rays.

nbins : array

Number of bins in the rays.

time : array

Seconds since the start of the sweep for the rays.

pyart.io._sigmetfile._unpack_structure

pyart.io._sigmetfile.**_unpack_structure**()
Unpack a structure

pyart.io._sigmetfile._unpack_key

pyart.io._sigmetfile.**_unpack_key**()
Unpack a key.

pyart.io._sigmetfile._unpack_ingest_data_headers

pyart.io._sigmetfile.**_unpack_ingest_data_headers**()
Unpack one or more ingest_data_header from a record.

Returns a list of dictionaries.

pyart.io._sigmetfile._unpack_ingest_data_header

pyart.io._sigmetfile.**_unpack_ingest_data_header**()
Unpack a single ingest_data_header from record.

pyart.io._sigmetfile._unpack_raw_prod_bhdr

pyart.io._sigmetfile.**_unpack_raw_prod_bhdr**()
Return a dict with the unpacked raw_prod_bhdr from a record.

pyart.io._sigmetfile._unpack_product_hdr

pyart.io._sigmetfile.**_unpack_product_hdr**()
Return a dict with the unpacked product_hdr from the first record.

pyart.io._sigmetfile._unpack_ingest_header

pyart.io._sigmetfile.**_unpack_ingest_header**()
Return a dict with the unpacked ingest_header from the second record.

1.2.2 pyart.config

Py-ART configuration.

pyart.config

Py-ART configuration.

load_config([filename])	Load a Py-ART configuration from a config file.
get_metadata(p)	Return a dictionary of metadata for a given parameter, p.
get_fillvalue()	Return the current fill value.
get_field_name(field)	Return the field name from the configuration file for a given field.
FileMetadata(filetype[, field_names, ...])	A class for accessing metadata needed when reading files.

pyart.config.load_config

pyart.config.**load_config**(*filename=None*)

Load a Py-ART configuration from a config file.

The default values for a number of Py-ART parameters and metadata is controlled by a single Python configuration file. An self-descriping example of this file can be found in the Py-ART source directory named **default_config.py**. These defaults can modified by setting the environmental variable *PYART_CONFIG* to point to a new configuration file. If this variable is not set then the settings contained in the **default_config.py** file are used.

The code the configuration file is executed as-is with full permission, this may present a security issue, do not load un-trusted configuration files.

The recommended method for changing these defaults is for users to copy this file into their home directory, rename it to .pyart_config.py, make any changes, and adjust their login scripts to set the PYART_CONFIG environmental variable to point to .pyart_config.py in their home directory.

Py-ART's configuration can also be modified within a script or shell session using this function, the modification will last until a the end of the script/session or until a new configuration is loaded.

Parameters **filename** : str

Filename of configuration file. If None the default configuration file is loaded from the Py-ART source code directory.

pyart.config.get_metadata

pyart.config.**get_metadata**(*p*)

Return a dictionary of metadata for a given parameter, p.

An empty dictionary will be returned in no metadata dictionary exists for parameter p.

pyart.config.get_fillvalue

pyart.config.**get_fillvalue**()

Return the current fill value.

pyart.config.get_field_name

pyart.config.**get_field_name**(*field*)
Return the field name from the configuration file for a given field.

1.2.3 pyart.correct

Radar Moment correction routines.

pyart.correct.attenuation

Attenuation correction from polarimetric radars.

Code adapted from method in Gu et al, JAMC 2011, 50, 39.

Adapted by Scott Collis and Scott Giangrande, refactored by Jonathan Helmus.

calculate_attenuation(radar, z_offset[, ...])	Calculate the attenuation from a polarimetric radar using Z-PHI method.

pyart.correct.attenuation.calculate_attenuation

pyart.correct.attenuation.**calculate_attenuation**(*radar*, *z_offset*, *debug=False*, *doc=15*, *fzl=4000.0*, *rhv_min=0.8*, *ncp_min=0.5*, *a_coef=0.06*, *beta=0.8*, *refl_field=None*, *ncp_field=None*, *rhv_field=None*, *phidp_field=None*, *spec_at_field=None*, *corr_refl_field=None*)
Calculate the attenuation from a polarimetric radar using Z-PHI method.

Parameters **radar** : Radar

Radar object to use for attenuation calculations. Must have copol_coeff, norm_coherent_power, proc_dp_phase_shift, reflectivity_horizontal fields.

z_offset : float

Horizontal reflectivity offset in dBZ.

debug : bool

True to print debugging information, False supressed this printing.

Returns **spec_at** : dict

Field dictionary containing the specific attenuation.

cor_z : dict

Field dictionary containing the corrected reflectivity.

Other Parameters **doc** : float

Number of gates at the end of each ray to to remove from the calculation.

fzl : float

Freezing layer, gates above this point are not included in the correction.

rhv_min : float

Minimum copol_coeff value to consider valid.

ncp_min : float

Minimum norm_coherent_power to consider valid.

a_coef : float

A coefficient in attenuation calculation.

beta : float

Beta parameter in attenuation calculation.

refl_field, ncp_field, rhv_field, phidp_field : str

Field names within the radar object which represent the horizonal reflectivity, normal coherent power, the copolar coefficient, and the differential phase shift. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

spec_at_field, corr_refl_field : str

Names of the specific attenuation and the corrected reflectivity fields that will be used to fill in the metadata for the returned fields. A value of None for any of these parameters will use the default field names as defined in the Py-ART configuration file.

