pyart-mch library reference for developers

Release 0.0.1

meteoswiss-mdr

Dec 09, 2016

Contents:

CHAPTER

ONE

PYART.IO.ARM_SONDE

Utilities for ARM sonde NetCDF files.

read_arm_sonde(filename)	Read a ARM sonde file returning a wind profile.
read_arm_sonde_vap(filename[, radar, ...])	Read a ARM interpolated or merged sonde returning a wind profile.

pyart.io.arm_sonde.**read_arm_sonde**(*filename*)

Read a ARM sonde file returning a wind profile.

Parameters **filename** : str

Name of ARM sonde NetCDF file to read data from.

pyart.io.arm_sonde.**read_arm_sonde_vap**(*filename*, *radar=None*, *target_datetime=None*)

Read a ARM interpolated or merged sonde returning a wind profile.

Parameters **filename** : str

Name of ARM interpolate or merged sonde NetCDF file to read data from.

radar : Radar, optional

If provided the profile returned is that which is closest in time to the first ray collected in this radar. Either radar or target_datetime must be provided.

target_datetime : datetime, optional

If specified the profile returned is that which is closest in time to this datetime.

CHAPTER TWO

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.**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'
- 'NEXRADL3'
- 'UF'
- 'HDF4'
- 'RSL'
- 'DORAD'
- 'SIGMET'
- 'LASSEN'
- 'BZ2'
- 'GZ'
- 'UNKNOWN'

Parameters **filename** : str

Name of file to examine.

Returns **filetype** : str

Type of file.

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 will use the TRMM RSL library to read files which are supported both natively and by RSL. False will choose the native read function. RSL will always be used to read a file if it is not supported natively.

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.

delay_field_loading : bool

True to delay loading of field data from the file until the 'data' key in a particular field dictionary is accessed. In this case the field attribute of the returned Radar object will contain LazyLoadDict objects not dict objects. Not all file types support this parameter.

CHAPTER

THREE

PYART.IO.CFRADIAL

Utilities for reading CF/Radial files.

_NetCDFVariableDataExtractor(ncvar)	Class facilitating on demand extraction of data from a NetCDF variable.

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[, lazydict])	Convert a NetCDF Dataset variable to a dictionary.
_unpack_variable_gate_field_dic(dic, shape, ...)	Create a 2D array from a 1D field data, dic update in place
_create_ncvar(dic, dataset, name, dimensions)	Create and fill a Variable in a netCDF Dataset object.

class pyart.io.cfradial.**_NetCDFVariableDataExtractor**(*ncvar*)

Bases: object

Class facilitating on demand extraction of data from a NetCDF variable.

Parameters **ncvar** : netCDF4.Variable

NetCDF Variable from which data will be extracted.

Methods

__call__()	Return an array containing data from the stored variable.

__call__()

Return an array containing data from the stored variable.

__class__

alias of type

__delattr__

Implement delattr(self, name).

__dict__ = mappingproxy({'__dict__': <attribute '__dict__' of '_NetCDFVariableDataExtractor' objects>, '__doc__':

__dir__() → list

default dir() implementation

`__eq__`
Return self==value.

`__format__``()`
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__`(*ncvar*)
initialize the object.

`__le__`
Return self<=value.

`__lt__`
Return self<value.

`__module__` = 'pyart.io.cfradial'

`__ne__`
Return self!=value.

`__new__``()`
Create and return a new object. See help(type) for accurate signature.

`__reduce__``()`
helper for pickle

`__reduce_ex__``()`
helper for pickle

`__repr__`
Return repr(self).

`__setattr__`
Implement setattr(self, name, value).

`__sizeof__``()` → int
size of object in memory, in bytes

`__str__`
Return str(self).

`__subclasshook__``()`
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

`__weakref__`
list of weak references to the object (if defined)

`pyart.io.cfradial._calculate_scale_and_offset`(*dic*, *dtype*, *minimum=None*, *maximum=None*)

Calculate appropriated 'scale_factor' and 'add_offset' for nc variable in dic in order to scaling to fit dtype range.

Parameters **dic** : dict

Radar dictionary containing variable data and meta-data

dtype : Numpy Dtype

Integer numpy dtype to map to.

minimum, maximum : float

Greatest and smallest values in the data, those values will be mapped to the smallest+1 and greates values that dtype can hold. If equal to None, numpy.amin and numpy.amax will be used on the data contained in dic to determine these values.

`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.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`(*ncvar*, *lazydict=False*)

Convert a NetCDF Dataset variable to a dictionary.

`pyart.io.cfradial._unpack_variable_gate_field_dic`(*dic*, *shape*, *ray_n_gates*, *ray_start_index*)

Create a 2D array from a 1D field data, dic update in place

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

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.

delay_field_loading : bool

True to delay loading of field data from the file until the 'data' key in a particular field dictionary is accessed. In this case the field attribute of the returned Radar object will contain LazyLoadDict objects not dict objects. Delayed field loading will not provide any speedup in file where the number of gates vary between rays (ngates_vary=True) and is not recommended.

Returns radar : Radar

Radar object.

Notes

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

`pyart.io.cfradial.`**`write_cfradial`**(*filename*, *radar*, *format='NETCDF4'*, *time_reference=None*, *arm_time_variables=False*)

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

The files produced by this routine follow the CF/Radial standard. Attempts are also made to to meet many of the standards outlined in the ARM Data File Standards.

To control how the netCDF variables are created, set any of the following keys in the radar attribute dictionaries.

- _Zlib
- _DeflateLevel
- _Shuffle
- _Fletcher32
- _Continguous
- _ChunkSizes
- _Endianness
- _Least_significant_digit
- _FillValue

See the netCDF4 documentation for details on these settings.

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. The default, None, will include the time_reference variable when the first time value is non-zero.

arm_time_variables : bool

True to create the ARM standard time variables base_time and time_offset, False will not create these variables.

CHAPTER

FOUR

PYART.IO.CHL

Utilities for reading CSU-CHILL CHL files.

ChlFile(filename[, ns_time, debug])	A file object for CHL data.

read_chl(filename[, field_names, ...])	Read a CSU-CHILL CHL file.
_unpack_structure(string, structure)	Unpack a structure

class pyart.io.chl.**ChlFile** (*filename*, *ns_time=True*, *debug=False*)

Bases: object

A file object for CHL data.

Parameters **filename** : str or file-like.

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

ns_time : bool

True to determine ray collection times to the nano-second, False will only determine times to the second.

debug : bool

True to keep packet data in the _packets attribute to aid in debugging.

Attributes

ngates	(int) Number of gates per ray.
num_sweeps	(int) Number of sweeps in the volume.
gate_spacing	(float) Spacing in meters between gates.
first_gate_offset	(float) Distance in meters to the first range gate.
time	(list of ints) Time in seconds in epoch for each ray in the volume.
azimuth	(list of floats) Azimuth angle for each ray in the volume in degrees.
elevation	(list of floats) Elevation angle for each ray in the volume in degrees.
fixed_angle	(list of floats) Fixed angles for each sweep.
sweep_number	(list of ints) Sweep numbers reported in file.
scan_types	(list of ints) Chill defined scan type for each sweep.
rays_per_sweep	(list of ints) Number of rays in each sweep.
fields	(dict) Dictionary of field data index by field number.
radar_info	(dict) Radar information recorded in the file.
field_info	(dict) Field information (limits, name, etc.) recorded in the file.
processor_info	(dict) Porcessor information recorded in the file.

Methods

`close()`	Close the file.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` = mappingproxy({'_parse_field_scale_block': <function ChlFile._parse_field_scale_block>, '__doc__': '\n A fi

`__dir__``()` → list
default dir() implementation

`__eq__`
Return self==value.

`__format__``()`
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__`(*filename*, *ns_time=True*, *debug=False*)

`__le__`
Return self<=value.

`__lt__`
Return self<value.

`__module__` = 'pyart.io.chl'

`__ne__`
Return self!=value.

`__new__`()
Create and return a new object. See help(type) for accurate signature.

`__reduce__`()
helper for pickle

`__reduce_ex__`()
helper for pickle

`__repr__`
Return repr(self).

`__setattr__`
Implement setattr(self, name, value).

`__sizeof__`() → int
size of object in memory, in bytes

`__str__`
Return str(self).

`__subclasshook__`()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

`__weakref__`
list of weak references to the object (if defined)

`_extract_fields`()
Extract field data from _dstring attribute post read.

`_parse_field_scale_block`(*payload*)
Parse a field_scale block. Add scale to field_info attr.

`_parse_file_hdr_block`(*payload*)
Parse a field_hdr block.

`_parse_processor_info_block`(*payload*)
Parse a processor_info block. Set dr attribute.

`_parse_radar_info_block`(*payload*)
Parse a radar_info block. Update metadata attribute.

`_parse_ray_hdr_block`(*payload*)
Parse a ray_hdr block. Update associated attributes.

`_parse_scan_seg_block`(*payload*)
Parse a scan_seg_block. Update sweep attributes.

`_parse_sweep_block`(*payload*)
Parse a sweep block. Set num_sweeps attribute.

_read_block()
Read a block from an open CHL file

close()
Close the file.

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

pyart.io.chl.**read_chl**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=None*, *exclude_fields=None*, *use_file_field_attributes=True*, ***kwargs*)
Read a CSU-CHILL CHL file.

Parameters filename : str

Name of CHL file.

field_names : dict, optional

Dictionary mapping CHL field 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 CHL field names for the field names in the radar object. 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.

use_file_field_attributes : bool, optional

True to use information provided by in the file to set the field attribute *long_name*, *units*, *valid_max*, and *valid_min*. False will not set these unless they are defined in the configuration file or in *additional_metadata*.

Returns radar : Radar

Radar object containing data from CHL file.

CHAPTER

FIVE

PYART.IO.COMMON

Input/output routines common to many file formats.

prepare_for_read(filename)	Return a file like object read for reading.
stringarray_to_chararray(arr[, numchars])	Convert an string array to a character array with one extra dimension.
_test_arguments(dic)	Issue a warning if receive non-empty argument dict
make_time_unit_str(dtobj)	Return a time unit string from a datetime object.

pyart.io.common.**_test_arguments**(*dic*)

Issue a warning if receive non-empty argument dict

pyart.io.common.**make_time_unit_str**(*dtobj*)

Return a time unit string from a datetime object.

pyart.io.common.**prepare_for_read**(*filename*)

Return a file like object read for reading.

Open a file for reading in binary mode with transparent decompression of Gzip and BZip2 files. The resulting file-like object should be closed.

Parameters filename : str or file-like object

Filename or file-like object which will be opened. File-like objects will not be examined for compressed data.

Returns file_like : file-like object

File like object from which data can be read.

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,).

CHAPTER

SIX

PYART.IO.GRID_IO

Reading and writing Grid objects.

read_grid(filename[, exclude_fields])	Read a netCDF grid file produced by Py-ART.
write_grid(filename, grid[, format, ...])	Write a Grid object to a CF-1.5 and ARM standard netCDF file
_make_coordinatesystem_dict(grid)	Return a dictionary containing parameters for a coordinate transform.

pyart.io.grid_io.**_make_coordinatesystem_dict**(*grid*)

Return a dictionary containing parameters for a coordinate transform.

Examine the grid projection attribute and other grid attributes to return a dictionary containing parameters which can be written to a netCDF variable to specify a horizontal coordinate transform recognized by Unidata's CDM. Return None when the projection defined in the grid cannot be mapped to a CDM coordinate transform.

pyart.io.grid_io.**read_grid**(*filename*, *exclude_fields=None*, ***kwargs*)

Read a netCDF grid file produced by Py-ART.

Parameters filename : str

Filename of netCDF grid file to read. This file must have been produced by write_grid() or have identical layout.

Returns grid : Grid

Grid object containing gridded data.

Other Parameters exclude_fields : list

A list of fields to exclude from the grid object.

pyart.io.grid_io.**write_grid**(*filename*, *grid*, *format='NETCDF4'*, *write_proj_coord_sys=True*, *proj_coord_sys=None*, *arm_time_variables=False*, *write_point_x_y_z=False*, *write_point_lon_lat_alt=False*)

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

To control how the netCDF variables are created, set any of the following keys in the grid attribute dictionaries.

- _Zlib
- _DeflateLevel
- _Shuffle
- _Fletcher32
- _Continguous

- _ChunkSizes
- _Endianness
- _Least_significant_digit
- _FillValue

See the netCDF4 documentation for details on these settings.

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.

write_proj_coord_sys bool, optional

True to write information on the coordinate transform used in the map projection to the ProjectionCoordinateSystem variable following the CDM Object Model. The resulting file should be interpreted as containing geographic grids by tools which use the Java NetCDF library (THREDDS, toolsUI, etc).

proj_coord_sys : dict or None, optional

Dictionary of parameters which will be written to the ProjectionCoordinateSystem NetCDF variable if write_proj_coord_sys is True. A value of None will attempt to generate an appropriate dictionary by examining the projection attribute of the grid object. If the projection is not understood a warnings will be issued.

arm_time_variables : bool, optional

True to write the ARM standard time variables base_time and time_offset. False will not write these variables.

write_point_x_y_z : bool, optional

True to include the point_x, point_y and point_z variables in the written file, False will not write these variables.

write_point_lon_lat_alt : bool, optional

True to include the point_longitude, point_latitude and point_altitude variables in the written file, False will not write these variables.

CHAPTER

SEVEN

PYART.IO.MDV_COMMON

Functions and classes common between MDV grid and radar files.

MdvFile(filename[, debug, read_fields])	A file object for MDV data.
_MdvVolumeDataExtractor(mdvfile, field_num, ...)	Class facilitating on demand extraction of data from a MDV file.

class pyart.io.mdv_common.**MdvFile**(*filename*, *debug=False*, *read_fields=False*)

Bases: object

A file object for MDV data.

A *MdvFile* object stores metadata and data from a MDV file. Metadata is stored in dictionaries as attributes of the object, field data is stored as NumPy ndarrays as attributes with the field name. By default only metadata is read initially and field data must be read using the *read_a_field* or *read_all_fields* methods. This behavior can be changed by setting the *read_fields* parameter to True.

Parameters **filename** : str, file-like or None.

Name of MDV file to read or file-like object pointing to the beginning of such a file. None can be used to initalize an object which can be used for writing mdv files.

debug : bool

True to print out debugging information, False to supress

read_fields : bool

True to read all field during initalization, False (default) only reads metadata.

Notes

This class is not stable enough for general purpose MDV reading/writing, nor is that the intention, but with care it can provide sufficient read/write capacity.

Methods

close()	Close the MDV file.
read_a_field(fnum[, debug])	Read a field from the MDV file.
read_all_fields()	Read all fields, storing data to field name attributes.
write(filename[, debug])	Write object data to a MDV file.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` = mappingproxy({'_write_elevs': <function MdvFile._write_elevs>, '_pack_mapped': <function MdvFile._pa

`__dir__`() → list
default dir() implementation

`__eq__`
Return self==value.

`__format__`()
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__`(*filename*, *debug=False*, *read_fields=False*)
initalize

`__le__`
Return self<=value.

`__lt__`
Return self<value.

`__module__` = 'pyart.io.mdv_common'

`__ne__`
Return self!=value.

`__new__`()
Create and return a new object. See help(type) for accurate signature.

`__reduce__`()
helper for pickle

`__reduce_ex__`()
helper for pickle

`__repr__`
Return repr(self).

`__setattr__`
Implement setattr(self, name, value).

`__sizeof__`() → int
size of object in memory, in bytes

`__str__`
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_calc_file_offsets()
Calculate file offsets.

_calc_geometry()
Calculate geometry, return az_deg, range_km, el_deg.

_get_calib()
Get the calibration information, return a dict.

_get_chunk_header()
Get a single chunk header, return a dict.

_get_chunk_headers(*nchunks*)
Get nchunk chunk headers, return a list of dicts.

_get_chunks(*debug=False*)
Get data in chunks, return radar_info, elevations, calib_info.

_get_compression_info()
Get compression infomation, return a dict.

_get_elevs(*nbytes*)
Return an array of elevation read from current file position.

_get_field_header()
Read a single field header, return a dict.

_get_field_headers(*nfields*)
Read nfields field headers, return a list of dicts.

_get_levels_info(*nlevels*)
Get nlevel information, return a dict.

_get_master_header()
Read the MDV master header, return a dict.

_get_radar_info()
Get the radar information, return dict.

_get_unknown_chunk(*cnum*)
Get raw data from chunk

_get_vlevel_header()
Read a single vlevel header, return a dict.

_get_vlevel_headers(*nfields*)
Read nfields vlevel headers, return a list of dicts.

_make_carts_dict()
Return a carts dictionary, distances in meters.

_make_fields_list()
Return a list of fields.

_make_time_dict()
Return a time dictionary.

_pack_mapped(*d*, *mapper*, *fmt*)
Create a packed string using a mapper and format.

_secs_since_epoch(*dt*)
Return the number of seconds since the epoch for a datetime.

_time_dict_into_header()
Complete time information in master_header from the time dict

_unpack_mapped_tuple(*l*, *mapper*)
Create a dictionary from a tuple using a mapper.

_write_a_field(*fnum*, *debug=False*)
write field number 'fnum' to mdv file

_write_calib(*d*)
Write calibration information.

_write_chunk_header(*d*)
Write the a single chunk header.

_write_chunk_headers(*nchunks*)
Write nchunk chunk headers.

_write_chunks(*debug=False*)
write chunks data

_write_compression_info(*d*)
Write compression infomation

_write_elevs(*l*)
Write an array of elevation.

_write_field_header(*d*)
Write the a single field header.

_write_field_headers(*nfields*)
Write nfields field headers.

_write_levels_info(*nlevels*, *d*)
write levels information, return a dict.

_write_master_header()
Write the MDV master header.

_write_radar_info(*d*)
Write radar information.

_write_unknown_chunk(*data*)
Write raw data from chunk

_write_vlevel_header(*d*)
Write the a single vfield header.

_write_vlevel_headers(*nfields*)
Write nfields vlevel headers

calib_fmt = '>16s 6i 51f 14f'

calib_mapper = [('radar_name', 0, 1), ('year', 1, 2), ('month', 2, 3), ('day', 3, 4), ('hour', 4, 5), ('minute', 5, 6), ('second

chunk_header_fmt = '>5i 2i 480s i'

chunk_header_mapper = [('record_len1', 0, 1), ('struct_id', 1, 2), ('chunk_id', 2, 3), ('chunk_data_offset', 3, 4), ('size',

close()
Close the MDV file.

compression_info_fmt = '>I I I I 2I'

compression_info_mapper = [('magic_cookie', 0, 1), ('nbytes_uncompressed', 1, 2), ('nbytes_compressed', 2, 3), ('nb

field_header_fmt = '>17i 10i 9i 4i f f 8f 12f 4f 5f 64s 16s 16s 16s 16s i'

field_header_mapper = [('record_len1', 0, 1), ('struct_id', 1, 2), ('field_code', 2, 3), ('user_time1', 3, 4), ('forecast_del

master_header_fmt = b'>28i 8i i 5i 6f 3f 12f 512s 128s 128s i'

master_header_mapper = [('record_len1', 0, 1), ('struct_id', 1, 2), ('revision_number', 2, 3), ('time_gen', 3, 4), ('user_

radar_info_fmt = '>12i 2i 22f 4f 40s 40s'

radar_info_mapper = [('radar_id', 0, 1), ('radar_type', 1, 2), ('nfields', 2, 3), ('ngates', 3, 4), ('samples_per_beam', 4,

read_a_field(*fnum*, *debug=False*)
Read a field from the MDV file.

Parameters **fnum** : int

Field number to read.

debug : bool

True to print debugging information, False to supress.

Returns **field_data** : array

Field data. This data is also stored as a object attribute under the field name.

See also:

read_all_fields Read all fields in the MDV file.

read_all_fields()
Read all fields, storing data to field name attributes.

vlevel_header_fmt = '>i i 122i 4i 122f 5f i'

vlevel_header_mapper = [('record_len1', 0, 1), ('struct_id', 1, 2), ('type', 2, 124), ('unused_si32', 124, 128), ('level', 1

write(*filename*, *debug=False*)
Write object data to a MDV file.

Note that the file is not explicitly closes, use x.close() to close file object when complete.

Parameters **filename** : str or file-like

Filename or open file object to which data will be written.

debug : bool, options

True to print out debugging information, False to supress.

class pyart.io.mdv_common.**_MdvVolumeDataExtractor**(*mdvfile*, *field_num*, *fillvalue*, *two_dims=True*)

Bases: object

Class facilitating on demand extraction of data from a MDV file.

Parameters **mdvfile** : MdvFile

Open MdvFile object to extract data from.

field_num : int

Field number of data to be extracted.

fillvalue : int

Value used to fill masked values in the returned array.

two_dims : bool.

True to combine the first and second dimension of the array when returning the data, False will return a three dimensional array.

Methods

`__call__`()	Return an array containing data from the referenced volume.

`__call__`()
Return an array containing data from the referenced volume.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` = mappingproxy({'__dict__': <attribute '__dict__' of '_MdvVolumeDataExtractor' objects>, '__doc__': '\n C

`__dir__`() → list
default dir() implementation

`__eq__`
Return self==value.

`__format__`()
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__`(*mdvfile*, *field_num*, *fillvalue*, *two_dims=True*)
initialize the object.

`__le__`
Return self<=value.

`__lt__`
Return self<value.

`__module__` = 'pyart.io.mdv_common'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

pyart.io.mdv_common.**_decode_rle8**(*compr_data*, *key*, *decompr_size*)
Decode 8-bit MDV run length encoding.

CHAPTER

EIGHT

PYART.IO.MDV_RADAR

Utilities for reading of MDV radar files.

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

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

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.

delay_field_loading : bool

True to delay loading of field data from the file until the 'data' key in a particular field dictionary is accessed. In this case the field attribute of the returned Radar object will contain LazyLoadDict objects not dict objects. Not all file types support this parameter.

Returns **radar** : Radar

Radar object containing data from MDV file.

Notes

Currently this function can only read polar MDV files with fields compressed with gzip or zlib.

CHAPTER

NINE

PYART.IO.NEXRADL3_READ

Functions for reading NEXRAD Level 3 products.

read_nexrad_level3(filename[, field_names, ...])	Read a NEXRAD Level 3 product.

pyart.io.nexradl3_read.**read_nexrad_level3**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, ***kwargs*)

Read a NEXRAD Level 3 product.

Parameters filename : str

Filename of NEXRAD Level 3 product file. The files hosted by at the NOAA National Climate Data Center *[R5]* as well as on the NWS WSR-88D Level III Data Collection and Distribution Network have been tests. Other NEXRAD Level 3 files may or may not work. A file-like object pointing to the beginning of such a file is also supported.

field_names : dict, optional

Dictionary mapping NEXRAD level 3 product number to radar field names. If the product number of 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 product number for the field name. In 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]*

CHAPTER TEN

PYART.IO.NEXRAD_ARCHIVE

Functions for reading NEXRAD Level II Archive files.

_NEXRADLevel2StagedField(nfile, moment, ...)	A class to facilitate on demand loading of field data from a Level 2 file.

read_nexrad_archive(filename[, field_names, ...])	Read a NEXRAD Level 2 Archive file.
_find_range_params(scan_info, filemetadata)	Return range parameters, first_gate, gate_spacing, last_gate.
_find_scans_to_interp(scan_info, first_gate, ...)	Return a dict indicating what moments/scans need interpolation.
_interpolate_scan(mdata, start, end, ...[, ...])	Interpolate a single NEXRAD moment scan from 1000 m to 250 m.

class `pyart.io.nexrad_archive.`**`_NEXRADLevel2StagedField`**(*nfile*, *moment*, *max_ngates*, *scans*)

Bases: `object`

A class to facilitate on demand loading of field data from a Level 2 file.

Methods

__call__()	Return the array containing the field data.

`__call__`()
Return the array containing the field data.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` = mappingproxy({‘__dict__’: <attribute ‘__dict__’ of ‘_NEXRADLevel2StagedField’ objects>, ‘__doc__’: ‘\n

`__dir__`() → list
default dir() implementation

`__eq__`
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*nfile*, *moment*, *max_ngates*, *scans*)
initialize.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.io.nexrad_archive'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

pyart.io.nexrad_archive.**_find_range_params**(*scan_info*, *filemetadata*)
Return range parameters, first_gate, gate_spacing, last_gate.

`pyart.io.nexrad_archive._find_scans_to_interp`(*scan_info*, *first_gate*, *gate_spacing*, *filemetadata*)

Return a dict indicating what moments/scans need interpolation.

`pyart.io.nexrad_archive._interpolate_scan`(*mdata*, *start*, *end*, *moment_ngates*, *linear_interp=True*)

Interpolate a single NEXRAD moment scan from 1000 m to 250 m.

`pyart.io.nexrad_archive.read_nexrad_archive`(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *delay_field_loading=False*, *station=None*, *scans=None*, *linear_interp=True*, ***kwargs*)

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 *[R9]* as well as on the UCAR THREDDS Data Server *[R10]* have been tested. Other NEXRAD Level 2 Archive files may or may not work. Message type 1 file and message type 31 files are supported.

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.

delay_field_loading : bool, optional

True to delay loading of field data from the file until the 'data' key in a particular field dictionary is accessed. In this case the field attribute of the returned Radar object will contain LazyLoadDict objects not dict objects.

station : str or None, optional

Four letter ICAO name of the NEXRAD station used to determine the location in the returned radar object. This parameter is only used when the location is not contained in the file, which occur in older NEXRAD message 1 files.

scans : list or None, optional

Read only specified scans from the file. None (the default) will read all scans.

linear_interp : bool, optional

True (the default) to perform linear interpolation between valid pairs of gates in low resolution rays in files mixed resolution rays. False will perform a nearest neighbor interpolation. This parameter is not used if the resolution of all rays in the file or requested sweeps is constant.

Returns **radar** : Radar

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

References

[R9], *[R10]*

CHAPTER

ELEVEN

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

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

pyart.io.nexrad_cdm.**read_nexrad_cdm**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *station=None*, ***kwargs*)
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 *[R13]*. A URL of a OPeNDAP file on the UCAR THREDDS Data Server *[R14]* 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.

station : str

Four letter ICAO name of the NEXRAD station used to determine the location in the returned radar object. This parameter is only used when the location is not contained in the file, which occur in older NEXRAD files. If the location is not provided in the file and this parameter is set to None the station name will be determined from the filename.

Returns **radar** : Radar

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

References

[R13], *[R14]*

CHAPTER

TWELVE

PYART.IO.NEXRAD_COMMON

Data and functions common to all types of NEXRAD files.

get_nexrad_location(station)	Return the latitude, longitude and altitude of a NEXRAD station

`pyart.io.nexrad_common.`**`get_nexrad_location`**(*station*)

Return the latitude, longitude and altitude of a NEXRAD station

Parameters station : str

Four letter NEXRAD station ICAO name.

Returns lat, lon, alt : float

Latitude (in degrees), longitude (in degrees), and altitude (in meters above mean sea level) of the NEXRAD station.

CHAPTER

THIRTEEN

PYART.IO.NEXRAD_INTERPOLATE

Interpolation of NEXRAD moments from 1000 meter to 250 meter gate spacing.

_fast_interpolate_scan	Interpolate a single NEXRAD moment scan from 1000 m to 250 m.

pyart.io.nexrad_interpolate.**_fast_interpolate_scan**()
Interpolate a single NEXRAD moment scan from 1000 m to 250 m.

CHAPTER FOURTEEN

PYART.IO.NEXRAD_LEVEL2

NEXRADLevel2File(filename)	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

class `pyart.io.nexrad_level2.`**`NEXRADLevel2File`**(*filename*)

Bases: `object`

Class for accessing data in a NEXRAD (WSR-88D) Level II file.

NEXRAD Level II files *[R17]*, also know as NEXRAD Archive Level II or WSR-88D Archive level 2, are available from the NOAA National Climate Data Center *[R18]* as well as on the UCAR THREDDS Data Server *[R19]*. Files with uncompressed messages and compressed messages are supported. This class supports reading both "message 31" and "message 1" type files.

Parameters **filename** : str

Filename of Archive II file to read.

References

[R17], *[R18]*, *[R19]*

Attributes

radial_records	(list) Radial (1 or 31) messages in the file.
nscans	(int) Number of scans in the file.
scan_msgs	(list of arrays) Each element specifies the indices of the message in the radial_records attribute which belong to a given scan.
volume_header	(dict) Volume header.
vcp	(dict) VCP information dictionary.
_records	(list) A list of all records (message) in the file.
_fh	(file-like) File like object from which data is read.
_msg_type	('31' or '1':) Type of radial messages in file

Methods

`close`()	Close the file.
`get_azimuth_angles`([scans])	Retrieve the azimuth angles of all rays in the requested scans.
`get_data`(moment, max_ngates[, scans, raw_data])	Retrieve moment data for a given set of scans.
`get_elevation_angles`([scans])	Retrieve the elevation angles of all rays in the requested scans.
`get_nrays`(scan)	Return the number of rays in a given scan.
`get_nyquist_vel`([scans])	Retrieve the Nyquist velocities of the requested scans.
`get_range`(scan_num, moment)	Return an array of gate ranges for a given scan and moment.
`get_target_angles`([scans])	Retrieve the target elevation angle of the requested scans.
`get_times`([scans])	Retrieve the times at which the rays were collected.
`get_unambigous_range`([scans])	Retrieve the unambiguous range of the requested scans.
`get_vcp_pattern`()	Return the numerical volume coverage pattern (VCP) or None if unknown.
`location`()	Find the location of the radar.
`scan_info`([scans])	Return a list of dictionaries with scan information.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` = mappingproxy({'get_target_angles': <function NEXRADLevel2File.get_target_angles>, '_radial_array': <f

`__dir__`() → list
default dir() implementation

`__eq__`
Return self==value.

`__format__`()
default object formatter

`__ge__`
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__ (*filename*)
initalize the object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.io.nexrad_level2'

__ne__
Return self!=value.

__new__ ()
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__ () → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_msg_nums (*scans*)
Find the all message number for a list of scans.

_radial_array (*scans*, *key*)
Return an array of radial header elements for all rays in scans.

_radial_sub_array (*scans*, *key*)
Return an array of RAD or msg_header elements for all rays in scans.

close()
Close the file.

get_azimuth_angles(*scans=None*)
Retrieve the azimuth angles of all rays in the requested scans.

Parameters scans : list ot None

Scans (0 based) for which ray (radial) azimuth angles will be retrieved. None (the default) will return the angles for all scans in the volume.

Returns angles : ndarray

Azimuth angles in degress for all rays in the requested scans.

get_data(*moment*, *max_ngates*, *scans=None*, *raw_data=False*)
Retrieve moment data for a given set of scans.

Masked points indicate that the data was not collected, below threshold or is range folded.

Parameters moment : 'REF', 'VEL', 'SW', 'ZDR', 'PHI', or 'RHO'

Moment for which to to retrieve data.

max_ngates : int

Maximum number of gates (bins) in any ray. requested.

raw_data : bool

True to return the raw data, False to perform masking as well as applying the appropiate scale and offset to the data. When raw_data is True values of 1 in the data likely indicate that the gate was not present in the sweep, in some cases in will indicate range folded data.

scans : list or None.

Scans to retrieve data from (0 based). None (the default) will get the data for all scans in the volume.

Returns data : ndarray

get_elevation_angles(*scans=None*)
Retrieve the elevation angles of all rays in the requested scans.

Parameters scans : list or None

Scans (0 based) for which ray (radial) azimuth angles will be retrieved. None (the default) will return the angles for all scans in the volume.

Returns angles : ndarray

Elevation angles in degress for all rays in the requested scans.

get_nrays(*scan*)
Return the number of rays in a given scan.

Parameters scan : int

Scan of interest (0 based)

Returns nrays : int

Number of rays (radials) in the scan.

get_nyquist_vel(*scans=None*)
Retrieve the Nyquist velocities of the requested scans.

Parameters **scans** : list or None

Scans (0 based) for which the Nyquist velocities will be retrieved. None (the default) will return the velocities for all scans in the volume.

Returns **velocities** : ndarray

Nyquist velocities (in m/s) for the requested scans.

get_range(*scan_num*, *moment*)

Return an array of gate ranges for a given scan and moment.

Parameters **scan_num** : int

Scan number (0 based).

moment : 'REF', 'VEL', 'SW', 'ZDR', 'PHI', or 'RHO'

Moment of interest.

Returns **range** : ndarray

Range in meters from the antenna to the center of gate (bin).

get_target_angles(*scans=None*)

Retrieve the target elevation angle of the requested scans.

Parameters **scans** : list or None

Scans (0 based) for which the target elevation angles will be retrieved. None (the default) will return the angles for all scans in the volume.

Returns **angles** : ndarray

Target elevation angles in degress for the requested scans.

get_times(*scans=None*)

Retrieve the times at which the rays were collected.

Parameters **scans** : list or None

Scans (0-based) to retrieve ray (radial) collection times from. None (the default) will return the times for all scans in the volume.

Returns **time_start** : Datetime

Initial time.

time : ndarray

Offset in seconds from the initial time at which the rays in the requested scans were collected.

get_unambigous_range(*scans=None*)

Retrieve the unambiguous range of the requested scans.

Parameters **scans** : list or None

Scans (0 based) for which the unambiguous range will be retrieved. None (the default) will return the range for all scans in the volume.

Returns **unambiguous_range** : ndarray

Unambiguous range (in meters) for the requested scans.

get_vcp_pattern()

Return the numerical volume coverage pattern (VCP) or None if unknown.

location()
Find the location of the radar.

Returns all zeros if location is not available.

Returns latitude: float

Latitude of the radar in degrees.

longitude: float

Longitude of the radar in degrees.

height : int

Height of radar and feedhorn in meters above mean sea level.

scan_info(*scans=None*)
Return a list of dictionaries with scan information.

Parameters **scans** : list ot None

Scans (0 based) for which ray (radial) azimuth angles will be retrieved. None (the default) will return the angles for all scans in the volume.

Returns **scan_info** : list, optional

A list of the scan performed with a dictionary with keys 'moments', 'ngates', 'nrays', 'first_gate' and 'gate_spacing' for each scan. The 'moments', 'ngates', 'first_gate', and 'gate_spacing' keys are lists of the NEXRAD moments and gate information for that moment collected during the specific scan. The 'nrays' key provides the number of radials collected in the given scan.

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

pyart.io.nexrad_level2.**_get_msg1_from_buf**(*buf*, *pos*, *dic*)
Retrieve and unpack a MSG1 record from a buffer.

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

pyart.io.nexrad_level2.**_get_msg31_from_buf**(*buf*, *pos*, *dic*)
Retrieve and unpack a MSG31 record from a buffer.

pyart.io.nexrad_level2.**_get_msg5_from_buf**(*buf*, *pos*, *dic*)
Retrieve and unpack a MSG1 record from a buffer.

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

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

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

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

CHAPTER

FIFTEEN

PYART.IO.NEXRAD_LEVEL3

Class for reading data from NEXRAD Level 3 files.

NEXRADLevel3File(filename)	A Class for accessing data in NEXRAD Level III (3) files.

nexrad_level3_message_code(filename)	Return the message (product) code for a NEXRAD Level 3 file.
_datetime_from_mdate_mtime(mdate, mtime)	Returns a datetime for a given message date and time.
_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
_int16_to_float16(val)	Convert a 16 bit interger into a 16 bit float.

class `pyart.io.nexrad_level3.`**`NEXRADLevel3File`**(*filename*)

Bases: `object`

A Class for accessing data in NEXRAD Level III (3) files.

Attributes

text_header	(dic) File textual header.
msg_header	(dic) Message header.
prod_descr	(dic) Product description.
symbology_header	(dict) Symbology header.
packet_header	(dict) Radial data array packet header.
radial_headers	(list of dicts) List of radials headers
raw_data	(array) Raw unscaled, unmasked data.
data	(array) Scaled, masked radial data.
_fh	(file-like) File like object from which data is read.

Methods

close()	Close the file.
get_azimuth()	Return an array of starting azimuth angles in degrees.
get_data()	Return an masked array containing the field data.
get_elevation()	Return the sweep elevation angle in degrees.

Continued on next page

Table 15.3 – continued from previous page

get_location()	Return the latitude, longitude and height of the radar.
get_range()	Return an array of gate range spacing in meters.
get_volume_start_datetime()	Return a datetime of the start of the radar volume.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'get_range': <function NEXRADLevel3File.get_range>, '__doc__': '\n A Class for accessin

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*filename*)
initalize the object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.io.nexrad_level3'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_get_data_8_or_16_levels()
Return a masked array for products with 8 or 16 data levels.

_get_data_msg_134()
Return a masked array for product with message code 134.

_read_symbology_block(*buf2*)
Read symbology block.

close()
Close the file.

get_azimuth()
Return an array of starting azimuth angles in degrees.

get_data()
Return an masked array containing the field data.

get_elevation()
Return the sweep elevation angle in degrees.

get_location()
Return the latitude, longitude and height of the radar.

get_range()
Return an array of gate range spacing in meters.

get_volume_start_datetime()
Return a datetime of the start of the radar volume.

pyart.io.nexrad_level3.**_datetime_from_mdate_mtime**(*mdate*, *mtime*)
Returns a datetime for a given message date and time.

pyart.io.nexrad_level3.**_int16_to_float16**(*val*)
Convert a 16 bit interger into a 16 bit float.

pyart.io.nexrad_level3.**_structure_size**(*structure*)
Find the size of a structure in bytes.

pyart.io.nexrad_level3.**_unpack_from_buf**(*buf*, *pos*, *structure*)
Unpack a structure from a buffer.

pyart.io.nexrad_level3.**_unpack_structure**(*string*, *structure*)
Unpack a structure from a string

pyart.io.nexrad_level3.**nexrad_level3_message_code**(*filename*)
Return the message (product) code for a NEXRAD Level 3 file.

CHAPTER SIXTEEN

PYART.IO.RSL

Python wrapper around the RSL library.

_RslVolumeDataExtractor(rslfile, volume_num, ...)	Class facilitating on demand extraction of data from a RSL file.

read_rsl(filename[, field_names, ...])	Read a file supported by RSL
VOLUMENUM2RSLNAME	
RSLNAME2VOLUMENUM	

class pyart.io.rsl.**_RslVolumeDataExtractor**(*rslfile*, *volume_num*, *fillvalue*)

Bases: object

Class facilitating on demand extraction of data from a RSL file.

Parameters **rslfile** : RslFile

Open RslFile object to extract data from.

volume_num : int

Volume number of data to be extracted.

fillvalue : int

Value used to fill masked values in the returned array.

Methods

__call__()	Return an array containing data from the referenced volume.

__call__()

Return an array containing data from the referenced volume.

__class__

alias of type

__delattr__

Implement delattr(self, name).

__dict__ = mappingproxy({'__dict__': <attribute '__dict__' of '_RslVolumeDataExtractor' objects>, '__doc__': '\n Cla

`__dir__``()` → list
default dir() implementation

`__eq__`
Return self==value.

`__format__``()`
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__``(`*rslfile*, *volume_num*, *fillvalue*`)`
initialize the object.

`__le__`
Return self<=value.

`__lt__`
Return self<value.

`__module__` = 'pyart.io.rsl'

`__ne__`
Return self!=value.

`__new__``()`
Create and return a new object. See help(type) for accurate signature.

`__reduce__``()`
helper for pickle

`__reduce_ex__``()`
helper for pickle

`__repr__`
Return repr(self).

`__setattr__`
Implement setattr(self, name, value).

`__sizeof__``()` → int
size of object in memory, in bytes

`__str__`
Return str(self).

`__subclasshook__``()`
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

pyart.io.rsl.**_dms_to_d**(*dms*)
Degrees, minutes, seconds to degrees

pyart.io.rsl.**read_rsl**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *delay_field_loading=False*, *radar_format=None*, *callid=None*, *skip_range_check=False*)
Read a file supported by RSL

Parameters filename : str or RSL_radar

Name of file whose format is supported by RSL.

field_names : dict, optional

Dictionary mapping RSL 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 RSL 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.

delay_field_loading : bool

True to delay loading of field data from the file until the 'data' key in a particular field dictionary is accessed. In this case the field attribute of the returned Radar object will contain LazyLoadDict objects not dict objects.

radar_format : str or None

Format of the radar file. Must be 'wsr88d' or None.

callid : str or None

Four letter NEXRAD radar Call ID, only used when radar_format is 'wsr88d'.

skip_range_check : bool, optional

True to skip check for uniform range bin location, the reported range locations will only be verified true for the first ray. False will perform the check and raise a IOError when the locations of the gates change between rays.

Returns radar : Radar

Radar object.

