

---

# **pyart-mch library reference for users**

***Release 0.1.0***

**meteoswiss-mdr**

**Feb 19, 2020**



# CONTENTS

<b>1</b>	<b>Input and output (<code>pyart.io</code>)</b>	<b>3</b>
1.1	Reading radar data . . . . .	3
1.2	Writing radar data . . . . .	3
1.3	Reading grid data . . . . .	3
1.4	Writing grid data . . . . .	4
1.5	Reading Sonde data . . . . .	4
1.6	Special use . . . . .	4
<b>2</b>	<b>Auxiliary input and output (<code>pyart.aux_io</code>)</b>	<b>19</b>
2.1	Reading radar data . . . . .	19
2.2	Writing radar data . . . . .	19
<b>3</b>	<b>Core (<code>pyart.core</code>)</b>	<b>35</b>
3.1	Core classes . . . . .	35
3.2	Coordinate transformations . . . . .	35
<b>4</b>	<b>Bridging to other toolkits (<code>pyart.bridge</code>)</b>	<b>63</b>
4.1	Phase functions . . . . .	63
<b>5</b>	<b>Filters (<code>pyart.filters</code>)</b>	<b>65</b>
5.1	Filtering radar data . . . . .	65
<b>6</b>	<b>Radar Corrections (<code>pyart.correct</code>)</b>	<b>77</b>
6.1	Velocity unfolding . . . . .	77
6.2	Other corrections . . . . .	77
6.3	Helper functions . . . . .	78
<b>7</b>	<b>Radar Retrievals (<code>pyart.retrieve</code>)</b>	<b>111</b>
7.1	Radar retrievals . . . . .	111
<b>8</b>	<b>Mapping (<code>pyart.map</code>)</b>	<b>147</b>
<b>9</b>	<b>Graphing (<code>pyart.graph</code>)</b>	<b>153</b>
9.1	Plotting radar data . . . . .	153
9.2	Plotting grid data . . . . .	153
<b>10</b>	<b>Utilities (<code>pyart.util</code>)</b>	<b>229</b>
10.1	Direction statistics . . . . .	229
10.2	Miscellaneous functions . . . . .	229
<b>11</b>	<b>Testing Utilities (<code>pyart.testing</code>)</b>	<b>239</b>

11.1	Testing functions . . . . .	239
11.2	Testing classes . . . . .	239
<b>12</b>	<b>Indices and tables</b>	<b>243</b>
	<b>Bibliography</b>	<b>245</b>
	<b>Python Module Index</b>	<b>247</b>
	<b>Index</b>	<b>249</b>

Contents:

---



## INPUT AND OUTPUT (PYART.IO)

Functions to read and write radar and grid data to and from a number of file formats.

### 1.1 Reading radar data

In most cases the `pyart.io.read()` function should be used to read in radar data from a file. In certain cases the function the read function for the format in question should be used.

<code>read(filename[, use_rsl])</code>	Read a radar file and return a radar object.
<code>read_rsl(filename[, field_names, ...])</code>	Read a file supported by RSL.
<code>read_mdv(filename[, field_names, ...])</code>	Read a MDV file.
<code>read_sigmet(filename[, field_names, ...])</code>	Read a Sigmet (IRIS) product file.
<code>read_cfradial(filename[, field_names, ...])</code>	Read a Cfradial netCDF file.
<code>read_cfradial2(filename[, field_names, ...])</code>	Read a Cfradial2 netCDF file.
<code>read_chl(filename[, field_names, ...])</code>	Read a CSU-CHILL CHL file.
<code>read_nexrad_archive(filename[, field_names, ...])</code>	Read a NEXRAD Level 2 Archive file.
<code>read_nexrad_cdm(filename[, field_names, ...])</code>	Read a Common Data Model (CDM) NEXRAD Level 2 file.
<code>read_nexrad_level3(filename[, field_names, ...])</code>	Read a NEXRAD Level 3 product.
<code>read_uf(filename[, field_names, ...])</code>	Read a UF File.

### 1.2 Writing radar data

<code>write_cfradial(filename, radar[, format, ...])</code>	Write a Radar object to a CF/Radial compliant netCDF file.
<code>write_uf(filename, radar[, uf_field_names, ...])</code>	Write a Radar object to a UF file.

### 1.3 Reading grid data

<code>read_grid(filename[, exclude_fields, ...])</code>	Read a netCDF grid file produced by Py-ART.
<code>read_grid_mdv(filename[, field_names, ...])</code>	Read a MDV file to a Grid Object.

## 1.4 Writing grid data

<code>write_grid(filename, grid[, format, ...])</code>	Write a Grid object to a CF-1.5 and ARM standard netCDF file.
<code>write_grid_mdv(filename, grid[, ...])</code>	Write grid object to MDV file.
<code>write_grid_geotiff(grid, filename, field[, ...])</code>	Write a Py-ART Grid object to a GeoTIFF file.

## 1.5 Reading Sonde data

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

## 1.6 Special use

<code>prepare_for_read(filename)</code>	Return a file like object read for reading.
<code>make_time_unit_str(dtobj)</code>	Return a time unit string from a datetime object.

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

`pyart.io.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.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 introduce any additional 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.

`pyart.io.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.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.

`pyart.io.read_cfradial(filename, field_names=None, additional_metadata=None, file_field_names=False, exclude_fields=None, include_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 the *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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

**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.read_cfradial2(filename, field_names=None, additional_metadata=None,
                        file_field_names=False, exclude_fields=None, include_fields=None,
                        delay_field_loading=False, **kwargs)
```

Read a Cfradial2 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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

**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.read_chl(filename, field_names=None, additional_metadata=None, file_field_names=None,
                  exclude_fields=None, include_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 introduce any additional 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.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not in *exclude\_fields*.

**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.

```
pyart.io.read_grid(filename, exclude_fields=None, include_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 or None, optional] List of fields to exclude from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

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

Read a MDV file to a Grid Object.

#### 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 introduce any additional 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 grid 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.

#### Returns

**grid** [Grid] Grid object containing data from MDV file.

#### Notes

This function can only read cartesian MDV files with fields compressed with gzip or zlib. For polar files see [`pyart.io.read\_mdv\(\)`](#)

MDV files and Grid object are not fully interchangeable. Specific limitation include:

- All fields must have the same shape and dimensions.
- All fields must have the same projection.
- Vlevels types must not vary.
- Projection must not be PROJ\_POLAR\_RADAR (9) or PROJ\_RHI\_RADAR (13).
- Correct unit in the Z axis are just available for 'vlevel\_type' equal to VERT\_TYPE\_Z(4), VERT\_TYPE\_ELEV(9), VERT\_TYPE\_AZ(17), VERT\_TYPE\_PRESSURE(3) and VERT\_TYPE\_THETA(7).
- The behavior in cases of 2D data is unknown but most likely will not fail.

```
pyart.io.read_mdv(filename, field_names=None, additional_metadata=None, file_field_names=False,
                  exclude_fields=None, include_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 introduce any additional 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

**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.

```
pyart.io.read_nexrad_archive(filename, field_names=None, additional_metadata=None,
                             file_field_names=False, exclude_fields=None, include_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 [1] as well as on the UCAR THREDDS Data Server [2] 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 introduce any additional 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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

**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

[1], [2]

```
pyart.io.read_nexrad_cdm(filename, field_names=None, additional_metadata=None,
                        file_field_names=False, exclude_fields=None, include_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 [1]. A URL of a OPeNDAP file on the UCAR THREDDS Data Server [2] 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 introduce any additional 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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

**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

[1], [2]

```
pyart.io.read_nexrad_level3(filename, field_names=None, additional_metadata=None,
                           file_field_names=False, exclude_fields=None, include_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 [1] 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 introduce any additional 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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

#### Returns

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

#### References

[1], [2]

```
pyart.io.read_rsl(filename, field_names=None, additional_metadata=None, file_field_names=False,
                  exclude_fields=None, delay_field_loading=False, include_fields=None,
                  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 introduce any additional 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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

**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.



```
pyart.io.read_sigmet(filename, field_names=None, additional_metadata=None,
                    file_field_names=False, exclude_fields=None, include_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 introduce any additional 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

**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 appear in the Sigmet file. 'sequential' will determine 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 issued 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, this option 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 exist 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 use 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 explicitly 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.read_uf(filename, field_names=None, additional_metadata=None, file_field_names=False, exclude_fields=None, include_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 re-name 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

**delay\_field\_loading** [bool] This option is not implemented in the function but included for compatibility.

#### Returns

**radar** [Radar] Radar object.

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

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`
- `_Contiguous`
- `_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.

**physical** [bool] True to store the radar fields as physical numbers, False will store the radar fields as binary if the keyword '\_Write\_as\_dtype' is in the field metadata. The gain and offset can be specified in the keyword 'scale\_factor' and 'add\_offset' or calculated on the fly.

```
pyart.io.write_grid(filename, grid, format='NETCDF4', write_proj_coord_sys=True,
                    proj_coord_sys=None, arm_time_variables=False,
                    arm_alt_lat_lon_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`
- `_Contiguous`
- `_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.

**arm\_alt\_lat\_lon\_variables** [bool, optional] True to write the ARM standard alt, lat, lon variables. 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.

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

**use\_doublequotes** [bool, optional]

**True** - Use double quotes in the **gdalwarp** call (requires **warp=True**), which may help if that command is producing an error like: 'Translating source or target SRS failed'.

**False** - Use single quotes instead.

`pyart.io.write_grid_mdv(filename, grid, mdv_field_names=None, field_write_order=None)`

Write grid object to MDV file.

Create a MDV file containing data from the provided grid instance.

The MDV file will contain parameters from the 'source' key if contained in `grid.metadata`. If this key or parameters related to the radar location and name are not present in the grid a default or sentinel value. will be written in the MDV file in the place of the parameter.

Grid fields will be saved in float32 unless the `_Write_as_dtype` key is present.

#### Parameters

**filename** [str or file-like object.] Filename of MDV file to create. If a file-like object is specified data will be written using the write method.

**grid** [Grid] Grid object from which to create MDV file.

**mdv\_field\_names** [dict or None, optional] Mapping between grid fields and MDV data type names. Field names mapped to None or with no mapping will be excluded from writing. If None, the same field names will be used.

**field\_write\_order** [list or None, optional] Order in which grid fields should be written out in the MDV file. None, the default, will determine a valid order automatically.

#### Notes

Do to limitations of the MDV format, not all grid objects are writable. To write a grid the following conditions must be satisfied:

- XY grid must be regular (equal spacing), Z can be irregular.
- The number of Z levels must not exceed 122.
- Fields can be encoded in the file using the `'_Write_as_dtype'` key specifying one of `'uint8'`, `'uint16'` or `'float32'`. Use the `'scale_factor'` and `'add_offset'` keys to specify scaling. Field data in the Grid object should be uncompressed, that is to say it has had the scaling applied.

```
pyart.io.write_uf(filename, radar, uf_field_names=None, radar_field_names=False, ex-
                  clude_fields=None, field_write_order=None, volume_start=None, tem-
                  plates_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.

## AUXILIARY INPUT AND OUTPUT (PYART.AUX\_IO)

Additional classes and functions for reading and writing data from a number of file formats.

These auxiliary input/output routines are not as well polished as those in `pyart.io`. They may require additional dependencies beyond those required for a standard Py-ART install, use non-standard function parameter and naming, are not supported by the `pyart.io.read()` function and are not fully tested if tested at all. Please use these at your own risk.

Bugs in these functions should be reported but fixing them may not be a priority.

### 2.1 Reading radar data

<code>read_d3r_gcpep_nc(filename[, field_names, ...])</code>	Read a D3R GCPEX netCDF file.
<code>read_gamic(filename[, field_names, ...])</code>	Read a GAMIC hdf5 file.
<code>read_kazr(filename[, field_names, ...])</code>	Read K-band ARM Zenith Radar (KAZR) NetCDF ingest data.
<code>read_noxp_iphex_nc(filename[, field_names, ...])</code>	Read a NOXP IPHEX netCDF file.
<code>read_odim_h5(filename[, field_names, ...])</code>	Read a ODIM_H5 file.
<code>read_pattern(filename, **kwargs)</code>	Read a netCDF file from a PATTERN project X-band radar.
<code>read_radx(filename[, radx_dir])</code>	Read a file by first converting it to Cf/Radial using RadxConvert.
<code>read_rainbow_wrl(filename[, field_names, ...])</code>	Read a RAINBOW file.
<code>read_metrinet(filename[, field_names, rmax, ...])</code>	Read a METRANET file.
<code>read_cartesian_metrinet(filename[, ...])</code>	Read a METRANET product file.
<code>read_gif(filename[, additional_metadata, ...])</code>	Read a MeteoSwiss operational radar data gif file.
<code>read_bin(filename[, additional_metadata, ...])</code>	Read a MeteoSwiss operational radar data binary file.
<code>read_iq(filename, filenames_iq[, ...])</code>	Read a rad4alp IQ file.
<code>read_rainbow_psr(filename, filenames_psr[, ...])</code>	Read a PSR file.
<code>read_rainbow_psr_spectra(filename, filenames_psr)</code>	Read a PSR file to get the complex spectra
<code>read_spectra(filename[, field_names, ...])</code>	Read a spectra netCDF file.
<code>read_cfl(filename[, field_names, ...])</code>	Read a CF-1 netCDF file.

### 2.2 Writing radar data

<code>write_odim_h5(filename, radar[, ...])</code>	Write a Radar object to a EUMETNET OPERA compliant HDF5 file.
<code>write_spectra(filename, radar[, format, ...])</code>	Write a Radar Spectra object to a netCDF file.

`pyart.aux_io.convert_data(values)`

Converts an string of values into the corresponding format

#### Parameters

**values:** `str` string containing the values to convert

#### Returns

**values** [`int`, `float`, `str` or 1D array of `int`, `float` or `str`] The converted values

`pyart.aux_io.get_library(verbose=False, momentms=True)`

return the link to C-shared library

#### Parameters

**verbose** [`Boolean`] If true print out extra information

**momentms** [`Boolean`] If true returns the link to the MS library

#### Returns

**metranet\_lib** [`link`] loaded METRANET C-library

`pyart.aux_io.read_bin(filename, additional_metadata=None, chy0=255.0, chx0=-160.0, xres=1.0, yres=1.0, nx=710, ny=640, nz=1, **kwargs)`

Read a MeteoSwiss operational radar data binary file.

#### Parameters

**filename** [`str`] Name of the file to read.

**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 introduce any additional metadata and the file specific or default metadata as specified by the Py-ART configuration file will be used.

**chy0, chx0** [`float`] Swiss coordinates position of the south-western point in the grid

**xres, yres** [`float`] resolution of each grid point [`km`]

**nx, ny, nz** [`int`] dimensions of the grid

#### Returns

**grid** [`Grid`] Grid object containing data the data.

`pyart.aux_io.read_cartesian_metrane(filename, additional_metadata=None, chy0=255.0, chx0=-160.0, reader='C', **kwargs)`

Read a METRANET product file.

#### Parameters

**filename** [`str`] Name of the METRANET file to read.

**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 introduce any additional metadata and the file specific or default metadata as specified by the Py-ART configuration file will be used.

**chy0, chx0** [`float`] Swiss coordinates position of the south-western point in the grid



**reader** [str] The reader library to use. Can be either 'C' or 'python'

### Returns

**grid** [Grid] Grid object containing data from METRANET file.

```
pyart.aux_io.read_cf1(filename, field_names=None, additional_metadata=None,
                      file_field_names=False, exclude_fields=None, include_fields=None, de-
                      lay_field_loading=False, **kwargs)
```

Read a CF-1 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 re-name 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

**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.aux_io.read_d3r_gcpex_nc(filename, field_names=None, additional_metadata=None,
                               file_field_names=False, exclude_fields=None, in-
                               clude_fields=None, read_altitude_from_nc=False, **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 introduce any additional 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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

**read\_altitude\_from\_nc** [bool, optional] True if you want the altitude value to be read from the provider netCDF file. False will default to the value `np.array([295.], dtype='float64')`

#### Returns

**radar** [Radar] Radar object containing data from ODIM\_H5 file.

`pyart.aux_io.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.

`pyart.aux_io.read_file_c(file, moment='ZH', physic_value=False, masked_array=False, verbose=False)`  
Reads a METRANET data file

#### Parameters

**file** [str] file name

**moment** [str] moment name

**physic\_value** [boolean] If true returns the physical value. Otherwise the digital value

**masked\_array** [boolean] If true returns a numpy masked array with NaN values masked. Otherwise returns a regular masked array with NaN values

**verbose** [boolean] If true prints out extra information

#### Returns

**ret\_data** [RadarData object] An object containing the information read from the file

`pyart.aux_io.read_file_py(file, moment='ZH', physic_value=False, masked_array=False, re_order_angles=True, verbose=False)`  
Reads a METRANET data file

#### Parameters

**file** [str] file name

**moment** [str] moment name

**physic\_value** [boolean] If true returns the physical value. Otherwise the digital value

**masked\_array** [boolean] If true returns a numpy masked array with NaN values masked. Otherwise returns a regular masked array with NaN values

**reorder\_angles** [boolean] If true angles are reordered

**verbose** [boolean] If true prints out extra information

### Returns

**ret\_data** [RadarData object] An object containing the information read from the file

```
pyart.aux_io.read_gamic(filename, field_names=None, additional_metadata=None,
                        file_field_names=False, exclude_fields=None, include_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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

**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.

```
pyart.aux_io.read_gif(filename, additional_metadata=None, chy0=255.0, chx0=-160.0, xres=1.0,
                    yres=1.0, nx=710, ny=640, nz=1, **kwargs)
```

Read a MeteoSwiss operational radar data gif file.

### Parameters

**filename** [str] Name of the file to read.

**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 introduce any additional metadata and the file specific or default metadata as specified by the Py-ART configuration file will be used.

**chy0, chx0** [float] Swiss coordinates position of the south-western point in the grid

**xres, yres** [float] resolution of each grid point [km]

**nx, ny, nz** [int] dimensions of the grid

#### Returns

**grid** [Grid] Grid object containing the data.

```
pyart.aux_io.read_iq(filename, filenames_iq, field_names=None, additional_metadata=None,
                    file_field_names=False, exclude_fields=None, include_fields=None,
                    reader='C', nbytes=4, prf=None, ang_tol=0.4, noise_h=None, noise_v=None,
                    rconst_h=None, rconst_v=None, radconst_h=None, radconst_v=None,
                    mfloss_h=1.0, mfloss_v=1.0, azi_min=None, azi_max=None, ele_min=None,
                    ele_max=None, rng_min=None, rng_max=None, **kwargs)
```

Read a rad4alp IQ file.

#### Parameters

**filename** [str] Name of the METRANET file to be used as reference.

**filenames\_iq** [list of str] Name of the IQ files

**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 introduce any additional 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

**reader** [str] The library used to read the METRANET reference file. Can be either 'C' or 'python'

**nbytes** [int] The number of bytes used to store the data in numpy arrays, e.g. if nbytes=4 then floats are going to be stored as np.float32

**prf** [float] The PRF of the read scan

**ang\_tol** [float] Tolerated angle distance between nominal radar angle and angle in PSR files

**noise\_h, noise\_v** [float] The estimated H(V) noise power (ADU) of the scan

**rconst\_h, rconst\_v** [float] Dynamical factor used in the conversion from dBADU to dBm/dBZ

**radconst\_h, radconst\_v** [float] The H(V) radar constant

**mfloss\_h, mfloss\_v** [float] The H(V) matched filter losses in the receiver (dB)

**azi\_min, azi\_max, ele\_min, ele\_max** [float or None] The minimum and maximum angles to keep (deg)

**rng\_min, rng\_max** [float or None] The minimum and maximum ranges to keep (m)

### Returns

**radar** [Radar] Radar object containing data from PSR file.

`pyart.aux_io.read_kazr(filename, field_names=None, additional_metadata=None, file_field_names=False, exclude_fields=None, include_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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

### Returns

**radar** [Radar] Radar object.

`pyart.aux_io.read_metranet(filename, field_names=None, rmax=0.0, additional_metadata=None, file_field_names=False, exclude_fields=None, reader='C', nbytes=4, **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.

**rmax** [float, optional] Maximum radar range to store in the radar object [m]. If 0 all data will be stored

**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 introduce any additional 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.

**reader** [str] The reader library to use. Can be either 'C' or 'python'

**nbytes** [int] The number of bytes used to store the data in numpy arrays, e.g. if nbytes=4 then floats are going to be stored as np.float32

#### Returns

**radar** [Radar] Radar object containing data from METRANET file.

```
pyart.aux_io.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 introduce any additional 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

#### Returns

**radar** [Radar] Radar object containing data from netCDF file.

```
pyart.aux_io.read_odim_h5(filename, field_names=None, additional_metadata=None,  
                          file_field_names=False, exclude_fields=None, include_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 introduce any additional 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

#### Returns

**radar** [Radar] Radar object containing data from ODIM\_H5 file.

`pyart.aux_io.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.

`pyart.aux_io.read_product_c(radar_file, physic_value=False, masked_array=False, verbose=False)`

Reads a METRANET cartesian data file

#### Parameters

**radar\_file** [str] file name

**physic\_value** [boolean] If true returns the physical value. Otherwise the digital value

**masked\_array** [boolean] If true returns a numpy masked array with NaN values masked. Otherwise returns a regular masked array with NaN values

**verbose** [boolean] If true prints out extra information

#### Returns

**ret\_data** [RadarData object] An object containing the information read from the file. None if the file has not been properly read

`pyart.aux_io.read_product_py(radar_file, physic_value=False, masked_array=False, verbose=False)`

Reads a METRANET cartesian data file

#### Parameters

**radar\_file** [str] file name

**physic\_value** [boolean] If true returns the physical value. Otherwise the digital value

**masked\_array** [boolean] If true returns a numpy masked array with NaN values masked. Otherwise returns a regular masked array with NaN values

**verbose** [boolean] If true prints out extra information

#### Returns

**ret\_data** [RadarData object] An object containing the information read from the file. None if the file has not been properly read

`pyart.aux_io.read_psr_cpi_header(filename)`

Reads the CPI data headers contained in a PSR file

#### Parameters

**filename** [str] Name of the PSR file

#### Returns

**cpi\_header, header** [dict] Dictionary containing the PSR header data and the CPI headers data

`pyart.aux_io.read_psr_header(filename)`

Read a PSR file header.

#### Parameters

**filename** [str] Name of the PSR file

#### Returns

**header** [dict] Dictionary containing the PSR header data

`pyart.aux_io.read_psr_spectra(filename)`

Reads the complex spectral data contained in a PSR file

#### Parameters

**filename** [str] Name of the PSR file

#### Returns

**spectra** [3D complex ndarray] The complex spectra

`pyart.aux_io.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.

`pyart.aux_io.read_rainbow_psr(filename, filenames_psr, field_names=None, additional_metadata=None, file_field_names=False, exclude_fields=None, include_fields=None, undo_txcorr=True, cpi='mean', ang_tol=0.5, azi_min=None, azi_max=None, ele_min=None, ele_max=None, rng_min=None, rng_max=None, **kwargs)`

Read a PSR file.

#### Parameters



**filename** [str] Name of the rainbow file to be used as reference.

**filenames\_psr** [list of str] Name of the PSR files

**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 introduce any additional 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

**undo\_txcrr:** **Bool** If True the correction of the transmitted power is removed from the noise signal

**cpi** [str] The CPI to use. Can be 'low\_prf', 'intermediate\_prf', 'high\_prf', 'mean', 'all'. If 'mean' the mean within the angle step is taken

**ang\_tol** [float] Tolerated angle distance between nominal radar angle and angle in PSR files

**azi\_min, azi\_max, ele\_min, ele\_max** [float or None] The minimum and maximum angles to keep (deg)

**rng\_min, rng\_max** [float or None] The minimum and maximum ranges to keep (m)

## Returns

**radar** [Radar] Radar object containing data from PSR file.

```
pyart.aux_io.read_rainbow_psr_spectra(filename, filenames_psr, field_names=None, additional_metadata=None, file_field_names=False, exclude_fields=None, include_fields=None, undo_txcrr=True, fold=True, positive_away=True, cpi='low_prf', ang_tol=0.5, azi_min=None, azi_max=None, ele_min=None, ele_max=None, rng_min=None, rng_max=None, **kwargs)
```

Read a PSR file to get the complex spectra

## Parameters

**filename** [str] Name of the rainbow file to be used as reference.

**filenames\_psr** [list of str] list of PSR file names

**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 introduce any additional 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. Set to None to include all fields specified by *include\_fields*.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by *exclude\_fields*.

**undo\_txcrr:** **Bool** If True the correction of the transmitted power is removed from the noise signal

**fold:** **Bool** If True the spectra is folded so that 0-Doppler is in the middle

**positive\_away:** **Bool** If True the spectra is reversed so that positive velocities are away from the radar

**cpi** [str] The CPI to use. Can be 'low\_prf', 'intermediate\_prf', 'high\_prf' or 'all'

**ang\_tol** [float] Tolerated angle distance between nominal radar angle and angle in PSR files

**azi\_min, azi\_max, ele\_min, ele\_max** [float or None] The minimum and maximum angles to keep (deg)

**rng\_min, rng\_max** [float or None] The minimum and maximum ranges to keep (m)

## Returns

**radar** [Radar] Radar object containing data from PSR file.

```
pyart.aux_io.read_rainbow_wrl(filename, field_names=None, additional_metadata=None,
                             file_field_names=False, exclude_fields=None, include_fields=None, nbytes=4, **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 constantly 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 and raise an issue upstream.

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, Rho-HVu Co-polar differential phase: PhiDP, uPhiDP, uPhiDPu Specific differential phase: KDP, uKDP, uKDPu Signal quality parameters: SQL, SQLu, SQLv, SQLvu 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 introduce any additional 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

**nbytes** [int] The number of bytes used to store the data in numpy arrays, e.g. if nbytes=4 then floats are going to be stored as np.float32

### Returns

**radar** [Radar] Radar object containing data from RAINBOW file.

```
pyart.aux_io.read_sinaram_h5(filename, field_names=None, additional_metadata=None,
                             file_field_names=False, exclude_fields=None, include_fields=None, **kwargs)
```

Read a SINARAM\_H5 file.

### Parameters

**filename** [str] Name of the SINARAM\_H5 file to read.

**field\_names** [dict, optional] Dictionary mapping SINARAM\_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 introduce any additional 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. Set to None to include all fields specified by include\_fields.

**include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the *file\_field\_names* and *field\_names* parameters. Set to None to include all fields not specified by exclude\_fields.

### Returns

**radar** [Radar] Radar object containing data from SINARAM\_H5 file.

```
pyart.aux_io.read_spectra(filename, field_names=None, additional_metadata=None,
                          file_field_names=False, exclude_fields=None, include_fields=None,
                          delay_field_loading=False, **kwargs)
```

Read a spectra 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. Set to None to include all fields specified by `include_fields`.
- include\_fields** [list or None, optional] List of fields to include from the radar object. This is applied after the `file_field_names` and `field_names` parameters. Set to None to include all fields not specified by `exclude_fields`.
- 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.aux_io.write_odim_h5(filename, radar, field_names=None, physical=True, compression='gzip', compression_opts=6)
```

Write a Radar object to a EUMETNET OPERA compliant HDF5 file.

The files produced by this routine follow the EUMETNET OPERA information model: [http://eumetnet.eu/wp-content/uploads/2017/01/OPERA\\_hdf\\_description\\_2014.pdf](http://eumetnet.eu/wp-content/uploads/2017/01/OPERA_hdf_description_2014.pdf)

### Supported features:

- Writing PPIs: PVOL and SCAN objects - Different sweeps are saved in different dataset groups
- Writing sectorized PPIs and SCANS: AZIM objects
- Writing RHIs: ELEV objects

### Not yet supported:

- Mixed datasets (how group always on top level)
- Single ray data (e.g. from fixed staring mode)
- Profiles

### Parameters

**filename** [str] Filename of file to create.

**radar** [Radar] Radar object to process.

**field\_names** [list of str] The list of fields from the radar object to save. If none all fields in the radar object will be saved.

**physical** [Bool] If true the physical values are stored. nodata parameter is equal to the \_Fill-Value parameter in the field metadata or the default Py-ART fill value. If false the data is converted into binary values using a linear conversion. The gain and offset are either specified in the metadata of the field with keywords 'scale\_factor' and 'add\_offset' or calculated on the fly. keyword '\_Write\_as\_dtype' specifies the datatype. It can be either 'uint8' or 'uint16'. The default datatype is uint8. The 'undetected' parameter is not used

**compression** [str] The type of compression for the datasets. Typical are "gzip" and "lzf".

**compression\_opts** [any] The compression options. In the case of gzip is the level between 0 to 9 (recommended 1 to 6). In the case of lzf there are not options.

`pyart.aux_io.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.

`pyart.aux_io.write_spectra(filename, radar, format='NETCDF4', time_reference=None,  
arm_time_variables=False, physical=True)`

Write a Radar Spectra object to a netCDF file.

The files produced by this routine follow the [CF/Radial standard](#). Attempts are also made 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`
- `_Contiguous`
- `_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.

**physical** [bool] True to store the radar fields as physical numbers, False will store the radar fields as binary if the keyword '\_Write\_as\_dtype' is in the field metadata. The gain and offset can be specified in the keyword 'scale\_factor' and 'add\_offset' or calculated on the fly.

---

## CORE (PYART.CORE)

Core Py-ART classes and function for interacting with weather radar data.

### 3.1 Core classes

<code>Radar(time, _range, fields, metadata, ...[, ...])</code>	A class for storing antenna coordinate radar data.
<code>Grid(time, fields, metadata, ...[, ...])</code>	A class for storing rectilinear gridded radar data in Cartesian coordinate.
<code>HorizontalWindProfile(height, speed, direction)</code>	Horizontal wind profile.
<code>RadarSpectra(time, _range, fields, metadata, ...)</code>	A class for storing antenna coordinate radar spectra data.

### 3.2 Coordinate transformations

<code>antenna_to_cartesian(ranges, azimuths, ...)</code>	Return Cartesian coordinates from antenna coordinates.
<code>antenna_vectors_to_cartesian(ranges, ...[, ...])</code>	Calculate Cartesian coordinate for gates from antenna coordinate vectors.
<code>cartesian_to_geographic(x, y, projparams)</code>	Cartesian to Geographic coordinate transform.
<code>cartesian_vectors_to_geographic(x, y, projparams)</code>	Cartesian vectors to Geographic coordinate transform.
<code>cartesian_to_geographic_aeqd(x, y, lon_0, lat_0)</code>	Azimuthal equidistant Cartesian to geographic coordinate transform.
<code>cartesian_to_antenna(x, y, z)</code>	Returns antenna coordinates from Cartesian coordinates.
<code>geographic_to_cartesian(lon, lat, projparams)</code>	Geographic to Cartesian coordinate transform.
<code>geographic_to_cartesian_aeqd(lon, lat, ...)</code>	Azimuthal equidistant geographic to Cartesian coordinate transform.
<code>wgs84_to_swissCH1903(lon, lat, alt[, ...])</code>	Convert WGS84 coordinates to swiss coordinates (CH1903 / LV03)

```
class pyart.core.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

<code>add_field(self, field_name, field_dict[, ...])</code>	Add a field to the object.
<code>get_point_longitude_latitude(self[, level, ...])</code>	Return arrays of longitude and latitude for a given grid height level.
<code>get_projparams(self)</code>	Return a projparam dict from the projection attribute.
<code>init_point_altitude(self)</code>	Initialize the point_altitude attribute.
<code>init_point_longitude_latitude(self)</code>	Initialize or reset the point_{longitude, latitudes} attributes.
<code>init_point_x_y_z(self)</code>	Initialize or reset the point_{x, y, z} attributes.
<code>to_xarray(self)</code>	Convert the Grid object to an xarray format.
<code>write(self, filename[, format, ...])</code>	Write the the Grid object to a NetCDF file.

`__class__`

alias of `builtins.type`

`__delattr__ (self, name, /)`

Implement `delattr(self, name)`.

`__dict__ = mappingproxy({'__module__': 'pyart.core.grid', '__doc__': "\n A class for`

`__dir__ (self, /)`

Default `dir()` implementation.

`__eq__ (self, value, /)`

Return `self==value`.

`__format__ (self, format_spec, /)`

Default object formatter.

`__ge__ (self, value, /)`

Return `self>=value`.

`__getattr__ (self, name, /)`

Return `getattr(self, name)`.

`__getstate__ (self)`

Return object's state which can be pickled.

`__gt__ (self, value, /)`

Return `self>value`.

`__hash__ (self, /)`

Return `hash(self)`.

`__init__ (self, 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)`

Initialize object.

`__init_subclass__ ()`

This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

`__le__ (self, value, /)`

Return `self<=value`.

`__lt__ (self, value, /)`

Return `self<value`.

**\_\_module\_\_** = 'pyart.core.grid'  
**\_\_ne\_\_** (*self*, *value*, /)  
Return self!=value.  
**\_\_new\_\_** (\*args, \*\*kwargs)  
Create and return a new object. See help(type) for accurate signature.  
**\_\_reduce\_\_** (*self*, /)  
Helper for pickle.  
**\_\_reduce\_ex\_\_** (*self*, *protocol*, /)  
Helper for pickle.  
**\_\_repr\_\_** (*self*, /)  
Return repr(self).  
**\_\_setattr\_\_** (*self*, *name*, *value*, /)  
Implement setattr(self, name, value).  
**\_\_setstate\_\_** (*self*, *state*)  
Restore unpicklable entries from pickled object.  
**\_\_sizeof\_\_** (*self*, /)  
Size of object in memory, in bytes.  
**\_\_str\_\_** (*self*, /)  
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** (*self*)  
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** (*self*, *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** (*self*, *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** (*self*)

Return a projparam dict from the projection attribute.

**init\_point\_altitude** (*self*)

Initialize the point\_altitude attribute.

**init\_point\_longitude\_latitude** (*self*)

Initialize or reset the point\_{longitude, latitudes} attributes.

**init\_point\_x\_y\_z** (*self*)

Initialize or reset the point\_{x, y, z} attributes.

**projection\_proj**

**to\_xarray** (*self*)

Convert the Grid object to an xarray format.

#### Attributes

**time** [dict] Time of the grid.

**fields** [dict of dicts] Moments from radars or other variables.

**longitude, latitude** [dict, 2D] Arrays of latitude and longitude for the grid height level.

**x, y, z** [dict, 1D] Distance from the grid origin for each Cartesian coordinate axis in a one dimensional array.

**write** (*self*, *filename*, *format*='NETCDF4', *arm\_time\_variables*=False, *arm\_alt\_lat\_lon\_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, optional] True to write the ARM standard time variables base\_time and time\_offset. False will not write these variables.

**arm\_alt\_lat\_lon\_variables** [bool, optional] True to write the ARM standard alt, lat, lon variables. False will not write these variables.

**class** pyart.core.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

**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.

**u\_wind** [array, 1D] U component of horizontal wind in meters per second.

**v\_wind** [array, 1D] V component of horizontal wind in meters per second.

#### Methods

---

<code>from_u_and_v(height, u_wind, v_wind)</code>	Create a HorizontalWindProfile instance from U and V components.
---------------------------------------------------	------------------------------------------------------------------

---

**\_\_class\_\_**

alias of `builtins.type`

**\_\_delattr\_\_** (*self*, *name*, /)

Implement `delattr(self, name)`.

**\_\_dict\_\_** = `mappingproxy({'__module__': 'pyart.core.wind_profile', '__doc__': '\n Hor`

**\_\_dir\_\_** (*self*, /)

Default `dir()` implementation.

**\_\_eq\_\_** (*self*, *value*, /)

Return `self==value`.

**\_\_format\_\_** (*self*, *format\_spec*, /)

Default object formatter.

**\_\_ge\_\_** (*self*, *value*, /)

Return `self>=value`.

**\_\_getattr\_\_** (*self*, *name*, /)

Return `getattr(self, name)`.