References

Gu et al. Polarimetric Attenuation Correction in Heavy Rain at C Band, JAMC, 2011, 50, 39-58.

pyart.correct.phase_proc

Utilities for working with phase data.

Code based upon algorithm descriped in: S. E. Giangrande et al, J. of Atmos. and Ocean. Tech., 2013, 30, 1716.

Adapted by Scott Collis and Scott Giangrande, refactored by Jonathan Helmus

det_sys_phase(radar[, ncp_lev, rhohv_lev, ...])	Determine the system phase.
_det_sys_phase(ncp, rhv, phidp, last_ray_idx)	Determine the system phase, see det_sys_phase.
fzl_index(fzl, ranges, elevation, radar_height)	Return the index of the last gate below a given altitude.
det_process_range(radar, sweep, fzl[, doc])	Determine the processing range for a given sweep.
snr(line[, wl])	Return the signal to noise ratio after smoothing.
unwrap_masked(lon[, centered, copy])	Unwrap a sequence of longitudes or headings in degrees.
smooth_and_trim(x[, window_len, window])	Smooth data using a window with requested size.
smooth_and_trim_scan(x[, window_len, window])	Smooth data using a window with requested size.
noise(line[, wl])	Return the noise after smoothing.
get_phidp_unf(radar[, ncp_lev, rhohv_lev, ...])	Get Unfolded Phi differential phase
construct_A_matrix(n_gates, filt)	Construct a row-augmented A matrix. Equation 5 in Giangrande et al, 201
construct_B_vectors(phidp_mod, z_mod, filt)	Construct B vectors. See Giangrande et al, 2012.
LP_solver_cvxopt(A_Matrix, B_vectors, weights)	Solve the Linear Programming problem given in Giangrande et al, 2012 u
LP_solver_pyglpk(A_Matrix, B_vectors, weights)	Solve the Linear Programming problem given in Giangrande et al, 2012 u
solve_cylp(model, B_vectors, weights, ray, ...)	Worker process for LP_solver_cylp_mp.
LP_solver_cylp_mp(A_Matrix, B_vectors, weights)	Solve the Linear Programming problem given in Giangrande et al, 2012 u
LP_solver_cylp(A_Matrix, B_vectors, weights)	Solve the Linear Programming problem given in Giangrande et al, 2012 u

Continued on next p

Table 1.23 – continued from previous page

phase_proc_lp(radar, offset[, debug, ...])	Phase process using a LP method [1].

pyart.correct.phase_proc.det_sys_phase

pyart.correct.phase_proc.**det_sys_phase**(*radar*, *ncp_lev=0.4*, *rhohv_lev=0.6*, *ncp_field=None*, *rhv_field=None*, *phidp_field=None*)

Determine the system phase.

Parameters **radar** : Radar

Radar object for which to determine the system phase.

ncp_lev : :

Miminum normal coherent power level. Regions below this value will not be included in the phase calculation.

rhohv_lev : :

Miminum copolar coefficient level. Regions below this value will not be included in the phase calculation.

ncp_field, rhv_field, phidp_field : str

Field names within the radar object which represent the normal coherent power, the copolar coefficient, and the differential phase shift. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

Returns **sys_phase** : float or None

Estimate of the system phase. None is not estimate can be made.

pyart.correct.phase_proc._det_sys_phase

pyart.correct.phase_proc.**_det_sys_phase**(*ncp*, *rhv*, *phidp*, *last_ray_idx*, *ncp_lev=0.4*, *rhv_lev=0.6*)

Determine the system phase, see det_sys_phase.

pyart.correct.phase_proc.fzl_index

pyart.correct.phase_proc.**fzl_index**(*fzl*, *ranges*, *elevation*, *radar_height*)

Return the index of the last gate below a given altitude.

Parameters **fzl** : float

Maximum altitude.

ranges : array

Range to measurement volume/gate in meters.

elevation : float

Elevation of antenna in degrees.

radar_height : :

Altitude of radar in meters.

Returns **idx** : int

Index of last gate which has an altitude below *fzl*.

Notes

Standard atmosphere is assumed, R = 4 / 3 * Re

pyart.correct.phase_proc.det_process_range

pyart.correct.phase_proc.**det_process_range**(*radar*, *sweep*, *fzl*, *doc=10*)

Determine the processing range for a given sweep.

Queues the radar and returns the indices which can be used to slice the radar fields and select the desired sweep with gates which are below a given altitude.

Parameters **radar** : Radar

Radar object from which ranges will be determined.

sweep : int

Sweep (0 indexed) for which to determine processing ranges.

fzl : float

Maximum altitude in meters. The determined range will not include gates which are above this limit.

doc : int

Minimum number of gates which will be excluded from the determined range.