CHAPTER

SEVENTEEN

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.
_is_time_ordered_by_reversal(data, metadata, ...)	Returns if volume can be time ordered by reversing some or all sweeps.
_is_time_ordered_by_roll(data, metadata, ...)	Returns if volume can be time ordered by rolling some or all sweeps.
_is_time_ordered_by_reverse_roll(data, ...)	Returns if volume can be time ordered by reversing and rolling some or all sweeps.
_time_order_data_and_metadata_roll(data, ...)	Put Sigmet data and metadata in time increasing order using a roll operation.
_time_order_data_and_metadata_reverse(data, ...)	Put Sigmet data and metadata in time increasing order by reverse sweep in time reversed order.
_time_order_data_and_metadata_full(data, ...)	Put Sigmet data and metadata in time increasing order by sorting the times.

pyart.io.sigmet.**_is_time_ordered_by_reversal**(*data*, *metadata*, *rays_per_sweep*)
Returns if volume can be time ordered by reversing some or all sweeps. True if the volume can be time ordered, False if not.

pyart.io.sigmet.**_is_time_ordered_by_reverse_roll**(*data*, *metadata*, *rays_per_sweep*)
Returns if volume can be time ordered by reversing and rolling some or all sweeps. True if the volume can be time ordered, False if not.

pyart.io.sigmet.**_is_time_ordered_by_roll**(*data*, *metadata*, *rays_per_sweep*)
Returns if volume can be time ordered by rolling some or all sweeps. True if the volume can be time ordered, False if not.

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

pyart.io.sigmet.**_time_order_data_and_metadata_reverse**(*data*, *metadata*, *rays_per_sweep*)
Put Sigmet data and metadata in time increasing order by reverse sweep in time reversed order.

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

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

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 : 'none', 'sequential', 'full', ..., optional

Parameter controlling if and how the rays are re-ordered by time. The default, 'none' keeps the rays ordered in the same manner as they appears in the Sigmet file. 'sequential' will determind and apply an operation which maintains a sequential ray order in elevation or azimuth yet orders the rays according to time. If no operation can be found to accomplish this a warning is issue and the rays are returned in their original order. 'roll', 'reverse', and 'reverse_and_roll' will apply that operation to the rays in order to place them in time order, direct use of these is not recommended. 'full' will order the rays in strictly time increasing order, but the rays will likely become non-sequential, thisoption is not recommended unless strict time increasing order is required.

full_xhdr : bool or None

Flag to read in all extended headers for possible decoding. None will determine if extended headers should be read in automatically by examining the extended header type.

noaa_hh_hdr : bool or None

Flag indicating if the extended header should be decoded as those used by the NOAA Hurricane Hunters aircraft radars. None will determine if the extended header is of this type automatically by examining the header. The *full_xhdr* parameter is set to True when this parameter is True.

ignore_xhdr : bool, optional

True to ignore all data in the extended headers if they exist. False, the default, extracts milliseconds precision times and other parameter from the extended headers if they exists in the file.

ignore_sweep_start_ms : bool or None, optional

True to ignore the millisecond parameter in the start time for each sweep, False will uses this parameter when determining the timing of each ray. None, the default, will ignore the millisecond sweep start timing only when the file does not contain extended headers or when the extended header has been explicity ignored using the *ignore_xhdr* parameter. The TRMM RSL library ignores these times so setting this parameter to True is required to match the times determined when reading Sigmet files with `pyart.io.read_rsl()`. When there are not extended headers ignoring the millisecond sweep times provides time data which is always prior to the actual collection time with an error from 0 to 2 seconds.

debug : bool, optional

Print debug information during read.

Returns radar : Radar

Radar object

`pyart.io.sigmet.`**`ymds_time_to_datetime`**(*ymds*)

Return a datetime object from a Sigmet ymds_time dictionary.

CHAPTER

EIGHTEEN

PYART.IO.UF

Reading of Universal format (UF) files

read_uf(filename[, field_names, ...])	Read a UF File.
_get_scan_type(ufray)	Ruturn the scan type of a UF ray.
_get_instrument_parameters(ufile, filemetadata)	Return a dictionary containing instrument parameters.

pyart.io.uf.**_get_instrument_parameters**(*ufile*, *filemetadata*)
Return a dictionary containing instrument parameters.

pyart.io.uf.**_get_scan_type**(*ufray*)
Ruturn the scan type of a UF ray.

pyart.io.uf.**read_uf**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *delay_field_loading=False*, ***kwargs*)
Read a UF File.

Parameters **filename** : str or file-like

Name of Universal format file to read data from.

field_names : dict, optional

Dictionary mapping UF 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 introduce 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 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.

delay_field_loading : bool

This option is not implemented in the function but included for compatibility.

Returns radar : Radar

Radar object.

CHAPTER

NINETEEN

PYART.IO.UFFILE

Low level class for reading Universal Format (UF) files.

UFFile(filename)	A class for reading data from Universal Format (UF) files.
UFRay(record)	A class for reading data from a single ray (record) in a UF file.

_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

class pyart.io.uffile.**UFFile**(*filename*)

Bases: object

A class for reading data from Universal Format (UF) files.

Parameters filename : str or file-like

Filename or file-like object containing data in Universal format (UF).

Attributes

rays	(list of UFRay objects) List of rays within the UF file.
nrays, nsweeps	(int) Number of rays and sweep in the file.
ray_sweep_numbers	(array) Sweep number of each ray in the file.
first_ray_in_sweep, last_ray_in_sweep	(array) Indices of the first and last ray in each sweep.

Methods

close()	Close the file.
get_azimuths()	Return an array of azimuth angles for each ray in degrees.
get_datetimes()	Return a list of datetimes for each ray.
get_elevations()	Return an array of elevation angles for each ray in degrees.
get_field_data(field_number)	Return a 2D array of scale/masked field data for the volume.

Continued on next page

Table 19.3 – continued from previous page

get_nyquists()	Return an array of nyquist velocities for each ray in m/s.
get_prts()	Return an array of prts for each ray in microseconds.
get_pulse_widths()	Return an array of pulse widths for each ray in meters.
get_sweep_fixed_angles()	Return an array of fixed angles for each sweep in degrees.
get_sweep_polarizations()	Return an array of polarization modes for each sweep.
get_sweep_rates()	Return an array of sweep rates for each ray in degrees/sec.

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'_get_ray_sweep_numbers': <function UFFile._get_ray_sweep_numbers>, 'get_nyquists':

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*filename*)
initialize.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.io.uffile'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_get_ray_sweep_numbers()
Return an array of the sweep_number stored in each ray.

_get_sweep_limits()
Return arrays of indices of first and last ray in each sweep.

close()
Close the file.

get_azimuths()
Return an array of azimuth angles for each ray in degrees.

get_datetimes()
Return a list of datetimes for each ray.

get_elevations()
Return an array of elevation angles for each ray in degrees.

get_field_data(*field_number*)
Return a 2D array of scale/masked field data for the volume.

get_nyquists()
Return an array of nyquist velocities for each ray in m/s.

Returns None if nyquist velocities cannot be determined for all rays.

get_prts()
Return an array of prts for each ray in microseconds.

get_pulse_widths()
Return an array of pulse widths for each ray in meters.

get_sweep_fixed_angles()
Return an array of fixed angles for each sweep in degrees.

get_sweep_polarizations()
Return an array of polarization modes for each sweep.

get_sweep_rates()
Return an array of sweep rates for each ray in degrees/sec.

class `pyart.io.uffile.UFRay`(*record*)
Bases: `object`

A class for reading data from a single ray (record) in a UF file.

Parameters **record** : str

Byte string containing the binary data for a UF ray.

Attributes

mandatory_header	(dic) Mandatory header.
optional_header	(dic or None) Optional header or None if no optional header exists in the record.
data_header	(dic) Data header.
field_positions	(list) List of dictionaries containing the data type and data position.
field_headers	(list) List of field header dictionaries for all fields in the ray.
field_raw_data	(list) List containing array of raw field data for each field in the ray.
_buf	(str) Bytes which make up the record.

Methods

`get_datetime`()	Return a datetime object for the ray.
`get_field_data`(field_number)	Return array of raw data for a particular field in the ray.
`get_location`()	Return the latitude, longitude and height of the ray.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` = mappingproxy({'__dict__': <attribute '__dict__' of 'UFRay' objects>, '__doc__': '\n A class for reading dat

`__dir__`() → list
default dir() implementation

`__eq__`
Return self==value.

`__format__`()
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__`(*record*)
Initalize the object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.io.uffile'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

get_datetime()
Return a datetime object for the ray.

get_field_data(*field_number*)
Return array of raw data for a particular field in the ray.

Field header is appended to the list in the field_headers attribute.

get_location()
Return the latitude, longitude and height of the ray.

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

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

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

CHAPTER

TWENTY

PYART.IO.UF_WRITE

Functions for writing UF files.

UFRayCreator(radar, field_mapping, ...[, ...])	A class for generating UF rays for writing UF file.

write_uf(filename, radar[, uf_field_names, ...])	Write a Radar object to a UF file.
_d_to_dms(in_deg)	Degrees to degree, minutes, seconds.
_pack_structure(dic, structure)	Pack a structure from a dictionary

class pyart.io.uf_write.**UFRayCreator**(*radar*, *field_mapping*, *field_write_order*, *volume_start=None*, *templates_extra=None*)

Bases: object

A class for generating UF rays for writing UF file.

Parameters radar : Radar

Radar used to create rays.

field_write_order : list

Order in which radar fields should be written out in the UF file. None, the default, will determine a valid order automatically.

volume_start : datetime, optional

Start of volume used to set UF volume fields.

templates_extra : dict of dict, optional

Advanced usage parameter for setting UF structure templates. Elements defined in dictionaries with keys 'mandatory_header', 'optional_header', and 'field_header' will be added to the appropriate structure template.

Methods

make_data_array(field, ray_num[, scale])	Return an array of UF field data.
make_data_header()	Return a byte string representing a UF data header.
make_field_header(data_offset, ray_num, ...)	Return a byte string representing a field header.
make_field_position()	Return a byte string representing the UF field positions.
make_field_position_list()	Return a list of field position dictionaries.

Continued on next page

Table 20.3 – continued from previous page

make_fsi_vel(ray_num, scale)	Return a byte string representing a UF FSI velocity structure.
make_mandatory_header(ray_num)	Return a byte string representing a UF mandatory header.
make_optional_header()	Return a byte string representing a UF optional header.
make_ray(ray_num)	Return a byte string representing a complete UF ray.

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'make_field_position_list': <function UFRayCreator.make_field_position_list>, '_calc_ray_

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*radar*, *field_mapping*, *field_write_order*, *volume_start=None*, *templates_extra=None*)
Initialize the object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.io.uf_write'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

static **_calc_ray_num_to_sweep_num**(*radar*)
Return an array mapping ray number to sweep numbers.

static **_calc_record_length**(*radar*, *field_mapping*, *field_write_order*)
Return the record length in 2-byte words.

_parse_custom_templates(*templates_extra*)
Set additional template parameter using provided dictionary.

_set_field_header()
Populate the field header template with radar parameters.

_set_mandatory_header_location()
Populate the mandatory header template with the location.

_set_optional_header_time(*volume_start*)
Populate the optional header template with the volume start.

make_data_array(*field*, *ray_num*, *scale=100.0*)
Return an array of UF field data.

make_data_header()
Return a byte string representing a UF data header.

make_field_header(*data_offset*, *ray_num*, *scale_factor*)
Return a byte string representing a field header.

make_field_position()
Return a byte string representing the UF field positions.

make_field_position_list()
Return a list of field position dictionaries.

make_fsi_vel(*ray_num*, *scale*)
Return a byte string representing a UF FSI velocity structure.

make_mandatory_header(*ray_num*)
Return a byte string representing a UF mandatory header.

make_optional_header()
Return a byte string representing a UF optional header.

make_ray(*ray_num*)
Return a byte string representing a complete UF ray.

pyart.io.uf_write.**_d_to_dms**(*in_deg*)
Degrees to degree, minutes, seconds.

pyart.io.uf_write.**_find_field_mapping**(*radar*, *uf_field_names*, *radar_field_names*, *exclude_fields*)
Return a dictionary mapping radar fields to UF data types.

pyart.io.uf_write.**_pack_structure**(*dic*, *structure*)
Pack a structure from a dictionary

pyart.io.uf_write.**write_uf**(*filename*, *radar*, *uf_field_names=None*, *radar_field_names=False*, *exclude_fields=None*, *field_write_order=None*, *volume_start=None*, *templates_extra=None*)
Write a Radar object to a UF file.

Create a UF file containing data from the provided radar instance. The UF file will contain instrument parameters from the following dictionaries if they contained in radar.instrument_parameters:

- radar_beam_width_h
- radar_beam_width_v
- radar_receiver_bandwidth
- frequency
- pulse_width
- prt
- polarization_mode
- nyquist_velocity

If any of these parameter are not present a default or sentinel value will be written in the UF file in the place of the parameter. This is also true for the data in the scan_rate attribute.

Radar fields will be scaled and rounded to integer values when writing to UF files. The scale factor for each field can be specified in the *_UF_scale_factor* key for each field dictionary. If not specified the default scaling (100) will be used.

Parameters **filename** : str or file-like object.

Filename of UF file to create. If a file-like object is specified data will be written using the write method.

radar : Radar

Radar object from which to create UF file.

uf_field_names : dict or None, optional

Mapping between radar fields and two character UF data type names. Field names mapped to None or with no mapping will be excluded from writing. If None, the default mappings for UF files will be used.

radar_field_names : bool, optional

True to use the radar field names as the field names of the UF fields. False to use the uf_field_names mapping to generate UF field names. The *exclude_fields* argument can still be used to exclude fields from the UF file when this parameter is True. When reading a UF file using *file_field_names=True* set this parameter to True to write a UF file with the same field names.

exclude_fields : list or None, optional

List of radar fields to exclude from writing.

field_write_order : list or None, optional

Order in which radar fields should be written out in the UF file. None, the default, will determine a valid order automatically.

volume_start : datetime, optional

Start of volume used to set UF volume structure elements.

templates_extra : dict of dict or None

Advanced usage parameter for setting UF structure templates. Elements defined in dictionaries with keys 'mandatory_header', 'optional_header', and 'field_header' will be used to build the structure template.

CHAPTER

TWENTYONE

PYART.IO.WRITE_GRID_GEOTIFF

Write a Py-ART Grid object to a GeoTIFF file.

write_grid_geotiff(grid, filename, field[, ...])	Write a Py-ART Grid object to a GeoTIFF file.
_get_rgb_values(data, vmin, vmax, ...)	Get RGB values for later output to GeoTIFF, given a 2D data field, display min/max and color table info.
_create_sld(cmap, vmin, vmax, filename[, ...])	Develop a Style Layer Descriptor file given a color table and user-specified min/max files.

pyart.io.output_to_geotiff.**_create_sld**(*cmap*, *vmin*, *vmax*, *filename*, *color_levels=None*)

Develop a Style Layer Descriptor file given a color table and user-specified min/max files. Output color info to that file. Only called if sld is True in write_grid_geotiff.

Parameters **cmap** : str or matplotlib.colors.Colormap object, optional

Colormap to use for RGB output or SLD file.

vmin : int or float

Minimum value to color for RGB output or SLD file.

vmax : int or float

Maximum value to color for RGB output or SLD file.

filename : str

Template for SLD filename. The suffix (presumably .tif or .tiff) is removed and replaced with .sld. Thus, if provided a filename radar_reflectivity.tif, the output SLD file will be called radar_reflectivity.sld.

Other Parameters **color_levels** : int or None, optional

Number of color levels in cmap. Useful for categorical colormaps with steps << 255 (e.g., hydrometeor ID).

pyart.io.output_to_geotiff.**_get_rgb_values**(*data*, *vmin*, *vmax*, *color_levels*, *cmap*)

Get RGB values for later output to GeoTIFF, given a 2D data field, display min/max and color table info. Missing data get numpy.nan. Only called if rgb is True in write_grid_geotiff.

Parameters **data** : numpy.ndarray object, dtype int or float

Two-dimensional data array

vmin : int or float

Minimum value to color for RGB output or SLD file.

vmax : int or float

Maximum value to color for RGB output or SLD file.

color_levels : int

Number of color levels in cmap. Useful for categorical colormaps with steps << 255 (e.g., hydrometeor ID).

cmap : str or matplotlib.colors.Colormap object, optional

Colormap to use for RGB output or SLD file.

Returns rarr : numpy.ndarray object, dtype int

Red channel indices (range = 0-255)

barr : numpy.ndarray object, dtype int

Blue channel indices (range = 0-255)

garr : numpy.ndarray object, dtype int

Green channel indices (range = 0-255)

`pyart.io.output_to_geotiff.`**`write_grid_geotiff`**(*grid*, *filename*, *field*, *rgb=False*, *level=None*, *cmap='viridis'*, *vmin=0*, *vmax=75*, *color_levels=None*, *warp=False*, *sld=False*)

Write a Py-ART Grid object to a GeoTIFF file.

The GeoTIFF can be the standard Azimuthal Equidistant projection used in Py-ART, or a lat/lon projection on a WGS84 sphere. The latter is typically more usable in web mapping applications. The GeoTIFF can contain a single float-point raster band, or three RGB byte raster bands. The former will require an SLD file for colorful display using standard GIS or web mapping software, while the latter will show colors "out-of-the-box" but lack actual data values. The function also can output an SLD file based on the user-specified inputs. User can specify the 2D vertical level to be output. If this is not specified, a 2D composite is created. User also can specify the field to output.

This function requires GDAL Python libraries to be installed. These are available via conda; e.g., 'conda install gdal'

Parameters grid : pyart.core.Grid object

Grid object to write to file.

filename : str

Filename for the GeoTIFF.

field : str

Field name to output to file.

Other Parameters rbg : bool, optional

True - Output 3-band RGB GeoTIFF

False - Output single-channel, float-valued GeoTIFF. For display, likely will need an SLD file to provide a color table.

level: int or None, optional

Index for z-axis plane to output. None gives composite values (i.e., max in each vertical column).

cmap : str or matplotlib.colors.Colormap object, optional

Colormap to use for RGB output or SLD file.

vmin : int or float, optional

Minimum value to color for RGB output or SLD file.

vmax : int or float, optional

Maximum value to color for RGB output or SLD file.

color_levels : int or None, optional

Number of color levels in cmap. Useful for categorical colormaps with steps << 255 (e.g., hydrometeor ID).

warp : bool, optional

True - Use gdalwarp (called from command line using os.system) to warp to a lat/lon WGS84 grid.

False - No warping will be performed. Output will be Az. Equidistant.

sld : bool, optional

True - Create a Style Layer Descriptor file (SLD) mapped to vmin/vmax and cmap. File is named same as output TIFF, except for .sld extension.

False - Don't do this.

CHAPTER

TWENTYTWO

PYART.IO._SIGMET_NOAA_HH

Functions needed for reading Sigmet files from the airborne radar located on NOAA's Hurricane Hunter aircraft.

_decode_noaa_hh_hdr(raw_extended_headers, ...)	Extract data from Sigmet extended headers produced by NOAA Hurricane Hunter airborne radars.
_georeference_yprime(roll, pitch, heading, ...)	Compute georeferenced azimuth and elevation angles for a Y-prime radar.

pyart.io._sigmet_noaa_hh.**_decode_noaa_hh_hdr**(*raw_extended_headers*, *filemetadata*, *azimuth*, *elevation*, *position_source='irs'*, *heading_source='irs'*)

Extract data from Sigmet extended headers produced by NOAA Hurricane Hunter airborne radars.

Parameters raw_extended_headers : ndarray

Raw Sigmet extended headers.

filemetadata : FileMetadata

FileMetadata class from which metadata will be derived.

azimuth : dict

Dictionary of azimuth angles recorded in Sigmet file.

elevation : dict

Dictionary of elevation angles recorded in Sigmet file.

position_source: {'irs', 'gps', 'aamps'}, optional

Instrument from which to derive position parameters.

heading_source: {'irs', 'aamps'}

Instrument from which to derive heading parameters.

Returns latitude : dict

Dictionary containing latitude data and metadata.

longitude : dict

Dictionary containing longitude data and metadata.

altitude : dict

Dictionary containing altitude data and metadata.

heading_params : dict

Dictionary of dictionary containing aircraft heading data and metadata. Contains 'heading', 'roll', pitch', 'drift', 'rotation', 'tilt' and 'georefs_applied' dictionaries.

pyart.io._sigmet_noaa_hh.**_georeference_yprime**(*roll*, *pitch*, *heading*, *drift*, *rotation*, *tilt*)

Compute georeferenced azimuth and elevation angles for a Y-prime radar.

This is the georeferencing needed for the tail doppler radar on the NOAA P3 aircraft.

CHAPTER

TWENTYTHREE

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.
_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.

class `pyart.io._sigmetfile.`**`SigmetFile`**

Bases: `object`

A class for accessing data from Sigmet (IRIS) product files.

Parameters **filename** : str

Filename or file-like object.

Attributes

debug	(bool) Set True to print out debugging information, False otherwise.
product_hdr	(dict) Product_hdr structure.
ingest_header	(dict) Ingest_header structure.
ingest_data_headers	(list of dict) Ingest_data_header structures for each data type. Indexed by the data type name (str). None when data has not yet been read.
data_types	(list) List of data types (int) in the file.
data_type_names	(list) List of data type names (stR) in the file.
ndata_types	(int) Number of data types in the file.
_fh	(file) Open file being read.
_raw_product_bhdrs	(list) List of raw_product_bhdr structure dictionaries seperated by sweep. None when data has not yet been read.

Methods

`close`	Close the file.
`read_data`	Read all data from the file.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dir__``()` → list
default dir() implementation

`__eq__`
Return self==value.

`__format__``()`
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__`
initalize the object.

`__le__`
Return self<=value.

`__lt__`
Return self<value.

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__pyx_vtable__ = <capsule object NULL>

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

_determine_data_types()
Determine the available data types in the file.

_fh

_get_sweep()
Get the data and metadata from the next sweep.

If the file ends early None is returned for all values.

Parameters **full_xhdr** : bool

True to return the full extended headers if they exist padded with ones. False will return a length 1 extended header converted to int32. This is useful when the file contains a customer specified extended header (for example aircraft radar).

raw_data : bool, optional

True to return the raw_data for the given sweep, False to convert the data to floating point representation.

Returns **ingest_data_headers** : list of dict

List of ingest_data_header structures for each data type.

sweep_data : list of arrays

Sweep data for each data types in the order they appear in the file.

sweep_metadata : list of tuples

Sweep metadata for each data type in the same order as sweep_data.

_raw_product_bhdrs

_rbuf_pos

_record_number

close()

Close the file.

data_type_names

data_types

debug

ingest_data_headers

ingest_header

ndata_types

product_hdr

read_data()

Read all data from the file.

Parameters **full_xhdr** : bool

True to return the full extended headers if they exist padded with ones. False will return a length 1 extended header converted to int32. This is useful when the file contains a customer specified extended header (for example aircraft radar).

Returns **data** : dict of ndarrays

Data arrays of shape=(nsweeps, nrays, nbins) for each data type. Indexed by data type name (str).

metadata : dict of dicts

Arrays of 'azimuth_0', 'azimuth_1', 'elevation_0', 'elevation_1', 'nbins', and 'time' for each data type. Indexed by data type name (str). Rays which were not collected are marked with a value of -1 in the 'nbins' array.

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**()

Return True if bit is set in number.

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.

prf_flag : array

Numerical indication of what PRF was used, 0 for high, 1 for low. Not applicable if dual-PRF is not used during collection.

`pyart.io._sigmetfile.`**`_unpack_ingest_data_header`**`()`

Unpack a single ingest_data_header from record. Return None on error.

`pyart.io._sigmetfile.`**`_unpack_ingest_data_headers`**`()`

Unpack one or more ingest_data_header from a record.

Returns a list of dictionaries or None when an error occurs.

`pyart.io._sigmetfile.`**`_unpack_ingest_header`**`()`

Return a dict with the unpacked ingest_header from the second record.

`pyart.io._sigmetfile.`**`_unpack_key`**`()`

Unpack a key.

`pyart.io._sigmetfile.`**`_unpack_product_hdr`**`()`

Return a dict with the unpacked product_hdr from the first record.

`pyart.io._sigmetfile.`**`_unpack_raw_prod_bhdr`**`()`

Return a dict with the unpacked raw_prod_bhdr from a record.

`pyart.io._sigmetfile.`**`_unpack_structure`**`()`

Unpack a structure

`pyart.io._sigmetfile.`**`bin2_to_angle`**`()`

Return an angle from Sigmet bin2 encoded value (or array).

`pyart.io._sigmetfile.`**`bin4_to_angle`**`()`

Return an angle from Sigmet bin4 encoded value (or array).

`pyart.io._sigmetfile.`**`convert_sigmet_data`**`()`

Convert sigmet data.

CHAPTER TWENTYFOUR

PYART.AUX_IO.ARM_VPT

Routines for reading ARM vertically-pointing radar ingest (e.g., a1) files. These files are characterized by being NetCDF files that do not fully conform to the CF/Radial convention. Nonetheless this module borrows heavily from the existing CF/Radial module.

`pyart.aux_io.arm_vpt.`**`read_kazr`**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*)

Read K-band ARM Zenith Radar (KAZR) NetCDF ingest data.

Parameters filename : str

Name of 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.

CHAPTER

TWENTYFIVE

PYART.AUX_IO.D3R_GCPEX_NC

Routines for reading GCPEX D3R files.

read_d3r_gcpex_nc(filename[, field_names, ...])	Read a D3R GCPEX netCDF file.
_ncvar_to_dict(ncvar)	Convert a NetCDF Dataset variable to a dictionary.

pyart.aux_io.d3r_gcpex_nc.**_ncvar_to_dict**(*ncvar*)

Convert a NetCDF Dataset variable to a dictionary.

pyart.aux_io.d3r_gcpex_nc.**read_d3r_gcpex_nc**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, ***kwargs*)

Read a D3R GCPEX netCDF file.

Parameters filename : str

Name of the ODIM_H5 file to read.

field_names : dict, optional

Dictionary mapping ODIM_H5 field 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 ODIM_H5 file.

CHAPTER

TWENTYSIX

PYART.AUX_IO.EDGE_NECDF

Utilities for reading EDGE NetCDF files.

`read_edge_netcdf`(filename, **kwargs)	Read a EDGE NetCDF file.

`pyart.aux_io.edge_netcdf.`**`read_edge_netcdf`**(*filename*, ***kwargs*)

Read a EDGE NetCDF file.

Parameters filename : str

Name of EDGE NetCDF file to read data from.

Returns radar : Radar

Radar object.

CHAPTER

TWENTYSEVEN

PYART.AUX_IO.READ_GAMIC

Utilities for reading gamic hdf5 files.

`read_gamic`(filename[, field_names, ...])	Read a GAMIC hdf5 file.
`_get_instrument_params`(gfile, filemetadata, ...)	Return a dictionary containing instrument parameters.
`_avg_radial_angles`(angle1, angle2)	Return the average angle between two radial angles.
`_prt_mode_from_unfolding`(unfolding)	Return 'fixed' or 'staggered' depending on unfolding flag

`pyart.aux_io.gamic_hdf5.`**`_avg_radial_angles`**(*angle1*, *angle2*)

Return the average angle between two radial angles.

`pyart.aux_io.gamic_hdf5.`**`_get_instrument_params`**(*gfile*, *filemetadata*, *pulse_width*)

Return a dictionary containing instrument parameters.

`pyart.aux_io.gamic_hdf5.`**`_prt_mode_from_unfolding`**(*unfolding*)

Return 'fixed' or 'staggered' depending on unfolding flag

`pyart.aux_io.gamic_hdf5.`**`read_gamic`**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, *valid_range_from_file=True*, *units_from_file=True*, *pulse_width=None*, ***kwargs*)

Read a GAMIC hdf5 file.

Parameters **filename** : str

Name of GAMIC HDF5 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.

valid_range_from_file : bool, optional

True to extract valid range (valid_min and valid_max) for all field from the file when they are present. False will not extract these parameters.

units_from_file : bool, optional

True to extract the units for all fields from the file when available. False will not extract units using the default units for the fields.

pulse_width : list or None,

Mandatory for gamic radar processors which have pulsewidth enums. pulse_width should contain the pulsewidth' in us.

Returns radar : Radar

Radar object.

CHAPTER TWENTYEIGHT

PYART.AUX_IO.GAMICFILE

GAMICFile class and utility functions.

`GAMICFile`(filename)	A class to read GAMIC files.

`_get_gamic_sweep_data`(group)	Get GAMIC HDF5 sweep data from an HDF5 group.

class `pyart.aux_io.gamicfile.`**`GAMICFile`**(*filename*)

Bases: `object`

A class to read GAMIC files.

Parameters filename : str

Filename of GAMIC HDF5 file.

Attributes

nsweeps	(int) Number of sweeps (or scans) in the file.
rays_per_sweep	(array of int32) Number of rays in each sweep.
total_rays	(int) Total number of rays in all sweeps.
start_ray, end_ray	(array of int32) Index of the first (start) and last (end) ray in each sweep, 0-based.
_hfile	(HDF5 file) Open HDF5 file object from which data is read.
_scans	(list) Name of the HDF5 group for each scan.

Methods

`close`()	Close the file.
`how_attr`(attr, dtype)	Return an array containing a attribute from the how group.
`how_attrs`(attr, dtype)	Return an array of an attribute for each scan's how group.
`how_ext_attrs`(attr)	Return a list of an attribute in each scan's how/extended group.
`is_attr_in_group`(group, attr)	True is attribute is present in the group, False otherwise.
`is_field_in_ray_header`(field)	True if field is present in ray_header, False otherwise.
`is_file_complete`()	True if all scans in file, False otherwise.

Continued on next page

Table 28.3 – continued from previous page

is_file_single_scan_type()	True is all scans are the same scan type, False otherwise.
moment_data(group, dtype)	Read in moment data from all sweeps.
moment_groups()	Return a list of groups under scan0 where moments are stored.
moment_names(scan0_groups)	Return a list of moment names for a list of scan0 groups.
raw_group_attr(group, attr)	Return an attribute from a group with no reformatting.
raw_scan0_group_attr(group, attr)	Return an attribute from the scan0 group with no reformatting.
ray_header(field, dtype)	Return an array containing a ray_header field for each sweep.
sweep_expand(arr[, dtype])	Expand an sweep indexed array to be ray indexed
what_attrs(attr, dtype)	Return a list of an attribute for each scan's what group.
where_attr(attr, dtype)	Return an array containing a attribute from the where group.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyart.aux_io.gamicfile', 'how_ext_attrs': <function GAMICFile.how_ext_a

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*filename*)
initialize object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.aux_io.gamicfile'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

close()
Close the file.

how_attr(*attr*, *dtype*)
Return an array containing a attribute from the how group.

how_attrs(*attr*, *dtype*)
Return an array of an attribute for each scan's how group.

how_ext_attrs(*attr*)
Return a list of an attribute in each scan's how/extended group.

is_attr_in_group(*group*, *attr*)
True is attribute is present in the group, False otherwise.

is_field_in_ray_header(*field*)
True if field is present in ray_header, False otherwise.

is_file_complete()
True if all scans in file, False otherwise.

is_file_single_scan_type()
True is all scans are the same scan type, False otherwise.

moment_data(*group*, *dtype*)
Read in moment data from all sweeps.

moment_groups()
Return a list of groups under scan0 where moments are stored.

moment_names(*scan0_groups*)
Return a list of moment names for a list of scan0 groups.

raw_group_attr(*group*, *attr*)
Return an attribute from a group with no reformatting.

raw_scan0_group_attr(*group*, *attr*)
Return an attribute from the scan0 group with no reformatting.

ray_header(*field*, *dtype*)
Return an array containing a ray_header field for each sweep.

sweep_expand(*arr*, *dtype='float32'*)
Expand an sweep indexed array to be ray indexed

what_attrs(*attr*, *dtype*)
Return a list of an attribute for each scan's what group.

where_attr(*attr*, *dtype*)
Return an array containing a attribute from the where group.

pyart.aux_io.gamicfile.**_get_gamic_sweep_data**(*group*)
Get GAMIC HDF5 sweep data from an HDF5 group.

CHAPTER

TWENTYNINE

PYART.AUX_IO.METRANET

Routines for reading METRANET files. (Used by ELDES www.eldesradar.it)

metranet_read_polar(radar_file[, moment, ...])	Reads a METRANET polar data file
read_metranet(filename[, field_names, ...])	Read a METRANET file.

class `pyart.aux_io.metranet.`**`Header_stru`**

Bases: `_ctypes.Structure`

Class that stores the header of a METRANET POLAR data file contained in a structure used by the C-library reader

C-Structure of METRANET POLAR data:

struct moment_header_struct {

unsigned int record_type; /* data format (moment1) + moment mask */ unsigned int scan_id; unsigned int host_id; unsigned int start_angle; unsigned int end_angle;

unsigned char ant_mode; unsigned char total_sweep; unsigned char current_sweep; /* 1-any number up to 99 / *unsigned char end_of_sweep;* / 0=not end, 1=end sweep, 2=end volume

*/

short sequence; /* ray sequence number in a sweep / *short total_record;* / total ray number in sweep */ short pulses; short num_gates;

int data_bytes; unsigned short data_flag; short data_time_residue; /* data time residue in 0.01 sec / *unsigned int data_time;* / data time in second / *short repeat_time; char compressed;* / flag for compression of data / *char priority;* / for file name use */

float ny_quest; float gate_width; float w_ny_quest; /* may be used for other variable */ float start_range;

};

`__class__`

alias of `PyCStructType`

`__ctypes_from_outparam__``()`

`__delattr__`

Implement delattr(self, name).

`__dict__` **= mappingproxy({'host_id': <Field type=c_ubyte_Array_4, ofs=8, size=4>, 'start_angle': <Field type=c_int, o**

`__dir__``()` → list

default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
Initialize self. See help(type(self)) for accurate signature.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.aux_io.metranet'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__setstate__()

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

`_b_base_`
the base object

`_b_needsfree_`
whether the object owns the memory or not

`_fields_` = [('record_type', <class 'ctypes.c_uint'>), ('scan_id', <class 'pyart.aux_io.metranet.c_ubyte_Array_4'>), ('ho

`_objects`
internal objects tree (NEVER CHANGE THIS OBJECT!)

`ant_mode`
Structure/Union member

`compressed`
Structure/Union member

`current_sweep`
Structure/Union member

`data_bytes`
Structure/Union member

`data_flag`
Structure/Union member

`data_time`
Structure/Union member

`data_time_residue`
Structure/Union member

`end_angle`
Structure/Union member

`end_of_sweep`
Structure/Union member

`gate_width`
Structure/Union member

`host_id`
Structure/Union member

`num_gates`
Structure/Union member

`ny_quest`
Structure/Union member

`priority`
Structure/Union member

`pulses`
Structure/Union member

`record_type`
Structure/Union member

`repeat_time`
Structure/Union member

`scan_id`
Structure/Union member

sequence
Structure/Union member

start_angle
Structure/Union member

start_range
Structure/Union member

total_record
Structure/Union member

total_sweep
Structure/Union member

w_ny_quest
Structure/Union member

class pyart.aux_io.metranet.**Radar_Metranet**(*data=-1, scale=array([0., 0.]), header=(), pol_header=(), moment='ZH'*)

Bases: object

Class containing the information read from the METRANET file

Attributes

type	(str) Information type
data	(matrix) The digital number values
scale	(array) The scale used to convert from digital units to physical units
pol_header	(Header_stru object) Object containing the polar data header
moment	(str) moment name

__class__
alias of *type*

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__dict__': <attribute '__dict__' of 'Radar_Metranet' objects>, '__doc__': '\n Class contain

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*data=-1*, *scale=array([0., 0.])*, *header=()*, *pol_header=()*, *moment='ZH'*)

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.aux_io.metranet'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

type = 'Radar'

class pyart.aux_io.metranet.**Selex_Angle**(*angle=0*, *radiant=False*)
Bases: object

Class used to convert from digital number to angle

Attributes

az	(float) azimuth angle value (degrees or radiants)
el	(float) elevation angle value (degrees or radiants)

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__dict__': <attribute '__dict__' of 'Selex_Angle' objects>, '__doc__': '\n Class used to con

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*angle=0*, *radiant=False*)

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.aux_io.metranet'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

pyart.aux_io.metranet.**metranet_read_polar**(*radar_file*, *moment='ZH'*, *physic_value=True*)
Reads a METRANET polar data file

Parameters **radar_file** : str

file name

moment : str

moment name

physica_value : boolean

If true returns the physical value. Otherwise the digital value.

Returns **ret_data** : Radar_Metranet object

An object containing the information read from the file

pyart.aux_io.metranet.**read_metranet**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, ***kwargs*)
Read a METRANET file.

Parameters **filename** : str

Name of the METRANET file to read.

field_names : dict, optional

Dictionary mapping METRANET field 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 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 METRANET file.

CHAPTER THIRTY

PYART.AUX_IO.NOXP_IPHEX_NC

Routines for reading IPHEx NOXP files.

read_noxp_iphex_nc(filename[, field_names, ...])	Read a NOXP IPHEX netCDF file.
_ncvar_to_dict(ncvar)	Convert a NetCDF Dataset variable to a dictionary.

pyart.aux_io.noxp_iphex_nc.**_ncvar_to_dict**(*ncvar*)

Convert a NetCDF Dataset variable to a dictionary.

pyart.aux_io.noxp_iphex_nc.**read_noxp_iphex_nc**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, ***kwargs*)

Read a NOXP IPHEX netCDF file.

Parameters **filename** : str

Name of the netCDF file to read.

field_names : dict, optional

Dictionary mapping netCDF field 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 netCDF 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 netCDF file.

CHAPTER THIRTYONE

PYART.AUX_IO.ODIM_H5

Routines for reading ODIM_H5 files.

read_odim_h5(filename[, field_names, ...])	Read a ODIM_H5 file.
_to_str(text)	Convert bytes to str if necessary.
_get_odim_h5_sweep_data(group)	Get ODIM_H5 sweet data from an HDF5 group.

`pyart.aux_io.odim_h5.`**`_get_odim_h5_sweep_data`**(*group*)

Get ODIM_H5 sweet data from an HDF5 group.

`pyart.aux_io.odim_h5.`**`_to_str`**(*text*)

Convert bytes to str if necessary.

`pyart.aux_io.odim_h5.`**`read_odim_h5`**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, ***kwargs*)

Read a ODIM_H5 file.

Parameters **filename** : str

Name of the ODIM_H5 file to read.

field_names : dict, optional

Dictionary mapping ODIM_H5 field 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 ODIM_H5 file.

CHAPTER

THIRTYTWO

PYART.AUX_IO.PATTERN

Routines for reading files from the X-band radar from the PATTERN project.

read_pattern(filename, **kwargs)	Read a netCDF file from a PATTERN project X-band radar.

pyart.aux_io.pattern.**read_pattern**(*filename*, ***kwargs*)

Read a netCDF file from a PATTERN project X-band radar.

Parameters **filename** : str

Name of netCDF file to read data from.

Returns **radar** : Radar

Radar object.

CHAPTER

THIRTYTHREE

PYART.AUX_IO.RADX

Reading files using Radx to first convert the file to Cf.Radial format

read_radx(filename[, radx_dir])	Read a file by first converting it to Cf/Radial using Radx-Convert.

`pyart.aux_io.radx.`**`read_radx`**(*filename*, *radx_dir=None*, ***kwargs*)

Read a file by first converting it to Cf/Radial using RadxConvert.

Parameters **filename** : str

Name of file to read using RadxConvert.

radx_dir : str, optional

path to the radx install

Returns **radar** : Radar

Radar object.

CHAPTER

THIRTYFOUR

PYART.AUX_IO.RAINBOW

Routines for reading RAINBOW files (Used by SELEX) using the wradlib library

`read_rainbow_wrl`(filename[, field_names, ...])	Read a RAINBOW file.
`_get_angle`(ray_info[, angle_step, scan_type])	obtains the ray angle start, stop and center
`_get_data`(rawdata, nrays, nbins)	Obtains the raw data
`_get_time`(date_sweep, time_sweep, ...[, ...])	Computes the time at the center of each ray

pyart.aux_io.rainbow_wrl.**_get_angle**(*ray_info*, *angle_step=None*, *scan_type='ppi'*)
obtains the ray angle start, stop and center

Parameters **ray_info** : dictionary of dictionaries

contains the ray info

angle_step : float

Optional. The angle step. Used in case there is no information of angle stop. Otherwise ignored.

scan_type : str

Default ppi. scan_type. Either ppi or rhi.