**\_\_gt\_\_** (*self*, *value*, /)

Return `self>value`.

**\_\_hash\_\_** (*self*, /)

Return `hash(self)`.

**\_\_init\_\_** (*self*, *height*, *speed*, *direction*, *latitude=None*, *longitude=None*)

initialize

**\_\_init\_subclass\_\_** ()

This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

**\_\_le\_\_** (*self*, *value*, /)  
Return self<=value.

**\_\_lt\_\_** (*self*, *value*, /)  
Return self<value.

**\_\_module\_\_** = 'pyart.core.wind\_profile'

**\_\_ne\_\_** (*self*, *value*, /)  
Return self!=value.

**\_\_new\_\_** (\*args, \*\*kwargs)  
Create and return a new object. See help(type) for accurate signature.

**\_\_reduce\_\_** (*self*, /)  
Helper for pickle.

**\_\_reduce\_ex\_\_** (*self*, *protocol*, /)  
Helper for pickle.

**\_\_repr\_\_** (*self*, /)  
Return repr(self).

**\_\_setattr\_\_** (*self*, *name*, *value*, /)  
Implement setattr(self, name, value).

**\_\_sizeof\_\_** (*self*, /)  
Size of object in memory, in bytes.

**\_\_str\_\_** (*self*, /)  
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** (*self*, *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.

```
class pyart.core.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, tar-  
get_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 modes this should be 'other'.

**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 None, indicating this parameter not available.

**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 access from the data in the sweep\_start\_ray\_index and sweep\_end\_ray\_index attributes. If the sweep locations needs to be modified, do this prior to accessing this attribute or use `init_rays_per_sweep()` to reset the attribute.

**target\_scan\_rate** [dict or None] Intended scan rate for each sweep. If not provided this attribute is set to None, indicating this parameter is not available.

- rays\_are\_indexed** [dict or None] Indication of whether ray angles are indexed to a regular grid in each sweep. If not provided this attribute is set to None, indicating ray angle spacing is not determined.
- ray\_angle\_res** [dict or None] If `rays_are_indexed` is not None, this provides the angular resolution of the grid. If not provided or available this attribute is set to None.
- azimuth** [dict] Azimuth of antenna, relative to true North. Azimuth angles are recommended to be expressed in the range of [0, 360], but other representations are not forbidden.
- elevation** [dict] Elevation of antenna, relative to the horizontal plane. Elevation angles are recommended to be expressed in the range of [-180, 180], but other representations are not forbidden.
- gate\_x, gate\_y, gate\_z** [LazyLoadDict] Location of each gate in a Cartesian coordinate system assuming a standard atmosphere with a 4/3 Earth's radius model. The data keys of these attributes are created upon first access from the data in the range, azimuth and elevation attributes. If these attributes are changed use `init_gate_x_y_z()` to reset.
- gate\_longitude, gate\_latitude** [LazyLoadDict] Geographic location of each gate. The projection parameter(s) defined in the `projection` attribute are used to perform an inverse map projection from the Cartesian gate locations relative to the radar location to longitudes and latitudes. If these attributes are changed use `init_gate_longitude_latitude()` to reset the attributes.
- projection** [dict or str] Projection parameters defining the map projection used to transform from Cartesian to geographic coordinates. The default dictionary sets the 'proj' key to 'pyart\_aeqd' indicating that the native Py-ART azimuthal equidistant projection is used. This can be modified to 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'.
- gate\_altitude** [LazyLoadDict] The altitude of each radar gate as calculated from the altitude of the radar and the Cartesian z location of each gate. If this attribute is changed use `init_gate_altitude()` to reset the attribute.
- scan\_rate** [dict or None] Actual antenna scan rate. If not provided this attribute is set to None, indicating this parameter is not available.
- antenna\_transition** [dict or None] Flag indicating if the antenna is in transition, 1 = yes, 0 = no. If not provided this attribute is set to None, indicating this parameter is not available.
- rotation** [dict or None] The rotation angle of the antenna. The angle about the aircraft longitudinal axis for a vertically scanning radar.
- 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 Earth-centric azimuth and elevation angles.

**instrument\_parameters** [dict of dicts or None] Instrument parameters, if not provided this attribute is set to None, indicating these parameters are not available. This dictionary also includes variables in the radar\_parameters CF/Radial subconvention.

**radar\_calibration** [dict of dicts or None] Instrument calibration parameters. If not provided this attribute is set to None, indicating these parameters are not available

**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

<code>add_field(self, field_name, dic[, ...])</code>	Add a field to the object.
<code>add_field_like(self, existing_field_name, ...)</code>	Add a field to the object with metadata from a existing field.
<code>check_field_exists(self, field_name)</code>	Check that a field exists in the fields dictionary.
<code>extract_sweeps(self, sweeps)</code>	Create a new radar contains only the data from select sweeps.
<code>get_azimuth(self, sweep[, copy])</code>	Return an array of azimuth angles for a given sweep.
<code>get_elevation(self, sweep[, copy])</code>	Return an array of elevation angles for a given sweep.
<code>get_end(self, sweep)</code>	Return the ending ray for a given sweep.
<code>get_field(self, sweep, field_name[, copy])</code>	Return the field data for a given sweep.
<code>get_gate_lat_lon_alt(self, sweep[, ...])</code>	Return the longitude, latitude and altitude gate locations.
<code>get_gate_x_y_z(self, sweep[, edges, ...])</code>	Return the x, y and z gate locations in meters for a given sweep.
<code>get_nyquist_vel(self, sweep[, check_uniform])</code>	Return the Nyquist velocity in meters per second for a given sweep.
<code>get_slice(self, sweep)</code>	Return a slice for selecting rays for a given sweep.
<code>get_start(self, sweep)</code>	Return the starting ray index for a given sweep.
<code>get_start_end(self, sweep)</code>	Return the starting and ending ray for a given sweep.
<code>info(self[, level, out])</code>	Print information on radar.
<code>init_gate_altitude(self)</code>	Initialize the gate_altitude attribute.
<code>init_gate_longitude_latitude(self)</code>	Initialize or reset the gate_longitude and gate_latitude attributes.
<code>init_gate_x_y_z(self)</code>	Initialize or reset the gate_{x, y, z} attributes.
<code>init_rays_per_sweep(self)</code>	Initialize or reset the rays_per_sweep attribute.
<code>iter_azimuth(self)</code>	Return an iterator which returns sweep azimuth data.
<code>iter_elevation(self)</code>	Return an iterator which returns sweep elevation data.
<code>iter_end(self)</code>	Return an iterator over the sweep end indices.
<code>iter_field(self, field_name)</code>	Return an iterator which returns sweep field data.
<code>iter_slice(self)</code>	Return an iterator which returns sweep slice objects.
<code>iter_start(self)</code>	Return an iterator over the sweep start indices.
<code>iter_start_end(self)</code>	Return an iterator over the sweep start and end indices.

`__class__`  
alias of `builtins.type`  
`__delattr__` (*self, name, /*)



Implement `delattr(self, name)`.

**`__dict__`** = `mappingproxy({'__module__': 'pyart.core.radar', '__doc__': '\n A class fo`

**`__dir__`**(*self*, /)  
Default `dir()` implementation.

**`__eq__`**(*self*, *value*, /)  
Return `self==value`.

**`__format__`**(*self*, *format\_spec*, /)  
Default object formatter.

**`__ge__`**(*self*, *value*, /)  
Return `self>=value`.

**`__getattr__`**(*self*, *name*, /)  
Return `getattr(self, name)`.

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

**`__gt__`**(*self*, *value*, /)  
Return `self>value`.

**`__hash__`**(*self*, /)  
Return `hash(self)`.

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

**`__init_subclass__`**( )  
This method is called when a class is subclassed.  
  
The default implementation does nothing. It may be overridden to extend subclasses.

**`__le__`**(*self*, *value*, /)  
Return `self<=value`.

**`__lt__`**(*self*, *value*, /)  
Return `self<value`.

**`__module__`** = `'pyart.core.radar'`

**`__ne__`**(*self*, *value*, /)  
Return `self!=value`.

**`__new__`**(*\*args*, *\*\*kwargs*)  
Create and return a new object. See `help(type)` for accurate signature.

**`__reduce__`**(*self*, /)  
Helper for pickle.

**`__reduce_ex__`**(*self*, *protocol*, /)  
Helper for pickle.

**`__repr__`**(*self*, /)  
Return `repr(self)`.

**\_\_setattr\_\_** (*self, name, value, /*)

Implement setattr(self, name, value).

**\_\_setstate\_\_** (*self, state*)

Restore unpicklable entries from pickled object.

**\_\_sizeof\_\_** (*self, /*)

Size of object in memory, in bytes.

**\_\_str\_\_** (*self, /*)

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** (*self, sweep*)

Check that a sweep number is in range.

**\_\_dic\_info** (*self, attr, level, out, dic=None, ident\_level=0*)

Print information on a dictionary attribute.

**add\_field** (*self, 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, optional] 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** (*self, 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, optional] 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** (*self*, *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** (*self*, *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** (*self*, *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** (*self*, *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** (*self*, *sweep*)

Return the ending ray for a given sweep.

**get\_field** (*self*, *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\_lat\_lon\_alt** (*self*, *sweep*, *reset\_gate\_coords=False*, *filter\_transitions=False*)

Return the longitude, latitude and altitude gate locations. Longitude and latitude are in degrees and altitude in meters.

With the default parameter this method returns the same data as contained in the `gate_latitude`, `gate_longitude` and `gate_altitude` 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. If coordinates have at all, please use the `reset_gate_coords` parameter.

#### Parameters

**sweep** [int] Sweep number to retrieve gate locations from, 0 based.

**reset\_gate\_coords** [bool, optional] Optional to reset the gate latitude, gate longitude and gate altitude attributes before using them in this function. This is useful when the geographic coordinates have changed and gate latitude, gate longitude and gate altitude need to be reset.

**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

**lat, lon, alt** [2D array] Array containing the latitude, longitude and altitude, for all gates in the sweep.

**get\_gate\_x\_y\_z** (*self*, *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** (*self*, *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** (*self*, *sweep*)

Return a slice for selecting rays for a given sweep.

**get\_start** (*self*, *sweep*)

Return the starting ray index for a given sweep.

**get\_start\_end** (*self*, *sweep*)

Return the starting and ending ray for a given sweep.

**info** (*self*, *level='standard'*, *out=<\_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Print information on radar.

#### Parameters

**level** [{‘compact’, ‘standard’, ‘full’, ‘c’, ‘s’, ‘f’}, optional] Level of information on radar object to print, compact is minimal information, standard more and full everything.

**out** [file-like, optional] Stream to direct output to, default is to print information to standard out (the screen).

**init\_gate\_altitude** (*self*)

Initialize the gate\_altitude attribute.

**init\_gate\_longitude\_latitude** (*self*)

Initialize or reset the gate\_longitude and gate\_latitude attributes.

**init\_gate\_x\_y\_z** (*self*)

Initialize or reset the gate\_{x, y, z} attributes.

**init\_rays\_per\_sweep** (*self*)

Initialize or reset the rays\_per\_sweep attribute.

**iter\_azimuth** (*self*)

Return an iterator which returns sweep azimuth data.

**iter\_elevation** (*self*)

Return an iterator which returns sweep elevation data.

**iter\_end** (*self*)

Return an iterator over the sweep end indices.

**iter\_field** (*self*, *field\_name*)

Return an iterator which returns sweep field data.

**iter\_slice** (*self*)  
Return an iterator which returns sweep slice objects.

**iter\_start** (*self*)  
Return an iterator over the sweep start indices.

**iter\_start\_end** (*self*)  
Return an iterator over the sweep start and end indices.

```
class pyart.core.RadarSpectra(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, npulses, Doppler_velocity=None, Doppler_frequency=None, 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: `pyart.core.radar.Radar`

A class for storing antenna coordinate radar spectra data. Based on the radar object class

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).

**npulses** [dict] number of pulses for each ray

**Doppler\_velocity** [dict or None] The Doppler velocity of each Doppler bin. The data has dimensions `nrays x npulses_max`

**Doppler\_frequency** [dict or None] The Doppler frequency of each Doppler bin. The data has dimensions `nrays x npulses_max`

**fields** [dict of dicts] Moment fields. The data has dimensions `nrays x ngates x npulses_max`

**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 modes this should be 'other'.

**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 None, indicating this parameter not available.

**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 access from the data in the `sweep_start_ray_index` and `sweep_end_ray_index` attributes. If the sweep locations needs to be modified, do this prior to accessing this attribute or use `init_rays_per_sweep()` to reset the attribute.
- target\_scan\_rate** [dict or None] Intended scan rate for each sweep. If not provided this attribute is set to None, indicating this parameter is not available.
- rays\_are\_indexed** [dict or None] Indication of whether ray angles are indexed to a regular grid in each sweep. If not provided this attribute is set to None, indicating ray angle spacing is not determined.
- ray\_angle\_res** [dict or None] If `rays_are_indexed` is not None, this provides the angular resolution of the grid. If not provided or available this attribute is set to None.
- azimuth** [dict] Azimuth of antenna, relative to true North. Azimuth angles are recommended to be expressed in the range of [0, 360], but other representations are not forbidden.
- elevation** [dict] Elevation of antenna, relative to the horizontal plane. Elevation angles are recommended to be expressed in the range of [-180, 180], but other representations are not forbidden.
- gate\_x, gate\_y, gate\_z** [LazyLoadDict] Location of each gate in a Cartesian coordinate system assuming a standard atmosphere with a 4/3 Earth's radius model. The data keys of these attributes are create upon first access from the data in the range, azimuth and elevation attributes. If these attributes are changed use `init_gate_x_y_z()` to reset.
- gate\_longitude, gate\_latitude** [LazyLoadDict] Geographic location of each gate. The projection parameter(s) defined in the `projection` attribute are used to perform an inverse map projection from the Cartesian gate locations relative to the radar location to longitudes and latitudes. If these attributes are changed use `init_gate_longitude_latitude()` to reset the attributes.
- projection** [dic or str] Projection parameters defining the map projection used to transform from Cartesian to geographic coordinates. The default dictionary sets the 'proj' key to 'pyart\_aeqd' indicating that the native Py-ART azimuthal equidistant projection is used. This can be modified to 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'.
- gate\_altitude** [LazyLoadDict] The altitude of each radar gate as calculated from the altitude of the radar and the Cartesian z location of each gate. If this attribute is changed use `init_gate_altitude()` to reset the attribute.
- scan\_rate** [dict or None] Actual antenna scan rate. If not provided this attribute is set to None, indicating this parameter is not available.
- antenna\_transition** [dict or None] Flag indicating if the antenna is in transition, 1 = yes, 0 = no. If not provided this attribute is set to None, indicating this parameter is not available.

**rotation** [dict or None] The rotation angle of the antenna. The angle about the aircraft longitudinal axis for a vertically scanning radar.

**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 Earth-centric azimuth and elevation angles.

**instrument\_parameters** [dict of dicts or None] Instrument parameters, if not provided this attribute is set to None, indicating these parameters are not available. This dictionary also includes variables in the radar\_parameters CF/Radial subconvention.

**radar\_calibration** [dict of dicts or None] Instrument calibration parameters. If not provided this attribute is set to None, indicating these parameters are not available

**ngates** [int] Number of gates (bins) in a ray.

**nrays** [int] Number of rays in the volume.

**npulses\_max** [int] Maximum number of pulses per ray in the volume.

**nsweeps** [int] Number of sweep in the volume.

## Methods

<code>add_field(self, field_name, dic[, ...])</code>	Add a field to the object.
<code>add_field_like(self, existing_field_name, ...)</code>	Add a field to the object with metadata from a existing field.
<code>check_field_exists(self, field_name)</code>	Check that a field exists in the fields dictionary.
<code>extract_sweeps(self, sweeps)</code>	Create a new radar contains only the data from select sweeps.
<code>get_azimuth(self, sweep[, copy])</code>	Return an array of azimuth angles for a given sweep.
<code>get_elevation(self, sweep[, copy])</code>	Return an array of elevation angles for a given sweep.
<code>get_end(self, sweep)</code>	Return the ending ray for a given sweep.
<code>get_field(self, sweep, field_name[, copy])</code>	Return the field data for a given sweep.
<code>get_gate_lat_lon_alt(self, sweep[, ...])</code>	Return the longitude, latitude and altitude gate locations.
<code>get_gate_x_y_z(self, sweep[, edges, ...])</code>	Return the x, y and z gate locations in meters for a given sweep.
<code>get_nyquist_vel(self, sweep[, check_uniform])</code>	Return the Nyquist velocity in meters per second for a given sweep.
<code>get_slice(self, sweep)</code>	Return a slice for selecting rays for a given sweep.
<code>get_start(self, sweep)</code>	Return the starting ray index for a given sweep.
<code>get_start_end(self, sweep)</code>	Return the starting and ending ray for a given sweep.
<code>info(self[, level, out])</code>	Print information on radar.
<code>init_gate_altitude(self)</code>	Initialize the gate_altitude attribute.
<code>init_gate_longitude_latitude(self)</code>	Initialize or reset the gate_longitude and gate_latitude attributes.

Continued on next page



Table 6 – continued from previous page

<code>init_gate_x_y_z(self)</code>	Initialize or reset the <code>gate_{x, y, z}</code> attributes.
<code>init_rays_per_sweep(self)</code>	Initialize or reset the <code>rays_per_sweep</code> attribute.
<code>iter_azimuth(self)</code>	Return an iterator which returns sweep azimuth data.
<code>iter_elevation(self)</code>	Return an iterator which returns sweep elevation data.
<code>iter_end(self)</code>	Return an iterator over the sweep end indices.
<code>iter_field(self, field_name)</code>	Return an iterator which returns sweep field data.
<code>iter_slice(self)</code>	Return an iterator which returns sweep slice objects.
<code>iter_start(self)</code>	Return an iterator over the sweep start indices.
<code>iter_start_end(self)</code>	Return an iterator over the sweep start and end indices.

```

__class__
    alias of builtins.type

__delattr__(self, name, /)
    Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyart.core.radar_spectra', '__doc__': "\n A
__dir__(self, /)
    Default dir() implementation.

__eq__(self, value, /)
    Return self==value.

__format__(self, format_spec, /)
    Default object formatter.

__ge__(self, value, /)
    Return self>=value.

__getattr__(self, name, /)
    Return getattr(self, name).

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

__gt__(self, value, /)
    Return self>value.

__hash__(self, /)
    Return hash(self).

__init__(self, time, _range, fields, metadata, scan_type, latitude, longitude, altitude, sweep_number,
sweep_mode, fixed_angle, sweep_start_ray_index, sweep_end_ray_index, azimuth, ele-
vation, npulses, Doppler_velocity=None, Doppler_frequency=None, 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, ro-
tation=None, tilt=None, roll=None, drift=None, heading=None, pitch=None, geo-
refs_applied=None)
    Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be overridden to extend subclasses.

__le__(self, value, /)
    Return self<=value.

```

**\_\_lt\_\_** (*self, value, /*)  
Return self<value.

**\_\_module\_\_** = 'pyart.core.radar\_spectra'

**\_\_ne\_\_** (*self, value, /*)  
Return self!=value.

**\_\_new\_\_** (*\*args, \*\*kwargs*)  
Create and return a new object. See help(type) for accurate signature.

**\_\_reduce\_\_** (*self, /*)  
Helper for pickle.

**\_\_reduce\_ex\_\_** (*self, protocol, /*)  
Helper for pickle.

**\_\_repr\_\_** (*self, /*)  
Return repr(self).

**\_\_setattr\_\_** (*self, name, value, /*)  
Implement setattr(self, name, value).

**\_\_setstate\_\_** (*self, state*)  
Restore unpicklable entries from pickled object.

**\_\_sizeof\_\_** (*self, /*)  
Size of object in memory, in bytes.

**\_\_str\_\_** (*self, /*)  
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** (*self, sweep*)  
Check that a sweep number is in range.

**\_\_dic\_info** (*self, attr, level, out, dic=None, ident\_level=0*)  
Print information on a dictionary attribute.

**add\_field** (*self, 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** (*self, 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** (*self*, *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** (*self*, *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** (*self*, *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** (*self*, *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** (*self*, *sweep*)

Return the ending ray for a given sweep.

**get\_field** (*self*, *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\_lat\_lon\_alt** (*self*, *sweep*, *reset\_gate\_coords=False*, *filter\_transitions=False*)

Return the longitude, latitude and altitude gate locations. Longitude and latitude are in degrees and altitude in meters.

With the default parameter this method returns the same data as contained in the `gate_latitude`, `gate_longitude` and `gate_altitude` 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. If coordinates have at all, please use the `reset_gate_coords` parameter.

#### Parameters

**sweep** [int] Sweep number to retrieve gate locations from, 0 based.

**reset\_gate\_coords** [bool, optional] Optional to reset the gate latitude, gate longitude and gate altitude attributes before using them in this function. This is useful when the geographic coordinates have changed and gate latitude, gate longitude and gate altitude need to be reset.

**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

**lat, lon, alt** [2D array] Array containing the latitude, longitude and altitude, for all gates in the sweep.

**get\_gate\_x\_y\_z** (*self*, *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** (*self*, *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** (*self*, *sweep*)

Return a slice for selecting rays for a given sweep.

**get\_start** (*self*, *sweep*)

Return the starting ray index for a given sweep.

**get\_start\_end** (*self*, *sweep*)

Return the starting and ending ray for a given sweep.

**info** (*self*, *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** (*self*)

Initialize the `gate_altitude` attribute.

**init\_gate\_longitude\_latitude** (*self*)

Initialize or reset the `gate_longitude` and `gate_latitude` attributes.

**init\_gate\_x\_y\_z** (*self*)

Initialize or reset the gate\_{x, y, z} attributes.

**init\_rays\_per\_sweep** (*self*)

Initialize or reset the rays\_per\_sweep attribute.

**iter\_azimuth** (*self*)

Return an iterator which returns sweep azimuth data.

**iter\_elevation** (*self*)

Return an iterator which returns sweep elevation data.

**iter\_end** (*self*)

Return an iterator over the sweep end indices.

**iter\_field** (*self*, *field\_name*)

Return an iterator which returns sweep field data.

**iter\_slice** (*self*)

Return an iterator which returns sweep slice objects.

**iter\_start** (*self*)

Return an iterator over the sweep start indices.

**iter\_start\_end** (*self*)

Return an iterator over the sweep start and end indices.

**pyart.core.antenna\_to\_cartesian** (*ranges*, *azimuths*, *elevations*)

Return Cartesian coordinates from antenna coordinates.

#### Parameters

**ranges** [array] Distances to the center of the radar gates (bins) in kilometers.

**azimuths** [array] Azimuth angle of the radar in degrees.

**elevations** [array] Elevation angle of the radar in degrees.

#### Returns

**x, y, z** [array] Cartesian coordinates in meters from the radar.

## Notes

The calculation for Cartesian coordinate is adapted from equations 2.28(b) and 2.28(c) of Doviak and Zrnic [1] assuming a standard atmosphere (4/3 Earth's radius model).

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

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

## References

[1]

`pyart.core.antenna_vectors_to_cartesian` (*ranges, azimuths, elevations, edges=False*)

Calculate Cartesian coordinate for gates from antenna coordinate vectors.

Calculates the Cartesian coordinates for the gate centers or edges for all gates from antenna coordinate vectors assuming a standard atmosphere (4/3 Earth's radius model). See `pyart.util.antenna_to_cartesian()` for details.

#### Parameters

**ranges** [array, 1D.] Distances to the center of the radar gates (bins) in meters.

**azimuths** [array, 1D.] Azimuth angles of the rays in degrees.

**elevations** [array, 1D.] Elevation angles of the rays in degrees.

**edges** [bool, optional] True to calculate the coordinates of the gate edges by interpolating between gates and extrapolating at the boundaries. False to calculate the gate centers.

#### Returns

**x, y, z** [array, 2D] Cartesian coordinates in meters from the center of the radar to the gate centers or edges.

`pyart.core.cartesian_to_antenna` (*x, y, z*)

Returns antenna coordinates from Cartesian coordinates.

#### Parameters

**x, y, z** [array] Cartesian coordinates in meters from the radar.

#### Returns

**ranges** [array] Distances to the center of the radar gates (bins) in m.

**azimuths** [array] Azimuth angle of the radar in degrees. [-180., 180]

**elevations** [array] Elevation angle of the radar in degrees.

`pyart.core.cartesian_to_geographic` (*x, y, projparams*)

Cartesian to Geographic coordinate transform.

Transform a set of Cartesian/Cartographic coordinates (*x, y*) to a geographic coordinate system (*lat, lon*) using *pyproj* or a build in Azimuthal equidistant projection.

#### Parameters

**x, y** [array-like] Cartesian coordinates in meters unless *R* is defined in different units in the *projparams* parameter.

**projparams** [dict or str] Projection parameters passed to *pyproj.Proj*. If this parameter is a dictionary with a 'proj' key equal to 'pyart\_aeqd' then a azimuthal equidistant projection will be used that is native to Py-ART and does not require *pyproj* to be installed. In this case a non-default value of *R* can be specified by setting the 'R' key to the desired value.

#### Returns

**lon, lat** [array] Longitude and latitude of the Cartesian coordinates in degrees.

`pyart.core.cartesian_to_geographic_aeqd` (*x, y, lon\_0, lat\_0, R=6370997.0*)

Azimuthal equidistant Cartesian to geographic coordinate transform.

Transform a set of Cartesian/Cartographic coordinates (*x, y*) to geographic coordinate system (*lat, lon*) using a

azimuthal equidistant map projection [1].

$$\begin{aligned}lat &= \arcsin(\cos(c) * \sin(lat_0) + (y * \sin(c) * \cos(lat_0) / \rho)) \\lon &= lon_0 + \arctan 2(x * \sin(c), \rho * \cos(lat_0) * \cos(c) - y * \sin(lat_0) * \sin(c)) \\ \rho &= \sqrt{x^2 + y^2} \\ c &= \rho / R\end{aligned}$$

Where x, y are the Cartesian position from the center of projection; lat, lon the corresponding latitude and longitude; lat\_0, lon\_0 are the latitude and longitude of the center of the projection; R is the radius of the earth (defaults to ~6371 km). lon is adjusted to be between -180 and 180.

#### Parameters

- x, y** [array-like] Cartesian coordinates in the same units as R, typically meters.
- lon\_0, lat\_0** [float] Longitude and latitude, in degrees, of the center of the projection.
- R** [float, optional] Earth radius in the same units as x and y. The default value is in units of meters.

#### Returns

- lon, lat** [array] Longitude and latitude of Cartesian coordinates in degrees.

### References

[1]

`pyart.core.cartesian_vectors_to_geographic(x, y, projparams, edges=False)`

Cartesian vectors to Geographic coordinate transform.

Transform a set of Cartesian/Cartographic coordinate vectors (x, y) to a geographic coordinate system (lat, lon) using pyproj or a build in Azimuthal equidistant projection finding the coordinates edges in Cartesian space if requested.

#### Parameters

- x, y** [array 1D.] Cartesian coordinate vectors in meters unless R is defined in different units in the projparams parameter.
- projparams** [dict or str] Projection parameters passed to pyproj.Proj. If this parameter is a dictionary with a 'proj' key equal to 'pyart\_aeqd' then a azimuthal equidistant projection will be used that is native to Py-ART and does not require pyproj to be installed. In this case a non-default value of R can be specified by setting the 'R' key to the desired value.
- edges** [bool, optional] True to calculate the coordinates of the geographic edges by interpolating between Cartesian points and extrapolating at the boundaries. False to calculate the coordinate centers.

#### Returns

- lon, lat** [array] Longitude and latitude of the Cartesian coordinates in degrees.

`pyart.core.geographic_to_cartesian(lon, lat, projparams)`

Geographic to Cartesian coordinate transform.

Transform a set of Geographic coordinate (lat, lon) to a Cartesian/Cartographic coordinate (x, y) using pyproj or a build in Azimuthal equidistant projection.

#### Parameters

- lon, lat** [array-like] Geographic coordinates in degrees.



**projparams** [dict or str] Projection parameters passed to pyproj.Proj. If this parameter is a dictionary with a 'proj' key equal to 'pyart\_aeqd' then an azimuthal equidistant projection will be used that is native to Py-ART and does not require pyproj to be installed. In this case a non-default value of R can be specified by setting the 'R' key to the desired value.

#### Returns

**x, y** [array-like] Cartesian coordinates in meters unless projparams defines a value for R in different units.

`pyart.core.geographic_to_cartesian_aeqd(lon, lat, lon_0, lat_0, R=6370997.0)`

Azimuthal equidistant geographic to Cartesian coordinate transform.

Transform a set of geographic coordinates (lat, lon) to Cartesian/Cartographic coordinates (x, y) using an azimuthal equidistant map projection [1].

$$\begin{aligned}x &= R * k * \cos(lat) * \sin(lon - lon_0) \\y &= R * k * [\cos(lat_0) * \sin(lat) - \sin(lat_0) * \cos(lat) * \cos(lon - lon_0)] \\k &= c / \sin(c) \\c &= \arccos(\sin(lat_0) * \sin(lat) + \cos(lat_0) * \cos(lat) * \cos(lon - lon_0))\end{aligned}$$

Where x, y are the Cartesian position from the center of projection; lat, lon the corresponding latitude and longitude; lat\_0, lon\_0 are the latitude and longitude of the center of the projection; R is the radius of the earth (defaults to ~6371 km).

#### Parameters

**lon, lat** [array-like] Longitude and latitude coordinates in degrees.

**lon\_0, lat\_0** [float] Longitude and latitude, in degrees, of the center of the projection.

**R** [float, optional] Earth radius in the same units as x and y. The default value is in units of meters.

#### Returns

**x, y** [array] Cartesian coordinates in the same units as R, typically meters.

## References

[1]

`pyart.core.wgs84_to_swissCH1903(lon, lat, alt, no_altitude_transform=False)`

Convert WGS84 coordinates to swiss coordinates (CH1903 / LV03)

The formulas for the coordinates transformation are taken from: "Formeln und Konstanten für die Berechnung der Schweizerischen schiefachsigen Zylinderprojektion und der Transformation zwischen Koordinatensystemen", chapter 4. "Näherungslösungen CH1903 <=> WGS84" Bundesamt für Landestopografie swisstopo (<http://www.swisstopo.admin.ch>), Oktober 2008

#### Parameters

**lon, lat** [array-like] Geographic coordinates WGS84 in degrees.

**alt** [array-like] Altitude in m

**no\_altitude\_transform** [bool] If set, do not convert altitude

#### Returns

**chy, chx, chh** [array-like] Coordinates in swiss CH1903 coordinates in meter



## BRIDGING TO OTHER TOOLKITS (PYART.BRIDGE)

Py-ART can act as bridge to other community software projects.

The functionality in this namespace is available in other pyart namespaces.

### 4.1 Phase functions

---

<code>texture_of_complex_phase(radar[, ...])</code>	Calculate the texture of the differential phase field.
-----------------------------------------------------	--------------------------------------------------------

---



## FILTERS (PYART.FILTERS)

Classes for specifying what gates are included and excluded from routines.

### 5.1 Filtering radar data

<code>GateFilter(radar[, exclude_based])</code>	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.
<code>moment_based_gate_filter(radar[, ncp_field, ...])</code>	Create a filter which removes undesired gates based on moments.
<code>moment_and_texture_based_gate_filter(radar[, ...])</code>	Create a filter which removes undesired gates based on texture of moments.
<code>snr_based_gate_filter(radar[, snr_field, ...])</code>	Create a filter which removes undesired gates based on SNR.
<code>visibility_based_gate_filter(radar[, ...])</code>	Create a filter which removes undesired gates based on visibility.
<code>class_based_gate_filter(radar[, field, ...])</code>	Create a filter which removes undesired gates based on class values
<code>temp_based_gate_filter(radar[, temp_field, ...])</code>	Create a filter which removes undesired gates based on temperature.
<code>iso0_based_gate_filter(radar[, iso0_field, ...])</code>	Create a filter which removes undesired gates based height over the iso0.
<code>birds_gate_filter(radar[, zdr_field, ...])</code>	Create a filter which removes data not suspected of being birds

**class** `pyart.filters.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] Return a copy of the excluded gates.

***gate\_included*** [array, dtype=bool] Return a copy of the included gates.

## Methods

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

Continued on next page

Table 2 – continued from previous page

<code>include_not_masked(self, field[, ...])</code>	Include gates where a given field is not masked.
<code>include_not_transition(self[, trans_value, ...])</code>	Include all gates in rays not marked as in transition between sweeps.
<code>include_outside(self, field, v1, v2[, ...])</code>	Include gates where a given field is outside a given interval.
<code>include_valid(self, field[, exclude_masked, op])</code>	Include gates where a valid value occurs in a field (not NaN or inf).

```

__class__
    alias of builtins.type

__delattr__(self, name, /)
    Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyart.filters.gatefilter', '__doc__': "\n A
__dir__(self, /)
    Default dir() implementation.

__eq__(self, value, /)
    Return self==value.

__format__(self, format_spec, /)
    Default object formatter.

__ge__(self, value, /)
    Return self>=value.

__getattr__(self, name, /)
    Return getattr(self, name).

__gt__(self, value, /)
    Return self>value.

__hash__(self, /)
    Return hash(self).

__init__(self, radar, exclude_based=True)
    initialize

__init_subclass__()
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be overridden to extend subclasses.

__le__(self, value, /)
    Return self<=value.

__lt__(self, value, /)
    Return self<value.

__module__ = 'pyart.filters.gatefilter'

__ne__(self, value, /)
    Return self!=value.

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

__reduce__(self, /)
    Helper for pickle.

```

**\_\_reduce\_ex\_\_** (*self, protocol, /*)  
Helper for pickle.

**\_\_repr\_\_** (*self, /*)  
Return repr(self).

**\_\_setattr\_\_** (*self, name, value, /*)  
Implement setattr(self, name, value).

**\_\_sizeof\_\_** (*self, /*)  
Size of object in memory, in bytes.

**\_\_str\_\_** (*self, /*)  
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** (*self, field*)  
Check that the field exists and retrieve field data.

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

**copy** (*self*)  
Return a copy of the gatefilter.

**exclude\_above** (*self, field, value, exclude\_masked=True, op='or', inclusive=False*)  
Exclude gates where a given field is above a given value.

**exclude\_all** (*self*)  
Exclude all gates.

**exclude\_below** (*self, 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** (*self*, *field*, *value*, *exclude\_masked=True*, *op='or'*)  
Exclude gates where a given field is equal to a value.

**exclude\_gates** (*self*, *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** (*self*, *field*, *v1*, *v2*, *exclude\_masked=True*, *op='or'*, *inclusive=True*)  
Exclude gates where a given field is inside a given interval.

**exclude\_invalid** (*self*, *field*, *exclude\_masked=True*, *op='or'*)  
Exclude gates where an invalid value occurs in a field (NaNs or infs).

**exclude\_masked** (*self*, *field*, *exclude\_masked=True*, *op='or'*)  
Exclude gates where a given field is masked.

**exclude\_none** (*self*)  
Exclude no gates, include all gates.

**exclude\_not\_equal** (*self*, *field*, *value*, *exclude\_masked=True*, *op='or'*)  
Exclude gates where a given field is not equal to a value.

**exclude\_outside** (*self*, *field*, *v1*, *v2*, *exclude\_masked=True*, *op='or'*, *inclusive=False*)  
Exclude gates where a given field is outside a given interval.

**exclude\_transition** (*self*, *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**

Return a copy of the excluded gates.

**gate\_included**

Return a copy of the included gates.

**include\_above** (*self*, *field*, *value*, *exclude\_masked=True*, *op='and'*, *inclusive=False*)

Include gates where a given field is above a given value.

**include\_all** (*self*)

Include all gates.

**include\_below** (*self*, *field*, *value*, *exclude\_masked=True*, *op='and'*, *inclusive=False*)

Include gates where a given field is below a given value.

**include\_equal** (*self*, *field*, *value*, *exclude\_masked=True*, *op='and'*)

Include gates where a given field is equal to a value.