Returns **gate_end** : int

Index of last gate below *fzl* and satisfying the *doc* parameter.

ray_start : int

Ray index which defines the start of the region.

ray_end : int

Ray index which defined the end of the region.

pyart.correct.phase_proc.snr

pyart.correct.phase_proc.**snr**(*line*, *wl=11*)

Return the signal to noise ratio after smoothing.

pyart.correct.phase_proc.unwrap_masked

pyart.correct.phase_proc.**unwrap_masked**(*lon*, *centered=False*, *copy=True*)

Unwrap a sequence of longitudes or headings in degrees.

Parameters **lon** : array

Longtiudes or heading in degress. If masked output will also be masked.

centered : bool, optional

Center the unwrapping as close to zero as possible.

copy : bool, optional.

True to return a copy, False will avoid a copy when possible.

Returns **unwrap** : array

Array of unwrapped longtitudes or headings, in degrees.

pyart.correct.phase_proc.smooth_and_trim

pyart.correct.phase_proc.**smooth_and_trim**(*x*, *window_len=11*, *window='hanning'*)

Smooth data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

Parameters **x** : array

The input signal

window_len: int :

The dimension of the smoothing window; should be an odd integer.

window : str

The type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman' or 'sg_smooth'. A flat window will produce a moving average smoothing.

Returns **y** : array

The smoothed signal with length equal to the input signal.

pyart.correct.phase_proc.smooth_and_trim_scan

pyart.correct.phase_proc.**smooth_and_trim_scan**(*x*, *window_len=11*, *window='hanning'*)

Smooth data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

Parameters **x** : ndarray

The input signal

window_len: int :

The dimension of the smoothing window; should be an odd integer.

window : str

The type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman' or 'sg_smooth'. A flat window will produce a moving average smoothing.

Returns **y** : ndarray

The smoothed signal with length equal to the input signal.

pyart.correct.phase_proc.noise

pyart.correct.phase_proc.**noise**(*line*, *wl=11*)

Return the noise after smoothing.

pyart.correct.phase_proc.get_phidp_unf

pyart.correct.phase_proc.**get_phidp_unf**(*radar*, *ncp_lev=0.4*, *rhohv_lev=0.6*, *debug=False*, *ncpts=20*, *doc=-10*, *overide_sys_phase=False*, *sys_phase=-135*, *nowrap=None*, *refl_field=None*, *ncp_field=None*, *rhv_field=None*, *phidp_field=None*)

Get Unfolded Phi differential phase

Parameters **radar** : Radar

The input radar.

ncp_lev : :

Miminum normal coherent power level. Regions below this value will not be included in the calculation.

rhohv_lev : :

Miminum copolar coefficient level. Regions below this value will not be included in the calculation.

debug : bool, optioanl

True to print debugging information, False to supress printing.

ncpts : int

Minimum number of points in a ray. Regions within a ray smaller than this or beginning before this gate number are excluded from calculations.

doc : int or None.

Index of first gate not to include in field data, None include all.

overide_sys_phase : bool, optional

True to use *sys_phase* as the system phase. False will determine a value automatically.

sys_phase : float, optional

System phase, not used if overide_sys_phase is False.

nowrap : or None

Gate number where unwrapping should begin. *None* will unwrap all gates.

refl_field ncp_field, rhv_field, phidp_field : str

Field names within the radar object which represent the horizonal reflectivity, normal coherent power, the copolar coefficient, and the differential phase shift. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

Returns **cordata** : array

Unwrapped phi differential phase.

pyart.correct.phase_proc.construct_A_matrix

pyart.correct.phase_proc.**construct_A_matrix**(*n_gates*, *filt*)

Construct a row-augmented A matrix. Equation 5 in Giangrande et al, 2012.

A is a block matrix given by:

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & -\mathbf{I} \\ -\mathbf{I} & \mathbf{I} \\ \mathbf{Z} & \mathbf{M} \end{bmatrix}$$

where $\mathbf{I}$ is the identity matrix $\mathbf{Z}$ is a matrix of zeros $\mathbf{M}$ contains our differential constraints.

Each block is of shape n_gates by n_gates making shape($\mathbf{A}$) = (3 * n, 2 * n).

Note that $\mathbf{M}$ contains some side padding to deal with edge issues

Parameters **n_gates** : int

Number of gates, determines size of identity matrix

filt : array

Input filter.

Returns **a** : matrix

Row-augmented A matrix.

pyart.correct.phase_proc.construct_B_vectors

pyart.correct.phase_proc.**construct_B_vectors**(*phidp_mod*, *z_mod*, *filt*, *coef=0.914*, *dweight=60000.0*)

Construct B vectors. See Giangrande et al, 2012.

Parameters **phidp_mod** : 2D array

Phi differential phases.

z_mod : 2D array.

Reflectivity, modified as needed.

filt : array

Input filter.

coef : float, optional.

Cost coefficients.

dweight : float, optional.

Weights.

Returns **b** : matrix

Matrix containing B vectors.

pyart.correct.phase_proc.LP_solver_cvxopt

pyart.correct.phase_proc.**LP_solver_cvxopt**(*A_Matrix*, *B_vectors*, *weights*, *solver='glpk'*)

Solve the Linear Programming problem given in Giangrande et al, 2012 using the CVXOPT module.