Returns **moving_angle** : numpy array

the central point of the angle [Deg]

angle_start :

the starting point of the angle [Deg]

angle_stop :

the end point of the angle [Deg]

pyart.aux_io.rainbow_wrl.**_get_data**(*rawdata*, *nrays*, *nbins*)
Obtains the raw data

Parameters **rawdata** : dictionary of dictionaries

contains the raw data information

nrays : int

Number of rays in sweep

nbins : int

Number of bins in ray

Returns **data** : numpy array

the data

pyart.aux_io.rainbow_wrl.**_get_time**(*date_sweep*, *time_sweep*, *first_angle_start*, *last_angle_stop*, *angle_step*, *nrays*, *ant_speed*, *scan_type='ppi'*)

Computes the time at the center of each ray

Parameters **date_sweep, time_sweep** : str

the date and time of the sweep

first_angle_start : float

The starting point of the first angle in the sweep

last_angle_stop : float

The end point of the last angle in the sweep

nrays : int

Number of rays in sweep

ant_speed : float

antenna speed [deg/s]

scan_type : str

Default ppi. scan_type. Either ppi or rhi.

Returns **time_data** : numpy array

the time of each ray

sweep_start_epoch : float

sweep start time in seconds since 1.1.1970

pyart.aux_io.rainbow_wrl.**read_rainbow_wrl**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, ***kwargs*)

Read a RAINBOW file. This routine has been tested to read rainbow5 files version 5.22.3, 5.34.16 and 5.35.1. Since the rainbow file format is evolving constanly there is no guaranty that it can work with other versions. If necessary, the user should adapt to code according to its own file version.

Data types read by this routine: Reflectivity: dBZ, dBuZ, dBZv, dBuZv Velocity: V, Vu, Vv, Vvu Spectrum width: W, Wu, Wv, Wvu Differential reflectivity: ZDR, ZDRu Co-polar correlation coefficient: RhoHV, RhoHVu Co-polar differential phase: PhiDP, uPhiDP, uPhiDPu Specific differential phase: KDP, uKDP, uKDPu Signal quality parameters: SQI, SQIu, SQIv, SQIvu Temperature: TEMP Position of the range bin respect to the ISO0: ISO0 radar visibility according to Digital Elevation Model (DEM): VIS

Parameters **filename** : str

Name of the RAINBOW file to read.

field_names : dict, optional

Dictionary mapping RAINBOW field 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 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 RAINBOW file.

CHAPTER

THIRTYFIVE

PYART.AUX_IO.SINARAME_H5

Routines for reading sinarame_H5 files.

read_sinarame_h5(filename[, field_names, ...])	Read a SINARAME_H5 file.
write_sinarame_cfradial(path)	This function takes SINARAME_H5 files (where every file has only one field and one volume) from a folder and writes a CfRadial file for each volume including all fields.
_to_str(text)	Convert bytes to str if necessary.
_get_SINARAME_h5_sweep_data(group)	Get SINARAME_H5 sweet data from an HDF5 group.

pyart.aux_io.sinarame_h5.**_get_SINARAME_h5_sweep_data**(*group*)

Get SINARAME_H5 sweet data from an HDF5 group.

pyart.aux_io.sinarame_h5.**_to_str**(*text*)

Convert bytes to str if necessary.

pyart.aux_io.sinarame_h5.**read_sinarame_h5**(*filename*, *field_names=None*, *additional_metadata=None*, *file_field_names=False*, *exclude_fields=None*, ***kwargs*)

Read a SINARAME_H5 file.

Parameters filename : str

Name of the SINARAME_H5 file to read.

field_names : dict, optional

Dictionary mapping SINARAME_H5 field 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 SINARAME_H5 file.

`pyart.aux_io.sinarame_h5.`**`write_sinarame_cfradial`**(*path*)

This function takes SINARAME_H5 files (where every file has only one field and one volume) from a folder and writes a CfRadial file for each volume including all fields.

Parameters path : str

Where the SINARAME_H5 files are.

CHAPTER

THIRTYSIX

PYART.CORE.GRID

An class for holding gridded Radar data.

Grid(time, fields, metadata, ...[, ...])	A class for storing rectilinear gridded radar data in Cartesian coordinate.

_point_data_factory(grid, coordinate)	Return a function which returns the locations of all points.
_point_lon_lat_data_factory(grid, coordinate)	Return a function which returns the geographic locations of points.
_point_altitude_data_factory(grid)	Return a function which returns the point altitudes.

class `pyart.core.grid.`**`Grid`**(*time*, *fields*, *metadata*, *origin_latitude*, *origin_longitude*, *origin_altitude*, *x*, *y*, *z*, *projection=None*, *radar_latitude=None*, *radar_longitude=None*, *radar_altitude=None*, *radar_time=None*, *radar_name=None*)

Bases: `object`

A class for storing rectilinear gridded radar data in Cartesian coordinate.

Refer to the attribute section for information on the parameters.

To create a Grid object using legacy parameters present in Py-ART version 1.5 and before, use `from_legacy_parameters()`, grid = Grid.from_legacy_parameters(fields, axes, metadata).

Attributes

time	(dict) Time of the grid.
fields: dict of dicts	Moments from radars or other variables.
metadata: dict	Metadata describing the grid.
origin_longitude, origin_latitude, origin_altitude	(dict) Geographic coordinate of the origin of the grid.
x, y, z	(dict, 1D) Distance from the grid origin for each Cartesian coordinate axis in a one dimensional array. Defines the spacing along the three grid axes which is repeated throughout the grid, making a rectilinear grid.
nx, ny, nz	(int) Number of grid points along the given Cartesian dimension.
projection	(dic or str) Projection parameters defining the map projection used to transform from Cartesian to geographic coordinates. None will use the default dictionary with the 'proj' key set to 'pyart_aeqd' indicating that the native Py-ART azimuthal equidistant projection is used. Other values should specify a valid pyproj.Proj projparams dictionary or string. The special key '_include_lon_0_lat_0' is removed when interpreting this dictionary. If this key is present and set to True, which is required when proj='pyart_aeqd', then the radar longitude and latitude will be added to the dictionary as 'lon_0' and 'lat_0'. Use the `get_projparams()` method to retrieve a copy of this attribute dictionary with this special key evaluated.
radar_longitude, radar_latitude, radar_altitude	(dict or None, optional) Geographic location of the radars which make up the grid.
radar_time	(dict or None, optional) Start of collection for the radar which make up the grid.
radar_name	(dict or None, optional) Names of the radars which make up the grid.
nradar	(int) Number of radars whose data was used to make the grid.
projection_proj	(Proj) pyproj.Proj instance for the projection specified by the projection attribute. If the 'pyart_aeqd' projection is specified accessing this attribute will raise a ValueError.
point_x, point_y, point_z	(LazyLoadDict) The Cartesian locations of all grid points from the origin in the three Cartesian coordinates. The three dimensional data arrays contained these attributes are calculated from the x, y, and z attributes. If these attributes are changed use :py:func: *init_point_x_y_z* to reset the attributes.
point_longitude, point_latitude	(LazyLoadDict) Geographic location of each grid point. The projection parameter(s) defined in the *projection* attribute are used to perform an inverse map projection from the Cartesian grid point locations relative to the grid origin. If these attributes are changed use `init_point_longitude_latitude()` to reset the attributes.
point_altitude	(LazyLoadDict) The altitude of each grid point as calculated from the altitude of the grid origin and the Cartesian z location of each grid point. If this attribute is changed use `init_point_altitude()` to reset the attribute.

Methods

`add_field`(field_name, field_dict[, ...])	Add a field to the object.
`get_point_longitude_latitude`([level, edges])	Return arrays of longitude and latitude for a given grid height level.
`get_projparams`()	Return a projparam dict from the projection attribute.

Continued on next page

Table 36.3 – continued from previous page

init_point_altitude()	Initialize the point_altitude attribute.
init_point_longitude_latitude()	Initialize or reset the point_{longitude, latitudes} attributes.
init_point_x_y_z()	Initialize or reset the point_{x, y, z} attributes.
write(filename[, format, arm_time_variables])	Write the the Grid object to a NetCDF file.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` = mappingproxy({'__doc__': "\n A class for storing rectilinear gridded radar data in Cartesian coordinate.\n

`__dir__``()` → list
default dir() implementation

`__eq__`
Return self==value.

`__format__``()`
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__getstate__``()`
Return object's state which can be pickled.

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__`(*time*, *fields*, *metadata*, *origin_latitude*, *origin_longitude*, *origin_altitude*, *x*, *y*, *z*, *projection=None*, *radar_latitude=None*, *radar_longitude=None*, *radar_altitude=None*, *radar_time=None*, *radar_name=None*)
Initalize object.

`__le__`
Return self<=value.

`__lt__`
Return self<value.

`__module__` = 'pyart.core.grid'

`__ne__`
Return self!=value.

`__new__``()`
Create and return a new object. See help(type) for accurate signature.

`__reduce__``()`
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__setstate__(*state*)
Restore unpicklable entries from pickled object.

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_find_and_check_nradar()
Return the number of radars which were used to create the grid.

Examine the radar attributes to determine the number of radars which were used to create the grid. If the size of the radar attributes are inconsistent a ValueError is raised by this method.

add_field(*field_name*, *field_dict*, *replace_existing=False*)
Add a field to the object.

Parameters **field_name** : str

Name of the field to the fields dictionary.

field_dict : dict

Dictionary containing field data and metadata.

replace_existing : bool, optional

True to replace the existing field with key field_name if it exists, overwriting the existing data. If False, a ValueError is raised if field_name already exists.

get_point_longitude_latitude(*level=0*, *edges=False*)
Return arrays of longitude and latitude for a given grid height level.

Parameters **level** : int, optional

Grid height level at which to determine latitudes and longitudes. This is not currently used as all height level have the same layout.

edges : bool, optional

True to calculate the latitude and longitudes of the edges by interpolating between Cartesian coordinates points and extrapolating at the boundaries. False to calculate the locations at the centers.

Returns **longitude, latitude** : 2D array

Arrays containing the latitude and longitudes, in degrees, of the grid points or edges between grid points for the given height.

get_projparams()
Return a projparam dict from the projection attribute.

init_point_altitude()
Initialize the point_altitude attribute.

init_point_longitude_latitude()
Initialize or reset the point_{longitude, latitudes} attributes.

init_point_x_y_z()
Initialize or reset the point_{x, y, z} attributes.

projection_proj

write(*filename*, *format='NETCDF4'*, *arm_time_variables=False*)
Write the the Grid object to a NetCDF file.

Parameters **filename** : str

Filename to save to.

format : str, optional

NetCDF format, one of 'NETCDF4', 'NETCDF4_CLASSIC', 'NETCDF3_CLASSIC' or 'NETCDF3_64BIT'.

arm_time_variables : bool

True to write the ARM standard time variables base_time and time_offset. False will not write these variables.

pyart.core.grid.**_point_altitude_data_factory**(*grid*)
Return a function which returns the point altitudes.

pyart.core.grid.**_point_data_factory**(*grid*, *coordinate*)
Return a function which returns the locations of all points.

pyart.core.grid.**_point_lon_lat_data_factory**(*grid*, *coordinate*)
Return a function which returns the geographic locations of points.

CHAPTER

THIRTYSEVEN

PYART.CORE.RADAR

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

_rays_per_sweep_data_factory(radar)	Return a function which returns the number of rays per sweep.
_gate_data_factory(radar, coordinate)	Return a function which returns the Cartesian locations of gates.
_gate_lon_lat_data_factory(radar, coordinate)	Return a function which returns the geographic locations of gates.
_gate_altitude_data_factory(radar)	Return a function which returns the gate altitudes.

Radar(time, _range, fields, metadata, ...[, ...])	A class for storing antenna coordinate radar data.

class `pyart.core.radar.`**`Radar`**(*time*, *_range*, *fields*, *metadata*, *scan_type*, *latitude*, *longitude*, *altitude*, *sweep_number*, *sweep_mode*, *fixed_angle*, *sweep_start_ray_index*, *sweep_end_ray_index*, *azimuth*, *elevation*, *altitude_agl=None*, *target_scan_rate=None*, *rays_are_indexed=None*, *ray_angle_res=None*, *scan_rate=None*, *antenna_transition=None*, *instrument_parameters=None*, *radar_calibration=None*, *rotation=None*, *tilt=None*, *roll=None*, *drift=None*, *heading=None*, *pitch=None*, *georefs_applied=None*)

Bases: `object`

A class for storing antenna coordinate radar data.

The structure of the Radar class is based on the CF/Radial Data file format. Global attributes and variables (section 4.1 and 4.3) are represented as a dictionary in the metadata attribute. Other required and optional variables are represented as dictionaries in a attribute with the same name as the variable in the CF/Radial standard. When a optional attribute not present the attribute has a value of None. The data for a given variable is stored in the dictionary under the 'data' key. Moment field data is stored as a dictionary of dictionaries in the fields attribute. Sub-convention variables are stored as a dictionary of dictionaries under the meta_group attribute.

Refer to the attribute section for information on the parameters.

Attributes

time	(dict) Time at the center of each ray.
range	(dict) Range to the center of each gate (bin).

fields	(dict of dicts) Moment fields.
metadata	(dict) Metadata describing the instrument and data.
scan_type	(str) Type of scan, one of 'ppi', 'rhi', 'sector' or 'other'. If the scan volume contains multiple sweep m
latitude	(dict) Latitude of the instrument.
longitude	(dict) Longitude of the instrument.
altitude	(dict) Altitude of the instrument, above sea level.
altitude_agl	(dict or None) Altitude of the instrument above ground level. If not provided this attribute is set to No
sweep_number	(dict) The number of the sweep in the volume scan, 0-based.
sweep_mode	(dict) Sweep mode for each mode in the volume scan.
fixed_angle	(dict) Target angle for thr sweep. Azimuth angle in RHI modes, elevation angle in all other modes.
sweep_start_ray_index	(dict) Index of the first ray in each sweep relative to the start of the volume, 0-based.
sweep_end_ray_index	(dict) Index of the last ray in each sweep relative to the start of the volume, 0-based.
rays_per_sweep	(LazyLoadDict) Number of rays in each sweep. The data key of this attribute is create upon first acces
target_scan_rate	(dict or None) Intended scan rate for each sweep. If not provided this attribute is set to None, indicatin
rays_are_indexed	(dict or None) Indication of whether ray angles are indexed to a regular grid in each sweep. If not prov
ray_angle_res	(dict or None) If rays_are_indexed is not None, this provides the angular resolution of the grid. If not
azimuth	(dict) Azimuth of antenna, relative to true North.
elevation	(dict) Elevation of antenna, relative to the horizontal plane.
gate_x, gate_y, gate_z	(LazyLoadDict) Location of each gate in a Cartesian coordinate system assuming a standard atmosphe
gate_longitude, gate_latitude	(LazyLoadDict) Geographic location of each gate. The projection parameter(s) defined in the *projecti*
projection	(dic or str) Projection parameters defining the map projection used to transform from Cartesian to geo
gate_altitude	(LazyLoadDict) The altitude of each radar gate as calculated from the altitude of the radar and the Car
scan_rate	(dict or None) Actual antenna scan rate. If not provided this attribute is set to None, indicating this par
antenna_transition	(dict or None) Flag indicating if the antenna is in transition, 1 = yes, 0 = no. If not provided this attrib
rotation	(dict or None) The rotation angle of the antenna. The angle about the aircraft longitudinal axis for a ve
tilt	(dict or None) The tilt angle with respect to the plane orthogonal (Z-axis) to aircraft longitudinal axis.
roll	(dict or None) The roll angle of platform, for aircraft right wing down is positive.
drift	(dict or None) Drift angle of antenna, the angle between heading and track.
heading	(dict or None) Heading (compass) angle, clockwise from north.
pitch	(dict or None) Pitch angle of antenna, for aircraft nose up is positive.
georefs_applied	(dict or None) Indicates whether the variables have had georeference calculation applied. Leading to E
instrument_parameters	(dict of dicts or None) Instrument parameters, if not provided this attribute is set to None, indicating th
radar_calibration	(dict of dicts or None) Instrument calibration parameters. If not provided this attribute is set to None, i
ngates	(int) Number of gates (bins) in a ray.
nrays	(int) Number of rays in the volume.
nsweeps	(int) Number of sweep in the volume.

Methods

add_field(field_name, dic[, replace_existing])	Add a field to the object.
add_field_like(existing_field_name, ...[, ...])	Add a field to the object with metadata from a existing field.
check_field_exists(field_name)	Check that a field exists in the fields dictionary.
extract_sweeps(sweeps)	Create a new radar contains only the data from select sweeps.
get_azimuth(sweep[, copy])	Return an array of azimuth angles for a given sweep.
get_elevation(sweep[, copy])	Return an array of elevation angles for a given sweep.
get_end(sweep)	Return the ending ray for a given sweep.
get_field(sweep, field_name[, copy])	Return the field data for a given sweep.

Continued on next page

Table 37.4 – continued from previous page

get_gate_x_y_z(sweep[, edges, ...])	Return the x, y and z gate locations in meters for a given sweep.
get_nyquist_vel(sweep[, check_uniform])	Return the Nyquist velocity in meters per second for a given sweep.
get_slice(sweep)	Return a slice for selecting rays for a given sweep.
get_start(sweep)	Return the starting ray index for a given sweep.
get_start_end(sweep)	Return the starting and ending ray for a given sweep.
info([level, out])	Print information on radar.
init_gate_altitude()	Initialize the gate_altitude attribute.
init_gate_longitude_latitude()	Initialize or reset the gate_longitude and gate_latitude attributes.
init_gate_x_y_z()	Initialize or reset the gate_{x, y, z} attributes.
init_rays_per_sweep()	Initialize or reset the rays_per_sweep attribute.
iter_azimuth()	Return an iterator which returns sweep azimuth data.
iter_elevation()	Return an iterator which returns sweep elevation data.
iter_end()	Return an iterator over the sweep end indices.
iter_field(field_name)	Return an iterator which returns sweep field data.
iter_slice()	Return an iterator which returns sweep slice objects.
iter_start()	Return an iterator over the sweep start indices.
iter_start_end()	Return an iterator over the sweep start and end indices.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'get_field': <function Radar.get_field>, 'get_start_end': <function Radar.get_start_end>, '_

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__getstate__()
Return object's state which can be pickled.

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*time*, *_range*, *fields*, *metadata*, *scan_type*, *latitude*, *longitude*, *altitude*, *sweep_number*, *sweep_mode*, *fixed_angle*, *sweep_start_ray_index*, *sweep_end_ray_index*, *azimuth*, *elevation*, *altitude_agl=None*, *target_scan_rate=None*, *rays_are_indexed=None*, *ray_angle_res=None*, *scan_rate=None*, *antenna_transition=None*, *instrument_parameters=None*, *radar_calibration=None*, *rotation=None*, *tilt=None*, *roll=None*, *drift=None*, *heading=None*, *pitch=None*, *georefs_applied=None*)

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.core.radar'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__setstate__(*state*)
Restore unpicklable entries from pickled object.

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_check_sweep_in_range(*sweep*)
Check that a sweep number is in range.

_dic_info(*attr*, *level*, *out*, *dic=None*, *ident_level=0*)
Print information on a dictionary attribute.

add_field(*field_name*, *dic*, *replace_existing=False*)
Add a field to the object.

Parameters field_name : str

Name of the field to add to the dictionary of fields.

dic : dict

Dictionary contain field data and metadata.

replace_existing : bool

True to replace the existing field with key field_name if it exists, loosing any existing data. False will raise a ValueError when the field already exists.

add_field_like(*existing_field_name*, *field_name*, *data*, *replace_existing=False*)

Add a field to the object with metadata from a existing field.

Note that the data parameter is not copied by this method. If data refers to a 'data' array from an existing field dictionary, a copy should be made within or prior to using this method. If this is not done the 'data' key in both field dictionaries will point to the same NumPy array and modification of one will change the second. To copy NumPy arrays use the copy() method. See the Examples section for how to create a copy of the 'reflectivity' field as a field named 'reflectivity_copy'.

Parameters **existing_field_name** : str

Name of an existing field to take metadata from when adding the new field to the object.

field_name : str

Name of the field to add to the dictionary of fields.

data : array

Field data. A copy of this data is not made, see the note above.

replace_existing : bool

True to replace the existing field with key field_name if it exists, loosing any existing data. False will raise a ValueError when the field already exists.

Examples

```
>>> radar.add_field_like('reflectivity', 'reflectivity_copy',
...                      radar.fields['reflectivity']['data'].copy())
```

check_field_exists(*field_name*)

Check that a field exists in the fields dictionary.

If the field does not exist raise a KeyError.

Parameters **field_name** : str

Name of field to check.

extract_sweeps(*sweeps*)

Create a new radar contains only the data from select sweeps.

Parameters **sweeps** : array_like

Sweeps (0-based) to include in new Radar object.

Returns **radar** : Radar

Radar object which contains a copy of data from the selected sweeps.

get_azimuth(*sweep*, *copy=False*)

Return an array of azimuth angles for a given sweep.

Parameters **sweep** : int

Sweep number to retrieve data for, 0 based.

copy : bool, optional

True to return a copy of the azimuths. False, the default, returns a view of the azimuths (when possible), changing this data will change the data in the underlying Radar object.

Returns azimuths : array

Array containing the azimuth angles for a given sweep.

get_elevation(*sweep*, *copy=False*)

Return an array of elevation angles for a given sweep.

Parameters sweep : int

Sweep number to retrieve data for, 0 based.

copy : bool, optional

True to return a copy of the elevations. False, the default, returns a view of the elevations (when possible), changing this data will change the data in the underlying Radar object.

Returns azimuths : array

Array containing the elevation angles for a given sweep.

get_end(*sweep*)

Return the ending ray for a given sweep.

get_field(*sweep*, *field_name*, *copy=False*)

Return the field data for a given sweep.

When used with `get_gate_x_y_z()` this method can be used to obtain the data needed for plotting a radar field with the correct spatial context.

Parameters sweep : int

Sweep number to retrieve data for, 0 based.

field_name : str

Name of the field from which data should be retrieved.

copy : bool, optional

True to return a copy of the data. False, the default, returns a view of the data (when possible), changing this data will change the data in the underlying Radar object.

Returns data : array

Array containing data for the requested sweep and field.

get_gate_x_y_z(*sweep*, *edges=False*, *filter_transitions=False*)

Return the x, y and z gate locations in meters for a given sweep.

With the default parameter this method returns the same data as contained in the gate_x, gate_y and gate_z attributes but this method performs the gate location calculations only for the specified sweep and therefore is more efficient than accessing this data through these attribute.

When used with `get_field()` this method can be used to obtain the data needed for plotting a radar field with the correct spatial context.

Parameters sweep : int

Sweep number to retrieve gate locations from, 0 based.

edges : bool, optional

True to return the locations of the gate edges calculated by interpolating between the range, azimuths and elevations. False (the default) will return the locations of the gate centers with no interpolation.

filter_transitions : bool, optional

True to remove rays where the antenna was in transition between sweeps. False will include these rays. No rays will be removed if the antenna_transition attribute is not available (set to None).

Returns **x, y, z** : 2D array

Array containing the x, y and z, distances from the radar in meters for the center (or edges) for all gates in the sweep.

get_nyquist_vel(*sweep*, *check_uniform=True*)
Return the Nyquist velocity in meters per second for a given sweep.

Raises a LookupError if the Nyquist velocity is not available, an Exception is raised if the velocities are not uniform in the sweep unless check_uniform is set to False.

Parameters **sweep** : int

Sweep number to retrieve data for, 0 based.

check_uniform : bool

True to check to perform a check on the Nyquist velocities that they are uniform in the sweep, False will skip this check and return the velocity of the first ray in the sweep.

Returns **nyquist_velocity** : float

Array containing the Nyquist velocity in m/s for a given sweep.

get_slice(*sweep*)
Return a slice for selecting rays for a given sweep.

get_start(*sweep*)
Return the starting ray index for a given sweep.

get_start_end(*sweep*)
Return the starting and ending ray for a given sweep.

info(*level='standard'*, *out=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Print information on radar.

Parameters **level** : {'compact', 'standard', 'full', 'c', 's', 'f'}

Level of information on radar object to print, compact is minimal information, standard more and full everything.

out : file-like

Stream to direct output to, default is to print information to standard out (the screen).

init_gate_altitude()
Initialize the gate_altitude attribute.

init_gate_longitude_latitude()
Initialize or reset the gate_longitude and gate_latitude attributes.

init_gate_x_y_z()
Initialize or reset the gate_{x, y, z} attributes.

init_rays_per_sweep()
Initialize or reset the rays_per_sweep attribute.

iter_azimuth()
Return an iterator which returns sweep azimuth data.

iter_elevation()
Return an iterator which returns sweep elevation data.

iter_end()
Return an iterator over the sweep end indices.

iter_field(*field_name*)
Return an iterator which returns sweep field data.

iter_slice()
Return an iterator which returns sweep slice objects.

iter_start()
Return an iterator over the sweep start indices.

iter_start_end()
Return an iterator over the sweep start and end indices.

pyart.core.radar.**_gate_altitude_data_factory**(*radar*)
Return a function which returns the gate altitudes.

pyart.core.radar.**_gate_data_factory**(*radar*, *coordinate*)
Return a function which returns the Cartesian locations of gates.

pyart.core.radar.**_gate_lon_lat_data_factory**(*radar*, *coordinate*)
Return a function which returns the geographic locations of gates.

pyart.core.radar.**_rays_per_sweep_data_factory**(*radar*)
Return a function which returns the number of rays per sweep.

CHAPTER

THIRTYEIGHT

PYART.CORE.WIND_PROFILE

Storage of wind profiles.

`HorizontalWindProfile`(height, speed, direction)	Horizontal wind profile.

class `pyart.core.wind_profile.`**`HorizontalWindProfile`**(*height*, *speed*, *direction*, *latitude=None*, *longitude=None*)

Bases: `object`

Horizontal wind profile.

Parameters **height** : array-like, 1D

Heights in meters above sea level at which horizontal winds were sampled.

speed : array-like, 1D

Horizontal wind speed in meters per second at each height sampled.

direction : array-like, 1D

Horizontal wind direction in degrees at each height sampled.

Other Parameters **latitude** : array-like, 1D, optional

Latitude in degrees north at each height sampled.

longitude : array-like, 1D, optional

Longitude in degrees east at each height sampled.

Attributes

u_wind	U component of horizontal wind in meters per second.
v_wind	V component of horizontal wind in meters per second.

height	(array, 1D) Heights in meters above sea level at which horizontal winds were sampled.
speed	(array, 1D) Horizontal wind speed in meters per second at each height.
direction	(array, 1D) Horizontal wind direction in degrees at each height.

Methods

from_u_and_v(height, u_wind, v_wind)	Create a HorizontalWindProfile instance from U and V components.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__doc__': '\n Horizontal wind profile.\n\n Parameters\n ———-\n height : array-like, 1D\n

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*height*, *speed*, *direction*, *latitude=None*, *longitude=None*)
initialize

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.core.wind_profile'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_parse_location_data(*latitude*, *longitude*)
Parse profile location data.

classmethod from_u_and_v(*height*, *u_wind*, *v_wind*)
Create a HorizontalWindProfile instance from U and V components.

Parameters **height** : array-like, 1D

Heights in meters above sea level at which horizontal winds were sampled.

u_wind : array-like, 1D

U component of horizontal wind speed in meters per second.

v_wind : array-like, 1D

V component of horizontal wind speed in meters per second.

u_wind
U component of horizontal wind in meters per second.

v_wind
V component of horizontal wind in meters per second.

CHAPTER

THIRTYNINE

PYART.BRIDGE.WRADLIB

Py-ART methods linking to wradlib functions, http://wradlib.bitbucket.org/

`texture_of_complex_phase`(radar[, ...])	Calculate the texture of the differential phase field.

`pyart.bridge.wradlib_bridge.`**`texture_of_complex_phase`**(*radar*, *phidp_field=None*, *phidp_texture_field=None*)

Calculate the texture of the differential phase field.

Calculate the texture of the real part of the complex differential phase field

Parameters **radar** : Radar

Radar object from which to .

phidp_field : str, optional

Name of field in radar which contains the differential phase shift. None will use the default field name in the Py-ART configuration file.

phidp_texture_field : str, optional

Name to use for the differential phase texture field metadata. None will use the default field name in the Py-ART configuration file.

Returns **texture_field** : dict

Field dictionary containing the texture of the real part of the complex differential phase.

References

Gourley, J. J., P. Tabary, and J. Parent du Chatelet, A fuzzy logic algorithm for the separation of precipitating from nonprecipitating echoes using polarimetric radar observations, Journal of Atmospheric and Oceanic Technology 24 (8), 1439-1451

CHAPTER

FORTY

PYART.CORRECT.FILTERS

Functions for creating gate filters (masks) which can be used it various corrections routines in Py-ART.

`moment_based_gate_filter`(radar[, ncp_field, ...])	Create a filter which removes undesired gates based on moments.
`moment_and_texture_based_gate_filter`(radar)	Create a filter which removes undesired gates based on texture of moments.
`snr_based_gate_filter`(radar[, snr_field, ...])	Create a filter which removes undesired gates based on SNR.
`visibility_based_gate_filter`(radar[, ...])	Create a filter which removes undesired gates based on visibility.
`temp_based_gate_filter`(radar[, temp_field, ...])	Create a filter which removes undesired gates based on temperature.

`GateFilter`(radar[, exclude_based])	A class for building a boolean arrays for filtering gates based on a set of condition typically based on the values in the radar fields.

class `pyart.filters.gatefilter.`**`GateFilter`**(*radar*, *exclude_based=True*)

Bases: `object`

A class for building a boolean arrays for filtering gates based on a set of condition typically based on the values in the radar fields. These filter can be used in various algorithms and calculations within Py-ART.

See `pyart.correct.GateFilter.exclude_below()` for method parameter details.

Parameters radar : Radar

Radar object from which gate filter will be build.

exclude_based : bool, optional

True, the default and suggested method, will begin with all gates included and then use the exclude methods to exclude gates based on conditions. False will begin with all gates excluded from which a set of gates to include should be set using the include methods.

Examples

```
>>> import pyart
>>> radar = pyart.io.read('radar_file.nc')
>>> gatefilter = pyart.correct.GateFilter(radar)
```

```
>>> gatefilter.exclude_below('reflectivity', 10)
>>> gatefilter.exclude_below('normalized_coherent_power', 0.75)
```

Attributes

gate_excluded	(array, dtype=bool) Boolean array indicating if a gate should be excluded from a calculation. Elements marked True indicate the corresponding gate should be excluded. Those marked False should be included. This is read-only attribute, any changes to the array will NOT be reflected in gate_included and will be lost when the attribute is accessed again.
gate_included	(array, dtype=bool) Boolean array indicating if a gate should be included in a calculation. Elements marked True indicate the corresponding gate should be include. Those marked False should be excluded. This is read-only attribute, any changes to the array will NOT be reflected in gate_excluded and will be lost when the attribute is accessed again.

Methods

copy()	Return a copy of the gatefilter.
exclude_above(field, value[, ...])	Exclude gates where a given field is above a given value.
exclude_all()	Exclude all gates.
exclude_below(field, value[, ...])	Exclude gates where a given field is below a given value.
exclude_equal(field, value[, exclude_masked, op])	Exclude gates where a given field is equal to a value.
exclude_gates(mask[, exclude_masked, op])	Exclude gates where a given mask is equal True.
exclude_inside(field, v1, v2[, ...])	Exclude gates where a given field is inside a given interval.
exclude_invalid(field[, exclude_masked, op])	Exclude gates where an invalid value occurs in a field (NaNs or infs).
exclude_masked(field[, exclude_masked, op])	Exclude gates where a given field is masked.
exclude_none()	Exclude no gates, include all gates.
exclude_not_equal(field, value[, ...])	Exclude gates where a given field is not equal to a value.
exclude_outside(field, v1, v2[, ...])	Exclude gates where a given field is outside a given interval.
exclude_transition([trans_value, ...])	Exclude all gates in rays marked as in transition between sweeps.
include_above(field, value[, ...])	Include gates where a given field is above a given value.
include_all()	Include all gates.
include_below(field, value[, ...])	Include gates where a given field is below a given value.
include_equal(field, value[, exclude_masked, op])	Include gates where a given field is equal to a value.
include_gates(mask[, exclude_masked, op])	Include gates where a given mask is equal True.
include_inside(field, v1, v2[, ...])	Include gates where a given field is inside a given interval.
include_none()	Include no gates, exclude all gates.
include_not_equal(field, value[, ...])	Include gates where a given field is not equal to a value.
include_not_masked(field[, exclude_masked, op])	Include gates where a given field in not masked.
include_not_transition([trans_value, ...])	Include all gates in rays not marked as in transition between sweeps.
include_outside(field, v1, v2[, ...])	Include gates where a given field is outside a given interval.

Continued on next page

Table 40.3 – continued from previous page

include_valid(field[, exclude_masked, op])	Include gates where a valid value occurs in a field (not NaN or inf).

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'include_above': <function GateFilter.include_above>, '__module__': 'pyart.filters.gatefilte

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*radar*, *exclude_based=True*)
initialize

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.filters.gatefilter'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_get_fdata(*field*)
Check that the field exists and retrieve field data.

_merge(*marked*, *op*, *exclude_masked*)
Merge an array of marked gates with the exclude array.

copy()
Return a copy of the gatefilter.

exclude_above(*field*, *value*, *exclude_masked=True*, *op='or'*, *inclusive=False*)
Exclude gates where a given field is above a given value.

exclude_all()
Exclude all gates.

exclude_below(*field*, *value*, *exclude_masked=True*, *op='or'*, *inclusive=False*)
Exclude gates where a given field is below a given value.

Parameters **field** : str

Name of field compared against the value.

value : float

Gates with a value below this value in the specified field will be marked for exclusion in the filter.

exclude_masked : bool, optional

True to filter masked values in the specified field if the data is a masked array, False to include any masked values.

op : {'and', 'or', 'new'}

Operation to perform when merging the existing set of excluded gates with the excluded gates from the current operation. 'and' will perform a logical AND operation, 'or' a logical OR, and 'new' will replace the existing excluded gates with the one generated here. 'or', the default for exclude methods, is typically desired when building up a set of conditions for excluding gates where the desired effect is to exclude gates which meet any of the conditions. 'and', the default for include methods, is typically desired when building up a set of conditions where the desired effect is to include gates which meet any of the conditions. Note that the 'and' method MAY results in including gates which have previously been excluded because they were masked or invalid.

inclusive : bool

Indicates whether the specified value should also be excluded.

exclude_equal (*field*, *value*, *exclude_masked=True*, *op='or'*)
Exclude gates where a given field is equal to a value.

exclude_gates (*mask*, *exclude_masked=True*, *op='or'*)
Exclude gates where a given mask is equal True.

Parameters **mask** : numpy array

Boolean numpy array with same shape as a field array.

exclude_masked : bool, optional

True to filter masked values in the specified mask if it is a masked array, False to include any masked values.

op : {'and', 'or', 'new'}

Operation to perform when merging the existing set of excluded gates with the excluded gates from the current operation. 'and' will perform a logical AND operation, 'or' a logical OR, and 'new' will replace the existing excluded gates with the one generated here. 'or', the default for exclude methods, is typically desired when building up a set of conditions for excluding gates where the desired effect is to exclude gates which meet any of the conditions. 'and', the default for include methods, is typically desired when building up a set of conditions where the desired effect is to include gates which meet any of the conditions. Note that the 'and' method MAY results in including gates which have previously been excluded because they were masked or invalid.

exclude_inside (*field*, *v1*, *v2*, *exclude_masked=True*, *op='or'*, *inclusive=True*)
Exclude gates where a given field is inside a given interval.

exclude_invalid (*field*, *exclude_masked=True*, *op='or'*)
Exclude gates where an invalid value occurs in a field (NaNs or infs).

exclude_masked (*field*, *exclude_masked=True*, *op='or'*)
Exclude gates where a given field is masked.

exclude_none ()
Exclude no gates, include all gates.

exclude_not_equal (*field*, *value*, *exclude_masked=True*, *op='or'*)
Exclude gates where a given field is not equal to a value.

exclude_outside (*field*, *v1*, *v2*, *exclude_masked=True*, *op='or'*, *inclusive=False*)
Exclude gates where a given field is outside a given interval.

exclude_transition (*trans_value=1*, *exclude_masked=True*, *op='or'*)
Exclude all gates in rays marked as in transition between sweeps.

Exclude all gates in rays marked as "in transition" by the antenna_transition attribute of the radar used to construct the filter. If no antenna transition information is available no gates are excluded.

Parameters **trans_value** : int, optional

Value used in the antenna transition data to indicate that the instrument was between sweeps (in transition) during the collection of a specific ray. Typically a value of 1 is used to indicate this transition and the default can be used in these cases.

exclude_masked : bool, optional

True to filter masked values in antenna_transition if the data is a masked array, False to include any masked values.

op : {'and', 'or', 'new'}

Operation to perform when merging the existing set of excluded gates with the excluded gates from the current operation. 'and' will perform a logical AND operation, 'or' a logical OR, and 'new' will replace the existing excluded gates with the one generated here. 'or', the default for exclude methods, is typically desired when building up a set of conditions for excluding gates where the desired effect is to exclude gates which meet any of the conditions. 'and', the default for include methods, is typically desired when building up a set of conditions where the desired effect is to include gates which meet any of the conditions. Note that the 'and' method MAY results in including gates which have previously been excluded because they were masked or invalid.

gate_excluded

gate_included

include_above (*field*, *value*, *exclude_masked=True*, *op='and'*, *inclusive=False*)
Include gates where a given field is above a given value.

include_all ()
Include all gates.

include_below (*field*, *value*, *exclude_masked=True*, *op='and'*, *inclusive=False*)
Include gates where a given field is below a given value.

include_equal (*field*, *value*, *exclude_masked=True*, *op='and'*)
Include gates where a given field is equal to a value.

include_gates (*mask*, *exclude_masked=True*, *op='and'*)
Include gates where a given mask is equal True.

Parameters mask : numpy array

Boolean numpy array with same shape as a field array.

exclude_masked : bool, optional

True to filter masked values in the specified mask if it is a masked array, False to include any masked values.

op : {'and', 'or', 'new'}

Operation to perform when merging the existing set of excluded gates with the excluded gates from the current operation. 'and' will perform a logical AND operation, 'or' a logical OR, and 'new' will replace the existing excluded gates with the one generated here. 'or', the default for exclude methods, is typically desired when building up a set of conditions for excluding gates where the desired effect is to exclude gates which meet any of the conditions. 'and', the default for include methods, is typically desired when building up a set of conditions where the desired effect is to include gates which meet any of the conditions. Note that the 'or' method MAY results in excluding gates which have previously been included.

include_inside (*field*, *v1*, *v2*, *exclude_masked=True*, *op='and'*, *inclusive=True*)
Include gates where a given field is inside a given interval.

include_none ()
Include no gates, exclude all gates.

include_not_equal (*field*, *value*, *exclude_masked=True*, *op='and'*)
Include gates where a given field is not equal to a value.

include_not_masked (*field*, *exclude_masked=True*, *op='and'*)
Include gates where a given field in not masked.

include_not_transition (*trans_value=0*, *exclude_masked=True*, *op='and'*)
Include all gates in rays not marked as in transition between sweeps.

Include all gates in rays not marked as "in transition" by the antenna_transition attribute of the radar used to construct the filter. If no antenna transition information is available all gates are included.

Parameters **trans_value** : int, optional

Value used in the antenna transition data to indicate that the instrument is not between sweeps (in transition) during the collection of a specific ray. Typically a value of 0 is used to indicate no transition and the default can be used in these cases.

exclude_masked : bool, optional

True to filter masked values in antenna_transition if the data is a masked array, False to include any masked values.

op : {'and', 'or', 'new'}

Operation to perform when merging the existing set of excluded gates with the excluded gates from the current operation. 'and' will perform a logical AND operation, 'or' a logical OR, and 'new' will replace the existing excluded gates with the one generated here. 'or', the default for exclude methods, is typically desired when building up a set of conditions for excluding gates where the desired effect is to exclude gates which meet any of the conditions. 'and', the default for include methods, is typically desired when building up a set of conditions where the desired effect is to include gates which meet any of the conditions. Note that the 'or' method MAY results in excluding gates which have previously been included.

include_outside (*field*, *v1*, *v2*, *exclude_masked=True*, *op='and'*, *inclusive=False*)
Include gates where a given field is outside a given interval.

include_valid (*field*, *exclude_masked=True*, *op='and'*)
Include gates where a valid value occurs in a field (not NaN or inf).

pyart.filters.gatefilter.**moment_and_texture_based_gate_filter** (*radar*, *zdr_field=None*, *rhv_field=None*, *phi_field=None*, *refl_field=None*, *textzdr_field=None*, *textrhv_field=None*, *textphi_field=None*, *textrefl_field=None*, *wind_size=7*, *max_textphi=20.0*, *max_textrhv=0.3*, *max_textzdr=2.85*, *max_textrefl=8.0*, *min_rhv=0.6*)
Create a filter which removes undesired gates based on texture of moments.

Creates a gate filter in which the following gates are excluded: * Gates where the instrument is transitioning between sweeps. * Gates where RhoHV is below min_rhv * Gates where the PhiDP texture is above max_textphi. * Gates where the RhoHV texture is above max_textrhv. * Gates where the ZDR texture is above max_textzdr * Gates where the reflectivity texture is above max_textrefl * If any of the thresholds is not set or the field (RhoHV, ZDR, PhiDP, reflectivity) do not exist in the radar the filter is not applied.