**include\_gates** (*self*, *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** (*self*, *field*, *v1*, *v2*, *exclude\_masked=True*, *op='and'*, *inclusive=True*)

Include gates where a given field is inside a given interval.

**include\_none** (*self*)

Include no gates, exclude all gates.

**include\_not\_equal** (*self*, *field*, *value*, *exclude\_masked=True*, *op='and'*)

Include gates where a given field is not equal to a value.

**include\_not\_masked** (*self*, *field*, *exclude\_masked=True*, *op='and'*)

Include gates where a given field in not masked.

**include\_not\_transition** (*self*, *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** (*self*, *field*, *v1*, *v2*, *exclude\_masked=True*, *op='and'*, *inclusive=False*)

Include gates where a given field is outside a given interval.

**include\_valid** (*self*, *field*, *exclude\_masked=True*, *op='and'*)

Include gates where a valid value occurs in a field (not NaN or inf).

```
pyart.filters.birds_gate_filter(radar, zdr_field=None, rhv_field=None, refl_field=None,
                                vel_field=None, max_zdr=3.0, max_rhv=0.9, min_refl=0.0,
                                max_refl=20.0, vel_lim=1.0, rmin=2000.0, rmax=25000.0,
                                elmin=1.0, elmax=85.0)
```

Create a filter which removes data not suspected of being birds

Creates a gate filter in which the following gates are excluded:

- Gates where the instrument is transitioning between sweeps.
- Gates where the reflectivity is beyond min\_refl and max\_refl
- Gates where the co-polar correlation coefficient is above max\_rhv
- Gates where the differential reflectivity is above max\_zdr
- Gates where the Doppler velocity is within the interval given by +-vel\_lim
- Gates where any of the above fields are masked or contain invalid values (NaNs or infs).
- Gates outside the range given by range min and range max
- 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, zdr\_field, rhv\_field, vel\_field** [str] Names of the radar fields which contain the reflectivity, differential reflectivity, co-polar correlation coefficient, and Doppler velocity 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.

**max\_zdr, max\_rhv** [float] Maximum values for the differential reflectivity and co-polar correlation coefficient. 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\_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.

**rmin, rmax** [float] Minimum and maximum ranges [m]

**elmin, elmax** [float] Minimum and maximum elevations [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.class_based_gate_filter(radar, field=None, kept_values=None)`

Create a filter which removes undesired gates based on class values

#### Parameters

**radar** [Radar] Radar object from which the gate filter will be built.

**field** [str] Name of the radar field which contains the classification. A value of None for will use the default field name for the hydrometeor classification as defined in the Py-ART configuration file.

**kept\_values** [list of ints or none] The class values to keep

#### 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.iso0_based_gate_filter(radar, iso0_field=None, max_h_iso0=0.0, thickness=400.0, beamwidth=None)`

Create a filter which removes undesired gates based height over the iso0. 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.

**iso0\_field** [str] Name of the radar field which contains the height relative to the iso0. A value of None for will use the default field name as defined in the Py-ART configuration file.

**max\_h\_iso0** [float] Maximum height relative to the iso0 in m. 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.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,      tex-
                                                    trefl_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.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.snr_based_gate_filter` (*radar*, *snr\_field=None*, *min\_snr=10.0*, *max\_snr=None*)  
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.

**max\_snr** [float] Maximum value for the SNR

#### 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.temp_based_gate_filter(radar, temp_field=None, min_temp=0.0, thick-  
                                     ness=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.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.

---





## RADAR CORRECTIONS (PYART.CORRECT)

Correct radar fields.

### 6.1 Velocity unfolding

<code>dealias_fourdd(radar[, last_radar, ...])</code>	Dealias Doppler velocities using the 4DD algorithm.
<code>dealias_unwrap_phase(radar[, unwrap_unit, ...])</code>	Dealias Doppler velocities using multi-dimensional phase unwrapping.
<code>dealias_region_based(radar[, ref_vel_field, ...])</code>	Dealias Doppler velocities using a region based algorithm.

### 6.2 Other corrections

<code>calculate_attenuation(radar, z_offset[, ...])</code>	Calculate the attenuation from a polarimetric radar using Z-PHI method.
<code>calculate_attenuation_zphi(radar[, doc, ...])</code>	Calculate the attenuation and the differential attenuation from a polarimetric radar using Z-PHI method..
<code>calculate_attenuation_philinear(radar[, ...])</code>	Calculate the attenuation and the differential attenuation from a polarimetric radar using linear dependence with PhiDP.
<code>phase_proc_lp(radar, offset[, debug, ...])</code>	Phase process using a LP method [1].
<code>det_sys_phase_ray(radar[, ind_rmin, ...])</code>	Public method Alternative determination of the system phase.
<code>correct_sys_phase(radar[, ind_rmin, ...])</code>	correction of the system offset.
<code>smooth_phidp_single_window(radar[, ...])</code>	correction of the system offset and smoothing using one window
<code>smooth_phidp_double_window(radar[, ...])</code>	correction of the system offset and smoothing using two window
<code>despeckle_field(radar, field[, label_dict, ...])</code>	Despeckle a radar volume by identifying small objects in each scan and masking them out.
<code>correct_noise_rhoHV(radar[, urhohv_field, ...])</code>	Corrects RhoHV for noise according to eq.
<code>correct_bias(radar[, bias, field_name])</code>	Corrects a radar data bias.
<code>correct_visibility(radar[, vis_field, ...])</code>	Corrects the reflectivity according to visibility.
<code>est_rhoHV_rain(radar[, ind_rmin, ind_rmax, ...])</code>	Estimates the quantiles of RhoHV in rain for each sweep

Continued on next page

Table 2 – continued from previous page

<code>est_zdr_precip(radar[, ind_rmin, ind_rmax, ...])</code>	Filters out all undesired data to be able to estimate ZDR bias, either in moderate rain or from vertically pointing scans
<code>est_zdr_snow(radar[, ind_rmin, ind_rmax, ...])</code>	Filters out all undesired data to be able to estimate ZDR bias in snow
<code>selfconsistency_bias(radar, zdr_kdpzh_dict)</code>	Estimates reflectivity bias at each ray using the self-consistency algorithm by Gourley
<code>selfconsistency_bias2(radar, zdr_kdpzh_dict)</code>	Estimates reflectivity bias at each ray using the self-consistency algorithm by Gourley
<code>selfconsistency_kdp_phidp(radar, zdr_kdpzh_dict)</code>	Estimates KDP and PhiDP in rain from Zh and ZDR using a selfconsistency relation between ZDR, Zh and KDP.
<code>get_sun_hits(radar[, delev_max, dazim_max, ...])</code>	get data from suspected sun hits.
<code>get_sun_hits_ivic(radar[, delev_max, ...])</code>	get data from suspected sun hits.
<code>sun_retrieval(az_rad, az_sun, el_rad, ..., ...)</code>	Estimates sun parameters from sun hits
<code>phase_proc_lp_gf(radar[, gatefilter, debug, ...])</code>	Phase process using a LP method [1] using Py-ART's Gatefilter.

## 6.3 Helper functions

<code>find_objects(radar, field, threshold[, ...])</code>	Find objects (i.e., contiguous gates) in one or more sweeps that match thresholds.
<code>get_mask_fzl(radar[, fzl, doc, min_temp, ...])</code>	Constructs a mask to mask data placed thickness m below data at min_temp and beyond.
<code>sun_power(solar_flux, pulse_width, wavelen, ...)</code>	computes the theoretical sun power detected at the antenna [dBm] as it would be without atmospheric attenuation (sun power at top of the atmosphere) for a given solar flux and radar characteristics
<code>ptoa_to_sf(ptoa, pulse_width, wavelen, ...)</code>	Converts the sun power at the top of the atmosphere (in dBm) into solar flux.
<code>solar_flux_lookup(solar_flux, wavelen)</code>	Given the observed solar flux at 10.7 cm wavelength, returns the solar flux at the given radar wavelength
<code>scanning_losses(angle_step, beamwidth)</code>	Given the antenna beam width and the integration angle, compute the losses due to the fact that the sun is not a point target and the antenna is scanning
<code>smooth_masked(raw_data[, wind_len, ...])</code>	smoothes the data using a rolling window.
<code>sun_position_pysolar(dt, lat, lon[, elevation])</code>	obtains the sun position in antenna coordinates using the pysolar library.
<code>sun_position_mfr(dt, lat_deg, lon_deg[, ...])</code>	Calculate the sun position for the given time (dt) at the given position (lat, lon).

**class** `pyart.correct.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] Return a copy of the excluded gates.

**gate\_included** [array, dtype=bool] Return a copy of the included gates.

## Methods

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

Continued on next page

Table 4 – continued from previous page

<code>include_inside(self, field, v1, v2[, ...])</code>	Include gates where a given field is inside a given interval.
<code>include_none(self)</code>	Include no gates, exclude all gates.
<code>include_not_equal(self, field, value[, ...])</code>	Include gates where a given field is not equal to a value.
<code>include_not_masked(self, field[, ...])</code>	Include gates where a given field is not masked.
<code>include_not_transition(self[, trans_value, ...])</code>	Include all gates in rays not marked as in transition between sweeps.
<code>include_outside(self, field, v1, v2[, ...])</code>	Include gates where a given field is outside a given interval.
<code>include_valid(self, field[, exclude_masked, op])</code>	Include gates where a valid value occurs in a field (not NaN or inf).

```

__class__
    alias of builtins.type

__delattr__(self, name, /)
    Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyart.filters.gatefilter', '__doc__': "\n A
__dir__(self, /)
    Default dir() implementation.

__eq__(self, value, /)
    Return self==value.

__format__(self, format_spec, /)
    Default object formatter.

__ge__(self, value, /)
    Return self>=value.

__getattr__(self, name, /)
    Return getattr(self, name).

__gt__(self, value, /)
    Return self>value.

__hash__(self, /)
    Return hash(self).

__init__(self, radar, exclude_based=True)
    initialize

__init_subclass__()
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be overridden to extend subclasses.

__le__(self, value, /)
    Return self<=value.

__lt__(self, value, /)
    Return self<value.

__module__ = 'pyart.filters.gatefilter'

__ne__(self, value, /)
    Return self!=value.

```

**\_\_new\_\_** (\*args, \*\*kwargs)  
Create and return a new object. See help(type) for accurate signature.

**\_\_reduce\_\_** (self, /)  
Helper for pickle.

**\_\_reduce\_ex** (self, protocol, /)  
Helper for pickle.

**\_\_repr\_\_** (self, /)  
Return repr(self).

**\_\_setattr\_\_** (self, name, value, /)  
Implement setattr(self, name, value).

**\_\_sizeof\_\_** (self, /)  
Size of object in memory, in bytes.

**\_\_str\_\_** (self, /)  
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** (self, field)  
Check that the field exists and retrieve field data.

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

**copy** (self)  
Return a copy of the gatefilter.

**exclude\_above** (self, field, value, exclude\_masked=True, op='or', inclusive=False)  
Exclude gates where a given field is above a given value.

**exclude\_all** (self)  
Exclude all gates.

**exclude\_below** (self, 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** (*self*, *field*, *value*, *exclude\_masked=True*, *op='or'*)  
Exclude gates where a given field is equal to a value.

**exclude\_gates** (*self*, *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** (*self*, *field*, *v1*, *v2*, *exclude\_masked=True*, *op='or'*, *inclusive=True*)  
Exclude gates where a given field is inside a given interval.

**exclude\_invalid** (*self*, *field*, *exclude\_masked=True*, *op='or'*)  
Exclude gates where an invalid value occurs in a field (NaNs or infs).

**exclude\_masked** (*self*, *field*, *exclude\_masked=True*, *op='or'*)  
Exclude gates where a given field is masked.

**exclude\_none** (*self*)  
Exclude no gates, include all gates.

**exclude\_not\_equal** (*self*, *field*, *value*, *exclude\_masked=True*, *op='or'*)  
Exclude gates where a given field is not equal to a value.

**exclude\_outside** (*self*, *field*, *v1*, *v2*, *exclude\_masked=True*, *op='or'*, *inclusive=False*)  
Exclude gates where a given field is outside a given interval.

**exclude\_transition** (*self*, *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**

Return a copy of the excluded gates.

#### **gate\_included**

Return a copy of the included gates.

**include\_above** (*self*, *field*, *value*, *exclude\_masked=True*, *op='and'*, *inclusive=False*)

Include gates where a given field is above a given value.

**include\_all** (*self*)

Include all gates.

**include\_below** (*self*, *field*, *value*, *exclude\_masked=True*, *op='and'*, *inclusive=False*)

Include gates where a given field is below a given value.

**include\_equal** (*self*, *field*, *value*, *exclude\_masked=True*, *op='and'*)

Include gates where a given field is equal to a value.

**include\_gates** (*self*, *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** (*self*, *field*, *v1*, *v2*, *exclude\_masked=True*, *op='and'*, *inclusive=True*)

Include gates where a given field is inside a given interval.

**include\_none** (*self*)

Include no gates, exclude all gates.

**include\_not\_equal** (*self*, *field*, *value*, *exclude\_masked=True*, *op='and'*)

Include gates where a given field is not equal to a value.

**include\_not\_masked** (*self*, *field*, *exclude\_masked=True*, *op='and'*)

Include gates where a given field in not masked.

**include\_not\_transition** (*self*, *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** (*self*, *field*, *v1*, *v2*, *exclude\_masked=True*, *op='and'*, *inclusive=False*)

Include gates where a given field is outside a given interval.

**include\_valid** (*self*, *field*, *exclude\_masked=True*, *op='and'*)

Include gates where a valid value occurs in a field (not NaN or inf).

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

Calculate the attenuation from a polarimetric radar using Z-PHI method. Parameters ——— radar : Radar

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

**z\_offset** [float] Horizontal reflectivity offset in dBZ.

**debug** [bool, optional] True to print debugging information, False suppressed this printing.

**doc** [float, optional] Number of gates at the end of each ray to to remove from the calculation.

**fzl** [float, optional] Freezing layer, gates above this point are not included in the correction.

**rhv\_min** [float, optional] Minimum `copol_coeff` value to consider valid.

**ncp\_min** [float, optional] Minimum `norm_coherent_power` to consider valid.

**a\_coef** [float, optional] A coefficient in attenuation calculation.

**beta** [float, optional] Beta parameter in attenuation calculation.

**refl\_field** [str, optional] Name of the reflectivity field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

**phidp\_field** [str, optional] Name of the differential phase field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.



**nco\_field** [str, optional] Name of the normalized coherent power field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

**zdr\_field** [str, optional] Name of the differential reflectivity field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file. This will only be used if it is available.

**spec\_at\_field** [str, optional] Name of the specific attenuation field 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.

**corr\_refl\_field** [str, optional] Name of the corrected reflectivity field 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.

### Returns

**spec\_at** [dict] Field dictionary containing the specific attenuation.

**cor\_z** [dict] Field dictionary containing the corrected reflectivity.

### References

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

```
pyart.correct.calculate_attenuation_philinear(radar, doc=None, fzl=None,
                                             pia_coef=None, gate-
                                             filter=None, pida_coef=None,
                                             refl_field=None, phidp_field=None,
                                             zdr_field=None, temp_field=None,
                                             iso0_field=None, spec_at_field=None,
                                             pia_field=None, corr_refl_field=None,
                                             spec_diff_at_field=None,
                                             pida_field=None, corr_zdr_field=None,
                                             temp_ref='temperature')
```

Calculate the attenuation and the differential attenuation from a polarimetric radar using linear dependence with PhiDP. The attenuation is computed up to a user defined freezing level height, where temperatures in a temperature field are positive or where the height relative to the iso0 is 0. 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, optional] Number of gates at the end of each ray to remove from the calculation.

**fzl** [float, optional] Freezing layer, gates above this point are not included in the correction.

**gatefilter** [GateFilter, optional] The gates to exclude from the calculation. This, combined with the gates above fzl, will be excluded from the correction. Set to None to not use a gatefilter.

**pia\_coef** [float, optional] Coefficient in path integrated attenuation calculation

**pida\_coef** [float, optional] Coefficient in path integrated differential attenuation calculation

**refl\_field** [str, optional] Name of the reflectivity field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

**phidp\_field** [str, optional] Name of the differential phase field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

**zdr\_field** [str, optional] Name of the differential reflectivity field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file. This will only be used if it is available.

**temp\_field** [str, optional] Name of the temperature field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

**iso0\_field** [str, optional] Name of the field for the height above the 0C isotherm for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file. This will only be used if it is available.

**spec\_at\_field** [str, optional] Name of the specific attenuation field 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.

**pia\_field** [str, optional] Name of the path integrated attenuation field 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.

**corr\_refl\_field** [str, optional] Name of the corrected reflectivity field 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** [str, optional] Name of the specific differential attenuation field 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. This will only be calculated if ZDR is available.

**corr\_zdr\_field** [str, optional] Name of the corrected differential reflectivity field 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. This will only be calculated if ZDR is available.

**temp\_ref** [str, optional] The field use as reference for temperature. Can be either temperature, height\_over\_iso0 or fixed\_fzl.

## Returns

**spec\_at** [dict] Field dictionary containing the specific attenuation.

**pia\_dict** [dict] Field dictionary containing the path integrated attenuation.

**cor\_z** [dict] Field dictionary containing the corrected reflectivity.

**spec\_diff\_at** [dict] Field dictionary containing the specific differential attenuation.

**pida\_dict** [dict] Field dictionary containing the path integrated differential attenuation.

**cor\_zdr** [dict] Field dictionary containing the corrected differential reflectivity.

```
pyart.correct.calculate_attenuation_zphi (radar,          doc=None,          fzl=None,
                                         smooth_window_len=5,      gatefilter=None,
                                         a_coef=None,      beta=None,      c=None,
                                         d=None,  refl_field=None,  phidp_field=None,
                                         zdr_field=None,      temp_field=None,
                                         iso0_field=None,      spec_at_field=None,
                                         pia_field=None,      corr_refl_field=None,
                                         spec_diff_at_field=None,  pida_field=None,
                                         corr_zdr_field=None, temp_ref='temperature')
```

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, optional] Number of gates at the end of each ray to remove from the calculation.
- fzl** [float, optional] Freezing layer, gates above this point are not included in the correction.
- gatefilter** [GateFilter, optional] The gates to exclude from the calculation. This, combined with the gates above fzl, will be excluded from the correction. Set to None to not use a gatefilter.
- smooth\_window\_len** [int, optional] Size, in range bins, of the smoothing window
- a\_coef** [float, optional] A coefficient in attenuation calculation.
- beta** [float, optional] Beta parameter in attenuation calculation.
- c, d** [float, optional] coefficient and exponent of the power law that relates attenuation with differential attenuation
- refl\_field** [str, optional] Name of the reflectivity field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.
- phidp\_field** [str, optional] Name of the differential phase field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.
- zdr\_field** [str, optional] Name of the differential reflectivity field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file. This will only be used if it is available.
- temp\_field** [str, optional] Name of the temperature field used for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.
- iso0\_field** [str, optional] Name of the field for the height above the 0C isotherm for the attenuation correction. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file. This will only be used if it is available.
- spec\_at\_field** [str, optional] Name of the specific attenuation field 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.
- pia\_field** [str, optional] Name of the path integrated attenuation field 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.

**corr\_refl\_field** [str, optional] Name of the corrected reflectivity field 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** [str, optional] Name of the specific differential attenuation field 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. This will only be calculated if ZDR is available.

**pida\_field** [str, optional] Name of the path integrated differential attenuation field 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. This will only be calculated if ZDR is available.

**corr\_zdr\_field** [str, optional] Name of the corrected differential reflectivity field 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. This will only be calculated if ZDR is available.

**temp\_ref** [str, optional] the field use as reference for temperature. Can be either temperature, height\_over\_iso0 or fixed\_fz1

#### Returns

**spec\_at** [dict] Field dictionary containing the specific attenuation.

**pia\_dict** [dict] Field dictionary containing the path integrated attenuation.

**cor\_z** [dict] Field dictionary containing the corrected reflectivity.

**spec\_diff\_at** [dict] Field dictionary containing the specific differential attenuation.

**pida\_dict** [dict] Field dictionary containing the path integrated 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.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, optional] The bias magnitude.

**field\_name: str, optional** Names of the field to be corrected.

#### Returns

**corrected\_field** [dict] The corrected field

`pyart.correct.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, optional] Name of the RhoHV uncorrected for noise field.
- snr\_field, zdr\_field, nh\_field, nv\_field** [str, optional] Names of the SNR, ZDR, horizontal channel noise in dBZ and vertical channel noise in dBZ used to correct RhoHV.
- rhohv\_field** [str, optional] 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.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.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.dealias_fourdd(radar, last_radar=None, sonde_profile=None, gate-
                             filter=False, 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, optional] Profile of horizontal winds from a sounding 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.

**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_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 sounding 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.

```
pyart.correct.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 are 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 verifying that an appropriate `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

[1], [2]

`pyart.correct.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.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.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, iso0_field=None, refl_field=None, temp_ref='temperature')`

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, iso0\_field, rhohv\_field, refl\_field** [str] Field names within the radar object which represent the temperature, the height over the iso0, co-polar correlation and reflectivity fields. A value of None will use the default field name as defined in the Py-ART configuration file.
- temp\_ref** [str] the field use as reference for temperature. Can be either temperature or height\_over\_iso0

#### Returns

- rhohv\_rain\_dict** [dict] The estimated RhoHV in rain for each sweep and metadata

`pyart.correct.est_zdr_precip(radar, ind_rmin=10, ind_rmax=500, zmin=20.0, zmax=22.0, rhohvmin=0.97, phidpmax=10.0, elmax=None, thickness=700.0, doc=None, fzl=None, zdr_field=None, rhohv_field=None, phidp_field=None, temp_field=None, iso0_field=None, refl_field=None, temp_ref='temperature')`

Filters out all undesired data to be able to estimate ZDR bias, either in moderate rain or from vertically pointing scans

#### 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 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, iso0\_field** : str Field names within the radar object which represent the differential reflectivity, co-polar correlation, reflectivity, differential phase, temperature and height relative to the iso0 fields. A value of None will use the default field name as defined in the Py-ART configuration file.

**temp\_ref** [str] the field use as reference for temperature. Can be either temperature, height\_over\_iso0, fixed\_fzl or None

#### Returns

**zdr\_prec\_dict** [dict] The ZDR data complying with specifications and metadata

```
pyart.correct.est_zdr_snow(radar, ind_rmin=10, ind_rmax=500, zmin=0.0, zmax=30.0, snr-  
min=10.0, snrmax=50.0, rhohvmin=0.97, kept_values=[2],  
phidpmax=10.0, kdpmax=None, tempmin=None, temp-  
max=None, elmax=None, zdr_field=None, rhohv_field=None,  
phidp_field=None, temp_field=None, snr_field=None, hy-  
dro_field=None, kdp_field=None, refl_field=None)
```

Filters out all undesired data to be able to estimate ZDR bias in snow

#### Parameters

**radar** [Radar] radar object

**ind\_rmin, ind\_rmax** [int] Min and max range index where to look for snow

**zmin, zmax** [float] The minimum and maximum reflectivity to consider the radar bin suitable snow

**snrmin, snrmax** [float] The minimum and maximum SNR to consider the radar bin suitable snow

**rhohvmin** [float] Minimum RhoHV to consider the radar bin suitable snow

**kept\_values** [list of int] The hydrometeor classification values to keep

**phidpmax** [float] Maximum PhiDP to consider the radar bin suitable snow

**kdpmax** [float or None] Maximum KDP. If not none this is the maximum KDP value to consider the radar bin suitable snow

**tempmin, tempmax** [float or None] If not None, the minimum and maximum temperature to consider the radar bin suitable snow

**elmax** [float] Maximum elevation

**zdr\_field, rhohv\_field, refl\_field, phidp\_field, kdp\_field, temp\_field,**

**snr\_field, hydro\_field** [str] Field names within the radar object which represent the differential reflectivity, co-polar correlation, reflectivity, differential phase, specific differential phase, signal to noise ratio, hydrometeor classification and temperature fields. A value of None will use the default field name as defined in the Py-ART configuration file.

### Returns

**zdr\_snow\_dict** [dict] The ZDR data complying with specifications and metadata

`pyart.correct.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, optional] 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.

`pyart.correct.get_mask_fz1` (*radar, fz1=None, doc=None, min\_temp=0.0, max\_h\_iso0=0.0, thickness=None, beamwidth=None, temp\_field=None, iso0\_field=None, temp\_ref='temperature'*)

Constructs a mask to mask data placed thickness m below data at min\_temp and beyond.

### Parameters

**radar** [Radar] The radar object.

**fz1** [float, optional] Freezing layer, gates above this point are not included in the correction.

**doc** [float, optional] Number of gates at the end of each ray to remove from the calculation.

**min\_temp** [float, optional] Minimum temperature below which the data is mask in degrees.

**max\_h\_iso0** [float, optional] Maximum height relative to the iso0 below which the data is mask in meters.

**thickness** [float, optional] Extent of the layer below the first gate where min\_temp is reached that is going to be masked.

**beamwidth** [float, optional] The radar antenna 3 dB beamwidth.

**temp\_field: str, optional** 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.

**iso0\_field: str, optional** The field containing the height over the 0C isotherm. 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.

**temp\_ref** [str, optional] The field use as reference for temperature. Can be either temperature, height\_over\_iso0 or fixed\_fzl.

#### 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.

`pyart.correct.get_sun_hits` (*radar*, *delev\_max*=2.0, *dazim\_max*=2.0, *elmin*=1.0, *rmin*=50000.0, *hmin*=10000.0, *nbins\_min*=20, *attg*=None, *max\_std\_pwr*=1.0, *max\_std\_zdr*=1.5, *sun\_position*='MF', *pwrh\_field*=None, *pwrv\_field*=None, *zdr\_field*=None)

get data from suspected sun hits. The sun hits are detected using the Hildebrand and Sekhon noise estimation

#### 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

**rmin** [float] minimum range from which we can look for noise [m]

**hmin** [float] minimum altitude from which we can look for noise [m]. The actual range min will be the minimum between rmin and the range bin higher than hmin.

**nbins\_min** [int] Minimum number of bins with valid data to consider a ray as potentially sun hit

**attg** [float] gas attenuation coefficient (1-way)

**max\_std\_pwr** [float] Maximum standard deviation of the estimated sun power to consider the sun signal valid [dB]

**max\_std\_zdr** [float] Maximum standard deviation of the estimated sun ZDR to consider the sun signal valid [dB]

**sun\_position** [str] The function to use to compute the sun position. Can be 'MF' or 'pysolar'

**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.get_sun_hits_ivic(radar, delelev_max=2.0, dazim_max=2.0, elmin=1.0,
                               npulses_ray=30, flat_reg_wlen=96, nbins_min=800, iterations=10,
                               attg=None, sun_position='MF', max_std_zdr=1.5,
                               pwrh_field=None, pwrv_field=None, zdr_field=None)
```

get data from suspected sun hits. The sun hits are detected using the Ivic et al. (2003) noise estimator

#### 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
- npulses\_ray** [int] Default number of pulses used in the computation of the ray. If the number of pulses is not in radar.instrument\_parameters this will be used instead
- flat\_reg\_wlen** [int] number of gates considered to find flat regions. The number represents 8 km length with a 83.3 m resolution
- nbins\_min: int** minimum number of gates with noise to consider the retrieval valid
- iterations: int** number of iterations in step 7 of Ivic algorithm
- attg** [float] gas attenuation coefficient (1-way)
- sun\_position** [str] The function to use to compute the sun position. Can be 'MF' or 'pysolar'
- max\_std\_zdr** [float] Maximum standard deviation of the estimated sun ZDR to consider the sun signal valid [dB]
- 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.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.correct.phase_proc_lp(radar, offset, debug=False, self_const=60000.0, low_z=10.0,
                             high_z=53.0, min_phidp=0.01, min_ncp=0.5, min_rhv=0.8,
                             fzl=4000.0, sys_phase=0.0, override_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, win-
                             dow_len=35, proc=1, coef=0.914)
```

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

**high\_z** [float, optional] High limit for reflectivity. Reflectivity above this value is set to this limit.

**min\_phidp** [float, optional] Minimum Phi differential phase.

**min\_ncp** [float, optional] Minimum normal coherent power.

**min\_rhv** [float, optional] Minimum copolar coefficient.

**fzl** [float, optional] Maximum altitude.

**sys\_phase** [float, optional] System phase in degrees.

**override\_sys\_phase** [bool, optional] True to use *sys\_phase* as the system phase. False will calculate a value automatically.



**nowrap** [int or None, optional] Gate number to begin phase unwrapping. None will unwrap all phases.

**really\_verbose** [bool, optional] True to print LPX messaging. False to suppress.

**LP\_solver** ['pyglpk' or 'cvxopt', 'cylp', or 'cylp\_mp', optional] Module to use to solve LP problem. Default is 'cylp'.

**refl\_field, ncp\_field, rhv\_field, phidp\_field, kdp\_field** [str, optional] Name of field in radar which contains the horizontal 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, optional] 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, optional] Length of Sobel window applied to PhiDP field when prior to calculating KDP.

**proc** [int, optional] Number of worker processes, only used when *LP\_solver* is 'cylp\_mp'.

**coef** [float, optional] Exponent linking Z to KDP in self consistency.  $kdp=(10^{**}(0.1z))^{*coef}$

#### 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_lp_gf(radar, gatefilter=None, debug=False, self_const=60000.0,
                               low_z=10.0, high_z=53.0, min_phidp=0.01, fzl=4000.0,
                               system_phase=None, nowrap=None, really_verbose=False,
                               LP_solver='cylp', refl_field=None, phidp_field=None,
                               kdp_field=None, unf_field=None, window_len=35, proc=1,
                               coef=0.914, ncpts=None, first_gate_sysp=None, offset=0.0,
                               doc=0)
```

Phase process using a LP method [1] using Py-ART's Gatefilter.

#### Parameters

**radar** [Radar] Input radar.

**gatefilter** [Gatefilter, optional] Py-ART gatefilter object indicating where processing should be carried out

**debug** [bool, optional] True to print debugging information.

**self\_const** [float, optional] Self consistency factor.

**low\_z** [float, optional] Low limit for reflectivity. Reflectivity below this value is set to this limit.

**high\_z** [float, optional] High limit for reflectivity. Reflectivity above this value is set to this limit.

**fzl** [float, optional] Maximum altitude.

**system\_phase** [float, optional] System phase in degrees.

**nowrap** [int or None, optional] Gate number to begin phase unwrapping. None will unwrap all phases.

**really\_verbose** [bool, optional] True to print LPX messaging. False to suppress.

**LP\_solver** ['pyglpk' or 'cvxopt', 'cylp', or 'cylp\_mp', optional] Module to use to solve LP problem. Default is 'cylp'.

**refl\_field, ncp\_field, rhv\_field, phidp\_field, kdp\_field** [str, optional] Name of field in radar which contains the horizontal 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, optional] 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, optional] Length of Sobel window applied to PhiDP field when prior to calculating KDP.

**proc** [int, optional] Number of worker processes, only used when *LP\_solver* is 'cylp\_mp'.

**coef** [float, optional] Exponent linking Z to KDP in self consistency.  $kdp=(10^{**}(0.1z))^{*coef}$

**ncpts** [int, optional] Minimum number of points in a ray. Regions within a ray smaller than this or beginning before this gate number are excluded from unfolding.

**offset** [float, optional] Reflectivity offset to add in dBz.

**doc** [int, optional] Number of gates to “doc” off the end of a ray.

#### 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.ptoa_to_sf` (*ptoa, pulse\_width, wavelen, antenna\_gain, coeff\_band=1.2*)

Converts the sun power at the top of the atmosphere (in dBm) into solar flux.

#### Parameters

**ptoa** [float] sun power at the top of the atmosphere. It already takes into account the correction for antenna polarization

**pulse\_width** [float] pulse width [s]

**wavelen** [float] radar wavelength [m]

**antenna\_gain** [float] the antenna gain [dB]

**coeff\_band** [float] multiplicative coefficient applied to the inverse of the pulse width to get the effective bandwidth

#### Returns

**s0** [float] solar flux [10e-22 W/(m<sup>2</sup> Hz)]

## References

Altube P., J. Bech, O. Argemi, T. Rigo, 2015: Quality Control of Antenna Alignment and Receiver Calibration Using the Sun: Adaptation to Midrange Weather Radar Observations at Low Elevation Angles

`pyart.correct.scanning_losses` (*angle\_step*, *beamwidth*)

Given the antenna beam width and the integration angle, compute the losses due to the fact that the sun is not a point target and the antenna is scanning

### Parameters

**angle\_step** [float] integration angle [deg]

**beamwidth** [float] 3 dB-beamwidth [deg]

### Returns

**la** [float] The losses due to the scanning of the antenna [dB positive]

## References

Altube P., J. Bech, O. Argemi, T. Rigo, 2015: Quality Control of Antenna Alignment and Receiver Calibration Using the Sun: Adaptation to Midrange Weather Radar Observations at Low Elevation Angles

`pyart.correct.selfconsistency_bias` (*radar*, *zdr\_kdpzh\_dict*, *min\_rhohv*=0.92, *filter\_rain*=True, *max\_phidp*=20.0, *smooth\_wind\_len*=5, *doc*=None, *fzl*=None, *thickness*=700.0, *min\_rcons*=20, *dphidp\_min*=2, *dphidp\_max*=16, *parametrization*='None', *refl\_field*=None, *phidp\_field*=None, *zdr\_field*=None, *temp\_field*=None, *iso0\_field*=None, *hydro\_field*=None, *rhohv\_field*=None, *temp\_ref*='temperature', *check\_wet\_radome*=True, *wet\_radome\_refl*=25.0, *wet\_radome\_len\_min*=4, *wet\_radome\_len\_max*=8, *wet\_radome\_ngates\_min*=180, *valid\_gates\_only*=False, *keep\_points*=False, *kdp\_wind\_len*=12)

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

**filter\_rain** [bool] If True the hydrometeor classification is going to be used to filter out all gates that are not rain

**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]

**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

**parametrization** [str] The type of parametrization for the self-consistency curves. Can be 'None', 'Gourley', 'Wolfensberger', 'Louf', 'Gorgucci' or 'Vaccarono'. 'None' will use tables contained in `zdr_kdpzh_dict`.

**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, iso0\_field, hydro\_field, rhohv\_field** [str] Field names within the radar object which represent the temperature, the height relative to the iso0, the hydrometeor classification and the 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.

**kdp\_sim\_field, phidp\_sim\_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.

**temp\_ref** [str] the field use as reference for temperature. Can be either temperature, height\_over\_iso0 or fixed\_fz1

**check\_wet\_radome** [Bool] if True the average reflectivity of the closest gates to the radar is going to be checked to find out whether there is rain over the radome. If there is rain no bias will be computed

**wet\_radome\_refl** [Float] Average reflectivity of the gates close to the radar to consider the radome as wet

**wet\_radome\_len\_min, wet\_radome\_len\_max** [int] Min and max gate indices of the disk around the radome used to decide whether the radome is wet

**wet\_radome\_ngates\_min** [int] Minimum number of valid gates to consider that the radome is wet

**valid\_gates\_only** [Bool] If True the reflectivity bias obtained for each valid ray is going to be assigned only to gates of the segment used. That will give more weight to longer segments when computing the total bias.