Parameters **A_Matrix** : matrix

Row augmented A matrix, see `construct_A_matrix`

B_vectors : matrix

Matrix containing B vectors, see `construct_B_vectors`

weights : array

Weights.

solver : str or None

LP solver backend to use, choices are 'glpk', 'mosek' or None to use the conelp function in CVXOPT. 'glpk' and 'mosek' are only available if they are installed and CVXOPT was build with the correct bindings.

Returns **soln** : array

Solution to LP problem.

See also:

LP_solver_pyglpk Solve LP problem using the PyGLPK module.

LP_solver_cylp Solve LP problem using the cylp module.

LP_solver_cylp_mp Solve LP problem using the cylp module using multi processes.

pyart.correct.phase_proc.LP_solver_pyglpk

pyart.correct.phase_proc.**LP_solver_pyglpk**(*A_Matrix*, *B_vectors*, *weights*, *it_lim=7000*, *presolve=True*, *really_verbose=False*)

Solve the Linear Programming problem given in Giangrande et al, 2012 using the PyGLPK module.

Parameters **A_Matrix** : matrix

Row augmented A matrix, see `construct_A_matrix`

B_vectors : matrix

Matrix containing B vectors, see `construct_B_vectors`

weights : array

Weights.

it_lim : int

Simplex iteration limit.

presolve : bool

True to use the LP presolver.

really_verbose : bool

True to print LPX messaging. False to suppress.

Returns **soln** : array

Solution to LP problem.

See also:

`LP_solver_cvxopt` Solve LP problem using the CVXOPT module.

`LP_solver_cylp` Solve LP problem using the cylp module.

`LP_solver_cylp_mp` Solve LP problem using the cylp module using multi processes.

pyart.correct.phase_proc.solve_cylp

pyart.correct.phase_proc.**solve_cylp**(*model*, *B_vectors*, *weights*, *ray*, *chunksize*)

Worker process for LP_solver_cylp_mp.

Parameters **model** : CyClpModel

Model of the LP Problem, see `LP_solver_cylp_mp`

B_vectors : matrix

Matrix containing B vectors, see `construct_B_vectors`

weights : array

Weights.

ray : int

Starting ray.

chunksize : int

Number of rays to process.

Returns **soln** : array

Solution to LP problem.

See also:

`LP_solver_cylp_mp` Parent function.

`LP_solver_cylp` Single Process Solver.

pyart.correct.phase_proc.LP_solver_cylp_mp

pyart.correct.phase_proc.**LP_solver_cylp_mp**(*A_Matrix*, *B_vectors*, *weights*, *really_verbose=False*, *proc=1*)

Solve the Linear Programming problem given in Giangrande et al, 2012 using the CyLP module using multiple processes.

Parameters **A_Matrix** : matrix

Row augmented A matrix, see `construct_A_matrix`

B_vectors : matrix

Matrix containing B vectors, see `construct_B_vectors`

weights : array

Weights.

really_verbose : bool

True to print CLP messaging. False to suppress.

proc : int

Number of worker processes.

Returns **soln** : array

Solution to LP problem.

See also:

`LP_solver_cvxopt` Solve LP problem using the CVXOPT module.

`LP_solver_pyglpk` Solve LP problem using the PyGLPK module.

`LP_solver_cylp` Solve LP problem using the CyLP module using single process.

pyart.correct.phase_proc.LP_solver_cylp

`pyart.correct.phase_proc.`**`LP_solver_cylp`**(*A_Matrix*, *B_vectors*, *weights*, *really_verbose=False*)

Solve the Linear Programming problem given in Giangrande et al, 2012 using the CyLP module.

Parameters **A_Matrix** : matrix

Row augmented A matrix, see `construct_A_matrix`

B_vectors : matrix

Matrix containing B vectors, see `construct_B_vectors`

weights : array

Weights.

really_verbose : bool

True to print CLP messaging. False to suppress.

Returns **soln** : array

Solution to LP problem.

See also:

`LP_solver_cvxopt` Solve LP problem using the CVXOPT module.

`LP_solver_pyglpk` Solve LP problem using the PyGLPK module.

pyart.correct.phase_proc.phase_proc_lp

pyart.correct.phase_proc.**phase_proc_lp**(*radar*, *offset*, *debug=False*, *self_const=60000.0*, *low_z=10.0*, *high_z=53.0*, *min_phidp=0.01*, *min_ncp=0.5*, *min_rhv=0.8*, *fzl=4000.0*, *sys_phase=0.0*, *overide_sys_phase=False*, *nowrap=None*, *really_verbose=False*, *LP_solver='pyglpk'*, *refl_field=None*, *ncp_field=None*, *rhv_field=None*, *phidp_field=None*, *kdp_field=None*, *unf_field=None*, *window_len=35*, *proc=1*)

Phase process using a LP method [1].

Parameters **radar** : Radar

Input radar.

offset : float

Reflectivity offset in dBz.

debug : bool, optional

True to print debugging information.

self_const : float, optional

Self consistency factor.

low_z : float

Low limit for reflectivity. Reflectivity below this value is set to this limit.

high_z : float

High limit for reflectivity. Reflectivity above this value is set to this limit.

min_phidp : float

Minimum Phi differential phase.

min_ncp : float

Minimum normal coherent power.

min_rhv : float

Minimum copolar coefficient.

fzl : :

Maximum altitude.

sys_phase : float

System phase in degrees.

overide_sys_phase: bool. :

True to use *sys_phase* as the system phase. False will calculate a value automatically.

nowrap : int or None.