Parameters **radar** : Radar

Radar object from which the gate filter will be built.

zdr_field, rhv_field, phi_field, refl_field : str

Names of the radar fields which contain the differential reflectivity, cross correlation ratio, differential phase and reflectivity from which the textures will be computed. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

textzdr_field, textrhv_field, textphi_field, textrefl_field : str

Names of the radar fields given to the texture of the differential reflectivity, texture of the cross correlation ratio, texture of differential phase and texture of reflectivity. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file

wind_size : int

Size of the moving window used to compute the ray texture.

max_textphi, max_textrhv, max_textzdr, max_textrefl : float

Maximum value for the texture of the differential phase, texture of RhoHV, texture of Zdr and texture of reflectivity. Gates in these fields above these limits as well as gates which are masked or contain invalid values will be excluded and not used in calculation which use the filter. A value of None will disable filtering based upon the given field including removing masked or gates with an invalid value. To disable the thresholding but retain the masked and invalid filter set the parameter to a value above the highest value in the field.

min_rhv : float

Minimum value for the RhoHV. Gates below this limits as well as gates which are masked or contain invalid values will be excluded and not used in calculation which use the filter. A value of None will disable filtering based upon the given field including removing masked or gates with an invalid value. To disable the thresholding but retain the masked and invalid filter set the parameter to a value below the lowest value in the field.

Returns **gatefilter** : `GateFilter`

A gate filter based upon the described criteria. This can be used as a gatefilter parameter to various functions in pyart.correct.

`pyart.filters.gatefilter.`**`moment_based_gate_filter`**(*radar*, *ncp_field=None*, *rhv_field=None*, *refl_field=None*, *min_ncp=0.5*, *min_rhv=None*, *min_refl=-20.0*, *max_refl=100.0*)

Create a filter which removes undesired gates based on moments.

Creates a gate filter in which the following gates are excluded:

- Gates where the instrument is transitioning between sweeps.
- Gates where the reflectivity is outside the interval min_refl, max_refl.
- Gates where the normalized coherent power is below min_ncp.
- Gates where the cross correlation ratio is below min_rhi. Using the default parameter this filtering is disabled.
- Gates where any of the above three fields are masked or contain invalid values (NaNs or infs).

•If any of these three fields do not exist in the radar that fields filter criteria is not applied.

Parameters radar : Radar

Radar object from which the gate filter will be built.

refl_field, ncp_field, rhv_field : str

Names of the radar fields which contain the reflectivity, normalized coherent power (signal quality index) and cross correlation ratio (RhoHV) from which the gate filter will be created using the above criteria. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

min_ncp, min_rhv : float

Minimum values for the normalized coherence power and cross correlation ratio. Gates in these fields below these limits as well as gates which are masked or contain invalid values will be excluded and not used in calculation which use the filter. A value of None will disable filtering based upon the given field including removing masked or gates with an invalid value. To disable the thresholding but retain the masked and invalid filter set the parameter to a value below the lowest value in the field.

min_refl, max_refl : float

Minimum and maximum values for the reflectivity. Gates outside of this interval as well as gates which are masked or contain invalid values will be excluded and not used in calculation which use this filter. A value or None for one of these parameters will disable the minimum or maximum filtering but retain the other. A value of None for both of these values will disable all filtering based upon the reflectivity including removing masked or gates with an invalid value. To disable the interval filtering but retain the masked and invalid filter set the parameters to values above and below the lowest and greatest values in the reflectivity field.

Returns gatefilter : `GateFilter`

A gate filter based upon the described criteria. This can be used as a gatefilter parameter to various functions in pyart.correct.

`pyart.filters.gatefilter.`**`snr_based_gate_filter`**(*radar*, *snr_field=None*, *min_snr=10.0*)

Create a filter which removes undesired gates based on SNR.

Parameters radar : Radar

Radar object from which the gate filter will be built.

snr_field : str

Name of the radar field which contains the signal to noise ratio. A value of None for will use the default field name as defined in the Py-ART configuration file.

min_snr : float

Minimum value for the SNR. Gates below this limits as well as gates which are masked or contain invalid values will be excluded and not used in calculation which use the filter. A value of None will disable filtering based upon the field including removing masked or gates with an invalid value. To disable the thresholding but retain the masked and invalid filter set the parameter to a value below the lowest value in the field.

Returns gatefilter : `GateFilter`

A gate filter based upon the described criteria. This can be used as a gatefilter parameter to various functions in pyart.correct.

`pyart.filters.gatefilter.`**`temp_based_gate_filter`**(*radar*, *temp_field=None*, *min_temp=0.0*, *thickness=400.0*, *beamwidth=None*)

Create a filter which removes undesired gates based on temperature. Used primarily to filter out the melting layer and gates above it.

Parameters **radar** : Radar

Radar object from which the gate filter will be built.

temp_field : str

Name of the radar field which contains the temperature. A value of None for will use the default field name as defined in the Py-ART configuration file.

min_temp : float

Minimum value for the temperature in degrees. Gates below this limits as well as gates which are masked or contain invalid values will be excluded and not used in calculation which use the filter. A value of None will disable filtering based upon the field including removing masked or gates with an invalid value. To disable the thresholding but retain the masked and invalid filter set the parameter to a value below the lowest value in the field.

thickness : float

The estimated thickness of the melting layer in m

beamwidth : float

The radar antenna 3 dB beamwidth [deg]

Returns **gatefilter** : *GateFilter*

A gate filter based upon the described criteria. This can be used as a gatefilter parameter to various functions in pyart.correct.

`pyart.filters.gatefilter.`**`visibility_based_gate_filter`**(*radar*, *vis_field=None*, *min_vis=10.0*)

Create a filter which removes undesired gates based on visibility.

Parameters **radar** : Radar

Radar object from which the gate filter will be built.

vis_field : str

Name of the radar field which contains the visibility. A value of None for will use the default field name as defined in the Py-ART configuration file.

min_vis : float

Minimum value for the visibility. Gates below this limits as well as gates which are masked or contain invalid values will be excluded and not used in calculation which use the filter. A value of None will disable filtering based upon the field including removing masked or gates with an invalid value. To disable the thresholding but retain the masked and invalid filter set the parameter to a value below the lowest value in the field.

Returns **gatefilter** : *GateFilter*

A gate filter based upon the described criteria. This can be used as a gatefilter parameter to various functions in pyart.correct.

CHAPTER

FORTYONE

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_zphi(radar[, doc, ...])	Calculate the attenuation and the differential attenuation from a polarimetric radar using Z-PHI method..
calculate_attenuation_philinear(radar[, ...])	Calculate the attenuation and the differential attenuation from a polarimetric radar using linear dependece with PhiDP.
get_mask_fzl(radar[, fzl, doc, min_temp, ...])	constructs a mask to mask data placed thickness m below data at min_temp
_prepare_phidp(phidp, mask_fzl)	Prepares phidp to be used in attenuation correction by masking values
_get_param_attzphi(freq)	get the parameters of Z-Phi attenuation estimation for a particular
_param_attzphi_table()	defines the parameters of Z-Phi attenuation estimation at each frequency
_get_param_attphilinear(freq)	get the parameters of attenuation estimation based on phidp for a
_param_attphilinear_table()	defines the parameters of attenuation estimation based on phidp at each

pyart.correct.attenuation.**_get_param_attphilinear**(*freq*)

get the parameters of attenuation estimation based on phidp for a particular frequency

Parameters freq : float

radar frequency [Hz]

Returns a_coeff, beta, c, d : floats

the coefficient and exponent of the power law

pyart.correct.attenuation.**_get_param_attzphi**(*freq*)

get the parameters of Z-Phi attenuation estimation for a particular frequency

Parameters freq : float

radar frequency [Hz]

Returns a_coeff, beta, c, d : floats

the coefficient and exponent of the power law pyart.correct.attenuation.**_param_attphilinear_table**()

defines the parameters of attenuation estimation based on phidp at each frequency band.

Returns param_att_dict : dict

A dictionary with the coefficients at each band

pyart.correct.attenuation.**_param_attzphi_table**()

defines the parameters of Z-Phi attenuation estimation at each frequency band.

Returns param_att_dict : dict

A dictionary with the coefficients at each band

pyart.correct.attenuation.**_prepare_phidp**(*phidp*, *mask_fzl*)

Prepares phidp to be used in attenuation correction by masking values above freezing level setting negative values to 0 and make sure it is monotously increasing

Parameters phidp : ndarray 2D

The phidp field

mask_fzl : ndarray 2D

a mask of the data above freezing level height

Returns corr_phidp: ndarray 2D

the corrected PhiDP field

pyart.correct.attenuation.**calculate_attenuation_philinear**(*radar*, *doc=None*, *fzl=None*, *pia_coef=None*, *pida_coef=None*, *refl_field=None*, *phidp_field=None*, *zdr_field=None*, *temp_field=None*, *spec_at_field=None*, *corr_refl_field=None*, *spec_diff_at_field=None*, *corr_zdr_field=None*)

Calculate the attenuation and the differential attenuation from a polarimetric radar using linear dependece with PhiDP. The attenuation is computed up to a user defined freezing level height or up to where temperatures in a temperature field are positive. The coefficients are either user-defined or radar frequency dependent.

Parameters radar : Radar

Radar object to use for attenuation calculations. Must have phidp and refl fields.

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.

pia_coef : float

Coefficient in path integrated attenuation calculation

pida_coeff : float

Coefficient in path integrated differential attenuation calculation

refl_field, phidp_field, zdr_field, temp_field : str

Field names within the radar object which represent the horizonal reflectivity, the differential phase shift, the differential reflectivity and the temperature field. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file. The ZDR field and temperature field are going to be used only if available.

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.

spec_diff_at_field, corr_zdr_field : str

Names of the specific differential attenuation and the corrected differential 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. These fields will be computed only if the ZDR field is available.

Returns spec_at : dict

Field dictionary containing the specific attenuation.

cor_z : dict

Field dictionary containing the corrected reflectivity.

spec_diff_at : dict

Field dictionary containing the specific differential attenuation.

cor_zdr : dict

Field dictionary containing the corrected differential reflectivity.

`pyart.correct.attenuation.`**`calculate_attenuation_zphi`**(*radar*, *doc=None*, *fzl=None*, *smooth_window_len=5*, *a_coef=None*, *beta=None*, *c=None*, *d=None*, *refl_field=None*, *phidp_field=None*, *zdr_field=None*, *temp_field=None*, *spec_at_field=None*, *corr_refl_field=None*, *spec_diff_at_field=None*, *corr_zdr_field=None*)

Calculate the attenuation and the differential attenuation from a polarimetric radar using Z-PHI method.. The attenuation is computed up to a user defined freezing level height or up to where temperatures in a temperature field are positive. The coefficients are either user-defined or radar frequency dependent.

Parameters radar : Radar

Radar object to use for attenuation calculations. Must have phidp and refl fields.

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.

smooth_window_len : int

Size, in range bins, of the smoothing window

a_coef : float

A coefficient in attenuation calculation.

beta : float

Beta parameter in attenuation calculation.

c, d : float

coefficient and exponent of the power law that relates attenuation with differential attenuation

refl_field, phidp_field, zdr_field, temp_field : str

Field names within the radar object which represent the horizonal reflectivity, the differential phase shift, the differential reflectivity and the temperature field. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file. The ZDR field and temperature field are going to be used only if available.

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.

spec_diff_at_field, corr_zdr_field : str

Names of the specific differential attenuation and the corrected differential 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. These fields will be computed only if the ZDR field is available.

Returns **spec_at** : dict

Field dictionary containing the specific attenuation.

cor_z : dict

Field dictionary containing the corrected reflectivity.

spec_diff_at : dict

Field dictionary containing the specific differential attenuation.

cor_zdr : dict

Field dictionary containing the corrected differential reflectivity.

References

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

Ryzhkov et al. Potential Utilization of Specific Attenuation for Rainfall Estimation, Mitigation of Partial Beam Blockage, and Radar Networking, JAOT, 2014, 31, 599-619.

`pyart.correct.attenuation.`**`get_mask_fzl`**(*radar*, *fzl=None*, *doc=None*, *min_temp=0.0*, *thickness=None*, *beamwidth=None*, *temp_field=None*)

constructs a mask to mask data placed thickness m below data at min_temp and beyond

Parameters **radar** : Radar

the radar object

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.

min_temp : float

minimum temperature below which the data is mask in degrees

thickness : float

extent of the layer below the first gate where min_temp is reached that is going to be masked

beamwidth : float

the radar antenna 3 dB beamwidth

temp_field : str

Field names within the radar object which represent the temperature field. A value of None will use the default field name as defined in the Py-ART configuration file. It is going to be used only if available.

Returns **mask_fzl** : 2D array

the values that should be masked

end_gate_arr : 1D array

the index of the last valid gate in the ray

CHAPTER FORTYTWO

PYART.CORRECT.BIAS_AND_NOISE

Corrects polarimetric variables for noise

correct_noise_rhohv(radar[, urhohv_field, ...])	Corrects RhoHV for noise according to eq.
correct_bias(radar[, bias, field_name])	Corrects a radar data bias.
correct_visibility(radar[, vis_field, ...])	Corrects the reflectivity according to visibility.
get_sun_hits(radar[, delev_max, dazim_max, ...])	get data from suspected sun hits
sun_retrieval(az_rad, az_sun, el_rad, ...[, ...])	Estimates sun parameters from sun hits
est_rhohv_rain(radar[, ind_rmin, ind_rmax, ...])	Estimates the quantiles of RhoHV in rain for each sweep
est_zdr_rain(radar[, ind_rmin, ind_rmax, ...])	Estimates the average ZDR in moderate rain
selfconsistency_bias(radar, zdr_kdpzh_dict)	Estimates reflectivity bias at each ray using the self-consistency
selfconsistency_kdp_phidp(radar, zdr_kdpzh_dict)	Estimates KDP and PhiDP in rain from Zh and ZDR using a selfconsistency relation between ZDR, Zh and KDP.
get_kdp_selfcons(zdr, refl, ele_vec, ...)	Estimates KDP and PhiDP in rain from Zh and ZDR using a selfconsistency
_est_sun_hit_pwr(pwr, sun_hit, attg_sun, ...)	estimates sun hit power, standard deviation, and number and position of
_est_sun_hit_zdr(zdr, sun_hit_zdr, ...)	estimates sun hit ZDR, standard deviation, and number and position of
_selfconsistency_kdp_phidp(radar, refl, zdr, ...)	Estimates KDP and PhiDP in rain from Zh and ZDR using a selfconsistency relation between ZDR, Zh and KDP.

pyart.correct.bias_and_noise.**_est_sun_hit_pwr**(*pwr*, *sun_hit*, *attg_sun*, *max_std*, *nbins_min*, *ind_rmin*)

estimates sun hit power, standard deviation, and number and position of affected range bins in a ray

Parameters **pwr** : 1D float array

the power at each range bin in a ray

sun_hit : 1D float array

array used to flag sun hit range bins

attg_sun : float

attenuation suffered by the sun signal from the top of the atmosphere to the radar position

max_std : float

maximum standard deviation to consider the sun hit valid

nbins_min : int

minimum number of range gates with valid signal in the ray to consider the ray affected by a noise-like signal

ind_rmin : int

minimum range from which we can look for noise

Returns **sunpwr_dBm** : float

the estimated sun power

sunpwr_std : float

the standard deviation of the estimation in dB

sunpwr_npoints : int

the number of range gates affected by the sun hit

sun_hit : 1D array

array with flagged range bins

`pyart.correct.bias_and_noise._est_sun_hit_zdr`(*zdr*, *sun_hit_zdr*, *sun_hit_h*, *sun_hit_v*, *max_std*, *nbins_min*, *ind_rmin*)

estimates sun hit ZDR, standard deviation, and number and position of affected range bins in a ray

Parameters **zdr** : 1D float array

the ZDR at each range bin in a ray

sun_hit_zdr : 1D float array

array used to flag sun hit range bins

sun_hit_h, sun_hit_v : 1D float array

The position of sun hit range bins in eanch channel

max_std : float

maximum standard deviation

nbins_min : int

minimum number of range gates with valid signal in the ray to consider the ray affected by a noise-like signal

ind_rmin : int

minimum range from which we can look for noise

Returns **sunzdr** : float

the estimated sun power

sunzdr_std : float

the standard deviation of the estimation in dB

sunzdr_npoints : int

the number of range gates affected by the sun hit

sun_hit_zdr : 1D array

array with flagged range bins

`pyart.correct.bias_and_noise.`**`_selfconsistency_kdp_phidp`**(*radar*, *refl*, *zdr*, *phidp*, *zdr_kdpzh_dict*, *max_phidp=20.0*, *smooth_wind_len=5*, *rhohv=None*, *min_rhohv=None*, *doc=None*, *fzl=None*, *thickness=700.0*, *temp_field=None*)

Estimates KDP and PhiDP in rain from Zh and ZDR using a selfconsistency relation between ZDR, Zh and KDP. Private method

Parameters radar : Radar

radar object

refl, zdr, phidp : ndarray 2D

reflectivity field, differential reflectivity field and differential phase field. They must exist

zdr_kdpzh_dict : dict

dictionary containing a look up table relating ZDR with KDP/Zh for different elevations

rhohv : ndarray 2D

copolar correlation field used for masking data. Optional

max_phidp : float

maximum PhiDP value to consider the data valid

smooth_wind_len : int

length of the smoothing window for Zh and ZDR data

min_rhohv : float

minimum RhoHV value to consider the data valid

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.

thickness : float

Assumed thickness of the melting layer [m]

temp_field : str

Field name within the radar object which represent the temperature field. A value of None will use the default field name as defined in the Py-ART configuration file. It is going to be used only if available.

Returns kdp_sim, phidp_sim : ndarray 2D

the KDP and PhiDP estimated fields

`pyart.correct.bias_and_noise.`**`correct_bias`**(*radar*, *bias=0.0*, *field_name=None*)

Corrects a radar data bias. If field name is none the correction is applied to horizontal reflectivity by default

Parameters radar : Radar

radar object

bias : float

the bias magnitude

field_name: str

names of the field to be corrected

Returns corrected_field : dict

The corrected field

`pyart.correct.bias_and_noise.correct_noise_rhohv`(*radar*, *urhohv_field=None*, *snr_field=None*, *zdr_field=None*, *nh_field=None*, *nv_field=None*, *rhohv_field=None*)

Corrects RhoHV for noise according to eq. 6 in Gourley et al. 2006. This correction should only be performed if noise has not been subtracted from the signal during the moments computation.

Parameters radar : Radar

radar object

urhohv_field : str

name of the RhoHV uncorrected for noise field

snr_field, zdr_field, nh_field, nv_field: str

names of the SNR, ZDR, horizontal channel noise in dBZ and vertical channel noise in dBZ used to correct RhoHV

rhohv_field: str

name of the rhohv field to output

Returns rhohv : dict

noise corrected RhoHV field

References

Gourley et al. Data Quality of the Meteo-France C-Band Polarimetric Radar, JAOT, 23, 1340-1356

`pyart.correct.bias_and_noise.correct_visibility`(*radar*, *vis_field=None*, *field_name=None*)

Corrects the reflectivity according to visibility. Applied to horizontal reflectivity by default

Parameters radar : Radar

radar object

vis_field : str

the name of the visibility field

field_name: str

names of the field to be corrected

Returns corrected_field : dict

The corrected field

`pyart.correct.bias_and_noise.est_rhohv_rain(radar, ind_rmin=10, ind_rmax=500, zmin=20.0, zmax=40.0, thickness=700.0, doc=None, fzl=None, rhohv_field=None, temp_field=None, refl_field=None)`

Estimates the quantiles of RhoHV in rain for each sweep

Parameters **radar** : Radar

radar object

ind_rmin, ind_rmax : int

Min and max range index where to look for rain

zmin, zmax : float

The minimum and maximum reflectivity to consider the radar bin suitable rain

thickness : float

Assumed thickness of the melting layer

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.

temp_field, rhohv_field, refl_field : str

Field names within the radar object which represent the temperature, co-polar correlation and reflectivity fields. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns **rhohv_rain_dict** : dict

The estimated RhoHV in rain for each sweep and metadata

`pyart.correct.bias_and_noise.est_zdr_rain(radar, ind_rmin=10, ind_rmax=500, zmin=20.0, zmax=22.0, rhohvmin=0.97, phidpmax=10.0, elmax=20.0, thickness=700.0, doc=None, fzl=None, zdr_field=None, rhohv_field=None, phidp_field=None, temp_field=None, refl_field=None)`

Estimates the average ZDR in moderate rain

Parameters **radar** : Radar

radar object

ind_rmin, ind_rmax : int

Min and max range index where to look for rain

zmin, zmax : float

The minimum and maximum reflectivity to consider the radar bin suitable rain

rhohvmin : float

Minimum RhoHV to consider the radar bin suitable rain

phidpmax : float

Maximum PhiDP to consider the radar bin suitable rain

elmax : float

Maximum elevation

thickness : float

Assumed thickness of the melting layer

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.

zdr_field, rhohv_field, refl_field, phidp_field, temp_field : str

Field names within the radar object which represent the differential reflectivity, co-polar correlation, reflectivity, differential phase and temperature fields. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns zdr_rain_dict : dict

The estimated RhoHV in rain for each sweep and metadata

`pyart.correct.bias_and_noise.`**`get_kdp_selfcons`**(*zdr*, *refl*, *ele_vec*, *zdr_kdpzh_dict*)

Estimates KDP and PhiDP in rain from Zh and ZDR using a selfconsistency relation between ZDR, Zh and KDP

Parameters zdr, refl : ndarray 2D

reflectivity and differential reflectivity fields

ele_vec : ndarray 1D

vector containing the elevation angles of each ray

zdr_kdpzh_dict : dict

dictionary containing a look up table relating ZDR with KDP/Zh for different elevations

Returns kdp_sim : ndarray 2D

the KDP estimated from zdr and refl

`pyart.correct.bias_and_noise.`**`get_sun_hits`**(*radar*, *delev_max=2.0*, *dazim_max=2.0*, *elmin=1.0*, *ind_rmin=100*, *percent_bins=10.0*, *attg=None*, *max_std=1.0*, *pwrh_field=None*, *pwrv_field=None*, *zdr_field=None*)

get data from suspected sun hits

Parameters radar : Radar

radar object

delev_max, dazim_max : float

maximum difference in elevation and azimuth between sun position and antenna pointing

elmin : float

minimum radar elevation angle

ind_rmin : int

minimum range from which we can look for noise

percent_bins : float

percentage of bins with valid data to consider a ray as potentially sun hit

attg : float

gas attenuation coefficient (1-way)

pwrh_field, pwrv_field, zdr_field : str

names of the signal power in dBm for the H and V polarizations and the differential reflectivity

Returns **sun_hits** : dict

a dictionary containing information of the sun hits

new_radar : radar object

radar object containing sweeps that contain sun hits

`pyart.correct.bias_and_noise.`**`selfconsistency_bias`**(*radar*, *zdr_kdpzh_dict*, *min_rhohv=0.92*, *max_phidp=20.0*, *smooth_wind_len=5*, *doc=None*, *fzl=None*, *thickness=700.0*, *min_rcons=20*, *dphidp_min=2*, *dphidp_max=16*, *refl_field=None*, *phidp_field=None*, *zdr_field=None*, *temp_field=None*, *rhohv_field=None*)

Estimates reflectivity bias at each ray using the self-consistency algorithm by Gourley

Parameters **radar** : Radar

radar object

zdr_kdpzh_dict : dict

dictionary containing a look up table relating ZDR with KDP/Zh for different elevations

min_rhohv : float

minimum RhoHV value to consider the data valid

max_phidp : float

maximum PhiDP value to consider the data valid

smooth_wind_len : int

length of the smoothing window

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.

min_rcons : int

minimum number of consecutive gates to consider a valid segment of PhiDP

dphidp_min : float

minimum differential phase shift in a segment

dphidp_max : float

maximum differential phase shift in a segment

refl_field, phidp_field, zdr_field : str

Field names within the radar object which represent the reflectivity, differential phase and differential reflectivity fields. A value of None will use the default field name as defined in the Py-ART configuration file.

temp_field, rhohv_field : str

Field names within the radar object which represent the temperature, and co-polar correlation fields. A value of None will use the default field name as defined in the Py-ART configuration file. They are going to be used only if available.

kdpsim_field, phidpsim_field : str

Field names which represent the estimated specific differential phase and differential phase. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns refl_bias_dict : dict

the bias at each ray field and metadata

`pyart.correct.bias_and_noise.`**`selfconsistency_kdp_phidp`**(*radar*, *zdr_kdpzh_dict*, *min_rhohv=0.92*, *max_phidp=20.0*, *smooth_wind_len=5*, *doc=None*, *fzl=None*, *thickness=700.0*, *refl_field=None*, *phidp_field=None*, *zdr_field=None*, *temp_field=None*, *rhohv_field=None*, *kdpsim_field=None*, *phidpsim_field=None*)

Estimates KDP and PhiDP in rain from Zh and ZDR using a selfconsistency relation between ZDR, Zh and KDP. Private method

Parameters radar : Radar

radar object

zdr_kdpzh_dict : dict

dictionary containing a look up table relating ZDR with KDP/Zh for different elevations

min_rhohv : float

minimum RhoHV value to consider the data valid

max_phidp : float

maximum PhiDP value to consider the data valid

smooth_wind_len : int

length of the smoothing window

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.

thickness : float

assumed melting layer thickness [m]

refl_field, phidp_field, zdr_field : str

Field names within the radar object which represent the reflectivity, differential phase and differential reflectivity fields. A value of None will use the default field name as defined in the Py-ART configuration file.

temp_field, rhohv_field : str

Field names within the radar object which represent the temperature, and co-polar correlation fields. A value of None will use the default field name as defined in the Py-ART configuration file. They are going to be used only if available.

kdpsim_field, phidpsim_field : str

Field names which represent the estimated specific differential phase and differential phase. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns kdp_sim_dict, phidp_sim_dict : dict

the KDP and PhiDP estimated fields and metadata

`pyart.correct.bias_and_noise.`**`sun_retrieval`**(*az_rad*, *az_sun*, *el_rad*, *el_sun*, *sun_hit*, *sun_hit_std*, *az_width_co=None*, *el_width_co=None*, *az_width_cross=None*, *el_width_cross=None*, *is_zdr=False*)

Estimates sun parameters from sun hits

Parameters az_rad, az_sun, el_rad, el_sun : float array

azimuth and elevation values of the sun and the radar

sun_hit : float array

sun hit value. Either power in dBm or ZDR in dB

sun_hit_std : float array

standard deviation of the sun hit value in dB

az_width_co, el_width_co, az_width_cross, el_width_cross : float

azimuth and elevation antenna width for each channel

is_zdr : boolean

boolean to signal that is ZDR data

Returns val, val_std : float

retrieved value and its standard deviation

az_bias, el_bias : float

retrieved azimuth and elevation antenna bias respect to the sun position

az_width, el_width : float

retrieved azimuth and elevation antenna widths

nhits : int

number of sun hits used in the retrieval

CHAPTER

FORTYTHREE

PYART.CORRECT.DEALIAS

Front end to the University of Washington 4DD code for Doppler dealiasing.

`dealias_fourdd`(radar[, last_radar, ...])	Dealias Doppler velocities using the 4DD algorithm.
`find_time_in_interp_sonde`(interp_sonde, target)	Find the wind parameter for a given time in a ARM interp-sonde file.
`_create_rsl_volume`(radar, field_name, ...[, ...])	Create a RSLVolume containing data from a field in radar.

`pyart.correct.dealias.`**`_create_rsl_volume`**(*radar*, *field_name*, *vol_num*, *rsl_badval*, *excluded=None*)

Create a RSLVolume containing data from a field in radar.

`pyart.correct.dealias.`**`dealias_fourdd`**(*radar*, *last_radar=None*, *sonde_profile=None*, *gatefilter=False*, *sounding_heights=None*, *sounding_wind_speeds=None*, *sounding_wind_direction=None*, *filt=1*, *rsl_badval=131072.0*, *keep_original=False*, *set_limits=True*, *vel_field=None*, *corr_vel_field=None*, *last_vel_field=None*, *debug=False*, *max_shear=0.05*, *sign=1*, ***kwargs*)

Dealias Doppler velocities using the 4DD algorithm.

Dealias the Doppler velocities field using the University of Washington 4DD algorithm utilizing information from a previous volume scan and/or sounding data. Either last_radar or sonde_profile must be provided. For best results provide both a previous volume scan and sounding data. Radar and last_radar must contain the same number of rays per sweep.

Additional arguments are passed to `_fourdd_interface.fourdd_dealias()`. These can be used to fine tune the behavior of the FourDD algorithm. See the documentation of Other Parameters for details. For the default values of these parameters see the documentation of `_fourdd_interface.fourdd_dealias()`.

Parameters radar : Radar

Radar object to use for dealiasing. Must have a Nyquist defined in the instrument_parameters attribute and have a reflectivity_horizontal and mean_doppler_velocity fields.

last_radar : Radar, optional

The previous radar volume, which has been successfully dealiased. Using a previous volume as an initial condition can greatly improve the dealiasing, and represents the final dimension in the 4DD algorithm.

sonde_profile : HorizontalWindProfile

Profile of horizontal winds from a sonding used for the initial condition of the dealiasing.

Returns vr_corr : dict

Field dictionary containing dealiased Doppler velocities. Dealiased array is stored under the 'data' key.

Other Parameters gatefilter : GateFilter, optional.

A GateFilter instance which specifies which gates should be ignored when performing velocity dealiasing. A value of None will create this filter from the radar moments using any additional arguments by passing them to `moment_based_gate_filter()`. The default value assumes all gates are valid.

sounding_heights : ndarray, optional

This argument is deprecated and should be specified using the sonde_profile argument. Sounding heights in meters above mean sea level. If altitude attribute of the radar object if reference against something other than mean sea level then this parameter should also be referenced in that manner.

sounding_wind_speeds : ndarray, optional

This argument is deprecated and should be specified using the sonde_profile argument. Sounding wind speeds in m/s.

sounding_wind_direction : ndarray, optional

This argument is deprecated and should be specified using the sonde_profile argument. Sounding wind directions in degrees.

filt : int, optional

Flag controlling Bergen and Albers filter, 1 = yes, 0 = no.

rsl_badval : float, optional

Value which represents a bad value in RSL.

keep_original : bool, optional

True to keep original doppler velocity values when the dealiasing procedure fails, otherwise these gates will be masked. NaN values are still masked.

set_limits : bool, optional

True to set valid_min and valid_max elements in the returned dictionary. False will not set these dictionary elements.

vel_field : str, optional

Field in radar to use as the Doppler velocities during dealiasing. None will use the default field name from the Py-ART configuration file.

corr_vel_field : str, optional

Name to use for the dealiased Doppler velocity field metadata. None will use the default field name from the Py-ART configuration file.

last_vel_field : str, optional

Name to use for the dealiased Doppler velocity field metadata in last_radar. None will use the corr_vel_field name.

maxshear : float, optional

Maximum vertical shear which will be incorporated into the created volume from the sounding data. Parameter not used when no sounding data is provided.

sign : int, optional

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

compthresh : float, optional

Fraction of the Nyquist velocity to use as a threshold when performing continuity (initial) dealiasing. Velocities differences above this threshold will not be marked as gate from which to begin unfolding during spatial dealiasing.

compthresh2 : float, optional

The same as compthresh but the value used during the second pass of dealiasing. This second pass is only performed in both a sounding and last volume are provided.

thresh : float, optional

Fraction of the Nyquist velocity to use as a threshold when performing spatial dealiasing. Horizontally adjacent gates with velocities above this threshold will count against assigning the gate in question the velocity value being tested.

ckval : float, optional

When the absolute value of the velocities are below this value they will not be marked as gates from which to begin unfolding during spatial dealiasing.

stdthresh : float, optional

Fraction of the Nyquist velocity to use as a standard deviation threshold in the window dealiasing portion of the algorithm.

epsilon : float, optional

Difference used when comparing a value to missing value, changing this from the default is not recommended.

maxcount : int, optional

Maximum allowed number of fold allowed when unfolding velocities.

pass2 : int, optional

Controls weather unfolded gates should be removed (a value of 0) or retained for unfolding during the second pass (a value of 1) when both a sounding volume and last volume are provided.

rm : int, optional

Determines what should be done with gates that are left unfolded after the first pass of dealiasing. A value of 1 will remove these gates, a value of 0 sets these gates to their initial velocity. If both a sounding volume and last volume are provided this parameter is ignored.

proximity : int, optional

Number of gates and rays to include of either side of the current gate during window dealiasing. This value may be doubled in cases where a standard sized window does not capture a sufficient number of good valued gates.

mingood : int, optional

Number of good valued gates required within the window before the current gate will be unfolded.

ba_mincount : int, optional

Number of neighbors required during Bergen and Albers filter for a given gate to be included, must be between 1 and 8, 5 recommended.

ba_edgecount : int, optional

Same as ba_mincount but used at ray edges, must be between 1 and 5, 3 recommended.

debug : bool, optional

Set True to return RSL Volume objects for debugging: usuccess, radialVelVolume, lastVelVolume, unfoldedVolume, sondVolume

Notes

Due to limitations in the C code do not call with sounding arrays over 999 elements long.

References

C. N. James and R. A Houze Jr, A Real-Time Four-Dimensional Doppler Dealising Scheme, Journal of Atmospheric and Oceanic Technology, 2001, 18, 1674.

`pyart.correct.dealias.`**`find_time_in_interp_sonde`**(*interp_sonde*, *target*, *debug=False*)

Find the wind parameter for a given time in a ARM interpsonde file.

This function is Deprecated and will be removed in future versions of Py-ART. Use the `pyart.io.read_arm_sonde_vap()` function for similar functionality.

Parameters **interp_sonde** : netCDF4.Dataset

netCDF4 object pointing to a ARM interpsonde file.

target : datetime

Target datetime, the closest time in the interpsonde file will be used.

Returns **height** : np.ndarray

Heights above the ground for the time closest to target.

speed : np.ndarray

Wind speeds at given height for the time closest to taget.

direction : np.ndarray

Wind direction at given height for the time closest to target.

Other Parameters **debug** : bool

Print debugging information.

CHAPTER

FORTYFOUR

PYART.CORRECT.DESPECKLE

Find contiguous objects in scans and despeckle away ones that are too small.

`despeckle_field`(radar, field[, label_dict, ...])	Despeckle a radar volume by identifying small objects in each scan and masking them out.
`find_objects`(radar, field, threshold[, ...])	Find objects (i.e., contiguous gates) in one or more sweeps that match thresholds.
`_adjust_for_periodic_boundary`(data)	Identify all the contiguous objects in a sweep, accounting for the periodic boundary in a 360-deg PPI.
`_append_labels`(labels, label_storage)	Appends consecutive sweeps of labels, creating a multi-sweep 2D array.
`_check_for_360`(az, delta)	Check if an array of azimuths indicates the sweep is a full 360 PPI.
`_check_sweeps`(sweeps, radar)	Parse the sweeps keyword and convert it to a list of ints.
`_check_threshold`(threshold)	Parse the threshold keyword and return the lower and upper boundaries for the object search.
`_generate_dict`(label_storage)	Build the dictionary that includes all the object label information.
`_get_data`(radar, iswp, field, tlo, thi, window)	Get data for a field from a given sweep in a Radar object.
`_get_labels`(data)	Identify all the contiguous objects in a sweep.
`_smooth_data`(data, window)	Perform box filtering along each ray of a sweep, and return the smoothed field.

`pyart.correct.despeckle.`**`_adjust_for_periodic_boundary`**(*data*)

Identify all the contiguous objects in a sweep, accounting for the periodic boundary in a 360-deg PPI. Contiguous means corners or sides of gates touch. The algorithm appends the sweep to itself, then looks for contiguous objects near the original PPI edges and relabels them. Then, the extra sweep is discarded before returning all the labels.

Parameters data : 2D array of ints

Sweep that will be checked for objects. Sweep has already been converted to binary 0s/1s based on user-supplied thresholds.

Returns labels : 2D array of ints

Numeric object labels, corrected for the periodic boundary. Zero values mean no object at that location.

nobj : int

Number of distinct objects identified in sweep.

`pyart.correct.despeckle.`**`_append_labels`**(*labels*, *label_storage*)

Appends consecutive sweeps of labels, creating a multi-sweep 2D array. Typically called iteratively.

Parameters **labels** : 2D array of ints

Sweep containing object labels.

label_storage : Empty list or 2D array of ints

Array to append new sweep of labels to.

Returns **label_storage** : 2D array of ints

Updated array of object labels

pyart.correct.despeckle.**_check_for_360**(*az*, *delta*)

Check if an array of azimuths indicates the sweep is a full 360 PPI. This should also spot RHIs (effectively, a narrow azimuth sector sweep).

Parameters **az** : array of int or float

Azimuths in the sweep

delta : int or float

Size of allowable gap near PPI edges, in deg, to consider it full 360.

Returns **Flag** : bool

True - Sweep is a 360 PPI

False - Sweep is not a 360 PPI.

pyart.correct.despeckle.**_check_sweeps**(*sweeps*, *radar*)

Parse the sweeps keyword and convert it to a list of ints. The output will be iterated over.

Parameters **sweeps** : int or list of ints or None

Sweep numbers to put into an iterable list. If None, all sweeps in the radar object will be examined.

radar : pyart.core.Radar object

Radar object to query.

Returns **sweeps** : list of ints

Sweep numbers as an iterable list

pyart.correct.despeckle.**_check_threshold**(*threshold*)

Parse the threshold keyword and return the lower and upper boundaries for the object search.

Parameters **threshold** : int or float, or 2-element tuple of ints or floats

Threshold values above (if single value) or between (if tuple) for objects to be identified.

Returns **tlo** : int or float

Lower bound for the threshold. Values below this will not be included in the hunt for objects.

thi : int or float or None

Upper bound for the threshold. Values above this will not be included in the hunt for objects. None means no upper bound.

pyart.correct.despeckle.**_generate_dict**(*label_storage*)

Build the dictionary that includes all the object label information. If the entire Radar object was searched, the dictionary is ready to be added as a new field.

Parameters **label_storage** : 2D array of ints

Object labels as a 2D array

Returns label_dict : dict

Dictionary containing object labels and associated metadata

`pyart.correct.despeckle._get_data`(*radar*, *iswp*, *field*, *tlo*, *thi*, *window*, *gatefilter=None*)

Get data for a field from a given sweep in a Radar object. Data are smoothed if desired, then converted to binary 0s/1s based on whether valid values are present.

Parameters radar : pyart.core.Radar object

Radar object to query.

iswp : int

Sweep number to query.

field : str

Name of field to investigate for speckles.

tlo : int or float

Lower bound for the threshold. Values below this will not be included in the hunt for objects.

thi : int or float or None

Upper bound for the threshold. Values above this will not be included in the hunt for objects. None means no upper bound.

window : int or None

Number of gates included in a smoothing box filter along a ray. If None, no smoothing is done.

Returns data : 2D array of ints

Sweep as array of binary 0s/1s based on whether valid values exist.

Other Parameters gatefilter : None or pyart.filters.GateFilter object

Py-ART GateFilter object to apply before labeling objects. If None, no filtering will be performed.

`pyart.correct.despeckle._get_labels`(*data*)

Identify all the contiguous objects in a sweep. Contiguous means corners or sides of gates touch. Uses scipy.ndimage.label.

Parameters data : 2D array of ints

Sweep that will be checked for objects. Sweep has already been converted to binary 0s/1s based on user-supplied thresholds.

Returns labels : 2D array of ints

Numeric object labels. Zero values mean no object at that location.

nobj : int

Number of distinct objects identified in sweep.

`pyart.correct.despeckle._smooth_data`(*data*, *window*)

Perform box filtering along each ray of a sweep, and return the smoothed field. Uses scipy.signal.convolve2d which provides excellent performance.

Parameters data : 2D array of ints or floats

Sweep of data for a specific field. Will be masked.

window : int or None

Number of gates included in a smoothing box filter along a ray. If None, no smoothing is done.

Returns data : 2D array of ints or floats

Smoothed sweep of data.

`pyart.correct.despeckle.`**`despeckle_field`**(*radar*, *field*, *label_dict=None*, *threshold=-100*, *size=10*, *gatefilter=None*, *delta=5.0*)

Despeckle a radar volume by identifying small objects in each scan and masking them out. User can define which field to investigate, as well as various thresholds to use on that field and any objects found within. Requires scipy to be installed, and returns a GateFilter object.

Parameters radar : pyart.core.Radar object

Radar object to query.

field : str

Name of field to investigate for speckles.