**keep\_points** [Bool] If True the ZDR, ZH and KDP of the gates used in the self-consistency algorithm are going to be stored for further analysis

**kdp\_wind\_len** [int] The length of the window used to compute KDP with the single window least square method

## Returns

**refl\_bias\_dict** [dict] the bias at each ray field and metadata

**selfconsistency\_dict** [dict] If keep\_pointst set, a dictionary containing the measured valid values of ZDR, Zh and KDP. None otherwise

```
pyart.correct.selfconsistency_bias2(radar, zdr_kdpzh_dict, min_rhohv=0.92, min_zdr=0.2,
                                     filter_rain=True, max_phidp=20.0, smooth_wind_len=5,
                                     doc=None, fzl=None, thickness=700.0, min_rcons=20,
                                     parametrization='None', refl_field=None,
                                     phidp_field=None, zdr_field=None, temp_field=None,
                                     iso0_field=None, hydro_field=None, rhohv_field=None,
                                     kdp_field=None, temp_ref='temperature',
                                     check_wet_radome=True, wet_radome_refl=25.0,
                                     wet_radome_len_min=4, wet_radome_len_max=8,
                                     wet_radome_ngates_min=180, keep_points=False,
                                     bias_per_gate=False)
```

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
- min\_zdr** [float] minimum ZDR value to consider the data valid
- filter\_rain** [bool] If True the hydrometeor classification is going to be used to filter out all gates that are not rain
- 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]
- min\_rcons** [int] minimum number of consecutive gates to consider a valid segment of PhiDP
- parametrization** [str] The type of parametrization for the self-consistency curves. Can be 'None', 'Gourley', 'Wolfensberger', 'Louf', 'Gorgucci' or 'Vaccaroni'. 'None' will use tables contained in zdr\_kdpzh\_dict.
- refl\_field, kdp\_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, iso0\_field, hydro\_field, rhohv\_field, phidp\_field** [str] Field names within the radar object which represent the temperature, the height relative to the iso0, the hydrometeor classification and the 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.
- kdpssim\_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.
- temp\_ref** [str] the field use as reference for temperature. Can be either temperature, height\_over\_iso0 or fixed\_fzl
- check\_wet\_radome** [Bool] if True the average reflectivity of the closest gates to the radar is going to be check to find out whether there is rain over the radome. If there is rain no bias will be computed

**wet\_radome\_refl** [Float] Average reflectivity of the gates close to the radar to consider the radome as wet

**wet\_radome\_len\_min, wet\_radome\_len\_max** [int] Mim and max gate indices of the disk around the radome used to decide whether the radome is wet

**wet\_radome\_ngates\_min** [int] Minimum number of valid gates to consider that the radome is wet

**keep\_points** [Bool] If True the ZDR, ZH and KDP of the gates used in the self- consistency algorithm are going to be stored for further analysis

**bias\_per\_gate** [Bool] If True the bias per gate will be computed

#### Returns

**kdp\_data\_dict** [dict] A dictionary containing valid observed and estimated using self- consistency values of KDP

**refl\_bias\_dict** [dict] If bias\_per\_gate is set, the bias at each gate field and metadata. None otherwise

**selfconsistency\_dict** [dict] If keep\_poinst set, a dictionary containing the measured valid values of ZDR, Zh and KDP. None otherwise

```
pyart.correct.selfconsistency_kdp_phidp(radar,      zdr_kdpzh_dict,      min_rhohv=0.92,
                                         filter_rain=True,              max_phidp=20.0,
                                         smooth_wind_len=5,             doc=None,      fzl=None,
                                         thickness=700.0,                parametrization='None',
                                         refl_field=None,                 phidp_field=None,
                                         zdr_field=None,                  temp_field=None,
                                         iso0_field=None,                 hydro_field=None,
                                         rhohv_field=None,                kdpsim_field=None, phidp-
                                         sim_field=None, temp_ref='temperature')
```

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

**filter\_rain** [bool] If True the hydrometeor classification is going to be used to filter out all gates that are not rain

**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]

**parametrization** [str] The type of parametrization for the self-consistency curves. Can be 'None', 'Gourley', 'Wolfensberger', 'Louf', 'Gorgucci' or 'Vaccarono'. 'None' will use tables contained in zdr\_kdpzh\_dict.

**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, iso0\_field, hydro\_field, rhohv\_field** [str] Field names within the radar object which represent the temperature, the height relative to the iso0, the hydrometeor classification and the 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.

**temp\_ref** [str] the field use as reference for temperature. Can be either temperature, height\_over\_iso0 or fixed\_fzl

#### Returns

**kdp\_sim\_dict, phidp\_sim\_dict** [dict] the KDP and PhiDP estimated fields and metadata

`pyart.correct.smooth_masked(raw_data, wind_len=11, min_valid=6, wind_type='median')`  
smoothes the data using a rolling 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.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.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.solar_flux_lookup(solar_flux, wavelen)
```

Given the observed solar flux at 10.7 cm wavelength, returns the solar flux at the given radar wavelength

#### Parameters

**solar\_flux** [float array] the solar fluxes measured at 10.7 cm wavelength [10e-22 W/(m2 Hz)]

**wavelen** [float] radar wavelength [m]

#### Returns

**s0** [float] the radar flux at the radar wavelength [10e-22 W/(m2 Hz)]

## References

Altube P., J. Bech, O. Argemi, T. Rigo, 2015: Quality Control of Antenna Alignment and Receiver Calibration Using the Sun: Adaptation to Midrange Weather Radar Observations at Low Elevation Angles

```
pyart.correct.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.sun_position_pysolar(dt, lat, lon, elevation=0.0)`  
 obtains the sun position in antenna 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

**Returns**

**el, az** [float] elevation and azimuth angles of the sun respect to the sensor in degrees

`pyart.correct.sun_power(solar_flux, pulse_width, wavelen, antenna_gain, angle_step, beamwidth, coeff_band=1.2)`  
 computes the theoretical sun power detected at the antenna [dBm] as it would be without atmospheric attenuation (sun power at top of the atmosphere) for a given solar flux and radar characteristics

**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]

**coeff\_band** [float] multiplicative coefficient applied to the inverse of the pulse width to get the effective bandwidth

**Returns**

**pwr\_det** [float array] the detected power

**References**

Altube P., J. Bech, O. Argemi, T. Rigo, 2015: Quality Control of Antenna Alignment and Receiver Calibration Using the Sun: Adaptation to Midrange Weather Radar Observations at Low Elevation Angles

`pyart.correct.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

**par** [float array] and array with the 5 parameters of the Gaussian fit

---

## RADAR RETRIEVALS (PYART.RETRIEVE)

Radar retrievals.

### 7.1 Radar retrievals

<code>kdp_maesaka(radar[, gatefilter, method, ...])</code>	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.
<code>kdp_schneebeli(radar[, gatefilter, ...])</code>	Estimates Kdp with the Kalman filter method by Schneebeli and al.
<code>kdp_vulpiani(radar[, gatefilter, ...])</code>	Estimates Kdp with the Vulpiani method for a 2D array of psidp measurements with the first dimension being the distance from radar and the second dimension being the angles (azimuths for PPI, elev for RHI).The input psidp is assumed to be pre-filtered (for ex.
<code>kdp_leastsquare_single_window(radar[, ...])</code>	Compute the specific differential phase (KDP) from differential phase data using a piecewise least square method.
<code>kdp_leastsquare_double_window(radar[, ...])</code>	Compute the specific differential phase (KDP) from differential phase data using a piecewise least square method.
<code>calculate_snr_from_reflectivity(radar[, ...])</code>	Calculate the signal to noise ratio, in dB, from the reflectivity field.
<code>calculate_velocity_texture(radar[, ...])</code>	Derive the texture of the velocity field.
<code>compute_ccor(radar[, filt_field, ...])</code>	Computes the clutter correction ratio (CCOR), i.e.
<code>compute_snr(radar[, refl_field, ...])</code>	Computes SNR from a reflectivity field and the noise in dBZ.
<code>compute_l(radar[, rhohv_field, l_field])</code>	Computes Rhohv in logarithmic scale according to $L = -\log_{10}(1 - \text{RhoHV})$ .
<code>compute_cdr(radar[, rhohv_field, zdr_field, ...])</code>	Computes the Circular Depolarization Ratio.
<code>compute_noisedBZ(nrays, noisedBZ_val, ..., ...)</code>	Computes noise in dBZ from reference noise value.
<code>compute_radial_noise_hs(radar[, ind_rmin, ...])</code>	Computes radial noise in dBm from signal power using the algorithm from Hildebrand and Sekhon 1974
<code>compute_radial_noise_ivic(radar[, ...])</code>	Computes radial noise in dBm from signal power using the algorithm described in Ivic et al.
<code>compute_signal_power(radar[, lmf, attg, ...])</code>	Computes received signal power OUTSIDE THE RADOME in dBm from a reflectivity field.

Continued on next page

Table 1 – continued from previous page

<code>compute_rcs(radar[, kw2, pulse_width, ...])</code>	Computes the radar cross-section (assuming a point target) from radar reflectivity.
<code>compute_rcs_from_pr(radar[, lmf, attg, ...])</code>	Computes the radar cross-section (assuming a point target) from radar reflectivity by first computing the received power and then the RCS from it.
<code>compute_vol_refl(radar[, kw, freq, ...])</code>	Computes the volumetric reflectivity from the effective reflectivity factor
<code>compute_bird_density(radar[, sigma_bird, ...])</code>	Computes the bird density from the volumetric reflectivity
<code>fetch_radar_time_profile(sonde_dset, radar)</code>	Extract the correct profile from a interpolated sonde.
<code>map_profile_to_gates(profile, heights, radar)</code>	Given a profile of a variable map it to the gates of radar assuming 4/3Re.
<code>steiner_conv_strat(grid[, dx, dy, intense, ...])</code>	Partition reflectivity into convective-stratiform using the Steiner et al.
<code>hydroclass_semisupervised(radar[, ...])</code>	Classifies precipitation echoes following the approach by Besic et al (2016).
<code>get_freq_band(freq)</code>	Returns the frequency band name (S, C, X, ...).
<code>texture_of_complex_phase(radar[, ...])</code>	Calculate the texture of the differential phase field.
<code>grid_displacement_pc(grid1, grid2, field, level)</code>	Calculate the grid displacement using phase correlation.
<code>grid_shift(grid, advection[, trim_edges, ...])</code>	Shift a grid by a certain number of pixels.
<code>est_rain_rate_zpoly(radar[, refl_field, ...])</code>	Estimates rainfall rate from reflectivity using a polynomial Z-R relation developed at McGill University.
<code>est_rain_rate_z(radar[, alpha, beta, ...])</code>	Estimates rainfall rate from reflectivity using a power law.
<code>est_rain_rate_kdp(radar[, alpha, beta, ...])</code>	Estimates rainfall rate from kdp using alpha power law.
<code>est_rain_rate_a(radar[, alpha, beta, ...])</code>	Estimates rainfall rate from specific attenuation using alpha power law.
<code>est_rain_rate_zkdp(radar[, alphaz, betaz, ...])</code>	Estimates rainfall rate from a blending of power law r-kdp and r-z relations.
<code>est_rain_rate_za(radar[, alphaz, betaz, ...])</code>	Estimates rainfall rate from a blending of power law r-alpha and r-z relations.
<code>est_rain_rate_hydro(radar[, alphazr, ...])</code>	Estimates rainfall rate using different relations between R and the polarimetric variables depending on the hydrometeor type.
<code>est_wind_vel(radar[, vert_proj, vel_field, ...])</code>	Estimates wind velocity.
<code>est_vertical_windshear(radar[, az_tol, ...])</code>	Estimates wind shear.
<code>atmospheric_gas_att(freq, elev, rng)</code>	Computes the one-way atmospheric gas attenuation [dB] according to the empirical formula in Doviak and Zrnica (1993) pp 44.
<code>get_coeff_attg(freq)</code>	get the 1-way gas attenuation for a particular frequency
<code>est_wind_profile(radar[, npoints_min, ...])</code>	Estimates the vertical wind profile using VAD techniques
<code>detect_ml(radar[, gatefilter, fill_value, ...])</code>	Detects the melting layer (ML) using the reflectivity and copolar correlation coefficient.
<code>melting_layer_giangrande(radar[, nVol, ...])</code>	Detects the melting layer following the approach by Giangrande et al (2008)
<code>melting_layer_hydroclass(radar[, ...])</code>	Using the results of the hydrometeor classification by Besic et al.
<code>_get_res_vol_sides(radar)</code>	Computes the height of the lower left and upper right points of the range resolution volume.

Continued on next page

Table 1 – continued from previous page

<code>velocity_azimuth_display(radar[, vel_field, ...])</code>	Velocity azimuth display.
<code>quasi_vertical_profile(radar[, ...])</code>	Quasi Vertical Profile.
<code>compute_qvp(radar, field_names[, ref_time, ...])</code>	Computes quasi vertical profiles.
<code>compute_rqvp(radar, field_names[, ref_time, ...])</code>	Computes range-defined quasi vertical profiles.
<code>compute_evp(radar, field_names, lon, lat[, ...])</code>	Computes enhanced vertical profiles.
<code>compute_svp(radar, field_names, lon, lat, angle)</code>	Computes slanted vertical profiles.
<code>compute_vp(radar, field_names, lon, lat[, ...])</code>	Computes vertical profiles.
<code>compute_ts_along_coord(radar, field_name[, ...])</code>	Computes time series along a particular antenna coordinate, i.e.
<code>compute_iq(spectra, fields_in_list, ...[, ...])</code>	Computes the IQ data from the spectra through an inverse Fourier transform
<code>compute_spectral_power(spectra[, units, ...])</code>	Computes the spectral power from the complex spectra in ADU.
<code>compute_spectral_noise(spectra[, units, ...])</code>	Computes the spectral noise power from the complex spectra in ADU.
<code>compute_spectral_phase(spectra[, signal_field])</code>	Computes the spectral phase from the complex spectra in ADU
<code>compute_spectral_reflectivity(spectra[, ...])</code>	Computes the spectral reflectivity from the complex spectra in ADU or from the signal power in ADU.
<code>compute_spectral_differential_reflectivity(spectra[, ...])</code>	Computes the spectral differential reflectivity from the complex spectras or the power in ADU
<code>compute_spectral_differential_phase(spectra[, ...])</code>	Computes the spectral differential reflectivity from the complex spectras in ADU or sRhoHV
<code>compute_spectral_rhohv(spectra[, ...])</code>	Computes the spectral RhoHV from the complex spectras in ADU
<code>compute_pol_variables(spectra, fields_list)</code>	Computes the polarimetric variables from the complex spectra in ADU or the spectral powers and spectral RhoHV
<code>compute_noise_power(spectra[, units, navg, ...])</code>	Computes the noise power from the complex spectra in ADU.
<code>compute_reflectivity(spectra[, sdBZ_field])</code>	Computes the reflectivity from the spectral reflectivity
<code>compute_differential_reflectivity(spectra[, ...])</code>	Computes the differential reflectivity from the horizontal and vertical spectral reflectivity
<code>compute_differential_phase(spectra[, ...])</code>	Computes the differential phase from the spectral differential phase and the spectral reflectivity
<code>compute_rhohv(spectra[, use_rhohv, ...])</code>	Computes RhoHV from the horizontal and vertical spectral reflectivity or from sRhoHV and the spectral powers
<code>compute_Doppler_velocity(spectra[, sdBZ_field])</code>	Computes the Doppler velocity from the spectral reflectivity
<code>compute_Doppler_width(spectra[, sdBZ_field])</code>	Computes the Doppler width from the spectral reflectivity
<code>compute_reflectivity_iq(radar[, ...])</code>	Computes the reflectivity from the IQ signal data
<code>compute_st1_iq(radar[, signal_field])</code>	Computes the statistical test one lag fluctuation from the horizontal or vertical channel IQ data
<code>compute_st2_iq(radar[, signal_field])</code>	Computes the statistical test two lag fluctuation from the horizontal or vertical channel IQ data
<code>compute_wbn_iq(radar[, signal_field])</code>	Computes the wide band noise from the horizontal or vertical channel IQ data

Continued on next page

Table 1 – continued from previous page

<code>compute_differential_reflectivity_iq(radar)</code>	Computes the differential reflectivity from the horizontal and vertical IQ data
<code>compute_mean_phase_iq(radar[, signal_field])</code>	Computes the differential phase from the horizontal or vertical channel IQ data
<code>compute_differential_phase_iq(radar[, ...])</code>	Computes the differential phase from the horizontal and vertical channels IQ data
<code>compute_rho_hv_iq(radar[, subtract_noise, ...])</code>	Computes RhoHV from the horizontal and vertical channels IQ data
<code>compute_Doppler_velocity_iq(radar[, ...])</code>	Computes the Doppler velocity from the IQ data
<code>compute_Doppler_width_iq(radar[, ...])</code>	Computes the Doppler width from the IQ data
<code>compute_pol_variables_iq(radar, fields_list)</code>	Computes the polarimetric variables from the IQ signals in ADU
<code>compute_spectra(radar, fields_in_list, ...)</code>	Computes the spectra from IQ data through a Fourier transform

`pyart.retrieve.atmospheric_gas_att(freq, elev, rng)`

Computes the one-way atmospheric gas attenuation [dB] according to the empirical formula in Doviak and Zrnic (1993) pp 44. This formula is valid for elev < 10 deg and rng < 200 km so values above these will be saturated to 10 deg and 200 km respectively

#### Parameters

**freq** [float] radar frequency [Hz]

**elev** [float or array of floats] elevation angle [deg]

**rng** [float or array of floats. If array must have the same size as elev] range [km]

#### Returns

**latm** [float or array of floats] 1-way gas attenuation [dB]

`pyart.retrieve.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.calculate_velocity_texture(radar, vel_field=None, wind_size=4, nyq=None, check_nyq_uniform=True)`

Derive the texture of the velocity field.

#### Parameters

**radar: Radar** Radar object from which velocity texture field will be made.

**vel\_field** [str, optional] Name of the velocity field. A value of None will force Py-ART to automatically determine the name of the velocity field.

**wind\_size** [int, optional] The size of the window to calculate texture from. The window is defined to be a square of size wind\_size by wind\_size.

**nyq** [float, optional] The nyquist velocity of the radar. A value of None will force Py-ART to try and determine this automatically.

**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 nyq parameter is not None.

#### Returns

**vel\_dict**: dict A dictionary containing the field entries for the radial velocity texture.

`pyart.retrieve.compute_Doppler_velocity(spectra, sdbZ_field=None)`

Computes the Doppler velocity from the spectral reflectivity

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields

**sdbZ\_field** [str] Name of the field that contains the spectral reflectivity. None will use the default field name in the Py-ART configuration file.

#### Returns

**vel\_dict** [field dictionary] Field dictionary containing the Doppler velocity

`pyart.retrieve.compute_Doppler_velocity_iq(radar, signal_field=None, direction='negative_away')`

Computes the Doppler velocity from the IQ data

#### Parameters

**radar** [IQ radar object] Object containing the required fields

**signal\_field** [str] Name of the field in the radar which contains the signal. None will use the default field name in the Py-ART configuration file.

**direction** [str] The convention used in the Doppler mean field. Can be negative\_away or negative\_towards

#### Returns

**vel\_dict** [field dictionary] Field dictionary containing the Doppler velocity

`pyart.retrieve.compute_Doppler_width(spectra, sdbZ_field=None)`

Computes the Doppler width from the spectral reflectivity

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields

**sdbZ\_field** [str] Name of the field that contains the spectral reflectivity. None will use the default field name in the Py-ART configuration file.

#### Returns

**width\_dict** [field dictionary] Field dictionary containing the Doppler spectrum width

`pyart.retrieve.compute_Doppler_width_iq(radar, subtract_noise=True, signal_field=None, noise_field=None, lag=1)`

Computes the Doppler width from the IQ data

#### Parameters

**radar** [Radar radar object] Object containing the required fields

**subtract\_noise** [Bool] If True noise will be subtracted from the signals

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

**signal\_field, noise\_field** [str] Name of the field in the radar which contains the signal and noise.  
None will use the default field name in the Py-ART configuration file.

#### Returns

**width\_dict** [field dictionary] Field dictionary containing the Doppler spectrum width

`pyart.retrieve.compute_bird_density(radar, sigma_bird=11, vol_refl_field=None, bird_density_field=None)`

Computes the bird density from the volumetric reflectivity

#### Parameters

**radar** [Radar] radar object

**sigma\_bird** [float] Estimated bird radar cross-section

**vol\_refl\_field** [str] name of the volumetric reflectivity used for the calculations

**bird\_density\_field** [str] name of the bird density field

#### Returns

**bird\_density\_dict** [dict] bird density data and metadata [birds/km<sup>3</sup>]

`pyart.retrieve.compute_ccor(radar, filt_field=None, unfilt_field=None, ccor_field=None)`

Computes the clutter correction ratio (CCOR), i.e. the ratio between the signal without Doppler filtering and the signal with Doppler filtering

#### Parameters

**radar** [Radar] Radar object

**filt\_field, unfilt\_field** [str] Name of Doppler filtered and unfiltered fields

**ccor\_field** [str] Name of the CCOR field

#### Returns

**ccor\_dict** [field dictionary] Field dictionary containing the CCOR

`pyart.retrieve.compute_cdr(radar, rhohv_field=None, zdr_field=None, cdr_field=None)`

Computes the Circular Depolarization Ratio.

#### Parameters

**radar** [Radar] Radar object.

**rhohv\_field** [str, optional] Name of the RhoHV field.

**zdr\_field** [str, optional] Name of the ZDR field.

**cdr\_field** [str, optional] Name of the CDR field.

#### Returns

**cdr** [dict] CDR field.

`pyart.retrieve.compute_differential_phase(spectra, sdBZ_field=None, sPhiDP_field=None)`

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

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields



**sdBZ\_field, sPhiDP\_field** [str] Name of the fields that contain the spectral reflectivity and the spectral differential phase. None will use the default field name in the Py-ART configuration file.

#### Returns

**PhiDP\_dict** [field dictionary] Field dictionary containing the differential phase

```
pyart.retrieve.compute_differential_phase_iq(radar, phase_offset=0.0, signal_h_field=None, signal_v_field=None)
```

Computes the differential phase from the horizontal and vertical channels IQ data

#### Parameters

**radar** [IQ radar object] Object containing the required fields

**phase\_offset** [float] system phase offset to add

**signal\_h\_field, signal\_v\_field** [str] Name of the fields that contain the H and V IQ data. None will use the default field name in the Py-ART configuration file.

#### Returns

**phidp\_dict** [field dictionary] Field dictionary containing the differential phase

```
pyart.retrieve.compute_differential_reflectivity(spectra, sdBZ_field=None, sdBZv_field=None)
```

Computes the differential reflectivity from the horizontal and vertical spectral reflectivity

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields

**sdBZ\_field, sdBZv\_field** [str] Name of the fields that contain the spectral reflectivity. None will use the default field name in the Py-ART configuration file.

#### Returns

**ZDR\_dict** [field dictionary] Field dictionary containing the differential reflectivity

```
pyart.retrieve.compute_differential_reflectivity_iq(radar, subtract_noise=False, lag=0, signal_h_field=None, signal_v_field=None, noise_h_field=None, noise_v_field=None)
```

Computes the differential reflectivity from the horizontal and vertical IQ data

#### Parameters

**radar** [IQ radar object] Object containing the required fields

**subtract\_noise** [Bool] If true the noise is subtracted from the power

**lag** [int] Time lag used to compute the differential reflectivity

**signal\_h\_field, signal\_v\_field, noise\_h\_field, noise\_v\_field** [str] Name of the signal and noise fields. None will use the default field name in the Py-ART configuration file.

#### Returns

**zdr\_dict** [field dictionary] Field dictionary containing the differential reflectivity

```
pyart.retrieve.compute_evdp(radar, field_names, lon, lat, ref_time=None, latlon_tol=0.0005, delta_rng=15000.0, delta_azi=10, hmax=10000.0, hres=250.0, avg_type='mean', nvalid_min=1, interp_kind='none', qvp=None)
```

Computes enhanced vertical profiles.

**Parameters**

- radar** [Radar] Radar object used.
- field\_names** [list of str] list of field names to add to the QVP
- lat, lon** [float] latitude and longitude of the point of interest [deg]
- ref\_time** [datetime object] reference time for current radar volume
- latlon\_tol** [float] tolerance in latitude and longitude in deg.
- delta\_rng, delta\_az** [float] maximum range distance [m] and azimuth distance [degree] from the central point of the evp containing data to average.
- hmax** [float] The maximum height to plot [m].
- hres** [float] The height resolution [m].
- avg\_type** [str] The type of averaging to perform. Can be either “mean” or “median”
- nvalid\_min** [int] Minimum number of valid points to accept average.
- interp\_kind** [str] type of interpolation when projecting to vertical grid: ‘none’, or ‘nearest’, etc. ‘none’ will select from all data points within the regular grid height bin the closest to the center of the bin. ‘nearest’ will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation
- qvp** [QVP object or None] If it is not None this is the QVP object where to store the data from the current time step. Otherwise a new QVP object will be created

**Returns**

- qvp** [qvp object] The computed enhanced vertical profile

`pyart.retrieve.compute_iq(spectra, fields_in_list, fields_out_list, window=None)`  
Computes the IQ data from the spectra through an inverse Fourier transform

**Parameters**

- spectra** [Spectra radar object] Object containing the spectra
- fields\_in\_list** [list of str] list of input spectra fields names
- fields\_out\_list** [list of str] list with the output IQ fields names obtained from the input fields
- window** [string, tuple or None] Parameters of the window used to obtain the spectra. The parameters are the ones corresponding to function `scipy.signal.windows.get_window`. If it is not None the inverse will be used to multiply the IQ data obtained by the IFFT

**Returns**

- radar** [IQ radar object] radar object containing the IQ fields

`pyart.retrieve.compute_l(radar, rhohv_field=None, l_field=None)`  
Computes Rhohv in logarithmic scale according to  $L = -\log_{10}(1 - \text{RhoHV})$ .

**Parameters**

- radar** [Radar] Radar object.
- rhohv\_field** [str, optional] Name of the RhoHV field to use.
- l\_field** [str, optional] Name of the L field.

**Returns**

**I** [dict] L field.

`pyart.retrieve.compute_mean_phase_iq(radar, signal_field=None)`

Computes the differential phase from the horizontal or vertical channel IQ data

#### Parameters

**radar** [IQ radar object] Object containing the required fields

**signal\_field** [str] Name of the field that contain the H or V IQ data. None will use the default field name in the Py-ART configuration file.

#### Returns

**mph\_dict** [field dictionary] Field dictionary containing the mean phase

`pyart.retrieve.compute_noise_power(spectra, units='dBADU', navg=1, rmin=0.0, nnoise_min=1, signal_field=None)`

Computes the noise power from the complex spectra in ADU. Requires key dBADU\_to\_dBm\_hh or dBADU\_to\_dBm\_vv in radar\_calibration if the units are to be dBm. The noise is computed using the method described in Hildebrand and Sekhon, 1974.

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields

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

**navg** [int] Number of spectra averaged

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

**nnoise\_min** [int] Minimum number of samples to consider the estimated noise power valid

**signal\_field** [str, optional] Name of the field in radar which contains the signal. None will use the default field name in the Py-ART configuration file.

#### Returns

**noise\_dict** [field dictionary] Field dictionary containing the noise power

## 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.

`pyart.retrieve.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, optional] Name of the noise field.

#### Returns

**noisedBZ** [dict] The noise field.

```
pyart.retrieve.compute_pol_variables(spectra, fields_list, use_pwr=False, subtract_noise=False, smooth_window=None, srhohv_field=None, pwr_h_field=None, pwr_v_field=None, signal_h_field=None, signal_v_field=None, noise_h_field=None, noise_v_field=None)
```

Computes the polarimetric variables from the complex spectra in ADU or the spectral powers and spectral RhoHV

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields

**fields\_list** [list of str] list of fields to compute

**use\_pwr** [Bool] If True the polarimetric variables will be computed from the spectral power and the spectral RhoHV. Otherwise from the complex spectra

**subtract\_noise** [Bool] If True noise will be subtracted from the signals

**smooth\_window** [int or None] Size of the moving Gaussian smoothing window. If none no smoothing will be applied

**srhohv\_field, pwr\_h\_field, pwr\_v\_field, signal\_h\_field, signal\_v\_field, noise\_h\_field, noise\_v\_field** [str] Name of the fields in radar which contains the signal and noise. None will use the default field name in the Py-ART configuration file.

#### Returns

**radar** [radar object] Object containing the computed fields

```
pyart.retrieve.compute_pol_variables_iq(radar, fields_list, subtract_noise=False, lag=0, direction='negative_away', phase_offset=0.0, signal_h_field=None, signal_v_field=None, noise_h_field=None, noise_v_field=None)
```

Computes the polarimetric variables from the IQ signals in ADU

#### Parameters

**radar** [IQ radar object] Object containing the required fields

**fields\_list** [list of str] list of fields to compute

**subtract\_noise** [Bool] If True noise will be subtracted from the signals

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

**direction** [str] The convention used in the Doppler mean field. Can be negative\_away or negative\_towards

**phase\_offset** [float. Dataset keyword] The system differential phase offset to remove

**signal\_h\_field, signal\_v\_field, noise\_h\_field, noise\_v\_field** [str] Name of the fields in radar which contains the signal and noise. None will use the default field name in the Py-ART configuration file.

#### Returns

**radar** [radar object] Object containing the computed fields

```
pyart.retrieve.compute_qvp(radar, field_names, ref_time=None, angle=0.0, ang_tol=1.0, hmax=10000.0, hres=50.0, avg_type='mean', nvalid_min=30, interp_kind='none', qvp=None)
```

Computes quasi vertical profiles.

**Parameters**

- radar** [Radar] Radar object used.
- field\_names** [list of str] list of field names to add to the QVP
- ref\_time** [datetime object] reference time for current radar volume
- angle** [int or float] If the radar object contains a PPI volume, the sweep number to use, if it contains an RHI volume the elevation angle.
- ang\_tol** [float] If the radar object contains an RHI volume, the tolerance in the elevation angle for the conversion into PPI
- hmax** [float] The maximum height to plot [m].
- hres** [float] The height resolution [m].
- avg\_type** [str] The type of averaging to perform. Can be either “mean” or “median”
- nvalid\_min** [int] Minimum number of valid points to accept average.
- interp\_kind** [str] type of interpolation when projecting to vertical grid: ‘none’, or ‘nearest’, etc. ‘none’ will select from all data points within the regular grid height bin the closest to the center of the bin. ‘nearest’ will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation
- qvp** [QVP object or None] If it is not None this is the QVP object where to store the data from the current time step. Otherwise a new QVP object will be created

**Returns**

- qvp** [qvp object] The computed QVP object

```
pyart.retrieve.compute_radial_noise_hs (radar,          ind_rmin=0,          nbins_min=1,
                                       max_std_pwr=2.0,      pwr_field=None,
                                       noise_field=None, get_noise_pos=False)
```

Computes radial noise in dBm from signal power using the algorithm from Hildebrand and Sekhon 1974

**Parameters**

- radar: radar object** radar object containing the signal power in dBm
- ind\_rmin: int** index of the gate nearest to the radar where start looking for noisy gates
- nbins\_min: int** min number of noisy gates to consider the estimation valid
- max\_std\_pwr: float** max standard deviation of the noise power to consider the noise valid
- pwr\_field: str** Name of the input signal power field
- noise\_field: str** name of the noise field to use
- get\_noise\_pos** [bool] If true an additional field with gates containing noise according to the algorithm is produced

**Returns**

- noise\_dict** [dict] the noise field in dBm
- noise\_pos\_dict** [dict or None] a dictionary containing a field where the gates with noise are set to 2 and those without are set to 1 (0 reserved)

## 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.

```
pyart.retrieve.compute_radial_noise_ivic(radar,    npulses_ray=30,    flat_reg_wlen=96,
                                         ngates_min=800,          iterations=10,
                                         pwr_field=None,          noise_field=None,
                                         get_noise_pos=False)
```

Computes radial noise in dBm from signal power using the algorithm described in Ivic et al. 2013

### Parameters

**radar:** **radar object** radar object containing the signal power in dBm

**npulses\_ray** [int] Default number of pulses used in the computation of the ray. If the number of pulses is not in radar.instrument\_parameters this will be used instead

**flat\_reg\_wlen** [int] number of gates considered to find flat regions. The number represents 8 km length with a 83.3 m resolution

**ngates\_min:** **int** minimum number of gates with noise to consider the retrieval valid

**iterations:** **int** number of iterations in step 7

**pwr\_field:** **str** Name of the input signal power field

**noise\_field:** **str** name of the noise field to use

**get\_noise\_pos** [bool] If true an additional field with gates containing noise according to the algorithm is produced

### Returns

**noise\_dict** [dict] the noise field in dBm

**noise\_pos\_dict** [dict] the position of the noisy gates

**get\_noise\_pos** [bool] If true an additional field with gates containing noise according to the algorithm is produced

## References

I.R. Ivic, C. Curtis and S.M. Torres, Radial-Based Noise Power Estimation for Weather Radars. Journal of Atmospheric and Oceanic Technology, 2013, 30, 2737-2753.

```
pyart.retrieve.compute_rcs(radar, kw2=0.93, pulse_width=None, beamwidth=None, freq=None,
                           refl_field=None, rcs_field=None)
```

Computes the radar cross-section (assuming a point target) from radar reflectivity.

### Parameters

**radar** [Radar] radar object

**kw2** [float] water constant

**pulse\_width** [float] pulse width [s]

**beamwidth** [float] beamwidth [degree]

**freq** [float] radar frequency [Hz]. If none it will be obtained from the radar metadata

**refl\_field** [str] name of the reflectivity used for the calculations

**rcs\_field** [str] name of the RCS field

**Returns**

**rsc\_dict** [dict] RCS field and metadata

```
pyart.retrieve.compute_rsc_from_pr(radar, lmf=None, attg=None, radconst=None,
                                   tx_pwr=None, antenna_gain=None, lrx=0.0,
                                   ltx=0.0, lradome=0.0, freq=None, refl_field=None,
                                   rsc_field=None, neglect_gas_att=False)
```

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

**Parameters**

**radar** [Radar] radar object

**lmf** [float] matched filter losses. If None it will be obtained from the attribute `radar_calibration` of the radar object

**attg** [float] 1-way gas attenuation

**radconst** [float] radar constant

**tx\_pwr** [float] radar transmitted power [dBm]

**antenna\_gain** [float] antenna gain [dB]. If None it will be obtain from the `instrument_parameters` attribute of the radar object

**lrx** [float] receiver losses from the antenna feed to the reference point (positive value) [dB]

**lradome** [float] 1-way losses due to the radome (positive value) [dB]

**freq** [float] radar frequency [Hz]. If none it will be obtained from the radar metadata

**refl\_field** [str] name of the reflectivity used for the calculations

**rsc\_field** [str] name of the RCS field

**neglect\_gas\_att** [bool] Whether to neglect or not gas attenuation in the estimation of the RCS

**Returns**

**rsc\_dict** [dict] RCS field and metadata

```
pyart.retrieve.compute_reflectivity(spectra, sdBZ_field=None)
```

Computes the reflectivity from the spectral reflectivity

**Parameters**

**spectra** [Radar spectra object] Object containing the required fields

**sdBZ\_field** [str] Name of the field that contains the spectral reflectivity. None will use the default field name in the Py-ART configuration file.

**Returns**

**dBZ\_dict** [field dictionary] Field dictionary containing the reflectivity

```
pyart.retrieve.compute_reflectivity_iq(radar, subtract_noise=False, signal_field=None,
                                       noise_field=None)
```

Computes the reflectivity from the IQ signal data

**Parameters**

**radar** [IQ radar object] Object containing the required fields

**subtract\_noise** [Bool] If true the noise is subtracted from the power

**signal\_field, noise\_field** [str] Name of the signal and noise fields. None will use the default field name in the Py-ART configuration file.

#### Returns

**dBZ\_dict** [field dictionary] Field dictionary containing the reflectivity

```
pyart.retrieve.compute_rhohv(spectra, use_rhohv=False, subtract_noise=False,
                             srhohv_field=None, pwr_h_field=None, pwr_v_field=None,
                             signal_h_field=None, signal_v_field=None, noise_h_field=None,
                             noise_v_field=None)
```

Computes RhoHV from the horizontal and vertical spectral reflectivity or from sRhoHV and the spectral powers

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields

**use\_rhohv** [Bool] If true the RhoHV will be computed from sRho\_hv. Otherwise it will be computed using the complex spectra

**subtract\_noise** [Bool] If True noise will be subtracted from the signals

**srhohv\_field, pwr\_h\_field, pwr\_v\_field, signal\_h\_field, signal\_v\_field,**

**noise\_h\_field, noise\_v\_field** [str] Name of the fields in radar which contains the signal and noise. None will use the default field name in the Py-ART configuration file.