Gate number to begin phase unwrapping. None will unwrap all phases.

really_verbose : bool

True to print LPX messaging. False to suppress.

LP_solver : 'pyglpk' or 'cvxopt', 'cylp', or 'cylp_mp'

Module to use to solve LP problem.

refl_field, ncp_field, rhv_field, phidp_field, kdp_field: str :

Name of field in radar which contains the horizonal reflectivity, normal coherent power, copolar coefficient, differential phase shift, and differential phase. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

unf_field : str

Name of field which will be added to the radar object which will contain the unfolded differential phase. Metadata for this field will be taken from the phidp_field. A value of None will use the default field name as defined in the Py-ART configuration file.

window_len : int

Length of Sobel window applied to PhiDP field when prior to calculating KDP.

proc : int

Number of worker processes, only used when *LP_solver* is 'cylp_mp'.

Returns **reproc_phase** : dict

Field dictionary containing processed differential phase shifts.

sob_kdp : dict

Field dictionary containing recalculated differential phases.

References

[1] Giangrande, S.E., R. McGraw, and L. Lei. An Application of Linear Programming to Polarimetric Radar Differential Phase Processing. J. Atmos. and Oceanic Tech, 2013, 30, 1716.

1.2.4 pyart.graph

Radar data graphing routines.

pyart.graph.common

Common graphing routines.

dms_to_d(dms)	Degrees, minutes, seconds to degrees
radar_coords_to_cart(rng, az, ele[, debug])	Calculate Cartesian coordinate from radar coordinates
corner_to_point(corner, point)	Return the x, y distances in meters from a corner to a point.
ax_radius(lat[, units])	Return the radius of a constant latitude circle for a given latitude.

pyart.graph.common.dms_to_d

pyart.graph.common.**dms_to_d**(*dms*)

Degrees, minutes, seconds to degrees

pyart.graph.common.radar_coords_to_cart

pyart.graph.common.**radar_coords_to_cart**(*rng*, *az*, *ele*, *debug=False*)

Calculate Cartesian coordinate from radar coordinates

Parameters **rng** : array

Distances to the center of the radar gates (bins) in kilometers.

az : array

Azimuth angle of the radar in degrees.

ele : array

Elevation angle of the radar in degrees.

Returns **x, y, z** : array

Cartesian coordinates in meters from the radar.

Notes

The calculation for Cartesian coordinate is adapted from equations 2.28(b) and 2.28(c) of Doviak and Zrnic [R1] assuming a standard atmosphere (4/3 Earth's radius model).

$$z = \sqrt{r^2 + R^2 + r * R * sin(\theta_e)} - R$$
$$s = R * arcsin(\frac{r * cos(\theta_e)}{R + z})$$
$$x = s * sin(\theta_a)$$
$$y = s * cos(\theta_a)$$

Where r is the distance from the radar to the center of the gate, θ_a is the azimuth angle, θ_e is the elevation angle, s is the arc length, and R is the effective radius of the earth, taken to be 4/3 the mean radius of earth (6371 km).

References

[R1]

pyart.graph.common.corner_to_point

pyart.graph.common.**corner_to_point**(*corner*, *point*)

Return the x, y distances in meters from a corner to a point.

Assumes a spherical earth model.

Parameters **corner** : (float, float)

Latitude and longitude in degrees of the corner.

point : (float, float)

Latitude and longitude in degrees of the point.

Returns **x, y** : floats

Distances from the corner to the point in meters.

pyart.graph.common.ax_radius

pyart.graph.common.**ax_radius**(*lat*, *units='radians'*)

Return the radius of a constant latitude circle for a given latitude.

Parameters **lat** : float

Latitude at which to calculate constant latitude circle (parallel) radius.

units : 'radians' or 'degrees'

Units of lat, either 'radians' or 'degrees'.

Returns **R** : float

Radius in meters of a constant latitude circle (parallel).

pyart.graph.cm

Radar related colormaps.

revcmap(data)	Can only handle specification *data* in dictionary format.
_reverser(f)	perform reversal.
_reverse_cmap_spec(spec)	Reverses cmap specification *spec*, can handle both dict and tuple
_generate_cmap(name, lutsize)	Generates the requested cmap from it's name *name*.

pyart.graph.cm.revcmap

pyart.graph.cm.**revcmap**(*data*)

Can only handle specification *data* in dictionary format.

pyart.graph.cm._reverser

pyart.graph.cm.**_reverser**(*f*)

perform reversal.

pyart.graph.cm._reverse_cmap_spec

pyart.graph.cm.**_reverse_cmap_spec**(*spec*)

Reverses cmap specification *spec*, can handle both dict and tuple type specs.

pyart.graph.cm._generate_cmap

pyart.graph.cm.**_generate_cmap**(*name*, *lutsize*)

Generates the requested cmap from it's name *name*. The lut size is *lutsize*.