Returns gatefilter : pyart.filters.GateFilter object

Py-ART GateFilter object that includes the despeckling mask

Other Parameters label_dict : dict or None, optional

Dictionary that is produced by find_objects. If None, find_objects will be called to produce it.

threshold : int or float, or 2-element tuple of ints or floats

Threshold values above (if single value) or between (if tuple) for objects to be identified. Default value assumes reflectivity.

size : int, optional

Number of contiguous gates in an object, below which it is a speckle.

gatefilter : None or pyart.filters.GateFilter object

Py-ART GateFilter object to which to add the despeckling mask. The GateFilter object will be permanently modified with the new filtering. If None, creates a new GateFilter.

delta : int or float, optional

Size of allowable gap near PPI edges, in deg, to consider it full 360. If gap is small, then PPI edges will be checked for matching objects.

`pyart.correct.despeckle.`**`find_objects`**(*radar*, *field*, *threshold*, *sweeps=None*, *smooth=None*, *gatefilter=None*, *delta=5.0*)

Find objects (i.e., contiguous gates) in one or more sweeps that match thresholds. Filtering & smoothing are available prior to labeling objects. In addition, periodic boundaries are accounted for if they exist (e.g., 360-deg PPIs). Requires scipy to be installed.

Parameters radar : pyart.core.Radar object

Radar object to query.

field : str

Name of field to investigate for objects.

threshold : int or float, or 2-element tuple of ints or floats

Threshold values above (if single value) or between (if tuple) for objects to be identified.

Returns label_dict : dict

Dictionary that contains all the labeled objects. If this function is performed on the full Radar object, then the dict is ready to be added as a field.

Other Parameters sweeps : int or array of ints or None, optional

Sweep numbers to examine. If None, all sweeps are examined.

smooth : int or None, optional

Number of gates included in a smoothing box filter along a ray. If None, no smoothing is done prior to labeling objects.

gatefilter : None or pyart.filters.GateFilter object

Py-ART GateFilter object to apply before labeling objects. If None, no filtering will be performed. Note: Filtering always occurs before smoothing.

delta : int or float, optional

Size of allowable gap near PPI edges, in deg, to consider it full 360. If gap is small, then PPI edges will be checked for matching objects along the periodic boundary.

CHAPTER

FORTYFIVE

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_ray(radar[, ind_rmin, ...])	Public method Alternative determination of the system phase.
correct_sys_phase(radar[, ind_rmin, ...])	correction of the system offset. Public method
smooth_phidp_single_window(radar[, ...])	correction of the system offset and smoothing using one window
smooth_phidp_double_window(radar[, ...])	correction of the system offset and smoothing using two window
smooth_masked(raw_data[, wind_len, ...])	smoothes the data using a rolling median window.
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.
construct_B_vectors(phidp_mod, z_mod, filt)	Construct B vectors.
LP_solver_cvxopt(A_Matrix, B_vectors, weights)	Solve the Linear Programming problem given in Giangrande et al, 2012 using the CVXOPT module.
LP_solver_pyglpk(A_Matrix, B_vectors, weights)	Solve the Linear Programming problem given in Giangrande et al, 2012 using the PyGLPK module.
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 using the CyLP module using multiple processes.
LP_solver_cylp(A_Matrix, B_vectors, weights)	Solve the Linear Programming problem given in Giangrande et al, 2012 using the CyLP module.
phase_proc_lp(radar, offset[, debug, ...])	Phase process using a LP method [1].
_det_sys_phase(ncp, rhv, phidp, last_ray_idx)	Determine the system phase, see det_sys_phase().
_det_sys_phase_ray(phidp, refl, nrays, ngates)	Private method Alternative determination of the system phase.
_correct_sys_phase(phidp, refl, nsweeps, ...)	correction of the system offset. Private method

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_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.**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_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.**_correct_sys_phase**(*phidp, refl, nsweeps, nrays, ngates, start_sweep, end_sweep, ind_rmin=10, ind_rmax=500, min_rcons=11, zmin=20.0, zmax=40.0*)

correction of the system offset. Private method

Parameters phidp : masked array

the phidp field to correct

refl : masked array

the reflectivity field

nsweeps, nrays, ngates : int

number of sweeps, total rays and gates per ray

start_sweep, end_sweep : int array

index of the starting and ending ray of each sweep

ind_rmin, ind_rmax : int

the minimum and maximum range indexes to use in the estimation

min_rcons : int

the number of consecutive range bins to consider a precipitation cell valid

Returns corr_phidp : masked array

The corrected phidp field

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.**_det_sys_phase_ray**(*phidp, refl, nrays, ngates, ind_rmin=10, ind_rmax=500, min_rcons=11, zmin=20.0, zmax=40.0*)

Private method Alternative determination of the system phase. Assumes that the valid gates of phidp are only precipitation. A system phase value is found for each ray.

Parameters phidp : masked array

the phidp data

refl : masked array

the reflectivity data

nrays : int

number of rays in phidp

ngates : int

number of gates per ray

ind_rmin, ind_rmax : int

Min and max range index where to look for continuous precipitation

min_rcons : int

The minimum number of consecutive gates to consider it a rain cell.

zmin, zmax : float

Returns **phidp0** : array of floats

Estimate of the system phase at each ray

first_gates : array of ints

The first gate where PhiDP is valid

`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`**(*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.**correct_sys_phase**(*radar*, *ind_rmin=10*, *ind_rmax=500*, *min_rcons=11*, *zmin=20.0*, *zmax=40.0*, *psidp_field=None*, *refl_field=None*, *phidp_field=None*)

correction of the system offset. Public method

Parameters **radar** : Radar

Radar object for which to determine the system phase.

ind_rmin, ind_rmax : int

Min and max range index where to look for continuous precipitation

min_rcons : int

The minimum number of consecutive gates to consider it a rain cell.

zmin, zmax : float

Minimum and maximum reflectivity to consider it a rain cell

psidp_field : str

Field name within the radar object which represent the differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

refl_field : str

Field name within the radar object which represent the reflectivity. A value of None will use the default field name as defined in the Py-ART configuration file.

phidp_field : str

Field name within the radar object which represent the corrected differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns **phidp_dict** : dict

The corrected phidp field

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.`**`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_ray`**(*radar*, *ind_rmin=10*, *ind_rmax=500*, *min_rcons=11*, *zmin=20.0*, *zmax=40.0*, *phidp_field=None*, *refl_field=None*)

Public method Alternative determination of the system phase. Assumes that the valid gates of phidp are only precipitation. A system phase value is found for each ray.

Parameters **radar** : Radar

Radar object for which to determine the system phase.

ind_rmin, ind_rmax : int

Min and max range index where to look for continuous precipitation

min_rcons : int

The minimum number of consecutive gates to consider it a rain cell.

zmin, zmax : float

The minimum and maximum reflectivity to consider the radar bin suitable precipitation

phidp_field : str

Field name within the radar object which represent the differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

refl_field : str

Field name within the radar object which represent the reflectivity. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns **phidp0_dict** : dict

Estimate of the system phase at each ray and metadata

first_gates_dict : dict

The first gate where PhiDP is valid and metadata

`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.`**`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.`**`noise`**`(`*line*, *wl=11*`)`

Return the noise after smoothing.

`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='cylp'*, *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.

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', 'median' 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**(*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', 'median' 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.**smooth_masked**(*raw_data*, *wind_len=11*, *min_valid=6*, *wind_type='median'*)

smoothes the data using a rolling median window. data with less than n valid points is masked

Parameters raw_data : float masked array

The data to smooth.

window_len : float

Length of the moving window

min_valid : float

Minimum number of valid points for the smoothing to be valid

wind_type : str

type of window. Can be median or mean

Returns data_smooth : float masked array smoothed data

pyart.correct.phase_proc.**smooth_phidp_double_window**(*radar*, *ind_rmin=10*, *ind_rmax=500*, *min_rcons=11*, *zmin=20.0*, *zmax=40*, *swind_len=11*, *smin_valid=6*, *lwind_len=31*, *lmin_valid=16*, *zthr=40.0*, *psidp_field=None*, *refl_field=None*, *phidp_field=None*)

correction of the system offset and smoothing using two window

Parameters **radar** : Radar

Radar object for which to determine the system phase.

ind_rmin, ind_rmax : int

Min and max range index where to look for continuous precipitation

min_rcons : int

The minimum number of consecutive gates to consider it a rain cell.

zmin, zmax : float

Minimum and maximum reflectivity to consider it a rain cell

swind_len : int

Length of the short moving window used to smooth

smin_valid : int

Minimum number of valid bins to consider the short window smooth data valid

lwind_len : int

Length of the long moving window used to smooth

lmin_valid : int

Minimum number of valid bins to consider the long window smooth data valid

zthr : float

reflectivity value above which the short window is used

psidp_field : str

Field name within the radar object which represent the differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

refl_field : str

Field name within the radar object which represent the reflectivity. A value of None will use the default field name as defined in the Py-ART configuration file.

phidp_field : str

Field name within the radar object which represent the corrected differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns **phidp_dict** : dict

The corrected phidp field

`pyart.correct.phase_proc.`**`smooth_phidp_single_window`**(*radar*, *ind_rmin=10*, *ind_rmax=500*, *min_rcons=11*, *zmin=20.0*, *zmax=40*, *wind_len=11*, *min_valid=6*, *psidp_field=None*, *refl_field=None*, *phidp_field=None*)

correction of the system offset and smoothing using one window

Parameters **radar** : Radar

Radar object for which to determine the system phase.

ind_rmin, ind_rmax : int

Min and max range index where to look for continuous precipitation

min_rcons : int

The minimum number of consecutive gates to consider it a rain cell.

zmin, zmax : float

Minimum and maximum reflectivity to consider it a rain cell

wind_len : int

Length of the moving window used to smooth

min_valid : int

Minimum number of valid bins to consider the smooth data valid

psidp_field : str

Field name within the radar object which represent the differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

refl_field : str

Field name within the radar object which represent the reflectivity. A value of None will use the default field name as defined in the Py-ART configuration file.

phidp_field : str

Field name within the radar object which represent the corrected differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns **phidp_dict** : dict

The corrected phidp field

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

Return the signal to noise ratio after smoothing.

`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.**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.

CHAPTER

FORTYSIX

PYART.CORRECT.REGION_DEALIAS

Region based dealiasing using a dynamic network reduction for region joining.

dealias_region_based(radar[, ref_vel_field, ...])	Dealias Doppler velocities using a region based algorithm.
_find_regions(vel, gfilter, limits)	Find regions of similar velocity.
_find_sweep_interval_splits(nyquist, ...)	Return the interval limits for a given sweep.
_combine_regions(region_tracker, edge_tracker)	Returns True when done.
_edge_sum_and_count(labels, ...)	Find all edges between labels regions.

_RegionTracker(region_sizes)	Tracks the location of radar volume regions contained in each node as the network is reduced.
_EdgeTracker(indices, edge_count, ...)	A class for tracking edges in a dynamic network.

class `pyart.correct.region_dealias.`**`_EdgeTracker`**(*indices*, *edge_count*, *velocities*, *nyquist_interval*, *nnodes*)

Bases: `object`

A class for tracking edges in a dynamic network.

Methods

merge_nodes(base_node, merge_node, foo_edge)	Merge nodes.
pop_edge()	Pop edge with largest weight.
unwrap_node(node, nwrap)	Unwrap a node.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` **= mappingproxy({'__doc__': ' A class for tracking edges in a dynamic network. ', '__module__': 'pyart.corre**

`__dir__`() → list
default dir() implementation

`__eq__`
Return self==value.

`__format__`()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*indices*, *edge_count*, *velocities*, *nyquist_interval*, *nnodes*)
initialize

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.correct.region_dealias'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_combine_edges(*base_edge*, *merge_edge*, *merge_node*, *neighbor_node*)
Combine edges into a single edge.

_reverse_edge_direction(*edge*)
Reverse an edges direction, change alpha and beta.

merge_nodes (*base_node*, *merge_node*, *foo_edge*)
Merge nodes.

pop_edge ()
Pop edge with largest weight. Return node numbers and diff

unwrap_node (*node*, *nwrap*)
Unwrap a node.

class pyart.correct.region_dealias.**_RegionTracker** (*region_sizes*)
Bases: object

Tracks the location of radar volume regions contained in each node as the network is reduced.

Methods

get_node_size(node)	Return the number of gates in a node.
merge_nodes(node_a, node_b)	Merge node b into node a.
unwrap_node(node, nwrap)	Unwrap all gates contained a node.

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__dict__': <attribute '__dict__' of '_RegionTracker' objects>, '__doc__': '\n Tracks the lo

__dir__ () → list
default dir() implementation

__eq__
Return self==value.

__format__ ()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__ (*region_sizes*)
initalize.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.correct.region_dealias'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

get_node_size(*node*)
Return the number of gates in a node.

merge_nodes(*node_a*, *node_b*)
Merge node b into node a.

unwrap_node(*node*, *nwrap*)
Unwrap all gates contained a node.

pyart.correct.region_dealias.**_combine_regions**(*region_tracker*, *edge_tracker*)
Returns True when done.

pyart.correct.region_dealias.**_edge_sum_and_count**(*labels*, *num_masked_gates*, *data*, *rays_wrap_around*, *max_gap_x*, *max_gap_y*)
Find all edges between labels regions.

Returns the indices, count and velocities of all edges.

pyart.correct.region_dealias.**_find_regions**(*vel*, *gfilter*, *limits*)
Find regions of similar velocity.

For each pair of values in the limits array (or list) find all connected velocity regions within these limits.

Parameters **vel** : 2D ndarray

Array containing velocity data for a single sweep.

gfilter : 2D ndarray

Filter indicating if a particular gate should be masked. True indicates the gate should be masked (excluded).

limits : array like

Velocity limits for region finding. For each pair of limits, taken from elements i and i+1 of the array, all connected regions with velocities within these limits will be found.

Returns label : ndarray

Interger array with each region labeled by a value. The array ranges from 0 to nfeatures, inclusive, where a value of 0 indicates masked gates and non-zero indicates a region of connected gates.

nfeatures : int

Number of regions found.

pyart.correct.region_dealias.**_find_sweep_interval_splits**(*nyquist*, *interval_splits*, *velocities*, *nsweep*)

Return the interval limits for a given sweep.

pyart.correct.region_dealias.**dealias_region_based**(*radar*, *ref_vel_field=None*, *interval_splits=3*, *interval_limits=None*, *skip_between_rays=100*, *skip_along_ray=100*, *centered=True*, *nyquist_vel=None*, *check_nyquist_uniform=True*, *gatefilter=False*, *rays_wrap_around=None*, *keep_original=False*, *set_limits=True*, *vel_field=None*, *corr_vel_field=None*, ***kwargs*)

Dealias Doppler velocities using a region based algorithm.

Performs Doppler velocity dealiasing by finding regions of similar velocities and unfolding and merging pairs of regions until all regions are unfolded. Unfolding and merging regions is accomplished by modeling the problem as a dynamic network reduction.

Parameters radar : Radar

Radar object containing Doppler velocities to dealias.

ref_vel_field : str or None, optional

Field in radar containing a reference velocity field used to anchor the unfolded velocities once the algorithm completes. Typically this field is created by simulating the radial velocities from wind data from an atmospheric sonding using `pyart.util.simulated_vel_from_profile()`.

interval_splits : int, optional

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

interval_limits : array like or None, optional

Velocity limits used for finding regions of similar velocity. Should cover the entire nyquist interval. None, the default value, will split the Nyquist interval into interval_splits equal sized intervals.

skip_between_rays, skip_along_ray : int, optional

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

centered : bool, optional

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

nyquist_velocity : array like or float, optional

Nyquist velocity in unit identical to those stored in the radar's velocity field, either for each sweep or a single value which will be used for all sweeps. None will attempt to determine this value from the Radar object.

check_nyquist_uniform : bool, optional

True to check if the Nyquist velocities are uniform for all rays within a sweep, False will skip this check. This parameter is ignored when the nyquist_velocity parameter is not None.

gatefilter : GateFilter, None or False, optional.

A GateFilter instance which specified which gates should be ignored when performing de-aliasing. A value of None created this filter from the radar moments using any additional arguments by passing them to `moment_based_gate_filter()`. False, the default, disables filtering including all gates in the dealiasing.

rays_wrap_around : bool or None, optional

True when the rays at the beginning of the sweep and end of the sweep should be interpreted as connected when de-aliasing (PPI scans). False if they edges should not be interpreted as connected (other scan types). None will determine the correct value from the radar scan type.

keep_original : bool, optional

True to retain the original Doppler velocity values at gates where the dealiasing procedure fails or was not applied. False does not replacement and these gates will be masked in the corrected velocity field.

set_limits : bool, optional

True to set valid_min and valid_max elements in the returned dictionary. False will not set these dictionary elements.

vel_field : str, optional

Field in radar to use as the Doppler velocities during dealiasing. None will use the default field name from the Py-ART configuration file.

corr_vel_field : str, optional

Name to use for the dealiased Doppler velocity field metadata. None will use the default field name from the Py-ART configuration file.

Returns **corr_vel** : dict

Field dictionary containing dealiased Doppler velocities. Dealiased array is stored under the 'data' key.

CHAPTER

FORTYSEVEN

PYRAD.CORRECT.SUNLIB

Library to deal with sun measurements

sun_position_pysolar(dt, lat, lon[, refraction])	obtains the sun position in atenna coordinates using the pysolar
sun_position_mfr(dt, lat_deg, lon_deg[, ...])	Calculate the sun position for the given time (dt) at the given position (lat, lon).
equation_of_time(dayjul)	Computes the solar hour for a given julian day.
hour_angle(htime, lon, eqt)	Computes the solar angle at a particular time.
solar_declination(dayjul, htime)	Computes the solar declination.
refraction_correction(es_deg)	Computes the correction that has to be applied to the sun elevation angle
gas_att_sun(es_deg, attg)	Computes the attenuation suffered by the sun signal through the atmosphere
gauss_fit(az_data, az_ref, el_data, el_ref, ...)	estimates a gaussian fit of sun hits data
retrieval_result(sunhits, alpha, beta, par, npar)	computes the physical parameters of the sun retrieval from the results of
sun_power(solar_flux, pulse_width, wavelen, ...)	computes the theoretical sun power detected at the antenna [dBm] as it

pyart.correct.sunlib.**equation_of_time**(*dayjul*)

Computes the solar hour for a given julian day.

Parameters dayjul : double

julian date

Returns eqt : float

hour

pyart.correct.sunlib.**gas_att_sun**(*es_deg*, *attg*)

Computes the attenuation suffered by the sun signal through the atmosphere

Parameters es_deg : float

sun elevation in degrees

attg : float

1-way gas attenuation in dB/km

Returns gas_att_sun : float

the sun attenuation in dB

pyart.correct.sunlib.**gauss_fit**(*az_data*, *az_ref*, *el_data*, *el_ref*, *sunhits*, *npar*, *degree=True*, *do_elcorr=True*)

estimates a gaussian fit of sun hits data

Parameters **az_data, el_data** : float array

azimuth and elevation radar data

az_ref, el_ref : float array

azimuth and elevation sun data

sunhits : float array

sun hits data

npar : int

number of parameters of the fit

degree : boolean

boolean indicating whether the data is in degree or radians

do_elcorr : boolean

indicates whether azimuth data is corrected so that azimuth differences are invalid with elevation

Returns **par** : 1D float array

the fit parameters

alpha: 2D float array

the matrix used in the fit

beta: 1D float array

the vector used in the fit

pyart.correct.sunlib.**hour_angle**(*htime*, *lon*, *eqt*)

Computes the solar angle at a particular time.

Parameters **htime** : double

time in seconds since midnight

lon : float

longitude in degrees

eqt : float

solar time

Returns **angle** : float

the solar angle in radiants

pyart.correct.sunlib.**refraction_correction**(*es_deg*)

Computes the correction that has to be applied to the sun elevation angle to account for refraction

Parameters **es_deg** : float

sun elevation in degrees

Returns **refr** : float

the correction due to refraction in degrees

References

Holleman & Huuskonen, 2013: analytical formulas for refraction of radiowaves from exoatmospheric sources, radio science, vol. 48, 226-231

`pyart.correct.sunlib.`**`retrieval_result`**(*sunhits*, *alpha*, *beta*, *par*, *npar*)

computes the physical parameters of the sun retrieval from the results of a Gaussian fit.

Parameters sunhits : float array

sun hits data

alpha: 2D float array

the matrix used in the fit

beta: 1D float array

the vector used in the fit

par : 1D float array

the fit parameters

npar : int

number of parameters of the fit

Returns val, val_std : float

retrieved value and its standard deviation

az_bias, el_bias : float

retrieved azimuth and elevation antenna bias respect to the sun position

az_width, el_width : float

retrieved azimuth and elevation antenna widths

`pyart.correct.sunlib.`**`solar_declination`**(*dayjul*, *htime*)

Computes the solar declination.

Parameters dayjul : double

julian date

htime : double

time in seconds since midnight

Returns angle : float

the solar declination in radiants

`pyart.correct.sunlib.`**`sun_position_mfr`**(*dt*, *lat_deg*, *lon_deg*, *refraction=True*)

Calculate the sun position for the given time (dt) at the given position (lat, lon).

Parameters dt : datetime object

the time when to look for the sun

lat_deg, lon_deg: floats

latitude and longitude in degrees

refraction : boolean

whether to correct for refraction or not

Returns **elev_sun, azim_sun** : floats

elevation and azimuth angles of the sun respect to the sensor in degrees

pyart.correct.sunlib.**sun_position_pysolar**(*dt*, *lat*, *lon*, *refraction=True*)
obtains the sun position in atenna coordinates using the pysolar library.

Parameters **dt** : datetime object

the time when to look for the sun

lat, lon : float

latitude and longitude of the sensor in degrees

refraction : boolean

whether to correct for refraction or not

Returns **el, az** : float

elevation and azimuth angles of the sun respect to the sensor in degrees

pyart.correct.sunlib.**sun_power**(*solar_flux*, *pulse_width*, *wavelen*, *antenna_gain*, *angle_step*, *beamwidth*)
computes the theoretical sun power detected at the antenna [dBm] as it would be without atmospheric attenuation (sun power at top of the atmosphere

Parameters **solar_flux** : float array

the solar fluxes measured at 10.7 cm wavelength [10e-22 W/(m2 Hz)]

pulse_width : float

pulse width [s]

wavelen : float

radar wavelength [m]

antenna_gain : float

the antenna gain [dB]

angle_step : float

integration angle [deg]

beamwidth : float

3 dB-beamwidth [deg]

Returns **pwr_det** : float array

the detected power

CHAPTER

FORTYEIGHT

PYART.CORRECT.UNWRAP

Dealias using multidimensional phase unwrapping algorithms.

dealias_unwrap_phase(radar[, unwrap_unit, ...])	Dealias Doppler velocities using multi-dimensional phase unwrapping.
_dealias_unwrap_3d(radar, vdata, ...)	Dealias using 3D phase unwrapping (full volume at once).
_dealias_unwrap_2d(radar, vdata, ...)	Dealias using 2D phase unwrapping (sweep-by-sweep).
_dealias_unwrap_1d(vdata, nyquist_vel)	Dealias using 1D phase unwrapping (ray-by-ray)
_verify_unwrap_unit(radar, unwrap_unit)	Verify that the radar supports the requested unwrap unit
_is_radar_cubic(radar)	Test if a radar is cubic (sweeps have the same number of rays).
_is_radar_sweep_aligned(radar[, diff])	Test that all sweeps in the radar sample nearly the same angles.
_is_radar_sequential(radar)	Test if all sweeps in radar are sequentially ordered.
_is_sweep_sequential(radar, sweep_number)	Test if a specific sweep is sequentially ordered.

pyart.correct.unwrap.**_dealias_unwrap_1d**(*vdata*, *nyquist_vel*)

Dealias using 1D phase unwrapping (ray-by-ray)

pyart.correct.unwrap.**_dealias_unwrap_2d**(*radar*, *vdata*, *nyquist_vel*, *gfilter*, *rays_wrap_around*)

Dealias using 2D phase unwrapping (sweep-by-sweep).

pyart.correct.unwrap.**_dealias_unwrap_3d**(*radar*, *vdata*, *nyquist_vel*, *gfilter*, *rays_wrap_around*)

Dealias using 3D phase unwrapping (full volume at once).

pyart.correct.unwrap.**_is_radar_cubic**(*radar*)

Test if a radar is cubic (sweeps have the same number of rays).

pyart.correct.unwrap.**_is_radar_sequential**(*radar*)

Test if all sweeps in radar are sequentially ordered.

pyart.correct.unwrap.**_is_radar_sweep_aligned**(*radar*, *diff=0.1*)

Test that all sweeps in the radar sample nearly the same angles.

Test that the maximum difference in sweep sampled angles is below *diff* degrees. The radar should first be tested to verify that is cubic before calling this function using the _is_radar_cubic function.

pyart.correct.unwrap.**_is_sweep_sequential**(*radar*, *sweep_number*)

Test if a specific sweep is sequentially ordered.

pyart.correct.unwrap.**_verify_unwrap_unit**(*radar*, *unwrap_unit*)

Verify that the radar supports the requested unwrap unit

raises a ValueError if the unwrap_unit is not supported.

pyart.correct.unwrap.**dealias_unwrap_phase**(*radar*, *unwrap_unit='sweep'*, *nyquist_vel=None*, *check_nyquist_uniform=True*, *gatefilter=False*, *rays_wrap_around=None*, *keep_original=False*, *set_limits=True*, *vel_field=None*, *corr_vel_field=None*, *skip_checks=False*, ***kwargs*)

Dealias Doppler velocities using multi-dimensional phase unwrapping.

Parameters **radar** : Radar

Radar object containing Doppler velocities to dealias.

unwrap_unit : {'ray', 'sweep', 'volume'}, optional

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

nyquist_velocity : array like or float, optional

Nyquist velocity in unit identical to those stored in the radar's velocity field, either for each sweep or a single value which will be used for all sweeps. None will attempt to determine this value from the Radar object. The Nyquist velocity of the first sweep is used for all dealiasing unless the unwrap_unit is 'sweep' when the velocities of each sweep are used.

check_nyquist_uniform : bool, optional

True to check if the Nyquist velocities are uniform for all rays within a sweep, False will skip this check. This parameter is ignored when the nyquist_velocity parameter is not None.

gatefilter : GateFilter, None or False, optional.

A GateFilter instance which specified which gates should be ignored when performing de-aliasing. A value of None created this filter from the radar moments using any additional arguments by passing them to `moment_based_gate_filter()`. False, the default, disables filtering including all gates in the dealiasing.

rays_wrap_around : bool or None, optional

True when the rays at the beginning of the sweep and end of the sweep should be interpreted as connected when de-aliasing (PPI scans). False if they edges should not be interpreted as connected (other scan types). None will determine the correct value from the radar scan type.

keep_original : bool, optional

True to retain the original Doppler velocity values at gates where the dealiasing procedure fails or was not applied. False does not replacement and these gates will be masked in the corrected velocity field.

set_limits : bool, optional

True to set valid_min and valid_max elements in the returned dictionary. False will not set these dictionary elements.

vel_field : str, optional

Field in radar to use as the Doppler velocities during dealiasing. None will use the default field name from the Py-ART configuration file.

corr_vel_field : str, optional

Name to use for the dealiased Doppler velocity field metadata. None will use the default field name from the Py-ART configuration file.

skip_checks : bool

True to skip checks verifing that an appropiate unwrap_unit is selected, False retains these checked. Setting this parameter to True is not recommended and is only offered as an option for extreme cases.

Returns corr_vel : dict

Field dictionary containing dealiased Doppler velocities. Dealiased array is stored under the 'data' key.

References

[R1], *[R2]*

CHAPTER

FORTYNINE

PYART.CORRECT._COMMON_DEALIAS

Routines used by multiple dealiasing functions.

`_parse_fields`(vel_field, corr_vel_field)	Parse and return the radar fields for dealiasing.
`_parse_nyquist_vel`(nyquist_vel, radar, ...)	Parse the nyquist_vel parameter, extract from the radar if needed.
`_parse_gatefilter`(gatefilter, radar, **kwargs)	Parse the gatefilter, return a valid GateFilter object.
`_parse_rays_wrap_around`(rays_wrap_around, radar)	Parse the rays_wrap_around parameter.
`_set_limits`(data, nyquist_vel, dic)	Set the valid_min and valid_max keys in dic from dealiased data.

pyart.correct._common_dealias.**_parse_fields**(*vel_field*, *corr_vel_field*)
Parse and return the radar fields for dealiasing.

pyart.correct._common_dealias.**_parse_gatefilter**(*gatefilter*, *radar*, ***kwargs*)
Parse the gatefilter, return a valid GateFilter object.

pyart.correct._common_dealias.**_parse_nyquist_vel**(*nyquist_vel*, *radar*, *check_uniform*)
Parse the nyquist_vel parameter, extract from the radar if needed.

pyart.correct._common_dealias.**_parse_rays_wrap_around**(*rays_wrap_around*, *radar*)
Parse the rays_wrap_around parameter.

pyart.correct._common_dealias.**_set_limits**(*data*, *nyquist_vel*, *dic*)
Set the valid_min and valid_max keys in dic from dealiased data.

CHAPTER

FIFTY

PYART.CORRECT._FAST_EDGE_FINDER

Cython routine for quickly finding edges between connected regions.

_fast_edge_finder	Return the gate indices and velocities of all edges between regions.

class pyart.correct._fast_edge_finder.**_EdgeCollector**

Bases: object

Class for collecting edges, used by _edge_sum_and_count function.

Methods

get_indices_and_velocities	Return the edge indices and velocities.

__class__

alias of type

__delattr__

Implement delattr(self, name).

__dir__() → list

default dir() implementation

__eq__

Return self==value.

__format__()

default object formatter

__ge__

Return self>=value.

__getattribute__

Return getattr(self, name).

__gt__

Return self>value.

__hash__

Return hash(self).

__init__

initalize.

__le__
Return self<=value.

__lt__
Return self<value.

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__pyx_vtable__ = <capsule object NULL>

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

get_indices_and_velocities()
Return the edge indices and velocities.

pyart.correct._fast_edge_finder.**_fast_edge_finder**()
Return the gate indices and velocities of all edges between regions.

CHAPTER

FIFTYONE

PYART.CORRECT._UNWRAP_1D

unwrap_1d	Phase unwrapping using the naive approach.

`pyart.correct._unwrap_1d.unwrap_1d()`

Phase unwrapping using the naive approach.

CHAPTER FIFTYTWO

PYART.CORRECT._UNWRAP_2D

unwrap_2d	2D phase unwrapping.

`pyart.correct._unwrap_2d.`**`unwrap_2d`**`()`

2D phase unwrapping.

CHAPTER

FIFTYTHREE

PYART.CORRECT._UNWRAP_3D

unwrap_3d	3D phase unwrapping.

`pyart.correct._unwrap_3d.`**`unwrap_3d`**`()`

3D phase unwrapping.

CHAPTER

FIFTYFOUR

PYART.RETRIEVE.ECHO_CLASS

Functions for echo classification

`steiner_conv_strat`(grid[, dx, dy, intense, ...])	Partition reflectivity into convective-stratiform using the Steiner et al.
`hydroclass_semisupervised`(radar[, ...])	Classifies precipitation echoes following the approach by
`_standardize`(data, field_name[, mx, mn])	Streches the radar data to -1 to 1 interval
`_assign_to_class`(zh, zdr, kdp, rhohv, relh, ...)	assigns an hydrometeor class to a radar range bin computing
`_get_mass_centers`(freq)	get mass centers for a particular frequency
`_mass_centers_table`()	defines the mass centers look up table for each frequency band.
`_data_limits_table`()	defines the data limits used in the standardization.
`get_freq_band`(freq)	returns the frequency band name (S, C, X, ...)

`pyart.retrieve.echo_class.`**`_assign_to_class`**(*zh, zdr, kdp, rhohv, relh, mass_centers, weights=array([1., 1., 1., 0.75, 0.5])*)

assigns an hydrometeor class to a radar range bin computing the distance between the radar variables an a centroid

Parameters **zh,zdr,kdp,rhohv,relh** : radar field

variables used for assigment normalized to [-1, 1] values

mass_centers : matrix

centroids normalized to [-1, 1] values

weights : array

optional. The weight given to each variable

Returns **hydroclass** : int array

the index corresponding to the assigned class

mind_dist : float array

the minimum distance to the centroids

`pyart.retrieve.echo_class.`**`_data_limits_table`**()

defines the data limits used in the standardization.

Returns **dlimits_dict** : dict

A dictionary with the limits for each variable

pyart.retrieve.echo_class.**_get_mass_centers**(*freq*)
get mass centers for a particular frequency

Parameters **freq** : float
radar frequency [Hz]

Returns **mass_centers** : ndarray 2D
The centroids for each variable and hydrometeor class in (nclasses, nvariables)

pyart.retrieve.echo_class.**_mass_centers_table**()
defines the mass centers look up table for each frequency band.

Returns **mass_centers_dict** : dict
A dictionary with the mass centers for each frequency band

pyart.retrieve.echo_class.**_standardize**(*data*, *field_name*, *mx=None*, *mn=None*)
Streches the radar data to -1 to 1 interval

Parameters **data** : array
radar field

field_name : str
type of field (relH, Zh, ZDR, KDP or RhoHV)

Returns **field_std** : dict
standardized radar data

pyart.retrieve.echo_class.**get_freq_band**(*freq*)
returns the frequency band name (S, C, X, ...)

Parameters **freq** : float
radar frequency [Hz]

Returns **freq_band** : str
frequency band name

pyart.retrieve.echo_class.**hydroclass_semisupervised**(*radar*, *mass_centers=None*, *weights=array([1., 1., 1., 0.75, 0.5])*, *refl_field=None*, *zdr_field=None*, *rhv_field=None*, *kdp_field=None*, *temp_field=None*, *hydro_field=None*)
Classifies precipitation echoes following the approach by Besic et al (2016)

Parameters **radar** : radar
radar object

Returns **hydro** : dict
hydrometeor classification field

Other Parameters **mass_centers** : ndarray 2D
The centroids for each variable and hydrometeor class in (nclasses, nvariables)

weights : ndarray 1D

The weight given to each variable.

refl_field, zdr_field, rhv_field, kdp_field, temp_field : str

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

hydro_field : str

Output. Field name which represents the hydrometeor class field. A value of None will use the default field name as defined in the Py-ART configuration file.

References

Besic, N., Figueras i Ventura, J., Grazioli, J., Gabella, M., Germann, U., and Berne, A.: Hydrometeor classification through statistical clustering of polarimetric radar measurements: a semi-supervised approach, Atmos. Meas. Tech., 9, 4425-4445, doi:10.5194/amt-9-4425-2016, 2016

`pyart.retrieve.echo_class.`**`steiner_conv_strat`**(*grid, dx=None, dy=None, intense=42.0, work_level=3000.0, peak_relation='default', area_relation='medium', bkg_rad=11000.0, use_intense=True, fill_value=None, refl_field=None*)

Partition reflectivity into convective-stratiform using the Steiner et al. (1995) algorithm.

Parameters grid : Grid

Grid containing reflectivity field to partition.

Returns eclass : dict

Steiner convective-stratiform classification dictionary.

Other Parameters dx, dy : float

The x- and y-dimension resolutions in meters, respectively. If None the resolution is determined from the first two axes values.

intense : float

The intensity value in dBZ. Grid points with a reflectivity value greater or equal to the intensity are automatically flagged as convective. See reference for more information.

work_level : float

The working level (separation altitude) in meters. This is the height at which the partitioning will be done, and should minimize bright band contamination. See reference for more information.

peak_relation : 'default' or 'sgp'

The peakedness relation. See reference for more information.

area_relation : 'small', 'medium', 'large', or 'sgp'

The convective area relation. See reference for more information.

bkg_rad : float

The background radius in meters. See reference for more information.

use_intense : bool

True to use the intensity criteria.

fill_value : float

Missing value used to signify bad data points. A value of None will use the default fill value as defined in the Py-ART configuration file.

refl_field : str

Field in grid to use as the reflectivity during partitioning. None will use the default reflectivity field name from the Py-ART configuration file.

References

Steiner, M. R., R. A. Houze Jr., and S. E. Yuter, 1995: Climatological Characterization of Three-Dimensional Storm Structure from Operational Radar and Rain Gauge Data. J. Appl. Meteor., 34, 1978-2007.

CHAPTER

FIFTYFIVE

PYART.RETRIEVE.GATE_ID

`map_profile_to_gates`(profile, heights, radar)	Given a profile of a variable map it to the gates of radar assuming 4/3Re.
`fetch_radar_time_profile`(sonde_dset, radar)	Extract the correct profile from a interpolated sonde.

`pyart.retrieve.gate_id.`**`fetch_radar_time_profile`**(*sonde_dset*, *radar*, *time_key='time'*, *height_key='height'*, *nvars=None*)

Extract the correct profile from a interpolated sonde.

This is an ARM specific method which extract the correct profile out of netCDF Variables from a Interpolated Sonde VAP for the volume start time of a radar object.

Parameters **sonde_dset** : Dataset

Interpolate sonde Dataset.

radar : Radar

Radar object from which the nearest profile will be found.

time_key : string, optional

Key to find a CF startard time variable

height_key : string, optional

Key to find profile height data

nvars : list, optional

NetCDF variable to generated profiles for. If None (the default) all variables with dimension of time, height will be found in ncvars.

Returns **return_dic** : dict

Profiles at the start time of the radar

`pyart.retrieve.gate_id.`**`map_profile_to_gates`**(*profile*, *heights*, *radar*, *toa=None*, *profile_field=None*, *height_field=None*)

Given a profile of a variable map it to the gates of radar assuming 4/3Re.

Parameters **profile** : array

Profile array to map.

heights : array

Monotonically increasing heights in meters with same shape as profile.

radar : Radar

Radar to map to

toa: float, optional

Top of atmosphere, where to use profile up to. If None check for mask and use lowest element, if no mask uses whole profile.

height_field : str

Name to use for height field metadata. None will use the default field name from the Py-ART configuration file.

profile_field : str

Name to use for interpolate profile field metadata. None will use the default field name from the Py-ART configuration file.

Returns height_dict, profile_dict : dict

Field dictionaries containing the height of the gates and the profile interpolated onto the radar gates.

CHAPTER

FIFTYSIX

PYART.RETRIEVE.KDP_PROC

Module for retrieving specific differential phase (KDP) from radar total differential phase (PSIDP) measurements. Total differential phase is a function of propagation differential phase (PHIDP), backscatter differential phase (DELTAHV), and the system phase offset.

`kdp_maesaka`(radar[, gatefilter, method, ...])	Compute the specific differential phase (KDP) from corrected (e.g., unfolded) total differential phase data based on the variational method outlined in Maesaka et al.
`boundary_conditions_maesaka`(radar[, ...])	Determine near range gate and far range gate propagation differential phase boundary conditions.
`_cost_maesaka`(x, psidp_o, bcs, dhv, dr, ...)	Compute the value of the cost functional similar to equations (12)-(15) in Maesaka et al.
`_jac_maesaka`(x, psidp_o, bcs, dhv, dr, Cobs, ...)	Compute the Jacobian (gradient) of the cost functional similar to equations (16)-(18) in Maesaka et al.
`_forward_reverse_phidp`(k, bcs[, verbose])	Compute the forward and reverse direction propagation differential phases from the control variable k and boundary conditions following equations (1) and (7) in Maesaka et al.
`_parse_range_resolution`(radar[, ...])	Parse the radar range gate resolution.
`kdp_leastsquare_single_window`(radar[, ...])	Compute the specific differential phase (KDP) from differential phase data using a piecewise least square method.
`kdp_leastsquare_double_window`(radar[, ...])	Compute the specific differential phase (KDP) from differential phase data using a piecewise least square method.
`leastsquare_method`(phidp, rng_m[, wind_len, ...])	Compute the specific differential phase (KDP) from differential phase data using a piecewise least square method.

`pyart.retrieve.kdp_proc.`**`_cost_maesaka`**(*x*, *psidp_o*, *bcs*, *dhv*, *dr*, *Cobs*, *Clpf*, *finite_order*, *fill_value*, *proc*, *debug=False*, *verbose=False*)

Compute the value of the cost functional similar to equations (12)-(15) in Maesaka et al. (2012).

Parameters x : ndarray

Analysis vector containing control variable k.

psidp_o : ndarray

Total differential phase measurements.

bcs : array_like

The near and far range gate propagation differential phase boundary conditions.

dhv : ndarray

Backscatter differential phase.

dr : float

Range resolution in meters.

Cobs : ndarray

The differential phase measurement constraint weights. The weight should vanish where no differential phase measurements are available.

Clpf : float

The low-pass filter (radial smoothness) constraint weight as in equation (15) of Maesaka et al. (2012).

finite_order : 'low' or 'high'

The finite difference accuracy to use when computing derivatives.

fill_value : float

Value indicating missing or bad data in radar field data.

proc : int

The number of parallel threads (CPUs) to use.

debug : bool, optional

True to print debugging information, False to suppress.

verbose : bool, optional

True to print progress information, False to suppress.