#### Returns

**RhoHV\_dict** [field dictionary] Field dictionary containing the RhoHV

```
pyart.retrieve.compute_rhohv_iq(radar, subtract_noise=False, lag=0, signal_h_field=None, signal_v_field=None, noise_h_field=None, noise_v_field=None)
```

Computes RhoHV from the horizontal and vertical channels IQ data

#### Parameters

**radar** [IQ radar object] Object containing the required fields

**subtract\_noise** [Bool] If True noise will be subtracted from the signals

**lag** [int] Time lag used in the computation

**signal\_h\_field, signal\_v\_field, noise\_h\_field, noise\_v\_field** [str] Name of the fields in radar which contains the signal and noise. None will use the default field name in the Py-ART configuration file.

#### Returns

**rhohv\_dict** [field dictionary] Field dictionary containing the RhoHV

```
pyart.retrieve.compute_rqvp(radar, field_names, ref_time=None, hmax=10000.0, hres=2.0,
                           avg_type='mean', nvalid_min=30, interp_kind='nearest',
                           rmax=50000.0, weight_power=2.0, qvp=None)
```

Computes range-defined quasi vertical profiles.

#### Parameters

**radar** [Radar] Radar object used.

**field\_names** [list of str] list of field names to add to the QVP

**ref\_time** [datetime object] reference time for current radar volume

**hmax** [float] The maximum height to plot [m].

**hres** [float] The height resolution [m].



**avg\_type** [str] The type of averaging to perform. Can be either “mean” or “median”

**nvalid\_min** [int] Minimum number of valid points to accept average.

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

**rmax** [float] ground range up to which the data is intended for use [m].

**weight\_power** [float] Power  $p$  of the weighting function  $1/abs(grng-(rmax-1))^{**p}$  given to the data outside the desired range. -1 will set the weight to 0.

**qvp** [QVP object or None] If it is not None this is the QVP object where to store the data from the current time step. Otherwise a new QVP object will be created

### Returns

**qvp** [qvp object] The computed range defined quasi vertical profile

`pyart.retrieve.compute_signal_power(radar, lmf=None, attg=None, radconst=None, lrx=0.0, lradome=0.0, refl_field=None, pwr_field=None)`

Computes received signal power OUTSIDE THE RADOME 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

**lrx** [float] receiver losses from the antenna feed to the reference point (positive value) [dB]

**lradome** [float] 1-way losses due to the radome (positive value) [dB]

**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.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** [str, optional] Name of the reflectivity field to use.

**noise\_field** [str, optional] Name of the noise field to use.

**snr\_field** [str, optional] Name of the SNR field.

### Returns

**snr** [dict] The SNR field.

`pyart.retrieve.compute_spectra(radar, fields_in_list, fields_out_list, window=None)`

Computes the spectra from IQ data through a Fourier transform

**Parameters**

- radar** [radar object] Object containing the IQ data
- fields\_in\_list** [list of str] list of input IQ data fields names
- fields\_out\_list** [list of str] list with the output spectra fields names obtained from the input fields
- window** [string, tuple or None] Parameters of the window used to obtain the spectra. The parameters are the ones corresponding to function `scipy.signal.windows.get_window`. If None no window will be used

**Returns**

- spectra** [spectra radar object] radar object containing the spectra fields

```
pyart.retrieve.compute_spectral_differential_phase(spectra, use_rhohv=False,
                                                  srhohv_field=None, signal_h_field=None,
                                                  signal_v_field=None)
```

Computes the spectral differential reflectivity from the complex spectras in ADU or sRhoHV

**Parameters**

- spectra** [Radar spectra object] Object containing the required fields
- use\_rhohv** [Bool] If true sRhoHV is going to be used to compute the differential phase. Otherwise the complex signals are used
- signal\_h\_field, signal\_v\_field** [str] Name of the fields in radar which contains the signal. None will use the default field name in the Py-ART configuration file.

**Returns**

- sPhiDP\_dict** [field dictionary] Field dictionary containing the spectral differential phase

```
pyart.retrieve.compute_spectral_differential_reflectivity(spectra, compute_power=True,
                                                         subtract_noise=False,
                                                         smooth_window=None,
                                                         pwr_h_field=None,
                                                         pwr_v_field=None,
                                                         signal_h_field=None,
                                                         signal_v_field=None,
                                                         noise_h_field=None,
                                                         noise_v_field=None)
```

Computes the spectral differential reflectivity from the complex spectras or the power in ADU

**Parameters**

- spectra** [Radar spectra object] Object containing the required fields
- compute\_power** [Bool] If True the signal power will be computed. Otherwise the field given by the user will be used
- subtract\_noise** [Bool] If True noise will be subtracted from the signals
- smooth\_window** [int or None] Size of the moving Gaussian smoothing window. If none no smoothing will be applied
- pwr\_h\_field, pwr\_v\_field, signal\_h\_field, signal\_v\_field, noise\_h\_field,**
- noise\_v\_field** [str] Name of the fields in radar which contains the signal and noise. None will use the default field name in the Py-ART configuration file.

**Returns**

**sZDR\_dict** [field dictionary] Field dictionary containing the spectral differential reflectivity

```
pyart.retrieve.compute_spectral_noise(spectra, units='dBADU', navg=1, rmin=0.0,
                                     nnoise_min=1, signal_field=None)
```

Computes the spectral noise power from the complex spectra in ADU. Requires key dBADU\_to\_dBm\_hh or dBADU\_to\_dBm\_vv in radar\_calibration if the units are to be dBm. The noise is computed using the method described in Hildebrand and Sekhon, 1974.

**Parameters**

**spectra** [Radar spectra object] Object containing the required fields

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

**navg** [int] Number of spectra averaged

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

**nnoise\_min** [int] Minimum number of samples to consider the estimated noise power valid

**signal\_field** [str, optional] Name of the field in radar which contains the signal. None will use the default field name in the Py-ART configuration file.

**Returns**

**noise\_dict** [field dictionary] Field dictionary containing the spectral noise power

**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.

```
pyart.retrieve.compute_spectral_phase(spectra, signal_field=None)
```

Computes the spectral phase from the complex spectra in ADU

**Parameters**

**spectra** [Radar spectra object] Object containing the required fields

**signal\_field** [str, optional] Name of the field in radar which contains the signal. None will use the default field name in the Py-ART configuration file.

**Returns**

**phase\_dict** [field dictionary] Field dictionary containing the spectral phase

```
pyart.retrieve.compute_spectral_power(spectra, units='dBADU', subtract_noise=False,
                                     smooth_window=None, signal_field=None,
                                     noise_field=None)
```

Computes the spectral power from the complex spectra in ADU. Requires key dBADU\_to\_dBm\_hh or dBADU\_to\_dBm\_vv in radar\_calibration if the units are to be dBm

**Parameters**

**spectra** [Radar spectra object] Object containing the required fields

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

**subtract\_noise** [Bool] If True noise will be subtracted from the signal

**smooth\_window** [int or None] Size of the moving Gaussian smoothing window. If none no smoothing will be applied

**signal\_field, noise\_field** [str, optional] Name of the fields in radar which contains the signal and noise. None will use the default field name in the Py-ART configuration file.

#### Returns

**pwr\_dict** [field dictionary] Field dictionary containing the spectral power

```
pyart.retrieve.compute_spectral_reflectivity(spectra, compute_power=True, subtract_noise=False, smooth_window=None,  
                                             pwr_field=None, signal_field=None,  
                                             noise_field=None)
```

Computes the spectral reflectivity from the complex spectra in ADU or from the signal power in ADU. Requires keys dBADU\_to\_dBm\_hh or dBADU\_to\_dBm\_vv in radar\_calibration if the to be computed

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields

**compute\_power** [Bool] If True the signal power will be computed. Otherwise the field given by the user will be used

**subtract\_noise** [Bool] If True noise will be subtracted from the signal

**smooth\_window** [int or None] Size of the moving Gaussian smoothing window. If none no smoothing will be applied

**pwr\_field, signal\_field, noise\_field** [str, optional] Name of the fields in radar which contains the signal power, complex signal and noise. None will use the default field name in the Py-ART configuration file.

#### Returns

**sdBZ\_dict** [field dictionary] Field dictionary containing the spectral reflectivity

```
pyart.retrieve.compute_spectral_rhohv(spectra, subtract_noise=False, signal_h_field=None,  
                                       signal_v_field=None, noise_h_field=None,  
                                       noise_v_field=None)
```

Computes the spectral RhoHV from the complex spectras in ADU

#### Parameters

**spectra** [Radar spectra object] Object containing the required fields

**subtract\_noise** [Bool] If True noise will be subtracted from the signals

**signal\_h\_field, signal\_v\_field, noise\_h\_field, noise\_v\_field** [str] Name of the fields in radar which contains the signal and noise. None will use the default field name in the Py-ART configuration file.

#### Returns

**sRhoHV\_dict** [field dictionary] Field dictionary containing the spectral RhoHV

```
pyart.retrieve.compute_st1_iq(radar, signal_field=None)
```

Computes the statistical test one lag fluctuation from the horizontal or vertical channel IQ data

#### Parameters

**radar** [IQ radar object] Object containing the required fields

**signal\_field** [str] Name of the field that contain the H or V IQ data. None will use the default field name in the Py-ART configuration file.

#### Returns

**st1\_dict** [field dictionary] Field dictionary containing the st1

`pyart.retrieve.compute_st2_iq(radar, signal_field=None)`

Computes the statistical test two lag fluctuation from the horizontal or vertical channel IQ data

#### Parameters

**radar** [IQ radar object] Object containing the required fields

**signal\_field** [str] Name of the field that contain the H or V IQ data. None will use the default field name in the Py-ART configuration file.

#### Returns

**st2\_dict** [field dictionary] Field dictionary containing the st2

`pyart.retrieve.compute_svp(radar, field_names, lon, lat, angle, ref_time=None, ang_tol=1.0, latlon_tol=0.0005, delta_rng=15000.0, delta_az=10, hmax=10000.0, hres=250.0, avg_type='mean', nvalid_min=1, interp_kind='none', qvp=None)`

Computes slanted vertical profiles.

#### Parameters

**radar** [Radar] Radar object used.

**field\_names** [list of str] list of field names to add to the QVP

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

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

**ref\_time** [datetime object] reference time for current radar volume

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

**latlon\_tol** [float] tolerance in latitude and longitude in deg.

**delta\_rng, delta\_az** [float] maximum range distance [m] and azimuth distance [degree] from the central point of the evp containing data to average.

**hmax** [float] The maximum height to plot [m].

**hres** [float] The height resolution [m].

**avg\_type** [str] The type of averaging to perform. Can be either “mean” or “median”

**nvalid\_min** [int] Minimum number of valid points to accept average.

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

**qvp** [QVP object or None] If it is not None this is the QVP object where to store the data from the current time step. Otherwise a new QVP object will be created

#### Returns

**qvp** [qvp object] The computed slanted vertical profile

```
pyart.retrieve.compute_ts_along_coord(radar, field_name, mode='ALONG_AZI',
                                     fixed_range=None, fixed_azimuth=None,
                                     fixed_elevation=None, ang_tol=1.0, rng_tol=50.0,
                                     value_start=None, value_stop=None, ref_time=None,
                                     acoord=None)
```

Computes time series along a particular antenna coordinate, i.e. along azimuth, elevation or range

#### Parameters

**radar** [Radar] Radar object used.

**field\_name** [str] Name of the field

**mode** [str] coordinate to extract data along. Can be ALONG\_AZI, ALONG\_ELE or ALONG\_RNG

**fixed\_range, fixed\_azimuth, fixed\_elevation** [float] The fixed range [m], azimuth [deg] or elevation [deg] to extract. In each mode two of these parameters have to be defined. If they are not defined they default to 0.

**ang\_tol, rng\_tol** [float] The angle tolerance [deg] and range tolerance [m] around the fixed range or azimuth/elevation

**value\_start, value\_stop** [float] The minimum and maximum value at which the data along a coordinate start and stop

**ref\_time** [datetime object] reference time for current radar volume

**acoord** [acoord object or None] If it is not None this is the object where to store the data from the current time step. Otherwise a new acoord object will be created

#### Returns

**acoord** [acoord object] The computed data along a coordinate

```
pyart.retrieve.compute_vol_refl(radar, kw=0.93, freq=None, refl_field=None,
                               vol_refl_field=None)
```

Computes the volumetric reflectivity from the effective reflectivity factor

#### Parameters

**radar** [Radar] radar object

**kw** [float] water constant

**freq** [None or float] radar frequency

**refl\_field** [str] name of the reflectivity used for the calculations

**vol\_refl\_field** [str] name of the volumetric reflectivity

#### Returns

**vol\_refl\_dict** [dict] volumetric reflectivity and metadata in  $10\log_{10}(\text{cm}^2 \text{ km}^{-3})$

```
pyart.retrieve.compute_vp(radar, field_names, lon, lat, ref_time=None, latlon_tol=0.0005,
                          hmax=10000.0, hres=50.0, interp_kind='none', qvp=None)
```

Computes vertical profiles.

#### Parameters

**radar** [Radar] Radar object used.

**field\_names** [list of str] list of field names to add to the QVP

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

**ref\_time** [datetime object] reference time for current radar volume

**latlon\_tol** [float] tolerance in latitude and longitude in deg.

**hmax** [float] The maximum height to plot [m].

**hres** [float] The height resolution [m].

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

**qvp** [QVP object or None] If it is not None this is the QVP object where to store the data from the current time step. Otherwise a new QVP object will be created

### Returns

**qvp** [qvp object] The computed vertical profile

`pyart.retrieve.compute_wbn_iq(radar, signal_field=None)`

Computes the wide band noise from the horizontal or vertical channel IQ data

### Parameters

**radar** [IQ radar object] Object containing the required fields

**signal\_field** [str] Name of the field that contain the H or V IQ data. None will use the default field name in the Py-ART configuration file.

### Returns

**wbn\_dict** [field dictionary] Field dictionary containing the wide band noise

`pyart.retrieve.detect_ml(radar, gatefilter=None, fill_value=None, refl_field=None, rhohv_field=None, ml_field=None, ml_pos_field=None, iso0_field=None, max_range=20000, detect_threshold=0.02, interp_holes=False, max_length_holes=250, check_min_length=True, get_iso0=False)`

Detects the melting layer (ML) using the reflectivity and copolar correlation coefficient. Internally it uses RHIs

### Returns

**ml\_obj** [radar-like object] A radar-like object containing the field melting layer height with the bottom (at range position 0) and top (at range position one) of the melting layer at each ray

**ml\_dict** [dict] A dictionary containing the position of the range gate respect to the melting layer and metadata

**iso0\_dict** [dict or None] A dictionary containing the distance respect to the melting layer and metadata

**all\_ml** [dict] Dictionary containing internal parameters in polar and cartesian coordinates

`pyart.retrieve.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, optional] Name of the specific attenuation field to use.

**rr\_field** [str, optional] 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 Wavelengths. 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.est_rain_rate_hydro(radar, alphazr=0.0376, betazr=0.6112, alphazs=0.1,
                                   betazs=0.5, alphas=0.5, alphas=0.1,
                                   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, optional] Factor (alpha) and exponent (beta) of the z-r power law for rain.

**alphazs, betazs** [floats, optional] Factor (alpha) and exponent (beta) of the z-s power law for snow.

**alphas, betas** [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, optional] Factor applied to z-r relation in the melting layer.

**refl\_field** [str, optional] Name of the reflectivity field to use.

**a\_field** [str, optional] Name of the specific attenuation field to use.

**hydro\_field** [str, optional] Name of the hydrometeor classification field to use.

**rr\_field** [str, optional] Name of the rainfall rate field.

**master\_field** [str, optional] 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, optional] Value of the threshold that determines when to use the slave field.

**thresh\_max** [Bool, optional] 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.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, optional] Name of the specific differential phase field to use.

**rr\_field** [str, optional] Name of the rainfall rate field.

**Returns**

**rain** [dict] Field dictionary containing the rainfall rate.

```
pyart.retrieve.estimate_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, optional] Factor (alpha) and exponent (beta) of the power law.

**refl\_field** [str, optional] Name of the reflectivity field to use.

**rr\_field** [str, optional] Name of the rainfall rate field.

**Returns**

**rain** [dict] Field dictionary containing the rainfall rate.

```
pyart.retrieve.estimate_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=True)
```

Estimates rainfall rate from a blending of power law r-alpha and r-z relations.

**Parameters**

**radar** [Radar] Radar object

**alphaz, betaz** [floats, optional] 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, optional] Name of the reflectivity field to use.

**a\_field** [str, optional] Name of the specific attenuation field to use.

**rr\_field** [str, optional] Name of the rainfall rate field.

**master\_field** [str, optional] 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, optional] Value of the threshold that determines when to use the slave field.

**thresh\_max** [Bool, optional] 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.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, optional] 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, optional] Name of the reflectivity field to use.

**kdp\_field** [str, optional] Name of the specific differential phase field to use.

**rr\_field** [str, optional] Name of the rainfall rate field.

**master\_field** [str, optional] 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, optional] Value of the threshold that determines when to use the slave field [mm/h].

**thresh\_max** [Bool, optional] 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.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, optional] Name of the reflectivity field to use.

**rr\_field** [str, optional] Name of the rainfall rate field.

#### Returns

**rain** [dict] Field dictionary containing the rainfall rate.

```
pyart.retrieve.est_vertical_windshear(radar, az_tol=0.5, wind_field=None, windshear_field=None)
```

Estimates wind shear.

#### Parameters

**radar** [Radar] Radar object

**az\_tol** [float] azimuth tolerance to consider gate on top of selected one

**wind\_field** [str] name of the horizontal wind velocity field

**windshear\_field** [str] name of the vertical wind shear field

#### Returns

**windshear** [dict] Field dictionary containing the wind shear field

```
pyart.retrieve.est_wind_profile(radar, npoints_min=6, azi_spacing_max=45.0,
                               vel_diff_max=10.0, sign=1, rad_vel_field=None,
                               u_vel_field=None, v_vel_field=None, w_vel_field=None,
                               vel_est_field=None, vel_std_field=None, vel_diff_field=None)
```

Estimates the vertical wind profile using VAD techniques

#### Parameters

- radar** [Radar] Radar object
- npoints\_min** [int] Minimum number of points in the VAD to retrieve wind components. 0 will retrieve them regardless
- azi\_spacing\_max** [float] Maximum spacing between valid gates in the VAD to retrieve wind components. 0 will retrieve them regardless.
- vel\_diff\_max** [float] Maximum velocity difference allowed between retrieved and measured radial velocity at each range gate. Gates exceeding this threshold will be removed and VAD will be recomputed. If -1 there will not be a second pass.
- 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.
- rad\_vel\_field** [str] name of the measured radial velocity field
- u\_vel\_field, v\_vel\_field, w\_vel\_field** [str] names of the 3 wind components fields
- vel\_est\_field** [str] name of the retrieved radial Doppler velocity field
- vel\_std\_field** [str] name of the standard deviation of the velocity retrieval field
- vel\_diff\_field** [str] name of the difference between retrieved and measured radial velocity field

#### Returns

- wind** [dict] Field dictionary containing the estimated wind velocity

```
pyart.retrieve.est_wind_vel(radar, vert_proj=False, vel_field=None, wind_field=None)
```

Estimates wind velocity. Projects the radial wind component to the horizontal or vertical of the azimuth plane. It assumes that the orthogonal component is negligible.

**The horizontal wind component is given by:**  $v = v_r \cos(\text{el}) - v_{el} \sin(\text{el}) + v_{az}$

**where:**  $v_r$  is the radial wind component (measured by the radar)  $v_{el}$  is the perpendicular wind component in the azimuth plane.  $v_{az}$  is the horizontal component perpendicular to the radial direction and the azimuth plane  $\text{el}$  is the elevation

**The horizontal wind component in the azimuth plane is given by:**  $v_h = v_r \cos(\text{el}) - v_{el} \sin(\text{el})$

**which since we do not know  $v_{el}$  we assume:**  $v_h \sim v_r \cos(\text{el})$

This assumption holds for small elevation angles

**The vertical wind component in the azimuth plane is given by:**  $v_v = v_r \sin(\text{el}) - v_{el} \cos(\text{el})$

**which since we do not know  $v_{el}$  we assume:**  $v_v \sim v_r \sin(\text{el})$

This assumption holds for angles close to 90 deg

#### Parameters

- radar** [Radar] Radar object
- vert\_proj** [Boolean] If true estimates the vertical projection, otherwise the horizontal

**vel\_field** [str] name of the velocity field

**wind\_field** [str] name of the velocity field

#### Returns

**wind** [dict] Field dictionary containing the estimated wind velocity

`pyart.retrieve.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 nvars.

#### Returns

**return\_dic** [dict] Profiles at the start time of the radar.

`pyart.retrieve.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

`pyart.retrieve.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.grid_displacement_pc` (*grid1*, *grid2*, *field*, *level*, *return\_value='pixels'*)  
Calculate the grid displacement using phase correlation.

See: [http://en.wikipedia.org/wiki/Phase\\_correlation](http://en.wikipedia.org/wiki/Phase_correlation)

Implementation inspired by Christoph Gohlke: <http://www.lfd.uci.edu/~gohlke/code/imreg.py.html>

Note that the grid must have the same dimensions in x and y and assumed to have constant spacing in these dimensions.

#### Parameters

**grid1, grid2** [Grid] Py-ART Grid objects separated in time and square in x/y.

**field** [string] Field to calculate advection from. Field must be in both grid1 and grid2.

**level** [integer] The vertical (z) level of the grid to use in the calculation.

**return\_value** [str, optional] 'pixels', 'distance' or 'velocity'. Distance in pixels (default) or meters or velocity vector in m/s.

### Returns

**displacement** [two-tuple] Calculated displacement in units of y and x. Value returned in integers if pixels, otherwise floats.

`pyart.retrieve.grid_shift` (*grid*, *advection*, *trim\_edges=0*, *field\_list=None*)  
Shift a grid by a certain number of pixels.

### Parameters

**grid:** **Grid** Py-ART Grid object.

**advection** [two-tuple of floats] Number of Pixels to shift the image by.

**trim\_edges:** **integer, optional** Edges to cut off the grid and axes, both x and y. Defaults to zero.

**field\_list** [list, optional] List of fields to include in new grid. None, the default, includes all fields from the input grid.

### Returns

**shifted\_grid** [Grid] Grid with fields shifted and, if requested, subset.

`pyart.retrieve.hydroclass_semisupervised` (*radar*, *mass\_centers=None*, *weights=array([1., 1., 0.75, 0.5 ])*, *value=50.0*, *refl\_field=None*, *zdr\_field=None*, *rhv\_field=None*, *kdp\_field=None*, *temp\_field=None*, *iso0\_field=None*, *hydro\_field=None*, *entropy\_field=None*, *temp\_ref='temperature'*, *compute\_entropy=False*, *output\_distances=False*, *vectorize=False*)

Classifies precipitation echoes following the approach by Besic et al (2016).

### Parameters

**radar** [radar] Radar object.

**mass\_centers** [ndarray 2D, optional] The centroids for each variable and hydrometeor class in (nclasses, nvariables).

**weights** [ndarray 1D, optional] The weight given to each variable.

**value** [float] The value controlling the rate of decay in the distance transformation

**refl\_field, zdr\_field, rhv\_field, kdp\_field, temp\_field, iso0\_field** [str] Inputs. Field names within the radar object which represent the horizontal reflectivity, the differential reflectivity, the copolar correlation coefficient, the specific differential phase, the temperature and the height respect to the iso0 fields. 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.

**temp\_ref** [str] the field use as reference for temperature. Can be either temperature or height\_over\_iso0

**compute\_entropy** [bool] If true, the entropy is computed

**output\_distances** [bool] If true, the normalized distances to the centroids for each hydrometeor are provided as output

**vectorize** [bool] If true, a vectorized version of the class assignment is going to be used

#### Returns

**fields\_dict** [dict] Dictionary containing the retrieved fields

#### 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.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, vectorize=False)
```

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.

**vectorize** [bool] whether to use a vectorized version of the least square method

#### Returns

**kdp\_dict** [dict] Retrieved specific differential phase data and metadata.

```
pyart.retrieve.kdp_leastsquare_single_window(radar,    wind_len=11,    min_valid=6,
                                              phidp_field=None,    kdp_field=None,
                                              vectorize=False)
```

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.

**vectorize** [bool] whether to use a vectorized version of the least square method

### Returns

**kdp\_dict** [dict] Retrieved specific differential phase data and metadata.

```
pyart.retrieve.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 attenuation 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_schneebeli (radar, gatefilter=None, fill_value=None, psidp_field=None,
                              kdp_field=None, phidp_field=None, band='C', rcov=0, pcov=0,
                              prefilter_psidp=False, filter_opt=None, parallel=True)
```

Estimates Kdp with the Kalman filter method by Schneebeli and al. (2014) for a set of psidp measurements.

#### Parameters

**radar** [Radar] Radar containing differential phase field.

**gatefilter** [GateFilter, optional] A GateFilter indicating radar gates that should be excluded when analysing differential phase measurements.

**fill\_value** [float, optional] Value indicating missing or bad data in differential phase field, if not specified, the default in the Py-ART configuration file will be used.

**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.

**band** [char, optional] Radar frequency band string. Accepted “X”, “C”, “S” (capital or not). The band is used to compute intercepts -c and slope b of the  $\delta = b \cdot Kdp + c$  relation.



**rcov** [3x3 float array, optional] Measurement error covariance matrix.

**pcov** [4x4 float array, optional] Scaled state transition error covariance matrix.

**prefilter\_psidp** [bool, optional] If set, the psidp measurements will first be filtered with the `filter_psidp` method, which can improve the quality of the final Kdp.

**filter\_opt** [dict, optional] The arguments for the `prefilter_psidp` method, if empty, the defaults arguments of this method will be used.

**parallel** [bool, optional] Flag to enable parallel computation (one core for every psidp profile).

#### Returns

**kdp\_dict** [dict] Retrieved specific differential phase data and metadata.

**kdp\_std\_dict** [dict] Estimated specific differential phase standard dev. data and metadata.

**phidpr\_dict, dict** Retrieved differential phase data and metadata.

#### References

Schneebeli, M., Grazioli, J., and Berne, A.: Improved Estimation of the Specific Differential Phase SHIFT Using a Compilation of Kalman Filter Ensembles, *IEEE T. Geosci. Remote Sens.*, 52, 5137-5149, doi:10.1109/TGRS.2013.2287017, 2014.

```
pyart.retrieve.kdp_vulpiani (radar, gatefilter=None, fill_value=None, psidp_field=None,
                             kdp_field=None, phidp_field=None, band='C', windsize=10,
                             n_iter=10, interp=False, prefilter_psidp=False, filter_opt=None,
                             parallel=False)
```

Estimates Kdp with the Vulpiani method for a 2D array of psidp measurements with the first dimension being the distance from radar and the second dimension being the angles (azimuths for PPI, elev for RHI). The input psidp is assumed to be pre-filtered (for ex. with the `filter_psidp` function)

#### Parameters

**radar** [Radar] Radar containing differential phase field.

**gatefilter** [GateFilter, optional] A GateFilter indicating radar gates that should be excluded when analysing differential phase measurements.

**fill\_value** [float, optional] Value indicating missing or bad data in differential phase field, if not specified, the default in the Py-ART configuration file will be used

**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.

**band** [char, optional] Radar frequency band string. Accepted "X", "C", "S" (capital or not). It is used to set default boundaries for expected values of Kdp.

**windsize** [int, optional] Size in # of gates of the range derivative window. Should be even.

**n\_iter** [int, optional] Number of iterations of the method. Default is 10.

**interp** [bool, optional] If True, all the nans are interpolated. The advantage is that less data are lost (the iterations in fact are "eating the edges") but some non-linear errors may be introduced.

**prefilter\_psidp** [bool, optional] If set, the psidp measurements will first be filtered with the `filter_psidp` method, which can improve the quality of the final Kdp.

**filter\_opt** [dict, optional] The arguments for the `prefilter_psidp` method, if empty, the defaults arguments of this method will be used.

**parallel** [bool, optional] Flag to enable parallel computation (one core for every psidp profile).

#### Returns

**kdp\_dict** [dict] Retrieved specific differential phase data and metadata.

**phidpr\_dict,; dict** Retrieved differential phase data and metadata.

## References

Gianfranco Vulpiani, Mario Montopoli, Luca Delli Passeri, Antonio G. Gioia, Pietro Giordano, and Frank S. Marzano, 2012: On the Use of Dual-Polarized C-Band Radar for Operational Rainfall Retrieval in Mountainous Areas. *J. Appl. Meteor. Climatol.*, 51, 405-425, doi: 10.1175/JAMC-D-10-05024.1.

```
pyart.retrieve.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, optional] Name to use for height field metadata. None will use the default field name from the Py-ART configuration file.

**profile\_field** [str, optional] 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.

```
pyart.retrieve.melting_layer_giangrande(radar, nVol=3, maxh=6000.0, hres=50.0,
                                         rmin=1000.0, elmin=4.0, elmax=10.0,
                                         rhomin=0.75, rhomax=0.94, zhmin=20.0,
                                         hwindow=500.0, mlzhmin=30.0, mlzh-
                                         max=50.0, mlzdrmin=1.0, mlzdrmax=5.0,
                                         htol=500.0, ml_bottom_diff_max=1000.0,
                                         time_accu_max=1800.0, nml_points_min=None,
                                         wlength=20.0, percentile_bottom=0.3,
                                         percentile_top=0.9, interpol=True,
                                         time_nodata_allowed=3600.0, refl_field=None,
                                         zdr_field=None, rhv_field=None,
                                         temp_field=None, iso0_field=None,
                                         ml_field=None, ml_pos_field=None,
                                         temp_ref=None, get_iso0=False,
                                         ml_global=None)
```

Detects the melting layer following the approach by Giangrande et al (2008)

**Parameters**

**radar** [radar] radar object

**Returns**

**ml\_obj** [radar-like object] A radar-like object containing the field melting layer height with the bottom (at range position 0) and top (at range position one) of the melting layer at each ray

**ml\_dict** [dict] A dictionary containing the position of the range gate respect to the melting layer and metadata

**iso0\_dict** [dict or None] A dictionary containing the distance respect to the melting layer and metadata

**ml\_global** [dict or None] stack of previous volume data to introduce some time dependency. Its max size is controlled by the nVol parameter. It is always in (pseudo-)RHI mode.

**Other Parameters**

**nVol** [int] Number of volume scans to aggregate

**maxh** [float] Maximum possible height of the melting layer [m MSL]

**hres** [float] Step of the height of the melting layer [m]

**rmin** [float] Minimum range from radar where to look for melting layer contaminated range gates [m]

**elmin, elmax** [float] Minimum and maximum elevation angles where to look for melting layer contaminated range gates [degree]

**rhomin, rhomax** [float] min and max rhohv to consider pixel potential melting layer pixel

**zhmin** [float] Minimum reflectivity level of a range gate to consider it a potential melting layer gate [dBZ]

**hwindow** [float] Maximum distance (in range) from potential melting layer gate where to look for a maximum [m]

**mlzhmin, mlzhmax** [float] Minimum and maximum values that a peak in reflectivity within the melting layer may have to consider the range gate melting layer contaminated [dBZ]

**mlzdrmin, mlzdrmax** [float] Minimum and maximum values that a peak in differential reflectivity within the melting layer may have to consider the range gate melting layer contaminated [dB]

**htol** [float] maximum distance from the iso0 coming from model allowed to consider the range gate melting layer contaminated [m]

**ml\_bottom\_dif\_max** [float] Maximum distance from the bottom of the melting layer computed in the previous time step to consider a range gate melting layer contaminated [m]

**time\_accu\_max** [float] Maximum time allowed to accumulate data from consecutive scans [s]

**nml\_points\_min** [int] minimum number of melting layer points to consider valid melting layer detection

**wlength** [float] length of the window to select the azimuth angles used to compute the melting layer limits at a particular azimuth [degree]

**percentile\_bottom, percentile\_top** [float [0,1]] percentile of ml points above which is considered that the bottom of the melting layer starts and the top ends

**interpol** [bool] Whether to interpolate the obtained results in order to get a value for each azimuth

**time\_nodata\_allowed** [float] The maximum time allowed for no data before considering the melting layer not valid [s]

**refl\_field, zdr\_field, rhv\_field, temp\_field, iso0\_field** [str] Inputs. Field names within the radar object which represent the horizontal reflectivity, the differential reflectivity, the copolar correlation coefficient, the temperature and the height respect to the iso0 fields. A value of None for any of these parameters will use the default field name as defined in the Py-ART configuration file.

**ml\_field** [str] Output. Field name which represents the melting layer field. A value of None will use the default field name as defined in the Py-ART configuration file.

**ml\_pos\_field** [str] Output. Field name which represents the melting layer top and bottom height field. A value of None will use the default field name as defined in the Py-ART configuration file.

**temp\_ref** [str] the field use as reference for temperature. Can be temperature or height\_over\_iso0. If None, it excludes model data from the algorithm.

**get\_iso0** [bool] returns height w.r.t. freezing level top for each gate in the radar volume.

**ml\_global** : stack of previous volume data to introduce some time dependency. Its max size is controlled by the nVol parameter. It is always in (pseudo-)RHI mode.

## References

Giangrande, S.E., Krause, J.M., Ryzhkov, A.V.: Automatic Designation of the Melting Layer with a Polarimetric Prototype of the WSR-88D Radar, J. of Applied Meteo. and Clim., 47, 1354-1364, doi:10.1175/2007JAMC1634.1, 2008

```
pyart.retrieve.melting_layer_hydroclass (radar,      hydro_field=None,      ml_field=None,
                                         ml_pos_field=None,      iso0_field=None,
                                         force_continuity=True,      dist_max=350.0,
                                         get_iso0=False)
```

Using the results of the hydrometeor classification by Besic et al. estimates the position of the range gates respect to the melting layer, the melting layer top and bottom height and the distance of the range gate with respect to the freezing level.

## Parameters

**radar** [Radar] Radar object. Must have and hydrometeor classification field

**hydro\_field** [str] Name of the hydrometeor classification field. A value of None will use the default field name as defined in the Py-ART configuration file.

**ml\_field, ml\_pos\_field, iso0\_field** [str] Name of the melting layer, melting layer height and iso0 field. A value of None for any of these parameters will use the default field names as defined in the Py-ART configuration file.

**force\_continuity** [Bool] If True, the melting layer is forced to be continuous in range

**dist\_max** [float] The maximum distance between range gates flagged as inside the melting layer to consider them as gates in the melting layer.

## Returns

**ml\_obj** [radar-like object] A radar-like object containing the field melting layer height with the bottom (at range position 0) and top (at range position one) of the melting layer at each ray

**ml\_dict** [dict] A dictionary containing the position of the range gate respect to the melting layer and metadata

**iso0\_dict** [dict or None] A dictionary containing the distance respect to the melting layer and metadata

`pyart.retrieve.quasi_vertical_profile` (*radar*, *desired\_angle=None*, *fields=None*, *gatefilter=None*)

Quasi Vertical Profile.

Creates a QVP object containing fields from a radar object that can be used to plot and produce the quasi vertical profile

#### Parameters

**radar** [Radar] Radar object used.

**field** [string] Radar field to use for QVP calculation.

**desired\_angle** [float] Radar tilt angle to use for indexing radar field data. None will result in `wanted_angle = 20.0`

#### Returns

**qvp** [Dictionary] A quasi vertical profile object containing fields from a radar object

#### Other Parameters

**gatefilter** [GateFilter] A GateFilter indicating radar gates that should be excluded from the import qvp calculation

## References

Troemel, S., M. Kumjian, A. Ryzhkov, and C. Simmer, 2013: Backscatter differential phase - estimation and variability. J Appl. Meteor. Clim.. 52, 2529 - 2548.

Troemel, S., A. Ryzhkov, P. Zhang, and C. Simmer, 2014: Investigations of backscatter differential phase in the melting layer. J. Appl. Meteorol. Clim. 54, 2344 - 2359.

Ryzhkov, A., P. Zhang, H. Reeves, M. Kumjian, T. Tschallener, S. Tromel, C. Simmer, 2015: Quasi-vertical profiles - a new way to look at polarimetric radar data. Submitted to J. Atmos. Oceanic Technol.

`pyart.retrieve.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.

**dx, dy** [float, optional] The x- and y-dimension resolutions in meters, respectively. If None the resolution is determined from the first two axes values.

**intense** [float, optional] 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, optional] 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', optional] The peakedness relation. See reference for more information.

**area\_relation** ['small', 'medium', 'large', or 'sgp', optional] The convective area relation. See reference for more information.

**bkg\_rad** [float, optional] The background radius in meters. See reference for more information.

**use\_intense** [bool, optional] True to use the intensity criteria.

**fill\_value** [float, optional] 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, optional] Field in grid to use as the reflectivity during partitioning. None will use the default reflectivity field name from the Py-ART configuration file.

#### Returns

**eclass** [dict] Steiner convective-stratiform classification dictionary.

#### 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.

`pyart.retrieve.velocity_azimuth_display(radar, vel_field=None, z_want=None, gatefilter=None)`

Velocity azimuth display.

Creates a VAD object containing U Wind, V Wind and height that can then be used to plot and produce the velocity azimuth display.

#### Parameters

**radar** [Radar] Radar object used.

**vel\_field** [string, optional] Velocity field to use for VAD calculation.

**z\_want** [array, optional] Heights for where to sample vads from. None will result in `np.linspace(0, 10000, 100)`.

**gatefilter** [GateFilter, optional] A GateFilter indicating radar gates that should be excluded from the import vad calculation.

#### Returns

**vad: HorizontalWindProfile** A velocity azimuth display object containing height, speed, direction, u\_wind, v\_wind from a radar object.

## MAPPING (PYART.MAP)

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

<code>grid_from_radars(radars, grid_shape, grid_limits)</code>	Map one or more radars to a Cartesian grid returning a Grid object.
<code>map_to_grid(radars, grid_shape, grid_limits)</code>	Map one or more radars to a Cartesian grid.
<code>map_gates_to_grid(radars, grid_shape, ...[, ...])</code>	Map gates from one or more radars to a Cartesian grid.
<code>example_roi_func_constant(zg, yg, xg)</code>	Example RoI function which returns a constant radius.
<code>example_roi_func_dist(zg, yg, xg)</code>	Example RoI function which returns a radius which grows with distance.
<code>example_roi_func_dist_beam(zg, yg, xg)</code>	Example RoI function which returns a radius which grows with distance and whose parameters are based on virtual beam size.

`pyart.map.example_roi_func_constant(zg, yg, xg)`

Example RoI function which returns a constant radius.

### Parameters

**zg, yg, xg** [float] Distance from the grid center in meters for the x, y and z axes.

### Returns

**roi** [float] Radius of influence in meters

`pyart.map.example_roi_func_dist(zg, yg, xg)`

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

### Parameters

**zg, yg, xg** [float] Distance from the grid center in meters for the x, y and z axes.

### Returns

**roi** [float]

`pyart.map.example_roi_func_dist_beam(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.get_earth_radius(latitude)`

Computes the earth radius for a given latitude

**Parameters**

**latitude:** latitude in degrees (WGS84)

**Returns**

**earth\_radius** [the radius of the earth at the given latitude]

`pyart.map.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.

**References**

Barnes S., 1964: A Technique for Maximizing Details in Numerical Weather Map Analysis. Journal of Applied Meteorology and Climatology, 3(4), 396-409.

Cressman G., 1959: An operational objective analysis system. Monthly Weather Review, 87(10), 367-374.

Pauley, P. M. and X. Wu, 1990: The theoretical, discrete, and actual response of the Barnes objective analysis scheme for one- and two-dimensional fields. Monthly Weather Review, 118, 1145-1164

`pyart.map.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=None,  
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 function 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 keys of 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.