Available colormaps, reversed versions (_r) are also provided:

- BlueBrown10
- BlueBrown11
- BrBu10
- BrBu12

- Bu10
- Bu7
- BuDOr12
- BuDOr18
- BuDRd12
- BuDRd18
- BuGr14
- BuGy8
- BuOr10
- BuOr12
- BuOr8
- BuOrR14
- Carbone11
- Carbone17
- Carbone42
- Cat12
- EWilson17
- GrMg16
- Gray5
- Gray9
- NWSRef
- NWSVel
- NWS_SPW
- PD17
- RRate11
- RdYlBu11b
- RefDiff
- SCook18
- StepSeq25
- SymGray12
- Theodore16
- Wild25

pyart.graph._cm

Data for radar related colormaps.

pyart.graph.plot_cfradial

Routines for plotting radar data from CF/Radial netCDF files.

CFRadialDisplay(dataset[, shift])	A display object for creating plots from data in NetCDF4 Dataset objects.

pyart.graph.plot_mdv

Routines for plotting radar data from MDV file.

MdvDisplay(mdvfile)	A display object for creating plots from data in a MdvFile objects.

_get_default_range(mdvfile, field)	Return the default range for a field.

pyart.graph.plot_mdv._get_default_range

pyart.graph.plot_mdv.**_get_default_range**(*mdvfile*, *field*)
Return the default range for a field.

pyart.graph.radar_display

Class for creating plots from Radar objects.

RadarDisplay(radar[, shift])	A display object for creating plots from data in a radar object.

1.2.5 pyart.map

Radar mapping routines.

pyart.map.grid_mapper

Utilities for mapping radar objects to Cartesian grids.

grid_from_radars(radars, grid_shape, ...)	Map one or more radars to a Cartesian grid returning a Grid object.
map_to_grid(radars, grid_shape, grid_limits)	Map one or more radars to a Cartesian grid.
example_roi_func_constant(zg, yg, xg)	Example RoI function which returns a constant radius.
example_roi_func_dist(zg, yg, xg)	Example RoI function which returns a radius which grows with distance.
_load_nn_field_data(data, nfields, npoints, ...)	Load the nearest neighbor field data into sdata
_gen_roi_func_constant(constant_roi)	Return a RoI function which returns a constant radius.
_gen_roi_func_dist(z_factor, xy_factor, ...)	Return a RoI function whose radius grows with distance.
_gen_roi_func_dist_beam(h_factor, nb, bsp, ...)	Return a RoI function whose radius which grows with distance

pyart.map.grid_mapper.grid_from_radars

pyart.map.grid_mapper.**grid_from_radars**(*radars*, *grid_shape*, *grid_limits*, ***kwargs*)
Map one or more radars to a Cartesian grid returning a Grid object.

Additional arguments are passed to `map_to_grid`

Parameters **radars** : tuple of Radar objects.

Radar objects which will be mapped to the Cartesian grid.

grid_shape : 3-tuple of floats

Number of points in the grid (z, y, x).

grid_limits : 3-tuple of 2-tuples

Minimum and maximum grid location (inclusive) in meters for the z, x, y coordinates.

Returns **grid** : Grid

A `pyart.io.Grid` object containing the gridded radar data.

See also:

`map_to_grid` Map to grid and return a dictionary of radar fields

pyart.map.grid_mapper.map_to_grid

pyart.map.grid_mapper.**map_to_grid**(*radars*, *grid_shape*, *grid_limits*, *grid_origin=None*, *fields=None*, *refl_filter_flag=True*, *refl_field=None*, *max_refl=None*, *map_roi=True*, *weighting_function='Barnes'*, *toa=17000.0*, *copy_field_data=True*, *algorithm='kd_tree'*, *leafsize=10.0*, *roi_func='dist_beam'*, *constant_roi=500.0*, *z_factor=0.05*, *xy_factor=0.02*, *min_radius=500.0*, *h_factor=1.0*, *nb=1.5*, *bsp=1.0*)

Map one or more radars to a Cartesian grid.

Generate a Cartesian grid of points for the requested fields from the collected points from one or more radars. The field value for a grid point is found by interpolating from the collected points within a given radius of influence and weighting these nearby points according to their distance from the grid points. Collected points are filtered according to a number of criteria so that undesired points are not included in the interpolation.

Parameters **radars** : tuple of Radar objects.

Radar objects which will be mapped to the Cartesian grid.

grid_shape : 3-tuple of floats

Number of points in the grid (z, y, x).

grid_limits : 3-tuple of 2-tuples

Minimum and maximum grid location (inclusive) in meters for the z, y, x coordinates.

grid_origin : (float, float) or None

Latitude and longitude of grid origin. None sets the origin to the location of the first radar.

fields : list or None

List of fields within the radar objects which will be mapped to the cartesian grid. None, the default, will map the fields which are present in all the radar objects.

refl_filter_flag : bool

True to filter the collected points based on the reflectivity field. False to perform no filtering. Gates where the reflectivity field, specified by the *refl_field* parameter, is not-finited, masked or has a value above the *max_refl* parameter are excluded from the grid interpolation.

refl_field : str

Name of the field which will be used to filter the collected points. A value of None will use the default field name as defined in the Py-ART configuration file.

max_refl : float

Maximum allowable reflectivity. Points in the *refl_field* which are above is value are not included in the interpolation. None will include skip this filtering.

roi_func : str or function

Radius of influence function. A functions which takes an z, y, x grid location, in meters, and returns a radius (in meters) within which all collected points will be included in the weighting for that grid points. Examples can be found in the `example_roi_func_constant`, `example_roi_func_dist`, and `example_roi_func_dist_beam`. Alternatively the following strings can use to specify a built in radius of influence function:

- constant: constant radius of influence.
- dist: radius grows with the distance from each radar.
- dist_beam: radius grows with the distance from each radar and parameter are based of virtual beam sizes.