Returns J : float

Value of total cost functional.

`pyart.retrieve.kdp_proc._forward_reverse_phidp`(*k*, *bcs*, *verbose=False*)

Compute the forward and reverse direction propagation differential phases from the control variable k and boundary conditions following equations (1) and (7) in Maesaka et al. (2012).

Parameters k : ndarray

Control variable k of the Maesaka et al. (2012) method. The control variable k is proportional to the square root of specific differential phase.

bcs : array_like

The near and far range gate boundary conditions.

verbose : bool, optional

True to print relevant information, False to suppress.

Returns phidp_f : ndarray

Forward direction propagation differential phase.

phidp_r : ndarray

Reverse direction propagation differential phase.

`pyart.retrieve.kdp_proc._jac_maesaka`(*x*, *psidp_o*, *bcs*, *dhv*, *dr*, *Cobs*, *Clpf*, *finite_order*, *fill_value*, *proc*, *debug=False*, *verbose=False*)

Compute the Jacobian (gradient) of the cost functional similar to equations (16)-(18) in Maesaka et al. (2012).

Parameters x : ndarray

Analysis vector containing control variable k.

psidp_o : ndarray

Total differential phase measurements.

bcs : array_like

The near and far range gate propagation differential phase boundary conditions.

dhv : ndarray

Backscatter differential phase.

dr : float

Range resolution in meters.

Cobs : ndarray

The differential phase measurement constraint weights. The weight should vanish where no differential phase measurements are available.

Clpf : float

The low-pass filter (radial smoothness) constraint weight as in equation (15) of Maesaka et al. (2012).

finite_order : 'low' or 'high'

The finite difference accuracy to use when computing derivatives.

fill_value : float

Value indicating missing or bad data in radar field data.

proc : int

The number of parallel threads (CPUs) to use.

debug : bool, optional

True to print debugging information, False to suppress.

verbose : bool, optional

True to print progress information, False to suppress.

Returns **jac** : ndarray

Jacobian of the cost functional.

`pyart.retrieve.kdp_proc.`**`_parse_range_resolution`**(*radar*, *check_uniform=True*, *atol=1.0*, *verbose=False*)

Parse the radar range gate resolution.

Parameters **radar** : Radar

Radar containing range data.

check_uniform : bool, optional

True to check if all range gates are equally spaced, and if so return a scalar value for range resolution. If False, the resolution between each range gate is returned.

atol : float, optional

The absolute tolerance in meters allowed for discrepancies in range gate spacings. Only applicable when check_uniform is True. This parameter may be necessary to catch instances where range gate spacings differ by a few meters or so.

verbose : bool, optional

True to print the range gate resolution. Only valid if check_uniform is True.

Returns dr : float or ndarray

The radar range gate spacing in meters.

pyart.retrieve.kdp_proc.**boundary_conditions_maesaka**(*radar*, *gatefilter=None*, *n=20*, *psidp_field=None*, *debug=False*, *verbose=False*, ***kwargs*)

Determine near range gate and far range gate propagation differential phase boundary conditions. This follows the method outlined in Maesaka et al. (2012), except instead of using the mean we use the median which is less susceptible to outliers. This function can also be used to estimate the system phase offset.

Parameters radar : Radar

Radar containing total differential phase measurements.

gatefilter : GateFilter

A GateFilter indicating radar gates that should be excluded when analysing differential phase measurements.

n : int, optional

The number of range gates necessary to define the near and far range gate boundary conditions. Maesaka et al. (2012) uses a value of 30. If this value is too small then a spurious spike in specific differential phase close to the radar may be retrieved.

check_outliers : bool, optional

True to check for near range gate boundary condition outliers. Outliers near the radar are primarily the result of ground clutter returns.

psidp_field : str, optional

Field name of total differential phase. If None, the default field name must be specified in the Py-ART configuration file.

debug : bool, optional

True to print debugging information, False to suppress.

verbose : bool, optional

True to print relevant information, False to suppress.

Returns phi_near : ndarray

The near range differential phase boundary condition for each ray.

phi_far : ndarray

The far range differential phase boundary condition for each ray.

range_near : ndarray

The near range gate in meters for each ray.

range_far : ndarray

The far range gate in meters for each ray.

idx_near : ndarray

Index of nearest range gate for each ray.

idx_far : ndarray

Index of furthest range gate for each ray.

`pyart.retrieve.kdp_proc.`**`kdp_leastsquare_double_window`**(*radar*, *swind_len=11*, *smin_valid=6*, *lwind_len=31*, *lmin_valid=16*, *zthr=40.0*, *phidp_field=None*, *refl_field=None*, *kdp_field=None*)

Compute the specific differential phase (KDP) from differential phase data using a piecewise least square method. For optimal results PhiDP should be already smoothed and clutter filtered out.

Parameters **radar** : Radar

Radar object.

swind_len : int

The lenght of the short moving window.

smin_valid : int

Minimum number of valid bins to consider the retrieval valid when using the short moving window

lwind_len : int

The lenght of the long moving window.

lmin_valid : int

Minimum number of valid bins to consider the retrieval valid when using the long moving window

zthr : float

reflectivity value above which the short window is used

phidp_field : str

Field name within the radar object which represent the differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

refl_field : str

Field name within the radar object which represent the reflectivity. A value of None will use the default field name as defined in the Py-ART configuration file.

kdp_field : str

Field name within the radar object which represent the specific differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns **kdp_dict** : dict

Retrieved specific differential phase data and metadata.

`pyart.retrieve.kdp_proc.`**`kdp_leastsquare_single_window`**(*radar*, *wind_len=11*, *min_valid=6*, *phidp_field=None*, *kdp_field=None*)

Compute the specific differential phase (KDP) from differential phase data using a piecewise least square method. For optimal results PhiDP should be already smoothed and clutter filtered out.

Parameters **radar** : Radar

Radar object.

wind_len : int

The lenght of the moving window.

min_valid : int

Minimum number of valid bins to consider the retrieval valid

phidp_field : str

Field name within the radar object which represent the differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

kdp_field : str

Field name within the radar object which represent the specific differential phase shift. A value of None will use the default field name as defined in the Py-ART configuration file.

Returns kdp_dict : dict

Retrieved specific differential phase data and metadata.

`pyart.retrieve.kdp_proc.`**`kdp_maesaka`**(*radar*, *gatefilter=None*, *method='cg'*, *backscatter=None*, *Clpf=1.0*, *length_scale=None*, *first_guess=0.01*, *finite_order='low'*, *fill_value=None*, *proc=1*, *psidp_field=None*, *kdp_field=None*, *phidp_field=None*, *debug=False*, *verbose=False*, ***kwargs*)

Compute the specific differential phase (KDP) from corrected (e.g., unfolded) total differential phase data based on the variational method outlined in Maesaka et al. (2012). This method assumes a monotonically increasing propagation differential phase (PHIDP) with increasing range from the radar, and therefore is limited to rainfall below the melting layer and/or warm clouds at weather radar frequencies (e.g., S-, C-, and X-band). This method currently only supports radar data with constant range resolution.

Following the notation of Maesaka et al. (2012), the primary control variable k is proportional to KDP,

k**2 = 2 * KDP * dr

which, because of the square, assumes that KDP always takes a positive value.

Parameters radar : Radar

Radar containing differential phase field.

gatefilter : GateFilter

A GateFilter indicating radar gates that should be excluded when analysing differential phase measurements.

method : str, optional

Type of scipy.optimize method to use when minimizing the cost functional. The default method uses a nonlinear conjugate gradient algorithm. In Maesaka et al. (2012) they use the Broyden-Fletcher- Goldfarb-Shanno (BFGS) algorithm, however for large functional size (e.g., 100K+ variables) this algorithm is considerably slower than a conjugate gradient algorithm.

backscatter : optional

Define the backscatter differential phase. If None, the backscatter differential phase is set to zero for all range gates. Note that backscatter differential phase can be parameterized using attentuation corrected differential reflectivity.

Clpf : float, optional

The low-pass filter (radial smoothness) constraint weight as in equation (15) of Maesaka et al. (2012).

length_scale : float, optional

Length scale in meters used to bring the dimension and magnitude of the low-pass filter cost functional in line with the observation cost functional. If None, the length scale is set to the range resolution.

first_guess : float, optional

First guess for control variable k. Since k is proportional to the square root of KDP, the first guess should be close to zero to signify a KDP field close to 0 deg/km everywhere. However, the first guess should not be exactly zero in order to avoid convergence criteria after the first iteration. In fact it is recommended to use a value closer to one than zero.

finite_order : 'low' or 'high', optional

The finite difference accuracy to use when computing derivatives.

maxiter : int, optional

Maximum number of iterations to perform during cost functional minimization. The maximum number of iterations are only performed if convergence criteria are not met. For variational schemes such as this one, it is generally not recommended to try and achieve convergence criteria since the values of the cost functional and/or its gradient norm are somewhat arbitrary.

fill_value : float, optional

Value indicating missing or bad data in differential phase field.

proc : int, optional

The number of parallel threads (CPUs) to use. Currently no multiprocessing capability exists.

psidp_field : str, optional

Total differential phase field. If None, the default field name must be specified in the Py-ART configuration file.

kdp_field : str, optional

Specific differential phase field. If None, the default field name must be specified in the Py-ART configuration file.

phidp_field : str, optional

Propagation differential phase field. If None, the default field name must be specified in the Py-ART configuration file.

debug : bool, optional

True to print debugging information, False to suppress.

verbose : bool, optional

True to print relevant information, False to suppress.

Returns kdp_dict : dict

Retrieved specific differential phase data and metadata.

phidpf_dict, phidpr_dict : dict

Retrieved forward and reverse direction propagation differential phase data and metadata.

References

Maesaka, T., Iwanami, K. and Maki, M., 2012: "Non-negative KDP Estimation by Monotone Increasing PHIDP Assumption below Melting Layer". The Seventh European Conference on Radar in Meteorology and Hydrology.

`pyart.retrieve.kdp_proc.`**`leastsquare_method`**(*phidp*, *rng_m*, *wind_len=11*, *min_valid=6*)

Compute the specific differential phase (KDP) from differential phase data using a piecewise least square method. For optimal results PhiDP should be already smoothed and clutter filtered out.

Parameters **phidp** : masked array

phidp field

rng_m : array

radar range in meters

wind_len : int

the window length

min_valid : int

Minimum number of valid bins to consider the retrieval valid

Returns **kdp** : masked array

Retrieved specific differential phase field

CHAPTER FIFTYSEVEN

PYART.RETRIEVE.QPE

Functions for rainfall rate estimation

est_rain_rate_zpoly(radar[, refl_field, ...])	Estimates rainfall rate from reflectivity using a polynomial Z-R relation
est_rain_rate_z(radar[, alpha, beta, ...])	Estimates rainfall rate from reflectivity using a power law
est_rain_rate_kdp(radar[, alpha, beta, ...])	Estimates rainfall rate from kdp using alpha power law
est_rain_rate_a(radar[, alpha, beta, ...])	Estimates rainfall rate from specific attenuation using alpha power law
est_rain_rate_zkdp(radar[, alphaz, betaz, ...])	Estimates rainfall rate from a blending of power law r-kdp and r-z relations.
est_rain_rate_za(radar[, alphaz, betaz, ...])	Estimates rainfall rate from a blending of power law r-alpha and r-z relations.
est_rain_rate_hydro(radar[, alphazr, ...])	Estimates rainfall rate using different relations between R and the
_get_coeff_rkdp(freq)	get the R(kdp) power law coefficients for a particular frequency
_coeff_rkdp_table()	defines the R(kdp) power law coefficients for each frequency band.
_get_coeff_ra(freq)	get the R(A) power law coefficients for a particular frequency
_coeff_ra_table()	defines the R(A) power law coefficients for each frequency band.

pyart.retrieve.qpe.**_coeff_ra_table**()

defines the R(A) power law coefficients for each frequency band.

Returns coeff_ra_dict : dict

A dictionary with the coefficients at each band

pyart.retrieve.qpe.**_coeff_rkdp_table**()

defines the R(kdp) power law coefficients for each frequency band.

Returns coeff_rkdp_dict : dict

A dictionary with the coefficients at each band

pyart.retrieve.qpe.**_get_coeff_ra**(*freq*)

get the R(A) power law coefficients for a particular frequency

Parameters freq : float

radar frequency [Hz]

Returns alpha, beta : floats

the coefficient and exponent of the power law

`pyart.retrieve.qpe._get_coeff_rkdp`(*freq*)

get the R(kdp) power law coefficients for a particular frequency

Parameters **freq** : float

radar frequency [Hz]

Returns **alpha, beta** : floats

the coefficient and exponent of the power law

`pyart.retrieve.qpe.est_rain_rate_a`(*radar*, *alpha=None*, *beta=None*, *a_field=None*, *rr_field=None*)

Estimates rainfall rate from specific attenuation using alpha power law

Parameters **radar** : Radar

Radar object

alpha,beta : floats

Optional. factor (alpha) and exponent (beta) of the power law. If not set the factors are going to be determined according to the radar frequency

a_field : str

name of the specific attenuation field to use

rr_field : str

name of the rainfall rate field

Returns **rain** : dict

Field dictionary containing the rainfall rate.

References

Diederich M., Ryzhkov A., Simmer C., Zhang P. and Tromel S., 2015: Use of Specific Attenuation for Rainfall Measurement at X-Band Radar Wavelenghts. Part I: Radar Calibration and Partial Beam Blockage Estimation. Journal of Hydrometeorology, 16, 487-502.

Ryzhkov A., Diederich M., Zhang P. and Simmer C., 2014: Potential Utilization of Specific Attenuation for Rainfall Estimation, Mitigation of Partial Beam Blockage, and Radar Networking. Journal of Atmospheric and Oceanic Technology, 31, 599-619.

`pyart.retrieve.qpe.est_rain_rate_hydro`(*radar*, *alphazr=0.0376*, *betazr=0.6112*, *alphazs=0.1*, *betazs=0.5*, *alphaa=None*, *betaa=None*, *mp_factor=0.6*, *refl_field=None*, *a_field=None*, *hydro_field=None*, *rr_field=None*, *master_field=None*, *thresh=None*, *thresh_max=False*)

Estimates rainfall rate using different relations between R and the polarimetric variables depending on the hydrometeor type

Parameters **radar** : Radar

Radar object

alphazr,betazr : floats

factor (alpha) and exponent (beta) of the z-r power law for rain.

alphazs,betazs : floats

factor (alpha) and exponent (beta) of the z-s power law for snow.

alphaa,betaa : floats

Optional. factor (alpha) and exponent (beta) of the a-r power law. If not set the factors are going to be determined according to the radar frequency

mp_factor : float

factor applied to z-r relation in the melting layer

refl_field : str

name of the reflectivity field to use

a_field : str

name of the specific attenuation field to use

hydro_field : str

name of the hydrometeor classification field to use

rr_field : str

name of the rainfall rate field

master_field : str

name of the field that is going to act as master. Has to be either refl_field or kdp_field. Default is refl_field

thresh : float

value of the threshold that determines when to use the slave field.

thresh_max : Boolean

If true the master field is used up to the thresh value maximum. Otherwise the master field is not used below thresh value.

Returns rain : dict

Field dictionary containing the rainfall rate.

`pyart.retrieve.qpe.`**`est_rain_rate_kdp`**(*radar*, *alpha=None*, *beta=None*, *kdp_field=None*, *rr_field=None*)

Estimates rainfall rate from kdp using alpha power law

Parameters radar : Radar

Radar object

alpha,beta : floats

Optional. factor (alpha) and exponent (beta) of the power law. If not set the factors are going to be determined according to the radar frequency

kdp_field : str

name of the specific differential phase field to use

rr_field : str

name of the rainfall rate field

Returns rain : dict

Field dictionary containing the rainfall rate.

pyart.retrieve.qpe.**est_rain_rate_z**(*radar*, *alpha=0.0376*, *beta=0.6112*, *refl_field=None*, *rr_field=None*)

Estimates rainfall rate from reflectivity using a power law

Parameters **radar** : Radar

Radar object

alpha,beta : floats

factor (alpha) and exponent (beta) of the power law

refl_field : str

name of the reflectivity field to use

rr_field : str

name of the rainfall rate field

Returns **rain** : dict

Field dictionary containing the rainfall rate.

pyart.retrieve.qpe.**est_rain_rate_za**(*radar*, *alphaz=0.0376*, *betaz=0.6112*, *alphaa=None*, *betaa=None*, *refl_field=None*, *a_field=None*, *rr_field=None*, *master_field=None*, *thresh=None*, *thresh_max=False*)

Estimates rainfall rate from a blending of power law r-alpha and r-z relations.

Parameters **radar** : Radar

Radar object

alphaz,betaz : floats

factor (alpha) and exponent (beta) of the z-r power law.

alphaa,betaa : floats

Optional. factor (alpha) and exponent (beta) of the a-r power law. If not set the factors are going to be determined according to the radar frequency

refl_field : str

name of the reflectivity field to use

a_field : str

name of the specific attenuation field to use

rr_field : str

name of the rainfall rate field

master_field : str

name of the field that is going to act as master. Has to be either refl_field or kdp_field. Default is refl_field

thresh : float

value of the threshold that determines when to use the slave field.

thresh_max : Boolean

If true the master field is used up to the thresh value maximum. Otherwise the master field is not used below thresh value.

Returns rain_master : dict

Field dictionary containing the rainfall rate.

`pyart.retrieve.qpe.est_rain_rate_zkdp`(*radar*, *alphaz=0.0376*, *betaz=0.6112*, *alphakdp=None*, *betakdp=None*, *refl_field=None*, *kdp_field=None*, *rr_field=None*, *master_field=None*, *thresh=None*, *thresh_max=True*)

Estimates rainfall rate from a blending of power law r-kdp and r-z relations.

Parameters radar : Radar

Radar object

alphaz,betaz : floats

factor (alpha) and exponent (beta) of the z-r power law.

alphakdp, betakdp : floats

Optional. factor (alpha) and exponent (beta) of the kdp-r power law. If not set the factors are going to be determined according to the radar frequency

refl_field : str

name of the reflectivity field to use

kdp_field : str

name of the specific differential phase field to use

rr_field : str

name of the rainfall rate field

master_field : str

name of the field that is going to act as master. Has to be either refl_field or kdp_field. Default is refl_field

thresh : float

value of the threshold that determines when to use the slave field.

thresh_max : Boolean

If true the master field is used up to the thresh value maximum. Otherwise the master field is not used below thresh value.

Returns rain_master : dict

Field dictionary containing the rainfall rate.

`pyart.retrieve.qpe.est_rain_rate_zpoly`(*radar*, *refl_field=None*, *rr_field=None*)

Estimates rainfall rate from reflectivity using a polynomial Z-R relation developed at McGill University

Parameters radar : Radar

Radar object

refl_field : str

name of the reflectivity field to use

rr_field : str

name of the rainfall rate field

Returns rain : dict

Field dictionary containing the rainfall rate.

CHAPTER

FIFTYEIGHT

PYART.RETRIEVE.SIMPLE_MOMENT_CALCULATIONS

Simple moment calculations.

`calculate_snr_from_reflectivity`(radar[, ...])	Calculate the signal to noise ratio, in dB, from the reflectivity field.
`compute_noisedBZ`(nrays, noisedBZ_val, range, ...)	Computes noise in dBZ from reference noise value.
`compute_signal_power`(radar[, lmf, attg, ...])	Computes signal power at the antenna in dBm from a reflectivity field.
`compute_snr`(radar[, refl_field, ...])	Computes SNR from a reflectivity field and the noise in dBZ.
`compute_l`(radar[, rhohv_field, l_field])	Computes Rhohv in logarithmic scale according to L=-log10(1-RhoHV)
`compute_cdr`(radar[, rhohv_field, zdr_field, ...])	Computes the Circular Depolarization Ratio
`get_coeff_attg`(freq)	get the 1-way gas attenuation for a particular frequency
`_coeff_attg_table`()	defines the 1-way gas attenuation for each frequency band.

`pyart.retrieve.simple_moment_calculations.`**`_coeff_attg_table`**`()`

defines the 1-way gas attenuation for each frequency band.

Returns **coeff_attg_dict** : dict

A dictionary with the coefficients at each band

`pyart.retrieve.simple_moment_calculations.`**`calculate_snr_from_reflectivity`**`(`*radar*, *refl_field=None*, *snr_field=None*, *toa=25000.0*`)`

Calculate the signal to noise ratio, in dB, from the reflectivity field.

Parameters **radar** : Radar

Radar object from which to retrieve reflectivity field.

refl_field : str, optional

Name of field in radar which contains the reflectivity. None will use the default field name in the Py-ART configuration file.

snr_field : str, optional

Name to use for snr metadata. None will use the default field name in the Py-ART configuration file.

toa : float, optional

Height above which to take noise floor measurements, in meters.

Returns snr : field dictionary

Field dictionary containing the signal to noise ratio.

`pyart.retrieve.simple_moment_calculations.`**`compute_cdr`**(*radar*, *rhohv_field=None*, *zdr_field=None*, *cdr_field=None*)

Computes the Circular Depolarization Ratio

Parameters radar : Radar

radar object

rhohv_field, zdr_field : str

name of the input RhoHV and ZDR fields

cdr_field : str

name of the CDR field

Returns cdr : dict

CDR field

`pyart.retrieve.simple_moment_calculations.`**`compute_l`**(*radar*, *rhohv_field=None*, *l_field=None*)

Computes Rhohv in logarithmic scale according to L=-log10(1-RhoHV)

Parameters radar : Radar

radar object

rhohv_field : str

name of the RhoHV field used for the calculation

l_field : str

name of the L field

Returns l : dict

L field

`pyart.retrieve.simple_moment_calculations.`**`compute_noisedBZ`**(*nrays*, *noisedBZ_val*, *range*, *ref_dist*, *noise_field=None*)

Computes noise in dBZ from reference noise value.

Parameters nrays: int

number of rays in the reflectivity field

noisedBZ_val: float

Estimated noise value in dBZ at reference distance

range: np array of floats

range vector in m

ref_dist: float

reference distance in Km

noise_field: str

name of the noise field to use

Returns **noisedBZ** : dict

the noise field

`pyart.retrieve.simple_moment_calculations.`**`compute_signal_power`**(*radar*, *lmf=None*, *attg=None*, *radconst=None*, *refl_field=None*, *pwr_field=None*)

Computes signal power at the antenna in dBm from a reflectivity field.

Parameters **radar** : Radar

radar object

lmf : float

matched filter losses

attg : float

1-way gas attenuation

radconst : float

radar constant

refl_field : str

name of the reflectivity used for the calculations

pwr_field : str

name of the signal power field

Returns **s_pwr_dict** : dict

power field and metadata

`pyart.retrieve.simple_moment_calculations.`**`compute_snr`**(*radar*, *refl_field=None*, *noise_field=None*, *snr_field=None*)

Computes SNR from a reflectivity field and the noise in dBZ.

Parameters **radar** : Radar

radar object

refl_field, noise_field : str

name of the reflectivity and noise field used for the calculations

snr_field : str

name of the SNR field

Returns **snr** : dict

the SNR field

`pyart.retrieve.simple_moment_calculations.`**`get_coeff_attg`**(*freq*)

get the 1-way gas attenuation for a particular frequency

Parameters **freq** : float

radar frequency [Hz]

Returns **attg** : float

1-way gas attenuation

CHAPTER

FIFTYNINE

PYART.RETRIEVE._KDP_PROC

Cython routines for specific differential phase retrievals.

`lowpass_maesaka_term`	Compute the filter term.
`lowpass_maesaka_jac`	Compute the Jacobian of the filter cost functional.

`pyart.retrieve._kdp_proc.`**`lowpass_maesaka_jac()`**

Compute the Jacobian of the filter cost functional.

Compute the Jacobian of the low-pass filter cost functional similar to equation (18) in Maesaka et al. (2012). This function does not currently support radars with variable range resolution.

Parameters **d2kdr2** : 2D array of float64

Second-order derivative of the control variable k with respect to range. The control variable k is proportional to the square root of specific differential phase.

dr : float

The range resolution in meters.

Clpf : float

The low-pass filter (radial smoothness) constraint weight.

finite_order : str, 'low' or 'high'

The finite difference accuracy used to compute the second-order range derivative of the control variable k.

dJlpfdk : 2D array of float64

The Jacobian of the low-pass filter cost functional with respect to the control variable k. Updated in place.

`pyart.retrieve._kdp_proc.`**`lowpass_maesaka_term()`**

Compute the filter term.

Compute the low-pass filter term found in Maesaka et al. (2012). This term represents the second-order derivative of the control variable k with respect to range. This subroutine does not currently support radars with variable range resolution.

Parameters **k** : 2D array of float64

Control variable k defined in Maesaka et al. (2012). This variable is proportional to the square root of specific differential phase.

dr : float

The range resolution in meters.

finite_order : str, 'low' or 'high'

The finite difference accuracy to use when computing the second-order range derivative of the control variable k.

d2kdr2 : 2D array of float64

Second-order derivative of k with respect to range. Updated in place.

CHAPTER

SIXTY

PYART.MAP.GATES_TO_GRID

Generate a Cartesian grid by mapping from radar gates onto the grid.

map_gates_to_grid(radars, grid_shape, ...[, ...])	Map gates from one or more radars to a Cartesian grid.
_detemine_cy_weighting_func(weighting_function)	Determine cython weight function value.
_find_projparams(grid_origin, radars, ...)	Determine the projection parameter.
_parse_gatefilters(gatefilters, radars)	Parse the gatefilters parameter.
_determine_fields(fields, radars)	Determine which field should be mapped to the grid.
_find_offsets(radars, projparams, ...)	Find offset between radars and grid origin.
_find_grid_params(grid_shape, grid_limits)	Find the starting points and step size of the grid.
_parse_roi_func(roi_func, constant_roi, ...)	Return the Radius of influence object.

pyart.map.gates_to_grid.**_detemine_cy_weighting_func**(*weighting_function*)
Determine cython weight function value.

pyart.map.gates_to_grid.**_determine_fields**(*fields*, *radars*)
Determine which field should be mapped to the grid.

pyart.map.gates_to_grid.**_find_grid_params**(*grid_shape*, *grid_limits*)
Find the starting points and step size of the grid.

pyart.map.gates_to_grid.**_find_offsets**(*radars*, *projparams*, *grid_origin_alt*)
Find offset between radars and grid origin.

pyart.map.gates_to_grid.**_find_projparams**(*grid_origin*, *radars*, *grid_projection*)
Determine the projection parameter.

pyart.map.gates_to_grid.**_parse_gatefilters**(*gatefilters*, *radars*)
Parse the gatefilters parameter.

pyart.map.gates_to_grid.**_parse_roi_func**(*roi_func*, *constant_roi*, *z_factor*, *xy_factor*, *min_radius*, *h_factor*, *nb*, *bsp*, *offsets*)
Return the Radius of influence object.

pyart.map.gates_to_grid.**map_gates_to_grid**(*radars*, *grid_shape*, *grid_limits*, *grid_origin=None*, *grid_origin_alt=None*, *grid_projection=None*, *fields=None*, *gatefilters=False*, *map_roi=True*, *weighting_function='Barnes'*, *toa=17000.0*, *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*, ***kwargs*)
Map gates from 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. For each radar gate that is not filtered a radius of influence is calculated. The weighted field values for that gate are added to all grid points within that radius. This routine scaled linearly with the number of radar gates and the effective grid size.

Parameters not defined below are identical to those in `map_to_grid()`.

Parameters **roi_func** : str or RoIFunction

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 Typically 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.

A custom RoIFunction can be defined using the RoIFunction class and defining a get_roi method which returns the radius. For efficient mapping this class should be implemented in Cython.

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.

See also:

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

`map_to_grid` Create grid by finding the radius of influence around each grid point.

CHAPTER

SIXTYONE

PYART.MAP.GRID_MAPPER

Utilities for mapping radar objects to Cartesian grids.

`grid_from_radars`(radars, grid_shape, grid_limits)	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.
`_unify_times_for_radars`(radars)	Return unified start times and units for a number of radars.
`_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 and whose parameters are based on virtual beam size.

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

class `pyart.map.grid_mapper.`**`NNLocator`**(*data*, *leafsize=10*, *algorithm='kd_tree'*)

Bases: `object`

Nearest neighbor locator.

Class for finding the neighbors of a points within a given distance.

Parameters **data** : array_like, (n_sample, n_dimensions)

Locations of points to be indexed. Note that if data is a C-contiguous array of dtype float64 the data will not be copied. Othersize and internal copy will be made.

leafsize : int

The number of points at which the algorithm switches over to brute-force. This can significantly impact the speed of the contruction and query of the tree.

algorithm : 'kd_tree', optional.

Algorithm used to compute the nearest neigbors. 'kd_tree' uses a k-d tree.

Methods

`find_neighbors_and_dists`(q, r)	Find all neighbors and distances within a given distance.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'find_neighbors_and_dists': <function NNLocator.find_neighbors_and_dists>, '__doc__': "

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*data*, *leafsize=10*, *algorithm='kd_tree'*)
initalize.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.map.grid_mapper'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

find_neighbors_and_dists(*q*, *r*)
Find all neighbors and distances within a given distance.

Parameters **q** : n_dimensional tuple

Point to query

r : float

Distance within which neighbors are returned.

Returns **ind** : array of intergers

Indices of the neighbors.

dist : array of floats

Distances to the neighbors.

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**(*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**(*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.

pyart.map.grid_mapper.**_unify_times_for_radars**(*radars*)
Return unified start times and units for a number of radars.

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**(*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.**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

pyart.map.grid_mapper.**grid_from_radars**(*radars*, *grid_shape*, *grid_limits*, *gridding_algo='map_gates_to_grid'*, ***kwargs*)

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

Additional arguments are passed to `map_to_grid()` or `map_gates_to_grid()`.

Parameters **radars** : Radar or 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.

gridding_algo : 'map_to_grid' or 'map_gates_to_grid'

Algorithm to use for gridding. 'map_to_grid' finds all gates within a radius of influence for each grid point, 'map_gates_to_grid' maps each radar gate onto the grid using a radius of influence and is typically significantly faster.

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.

`map_gates_to_grid` Map each gate onto a grid returning a dictionary of radar fields.

pyart.map.grid_mapper.**map_to_grid**(*radars*, *grid_shape*, *grid_limits*, *grid_origin=None*, *grid_origin_alt=None*, *grid_projection=None*, *fields=None*, *gatefilters=False*, *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*, ***kwargs*)

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** : Radar or 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.

grid_origin_alt: float or None

Altitude of grid origin, in meters. None sets the origin to the location of the first radar.

grid_projection : dic or str

Projection parameters defining the map projection used to transform the locations of the radar gates in geographic coordinate to Cartesian coodinates. None will use the default dictionary which uses a native azimutal equidistance projection. See `pyart.core.Grid()` for additional details on this parameter. The geographic coordinates of the radar gates are calculated using the projection defined for each radar. No transformation is used if a grid_origin and grid_origin_alt are None and a single radar is specified.

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.

gatefilters : GateFilter, tuple of GateFilter objects, optional

Specify what gates from each radar will be included in the interpolation onto the grid. Only gates specified in each gatefilters will be included in the mapping to the grid. A single GateFilter can be used if a single Radar is being mapped. A value of False for a specific element or the entire parameter will apply no filtering of gates for a specific radar or all radars (the default). Similarly a value of None will create a GateFilter from the radar moments using any additional arguments by passing them to `moment_based_gate_filter()`.

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'.

Algorithms to use for finding the nearest neighbors. 'kd_tree' is the only valid option.

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.

CHAPTER SIXTYTWO

PYART.MAP._GATE_TO_GRID_MAP

Cython classes and functions for efficient mapping of radar gates to a uniform grid.

GateToGridMapper	A class for efficient mapping of radar gates to a regular grid by weighting all gates within a specified radius of influence by distance.
RoIFunction	A class for storing radius of interest calculations.
ConstantRoI	Constant radius of influence class.
DistRoI	Radius of influence which expands with distance from the radar.
DistBeamRoI	Radius of influence which expands with distance from multiple radars.

class pyart.map._gate_to_grid_map.**ConstantRoI**

Bases: *pyart.map._gate_to_grid_map.RoIFunction*

Constant radius of influence class.

Methods

get_roi	Return contstant radius of influence.

__class__

alias of type

__delattr__

Implement delattr(self, name).

__dir__() → list

default dir() implementation

__eq__

Return self==value.

__format__()

default object formatter

__ge__

Return self>=value.

__getattribute__

Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
intialize.

__le__
Return self<=value.

__lt__
Return self<value.

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__pyx_vtable__ = <capsule object NULL>

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

get_roi()
Return contstant radius of influence.

class pyart.map._gate_to_grid_map.**DistBeamRoI**
Bases: *pyart.map._gate_to_grid_map.RoIFunction*

Radius of influence which expands with distance from multiple radars.

Methods

get_roi	Return the radius of influence for coordinates in meters.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dir__()` → list
default dir() implementation

`__eq__`
Return self==value.

`__format__()`
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__`
initalize.

`__le__`
Return self<=value.

`__lt__`
Return self<value.

`__ne__`
Return self!=value.

`__new__()`
Create and return a new object. See help(type) for accurate signature.

`__pyx_vtable__` **= <capsule object NULL>**

`__reduce__()`
helper for pickle

`__reduce_ex__()`
helper for pickle

`__repr__`
Return repr(self).

`__setattr__`
Implement setattr(self, name, value).

`__sizeof__()` → int
size of object in memory, in bytes

`__str__`
Return str(self).

`__subclasshook__()`
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

get_roi()
Return the radius of influence for coordinates in meters.

class pyart.map._gate_to_grid_map.**DistRoI**
Bases: *pyart.map._gate_to_grid_map.RoIFunction*

Radius of influence which expands with distance from the radar.

Methods

get_roi	Return the radius of influence for coordinates in meters.

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
initalize.

__le__
Return self<=value.

__lt__
Return self<value.

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__pyx_vtable__ = <capsule object NULL>

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

get_roi()
Return the radius of influence for coordinates in meters.

class pyart.map._gate_to_grid_map.**GateToGridMapper**
Bases: object

A class for efficient mapping of radar gates to a regular grid by weighting all gates within a specified radius of influence by distance.

Parameters grid_shape, : tuple of ints

Shape of the grid along the z, y, and x dimensions.

grid_starts, grid_steps : tuple of ints

Starting points and step sizes in meters of the grid along the z, y and x dimensions.

grid_sum, grid_wsum : 4D float32 array

Array for collecting grid weighted values and weights for each grid point and field. Dimension are order z, y, x, and fields. These array are modified in place when mapping gates unto the grid.

Methods

find_roi_for_grid	Fill in the radius of influence for each point in the grid.
map_gates_to_grid	Map radar gates unto the regular grid.

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
initialize.

__le__
Return self<=value.

__lt__
Return self<value.

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__pyx_vtable__ = <capsule object NULL>

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

find_roi_for_grid()
Fill in the radius of influence for each point in the grid.

Parameters roi_array : 3D float32 array

Array which will be filled by the radius of influence for each point in the grid.

roi_func : RoIFunction

Object whose get_roi method returns the radius of influence.

map_gates_to_grid()

Map radar gates unto the regular grid.

The grid_sum and grid_wsum arrays used to initalize the class are update with the mapped gate data.

Parameters **ngates, nrays** : int

Number of gates and rays in the radar volume.

gate_z, gate_y, gate_x : 2D float32 array

Cartesian locations of the gates in meters.

field_data : 3D float32 array

Array containing field data for the radar, dimension are ordered as nrays, ngates, nfields.

field_mask : 3D uint8 array

Array containing masking of the field data for the radar, dimension are ordered as nrays, ngates, nfields.

excluded_gates : 2D uint8 array

Array containing gate masking information. Gates with non-zero values will not be included in the mapping.

offset : tuple of floats

Offset of the radar from the grid origin. Dimension are ordered as z, y, x. Top of atmosphere. Gates above this level are considered.

roi_func : RoIFunction

Object whose get_roi method returns the radius of influence.

weighting_function : int

Function to use for weighting gates based upon distance. 0 for Barnes, 1 for Cressman weighting.

class pyart.map._gate_to_grid_map.**RoIFunction**

Bases: object

A class for storing radius of interest calculations.

Methods

get_roi	Return the radius of influence for coordinates in meters.

__class__

alias of type

__delattr__

Implement delattr(self, name).

__dir__() → list

default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
Initialize self. See help(type(self)) for accurate signature.

__le__
Return self<=value.

__lt__
Return self<value.

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__pyx_vtable__ = <capsule object NULL>

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

get_roi()
Return the radius of influence for coordinates in meters.

CHAPTER

SIXTYTHREE

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 type specs.
_generate_cmap(name, lutsize)	Generates the requested cmap from it's name *name*.

Available colormaps, reversed versions (_r) are also provided, these colormaps are available within matplotlib with names 'pyart_COLORMAP':

- 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
- LangRainbow12

pyart.graph.cm.**_generate_cmap**(*name*, *lutsize*)
    Generates the requested cmap from it's name *name*. The lut size is *lutsize*.

pyart.graph.cm.**_reverse_cmap_spec**(*spec*)
    Reverses cmap specification *spec*, can handle both dict and tuple type specs.

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

pyart.graph.cm.**revcmap**(*data*)
    Can only handle specification *data* in dictionary format.

CHAPTER

SIXTYFOUR

PYART.GRAPH.COMMON

Common graphing routines.

parse_ax(ax)	Parse and return ax parameter.
parse_ax_fig(ax, fig)	Parse and return ax and fig parameters.
parse_cmap(cmap[, field])	Parse and return the cmap parameter.
parse_vmin_vmax(container, field, vmin, vmax)	Parse and return vmin and vmax parameters.
parse_lon_lat(grid, lon, lat)	Parse lat and lon parameters
generate_colorbar_label(standard_name, units)	Generate and return a label for a colorbar.
generate_field_name(container, field)	Return a nice field name for a particular field.
generate_radar_name(radar)	Return radar name.
generate_grid_name(grid)	Return grid name.
generate_radar_time_begin(radar)	Return time begin in datetime instance.
generate_grid_time_begin(grid)	Return time begin in datetime instance.
generate_filename(radar, field, sweep[, ext])	Generate a filename for a plot.
generate_grid_filename(grid, field, level[, ext])	Generate a filename for a plot.
generate_title(radar, field, sweep)	Generate a title for a plot.
generate_grid_title(grid, field, level)	Generate a title for a plot.
generate_longitudinal_level_title(grid, ...)	Generate a title for a plot.
generate_latitudinal_level_title(grid, ...)	Generate a title for a plot.
generate_vpt_title(radar, field)	Generate a title for a VPT plot.
generate_ray_title(radar, field, ray)	Generate a title for a ray plot.
set_limits([xlim, ylim, ax])	Set the display limits.

pyart.graph.common.**generate_az_rhi_title**(*radar*, *field*, *azimuth*)

Generate a title for a pseudo-RHI from PPI azimuth plot.

Parameters **radar** : Radar

Radar structure.

field : str

Field plotted.

azimuth : float

Azimuth plotted.

Returns **title** : str

Plot title.

pyart.graph.common.**generate_colorbar_label**(*standard_name*, *units*)

Generate and return a label for a colorbar.

pyart.graph.common.**generate_field_name**(*container*, *field*)
Return a nice field name for a particular field.

pyart.graph.common.**generate_filename**(*radar*, *field*, *sweep*, *ext='png'*)
Generate a filename for a plot.

Generated filename has form: radar_name_field_sweep_time.ext

Parameters **radar** : Radar

Radar structure.

field : str

Field plotted.

sweep : int

Sweep plotted.

ext : str

Filename extension.

Returns **filename** : str

Filename suitable for saving a plot.

pyart.graph.common.**generate_grid_filename**(*grid*, *field*, *level*, *ext='png'*)
Generate a filename for a plot.

Generated filename has form: grid_name_field_level_time.ext

Parameters **grid** : Grid

Grid structure.

field : str

Field plotted.

level : int

Level plotted.

ext : str

Filename extension.

Returns **filename** : str

Filename suitable for saving a plot.

pyart.graph.common.**generate_grid_name**(*grid*)
Return grid name.

pyart.graph.common.**generate_grid_time_begin**(*grid*)
Return time begin in datetime instance.

pyart.graph.common.**generate_grid_title**(*grid*, *field*, *level*)
Generate a title for a plot.

Parameters **grid** : Grid

Radar structure.

field : str

Field plotted.

level : int

Verical level plotted.

Returns title : str

Plot title.

`pyart.graph.common.`**`generate_latitudinal_level_title`**(*grid*, *field*, *level*)

Generate a title for a plot.

Parameters grid : Grid

Radar structure.

field : str

Field plotted.

level : int

Latitudinal level plotted.

Returns title : str

Plot title.