```
pyart.map.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, algo-
                      rithm='kd_tree', leafsize=10.0, roi_func='dist_beam', constant_roi=None,
                      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 coordinates. None will use the default dictionary which uses a native azimuthal 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 'Barnes2' or 'Cressman' or 'Nearest'] 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 keys of 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 used when `roi_func` is `constant` or `constant_roi` is not None. If `constant_roi` is not None, the `constant_roi_func` is used automatically.

**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 meter 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.

`pyart.map.polar_to_cartesian(radar_sweep, field_name, cart_res=75, max_range=None, mapping=None)`

Interpolates a PPI or RHI scan in polar coordinates to a regular cartesian grid of South-North and West-East coordinates (for PPI) or distance at ground and altitude coordinates (for RHI)

#### Parameters

**radar** [Radar] Radar instance as generated py pyart

**sweep** [int] Sweep number to project to cartesian coordinates.

**field\_name** [str] Name of the radar field to be interpolated

**cart\_res** [int, optional] Resolution (in m.) of the cartesian grid to which polar data is interpolated

**max\_range** [int, optional] Maximal allowed range (in m.) from radar for gates to be interpolated

**mapping** [dict, optional] Dictionnary of mapping indexes (from polar to cartesian), gets returned by the function (see below). Can be used as input when interpolating sequentially several variables for the same scan, to save significant time

#### Returns

**coords** [tuple of 2 arrays] 2D coordinates of the cartesian grid

**cart\_data** [2D array] Interpolated radar measurements (on the cartesian grid)

**mapping,:** dict Dictionnary of mapping indexes (from polar to cartesian), which contains the indexes mapping the polar grid to the cartesian grid as well as some metadata.



## GRAPHING (PYART.GRAPH)

Creating plots of Radar and Grid fields.

### 9.1 Plotting radar data

<i>RadarDisplay</i> (radar[, shift])	A display object for creating plots from data in a radar object.
<i>RadarMapDisplay</i> (radar[, shift, grid_projection])	A display object for creating plots on a geographic map from data in a Radar object.
<i>AirborneRadarDisplay</i> (radar[, shift])	A display object for creating plots from data in a airborne radar object.
<i>RadarMapDisplayBasemap</i> (radar[, shift])	A display object for creating plots on a geographic map from data in a Radar object.

### 9.2 Plotting grid data

<i>GridMapDisplay</i> (grid[, debug])	A class for creating plots from a grid object using xarray with a cartopy projection.
<i>GridMapDisplayBasemap</i> (grid[, debug])	A class for creating plots from a grid object on top of a Basemap.

**class** pyart.graph.**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

<code>generate_az_rhi_title(self, field, azimuth)</code>	Generate a title for a ray plot.
<code>generate_filename(self, field, sweep[, ext, ...])</code>	Generate a filename for a plot.
<code>generate_ray_title(self, field, ray)</code>	Generate a title for a ray plot.
<code>generate_title(self, field, sweep[, ...])</code>	Generate a title for a plot.
<code>generate_vpt_title(self, field)</code>	Generate a title for a VPT plot.
<code>label_xaxis_r(self[, ax])</code>	Label the xaxis with the default label for r units.
<code>label_xaxis_rays([ax])</code>	Label the yaxis with the default label for rays.
<code>label_xaxis_time([ax])</code>	Label the yaxis with the default label for rays.
<code>label_xaxis_x(self[, ax])</code>	Label the xaxis with the default label for x units.
<code>label_yaxis_field(self, field[, ax])</code>	Label the yaxis with the default label for a field units.
<code>label_yaxis_y(self[, ax])</code>	Label the yaxis with the default label for y units.
<code>label_yaxis_z(self[, ax])</code>	Label the yaxis with the default label for z units.
<code>plot(self, field[, sweep])</code>	Create a plot appropriate for the radar.
<code>plot_azimuth_to_rhi(self, field, target_azimuth)</code>	Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.
<code>plot_colorbar(self[, mappable, field, ...])</code>	Plot a colorbar.
<code>plot_cross_hair(size[, npts, ax])</code>	Plot a cross-hair on a ppi plot.
<code>plot_grid_lines([ax, col, ls])</code>	Plot grid lines.
<code>plot_label(self, label, location[, symbol, ...])</code>	Plot a single symbol and label at a given location.
<code>plot_labels(self, labels, locations[, ...])</code>	Plot symbols and labels at given locations.
<code>plot_ppi(self, field[, sweep, mask_tuple, ...])</code>	Plot a PPI.
<code>plot_range_ring(range_ring_location_km[, ...])</code>	Plot a single range ring.
<code>plot_range_rings(self, range_rings[, ax, ...])</code>	Plot a series of range rings.
<code>plot_ray(self, field, ray[, format_str, ...])</code>	Plot a single ray.

Continued on next page

Table 3 – continued from previous page

<code>plot_rhi(self, field[, sweep, mask_tuple, ...])</code>	Plot a RHI.
<code>plot_sweep_grid(self, field[, sweep, ...])</code>	Plot a sweep as a grid.
<code>plot_vpt(self, field[, mask_tuple, vmin, ...])</code>	Plot a VPT scan.
<code>set_aspect_ratio([aspect_ratio, ax])</code>	Set the aspect ratio for plot area.
<code>set_limits([xlim, ylim, ax])</code>	Set the display limits.

```

__class__
    alias of builtins.type

__delattr__(self, name, /)
    Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyart.graph.radardisplay_airborne', '__doc__'
__dir__(self, /)
    Default dir() implementation.

__eq__(self, value, /)
    Return self==value.

__format__(self, format_spec, /)
    Default object formatter.

__ge__(self, value, /)
    Return self>=value.

__getattr__(self, name, /)
    Return getattr(self, name).

__gt__(self, value, /)
    Return self>value.

__hash__(self, /)
    Return hash(self).

__init__(self, radar, shift=(0.0, 0.0))
    Initialize the object.

__init_subclass__()
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be overridden to extend subclasses.

__le__(self, value, /)
    Return self<=value.

__lt__(self, value, /)
    Return self<value.

__module__ = 'pyart.graph.radardisplay_airborne'

__ne__(self, value, /)
    Return self!=value.

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

__reduce__(self, /)
    Helper for pickle.

__reduce_ex__(self, protocol, /)
    Helper for pickle.

```

**\_\_repr\_\_** (*self*, /)  
Return repr(*self*).

**\_\_setattr\_\_** (*self*, *name*, *value*, /)  
Implement setattr(*self*, *name*, *value*).

**\_\_sizeof\_\_** (*self*, /)  
Size of object in memory, in bytes.

**\_\_str\_\_** (*self*, /)  
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** (*self*, *field*, *target\_azimuth*, *edges*, *mask\_tuple*, *filter\_transitions*, *gatefilter*)  
Retrieve and return pseudo-RHI data from a plot function.

**\_get\_colorbar\_label** (*self*, *field*)  
Return a colorbar label for a given field.

**\_get\_data** (*self*, *field*, *sweep*, *mask\_tuple*, *filter\_transitions*, *gatefilter*)  
Retrieve and return data from a plot function.

**\_get\_ray\_data** (*self*, *field*, *ray*, *mask\_tuple*, *gatefilter*)  
Retrieve and return ray data from a plot function.

**\_get\_vpt\_data** (*self*, *field*, *mask\_tuple*, *filter\_transitions*, *gatefilter*)  
Retrieve and return vpt data from a plot function.

**\_get\_x\_y** (*self*, *sweep*, *edges*, *filter\_transitions*)  
Retrieve and return x and y coordinate in km.

**\_get\_x\_y\_z** (*self*, *sweep*, *edges*, *filter\_transitions*, *ignoreTilt=False*)  
Retrieve and return x, y, and z coordinate in km.

**\_get\_x\_z** (*self*, *sweep*, *edges*, *filter\_transitions*, *ignoreTilt=False*)  
Retrieve and return x and z coordinate in km.

**\_label\_axes\_ppi** (*self*, *axis\_labels*, *ax*)  
Set the x and y axis labels for a PPI plot.

**\_label\_axes\_ray** (*self*, *axis\_labels*, *field*, *ax*)  
Set the x and y axis labels for a ray plot.

**\_label\_axes\_rhi** (*self*, *axis\_labels*, *ax*)  
Set the x and y axis labels for a RHI plot.

**\_label\_axes\_vpt** (*self*, *axis\_labels*, *time\_axis\_flag*, *ax*)  
Set the x and y axis labels for a PPI plot.

**\_set\_az\_rhi\_title** (*self*, *field*, *azimuth*, *title*, *ax*)  
Set the figure title for a ray plot using a default title.

**\_set\_ray\_title** (*self*, *field*, *ray*, *title*, *ax*)  
Set the figure title for a ray plot using a default title.



**\_set\_title** (*self, field, sweep, title, ax, datetime\_format=None, use\_sweep\_time=True*)  
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** (*self, field, title, ax*)  
Set the figure title using a default title.

**generate\_az\_rhi\_title** (*self, 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** (*self, field, sweep, ext='png', datetime\_format='%Y%m%d%H%M%S', use\_sweep\_time=False*)  
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.

**datetime\_format** [str] Format of datetime (using strftime format).

**use\_sweep\_time** [bool] If true, the current sweep's beginning time is used.

#### Returns

**filename** [str] Filename suitable for saving a plot.

**generate\_ray\_title** (*self, 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** (*self, field, sweep, datetime\_format=None, use\_sweep\_time=True*)  
Generate a title for a plot.

**Parameters**

**field** [str] Field plotted.

**sweep** [int] Sweep plotted.

**datetime\_format** [str] Format of datetime (using strftime format).

**use\_sweep\_time** [bool] If true, the current sweep's beginning time is used.

**Returns**

**title** [str] Plot title.

**generate\_vpt\_title** (*self*, *field*)

Generate a title for a VPT plot.

**Parameters**

**field** [str] Field plotted.

**Returns**

**title** [str] Plot title.

**label\_xaxis\_r** (*self*, *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** (*self*, *ax=None*)

Label the xaxis with the default label for x units.

**label\_yaxis\_field** (*self*, *field*, *ax=None*)

Label the yaxis with the default label for a field units.

**label\_yaxis\_y** (*self*, *ax=None*)

Label the yaxis with the default label for y units.

**label\_yaxis\_z** (*self*, *ax=None*)

Label the yaxis with the default label for z units.

**plot** (*self*, *field*, *sweep=0*, *\*\*kwargs*)

Create a plot appropriate 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 appropriate 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 (self, 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, raster=False, **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 if norm is not None.

**vmax** [float] Luminance maximum value, None for default value. Parameter is ignored if 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 themselves 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.

**raster** [bool] False by default. Set to True to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**plot\_colorbar** (*self*, *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** (*self*, *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** (*self, 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** (*self, 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, raster=False, title\_datetime\_format=None, title\_use\_sweep\_time=True, \*\*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\_datetime\_format** [str] Format of datetime in the title (using strftime format).

**title\_use\_sweep\_time** [bool] True for the current sweep's beginning time to be used for the title. False for the radar's beginning time.

**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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**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** (*self, 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** (*self, 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** (*self, 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, raster=False, title\_datetime\_format=None, title\_use\_sweep\_time=True, \*\*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 if norm is not None.

**vmax** [float] Luminance maximum value, None for default value. Parameter is ignored if 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\_datetime\_format** [str] Format of datetime in the title (using strftime format).

**title\_use\_sweep\_time** [bool] True for the current sweep's beginning time to be used for the title. False for the radar's beginning time.

**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 west to east, False to have east to west. None (the default) will reverse the axis only when all the distances are negative. (i.e) axis will be absolute distance without taking into consideration the orientation

**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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

```
plot_sweep_grid(self, field, sweep=0, ignoreTilt=False, 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, color-
                 bar_flag=True, colorbar_label=None, colorbar_orient='vertical', edges=True,
                 filter_transitions=True, ax=None, fig=None, gatefilter=None, raster=False,
                 ticks=None, ticklabs=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

**ignoreTilt** [bool] True to ignore tilt angle when running the antenna\_to\_cartesian\_track\_relative coordinate transformation (by setting tilt angle to 0), effectively plotting data relative to slant range (the same plotting method utilized by the NCAR soloii/3 software). False (default) plots relative to the aircraft longitudinal axis.

**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 if norm is not None.

**vmax** [float] Luminance maximum value, None for default value. Parameter is ignored if 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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**ticks** [array] Colorbar custom tick label locations.

**ticklabs** [array] Colorbar custom tick labels.

**plot\_vpt** (*self*, *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*, *gatefilter=None*, *filter\_transitions=True*, *time\_axis\_flag=False*, *date\_time\_form=None*, *tz=None*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, *raster=False*, *\*\*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 themselves 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.

**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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**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.

**class** `pyart.graph.GridMapDisplay` (*grid*, *debug=False*)

Bases: `object`

A class for creating plots from a grid object using xarray with a cartopy projection.

#### Parameters

**grid** [Grid] Grid with data which will be used to create plots.

**debug** [bool] True to print debugging messages, False to suppress them.

#### Attributes

**grid** [Grid] Grid object.

**debug** [bool] True to print debugging messages, False to suppress them.

#### Methods

<code>cartopy_coastlines(self)</code>	Get coastlines using cartopy.
<code>cartopy_political_boundaries(self)</code>	Get political boundaries using cartopy.
<code>cartopy_states(self)</code>	Get state boundaries using cartopy.
<code>generate_filename(self, field, level[, ext])</code>	Generate a filename for a grid plot.
<code>generate_grid_title(self, field, level)</code>	Generate a title for a plot.
<code>generate_latitudinal_level_title(self, ...)</code>	Generate a title for a plot.
<code>generate_longitudinal_level_title(self, ...)</code>	Generate a title for a plot.
<code>get_dataset(self)</code>	Creating an xarray dataset from a radar object.
<code>plot_colorbar(self[, mappable, orientation, ...])</code>	Plot a colorbar.
<code>plot_crosshairs(self[, lon, lat, linestyle, ...])</code>	Plot crosshairs at a given longitude and latitude.
<code>plot_grid(self, field[, level, vmin, vmax, ...])</code>	Plot the grid using xarray and cartopy.
<code>plot_grid_contour(self, field[, level, ...])</code>	Plot the grid contour using xarray and cartopy.
<code>plot_latitude_slice(self, field[, lon, lat])</code>	Plot a slice along a given latitude.
<code>plot_latitudinal_level(self, field, y_index)</code>	Plot a slice along a given latitude.
<code>plot_latlon_level(self, field, ind_1, ind_2)</code>	Plot a slice along two points given by its lat, lon. Additional arguments are passed to Basemaps's <code>pcolormesh</code> function.
<code>plot_latlon_slice(self, field[, coord1, coord2])</code>	Plot a slice along a given longitude.
<code>plot_longitude_slice(self, field[, lon, lat])</code>	Plot a slice along a given longitude.
<code>plot_longitudinal_level(self, field, x_index)</code>	Plot a slice along a given longitude.

**\_\_class\_\_**

alias of `builtins.type`

**\_\_delattr\_\_** (*self*, *name*, /)

Implement `delattr(self, name)`.

```

__dict__ = mappingproxy({'__module__': 'pyart.graph.gridmapdisplay', '__doc__': '\n
__dir__ (self, /)
    Default dir() implementation.

__eq__ (self, value, /)
    Return self==value.

__format__ (self, format_spec, /)
    Default object formatter.

__ge__ (self, value, /)
    Return self>=value.

__getattr__ (self, name, /)
    Return getattr(self, name).

__gt__ (self, value, /)
    Return self>value.

__hash__ (self, /)
    Return hash(self).

__init__ (self, grid, debug=False)
    initialize the object.

__init_subclass__ ()
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (self, value, /)
    Return self<=value.

__lt__ (self, value, /)
    Return self<value.

__module__ = 'pyart.graph.gridmapdisplay'

__ne__ (self, value, /)
    Return self!=value.

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

__reduce__ (self, /)
    Helper for pickle.

__reduce_ex__ (self, protocol, /)
    Helper for pickle.

__repr__ (self, /)
    Return repr(self).

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

__sizeof__ (self, /)
    Size of object in memory, in bytes.

__str__ (self, /)
    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(self, lon, lat)`**

Find the nearest x, y grid indices for a given latitude and longitude.

**`_get_label_x(self)`**

Get default label for x units.

**`_get_label_y(self)`**

Get default label for y units.

**`_get_label_z(self)`**

Get default label for z units.

**`_label_axes_grid(self, axis_labels, ax)`**

Set the x and y axis labels for a grid plot.

**`_label_axes_latitude(self, axis_labels, ax)`**

Set the x and y axis labels for a latitude slice.

**`_label_axes_latlon(self, axis_labels, ax)`**

Set the x and y axis labels for a lat-lon slice.

**`_label_axes_longitude(self, axis_labels, ax)`**

Set the x and y axis labels for a longitude slice.

**`cartopy_coastlines(self)`**

Get coastlines using cartopy.

**`cartopy_political_boundaries(self)`**

Get political boundaries using cartopy.

**`cartopy_states(self)`**

Get state boundaries using cartopy.

**`generate_filename(self, 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(self, field, level)`**

Generate a title for a plot.

#### Parameters

**field** [str] Field plotted.

**level** [int] Vertical level plotted.

#### Returns

**title** [str] Plot title.

**generate\_latitudinal\_level\_title** (*self, field, level*)

Generate a title for a plot.

#### Parameters

**field** [str] Field plotted.

**level** [int] Latitudinal level plotted.

#### Returns

**title** [str] Plot title.

**generate\_longitudinal\_level\_title** (*self, field, level*)

Generate a title for a plot.

#### Parameters

**field** [str] Field plotted.

**level** [int] Longitudinal level plotted.

#### Returns

**title** [str] Plot title.

**get\_dataset** (*self*)

Creating an xarray dataset from a radar object. This function has been removed from Py-ART ARM-DOE

**plot\_colorbar** (*self, mappable=None, orientation='horizontal', label=None, cax=None, ax=None, fig=None, field=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\_crosshairs** (*self, lon=None, lat=None, linestyle='--', color='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.

**linestyle** [str] Matplotlib string describing the line style.

**color** [str] Matplotlib string for color of the line.

**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** (*self*, *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', *ax*=None, *fig*=None, *lat\_lines*=None, *lon\_lines*=None, *projection*=None, *embelish*=True, *maps\_list*=['countries', 'coastlines'], *resolution*='110m', *alpha*=None, *background\_zoom*=8, *ticks*=None, *ticklabs*=None, *imshow*=False, *\*\*kwargs*)

Plot the grid using xarray and cartopy.

Additional arguments are passed to Xarray's pcolormesh function.

### Parameters

**field** [str] Field to be plotted.

**level** [int] Index corresponding to the height level to be plotted.

### Other Parameters

**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 used for luminance scaling.

**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 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 will use the default generated from the field and level parameters. Parameter is ignored if the title\_flag is False.

**title\_flag** [bool] True to add title to 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.

**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] Location at which to draw latitude and longitude lines. None will use default values which are reasonable for maps of North America.

**projection** [cartopy.crs class] Map projection supported by cartopy. Used for all subsequent calls to the GeoAxes object generated. Defaults to PlateCarree.



**embelish** [bool] True by default. Set to False to suppress drawing of coastlines etc... Use for speedup when specifying shapefiles. Note that lat lon labels only work with certain projections.

**maps\_list: list of strings** if embelish is true the list of maps to use. default countries, coastlines

**resolution** ['10m', '50m', '110m'.] Resolution of NaturalEarthFeatures to use. See Cartopy documentation for details.

**alpha** [float or None] Set the alpha transparency of the grid plot. Useful for overplotting radar over other datasets.

**background\_zoom** [int] Zoom of the background image. A highest number provides more detail at the cost of processing speed

**ticks** [array] Colorbar custom tick label locations.

**ticklabs** [array] Colorbar custom tick labels.

**imshow** [bool] If used, plot uses ax.imshow instead of ax.pcolormesh. Default is False.

```
plot_grid_contour (self, field, level=0, vmin=None, vmax=None, mask_outside=False,
                    title=None, title_flag=True, ax=None, fig=None, lat_lines=None,
                    lon_lines=None, projection=None, contour_values=None, linewidths=1.5,
                    embelish=True, maps_list=['countries', 'coastlines'], resolution='110m',
                    background_zoom=8, **kwargs)
```

Plot the grid contour using xarray and cartopy.

Additional arguments are passed to Xarray's pcolormesh function.

#### Parameters

**field** [str] Field to be plotted.

**level** [int] Index corresponding to the height level to be plotted.

#### Other Parameters

**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 used for luminance scaling.

**mask\_outside** [bool] True to mask data outside of vmin, vmax. False performs no masking.

**title** [str] Title to label plot with, None will use the default generated from the field and level parameters. Parameter is ignored if the title\_flag is False.

**title\_flag** [bool] True to add title to plot, False does not add a title.

**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] Location at which to draw latitude and longitude lines. None will use default values which are reasonable for maps of North America.

**projection** [cartopy.crs class] Map projection supported by cartopy. Used for all subsequent calls to the GeoAxes object generated. Defaults to PlateCarree.

**contour\_values** [float array]

list of contours to plot

**linewidths** [float] width of the contour lines

**embelish** [bool] True by default. Set to False to suppress drawing of coastlines etc... Use for speedup when specifying shapefiles. Note that lat lon labels only work with certain projections.

**maps\_list: list of strings** if embelish is true the list of maps to use. default countries, coastlines

**resolution** ['10m', '50m', '110m'.] Resolution of NaturalEarthFeatures to use. See Cartopy documentation for details.

**background\_zoom** [int] Zoom of the background image. A highest number provides more detail at the cost of processing speed

**plot\_latitude\_slice** (*self, 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** (*self, 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, ticks=None, ticklabs=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 if `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.

**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 themselves as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are 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.

**ticks** [array] Colorbar custom tick label locations.

**ticklabs** [array] Colorbar custom tick labels.

**plot\_latlon\_level** (*self, field, ind\_1, ind\_2, 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, ticks=None, ticklabs=None, \*\*kwargs*)

Plot a slice along two points given by its lat, lon Additional arguments are passed to Basemaps's pcolormesh function. Parameters ——— field : str

Field to be plotted.

**ind\_1, ind\_2** [float] x,y indices of the two points crossed by the slice.

**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 if 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.

**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 themselves as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are 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\_latlon\_slice** (*self*, *field*, *coord1*=None, *coord2*=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.

**coord1, coord2** [tuple of floats] tuple of floats containing the longitude and latitude (in degrees) specifying the two points crossed by the slice. If none two extremes of the grid is used

**plot\_longitude\_slice** (*self*, *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** (*self*, *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, *ticks*=None, *ticklabs*=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 if 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.

**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 themselves 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.

**class** `pyart.graph.GridMapDisplayBasemap` (*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 suppress them.

#### Attributes

**grid** [Grid] Grid object.

**debug** [bool] True to print debugging messages, False to suppressed 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 color-bars.

**fields** [list] List of fields which have been plotted.

#### Methods

<code>generate_filename(self, field, level[, ext])</code>	Generate a filename for a grid plot.
<code>generate_grid_title(self, field, level)</code>	Generate a title for a plot.
<code>generate_latitudinal_level_title(self, ...)</code>	Generate a title for a plot.
<code>generate_longitudinal_level_title(self, ...)</code>	Generate a title for a plot.
<code>get_basemap(self)</code>	get basemap of the plot
<code>plot_basemap(self[, lat_lines, lon_lines, ...])</code>	Plot a basemap.

Continued on next page

Table 5 – continued from previous page

<code>plot_colorbar(self[, mappable, orientation, ...])</code>	Plot a colorbar.
<code>plot_crosshairs(self[, lon, lat, ...])</code>	Plot crosshairs at a given longitude and latitude.
<code>plot_grid(self, field[, level, vmin, vmax, ...])</code>	Plot the grid onto the current basemap.
<code>plot_latitude_slice(self, field[, lon, lat])</code>	Plot a slice along a given latitude.
<code>plot_latitudinal_level(self, field, y_index)</code>	Plot a slice along a given latitude.
<code>plot_longitude_slice(self, field[, lon, lat])</code>	Plot a slice along a given longitude.
<code>plot_longitudinal_level(self, field, x_index)</code>	Plot a slice along a given longitude.

```

__class__
    alias of builtins.type

__delattr__(self, name, /)
    Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyart.graph.gridmapdisplay_basemap', '__doc__
__dir__(self, /)
    Default dir() implementation.

__eq__(self, value, /)
    Return self==value.

__format__(self, format_spec, /)
    Default object formatter.

__ge__(self, value, /)
    Return self>=value.

__getattr__(self, name, /)
    Return getattr(self, name).

__gt__(self, value, /)
    Return self>value.

__hash__(self, /)
    Return hash(self).

__init__(self, grid, debug=False)
    initialize the object.

__init_subclass__()
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be overridden to extend subclasses.

__le__(self, value, /)
    Return self<=value.

__lt__(self, value, /)
    Return self<value.

__module__ = 'pyart.graph.gridmapdisplay_basemap'

__ne__(self, value, /)
    Return self!=value.

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

```

**\_\_reduce\_\_** (*self*, /)  
Helper for pickle.

**\_\_reduce\_ex\_\_** (*self*, *protocol*, /)  
Helper for pickle.

**\_\_repr\_\_** (*self*, /)  
Return repr(*self*).

**\_\_setattr\_\_** (*self*, *name*, *value*, /)  
Implement setattr(*self*, *name*, *value*).

**\_\_sizeof\_\_** (*self*, /)  
Size of object in memory, in bytes.

**\_\_str\_\_** (*self*, /)  
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** (*self*, *lon*, *lat*)  
Find the nearest x, y grid indices for a given latitude and longitude.

**\_get\_label\_x** (*self*)  
Get default label for x units.

**\_get\_label\_y** (*self*)  
Get default label for y units.

**\_get\_label\_z** (*self*)  
Get default label for z units.

**\_label\_axes\_grid** (*self*, *axis\_labels*, *ax*)  
Set the x and y axis labels for a grid plot.

**\_label\_axes\_latitude** (*self*, *axis\_labels*, *ax*)  
Set the x and y axis labels for a latitude slice.

**\_label\_axes\_longitude** (*self*, *axis\_labels*, *ax*)  
Set the x and y axis labels for a longitude slice.

**\_make\_basemap** (*self*, *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** (*self, 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** (*self, field, level*)

Generate a title for a plot.

#### Parameters

**field** [str] Field plotted.

**level** [int] Vertical level plotted.

#### Returns

**title** [str] Plot title.

**generate\_latitudinal\_level\_title** (*self, 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** (*self, 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** (*self*)

get basemap of the plot



```
plot_basemap (self, 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 reasonable 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 (self, mappable=None, orientation='horizontal', label=None, cax=None, ax=None,  
               fig=None, field=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_crosshairs (self, 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(self, field, level=0, vmin=None, vmax=None, norm=None, cmap=None,  
           mask_outside=False, title=None, title_flag=True, axislabels=(None, None), axisla-  
           bels_flag=False, colorbar_flag=True, colorbar_label=None, colorbar_orient='vertical',  
           edges=True, ax=None, fig=None, ticks=None, ticklabs=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 if 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.

**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 themselves as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are 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.

**ticks** [array] Colorbar custom tick label locations.

**ticklabs** [array] Colorbar custom tick labels.

```
plot_latitude_slice(self, 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** (*self, 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, ticks=None, ticklabs=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 if 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 themselves as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are 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.

**ticks** [array] Colorbar custom tick label locations.

**ticklabs** [array] Colorbar custom tick labels.

**plot\_longitude\_slice** (*self, 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** (*self, 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, ticks=None, ticklabs=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 if 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 themselves as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are 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.

**ticks** [array] Colorbar custom tick label locations.

**ticklabs** [array] Colorbar custom tick labels.

**class** `pyart.graph.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

<code>generate_az_rhi_title</code> (self, field, azimuth)	Generate a title for a ray plot.
<code>generate_filename</code> (self, field, sweep[, ext, ...])	Generate a filename for a plot.
<code>generate_ray_title</code> (self, field, ray)	Generate a title for a ray plot.
<code>generate_title</code> (self, field, sweep[, ...])	Generate a title for a plot.
<code>generate_vpt_title</code> (self, field)	Generate a title for a VPT plot.
<code>label_xaxis_r</code> (self[, ax])	Label the xaxis with the default label for r units.
<code>label_xaxis_rays</code> ([ax])	Label the yaxis with the default label for rays.
<code>label_xaxis_time</code> ([ax])	Label the yaxis with the default label for rays.
<code>label_xaxis_x</code> (self[, ax])	Label the xaxis with the default label for x units.
<code>label_yaxis_field</code> (self, field[, ax])	Label the yaxis with the default label for a field units.
<code>label_yaxis_y</code> (self[, ax])	Label the yaxis with the default label for y units.
<code>label_yaxis_z</code> (self[, ax])	Label the yaxis with the default label for z units.
<code>plot</code> (self, field[, sweep])	Create a plot appropriate for the radar.

Continued on next page

Table 6 – continued from previous page

<code>plot_azimuth_to_rhi(self, field, tar-</code> <code>get_azimuth)</code>	Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.
<code>plot_colorbar(self[, mappable, field, ...])</code>	Plot a colorbar.
<code>plot_cross_hair(size[, npts, ax])</code>	Plot a cross-hair on a ppi plot.
<code>plot_grid_lines([ax, col, ls])</code>	Plot grid lines.
<code>plot_label(self, label, location[, symbol, ...])</code>	Plot a single symbol and label at a given location.
<code>plot_labels(self, labels, locations[, ...])</code>	Plot symbols and labels at given locations.
<code>plot_ppi(self, field[, sweep, mask_tuple, ...])</code>	Plot a PPI.
<code>plot_range_ring(range_ring_location_km[,</code> <code>...])</code>	Plot a single range ring.
<code>plot_range_rings(self, range_rings[, ax, ...])</code>	Plot a series of range rings.
<code>plot_ray(self, field, ray[, format_str, ...])</code>	Plot a single ray.
<code>plot_rhi(self, field[, sweep, mask_tuple, ...])</code>	Plot a RHI.
<code>plot_vpt(self, field[, mask_tuple, vmin, ...])</code>	Plot a VPT scan.
<code>set_aspect_ratio([aspect_ratio, ax])</code>	Set the aspect ratio for plot area.
<code>set_limits([xlim, ylim, ax])</code>	Set the display limits.

**\_\_class\_\_**alias of `builtins.type`**\_\_delattr\_\_** (*self, name, /*)Implement `delattr(self, name)`.**\_\_dict\_\_** = `mappingproxy({'__module__': 'pyart.graph.radardisplay', '__doc__': '\n A`**\_\_dir\_\_** (*self, /*)Default `dir()` implementation.**\_\_eq\_\_** (*self, value, /*)Return `self==value`.**\_\_format\_\_** (*self, format\_spec, /*)

Default object formatter.

**\_\_ge\_\_** (*self, value, /*)Return `self>=value`.**\_\_getattr\_\_** (*self, name, /*)Return `getattr(self, name)`.**\_\_gt\_\_** (*self, value, /*)Return `self>value`.**\_\_hash\_\_** (*self, /*)Return `hash(self)`.**\_\_init\_\_** (*self, radar, shift=(0.0, 0.0)*)

Initialize the object.