The parameters which control these functions are listed in the *Other Parameters* section below.

map_roi : bool

True to include a radius of influence field in the returned dictionary under the 'ROI' key. This is the value of roi_func at all grid points.

weighting_function : 'Barnes' or 'Cressman'

Functions used to weight nearby collected points when interpolating a grid point.

toa : float

Top of atmosphere in meters. Collected points above this height are not included in the interpolation.

Returns **grids** : dict

Dictionary of mapped fields. The keysof the dictionary are given by parameter fields. Each elements is a *grid_size* float64 array containing the interpolated grid for that field.

Other Parameters **constant_roi** : float

Radius of influence parameter for the built in 'constant' function. This parameter is the constant radius in meter for all grid points. This parameter is only used when *roi_func* is *constant*.

z_factor, xy_factor, min_radius : float

Radius of influence parameters for the built in 'dist' function. The parameter correspond to the radius size increase, in meters, per meter increase in the z-dimension from the nearest radar, the same foreach meteter in the xy-distance from the nearest radar, and the

minimum radius of influence in meters. These parameters are only used when *roi_func* is 'dist'.

h_factor, nb, bsp, min_radius : float

Radius of influence parameters for the built in 'dist_beam' function. The parameter correspond to the height scaling, virtual beam width, virtual beam spacing, and minimum radius of influence. These parameters are only used when *roi_func* is 'dist_mean'.

copy_field_data : bool

True to copy the data within the radar fields for faster gridding, the dtype for all fields in the grid will be float64. False will not copy the data which preserves the dtype of the fields in the grid, may use less memory but results in significantly slower gridding times. When False gates which are masked in a particular field but are not masked in the *refl_field* field will still be included in the interpolation. This can be prevented by setting this parameter to True or by gridding each field individually setting the *refl_field* parameter and the *fields* parameter to the field in question. It is recommended to set this parameter to True.

algorithm : 'kd_tree' or 'ball_tree'

Algorithms to use for finding the nearest neighbors. 'kd_tree' tends to be faster. This value should only effects the speed of the gridding, not the results.

leafsize : int

Leaf size passed to the neighbor lookup tree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. This value should only effect the speed of the gridding, not the results.

See also:

`grid_from_radars` Map to grid and return a Grid object.

pyart.map.grid_mapper.example_roi_func_constant

pyart.map.grid_mapper.**example_roi_func_constant**(*zg*, *yg*, *xg*)

Example RoI function which returns a constant radius.

Parameters **zg, yg, xg** : float

Distance from the grid center in meters for the x, y and z axes.

Returns **roi** : float

Radius of influence in meters

pyart.map.grid_mapper.example_roi_func_dist

pyart.map.grid_mapper.**example_roi_func_dist**(*zg*, *yg*, *xg*)

Example RoI function which returns a radius which grows with distance.

Parameters **zg, yg, xg** : float

Distance from the grid center in meters for the x, y and z axes.

Returns **roi** : float

pyart.map.grid_mapper._load_nn_field_data

pyart.map.grid_mapper.**_load_nn_field_data**(*data*, *nfields*, *npoints*, *r_nums*, *e_nums*, *sdata*)
Load the nearest neighbor field data into sdata

pyart.map.grid_mapper._gen_roi_func_constant

pyart.map.grid_mapper.**_gen_roi_func_constant**(*constant_roi*)
Return a RoI function which returns a constant radius.

See map_to_grid for a description of the parameters.

pyart.map.grid_mapper._gen_roi_func_dist

pyart.map.grid_mapper.**_gen_roi_func_dist**(*z_factor*, *xy_factor*, *min_radius*, *offsets*)
Return a RoI function whose radius grows with distance.

See map_to_grid for a description of the parameters.

pyart.map.grid_mapper._gen_roi_func_dist_beam

pyart.map.grid_mapper.**_gen_roi_func_dist_beam**(*h_factor*, *nb*, *bsp*, *min_radius*, *offsets*)
Return a RoI function whose radius which grows with distance and whose parameters are based on virtual beam size.

See map_to_grid for a description of the parameters.

NNLocator(data[, leafsize, algorithm])	Nearest neighbor locator.

1.2.6 pyart.testing

Testing functions and files.

pyart.testing.sample_objects

Functions for creating sample Radar and Grid objects.

make_empty_ppi_radar(ngates, rays_per_sweep, ...)	Return an Radar object, representing a PPI scan.
make_target_radar()	Return a PPI radar with a target like reflectivity field.
make_velocity_aliased_radar()	Return a PPI radar with a target like reflectivity field.
make_single_ray_radar()	Return a PPI radar with a single ray taken from a ARM C-SAPR Radar
make_empty_grid(grid_shape, grid_limits)	Make an empty grid object without any fields or metadata.
make_target_grid()	Make a sample Grid with a rectangular target.

pyart.testing.sample_objects.make_empty_ppi_radar

pyart.testing.sample_objects.**make_empty_ppi_radar**(*ngates*, *rays_per_sweep*, *nsweeps*)
Return an Radar object, representing a PPI scan.