`pyart.graph.common.`**`generate_longitudinal_level_title`**(*grid*, *field*, *level*)

Generate a title for a plot.

Parameters grid : Grid

Radar structure.

field : str

Field plotted.

level : int

Longitudinal level plotted.

Returns title : str

Plot title.

`pyart.graph.common.`**`generate_radar_name`**(*radar*)

Return radar name.

`pyart.graph.common.`**`generate_radar_time_begin`**(*radar*)

Return time begin in datetime instance.

`pyart.graph.common.`**`generate_ray_title`**(*radar*, *field*, *ray*)

Generate a title for a ray plot.

Parameters radar : Radar

Radar structure.

field : str

Field plotted.

ray : int

Ray plotted.

Returns title : str

Plot title.

`pyart.graph.common.`**`generate_title`**`(`*radar*, *field*, *sweep*`)`

Generate a title for a plot.

Parameters **radar** : Radar

Radar structure.

field : str

Field plotted.

sweep : int

Sweep plotted.

Returns **title** : str

Plot title.

`pyart.graph.common.`**`generate_vpt_title`**`(`*radar*, *field*`)`

Generate a title for a VPT plot.

Parameters **radar** : Radar

Radar structure.

field : str

Field plotted.

Returns **title** : str

Plot title.

`pyart.graph.common.`**`parse_ax`**`(`*ax*`)`

Parse and return ax parameter.

`pyart.graph.common.`**`parse_ax_fig`**`(`*ax*, *fig*`)`

Parse and return ax and fig parameters.

`pyart.graph.common.`**`parse_cmap`**`(`*cmap*, *field=None*`)`

Parse and return the cmap parameter.

`pyart.graph.common.`**`parse_lon_lat`**`(`*grid*, *lon*, *lat*`)`

Parse lat and lon parameters

`pyart.graph.common.`**`parse_vmin_vmax`**`(`*container*, *field*, *vmin*, *vmax*`)`

Parse and return vmin and vmax parameters.

`pyart.graph.common.`**`set_limits`**`(`*xlim=None*, *ylim=None*, *ax=None*`)`

Set the display limits.

Parameters **xlim** : tuple, optional

2-Tuple containing y-axis limits in km. None uses default limits.

ylim : tuple, optional

2-Tuple containing x-axis limits in km. None uses default limits.

ax : Axis

Axis to adjust. None will adjust the current axis.

CHAPTER

SIXTYFIVE

PYART.GRAPH.GRIDMAPDISPLAY

A class for plotting grid objects with a basemap.

GridMapDisplay(grid[, debug])	A class for creating plots from a grid object on top of a Basemap.

class pyart.graph.gridmapdisplay.**GridMapDisplay**(*grid*, *debug=False*)

Bases: object

A class for creating plots from a grid object on top of a Basemap.

Parameters **grid** : Grid

Grid with data which will be used to create plots.

debug : bool

True to print debugging messages, False to supress them.

Attributes

grid	(Grid) Grid object.
debug	(bool) True to print debugging messages, False to supressed them.
basemap	(Basemap) Last plotted basemap, None when no basemap has been plotted.
mappables	(list) List of ContourSet, etc. which have been plotted, useful when adding colorbars.
fields	(list) List of fields which have been plotted.

Methods

generate_filename(field, level[, ext])	Generate a filename for a grid plot.
generate_grid_title(field, level)	Generate a title for a plot.
generate_latitudinal_level_title(field, level)	Generate a title for a plot.
generate_longitudinal_level_title(field, level)	Generate a title for a plot.
get_basemap()	get basemap of the plot
plot_basemap([lat_lines, lon_lines, ...])	Plot a basemap.
plot_colorbar([mappable, orientation, ...])	Plot a colorbar.
plot_crosshairs([lon, lat, line_style, ...])	Plot crosshairs at a given longitude and latitude.

Continued on next page

Table 65.2 – continued from previous page

plot_grid(field[, level, vmin, vmax, norm, ...])	Plot the grid onto the current basemap.
plot_latitude_slice(field[, lon, lat])	Plot a slice along a given latitude.
plot_latitudinal_level(field, y_index[, ...])	Plot a slice along a given latitude.
plot_longitude_slice(field[, lon, lat])	Plot a slice along a given longitude.
plot_longitudinal_level(field, x_index[, ...])	Plot a slice along a given longitude.

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'plot_colorbar': <function GridMapDisplay.plot_colorbar>, 'plot_latitudinal_level': <funct

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*grid*, *debug=False*)
initalize the object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.graph.gridmapdisplay'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_find_nearest_grid_indices(*lon*, *lat*)
Find the nearest x, y grid indices for a given latitude and longitude.

_get_label_x()
Get default label for x units.

_get_label_y()
Get default label for y units.

_get_label_z()
Get default label for z units.

_label_axes_grid(*axis_labels*, *ax*)
Set the x and y axis labels for a grid plot.

_label_axes_latitude(*axis_labels*, *ax*)
Set the x and y axis labels for a latitude slice.

_label_axes_longitude(*axis_labels*, *ax*)
Set the x and y axis labels for a longitude slice.

_make_basemap(*resolution='l'*, *area_thresh=10000*, *auto_range=True*, *min_lon=-92*, *max_lon=-86*, *min_lat=40*, *max_lat=44*, *ax=None*, ***kwargs*)
Make a basemap.

Parameters **auto_range** : bool

True to determine map ranges from the latitude and longitude limits of the grid. False will use the min_lon, max_lon, min_lat, and max_lat parameters for the map range.

min_lat, max_lat, min_lon, max_lon : float

Latitude and longitude ranges for the map projection region in degrees. These parameter are not used if auto_range is True.

resolution : 'c', 'l', 'i', 'h', or 'f'.

Resolution of boundary database to use. See Basemap documentation for details.

area_thresh : int

Basemap area_thresh parameter. See Basemap documentation.

ax : axes or None.

Axis to add the basemap to, if None the current axis is used.

kwargs: Basemap options

Options to be passed to Basemap. If projection is not specified here it uses proj='merc' (mercator).

generate_filename (*field*, *level*, *ext='png'*)
Generate a filename for a grid plot.

Generated filename has form: grid_name_field_level_time.ext

Parameters **field** : str

Field plotted.

level : int

Level plotted.

ext : str

Filename extension.

Returns **filename** : str

Filename suitable for saving a plot.

generate_grid_title (*field*, *level*)
Generate a title for a plot.

Parameters **field** : str

Field plotted.

level : int

Verical level plotted.

Returns **title** : str

Plot title.

generate_latitudinal_level_title (*field*, *level*)
Generate a title for a plot.

Parameters **field** : str

Field plotted.

level : int

Longitudinal level plotted.

Returns **title** : str

Plot title.

generate_longitudinal_level_title (*field*, *level*)
Generate a title for a plot.

Parameters **field** : str

Field plotted.

level : int

Longitudinal level plotted.

Returns **title** : str

Plot title.

get_basemap()

get basemap of the plot

plot_basemap(*lat_lines=None*, *lon_lines=None*, *resolution='l'*, *area_thresh=10000*, *auto_range=True*, *min_lon=-92*, *max_lon=-86*, *min_lat=40*, *max_lat=44*, *ax=None*, ***kwargs*)

Plot a basemap.

Parameters lat_lines, lon_lines : array or None

Locations at which to draw latitude and longitude lines. None will use default values which are resonable for maps of North America.

auto_range : bool

True to determine map ranges from the latitude and longitude limits of the grid. False will use the min_lon, max_lon, min_lat, and max_lat parameters for the map range.

min_lat, max_lat, min_lon, max_lon : float

Latitude and longitude ranges for the map projection region in degrees. These parameter are not used if auto_range is True.

resolution : 'c', 'l', 'i', 'h', or 'f'.

Resolution of boundary database to use. See Basemap documentation for details.

area_thresh : int

Basemap area_thresh parameter. See Basemap documentation.

ax : axes or None.

Axis to add the basemap to, if None the current axis is used.

kwargs: Basemap options

Options to be passed to Basemap. If projection is not specified here it uses proj='merc' (mercator).

plot_colorbar(*mappable=None*, *orientation='horizontal'*, *label=None*, *cax=None*, *ax=None*, *fig=None*, *field=None*)

Plot a colorbar.

Parameters mappable : Image, ContourSet, etc.

Image, ContourSet, etc to which the colorbar applied. If None the last mappable object will be used.

field : str

Field to label colorbar with.

label : str

Colorbar label. None will use a default value from the last field plotted.

orient : str

Colorbar orientation, either 'vertical' [default] or 'horizontal'.

cax : Axis

Axis onto which the colorbar will be drawn. None is also valid.

ax : Axes

Axis onto which the colorbar will be drawn. None is also valid.

fig : Figure

Figure to place colorbar on. None will use the current figure.

plot_crosshairs(*lon=None*, *lat=None*, *line_style='r–'*, *linewidth=2*, *ax=None*)

Plot crosshairs at a given longitude and latitude.

Parameters lon, lat : float

Longitude and latitude (in degrees) where the crosshairs should be placed. If None the center of the grid is used.

line_style : str

Matplotlib string describing the line style.

linewidth : float

Width of markers in points.

ax : axes or None.

Axis to add the crosshairs to, if None the current axis is used.

plot_grid(*field*, *level=0*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=False*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *ax=None*, *fig=None*, ***kwargs*)

Plot the grid onto the current basemap.

Additional arguments are passed to Basemaps's pcolormesh function.

Parameters field : str

Field to be plotted.

level : int

Index corresponding to the height level to be plotted.

vmin, vmax : float

Lower and upper range for the colormesh. If either parameter is None, a value will be determined from the field attributes (if available) or the default values of -8, 64 will be used. Parameters are ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and level parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_latitude_slice(*field*, *lon=None*, *lat=None*, ***kwargs*)

Plot a slice along a given latitude.

For documentation of additional arguments see `plot_latitudinal_level()`.

Parameters field : str

Field to be plotted.

lon, lat : float

Longitude and latitude (in degrees) specifying the slice. If None the center of the grid is used.

plot_latitudinal_level(*field*, *y_index*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *ax=None*, *fig=None*, ***kwargs*)

Plot a slice along a given latitude.

Additional arguments are passed to Basemaps's pcolormesh function.

Parameters field : str

Field to be plotted.

y_index : float

Index of the latitudinal level to plot.

vmin, vmax : float

Lower and upper range for the colormesh. If either parameter is None, a value will be determined from the field attributes (if available) or the default values of -8, 64 will be used. Parameters are ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and lat,lon parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

`plot_longitude_slice`(*field*, *lon=None*, *lat=None*, ***kwargs*)

Plot a slice along a given longitude.

For documentation of additional arguments see `plot_longitudinal_level()`.

Parameters **field** : str

Field to be plotted.

lon, lat : float

Longitude and latitude (in degrees) specifying the slice. If None the center of the grid is used.

plot_longitudinal_level(*field*, *x_index*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *ax=None*, *fig=None*, ***kwargs*)

Plot a slice along a given longitude.

Additional arguments are passed to Basemaps's pcolormesh function.

Parameters **field** : str

Field to be plotted.

x_index : float

Index of the longitudinal level to plot.

vmin, vmax : float

Lower and upper range for the colormesh. If either parameter is None, a value will be determined from the field attributes (if available) or the default values of -8, 64 will be used. Parameters are ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and lat,lon parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

CHAPTER

SIXTYSIX

PYART.GRAPH.RADARDISPLAY

Class for creating plots from Radar objects.

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

class `pyart.graph.radardisplay.`**`RadarDisplay`**(*radar*, *shift=(0.0, 0.0)*)

Bases: `object`

A display object for creating plots from data in a radar object.

Parameters **radar** : Radar

Radar object to use for creating plots.

shift : (float, float)

Shifts in km to offset the calculated x and y locations.

Attributes

plots	(list) List of plots created.
plot_vars	(list) List of fields plotted, order matches plot list.
cbs	(list) List of colorbars created.
origin	(str) 'Origin' or 'Radar'.
shift	((float, float)) Shift in meters.
loc	((float, float)) Latitude and Longitude of radar in degrees.
fields	(dict) Radar fields.
scan_type	(str) Scan type.
ranges	(array) Gate ranges in meters.
azimuths	(array) Azimuth angle in degrees.
elevations	(array) Elevations in degrees.
fixed_angle	(array) Scan angle in degrees.
antenna_transition	(array or None) Antenna transition flag (1 in transition, 0 in transition) or None if no antenna transition.

Methods

generate_az_rhi_title(field, azimuth)	Generate a title for a ray plot.

Continued on next page

Table 66.2 – continued from previous page

generate_filename(field, sweep[, ext])	Generate a filename for a plot.
generate_ray_title(field, ray)	Generate a title for a ray plot.
generate_title(field, sweep)	Generate a title for a plot.
generate_vpt_title(field)	Generate a title for a VPT plot.
label_xaxis_r([ax])	Label the xaxis with the default label for r units.
label_xaxis_rays([ax])	Label the yaxis with the default label for rays.
label_xaxis_time([ax])	Label the yaxis with the default label for rays.
label_xaxis_x([ax])	Label the xaxis with the default label for x units.
label_yaxis_field(field[, ax])	Label the yaxis with the default label for a field units.
label_yaxis_y([ax])	Label the yaxis with the default label for y units.
label_yaxis_z([ax])	Label the yaxis with the default label for z units.
plot(field[, sweep])	Create a plot appropiate for the radar.
plot_azimuth_to_rhi(field, target_azimuth[, ...])	Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.
plot_colorbar([mappable, field, label, ...])	Plot a colorbar.
plot_cross_hair(size[, npts, ax])	Plot a cross-hair on a ppi plot.
plot_grid_lines([ax, col, ls])	Plot grid lines.
plot_label(label, location[, symbol, ...])	Plot a single symbol and label at a given location.
plot_labels(labels, locations[, symbols, ...])	Plot symbols and labels at given locations.
plot_ppi(field[, sweep, mask_tuple, vmin, ...])	Plot a PPI.
plot_range_ring(range_ring_location_km[, ...])	Plot a single range ring.
plot_range_rings(range_rings[, ax, col, ls, lw])	Plot a series of range rings.
plot_ray(field, ray[, format_str, ...])	Plot a single ray.
plot_rhi(field[, sweep, mask_tuple, vmin, ...])	Plot a RHI.
plot_vpt(field[, mask_tuple, vmin, vmax, ...])	Plot a VPT scan.
set_aspect_ratio([aspect_ratio, ax])	Set the aspect ratio for plot area.
set_limits([xlim, ylim, ax])	Set the display limits.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'_set_title': <function RadarDisplay._set_title>, '_label_axes_ppi': <function RadarDisplay

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__ (*radar*, *shift=(0.0, 0.0)*)
Initialize the object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.graph.radardisplay'

__ne__
Return self!=value.

__new__ ()
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__ () → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_get_azimuth_rhi_data_x_y_z (*field*, *target_azimuth*, *edges*, *mask_tuple*, *filter_transitions*, *gatefilter*)
Retrieve and return pseudo-RHI data from a plot function.

_get_colorbar_label (*field*)
Return a colorbar label for a given field.

_get_data (*field*, *sweep*, *mask_tuple*, *filter_transitions*, *gatefilter*)
Retrieve and return data from a plot function.

_get_ray_data (*field*, *ray*, *mask_tuple*, *gatefilter*)
Retrieve and return ray data from a plot function.

_get_vpt_data (*field*, *mask_tuple*, *filter_transitions*)
Retrieve and return vpt data from a plot function.

_get_x_y (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x and y coordinate in km.

_get_x_y_z (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x, y, and z coordinate in km.

_get_x_z (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x and z coordinate in km.

_label_axes_ppi (*axis_labels*, *ax*)
Set the x and y axis labels for a PPI plot.

_label_axes_ray (*axis_labels*, *field*, *ax*)
Set the x and y axis labels for a ray plot.

_label_axes_rhi (*axis_labels*, *ax*)
Set the x and y axis labels for a RHI plot.

_label_axes_vpt (*axis_labels*, *time_axis_flag*, *ax*)
Set the x and y axis labels for a PPI plot.

_set_az_rhi_title (*field*, *azimuth*, *title*, *ax*)
Set the figure title for a ray plot using a default title.

_set_ray_title (*field*, *ray*, *title*, *ax*)
Set the figure title for a ray plot using a default title.

_set_title (*field*, *sweep*, *title*, *ax*)
Set the figure title using a default title.

static _set_vpt_time_axis (*ax*, *date_time_form=None*, *tz=None*)
Set the x axis as a time formatted axis.

Parameters **ax** : Matplotlib axis instance

Axis to plot. None will use the current axis.

date_time_form : str

Format of the time string for x-axis labels.

tz : str

Time zone info to use when creating axis labels (see datetime).

_set_vpt_title (*field*, *title*, *ax*)
Set the figure title using a default title.

generate_az_rhi_title (*field*, *azimuth*)
Generate a title for a ray plot.

Parameters **field** : str

Field plotted.

azimuth : float

Azimuth plotted.

Returns **title** : str

Plot title.

generate_filename (*field*, *sweep*, *ext='png'*)
Generate a filename for a plot.

Generated filename has form: radar_name_field_sweep_time.ext

Parameters **field** : str

Field plotted.

sweep : int

Sweep plotted.

ext : str

Filename extension.

Returns filename : str

Filename suitable for saving a plot.

generate_ray_title (*field*, *ray*)
Generate a title for a ray plot.

Parameters field : str

Field plotted.

ray : int

Ray plotted.

Returns title : str

Plot title.

generate_title (*field*, *sweep*)
Generate a title for a plot.

Parameters field : str

Field plotted.

sweep : int

Sweep plotted.

Returns title : str

Plot title.

generate_vpt_title (*field*)
Generate a title for a VPT plot.

Parameters field : str

Field plotted.

Returns title : str

Plot title.

label_xaxis_r (*ax=None*)
Label the xaxis with the default label for r units.

static **label_xaxis_rays** (*ax=None*)
Label the yaxis with the default label for rays.

static **label_xaxis_time** (*ax=None*)
Label the yaxis with the default label for rays.

label_xaxis_x (*ax=None*)
Label the xaxis with the default label for x units.

label_yaxis_field (*field*, *ax=None*)
Label the yaxis with the default label for a field units.

label_yaxis_y (*ax=None*)
Label the yaxis with the default label for y units.

label_yaxis_z (*ax=None*)
Label the yaxis with the default label for z units.

plot (*field*, *sweep=0*, ***kwargs*)
Create a plot appropiate for the radar.

This function calls the plotting function corresponding to the scan_type of the radar. Additional keywords can be passed to customize the plot, see the appropiate plot function for the allowed keywords.

Parameters field : str

Field to plot.

sweep : int

Sweep number to plot, not used for VPT scans.

See also:

plot_ppi Plot a PPI scan

plot_rhi Plot a RHI scan

plot_vpt Plot a VPT scan

plot_azimuth_to_rhi (*field*, *target_azimuth*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *gatefilter=None*, *reverse_xaxis=None*, *filter_transitions=True*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)
Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters field : str

Field to plot.

target_azimuth : integer

Azimuthal angle in degrees where cross section will be taken.

Other Parameters mask_tuple : (str, float)

2-Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

reverse_xaxis : bool or None

True to reverse the x-axis so the plot reads east to west, False to have east to west. None (the default) will reverse the axis only when all the distances are negative.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_colorbar (*mappable=None*, *field=None*, *label=None*, *orient='vertical'*, *cax=None*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*)

Plot a colorbar.

Parameters **mappable** : Image, ContourSet, etc.

Image, ContourSet, etc to which the colorbar applied. If None the last mappable object will be used.

field : str

Field to label colorbar with.

label : str

Colorbar label. None will use a default value from the last field plotted.

orient : str

Colorbar orientation, either 'vertical' [default] or 'horizontal'.

cax : Axis

Axis onto which the colorbar will be drawn. None is also valid.

ax : Axes

Axis onto which the colorbar will be drawn. None is also valid.

fig : Figure

Figure to place colorbar on. None will use the current figure.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

static **plot_cross_hair** (*size*, *npts=100*, *ax=None*)

Plot a cross-hair on a ppi plot.

Parameters **size** : float

Size of cross-hair in km.

npts: int

Number of points in the cross-hair, higher for better resolution.

ax : Axis

Axis to plot on. None will use the current axis.

static **plot_grid_lines** (*ax=None*, *col='k'*, *ls=':'*)

Plot grid lines.

Parameters **ax** : Axis

Axis to plot on. None will use the current axis.

col : str or value

Color to use for grid lines.

ls : str

Linestyle to use for grid lines.

plot_label (*label*, *location*, *symbol='r+'*, *text_color='k'*, *ax=None*)
Plot a single symbol and label at a given location.

Transforms of the symbol location in latitude and longitude units to x and y plot units is performed using an azimuthal equidistance map projection centered at the radar.

Parameters **label** : str

Label text to place just above symbol.

location : 2-tuples

Tuple of latitude, longitude (in degrees) at which the symbol will be place. The label is placed just above the symbol.

symbol : str

Matplotlib color+marker strings defining the symbol to place at the given location.

text_color : str

Matplotlib color defining the color of the label text.

ax : Axis

Axis to plot on. None will use the current axis.

plot_labels (*labels*, *locations*, *symbols='r+'*, *text_color='k'*, *ax=None*)
Plot symbols and labels at given locations.

Parameters **labels** : list of str

List of labels to place just above symbols.

locations : list of 2-tuples

List of latitude, longitude (in degrees) tuples at which symbols will be place. Labels are placed just above the symbols.

symbols : list of str or str

List of matplotlib color+marker strings defining symbols to place at given locations. If a single string is provided, that symbol will be placed at all locations.

text_color : str

Matplotlib color defining the color of the label text.

ax : Axis

Axis to plot on. None will use the current axis.

plot_ppi (*field*, *sweep=0*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *gatefilter=None*, *filter_transitions=True*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)
Plot a PPI.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

sweep : int, optional

Sweep number to plot.

Other Parameters **mask_tuple** : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

static plot_range_ring(*range_ring_location_km*, *npts=100*, *ax=None*, *col='k'*, *ls='-'*, *lw=2*)

Plot a single range ring.

Parameters **range_ring_location_km** : float

Location of range ring in km.

npts: int

Number of points in the ring, higher for better resolution.

ax : Axis

Axis to plot on. None will use the current axis.

col : str or value

Color to use for range rings.

ls : str

Linestyle to use for range rings.

plot_range_rings(*range_rings*, *ax=None*, *col='k'*, *ls='-'*, *lw=2*)

Plot a series of range rings.

Parameters **range_rings** : list

List of locations in km to draw range rings.

ax : Axis

Axis to plot on. None will use the current axis.

col : str or value

Color to use for range rings.

ls : str

Linestyle to use for range rings.

plot_ray (*field*, *ray*, *format_str='k-'*, *mask_tuple=None*, *ray_min=None*, *ray_max=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None*, *None)*, *gatefilter=None*, *axislabels_flag=True*, *ax=None*, *fig=None*)

Plot a single ray.

Parameters **field** : str

Field to plot.

ray : int

Ray number to plot.

Other Parameters **format_str** : str

Format string defining the line style and marker.

mask_tuple : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

ray_min : float

Minimum ray value, None for default value, ignored if mask_outside is False.

ray_max : float

Maximum ray value, None for default value, ignored if mask_outside is False.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and ray parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_rhi (*field*, *sweep=0*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *reverse_xaxis=None*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *gatefilter=None*, *filter_transitions=True*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a RHI.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

sweep : int,

Sweep number to plot.

Other Parameters **mask_tuple** : (str, float)

2-Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

reverse_xaxis : bool or None

True to reverse the x-axis so the plot reads east to west, False to have east to west. None (the default) will reverse the axis only when all the distances are negative.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_vpt(*field*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *filter_transitions=True*, *time_axis_flag=False*, *date_time_form=None*, *tz=None*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a VPT scan.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

Other Parameters **mask_tuple** : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

time_axis_flag : bool

True to plot the x-axis as time. False uses the index number. Default is False - index-based.

date_time_form : str, optional

Format of the time string for x-axis labels. Parameter is ignored if time_axis_flag is set to False.

tz : str, optional

Time zone info to use when creating axis labels (see datetime). Parameter is ignored if time_axis_flag is set to False.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

static set_aspect_ratio(*aspect_ratio=0.75*, *ax=None*)
Set the aspect ratio for plot area.

static set_limits(*xlim=None*, *ylim=None*, *ax=None*)
Set the display limits.

Parameters **xlim** : tuple, optional

2-Tuple containing y-axis limits in km. None uses default limits.

ylim : tuple, optional

2-Tuple containing x-axis limits in km. None uses default limits.

ax : Axis

Axis to adjust. None will adjust the current axis.

pyart.graph.radardisplay.**_mask_outside**(*flag*, *data*, *v1*, *v2*)
Return the data masked outside of v1 and v2 when flag is True.

CHAPTER

SIXTYSEVEN

PYART.GRAPH.RADARDISPLAY_AIRBORNE

Class for creating plots from Airborne Radar objects.

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

class `pyart.graph.radardisplay_airborne.`**`AirborneRadarDisplay`**(*radar*, *shift=(0.0, 0.0)*)

Bases: `pyart.graph.radardisplay.RadarDisplay`

A display object for creating plots from data in a airborne radar object.

Parameters **radar** : Radar

Radar object to use for creating plots, should be an airborne radar.

shift : (float, float)

Shifts in km to offset the calculated x and y locations.

Attributes

plots	(list) List of plots created.
plot_vars	(list) List of fields plotted, order matches plot list.
cbs	(list) List of colorbars created.
origin	(str) 'Origin' or 'Radar'.
shift	((float, float)) Shift in meters.
loc	((float, float)) Latitude and Longitude of radar in degrees.
fields	(dict) Radar fields.
scan_type	(str) Scan type.
ranges	(array) Gate ranges in meters.
azimuths	(array) Azimuth angle in degrees.
elevations	(array) Elevations in degrees.
fixed_angle	(array) Scan angle in degrees.
rotation	(array) Rotation angle in degrees.
roll	(array) Roll angle in degrees.
drift	(array) Drift angle in degrees.
tilt	(array) Tilt angle in degrees.
heading	(array) Heading angle in degrees.
pitch	(array) Pitch angle in degrees.
altitude	(array) Altitude angle in meters.

Methods

generate_az_rhi_title(field, azimuth)	Generate a title for a ray plot.
generate_filename(field, sweep[, ext])	Generate a filename for a plot.
generate_ray_title(field, ray)	Generate a title for a ray plot.
generate_title(field, sweep)	Generate a title for a plot.
generate_vpt_title(field)	Generate a title for a VPT plot.
label_xaxis_r([ax])	Label the xaxis with the default label for r units.
label_xaxis_rays([ax])	Label the yaxis with the default label for rays.
label_xaxis_time([ax])	Label the yaxis with the default label for rays.
label_xaxis_x([ax])	Label the xaxis with the default label for x units.
label_yaxis_field(field[, ax])	Label the yaxis with the default label for a field units.
label_yaxis_y([ax])	Label the yaxis with the default label for y units.
label_yaxis_z([ax])	Label the yaxis with the default label for z units.
plot(field[, sweep])	Create a plot appropiate for the radar.
plot_azimuth_to_rhi(field, target_azimuth[, ...])	Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.
plot_colorbar([mappable, field, label, ...])	Plot a colorbar.
plot_cross_hair(size[, npts, ax])	Plot a cross-hair on a ppi plot.
plot_grid_lines([ax, col, ls])	Plot grid lines.
plot_label(label, location[, symbol, ...])	Plot a single symbol and label at a given location.
plot_labels(labels, locations[, symbols, ...])	Plot symbols and labels at given locations.
plot_ppi(field[, sweep, mask_tuple, vmin, ...])	Plot a PPI.
plot_range_ring(range_ring_location_km[, ...])	Plot a single range ring.
plot_range_rings(range_rings[, ax, col, ls, lw])	Plot a series of range rings.
plot_ray(field, ray[, format_str, ...])	Plot a single ray.
plot_rhi(field[, sweep, mask_tuple, vmin, ...])	Plot a RHI.
plot_sweep_grid(field[, sweep, mask_tuple, ...])	Plot a sweep as a grid.
plot_vpt(field[, mask_tuple, vmin, vmax, ...])	Plot a VPT scan.
set_aspect_ratio([aspect_ratio, ax])	Set the aspect ratio for plot area.
set_limits([xlim, ylim, ax])	Set the display limits.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__doc__': "\n A display object for creating plots from data in a airborne radar object.\n\n

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*radar*, *shift=(0.0, 0.0)*)
Initialize the object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.graph.radardisplay_airborne'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_get_azimuth_rhi_data_x_y_z(*field*, *target_azimuth*, *edges*, *mask_tuple*, *filter_transitions*, *gatefilter*)
Retrieve and return pseudo-RHI data from a plot function.

_get_colorbar_label(*field*)
Return a colorbar label for a given field.

_get_data(*field*, *sweep*, *mask_tuple*, *filter_transitions*, *gatefilter*)
Retrieve and return data from a plot function.

_get_ray_data(*field*, *ray*, *mask_tuple*, *gatefilter*)
Retrieve and return ray data from a plot function.

_get_vpt_data (*field*, *mask_tuple*, *filter_transitions*)
Retrieve and return vpt data from a plot function.

_get_x_y (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x and y coordinate in km.

_get_x_y_z (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x, y, and z coordinate in km.

_get_x_z (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x and z coordinate in km.

_label_axes_ppi (*axis_labels*, *ax*)
Set the x and y axis labels for a PPI plot.

_label_axes_ray (*axis_labels*, *field*, *ax*)
Set the x and y axis labels for a ray plot.

_label_axes_rhi (*axis_labels*, *ax*)
Set the x and y axis labels for a RHI plot.

_label_axes_vpt (*axis_labels*, *time_axis_flag*, *ax*)
Set the x and y axis labels for a PPI plot.

_set_az_rhi_title (*field*, *azimuth*, *title*, *ax*)
Set the figure title for a ray plot using a default title.

_set_ray_title (*field*, *ray*, *title*, *ax*)
Set the figure title for a ray plot using a default title.

_set_title (*field*, *sweep*, *title*, *ax*)
Set the figure title using a default title.

_set_vpt_time_axis (*ax*, *date_time_form=None*, *tz=None*)
Set the x axis as a time formatted axis.

Parameters **ax** : Matplotlib axis instance

Axis to plot. None will use the current axis.

date_time_form : str

Format of the time string for x-axis labels.

tz : str

Time zone info to use when creating axis labels (see datetime).

_set_vpt_title (*field*, *title*, *ax*)
Set the figure title using a default title.

generate_az_rhi_title (*field*, *azimuth*)
Generate a title for a ray plot.

Parameters **field** : str

Field plotted.

azimuth : float

Azimuth plotted.

Returns **title** : str

Plot title.

generate_filename (*field*, *sweep*, *ext='png'*)
Generate a filename for a plot.

Generated filename has form: radar_name_field_sweep_time.ext

Parameters **field** : str

Field plotted.

sweep : int

Sweep plotted.

ext : str

Filename extension.

Returns **filename** : str

Filename suitable for saving a plot.

generate_ray_title (*field*, *ray*)
Generate a title for a ray plot.

Parameters **field** : str

Field plotted.

ray : int

Ray plotted.

Returns **title** : str

Plot title.

generate_title (*field*, *sweep*)
Generate a title for a plot.

Parameters **field** : str

Field plotted.

sweep : int

Sweep plotted.

Returns **title** : str

Plot title.

generate_vpt_title (*field*)
Generate a title for a VPT plot.

Parameters **field** : str

Field plotted.

Returns **title** : str

Plot title.

label_xaxis_r (*ax=None*)
Label the xaxis with the default label for r units.

label_xaxis_rays (*ax=None*)
Label the yaxis with the default label for rays.

label_xaxis_time (*ax=None*)
Label the yaxis with the default label for rays.

label_xaxis_x (*ax=None*)
Label the xaxis with the default label for x units.

label_yaxis_field (*field*, *ax=None*)
Label the yaxis with the default label for a field units.

label_yaxis_y (*ax=None*)
Label the yaxis with the default label for y units.

label_yaxis_z (*ax=None*)
Label the yaxis with the default label for z units.

plot (*field*, *sweep=0*, ***kwargs*)
Create a plot appropiate for the radar.

This function calls the plotting function corresponding to the scan_type of the radar. Additional keywords can be passed to customize the plot, see the appropiate plot function for the allowed keywords.

Parameters **field** : str

Field to plot.

sweep : int

Sweep number to plot, not used for VPT scans.

See also:

plot_ppi Plot a PPI scan

plot_sweep_grid Plot a RHI or VPT scan

plot_azimuth_to_rhi (*field*, *target_azimuth*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *gatefilter=None*, *reverse_xaxis=None*, *filter_transitions=True*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)
Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

target_azimuth : integer

Azimuthal angle in degrees where cross section will be taken.

Other Parameters **mask_tuple** : (str, float)

2-Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

reverse_xaxis : bool or None

True to reverse the x-axis so the plot reads east to west, False to have east to west. None (the default) will reverse the axis only when all the distances are negative.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_colorbar (*mappable=None*, *field=None*, *label=None*, *orient='vertical'*, *cax=None*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*)

Plot a colorbar.

Parameters **mappable** : Image, ContourSet, etc.

Image, ContourSet, etc to which the colorbar applied. If None the last mappable object will be used.

field : str

Field to label colorbar with.

label : str

Colorbar label. None will use a default value from the last field plotted.

orient : str

Colorbar orientation, either 'vertical' [default] or 'horizontal'.

cax : Axis

Axis onto which the colorbar will be drawn. None is also valid.

ax : Axes

Axis onto which the colorbar will be drawn. None is also valid.

fig : Figure

Figure to place colorbar on. None will use the current figure.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

plot_cross_hair (*size*, *npts=100*, *ax=None*)

Plot a cross-hair on a ppi plot.

Parameters **size** : float

Size of cross-hair in km.

npts: int

Number of points in the cross-hair, higher for better resolution.

ax : Axis

Axis to plot on. None will use the current axis.

plot_grid_lines (*ax=None*, *col='k'*, *ls=':'*)

Plot grid lines.

Parameters **ax** : Axis

Axis to plot on. None will use the current axis.

col : str or value

Color to use for grid lines.

ls : str

Linestyle to use for grid lines.

plot_label(*label*, *location*, *symbol='r+'*, *text_color='k'*, *ax=None*)

Plot a single symbol and label at a given location.

Transforms of the symbol location in latitude and longitude units to x and y plot units is performed using an azimuthal equidistance map projection centered at the radar.

Parameters **label** : str

Label text to place just above symbol.

location : 2-tuples

Tuple of latitude, longitude (in degrees) at which the symbol will be place. The label is placed just above the symbol.

symbol : str

Matplotlib color+marker strings defining the symbol to place at the given location.

text_color : str

Matplotlib color defining the color of the label text.

ax : Axis

Axis to plot on. None will use the current axis.

plot_labels(*labels*, *locations*, *symbols='r+'*, *text_color='k'*, *ax=None*)

Plot symbols and labels at given locations.

Parameters **labels** : list of str

List of labels to place just above symbols.

locations : list of 2-tuples

List of latitude, longitude (in degrees) tuples at which symbols will be place. Labels are placed just above the symbols.

symbols : list of str or str

List of matplotlib color+marker strings defining symbols to place at given locations. If a single string is provided, that symbol will be placed at all locations.

text_color : str

Matplotlib color defining the color of the label text.

ax : Axis

Axis to plot on. None will use the current axis.

plot_ppi (*field*, *sweep=0*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *gatefilter=None*, *filter_transitions=True*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a PPI.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

sweep : int, optional

Sweep number to plot.

Other Parameters **mask_tuple** : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

`plot_range_ring`(*range_ring_location_km*, *npts=100*, *ax=None*, *col='k'*, *ls='-'*, *lw=2*)

Plot a single range ring.

Parameters **range_ring_location_km** : float

Location of range ring in km.

npts: int

Number of points in the ring, higher for better resolution.

ax : Axis

Axis to plot on. None will use the current axis.

col : str or value

Color to use for range rings.

ls : str

Linestyle to use for range rings.

`plot_range_rings`(*range_rings*, *ax=None*, *col='k'*, *ls='-'*, *lw=2*)

Plot a series of range rings.

Parameters **range_rings** : list

List of locations in km to draw range rings.

ax : Axis

Axis to plot on. None will use the current axis.

col : str or value

Color to use for range rings.

ls : str

Linestyle to use for range rings.

plot_ray (*field, ray, format_str='k-', mask_tuple=None, ray_min=None, ray_max=None, mask_outside=False, title=None, title_flag=True, axislabels=(None, None), gatefilter=None, axislabels_flag=True, ax=None, fig=None*)

Plot a single ray.

Parameters field : str

Field to plot.

ray : int

Ray number to plot.

Other Parameters format_str : str

Format string defining the line style and marker.

mask_tuple : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

ray_min : float

Minimum ray value, None for default value, ignored if mask_outside is False.

ray_max : float

Maximum ray value, None for default value, ignored if mask_outside is False.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and ray parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_rhi (*field*, *sweep=0*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *reverse_xaxis=None*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *gatefilter=None*, *filter_transitions=True*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a RHI.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

sweep : int,

Sweep number to plot.

Other Parameters **mask_tuple** : (str, float)

2-Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

reverse_xaxis : bool or None

True to reverse the x-axis so the plot reads east to west, False to have east to west. None (the default) will reverse the axis only when all the distances are negative.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_sweep_grid(*field*, *sweep=0*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *cmap=None*, *norm=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None*, *None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *filter_transitions=True*, *ax=None*, *fig=None*, *gatefilter=None*, ***kwargs*)

Plot a sweep as a grid.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

sweep : int, optional

Sweep number to plot.

Other Parameters **mask_tuple** : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_vpt (*field*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None*, *None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *filter_transitions=True*, *time_axis_flag=False*, *date_time_form=None*, *tz=None*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a VPT scan.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

Other Parameters **mask_tuple** : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

time_axis_flag : bool

True to plot the x-axis as time. False uses the index number. Default is False - index-based.

date_time_form : str, optional

Format of the time string for x-axis labels. Parameter is ignored if time_axis_flag is set to False.

tz : str, optional

Time zone info to use when creating axis labels (see datetime). Parameter is ignored if time_axis_flag is set to False.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

set_aspect_ratio(*aspect_ratio=0.75*, *ax=None*)
Set the aspect ratio for plot area.

set_limits(*xlim=None*, *ylim=None*, *ax=None*)
Set the display limits.

Parameters **xlim** : tuple, optional

2-Tuple containing y-axis limits in km. None uses default limits.

ylim : tuple, optional

2-Tuple containing x-axis limits in km. None uses default limits.

ax : Axis

Axis to adjust. None will adjust the current axis.

CHAPTER

SIXTYEIGHT

PYART.GRAPH.RADARMAPDISPLAY

Class for creating plots on a geographic map using a Radar object.

RadarMapDisplay(radar[, shift])	A display object for creating plots on a geographic map from data in a Radar object.

class `pyart.graph.radarmapdisplay.`**`RadarMapDisplay`**(*radar*, *shift=(0.0, 0.0)*)
Bases: *`pyart.graph.radardisplay.RadarDisplay`*

A display object for creating plots on a geographic map from data in a Radar object.

This class is still a work in progress. Some functionality may not work correctly. Please report any problems to the Py-ART GitHub Issue Tracker.

Parameters **radar** : Radar

Radar object to use for creating plots.

shift : (float, float)

Shifts in km to offset the calculated x and y locations.

Attributes

plots	(list) List of plots created.
plot_vars	(list) List of fields plotted, order matches plot list.
cbs	(list) List of colorbars created.
origin	(str) 'Origin' or 'Radar'.
shift	((float, float)) Shift in meters.
loc	((float, float)) Latitude and Longitude of radar in degrees.
fields	(dict) Radar fields.
scan_type	(str) Scan type.
ranges	(array) Gate ranges in meters.
azimuths	(array) Azimuth angle in degrees.
elevations	(array) Elevations in degrees.
fixed_angle	(array) Scan angle in degrees.
proj	(Proj) Object for performing cartographic transformations specific to the geographic map plotted.
basemap	(Basemap) Last plotted basemap, None when no basemap has been plotted.