**\_\_init\_subclass\_\_** ()

This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

**\_\_le\_\_** (*self, value, /*)Return `self<=value`.**\_\_lt\_\_** (*self, value, /*)Return `self<value`.

---

```

__module__ = 'pyart.graph.radardisplay'

__ne__(self, value, /)
    Return self!=value.

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

__reduce__(self, /)
    Helper for pickle.

__reduce_ex__(self, protocol, /)
    Helper for pickle.

__repr__(self, /)
    Return repr(self).

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

__sizeof__(self, /)
    Size of object in memory, in bytes.

__str__(self, /)
    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(self, field, target_azimuth, edges, mask_tuple, filter_transitions,
                             gatefilter)
    Retrieve and return pseudo-RHI data from a plot function.

__get_colorbar_label(self, field)
    Return a colorbar label for a given field.

__get_data(self, field, sweep, mask_tuple, filter_transitions, gatefilter)
    Retrieve and return data from a plot function.

__get_ray_data(self, field, ray, mask_tuple, gatefilter)
    Retrieve and return ray data from a plot function.

__get_vpt_data(self, field, mask_tuple, filter_transitions, gatefilter)
    Retrieve and return vpt data from a plot function.

__get_x_y(self, sweep, edges, filter_transitions)
    Retrieve and return x and y coordinate in km.

__get_x_y_z(self, sweep, edges, filter_transitions)
    Retrieve and return x, y, and z coordinate in km.

__get_x_z(self, sweep, edges, filter_transitions)
    Retrieve and return x and z coordinate in km.

__label_axes_ppi(self, axis_labels, ax)
    Set the x and y axis labels for a PPI plot.

```

**`_label_axes_ray`** (*self*, *axis\_labels*, *field*, *ax*)

Set the x and y axis labels for a ray plot.

**`_label_axes_rhi`** (*self*, *axis\_labels*, *ax*)

Set the x and y axis labels for a RHI plot.

**`_label_axes_vpt`** (*self*, *axis\_labels*, *time\_axis\_flag*, *ax*)

Set the x and y axis labels for a PPI plot.

**`_set_az_rhi_title`** (*self*, *field*, *azimuth*, *title*, *ax*)

Set the figure title for a ray plot using a default title.

**`_set_ray_title`** (*self*, *field*, *ray*, *title*, *ax*)

Set the figure title for a ray plot using a default title.

**`_set_title`** (*self*, *field*, *sweep*, *title*, *ax*, *datetime\_format=None*, *use\_sweep\_time=True*)

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`** (*self*, *field*, *title*, *ax*)

Set the figure title using a default title.

**`generate_az_rhi_title`** (*self*, *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`** (*self*, *field*, *sweep*, *ext='png'*, *datetime\_format='%Y%m%d%H%M%S'*,  
*use\_sweep\_time=False*)

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.

**`datetime_format`** [str] Format of datetime (using `strftime` format).

**`use_sweep_time`** [bool] If true, the current sweep's beginning time is used.

#### Returns

**`filename`** [str] Filename suitable for saving a plot.



**generate\_ray\_title** (*self, 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** (*self, field, sweep, datetime\_format=None, use\_sweep\_time=True*)

Generate a title for a plot.

**Parameters**

**field** [str] Field plotted.

**sweep** [int] Sweep plotted.

**datetime\_format** [str] Format of datetime (using strftime format).

**use\_sweep\_time** [bool] If true, the current sweep's beginning time is used.

**Returns**

**title** [str] Plot title.

**generate\_vpt\_title** (*self, field*)

Generate a title for a VPT plot.

**Parameters**

**field** [str] Field plotted.

**Returns**

**title** [str] Plot title.

**label\_xaxis\_r** (*self, 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** (*self, ax=None*)

Label the xaxis with the default label for x units.

**label\_yaxis\_field** (*self, field, ax=None*)

Label the yaxis with the default label for a field units.

**label\_yaxis\_y** (*self, ax=None*)

Label the yaxis with the default label for y units.

**label\_yaxis\_z** (*self, ax=None*)

Label the yaxis with the default label for z units.

**plot** (*self, field, sweep=0, \*\*kwargs*)

Create a plot appropriate 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 appropriate 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` (*self, 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, raster=False, \*\*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 themselves 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.

**raster** [bool] False by default. Set to True to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**plot\_colorbar** (*self*, *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** (*self, 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** (*self, 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** (*self, 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, raster=False, title\_datetime\_format=None, title\_use\_sweep\_time=True, \*\*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\_datetime\_format** [str] Format of datetime in the title (using strftime format).

**title\_use\_sweep\_time** [bool] True for the current sweep's beginning time to be used for the title. False for the radar's beginning time.

**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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure

to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**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** (*self*, *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** (*self*, *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 (self, 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, raster=False, title_datetime_format=None,
          title_use_sweep_time=True, **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\_datetime\_format** [str] Format of datetime in the title (using strftime format).

**title\_use\_sweep\_time** [bool] True for the current sweep's beginning time to be used for the title. False for the radar's beginning time.

**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 west to east, False to have east to west. None (the default) will reverse the axis only when all the distances are negative. (i.e) axis will be absolute distance without taking into consideration the orientation

**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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**plot\_vpt** (*self*, *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*, *gatefilter=None*, *filter\_transitions=True*, *time\_axis\_flag=False*, *date\_time\_form=None*, *tz=None*, *ax=None*, *fig=None*, *ticks=None*, *ticklabs=None*, *raster=False*, *\*\*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 themselves 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.

**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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to `pcolormesh`. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**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.

**class** `pyart.graph.RadarMapDisplay` (*radar*, *shift*=(0.0, 0.0), *grid\_projection*=None)

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.

**grid\_projection** [cartopy.crs] AzimuthalEquidistant cartopy projection centered on radar. Used to transform points into map projection.

#### Methods

<code>generate_az_rhi_title(self, field, azimuth)</code>	Generate a title for a ray plot.
<code>generate_filename(self, field, sweep[, ext, ...])</code>	Generate a filename for a plot.
<code>generate_ray_title(self, field, ray)</code>	Generate a title for a ray plot.
<code>generate_title(self, field, sweep[, ...])</code>	Generate a title for a plot.
<code>generate_vpt_title(self, field)</code>	Generate a title for a VPT plot.
<code>label_xaxis_r(self[, ax])</code>	Label the xaxis with the default label for r units.
<code>label_xaxis_rays([ax])</code>	Label the yaxis with the default label for rays.
<code>label_xaxis_time([ax])</code>	Label the yaxis with the default label for rays.
<code>label_xaxis_x(self[, ax])</code>	Label the xaxis with the default label for x units.
<code>label_yaxis_field(self, field[, ax])</code>	Label the yaxis with the default label for a field units.
<code>label_yaxis_y(self[, ax])</code>	Label the yaxis with the default label for y units.
<code>label_yaxis_z(self[, ax])</code>	Label the yaxis with the default label for z units.
<code>plot(self, field[, sweep])</code>	Create a plot appropriate for the radar.

Continued on next page

Table 7 – continued from previous page

<code>plot_azimuth_to_rhi(self, field, tar-</code> <code>get_azimuth)</code>	Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.
<code>plot_colorbar(self[, mappable, field, ...])</code>	Plot a colorbar.
<code>plot_cross_hair(size[, npts, ax])</code>	Plot a cross-hair on a ppi plot.
<code>plot_grid_lines([ax, col, ls])</code>	Plot grid lines.
<code>plot_label(self, label, location[, symbol, ...])</code>	Plot a single symbol and label at a given location.
<code>plot_labels(self, labels, locations[, ...])</code>	Plot symbols and labels at given locations.
<code>plot_line_geo(self, line_lons, line_lats[, ...])</code>	Plot a line segments on the current map given values in lat and lon.
<code>plot_line_xy(self, line_x, line_y[, line_style])</code>	Plot a line segments on the current map given radar x, y values.
<code>plot_point(self, lon, lat[, symbol, ...])</code>	Plot a point on the current map.
<code>plot_ppi(self, field[, sweep, mask_tuple, ...])</code>	Plot a PPI.
<code>plot_ppi_map(self, field[, sweep, ...])</code>	Plot a PPI volume sweep onto a geographic map.
<code>plot_range_ring(self,</code> <code>range_ring_location_km)</code>	Plot a single range ring on the map.
<code>plot_range_rings(self, range_rings[, ax, ...])</code>	Plot a series of range rings.
<code>plot_ray(self, field, ray[, format_str, ...])</code>	Plot a single ray.
<code>plot_rhi(self, field[, sweep, mask_tuple, ...])</code>	Plot a RHI.
<code>plot_vpt(self, field[, mask_tuple, vmin, ...])</code>	Plot a VPT scan.
<code>set_aspect_ratio([aspect_ratio, ax])</code>	Set the aspect ratio for plot area.
<code>set_limits([xlim, ylim, ax])</code>	Set the display limits.

**\_\_class\_\_**alias of `builtins.type`**\_\_delattr\_\_** (*self, name, /*)Implement `delattr(self, name)`.**\_\_dict\_\_** = `mappingproxy({'__module__': 'pyart.graph.radarmapdisplay', '__doc__': '\n`**\_\_dir\_\_** (*self, /*)Default `dir()` implementation.**\_\_eq\_\_** (*self, value, /*)Return `self==value`.**\_\_format\_\_** (*self, format\_spec, /*)

Default object formatter.

**\_\_ge\_\_** (*self, value, /*)Return `self>=value`.**\_\_getattr\_\_** (*self, name, /*)Return `getattr(self, name)`.**\_\_gt\_\_** (*self, value, /*)Return `self>value`.**\_\_hash\_\_** (*self, /*)Return `hash(self)`.**\_\_init\_\_** (*self, radar, shift=(0.0, 0.0), grid\_projection=None*)

Initialize the object.

**\_\_init\_subclass\_\_** ()

This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

```
__le__ (self, value, /)
    Return self<=value.

__lt__ (self, value, /)
    Return self<value.

__module__ = 'pyart.graph.radarmapdisplay'

__ne__ (self, value, /)
    Return self!=value.

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

__reduce__ (self, /)
    Helper for pickle.

__reduce_ex__ (self, protocol, /)
    Helper for pickle.

__repr__ (self, /)
    Return repr(self).

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

__sizeof__ (self, /)
    Size of object in memory, in bytes.

__str__ (self, /)
    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_ax (self)
    Check that a GeoAxes object exists, raise ValueError if not

_get_azimuth_rhi_data_x_y_z (self, field, target_azimuth, edges, mask_tuple, filter_transitions,
                               gatefilter)
    Retrieve and return pseudo-RHI data from a plot function.

_get_colorbar_label (self, field)
    Return a colorbar label for a given field.

_get_data (self, field, sweep, mask_tuple, filter_transitions, gatefilter)
    Retrieve and return data from a plot function.

_get_ray_data (self, field, ray, mask_tuple, gatefilter)
    Retrieve and return ray data from a plot function.

_get_vpt_data (self, field, mask_tuple, filter_transitions, gatefilter)
    Retrieve and return vpt data from a plot function.

_get_x_y (self, sweep, edges, filter_transitions)
    Retrieve and return x and y coordinate in km.
```

**`_get_x_y_z`** (*self, sweep, edges, filter\_transitions*)  
Retrieve and return x, y, and z coordinate in km.

**`_get_x_z`** (*self, sweep, edges, filter\_transitions*)  
Retrieve and return x and z coordinate in km.

**`_label_axes_ppi`** (*self, axis\_labels, ax*)  
Set the x and y axis labels for a PPI plot.

**`_label_axes_ray`** (*self, axis\_labels, field, ax*)  
Set the x and y axis labels for a ray plot.

**`_label_axes_rhi`** (*self, axis\_labels, ax*)  
Set the x and y axis labels for a RHI plot.

**`_label_axes_vpt`** (*self, axis\_labels, time\_axis\_flag, ax*)  
Set the x and y axis labels for a PPI plot.

**`_set_az_rhi_title`** (*self, field, azimuth, title, ax*)  
Set the figure title for a ray plot using a default title.

**`_set_ray_title`** (*self, field, ray, title, ax*)  
Set the figure title for a ray plot using a default title.

**`_set_title`** (*self, field, sweep, title, ax, datetime\_format=None, use\_sweep\_time=True*)  
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`** (*self, field, title, ax*)  
Set the figure title using a default title.

**`generate_az_rhi_title`** (*self, 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`** (*self, field, sweep, ext='png', datetime\_format='%Y%m%d%H%M%S', use\_sweep\_time=False*)  
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.

**datetime\_format** [str] Format of datetime (using strftime format).

**use\_sweep\_time** [bool] If true, the current sweep's beginning time is used.

**Returns**

**filename** [str] Filename suitable for saving a plot.

**generate\_ray\_title** (*self*, *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** (*self*, *field*, *sweep*, *datetime\_format=None*, *use\_sweep\_time=True*)

Generate a title for a plot.

**Parameters**

**field** [str] Field plotted.

**sweep** [int] Sweep plotted.

**datetime\_format** [str] Format of datetime (using strftime format).

**use\_sweep\_time** [bool] If true, the current sweep's beginning time is used.

**Returns**

**title** [str] Plot title.

**generate\_vpt\_title** (*self*, *field*)

Generate a title for a VPT plot.

**Parameters**

**field** [str] Field plotted.

**Returns**

**title** [str] Plot title.

**label\_xaxis\_r** (*self*, *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** (*self*, *ax=None*)

Label the xaxis with the default label for x units.

**label\_yaxis\_field** (*self*, *field*, *ax=None*)

Label the yaxis with the default label for a field units.

**label\_yaxis\_y** (*self*, *ax=None*)

Label the yaxis with the default label for y units.

**label\_yaxis\_z** (*self*, *ax=None*)

Label the yaxis with the default label for z units.

**plot** (*self*, *field*, *sweep=0*, *\*\*kwargs*)

Create a plot appropriate 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 appropriate 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** (*self*, *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*, *raster=False*, *\*\*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 if norm is not None.

**vmax** [float] Luminance maximum value, None for default value. Parameter is ignored if 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 themselves 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.

**raster** [bool] False by default. Set to True to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**plot\_colorbar** (*self*, *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** (*self, 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** (*self, 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** (*self, 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 ax.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** (*self*, *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 `ax.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** (*self*, *lon*, *lat*, *symbol*='ro', *label\_text*=None, *label\_offset*=(None, None), *\*\*kwargs*)

Plot a point on the current map.

Additional arguments are passed to `ax.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.

**plot\_ppi** (*self*, *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, *raster*=False, *title\_datetime\_format*=None, *title\_use\_sweep\_time*=True, *\*\*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\_datetime\_format** [str] Format of datetime in the title (using strftime format).

**title\_use\_sweep\_time** [bool] True for the current sweep's beginning time to be used for the title. False for the radar's beginning time.

**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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

```
plot_ppi_map(self, 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, pro-
              jection=None, min_lon=None, max_lon=None, min_lat=None, max_lat=None,
              width=None, height=None, lon_0=None, lat_0=None, resolution='110m',
              shapefile=None, shapefile_kwargs=None, edges=True, gatefilter=None, fil-
              ter_transitions=True, embellish=True, maps_list=['countries', 'coastlines'],
              raster=False, ticks=None, ticklabs=None, alpha=None, background_zoom=8)
```

Plot a PPI volume sweep onto a geographic map.

#### 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** [Cartopy GeoAxes instance] If None, create GeoAxes instance using other keyword info. If provided, ax must have a Cartopy crs projection and projection kwarg below is ignored.

**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** [cartopy.crs class] Map projection supported by cartopy. Used for all subsequent calls to the GeoAxes object generated. Defaults to LambertConformal centered on radar.

**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.

**shapefile** [str] Filename for a shapefile to add to map.

**shapefile\_kwargs** [dict] Key word arguments used to format shapefile. Projection defaults to lat lon (cartopy.crs.PlateCarree())

**resolution** ['10m', '50m', '110m'.] Resolution of NaturalEarthFeatures to use. See Cartopy 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 themselves as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not plotted.

**embelish: bool** True by default. Set to False to suppress drawing of coastlines etc.. Use for speedup when specifying shapefiles. Note that lat lon labels only work with certain projections.

**maps\_list: list of strings** if embelish is true the list of maps to use. default countries, coastlines

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**alpha** [float or None] Set the alpha transparency of the radar plot. Useful for overplotting radar over other datasets.

**background\_zoom** [int] Zoom of the background image. A highest number provides more detail at the cost of processing speed

**plot\_range\_ring** (*self*, *range\_ring\_location\_km*, *npts=360*, *line\_style='k-'*, *\*\*kwargs*)

Plot a single range ring on the map.

Additional arguments are passed to ax.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** (*self*, *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** (*self*, *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)*, *gate\_filter=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** (*self, 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, raster=False, title\_datetime\_format=None, title\_use\_sweep\_time=True, \*\*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\_datetime\_format** [str] Format of datetime in the title (using strftime format).

**title\_use\_sweep\_time** [bool] True for the current sweep's beginning time to be used for the title. False for the radar's beginning time.

**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 west to east, False to have east to west. None (the default) will reverse the axis only when all the distances are negative. (i.e) axis will be absolute distance without taking into consideration the orientation

**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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

```
plot_vpt (self, field, mask_tuple=None, vmin=None, vmax=None, norm=None, cmap=None,
mask_outside=False, title=None, title_flag=True, axislabels=(None, None), axisla-
bels_flag=True, colorbar_flag=True, colorbar_label=None, colorbar_orient='vertical',
edges=True, gatefilter=None, filter_transitions=True, time_axis_flag=False,
date_time_form=None, tz=None, ax=None, fig=None, ticks=None, ticklabs=None,
raster=False, **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 themselves 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.

**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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to `pcolormesh`. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**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.

**class** `pyart.graph.RadarMapDisplayBasemap` (*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

<code>generate_az_rhi_title(self, field, azimuth)</code>	Generate a title for a ray plot.
<code>generate_filename(self, field, sweep[, ext, ...])</code>	Generate a filename for a plot.
<code>generate_ray_title(self, field, ray)</code>	Generate a title for a ray plot.
<code>generate_title(self, field, sweep[, ...])</code>	Generate a title for a plot.
<code>generate_vpt_title(self, field)</code>	Generate a title for a VPT plot.
<code>label_xaxis_r(self[, ax])</code>	Label the xaxis with the default label for r units.
<code>label_xaxis_rays([ax])</code>	Label the yaxis with the default label for rays.
<code>label_xaxis_time([ax])</code>	Label the yaxis with the default label for rays.
<code>label_xaxis_x(self[, ax])</code>	Label the xaxis with the default label for x units.
<code>label_yaxis_field(self, field[, ax])</code>	Label the yaxis with the default label for a field units.
<code>label_yaxis_y(self[, ax])</code>	Label the yaxis with the default label for y units.
<code>label_yaxis_z(self[, ax])</code>	Label the yaxis with the default label for z units.
<code>plot(self, field[, sweep])</code>	Create a plot appropriate for the radar.
<code>plot_azimuth_to_rhi(self, field, target_azimuth)</code>	Plot pseudo-RHI scan by extracting the vertical field associated with the given azimuth.
<code>plot_colorbar(self[, mappable, field, ...])</code>	Plot a colorbar.
<code>plot_cross_hair(size[, npts, ax])</code>	Plot a cross-hair on a ppi plot.
<code>plot_grid_lines([ax, col, ls])</code>	Plot grid lines.
<code>plot_label(self, label, location[, symbol, ...])</code>	Plot a single symbol and label at a given location.
<code>plot_labels(self, labels, locations[, ...])</code>	Plot symbols and labels at given locations.
<code>plot_line_geo(self, line_lons, line_lats[, ...])</code>	Plot a line segments on the current map given values in lat and lon.
<code>plot_line_xy(self, line_x, line_y[, line_style])</code>	Plot a line segments on the current map given radar x, y values.
<code>plot_point(self, lon, lat[, symbol, ...])</code>	Plot a point on the current map.
<code>plot_ppi(self, field[, sweep, mask_tuple, ...])</code>	Plot a PPI.
<code>plot_ppi_map(self, field[, sweep, ...])</code>	Plot a PPI volume sweep onto a geographic map.
<code>plot_range_ring(self, range_ring_location_km)</code>	Plot a single range ring on the map.
<code>plot_range_rings(self, range_rings[, ax, ...])</code>	Plot a series of range rings.
<code>plot_ray(self, field, ray[, format_str, ...])</code>	Plot a single ray.
<code>plot_rhi(self, field[, sweep, mask_tuple, ...])</code>	Plot a RHI.
<code>plot_vpt(self, field[, mask_tuple, vmin, ...])</code>	Plot a VPT scan.
<code>set_aspect_ratio([aspect_ratio, ax])</code>	Set the aspect ratio for plot area.
<code>set_limits([xlim, ylim, ax])</code>	Set the display limits.

`__class__`

alias of `builtins.type`

`__delattr__(self, name, /)`

Implement `delattr(self, name)`.

`__dict__ = mappingproxy({'__module__': 'pyart.graph.radarmapdisplay_base', '__doc__': ...})`

`__dir__(self, /)`

Default `dir()` implementation.

`__eq__(self, value, /)`

Return `self==value`.

`__format__(self, format_spec, /)`

Default object formatter.

**\_\_ge\_\_** (*self, value, /*)  
Return self>=value.

**\_\_getattr\_\_** (*self, name, /*)  
Return getattr(self, name).

**\_\_gt\_\_** (*self, value, /*)  
Return self>value.

**\_\_hash\_\_** (*self, /*)  
Return hash(self).

**\_\_init\_\_** (*self, radar, shift=(0.0, 0.0)*)  
Initialize the object.

**\_\_init\_subclass\_\_** ()  
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

**\_\_le\_\_** (*self, value, /*)  
Return self<=value.

**\_\_lt\_\_** (*self, value, /*)  
Return self<value.

**\_\_module\_\_** = 'pyart.graph.radarmapdisplay\_basemap'

**\_\_ne\_\_** (*self, value, /*)  
Return self!=value.

**\_\_new\_\_** (*\*args, \*\*kwargs*)  
Create and return a new object. See help(type) for accurate signature.

**\_\_reduce\_\_** (*self, /*)  
Helper for pickle.

**\_\_reduce\_ex\_\_** (*self, protocol, /*)  
Helper for pickle.

**\_\_repr\_\_** (*self, /*)  
Return repr(self).

**\_\_setattr\_\_** (*self, name, value, /*)  
Implement setattr(self, name, value).

**\_\_sizeof\_\_** (*self, /*)  
Size of object in memory, in bytes.

**\_\_str\_\_** (*self, /*)  
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`** (*self*)  
Check that basemap is not None, raise ValueError if it is.

**`_get_azimuth_rhi_data_x_y_z`** (*self, field, target\_azimuth, edges, mask\_tuple, filter\_transitions, gatefilter*)  
Retrieve and return pseudo-RHI data from a plot function.

**`_get_colorbar_label`** (*self, field*)  
Return a colorbar label for a given field.

**`_get_data`** (*self, field, sweep, mask\_tuple, filter\_transitions, gatefilter*)  
Retrieve and return data from a plot function.

**`_get_ray_data`** (*self, field, ray, mask\_tuple, gatefilter*)  
Retrieve and return ray data from a plot function.

**`_get_vpt_data`** (*self, field, mask\_tuple, filter\_transitions, gatefilter*)  
Retrieve and return vpt data from a plot function.

**`_get_x_y`** (*self, sweep, edges, filter\_transitions*)  
Retrieve and return x and y coordinate in km.

**`_get_x_y_z`** (*self, sweep, edges, filter\_transitions*)  
Retrieve and return x, y, and z coordinate in km.

**`_get_x_z`** (*self, sweep, edges, filter\_transitions*)  
Retrieve and return x and z coordinate in km.

**`_label_axes_ppi`** (*self, axis\_labels, ax*)  
Set the x and y axis labels for a PPI plot.

**`_label_axes_ray`** (*self, axis\_labels, field, ax*)  
Set the x and y axis labels for a ray plot.

**`_label_axes_rhi`** (*self, axis\_labels, ax*)  
Set the x and y axis labels for a RHI plot.

**`_label_axes_vpt`** (*self, axis\_labels, time\_axis\_flag, ax*)  
Set the x and y axis labels for a PPI plot.

**`_set_az_rhi_title`** (*self, field, azimuth, title, ax*)  
Set the figure title for a ray plot using a default title.

**`_set_ray_title`** (*self, field, ray, title, ax*)  
Set the figure title for a ray plot using a default title.

**`_set_title`** (*self, field, sweep, title, ax, datetime\_format=None, use\_sweep\_time=True*)  
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`** (*self, field, title, ax*)  
Set the figure title using a default title.

**`generate_az_rhi_title`** (*self, 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** (*self*, *field*, *sweep*, *ext*='png', *datetime\_format*='%Y%m%d%H%M%S',  
*use\_sweep\_time*=False)

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.**datetime\_format** [str] Format of datetime (using strftime format).**use\_sweep\_time** [bool] If true, the current sweep's beginning time is used.**Returns****filename** [str] Filename suitable for saving a plot.

**generate\_ray\_title** (*self*, *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** (*self*, *field*, *sweep*, *datetime\_format*=None, *use\_sweep\_time*=True)

Generate a title for a plot.

**Parameters****field** [str] Field plotted.**sweep** [int] Sweep plotted.**datetime\_format** [str] Format of datetime (using strftime format).**use\_sweep\_time** [bool] If true, the current sweep's beginning time is used.**Returns****title** [str] Plot title.

**generate\_vpt\_title** (*self*, *field*)

Generate a title for a VPT plot.

**Parameters****field** [str] Field plotted.

### Returns

**title** [str] Plot title.

**label\_xaxis\_r** (*self*, *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** (*self*, *ax=None*)

Label the xaxis with the default label for x units.

**label\_yaxis\_field** (*self*, *field*, *ax=None*)

Label the yaxis with the default label for a field units.

**label\_yaxis\_y** (*self*, *ax=None*)

Label the yaxis with the default label for y units.

**label\_yaxis\_z** (*self*, *ax=None*)

Label the yaxis with the default label for z units.

**plot** (*self*, *field*, *sweep=0*, *\*\*kwargs*)

Create a plot appropriate 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 appropriate 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** (*self*, *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*, *raster=False*, *\*\*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 themselves 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.

**raster** [bool] False by default. Set to True to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**plot\_colorbar** (*self*, *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** (*self*, *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** (*self*, *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** (*self*, *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** (*self*, *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** (*self*, *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** (*self*, *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, *raster*=False, *title\_datetime\_format*=None, *title\_use\_sweep\_time*=True, *\*\*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 if norm is not None.

**vmax** [float] Luminance maximum value, None for default value. Parameter is ignored if 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\_datetime\_format** [str] Format of datetime in the title (using strftime format).

**title\_use\_sweep\_time** [bool] True for the current sweep's beginning time to be used for the title. False for the radar's beginning time.

**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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

```
plot_ppi_map(self, 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, pro-
               jection='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, resolu-
               tion='h', shapefile=None, edges=True, gatefilter=None, basemap=None, fil-
               ter_transitions=True, embellish=True, ticks=None, ticklabs=None, raster=False, al-
               pha=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 reasonable 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<sup>2</sup> 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 themselves as the gate edges, resulting in a plot in which the last gate in each ray and the entire last ray are not plotted.

**embelish: bool** True by default. Set to false to suppress 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-specified basemap instance.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**alpha** [float or None] Set the alpha transparency of the radar plot. Useful for overplotting radar over other datasets.

**plot\_range\_ring** (*self*, *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** (*self, 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** (*self, 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 (self, 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, raster=False, title_datetime_format=None, title_use_sweep_time=True, **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 if norm is not None.

**vmax** [float] Luminance maximum value, None for default value. Parameter is ignored if 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\_datetime\_format** [str] Format of datetime in the title (using strftime format).

**title\_use\_sweep\_time** [bool] True for the current sweep's beginning time to be used for the title. False for the radar's beginning time.

**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 west to east, False to have east to west. None (the default) will reverse the axis only when all the distances are negative. (i.e) axis will be absolute distance without taking into consideration the orientation

**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 themselves 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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to pcolormesh. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

```
plot_vpt (self, field, mask_tuple=None, vmin=None, vmax=None, norm=None, cmap=None,
          mask_outside=False, title=None, title_flag=True, axislabels=(None, None), axisla-
          bels_flag=True, colorbar_flag=True, colorbar_label=None, colorbar_orient='vertical',
          edges=True, gatefilter=None, filter_transitions=True, time_axis_flag=False,
          date_time_form=None, tz=None, ax=None, fig=None, ticks=None, ticklabs=None,
          raster=False, **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 if norm is not None.

**vmax** [float] Luminance maximum value, None for default value. Parameter is ignored if 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 themselves 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.

**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.

**raster** [bool] False by default. Set to true to render the display as a raster rather than a vector in call to `pcolormesh`. Saves time in plotting high resolution data over large areas. Be sure to set the dpi of the plot for your application if you save it as a vector format (i.e., pdf, eps, svg).

**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.



## UTILITIES (PYART . UTIL)

Miscellaneous utility functions.

The location and names of these functions within Py-ART may change between versions without depreciation, use with caution.

### 10.1 Direction statistics

<i>angular_mean</i> (angles)	Compute the mean of a distribution of angles in radians.
<i>angular_std</i> (angles)	Compute the standard deviation of a distribution of angles in radians.
<i>angular_mean_deg</i> (angles)	Compute the mean of a distribution of angles in degrees.
<i>angular_std_deg</i> (angles)	Compute the standard deviation of a distribution of angles in degrees.
<i>interval_mean</i> (dist, interval_min, interval_max)	Compute the mean of a distribution within an interval.
<i>interval_std</i> (dist, interval_min, interval_max)	Compute the standard deviation of a distribution within an interval.
<i>mean_of_two_angles</i> (angles1, angles2)	Compute the element by element mean of two sets of angles.
<i>mean_of_two_angles_deg</i> (angle1, angle2)	Compute the element by element mean of two sets of angles in degrees.

### 10.2 Miscellaneous functions

<i>cross_section_ppi</i> (radar, target_azimuths[, ...])	Extract cross sections from a PPI volume along one or more azimuth angles.
<i>cross_section_rhi</i> (radar, target_elevations)	Extract cross sections from an RHI volume along one or more elevation angles.
<i>colocated_gates</i> (radar1, radar2[, h_tol, ...])	Flags radar gates of radar1 colocated with radar2
<i>intersection</i> (radar1, radar2[, h_tol, ...])	Flags region of radar1 that is intersecting with radar2 and complies with criteria regarding visibility, altitude, range, elevation angle and azimuth angle
<i>datetime_from_radar</i> (radar[, epoch])	Return a datetime for the first ray in a Radar.
<i>datetimes_from_radar</i> (radar[, epoch])	Return an array of datetimes for the rays in a Radar.
<i>datetime_from_dataset</i> (dataset[, epoch])	Return a datetime for the first time in a netCDF Dataset.
<i>datetimes_from_dataset</i> (dataset[, epoch])	Return an array of datetimes for the times in a netCDF Dataset.

Continued on next page

Table 2 – continued from previous page

<code>datetime_from_grid(grid[, epoch])</code>	Return a datetime for the volume start in a Grid.
<code>estimate_noise_hs74(spectrum[, navg, nnoise_min])</code>	Estimate noise parameters of a Doppler spectrum.
<code>estimate_noise_ivic13(pwr_w_ray[, pct, ...])</code>	Estimate noise parameters of a ray
<code>get_ivic_pct(npulses, pct, z[, prob_thr])</code>	Get the point clutter threshold (PCT)
<code>get_ivic_flat_reg_var_max(npulses, y, x, ...)</code>	Get the threshold for maximum local variance of noise [dB]
<code>get_ivic_snr_thr(npulses, snr_thr[, pfa_thr])</code>	Get the threshold for steps 3 and 6 of ivic
<code>ivic_pct_table(npulses)</code>	Get the point clutter threshold (PCT) of Ivic from a look up table.
<code>ivic_snr_thr_table(npulses)</code>	Get the threshold for steps 3 and 6 of ivic from a look up table The thresholds are computed for between 3 and 200 pulses.
<code>ivic_flat_reg_var_max_table(npulses, ...)</code>	Get maximum variance of noise of Ivic from a look up table.
<code>is_vpt(radar[, offset])</code>	Determine if a Radar appears to be a vertical pointing scan.
<code>to_vpt(radar[, single_scan])</code>	Convert an existing Radar object to represent a vertical pointing scan.
<code>join_radar(radar1, radar2)</code>	Combine two radar instances into one.
<code>join_spectra(spectral1, spectral2)</code>	Combine two spectra instances into one.
<code>cut_radar(radar, field_names[, rng_min, ...])</code>	Cuts the radar object into new dimensions
<code>cut_radar_spectra(radar, field_names[, ...])</code>	Cuts the radar spectra object into new dimensions
<code>radar_from_spectra(psr)</code>	obtain a Radar object from a RadarSpectra object
<code>interpol_spectra(psr[, kind, fill_value])</code>	Interpolates the spectra so that it has a uniform grid
<code>simulated_vel_from_profile(radar, profile[, ...])</code>	Create simulated radial velocities from a profile of horizontal winds.
<code>texture_along_ray(radar, var[, wind_size])</code>	Compute field texture along ray using a user specified window size.
<code>texture(radar, var)</code>	Determine a texture field using an 11pt stdev texar-ray=texture(pyradarobj, field).
<code>rolling_window(a, window)</code>	create a rolling window object for application of functions eg: result=np.ma.std(array, 11), 1)
<code>angular_texture_2d(image, N, interval)</code>	Compute the angular texture of an image.

`pyart.util.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.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.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.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.angular_texture_2d(image, N, interval)`

Compute the angular texture of an image. Uses convolutions in order to speed up texture calculation by a factor of ~50 compared to using `ndimage.generic_filter`.

**Parameters**

**image** [2D array of floats] The array containing the velocities in which to calculate texture from.

**N** [int] This is the window size for calculating texture. The texture will be calculated from an N by N window centered around the gate.

**interval** [float] The absolute value of the maximum velocity. In conversion to radial coordinates,  $\pi$  will be defined to be interval and  $-\pi$  will be  $-\text{interval}$ . It is recommended that interval be set to the Nyquist velocity.

**Returns**

**std\_dev** [float array] Texture of the radial velocity field.

`pyart.util.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]

**coloc\_gates\_field** [string] Name of the field to retrieve the data

**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, i.e: 1: not colocated gates 2: colocated (0 is reserved)

`pyart.util.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, optional] 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.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, optional] 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.cut_radar(radar, field_names, rng_min=None, rng_max=None, ele_min=None, ele_max=None, azi_min=None, azi_max=None)`

Cuts the radar object into new dimensions

#### Parameters

**radar** [radar object] The radar object containing the data

**field\_names** [str or None] The fields to keep in the new radar

**rng\_min, rng\_max** [float] The range limits [m]. If None the entire coverage of the radar is going to be used

**ele\_min, ele\_max, azi\_min, azi\_max** [float or None] The limits of the grid [deg]. If None the limits will be the limits of the radar volume

#### Returns

**radar** [radar object] The radar object containing only the desired data

`pyart.util.cut_radar_spectra(radar, field_names, rng_min=None, rng_max=None, ele_min=None, ele_max=None, azi_min=None, azi_max=None)`

Cuts the radar spectra object into new dimensions

#### Parameters

**radar** [radar object] The radar object containing the data

**field\_names** [str or None] The fields to keep in the new radar

**rng\_min, rng\_max** [float] The range limits [m]. If None the entire coverage of the radar is going to be used

**ele\_min, ele\_max, azi\_min, azi\_max** [float or None] The limits of the grid [deg]. If None the limits will be the limits of the radar volume

**Returns**

**radar** [radar object] The radar object containing only the desired data

`pyart.util.datetime_from_dataset(dataset, epoch=False, **kwargs)`

Return a datetime for the first time in a netCDF Dataset.

`pyart.util.datetime_from_grid(grid, epoch=False, **kwargs)`

Return a datetime for the volume start in a Grid.

`pyart.util.datetime_from_radar(radar, epoch=False, **kwargs)`

Return a datetime for the first ray in a Radar.

`pyart.util.datetimes_from_dataset(dataset, epoch=False, **kwargs)`

Return an array of datetimes for the times in a netCDF Dataset.

`pyart.util.datetimes_from_radar(radar, epoch=False, **kwargs)`

Return an array of datetimes for the rays in a Radar.

`pyart.util.estimate_noise_hs74(spectrum, navg=1, nnoise_min=1)`

Estimate noise parameters of a Doppler spectrum.

Use the method of estimating the noise level in Doppler spectra outlined by Hildebrand and Sekhon, 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 appropriate when no moving average has been applied to the spectrum.

**nnoise\_min** [int] Minimum number of noise samples to consider the estimation valid

**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.

`pyart.util.estimate_noise_ivic13(pwr_w_ray, pct=2.0, delay=1, flat_reg_wlen=96, flat_reg_var_max=1.5, snr_thr=1.6, npulses=30, ngates_min=800, iterations=10, get_noise_pos=False)`

Estimate noise parameters of a ray

Use the method of estimating the noise level in a ray outlined by Ivic, 2013.

**Parameters**

**pwr\_w\_ray** [array like] Doppler spectrum in linear units.

**pct** [float] Point Clutter Threshold

**delay** [int] distance of the gate to which compare for point target threshold

**flat\_reg\_wlen** [int] Minimum number of gates that should contain a valid region. Default gives a size of 8 km with 83.3 m resolution

**flat\_reg\_var\_max** [float] Maximum local variance of powers in decibels to consider the region as flat.

**snr\_thr** [float] Threshold applied in steps 3 and 6

**npulses** [int] Number of pulses used to compute the power of the array

**ngates\_min: int** minimum number of gates with noise to consider the retrieval valid

**iterations: int** number of iterations in step 7

**get\_noise\_pos** [bool] If True the indices of the gates identified as noise will be returned

#### Returns

**mean** [float-like] Mean of the gates in the ray identified as noise.

**var** [float-like] Variance of the gates in the ray identified as noise.

**nnoise** [int] Number of noise gates in the ray.

**inds\_ray** [1D-array] The indices of the gates containing noise

#### References

I.R. Ivic, C. Curtis and S.M. Torres, Radial-Based Noise Power Estimation for Weather Radars. Journal of Atmospheric and Oceanic Technology, 2013, 30, 2737-2753.

`pyart.util.get_ivic_flat_reg_var_max(npulses, y, x, flat_reg_wlen, ndBm=30.0, accuracy=0.001)`

Get the threshold for maximum local variance of noise [dB]

#### Parameters

**npulses** [1D array] array with the number of pulses in a ray

**y** [1D array] array of y values

**x** [1D array] array of x values. These are the possible values of the threshold

**flat\_reg\_wlen** [int] the lenght of the flat region window in bins

**ndBm** [float] the mean noise power in dBm

**accuracy** [float] The desired accuracy of the threshold (difference with target probability)

#### Returns

**var\_thr** [1D array] The thresholds corresponding to each number of pulses

`pyart.util.get_ivic_pct(npulses, pct, z, prob_thr=0.0001)`

Get the point clutter threshold (PCT)

#### Parameters

**npulses** [1D array] array with the number of pulses in a ray

**pct** [1D array] array with possible values of the PCT

**z** [1D array] array of z values

**prob\_thr** [float] Desired probability

**Returns**

**pct\_out** [1D array] The PCT threshold corresponding to each number of pulses

`pyart.util.get_ivic_snr_thr(npulses, snr_thr, pfa_thr=0.001)`

Get the threshold for steps 3 and 6 of ivic

**Parameters**

**npulses** [1D array] array with the number of pulses in a ray

**snr\_thr** [1D array] array with possible values of the snr threshold

**pfa\_thr** [float] the desired probability of false alarm

**Returns**

**snr\_thr\_out** [1D array] The snr threshold corresponding to each number of pulses according to the desired probability of false alarm

`pyart.util.interpolate_spectra(psr, kind='linear', fill_value=0.0)`

Interpolates the spectra so that it has a uniform grid

**Parameters**

**psr** [RadarSpectra object] The original spectra

**Returns**

**psr\_interp** [RadarSpectra object] The interpolated spectra

`pyart.util.intersection(radar1, radar2, h_tol=0.0, latlon_tol=0.0, vol_d_tol=None, 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, i.e.: 1 not intersecting, 2 intersecting, 0 is reserved

`pyart.util.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.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.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, optional] 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.ivic_flat_reg_var_max_table(npulses, flat_reg_wlen)`

Get maximum variance of noise of Ivic from a look up table.

**Parameters**

**npulses** [1D array] array with the number of pulses in a ray

**flat\_reg\_wlen** [float] The flat region window length in bins

**Returns**

**flat\_reg\_var\_max** [1D array] The maximum variance threshold corresponding to each number of pulses

`pyart.util.ivic_pct_table(npulses)`

Get the point clutter threshold (PCT) of Ivic from a look up table. The thresholds are computed for between 3 and 78 pulses. If there number of pulses is beyond this range it throws an error

**Parameters**

**npulses** [1D array] array with the number of pulses in a ray



**Returns**

**pct\_table** [1D array] The PCT corresponding to each number of pulses

`pyart.util.ivic_snr_thr_table(npulses)`

Get the threshold for steps 3 and 6 of ivic from a look up table The thresholds are computed for between 3 and 200 pulses. If there number of pulses is beyond this range it throws an error

**Parameters**

**npulses** [1D array] array with the number of pulses in a ray

**Returns**

**snr\_thr\_out** [1D array] The snr threshold corresponding to each number of pulses according to the desired probability of false alarm

`pyart.util.join_radar(radar1, radar2)`

Combine two radar instances into one.

**Parameters**

**radar1** [Radar] Radar object.

**radar2** [Radar] Radar object.

`pyart.util.join_spectra(spectra1, spectra2)`

Combine two spectra instances into one.

**Parameters**

**spectra1** [spectra] spectra object.

**spectra2** [spectra] spectra object.

`pyart.util.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.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.

`pyart.util.radar_from_spectra(psr)`

obtain a Radar object from a RadarSpectra object

**Parameters**

**psr** [RadarSpectra object] The reference object

**Returns**

**radar** [radar object] The new radar object

`pyart.util.rolling_window(a, window)`

create a rolling window object for application of functions eg: `result=np.ma.std(array, 11), 1)`

`pyart.util.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.

`pyart.util.texture(radar, var)`

Determine a texture field using an 11pt stdev texarray=`texture(pyradarobj, field)`.

`pyart.util.texture_along_ray(radar, var, wind_size=7)`

Compute field texture along ray using a user specified window size.

#### Parameters

**radar** [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] Optional. Size of the rolling window used.

#### Returns

**tex** [radar field] The texture of the specified field.

`pyart.util.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.

## TESTING UTILITIES (PYART . TESTING)

Utilities helpful when writing and running unit tests.

### 11.1 Testing functions

<code>make_empty_ppi_radar</code> (ngates, rays_per_sweep, ...)	Return an Radar object, representing a PPI scan.
<code>make_target_radar</code> ()	Return a PPI radar with a target like reflectivity field.
<code>make_single_ray_radar</code> ()	Return a PPI radar with a single ray taken from a ARM C-SAPR Radar.
<code>make_velocity_aliased_radar</code> ([alias])	Return a PPI radar with a target like reflectivity field.
<code>make_empty_grid</code> (grid_shape, grid_limits)	Make an empty grid object without any fields or meta-data.
<code>make_target_grid</code> ()	Make a sample Grid with a rectangular target.
<code>make_storm_grid</code> ()	Make a sample Grid with a rectangular storm target.
<code>make_normal_storm</code> (sigma, mu)	Make a sample Grid with a gaussian storm target.

### 11.2 Testing classes

<code>InTemporaryDirectory</code> ([suffix, prefix, dir])	Create, return, and change directory to a temporary directory.
-----------------------------------------------------------	----------------------------------------------------------------

**class** `pyart.testing.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
```