Parameters **ngates** : int

Number of gates per ray.

rays_per_sweep : int

Number of rays in each PPI sweep.

nsweeps : int

Number of sweeps.

Returns **radar** : Radar

Radar object with no fields, other parameters are set to default values.

pyart.testing.sample_objects.make_target_radar

pyart.testing.sample_objects.**make_target_radar**()
Return a PPI radar with a target like reflectivity field.

pyart.testing.sample_objects.make_velocity_aliased_radar

pyart.testing.sample_objects.**make_velocity_aliased_radar**()
Return a PPI radar with a target like reflectivity field.

pyart.testing.sample_objects.make_single_ray_radar

pyart.testing.sample_objects.**make_single_ray_radar**()
Return a PPI radar with a single ray taken from a ARM C-SAPR Radar

Radar object returned has 'reflectivity_horizontal', 'norm_coherent_power', 'copol_coeff', 'dp_phase_shift', and 'diff_phase' fields with no metadata but a 'data' key. This radar is used for unit tests in correct modules.

pyart.testing.sample_objects.make_empty_grid

pyart.testing.sample_objects.**make_empty_grid**(*grid_shape*, *grid_limits*)
Make an empty grid object without any fields or metadata.

Parameters **grid_shape** : 3-tuple of floats

Number of points in the grid (x, y, z).

grid_limits : 3-tuple of 2-tuples

Minimum and maximum grid location (inclusive) in meters for the x, y, z coordinates.

Returns **grid** : Grid

Empty Grid object, centered near the ARM SGP site (Oklahoma).

pyart.testing.sample_objects.make_target_grid

pyart.testing.sample_objects.**make_target_grid**()
Make a sample Grid with a rectangular target.

pyart.testing.sample_files

Sample radar files in a number of formats. Many of these files are incomplete, they should only be used for testing, not production.

MDV_PPI_FILE	str(object='') -> string
MDV_RHI_FILE	str(object='') -> string
CFRADIAL_PPI_FILE	str(object='') -> string
CFRADIAL_RHI_FILE	str(object='') -> string
SIGMET_PPI_FILE	str(object='') -> string
SIGMET_RHI_FILE	str(object='') -> string
NEXRAD_ARCHIVE_FILE	str(object='') -> string
NEXRAD_ARCHIVE_COMPRESSED_FILE	str(object='') -> string
NEXRAD_CDM_FILE	str(object='') -> string
INTERP_SOUNDE_FILE	str(object='') -> string

pyart.testing.sample_files.MDV_PPI_FILE

pyart.testing.sample_files.**MDV_PPI_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/envs/latest/**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.MDV_RHI_FILE

pyart.testing.sample_files.**MDV_RHI_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/envs/latest/**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.CFRADIAL_PPI_FILE

pyart.testing.sample_files.**CFRADIAL_PPI_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/envs**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.CFRADIAL_RHI_FILE

pyart.testing.sample_files.**CFRADIAL_RHI_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/envs**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.SIGMET_PPI_FILE

pyart.testing.sample_files.**SIGMET_PPI_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/envs/la**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.SIGMET_RHI_FILE

pyart.testing.sample_files.**SIGMET_RHI_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/envs/la**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.NEXRAD_ARCHIVE_FILE

pyart.testing.sample_files.**NEXRAD_ARCHIVE_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/e**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.NEXRAD_ARCHIVE_COMPRESSED_FILE

pyart.testing.sample_files.**NEXRAD_ARCHIVE_COMPRESSED_FILE = '/home/docs/checkouts/readthedocs.org/user_**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.NEXRAD_CDM_FILE

pyart.testing.sample_files.**NEXRAD_CDM_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/envs/la**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyart.testing.sample_files.INTERP_SOUNDE_FILE

pyart.testing.sample_files.**INTERP_SOUNDE_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/py-art/en**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

1.3 Examples

1.3.1 General examples

General-purpose and introductory example for Py-ART.

1.3.2 Moment correction examples

Performing radar moment corrections in antenna (radial) coordinates.

[R16] http://www.ncdc.noaa.gov/

[R17] http://thredds.ucar.edu/thredds/catalog.html

[R18] http://www.unidata.ucar.edu/software/netcdf-java/documentation.htm

[R19] http://thredds.ucar.edu/thredds/catalog.html

[R2] Doviak and Zrnic, Doppler Radar and Weather Observations, Second Edition, 1993, p. 21.

[R3] http://www.ncdc.noaa.gov/

[R4] http://thredds.ucar.edu/thredds/catalog.html

[R5] http://www.unidata.ucar.edu/software/netcdf-java/documentation.htm

[R6] http://thredds.ucar.edu/thredds/catalog.html

[R1] Doviak and Zrnic, Doppler Radar and Weather Observations, Second Edition, 1993, p. 21.

p

p