Methods

generate_az_rhi_title(field, azimuth)	Generate a title for a ray plot.
generate_filename(field, sweep[, ext])	Generate a filename for a plot.
generate_ray_title(field, ray)	Generate a title for a ray plot.
generate_title(field, sweep)	Generate a title for a plot.
generate_vpt_title(field)	Generate a title for a VPT plot.
label_xaxis_r([ax])	Label the xaxis with the default label for r units.
label_xaxis_rays([ax])	Label the yaxis with the default label for rays.
label_xaxis_time([ax])	Label the yaxis with the default label for rays.
label_xaxis_x([ax])	Label the xaxis with the default label for x units.
label_yaxis_field(field[, ax])	Label the yaxis with the default label for a field units.
label_yaxis_y([ax])	Label the yaxis with the default label for y units.
label_yaxis_z([ax])	Label the yaxis with the default label for z units.
plot(field[, sweep])	Create a plot appropiate for the radar.
plot_azimuth_to_rhi(field, target_azimuth[, ...])	Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.
plot_colorbar([mappable, field, label, ...])	Plot a colorbar.
plot_cross_hair(size[, npts, ax])	Plot a cross-hair on a ppi plot.
plot_grid_lines([ax, col, ls])	Plot grid lines.
plot_label(label, location[, symbol, ...])	Plot a single symbol and label at a given location.
plot_labels(labels, locations[, symbols, ...])	Plot symbols and labels at given locations.
plot_line_geo(line_lons, line_lats[, line_style])	Plot a line segments on the current map given values in lat and lon.
plot_line_xy(line_x, line_y[, line_style])	Plot a line segments on the current map given radar x, y values.
plot_point(lon, lat[, symbol, label_text, ...])	Plot a point on the current map.
plot_ppi(field[, sweep, mask_tuple, vmin, ...])	Plot a PPI.
plot_ppi_map(field[, sweep, mask_tuple, ...])	Plot a PPI volume sweep onto a geographic map.
plot_range_ring(range_ring_location_km[, ...])	Plot a single range ring on the map.
plot_range_rings(range_rings[, ax, col, ls, lw])	Plot a series of range rings.
plot_ray(field, ray[, format_str, ...])	Plot a single ray.
plot_rhi(field[, sweep, mask_tuple, vmin, ...])	Plot a RHI.
plot_vpt(field[, mask_tuple, vmin, vmax, ...])	Plot a VPT scan.
set_aspect_ratio([aspect_ratio, ax])	Set the aspect ratio for plot area.
set_limits([xlim, ylim, ax])	Set the display limits.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'plot_line_xy': <function RadarMapDisplay.plot_line_xy>, '__doc__': "\n A display object

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*radar*, *shift=(0.0, 0.0)*)
Initialize the object.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.graph.radarmapdisplay'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_check_basemap()
Check that basemap is not None, raise ValueError if it is.

_get_azimuth_rhi_data_x_y_z(*field*, *target_azimuth*, *edges*, *mask_tuple*, *filter_transitions*, *gatefilter*)
Retrieve and return pseudo-RHI data from a plot function.

_get_colorbar_label(*field*)
Return a colorbar label for a given field.

_get_data (*field*, *sweep*, *mask_tuple*, *filter_transitions*, *gatefilter*)
Retrieve and return data from a plot function.

_get_ray_data (*field*, *ray*, *mask_tuple*, *gatefilter*)
Retrieve and return ray data from a plot function.

_get_vpt_data (*field*, *mask_tuple*, *filter_transitions*)
Retrieve and return vpt data from a plot function.

_get_x_y (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x and y coordinate in km.

_get_x_y_z (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x, y, and z coordinate in km.

_get_x_z (*sweep*, *edges*, *filter_transitions*)
Retrieve and return x and z coordinate in km.

_label_axes_ppi (*axis_labels*, *ax*)
Set the x and y axis labels for a PPI plot.

_label_axes_ray (*axis_labels*, *field*, *ax*)
Set the x and y axis labels for a ray plot.

_label_axes_rhi (*axis_labels*, *ax*)
Set the x and y axis labels for a RHI plot.

_label_axes_vpt (*axis_labels*, *time_axis_flag*, *ax*)
Set the x and y axis labels for a PPI plot.

_set_az_rhi_title (*field*, *azimuth*, *title*, *ax*)
Set the figure title for a ray plot using a default title.

_set_ray_title (*field*, *ray*, *title*, *ax*)
Set the figure title for a ray plot using a default title.

_set_title (*field*, *sweep*, *title*, *ax*)
Set the figure title using a default title.

_set_vpt_time_axis (*ax*, *date_time_form=None*, *tz=None*)
Set the x axis as a time formatted axis.

Parameters **ax** : Matplotlib axis instance

Axis to plot. None will use the current axis.

date_time_form : str

Format of the time string for x-axis labels.

tz : str

Time zone info to use when creating axis labels (see datetime).

_set_vpt_title (*field*, *title*, *ax*)
Set the figure title using a default title.

generate_az_rhi_title (*field*, *azimuth*)
Generate a title for a ray plot.

Parameters **field** : str

Field plotted.

azimuth : float

Azimuth plotted.

Returns **title** : str

Plot title.

generate_filename (*field*, *sweep*, *ext='png'*)

Generate a filename for a plot.

Generated filename has form: radar_name_field_sweep_time.ext

Parameters **field** : str

Field plotted.

sweep : int

Sweep plotted.

ext : str

Filename extension.

Returns **filename** : str

Filename suitable for saving a plot.

generate_ray_title (*field*, *ray*)

Generate a title for a ray plot.

Parameters **field** : str

Field plotted.

ray : int

Ray plotted.

Returns **title** : str

Plot title.

generate_title (*field*, *sweep*)

Generate a title for a plot.

Parameters **field** : str

Field plotted.

sweep : int

Sweep plotted.

Returns **title** : str

Plot title.

generate_vpt_title (*field*)

Generate a title for a VPT plot.

Parameters **field** : str

Field plotted.

Returns **title** : str

Plot title.

label_xaxis_r (*ax=None*)
Label the xaxis with the default label for r units.

label_xaxis_rays (*ax=None*)
Label the yaxis with the default label for rays.

label_xaxis_time (*ax=None*)
Label the yaxis with the default label for rays.

label_xaxis_x (*ax=None*)
Label the xaxis with the default label for x units.

label_yaxis_field (*field, ax=None*)
Label the yaxis with the default label for a field units.

label_yaxis_y (*ax=None*)
Label the yaxis with the default label for y units.

label_yaxis_z (*ax=None*)
Label the yaxis with the default label for z units.

plot (*field, sweep=0, **kwargs*)
Create a plot appropiate for the radar.

This function calls the plotting function corresponding to the scan_type of the radar. Additional keywords can be passed to customize the plot, see the appropiate plot function for the allowed keywords.

Parameters **field** : str

Field to plot.

sweep : int

Sweep number to plot, not used for VPT scans.

See also:

plot_ppi Plot a PPI scan

plot_rhi Plot a RHI scan

plot_vpt Plot a VPT scan

plot_azimuth_to_rhi (*field, target_azimuth, mask_tuple=None, vmin=None, vmax=None, norm=None, cmap=None, mask_outside=False, title=None, title_flag=True, axislabels=(None, None), axislabels_flag=True, colorbar_flag=True, colorbar_label=None, colorbar_orient='vertical', edges=True, gatefilter=None, reverse_xaxis=None, filter_transitions=True, ax=None, fig=None, ticks=None, ticklabs=None, **kwargs*)
Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

target_azimuth : integer

Azimuthal angle in degrees where cross section will be taken.

Other Parameters **mask_tuple** : (str, float)

2-Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

reverse_xaxis : bool or None

True to reverse the x-axis so the plot reads east to west, False to have east to west. None (the default) will reverse the axis only when all the distances are negative.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_colorbar(*mappable=None*, *field=None*, *label=None*, *orient='vertical'*, *cax=None*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*)

Plot a colorbar.

Parameters **mappable** : Image, ContourSet, etc.

Image, ContourSet, etc to which the colorbar applied. If None the last mappable object will be used.

field : str

Field to label colorbar with.

label : str

Colorbar label. None will use a default value from the last field plotted.

orient : str

Colorbar orientation, either 'vertical' [default] or 'horizontal'.

cax : Axis

Axis onto which the colorbar will be drawn. None is also valid.

ax : Axes

Axis onto which the colorbar will be drawn. None is also valid.

fig : Figure

Figure to place colorbar on. None will use the current figure.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

plot_cross_hair(*size*, *npts=100*, *ax=None*)

Plot a cross-hair on a ppi plot.

Parameters **size** : float

Size of cross-hair in km.

npts: int

Number of points in the cross-hair, higher for better resolution.

ax : Axis

Axis to plot on. None will use the current axis.

plot_grid_lines (*ax=None*, *col='k'*, *ls=':'*)
Plot grid lines.

Parameters **ax** : Axis

Axis to plot on. None will use the current axis.

col : str or value

Color to use for grid lines.

ls : str

Linestyle to use for grid lines.

plot_label (*label*, *location*, *symbol='r+'*, *text_color='k'*, *ax=None*)
Plot a single symbol and label at a given location.

Transforms of the symbol location in latitude and longitude units to x and y plot units is performed using an azimuthal equidistance map projection centered at the radar.

Parameters **label** : str

Label text to place just above symbol.

location : 2-tuples

Tuple of latitude, longitude (in degrees) at which the symbol will be place. The label is placed just above the symbol.

symbol : str

Matplotlib color+marker strings defining the symbol to place at the given location.

text_color : str

Matplotlib color defining the color of the label text.

ax : Axis

Axis to plot on. None will use the current axis.

plot_labels (*labels*, *locations*, *symbols='r+'*, *text_color='k'*, *ax=None*)
Plot symbols and labels at given locations.

Parameters **labels** : list of str

List of labels to place just above symbols.

locations : list of 2-tuples

List of latitude, longitude (in degrees) tuples at which symbols will be place. Labels are placed just above the symbols.

symbols : list of str or str

List of matplotlib color+marker strings defining symbols to place at given locations. If a single string is provided, that symbol will be placed at all locations.

text_color : str

Matplotlib color defining the color of the label text.

ax : Axis

Axis to plot on. None will use the current axis.

plot_line_geo (*line_lons*, *line_lats*, *line_style='r-'*, ***kwargs*)

Plot a line segments on the current map given values in lat and lon.

Additional arguments are passed to basemap.plot.

Parameters **line_lons** : array

Longitude of line segment to plot.

line_lats : array

Latitude of line segment to plot.

line_style : str

Matplotlib compatible string which specifies the line style.

plot_line_xy (*line_x*, *line_y*, *line_style='r-'*, ***kwargs*)

Plot a line segments on the current map given radar x, y values.

Additional arguments are passed to basemap.plot.

Parameters **line_x** : array

X location of points to plot in meters from the radar.

line_y : array

Y location of points to plot in meters from the radar.

line_style : str, optional

Matplotlib compatible string which specifies the line style.

plot_point (*lon*, *lat*, *symbol='ro'*, *label_text=None*, *label_offset=(None*, *None)*, ***kwargs*)

Plot a point on the current map.

Additional arguments are passed to basemap.plot.

Parameters **lon** : float

Longitude of point to plot.

lat : float

Latitude of point to plot.

symbol : str

Matplotlib compatible string which specified the symbol of the point.

label_text : str, optional.

Text to label symbol with. If None no label will be added.

label_offset : [float, float]

Offset in lon, lat degrees for the bottom left corner of the label text relative to the point. A value of None will use 0.01 de

plot_ppi (*field*, *sweep=0*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None, None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *gatefilter=None*, *filter_transitions=True*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a PPI.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

sweep : int, optional

Sweep number to plot.

Other Parameters **mask_tuple** : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_ppi_map(*field*, *sweep=0*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *cmap=None*, *norm=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *ax=None*, *fig=None*, *lat_lines=None*, *lon_lines=None*, *projection='lcc'*, *area_thresh=10000*, *min_lon=None*, *max_lon=None*, *min_lat=None*, *max_lat=None*, *width=None*, *height=None*, *lon_0=None*, *lat_0=None*, *resolution='h'*, *shapefile=None*, *edges=True*, *gatefilter=None*, *basemap=None*, *filter_transitions=True*, *embelish=True*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a PPI volume sweep onto a geographic map.

Additional arguments are passed to Basemap.

Parameters **field** : str

Field to plot.

sweep : int, optional

Sweep number to plot.

Other Parameters **mask_tuple** : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and tilt parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

lat_lines, lon_lines : array or None

Locations at which to draw latitude and longitude lines. None will use default values which are resonable for maps of North America.

projection : str

Map projection supported by basemap. The use of cylindrical projections (mill, merc, etc) is not recommended as they exhibit large distortions at high latitudes. Equal area (aea, laea), conformal (lcc, tmerc, stere) or equidistant projection (aeqd, cass) work well even at high latitudes. The cylindrical equidistant projection (cyl) is not supported as coordinate transformations cannot be performed.

area_thresh : float

Coastline or lake with an area smaller than area_thresh in km^2 will not be plotted.

min_lat, max_lat, min_lon, max_lon : float

Latitude and longitude ranges for the map projection region in degrees.

width, height : float

Width and height of map domain in meters. Only this set of parameters or the previous set of parameters (min_lat, max_lat, min_lon, max_lon) should be specified. If neither set is specified then the map domain will be determined from the extend of the radar gate locations.

lon_0, lat_0 : float

Center of the map domain in degrees. If the default, None is used the latitude and longitude of the radar will be used.

shapefile : str

Filename for a ESRI shapefile as background (untested).

resolution : 'c', 'l', 'i', 'h', or 'f'.

Resolution of boundary database to use. See Basemap documentation for details.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

embelish: bool

True by default. Set to false to supress drawing of coastlines etc.. Use for speedup when specifying shapefiles.

basemap: Basemap instance

If None, create basemap instance using other keyword info. If not None, use the user-specifed basemap instance.

plot_range_ring(*range_ring_location_km*, *npts=360*, *line_style='k-'*, ***kwargs*)

Plot a single range ring on the map.

Additional arguments are passed to basemap.plot.

Parameters **range_ring_location_km** : float

Location of range ring in km.

npts: int

Number of points in the ring, higher for better resolution.

line_style : str

Matplotlib compatible string which specified the line style of the ring.

plot_range_rings (*range_rings*, *ax=None*, *col='k'*, *ls='-'*, *lw=2*)

Plot a series of range rings.

Parameters **range_rings** : list

List of locations in km to draw range rings.

ax : Axis

Axis to plot on. None will use the current axis.

col : str or value

Color to use for range rings.

ls : str

Linestyle to use for range rings.

plot_ray (*field*, *ray*, *format_str='k-'*, *mask_tuple=None*, *ray_min=None*, *ray_max=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None*, *None)*, *gatefilter=None*, *axislabels_flag=True*, *ax=None*, *fig=None*)

Plot a single ray.

Parameters **field** : str

Field to plot.

ray : int

Ray number to plot.

Other Parameters **format_str** : str

Format string defining the line style and marker.

mask_tuple : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

ray_min : float

Minimum ray value, None for default value, ignored if mask_outside is False.

ray_max : float

Maximum ray value, None for default value, ignored if mask_outside is False.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and ray parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_rhi (*field*, *sweep=0*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None*, *None)*, *axislabels_flag=True*, *reverse_xaxis=None*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *gatefilter=None*, *filter_transitions=True*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a RHI.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters **field** : str

Field to plot.

sweep : int,

Sweep number to plot.

Other Parameters **mask_tuple** : (str, float)

2-Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

reverse_xaxis : bool or None

True to reverse the x-axis so the plot reads east to west, False to have east to west. None (the default) will reverse the axis only when all the distances are negative.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

gatefilter : GateFilter

GateFilter instance. None will result in no gatefilter mask being applied to data.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

plot_vpt (*field*, *mask_tuple=None*, *vmin=None*, *vmax=None*, *norm=None*, *cmap=None*, *mask_outside=False*, *title=None*, *title_flag=True*, *axislabels=(None*, *None)*, *axislabels_flag=True*, *colorbar_flag=True*, *colorbar_label=None*, *colorbar_orient='vertical'*, *edges=True*, *filter_transitions=True*, *time_axis_flag=False*, *date_time_form=None*, *tz=None*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, ***kwargs*)

Plot a VPT scan.

Additional arguments are passed to Matplotlib's pcolormesh function.

Parameters field : str

Field to plot.

Other Parameters mask_tuple : (str, float)

Tuple containing the field name and value below which to mask field prior to plotting, for example to mask all data where NCP < 0.5 set mask_tuple to ['NCP', 0.5]. None performs no masking.

vmin : float

Luminance minimum value, None for default value. Parameter is ignored is norm is not None.

vmax : float

Luminance maximum value, None for default value. Parameter is ignored is norm is not None.

norm : Normalize or None, optional

matplotlib Normalize instance used to scale luminance data. If not None the vmax and vmin parameters are ignored. If None, vmin and vmax are used for luminance scaling.

cmap : str or None

Matplotlib colormap name. None will use the default colormap for the field being plotted as specified by the Py-ART configuration.

mask_outside : bool

True to mask data outside of vmin, vmax. False performs no masking.

title : str

Title to label plot with, None to use default title generated from the field and sweep parameters. Parameter is ignored if title_flag is False.

title_flag : bool

True to add a title to the plot, False does not add a title.

axislabels : (str, str)

2-tuple of x-axis, y-axis labels. None for either label will use the default axis label. Parameter is ignored if axislabels_flag is False.

axislabels_flag : bool

True to add label the axes, False does not label the axes.

colorbar_flag : bool

True to add a colorbar with label to the axis. False leaves off the colorbar.

colorbar_label : str

Colorbar label, None will use a default label generated from the field information.

ticks : array

Colorbar custom tick label locations.

ticklabs : array

Colorbar custom tick labels.

colorbar_orient : 'vertical' or 'horizontal'

Colorbar orientation.

edges : bool

True will interpolate and extrapolate the gate edges from the range, azimuth and elevations in the radar, treating these as specifying the center of each gate. False treats these coordinates themselved as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not not plotted.

filter_transitions : bool

True to remove rays where the antenna was in transition between sweeps from the plot. False will include these rays in the plot. No rays are filtered when the antenna_transition attribute of the underlying radar is not present.

time_axis_flag : bool

True to plot the x-axis as time. False uses the index number. Default is False - index-based.

date_time_form : str, optional

Format of the time string for x-axis labels. Parameter is ignored if time_axis_flag is set to False.

tz : str, optional

Time zone info to use when creating axis labels (see datetime). Parameter is ignored if time_axis_flag is set to False.

ax : Axis

Axis to plot on. None will use the current axis.

fig : Figure

Figure to add the colorbar to. None will use the current figure.

set_aspect_ratio(*aspect_ratio=0.75*, *ax=None*)
Set the aspect ratio for plot area.

set_limits(*xlim=None*, *ylim=None*, *ax=None*)
Set the display limits.

Parameters **xlim** : tuple, optional

2-Tuple containing y-axis limits in km. None uses default limits.

ylim : tuple, optional

2-Tuple containing x-axis limits in km. None uses default limits.

ax : Axis

Axis to adjust. None will adjust the current axis.

CHAPTER

SIXTYNINE

PYART.GRAPH._CM

Data for radar related colormaps.

CHAPTER

SEVENTY

PYART.UTIL.CIRCULAR_STATS

Functions for computing statistics on circular (directional) distributions.

`angular_mean`(angles)	Compute the mean of a distribution of angles in radians.
`angular_std`(angles)	Compute the standard deviation of a distribution of angles in radians.
`angular_mean_deg`(angles)	Compute the mean of a distribution of angles in degrees.
`angular_std_deg`(angles)	Compute the standard deviation of a distribution of angles in degrees.
`interval_mean`(dist, interval_min, interval_max)	Compute the mean of a distribution within an interval.
`interval_std`(dist, interval_min, interval_max)	Compute the standard deviation of a distribution within an interval.
`mean_of_two_angles`(angles1, angles2)	Compute the element by element mean of two sets of angles.
`mean_of_two_angles_deg`(angle1, angle2)	Compute the element by element mean of two sets of angles in degrees.

pyart.util.circular_stats.**angular_mean**(*angles*)

Compute the mean of a distribution of angles in radians.

Parameters **angles** : array like

Distribution of angles in radians.

Returns **mean** : float

The mean angle of the distribution in radians.

pyart.util.circular_stats.**angular_mean_deg**(*angles*)

Compute the mean of a distribution of angles in degrees.

Parameters **angles** : array like

Distribution of angles in degrees.

Returns **mean** : float

The mean angle of the distribution in degrees.

pyart.util.circular_stats.**angular_std**(*angles*)

Compute the standard deviation of a distribution of angles in radians.

Parameters **angles** : array like

Distribution of angles in radians.

Returns **std** : float

Standard deviation of the distribution.

`pyart.util.circular_stats.`**`angular_std_deg`**(*angles*)

Compute the standard deviation of a distribution of angles in degrees.

Parameters **angles** : array like

Distribution of angles in degrees.

Returns **std** : float

Standard deviation of the distribution.

`pyart.util.circular_stats.`**`interval_mean`**(*dist*, *interval_min*, *interval_max*)

Compute the mean of a distribution within an interval.

Return the average of the array elements which are interpreted as being taken from a circular interval with endpoints given by interval_min and interval_max.

Parameters **dist** : array like

Distribution of values within an interval.

interval_min, interval_max : float

The endpoints of the interval.

Returns **mean** : float

The mean value of the distribution

`pyart.util.circular_stats.`**`interval_std`**(*dist*, *interval_min*, *interval_max*)

Compute the standard deviation of a distribution within an interval.

Return the standard deviation of the array elements which are interpreted as being taken from a circular interval with endpoints given by interval_min and interval_max.

Parameters **dist** : array_like

Distribution of values within an interval.

interval_min, interval_max : float

The endpoints of the interval.

Returns **std** : float

The standard deviation of the distribution.

`pyart.util.circular_stats.`**`mean_of_two_angles`**(*angles1*, *angles2*)

Compute the element by element mean of two sets of angles.

Parameters **angles1** : array

First set of angles in radians.

angles2 : array

Second set of angles in radians.

Returns **mean** : array

Elements by element angular mean of the two sets of angles in radians.

`pyart.util.circular_stats.`**`mean_of_two_angles_deg`**(*angle1*, *angle2*)

Compute the element by element mean of two sets of angles in degrees.

Parameters **angle1** : array

First set of angles in degrees.

angle2 : array

Second set of angles in degrees.

Returns **mean** : array

Elements by element angular mean of the two sets of angles in degrees.

CHAPTER SEVENTYONE

PYART.UTIL.HILDEBRAND_SEKHON

Estimation of noise in Doppler spectra using the Hildebrand Sekhon method.

estimate_noise_hs74(spectrum[, navg])	Estimate noise parameters of a Doppler spectrum.

`pyart.util.hildebrand_sekhon.`**`estimate_noise_hs74`**(*spectrum*, *navg=1*)

Estimate noise parameters of a Doppler spectrum.

Use the method of estimating the noise level in Doppler spectra outlined by Hildebrand and Sehkon, 1974.

Parameters **spectrum** : array like

Doppler spectrum in linear units.

navg : int, optional

The number of spectral bins over which a moving average has been taken. Corresponds to the **p** variable from equation 9 of the article. The default value of 1 is appropiate when no moving average has been applied to the spectrum.

Returns **mean** : float-like

Mean of points in the spectrum identified as noise.

threshold : float-like

Threshold separating noise from signal. The point in the spectrum with this value or below should be considered as noise, above this value signal. It is possible that all points in the spectrum are identified as noise. If a peak is required for moment calculation then the point with this value should be considered as signal.

var : float-like

Variance of the points in the spectrum identified as noise.

nnoise : int

Number of noise points in the spectrum.

References

P. H. Hildebrand and R. S. Sekhon, Objective Determination of the Noise Level in Doppler Spectra. Journal of Applied Meteorology, 1974, 13, 808-811.

CHAPTER

SEVENTYTWO

PYART.UTIL.RADAR_UTILS

Functions for working radar instances.

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.
join_radar(radar1, radar2)	Combine two radar instances into one.

`pyart.util.radar_utils.`**`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.util.radar_utils.`**`join_radar`**(*radar1*, *radar2*)

Combine two radar instances into one.

Parameters **radar1** : Radar

Radar object.

radar2 : Radar

Radar object.

`pyart.util.radar_utils.`**`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.

Mathematical, signal processing and numerical routines

CHAPTER

SEVENTYTHREE

TODO

Put more stuff in here

pyart.util.sigmath.**rolling_window**(*a*, *window*)
create a rolling window object for application of functions eg: result=np.ma.std(array, 11), 1)

pyart.util.sigmath.**texture**(*pyradarobj*, *field*)

pyart.util.sigmath.**texture_along_ray**(*myradar*, *var*, *wind_size=7*)
Compute field texture along ray using a user specified window size.

Parameters **myradar** : radar object

The radar object where the field is

var : str

Name of the field which texture has to be computed

wind_size : int

Optional. Size of the rolling window used

Returns **tex** : radar field

the texture of the specified field

CHAPTER SEVENTYFOUR

PYART.UTIL.SIMULATED_VEL

Function for creating simulated velocity fields.

`simulated_vel_from_profile`(radar, profile[, ...])	Create simulated radial velocities from a profile of horizontal winds.

`pyart.util.simulated_vel.`**`simulated_vel_from_profile`**(*radar*, *profile*, *interp_kind='linear'*, *sim_vel_field=None*)

Create simulated radial velocities from a profile of horizontal winds.

Parameters **radar** : Radar

Radar instance which provides the scanning parameters for the simulated radial velocities.

profile : HorizontalWindProfile

Profile of horizontal winds.

interp_kind : str, optional

Specifies the kind of interpolation used to determine the winds at a given height. Must be one of 'linear', 'nearest', 'zero', 'slinear', 'quadratic', or 'cubic'. The the documentation for the SciPy scipy.interpolate.interp1d function for descriptions.

sim_vel_field : str, optional

Name to use for the simulated velocity field metadata. None will use the default field name from the Py-ART configuration file.

Returns **sim_vel** : dict

Dictionary containing a radar field of simulated radial velocities.

CHAPTER SEVENTYFIVE

PYART.UTIL.XSECT

Function for extracting cross sections from radar volumes.

`cross_section_ppi`(radar, target_azimuths[, ...])	Extract cross sections from a PPI volume along one or more azimuth angles.
`cross_section_rhi`(radar, target_elevations)	Extract cross sections from an RHI volume along one or more elevation angles.
`colocated_gates`(radar1, radar2[, h_tol, ...])	Flags radar gates of radar1 colocated with radar2
`intersection`(radar1, radar2[, h_tol, ...])	Flags region of radar1 that is intersecting with radar2 and complies with
`find_intersection_volume`(radar1, radar2[, ...])	Flags region of radar1 that is intersecting with radar2
`find_intersection_limits`(lat1, lon1, alt1, ...)	Find the limits of the intersection between two volumes
`find_equal_vol_region`(radar1, radar2[, ...])	Flags regions of radar1 that are equivolumetric
`get_ground_distance`(lat_array, lon_array, ...)	Computes the ground distance to a fixed point
`get_range`(rng_ground, alt_array, alt0)	Computes the range to a fixed point from the ground distance and the
`get_vol_diameter`(beamwidth, rng)	Computes the pulse volume diameter from the antenna beamwidth and the
`_construct_xsect_radar`(radar, scan_type, ...)	Constructs a new radar object that contains cross-sections at fixed angles of a PPI or RHI volume scan.
`_copy_dic`(orig_dic[, excluded_keys])	Return a copy of the original dictionary copying each element.

pyart.util.xsect.**_construct_xsect_radar**(*radar*, *scan_type*, *pxsect_rays*, *xsect_nsweeps*, *target_angles*)

Constructs a new radar object that contains cross-sections at fixed angles of a PPI or RHI volume scan.

Parameters **radar** : Radar

Radar volume containing RHI/PPI sweeps from which a cross sections will be extracted.

scan_type : str

Type of cross section scan (ppi or rhi)

pxsect_rays : list

list of rays from the radar volume to be copied in the cross-sections radar object

xsect_nsweeps : int

Number of sweeps in the cross-section radar

traget_angles : array

the target fixed angles

Returns **radar_xsect** : Radar

Radar volume containing sweeps which contain cross sections from the original volume.

`pyart.util.xsect._copy_dic(orig_dic, excluded_keys=None)`

Return a copy of the original dictionary copying each element.

`pyart.util.xsect.colocated_gates(radar1, radar2, h_tol=0.0, latlon_tol=0.0, coloc_gates_field=None)`

Flags radar gates of radar1 colocated with radar2

Parameters **radar1** : Radar

radar object that is going to be flagged

radar2 : Radar

radar object

h_tol : float

tolerance in altitude [m]

latlon_tol : float

tolerance in latitude/longitude [deg]

Returns **coloc_dict** : dict

a dictionary containing the colocated positions of radar 1 (ele, azi, rng) and radar 2

coloc_rad1 :

field with the colocated gates of radar1 flagged

`pyart.util.xsect.cross_section_ppi(radar, target_azimuths, az_tol=None)`

Extract cross sections from a PPI volume along one or more azimuth angles.

Parameters **radar** : Radar

Radar volume containing PPI sweeps from which azimuthal cross sections will be extracted.

target_azimuth : list

Azimuthal angles in degrees where cross sections will be taken.

az_tol : float

Azimuth angle tolerance in degrees. If none the nearest angle is used. If valid only angles within the tolerance distance are considered.

Returns **radar_rhi** : Radar

Radar volume containing RHI sweeps which contain azimuthal cross sections from the original PPI volume.

`pyart.util.xsect.cross_section_rhi(radar, target_elevations, el_tol=None)`

Extract cross sections from an RHI volume along one or more elevation angles.

Parameters **radar** : Radar

Radar volume containing RHI sweeps from which azimuthal cross sections will be extracted.

target_elevations : list

Elevation angles in degrees where cross sections will be taken.

el_tol : float

Elevation angle tolerance in degrees. If none the nearest angle is used. If valid only angles within the tolerance distance are considered.

Returns radar_ppi : Radar

Radar volume containing PPI sweeps which contain azimuthal cross sections from the original RHI volume.

pyart.util.xsect.**find_equal_vol_region**(*radar1*, *radar2*, *vol_d_tol=0*)

Flags regions of radar1 that are equivolumetric (similar pulse volume diameter) with radar2

Parameters radar1 : Radar

radar object that is going to be flagged

radar2 : Radar

radar object

vol_d_tol : float

pulse volume diameter tolerance

Returns equal_vol : 2D boolean array

field with true where both radars have a similar pulse volume diameter

pyart.util.xsect.**find_intersection_limits**(*lat1*, *lon1*, *alt1*, *lat2*, *lon2*, *alt2*, *h_tol=0.0*, *latlon_tol=0.0*)

Find the limits of the intersection between two volumes

Parameters lat1, lon1, alt1 : float array

array with the positions of first volume. lat, lon in decimal degrees, alt in m MSL.

lat2, lon2, alt2 : float array

array with the positions of second volume. lat, lon in decimal degrees, alt in m MSL.

h_tol: float

altitude tolerance [m MSL]

latlon_tol: float

latitude and longitude tolerance [decimal deg]

Returns min_lat, max_lat, min_lon, max_lon, min_alt, max_alt : floats

the limits of the intersecting region

pyart.util.xsect.**find_intersection_volume**(*radar1*, *radar2*, *h_tol=0.0*, *latlon_tol=0.0*)

Flags region of radar1 that is intersecting with radar2

Parameters radar1 : Radar

radar object that is going to be flagged

radar2 : Radar

radar object checked for intersecting region

h_tol : float

tolerance in altitude [m]

latlon_tol : float

latitude and longitude tolerance [decimal deg]

Returns **intersec** : 2d array

the field with gates within the common volume flagged

`pyart.util.xsect.get_ground_distance(lat_array, lon_array, lat0, lon0)`

Computes the ground distance to a fixed point

Parameters **lat_array** : float array

array of latitudes [decimal deg]

lon_array : float array

array of longitudes [decimal deg]

lat0: float

latitude of fix point

lon0: float

longitude of fix point

Returns **rng_ground** : float array

the ground range [m]

`pyart.util.xsect.get_range(rng_ground, alt_array, alt0)`

Computes the range to a fixed point from the ground distance and the altitudes

Parameters **rng_ground** : float array

array of ground distances [m]

alt_array : float array

array of altitudes [m MSL]

alt0: float

altitude of fixed point [m MSL]

Returns **rng** : float array

the range [m]

`pyart.util.xsect.get_vol_diameter(beamwidth, rng)`

Computes the pulse volume diameter from the antenna beamwidth and the range from the radar

Parameters **beamwidth** : float

the radar beamwidth [deg]

rng : float array

the range from the radar [m]

Returns **vol_d** : float array

the pulse volume diameter

`pyart.util.xsect.intersection(radar1, radar2, h_tol=0.0, latlon_tol=0.0, vol_d_tol=0.0, vismin=None, hmin=None, hmax=None, rmin=None, rmax=None, elmin=None, elmax=None, azmin=None, azmax=None, visib_field=None, intersec_field=None)`

Flags region of radar1 that is intersecting with radar2 and complies with criteria regarding visibility, altitude, range, elevation angle and azimuth angle

Parameters **radar1** : Radar

radar object that is going to be flagged

radar2 : Radar

radar object checked for intersecting region

h_tol : float

tolerance in altitude [m]

latlon_tol : float

latitude and longitude tolerance [decimal deg]

vol_d_tol : float

pulse volume diameter tolerance [m]

vismin : float

minimum visibility [percentage]

hmin, hmax : floats

min and max altitude [m MSL]

rmin, rmax : floats

min and max range from radar [m]

elmin, elmax : floats

min and max elevation angle [deg]

azmin, azmax : floats

min and max azimuth angle [deg]

Returns **intersec_rad1_dict** : dict

the field with the gates of radar1 in the same region as radar2 flagged

CHAPTER

SEVENTYSIX

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='') -> str
MDV_RHI_FILE	str(object='') -> str
CFRADIAL_PPI_FILE	str(object='') -> str
CFRADIAL_RHI_FILE	str(object='') -> str
CHL_RHI_FILE	str(object='') -> str
SIGMET_PPI_FILE	str(object='') -> str
SIGMET_RHI_FILE	str(object='') -> str
NEXRAD_ARCHIVE_MSG31_FILE	str(object='') -> str
NEXRAD_ARCHIVE_MSG31_COMPRESSED_FILE	str(object='') -> str
NEXRAD_ARCHIVE_MSG1_FILE	str(object='') -> str
NEXRAD_LEVEL3_MSG19	str(object='') -> str
NEXRAD_LEVEL3_MSG163	str(object='') -> str
NEXRAD_CDM_FILE	str(object='') -> str
UF_FILE	str(object='') -> str
INTERP_SOUNDE_FILE	str(object='') -> str

CHAPTER

SEVENTYSEVEN

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`([alias])	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.
`make_normal_storm`(sigma, mu)	Make a sample Grid with a gaussian storm target.

`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 (z, y, x).

grid_limits : 3-tuple of 2-tuples

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

Returns grid : Grid

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

`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_empty_rhi_radar`**(*ngates*, *rays_per_sweep*, *nsweeps*)

Return an Radar object, representing a RHI 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_normal_storm`**`(`*sigma*, *mu*`)`
Make a sample Grid with a gaussian storm target.

`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_storm_grid`**`()`
Make a sample Grid with a rectangular storm target.

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

`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`**`(`*alias=True*`)`
Return a PPI radar with a target like reflectivity field.

Set alias to False to return a de-aliased radar.

`pyart.testing.sample_objects.`**`make_velocity_aliased_rhi_radar`**`(`*alias=True*`)`
Return a RHI radar with a target like reflectivity field.

Set alias to False to return a de-aliased radar.

CHAPTER

SEVENTYEIGHT

PYART.TESTING.TMPDIRS

Classes for creating and cleaning temporary directories in unit tests.

TemporaryDirectory([suffix, prefix, dir])	Create and return a temporary directory.
InTemporaryDirectory([suffix, prefix, dir])	Create, return, and change directory to a temporary directory
InGivenDirectory([path])	Change directory to given directory for duration of `with` block

This module is taken from the nibable project. The following license applies:

```
The MIT License

Copyright (c) 2009-2014 Matthew Brett <matthew.brett@gmail.com>
Copyright (c) 2010-2013 Stephan Gerhard <git@unidesign.ch>
Copyright (c) 2006-2014 Michael Hanke <michael.hanke@gmail.com>
Copyright (c) 2011 Christian Haselgrove <christian.haselgrove@umassmed.edu>
Copyright (c) 2010-2011 Jarrod Millman <jarrod.millman@gmail.com>
Copyright (c) 2011-2014 Yaroslav Halchenko <debian@onerussian.com>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

class `pyart.testing.tmpdirs.`**`InGivenDirectory`**(*path=None*)

Bases: `object`

Change directory to given directory for duration of `with` block

Useful when you want to use *InTemporaryDirectory* for the final test, but you are still debugging. For example, you may want to do this in the end:

```
>>> with InTemporaryDirectory() as tmpdir:
...     # do something complicated which might break
...     pass
```

But indeed the complicated thing does break, and meanwhile the `InTemporaryDirectory` context manager wiped out the directory with the temporary files that you wanted for debugging. So, while debugging, you replace with something like:

```
>>> with InGivenDirectory() as tmpdir: # Use working directory by default
...     # do something complicated which might break
...     pass
```

You can then look at the temporary file outputs to debug what is happening, fix, and finally replace `InGivenDirectory` with `InTemporaryDirectory` again.

`__class__`
alias of `type`

`__delattr__`
Implement delattr(self, name).

`__dict__` = mappingproxy({'__dict__': <attribute '__dict__' of 'InGivenDirectory' objects>, '__doc__': ' Change direct

`__dir__``()` → list
default dir() implementation

`__enter__``()`

`__eq__`
Return self==value.

`__exit__``(`*exc*, *value*, *tb*`)`

`__format__``()`
default object formatter

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__``(`*path=None*`)`
Initialize directory context manager

Parameters **path** : None or str, optional

path to change directory to, for duration of `with` block. Defaults to `os.getcwd()` if None

`__le__`
Return self<=value.

`__lt__`
Return self<value.

`__module__` = 'pyart.testing.tmpdirs'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

class pyart.testing.tmpdirs.**InTemporaryDirectory**(*suffix=''*, *prefix='tmp'*, *dir=None*)
Bases: *pyart.testing.tmpdirs.TemporaryDirectory*

Create, return, and change directory to a temporary directory

Examples

```
>>> import os
>>> my_cwd = os.getcwd()
>>> with InTemporaryDirectory() as tmpdir:
...     _ = open('test.txt', 'wt').write('some text')
...     assert os.path.isfile('test.txt')
...     assert os.path.isfile(os.path.join(tmpdir, 'test.txt'))
>>> os.path.exists(tmpdir)
False
>>> os.getcwd() == my_cwd
True
```

Methods

cleanup()	

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyart.testing.tmpdirs', '__doc__': " Create, return, and change directory to

__dir__() → list
default dir() implementation

__enter__()

__eq__
Return self==value.

__exit__(*exc*, *value*, *tb*)

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*suffix=''*, *prefix='tmp'*, *dir=None*)

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.testing.tmpdirs'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

cleanup()

class pyart.testing.tmpdirs.**TemporaryDirectory**(*suffix='‘, prefix='tmp', dir=None*)
Bases: object

Create and return a temporary directory. This has the same behavior as mkdtemp but can be used as a context manager.

Upon exiting the context, the directory and everthing contained in it are removed.

Examples

```
>>> import os
>>> with TemporaryDirectory() as tmpdir:
...     fname = os.path.join(tmpdir, 'example_file.txt')
...     with open(fname, 'wt') as fobj:
...         _ = fobj.write('a string\n')
>>> os.path.exists(tmpdir)
False
```

Methods

cleanup()	

__class__
alias of type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__dict__': <attribute '__dict__' of 'TemporaryDirectory' objects>, '__doc__': "Create and

__dir__() → list
default dir() implementation

__enter__()

__eq__
Return self==value.

__exit__(*exc, value, tb*)

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*suffix=''*, *prefix='tmp'*, *dir=None*)

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyart.testing.tmpdirs'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

cleanup()

CHAPTER

SEVENTYNINE

INDICES AND TABLES

- genindex
- modindex
- search

[R5] http://www.ncdc.noaa.gov/

[R6] http://www.roc.noaa.gov/wsr88d/Level_III/Level3Info.asp

[R9] http://www.ncdc.noaa.gov/

[R10] http://thredds.ucar.edu/thredds/catalog.html

[R13] http://www.unidata.ucar.edu/software/netcdf-java/documentation.htm

[R14] http://thredds.ucar.edu/thredds/catalog.html

[R17] http://www.roc.noaa.gov/WSR88D/Level_II/Level2Info.aspx

[R18] http://www.ncdc.noaa.gov/

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

[R1] Miguel Arevallilo Herraez, David R. Burton, Michael J. Lalor, and Munther A. Gdeisat, "Fast two-dimensional phase-unwrapping algorithm based on sorting by reliability following a noncontinuous path", Journal Applied Optics, Vol. 41, No. 35 (2002) 7437,

[R2] Abdul-Rahman, H., Gdeisat, M., Burton, D., & Lalor, M., "Fast three-dimensional phase-unwrapping algorithm based on sorting by reliability following a non-continuous path. In W. Osten, C. Gorecki, & E. L. Novak (Eds.), Optical Metrology (2005) 32–40, International Society for Optics and Photonics.

p

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

P

R

S

T

U

V

W

Y