(continues on next page)

(continued from previous page)

```
>>> os.getcwd() == my_cwd
True
```

## Methods

cleanup	
---------	--

```
__class__
    alias of builtins.type

__delattr__ (self, name, /)
    Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyart.testing.tmpdirs', '__doc__': " Create,
__dir__ (self, /)
    Default dir() implementation.

__enter__ (self)

__eq__ (self, value, /)
    Return self==value.

__exit__ (self, exc, value, tb)

__format__ (self, format_spec, /)
    Default object formatter.

__ge__ (self, value, /)
    Return self>=value.

__getattr__ (self, name, /)
    Return getattr(self, name).

__gt__ (self, value, /)
    Return self>value.

__hash__ (self, /)
    Return hash(self).

__init__ (self, suffix="", prefix='tmp', dir=None)
    Initialize self. See help(type(self)) for accurate signature.

__init_subclass__ ()
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (self, value, /)
    Return self<=value.

__lt__ (self, value, /)
    Return self<value.

__module__ = 'pyart.testing.tmpdirs'

__ne__ (self, value, /)
    Return self!=value.
```

**\_\_new\_\_** (\*args, \*\*kwargs)  
Create and return a new object. See help(type) for accurate signature.

**\_\_reduce\_\_** (self, /)  
Helper for pickle.

**\_\_reduce\_ex** (self, protocol, /)  
Helper for pickle.

**\_\_repr\_\_** (self, /)  
Return repr(self).

**\_\_setattr\_\_** (self, name, value, /)  
Implement setattr(self, name, value).

**\_\_sizeof\_\_** (self, /)  
Size of object in memory, in bytes.

**\_\_str\_\_** (self, /)  
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** (self)

**pyart.testing.make\_empty\_grid** (grid\_shape, grid\_limits)  
Make an empty grid object without any fields or metadata.

#### Parameters

**grid\_shape** [3-tuple of floats] Number of points in the grid (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.make\_empty\_ppi\_radar** (ngates, rays\_per\_sweep, nsweeps)  
Return an Radar object, representing a PPI scan.

#### Parameters

**ngates** [int] Number of gates per ray.

**rays\_per\_sweep** [int] Number of rays in each PPI sweep.

**nsweeps** [int] Number of sweeps.

#### Returns

**radar** [Radar] Radar object with no fields, other parameters are set to default values.

**pyart.testing.make\_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.make_normal_storm(sigma, mu)`

Make a sample Grid with a gaussian storm target.

`pyart.testing.make_single_ray_radar()`

Return a PPI radar with a single ray taken from a ARM C-SAPR Radar.

Radar object returned has 'reflectivity\_horizontal', 'norm\_coherent\_power', 'copol\_coeff', 'dp\_phase\_shift', 'diff\_phase', and 'differential\_reflectivity' fields with no metadata but a 'data' key. This radar is used for unit tests in correct modules.

`pyart.testing.make_storm_grid()`

Make a sample Grid with a rectangular storm target.

`pyart.testing.make_target_grid()`

Make a sample Grid with a rectangular target.

`pyart.testing.make_target_radar()`

Return a PPI radar with a target like reflectivity field.

`pyart.testing.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.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.

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## BIBLIOGRAPHY

- [1] <http://www.ncdc.noaa.gov/>
- [2] <http://thredds.ucar.edu/thredds/catalog.html>
- [1] <http://www.unidata.ucar.edu/software/netcdf-java/documentation.htm>
- [2] <http://thredds.ucar.edu/thredds/catalog.html>
- [1] <http://www.ncdc.noaa.gov/>
- [2] [http://www.roc.noaa.gov/wsr88d/Level\\_III/Level3Info.asp](http://www.roc.noaa.gov/wsr88d/Level_III/Level3Info.asp)
- [1] Doviak and Zrnic, Doppler Radar and Weather Observations, Second Edition, 1993, p. 21.
- [1] Snyder, J. P. Map Projections—A Working Manual. U. S. Geological Survey Professional Paper 1395, 1987, pp. 191-202.
- [1] Snyder, J. P. Map Projections—A Working Manual. U. S. Geological Survey Professional Paper 1395, 1987, pp. 191-202.
- [1] Miguel Arevallilo Herraiez, 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,
- [2] 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.



## PYTHON MODULE INDEX

### p

- `pyart.aux_io`, 18
- `pyart.bridge`, 61
- `pyart.core`, 34
- `pyart.correct`, 75
- `pyart.filters`, 63
- `pyart.graph`, 151
- `pyart.io`, 1
- `pyart.map`, 146
- `pyart.retrieve`, 110
- `pyart.testing`, 238
- `pyart.util`, 228



## Symbols

`__class__` (*pyart.core.Grid* attribute), 37  
`__class__` (*pyart.core.HorizontalWindProfile* attribute), 40  
`__class__` (*pyart.core.Radar* attribute), 44  
`__class__` (*pyart.core.RadarSpectra* attribute), 53  
`__class__` (*pyart.correct.GateFilter* attribute), 80  
`__class__` (*pyart.filters.GateFilter* attribute), 67  
`__class__` (*pyart.graph.AirborneRadarDisplay* attribute), 155  
`__class__` (*pyart.graph.GridMapDisplay* attribute), 168  
`__class__` (*pyart.graph.GridMapDisplayBasemap* attribute), 178  
`__class__` (*pyart.graph.RadarDisplay* attribute), 186  
`__class__` (*pyart.graph.RadarMapDisplay* attribute), 199  
`__class__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 214  
`__class__` (*pyart.testing.InTemporaryDirectory* attribute), 240  
`__delattr__` (*pyart.core.Grid* attribute), 37  
`__delattr__` (*pyart.core.HorizontalWindProfile* attribute), 40  
`__delattr__` (*pyart.core.Radar* attribute), 44  
`__delattr__` (*pyart.core.RadarSpectra* attribute), 53  
`__delattr__` (*pyart.correct.GateFilter* attribute), 80  
`__delattr__` (*pyart.filters.GateFilter* attribute), 67  
`__delattr__` (*pyart.graph.AirborneRadarDisplay* attribute), 155  
`__delattr__` (*pyart.graph.GridMapDisplay* attribute), 168  
`__delattr__` (*pyart.graph.GridMapDisplayBasemap* attribute), 178  
`__delattr__` (*pyart.graph.RadarDisplay* attribute), 186  
`__delattr__` (*pyart.graph.RadarMapDisplay* attribute), 199  
`__delattr__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 214  
`__delattr__` (*pyart.testing.InTemporaryDirectory* attribute), 240  
`__dict__` (*pyart.core.Grid* attribute), 37  
`__dict__` (*pyart.core.HorizontalWindProfile* attribute), 40  
`__dict__` (*pyart.core.Radar* attribute), 45  
`__dict__` (*pyart.core.RadarSpectra* attribute), 53  
`__dict__` (*pyart.correct.GateFilter* attribute), 80  
`__dict__` (*pyart.filters.GateFilter* attribute), 67  
`__dict__` (*pyart.graph.AirborneRadarDisplay* attribute), 155  
`__dict__` (*pyart.graph.GridMapDisplay* attribute), 168  
`__dict__` (*pyart.graph.GridMapDisplayBasemap* attribute), 178  
`__dict__` (*pyart.graph.RadarDisplay* attribute), 186  
`__dict__` (*pyart.graph.RadarMapDisplay* attribute), 199  
`__dict__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 214  
`__dict__` (*pyart.testing.InTemporaryDirectory* attribute), 240  
`__dir__` () (*pyart.core.Grid* method), 37  
`__dir__` () (*pyart.core.HorizontalWindProfile* method), 40  
`__dir__` () (*pyart.core.Radar* method), 45  
`__dir__` () (*pyart.core.RadarSpectra* method), 53  
`__dir__` () (*pyart.correct.GateFilter* method), 80  
`__dir__` () (*pyart.filters.GateFilter* method), 67  
`__dir__` () (*pyart.graph.AirborneRadarDisplay* method), 155  
`__dir__` () (*pyart.graph.GridMapDisplay* method), 169  
`__dir__` () (*pyart.graph.GridMapDisplayBasemap* method), 178  
`__dir__` () (*pyart.graph.RadarDisplay* method), 186  
`__dir__` () (*pyart.graph.RadarMapDisplay* method), 199  
`__dir__` () (*pyart.graph.RadarMapDisplayBasemap* method), 214  
`__dir__` () (*pyart.testing.InTemporaryDirectory* method), 240  
`__enter__` () (*pyart.testing.InTemporaryDirectory* method), 240

`__eq__` (*pyart.core.Grid* attribute), 37  
`__eq__` (*pyart.core.HorizontalWindProfile* attribute), 40  
`__eq__` (*pyart.core.Radar* attribute), 45  
`__eq__` (*pyart.core.RadarSpectra* attribute), 53  
`__eq__` (*pyart.correct.GateFilter* attribute), 80  
`__eq__` (*pyart.filters.GateFilter* attribute), 67  
`__eq__` (*pyart.graph.AirborneRadarDisplay* attribute), 155  
`__eq__` (*pyart.graph.GridMapDisplay* attribute), 169  
`__eq__` (*pyart.graph.GridMapDisplayBasemap* attribute), 178  
`__eq__` (*pyart.graph.RadarDisplay* attribute), 186  
`__eq__` (*pyart.graph.RadarMapDisplay* attribute), 199  
`__eq__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 214  
`__eq__` (*pyart.testing.InTemporaryDirectory* attribute), 240  
`__exit__` () (*pyart.testing.InTemporaryDirectory* method), 240  
`__format__` () (*pyart.core.Grid* method), 37  
`__format__` () (*pyart.core.HorizontalWindProfile* method), 40  
`__format__` () (*pyart.core.Radar* method), 45  
`__format__` () (*pyart.core.RadarSpectra* method), 53  
`__format__` () (*pyart.correct.GateFilter* method), 80  
`__format__` () (*pyart.filters.GateFilter* method), 67  
`__format__` () (*pyart.graph.AirborneRadarDisplay* method), 155  
`__format__` () (*pyart.graph.GridMapDisplay* method), 169  
`__format__` () (*pyart.graph.GridMapDisplayBasemap* method), 178  
`__format__` () (*pyart.graph.RadarDisplay* method), 186  
`__format__` () (*pyart.graph.RadarMapDisplay* method), 199  
`__format__` () (*pyart.graph.RadarMapDisplayBasemap* method), 214  
`__format__` () (*pyart.testing.InTemporaryDirectory* method), 240  
`__ge__` (*pyart.core.Grid* attribute), 37  
`__ge__` (*pyart.core.HorizontalWindProfile* attribute), 40  
`__ge__` (*pyart.core.Radar* attribute), 45  
`__ge__` (*pyart.core.RadarSpectra* attribute), 53  
`__ge__` (*pyart.correct.GateFilter* attribute), 80  
`__ge__` (*pyart.filters.GateFilter* attribute), 67  
`__ge__` (*pyart.graph.AirborneRadarDisplay* attribute), 155  
`__ge__` (*pyart.graph.GridMapDisplay* attribute), 169  
`__ge__` (*pyart.graph.GridMapDisplayBasemap* attribute), 178  
`__ge__` (*pyart.graph.RadarDisplay* attribute), 186  
`__ge__` (*pyart.graph.RadarMapDisplay* attribute), 199  
`__ge__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 215  
`__ge__` (*pyart.testing.InTemporaryDirectory* attribute), 240  
`__gt__` (*pyart.core.Grid* attribute), 37  
`__gt__` (*pyart.core.HorizontalWindProfile* attribute), 40  
`__gt__` (*pyart.core.Radar* attribute), 45  
`__gt__` (*pyart.core.RadarSpectra* attribute), 53  
`__gt__` (*pyart.correct.GateFilter* attribute), 80  
`__gt__` (*pyart.filters.GateFilter* attribute), 67  
`__gt__` (*pyart.graph.AirborneRadarDisplay* attribute), 155  
`__gt__` (*pyart.graph.GridMapDisplay* attribute), 169  
`__gt__` (*pyart.graph.GridMapDisplayBasemap* attribute), 178  
`__gt__` (*pyart.graph.RadarDisplay* attribute), 186  
`__gt__` (*pyart.graph.RadarMapDisplay* attribute), 199  
`__gt__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 215  
`__gt__` (*pyart.testing.InTemporaryDirectory* attribute), 240  
`__hash__` (*pyart.core.Grid* attribute), 37  
`__hash__` (*pyart.core.HorizontalWindProfile* attribute), 40  
`__hash__` (*pyart.core.Radar* attribute), 45  
`__hash__` (*pyart.core.RadarSpectra* attribute), 53

---

```

__hash__ (pyart.correct.GateFilter attribute), 80
__hash__ (pyart.filters.GateFilter attribute), 67
__hash__ (pyart.graph.AirborneRadarDisplay attribute), 155
__hash__ (pyart.graph.GridMapDisplay attribute), 169
__hash__ (pyart.graph.GridMapDisplayBasemap attribute), 178
__hash__ (pyart.graph.RadarDisplay attribute), 186
__hash__ (pyart.graph.RadarMapDisplay attribute), 199
__hash__ (pyart.graph.RadarMapDisplayBasemap attribute), 215
__hash__ (pyart.testing.InTemporaryDirectory attribute), 240
__init__ () (pyart.core.Grid method), 37
__init__ () (pyart.core.HorizontalWindProfile method), 40
__init__ () (pyart.core.Radar method), 45
__init__ () (pyart.core.RadarSpectra method), 53
__init__ () (pyart.correct.GateFilter method), 80
__init__ () (pyart.filters.GateFilter method), 67
__init__ () (pyart.graph.AirborneRadarDisplay method), 155
__init__ () (pyart.graph.GridMapDisplay method), 169
__init__ () (pyart.graph.GridMapDisplayBasemap method), 178
__init__ () (pyart.graph.RadarDisplay method), 186
__init__ () (pyart.graph.RadarMapDisplay method), 199
__init__ () (pyart.graph.RadarMapDisplayBasemap method), 215
__init__ () (pyart.testing.InTemporaryDirectory method), 240
__init_subclass__ () (pyart.core.Grid method), 37
__init_subclass__ () (pyart.core.HorizontalWindProfile method), 40
__init_subclass__ () (pyart.core.Radar method), 45
__init_subclass__ () (pyart.core.RadarSpectra method), 53
__init_subclass__ () (pyart.correct.GateFilter method), 80
__init_subclass__ () (pyart.filters.GateFilter method), 67
__init_subclass__ () (pyart.graph.AirborneRadarDisplay method), 155
__init_subclass__ () (pyart.graph.GridMapDisplay method), 169
__init_subclass__ () (pyart.graph.GridMapDisplayBasemap method), 178
__init_subclass__ () (pyart.graph.RadarDisplay method), 186
__init_subclass__ () (pyart.graph.RadarMapDisplay method), 199
__init_subclass__ () (pyart.graph.RadarMapDisplayBasemap method), 215
__init_subclass__ () (pyart.testing.InTemporaryDirectory method), 240
__le__ (pyart.core.Grid attribute), 37
__le__ (pyart.core.HorizontalWindProfile attribute), 40
__le__ (pyart.core.Radar attribute), 45
__le__ (pyart.core.RadarSpectra attribute), 53
__le__ (pyart.correct.GateFilter attribute), 80
__le__ (pyart.filters.GateFilter attribute), 67
__le__ (pyart.graph.AirborneRadarDisplay attribute), 155
__le__ (pyart.graph.GridMapDisplay attribute), 169
__le__ (pyart.graph.GridMapDisplayBasemap attribute), 178
__le__ (pyart.graph.RadarDisplay attribute), 186
__le__ (pyart.graph.RadarMapDisplay attribute), 200
__le__ (pyart.graph.RadarMapDisplayBasemap attribute), 215
__le__ (pyart.testing.InTemporaryDirectory attribute), 240
__lt__ (pyart.core.Grid attribute), 37
__lt__ (pyart.core.HorizontalWindProfile attribute), 41
__lt__ (pyart.core.Radar attribute), 45
__lt__ (pyart.core.RadarSpectra attribute), 53
__lt__ (pyart.correct.GateFilter attribute), 80
__lt__ (pyart.filters.GateFilter attribute), 67
__lt__ (pyart.graph.AirborneRadarDisplay attribute), 155
__lt__ (pyart.graph.GridMapDisplay attribute), 169
__lt__ (pyart.graph.GridMapDisplayBasemap attribute), 178
__lt__ (pyart.graph.RadarDisplay attribute), 186
__lt__ (pyart.graph.RadarMapDisplay attribute), 200
__lt__ (pyart.graph.RadarMapDisplayBasemap attribute), 215
__lt__ (pyart.testing.InTemporaryDirectory attribute), 240
__module__ (pyart.core.Grid attribute), 37
__module__ (pyart.core.HorizontalWindProfile attribute), 41
__module__ (pyart.core.Radar attribute), 45
__module__ (pyart.core.RadarSpectra attribute), 54
__module__ (pyart.correct.GateFilter attribute), 80

```

`__module__` (*pyart.filters.GateFilter* attribute), 67  
`__module__` (*pyart.graph.AirborneRadarDisplay* attribute), 155  
`__module__` (*pyart.graph.GridMapDisplay* attribute), 169  
`__module__` (*pyart.graph.GridMapDisplayBasemap* attribute), 178  
`__module__` (*pyart.graph.RadarDisplay* attribute), 186  
`__module__` (*pyart.graph.RadarMapDisplay* attribute), 200  
`__module__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 215  
`__module__` (*pyart.testing.InTemporaryDirectory* attribute), 240  
`__ne__` (*pyart.core.Grid* attribute), 38  
`__ne__` (*pyart.core.HorizontalWindProfile* attribute), 41  
`__ne__` (*pyart.core.Radar* attribute), 45  
`__ne__` (*pyart.core.RadarSpectra* attribute), 54  
`__ne__` (*pyart.correct.GateFilter* attribute), 80  
`__ne__` (*pyart.filters.GateFilter* attribute), 67  
`__ne__` (*pyart.graph.AirborneRadarDisplay* attribute), 155  
`__ne__` (*pyart.graph.GridMapDisplay* attribute), 169  
`__ne__` (*pyart.graph.GridMapDisplayBasemap* attribute), 178  
`__ne__` (*pyart.graph.RadarDisplay* attribute), 187  
`__ne__` (*pyart.graph.RadarMapDisplay* attribute), 200  
`__ne__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 215  
`__ne__` (*pyart.testing.InTemporaryDirectory* attribute), 240  
`__new__` () (*pyart.core.Grid* method), 38  
`__new__` () (*pyart.core.HorizontalWindProfile* method), 41  
`__new__` () (*pyart.core.Radar* method), 45  
`__new__` () (*pyart.core.RadarSpectra* method), 54  
`__new__` () (*pyart.correct.GateFilter* method), 80  
`__new__` () (*pyart.filters.GateFilter* method), 67  
`__new__` () (*pyart.graph.AirborneRadarDisplay* method), 155  
`__new__` () (*pyart.graph.GridMapDisplay* method), 169  
`__new__` () (*pyart.graph.GridMapDisplayBasemap* method), 178  
`__new__` () (*pyart.graph.RadarDisplay* method), 187  
`__new__` () (*pyart.graph.RadarMapDisplay* method), 200  
`__new__` () (*pyart.graph.RadarMapDisplayBasemap* method), 215  
`__new__` () (*pyart.testing.InTemporaryDirectory* method), 240  
`__reduce__` () (*pyart.core.Grid* method), 38  
`__reduce__` () (*pyart.core.HorizontalWindProfile* method), 41  
`__reduce__` () (*pyart.core.Radar* method), 45  
`__reduce__` () (*pyart.core.RadarSpectra* method), 54  
`__reduce__` () (*pyart.correct.GateFilter* method), 81  
`__reduce__` () (*pyart.filters.GateFilter* method), 67  
`__reduce__` () (*pyart.graph.AirborneRadarDisplay* method), 155  
`__reduce__` () (*pyart.graph.GridMapDisplay* method), 169  
`__reduce__` () (*pyart.graph.GridMapDisplayBasemap* method), 178  
`__reduce__` () (*pyart.graph.RadarDisplay* method), 187  
`__reduce__` () (*pyart.graph.RadarMapDisplay* method), 200  
`__reduce__` () (*pyart.graph.RadarMapDisplayBasemap* method), 215  
`__reduce__` () (*pyart.testing.InTemporaryDirectory* method), 241  
`__repr__` (*pyart.core.Grid* attribute), 38  
`__repr__` (*pyart.core.HorizontalWindProfile* attribute), 41  
`__repr__` (*pyart.core.Radar* attribute), 45  
`__repr__` (*pyart.core.RadarSpectra* attribute), 54  
`__repr__` (*pyart.correct.GateFilter* attribute), 81  
`__repr__` (*pyart.filters.GateFilter* attribute), 68  
`__repr__` (*pyart.graph.AirborneRadarDisplay* attribute), 155





`__weakref__` (*pyart.core.Grid* attribute), 38  
`__weakref__` (*pyart.core.HorizontalWindProfile* attribute), 41  
`__weakref__` (*pyart.core.Radar* attribute), 46  
`__weakref__` (*pyart.core.RadarSpectra* attribute), 54  
`__weakref__` (*pyart.correct.GateFilter* attribute), 81  
`__weakref__` (*pyart.filters.GateFilter* attribute), 68  
`__weakref__` (*pyart.graph.AirborneRadarDisplay* attribute), 156  
`__weakref__` (*pyart.graph.GridMapDisplay* attribute), 170  
`__weakref__` (*pyart.graph.GridMapDisplayBasemap* attribute), 179  
`__weakref__` (*pyart.graph.RadarDisplay* attribute), 187  
`__weakref__` (*pyart.graph.RadarMapDisplay* attribute), 200  
`__weakref__` (*pyart.graph.RadarMapDisplayBasemap* attribute), 215  
`__weakref__` (*pyart.testing.InTemporaryDirectory* attribute), 241  
`_check_ax()` (*pyart.graph.RadarMapDisplay* method), 200  
`_check_basemap()` (*pyart.graph.RadarMapDisplayBasemap* method), 215  
`_check_sweep_in_range()` (*pyart.core.Radar* method), 46  
`_check_sweep_in_range()` (*pyart.core.RadarSpectra* method), 54  
`_dic_info()` (*pyart.core.Radar* method), 46  
`_dic_info()` (*pyart.core.RadarSpectra* method), 54  
`_find_and_check_nradar()` (*pyart.core.Grid* method), 38  
`_find_nearest_grid_indices()` (*pyart.graph.GridMapDisplay* method), 170  
`_find_nearest_grid_indices()` (*pyart.graph.GridMapDisplayBasemap* method), 179  
`_get_azimuth_rhi_data_x_y_z()` (*pyart.graph.AirborneRadarDisplay* method), 156  
`_get_azimuth_rhi_data_x_y_z()` (*pyart.graph.RadarDisplay* method), 187  
`_get_azimuth_rhi_data_x_y_z()` (*pyart.graph.RadarMapDisplay* method), 200  
`_get_azimuth_rhi_data_x_y_z()` (*pyart.graph.RadarMapDisplayBasemap* method), 216  
`_get_colorbar_label()` (*pyart.graph.AirborneRadarDisplay* method), 156  
`_get_colorbar_label()` (*pyart.graph.RadarDisplay* method), 187  
`_get_colorbar_label()` (*pyart.graph.RadarMapDisplay* method), 200  
`_get_colorbar_label()` (*pyart.graph.RadarMapDisplayBasemap* method), 216  
`_get_data()` (*pyart.graph.AirborneRadarDisplay* method), 156  
`_get_data()` (*pyart.graph.RadarDisplay* method), 187  
`_get_data()` (*pyart.graph.RadarMapDisplay* method), 200  
`_get_data()` (*pyart.graph.RadarMapDisplayBasemap* method), 216  
`_get_fdata()` (*pyart.correct.GateFilter* method), 81  
`_get_fdata()` (*pyart.filters.GateFilter* method), 68  
`_get_label_x()` (*pyart.graph.GridMapDisplay* method), 170  
`_get_label_x()` (*pyart.graph.GridMapDisplayBasemap* method), 179  
`_get_label_y()` (*pyart.graph.GridMapDisplay* method), 179  
`_get_label_y()` (*pyart.graph.GridMapDisplayBasemap* method), 179  
`_get_label_z()` (*pyart.graph.GridMapDisplay* method), 170  
`_get_label_z()` (*pyart.graph.GridMapDisplayBasemap* method), 179  
`_get_ray_data()` (*pyart.graph.AirborneRadarDisplay* method), 156  
`_get_ray_data()` (*pyart.graph.RadarDisplay* method), 187  
`_get_ray_data()` (*pyart.graph.RadarMapDisplay* method), 200  
`_get_ray_data()` (*pyart.graph.RadarMapDisplayBasemap* method), 216  
`_get_vpt_data()` (*pyart.graph.AirborneRadarDisplay* method), 156  
`_get_vpt_data()` (*pyart.graph.RadarDisplay* method), 187  
`_get_vpt_data()` (*pyart.graph.RadarMapDisplay* method), 200  
`_get_vpt_data()` (*pyart.graph.RadarMapDisplayBasemap* method), 216  
`_get_x_y()` (*pyart.graph.AirborneRadarDisplay* method), 156  
`_get_x_y()` (*pyart.graph.RadarDisplay* method), 187  
`_get_x_y()` (*pyart.graph.RadarMapDisplay* method), 200  
`_get_x_y()` (*pyart.graph.RadarMapDisplayBasemap* method), 216

---

```

_get_x_y_z() (pyart.graph.AirborneRadarDisplay
              method), 156
_get_x_y_z() (pyart.graph.RadarDisplay method),
              187
_get_x_y_z() (pyart.graph.RadarMapDisplay
              method), 201
_get_x_y_z() (pyart.graph.RadarMapDisplayBasemap
              method), 216
_get_x_z() (pyart.graph.AirborneRadarDisplay
            method), 156
_get_x_z() (pyart.graph.RadarDisplay method), 187
_get_x_z() (pyart.graph.RadarMapDisplay method),
            201
_get_x_z() (pyart.graph.RadarMapDisplayBasemap
            method), 216
_label_axes_grid()
            (pyart.graph.GridMapDisplay method),
            170
_label_axes_grid()
            (pyart.graph.GridMapDisplayBasemap
            method), 179
_label_axes_latitude()
            (pyart.graph.GridMapDisplay method),
            170
_label_axes_latitude()
            (pyart.graph.GridMapDisplayBasemap
            method), 179
_label_axes_latlon()
            (pyart.graph.GridMapDisplay method),
            170
_label_axes_longitude()
            (pyart.graph.GridMapDisplay method),
            170
_label_axes_longitude()
            (pyart.graph.GridMapDisplayBasemap
            method), 179
_label_axes_ppi()
            (pyart.graph.AirborneRadarDisplay method),
            156
_label_axes_ppi() (pyart.graph.RadarDisplay
                    method), 187
_label_axes_ppi() (pyart.graph.RadarMapDisplay
                    method),
                    201
_label_axes_ppi() (pyart.graph.RadarMapDisplayBasemap
                    method), 216
_label_axes_ray()
            (pyart.graph.AirborneRadarDisplay method),
            156
_label_axes_ray() (pyart.graph.RadarDisplay
                    method), 187
_label_axes_ray() (pyart.graph.RadarMapDisplay
                    method),
                    201
_label_axes_ray() (pyart.graph.RadarMapDisplayBasemap
                    method), 216
_label_axes_rhi()
            (pyart.graph.AirborneRadarDisplay method),
            156
_label_axes_rhi() (pyart.graph.RadarDisplay
                    method), 188
_label_axes_rhi() (pyart.graph.RadarMapDisplay
                    method),
                    201
_label_axes_rhi() (pyart.graph.RadarMapDisplayBasemap
                    method), 216
_label_axes_vpt()
            (pyart.graph.AirborneRadarDisplay method),
            156
_label_axes_vpt() (pyart.graph.RadarDisplay
                    method), 188
_label_axes_vpt() (pyart.graph.RadarMapDisplay
                    method),
                    201
_label_axes_vpt() (pyart.graph.RadarMapDisplayBasemap
                    method), 216
_make_basemap() (pyart.graph.GridMapDisplayBasemap
                 method), 179
_merge() (pyart.correct.GateFilter method), 81
_merge() (pyart.filters.GateFilter method), 68
_parse_location_data()
            (pyart.core.HorizontalWindProfile method), 41
_set_az_rhi_title()
            (pyart.graph.AirborneRadarDisplay method),
            156
_set_az_rhi_title() (pyart.graph.RadarDisplay
                     method), 188
_set_az_rhi_title() (pyart.graph.RadarMapDisplay
                     method),
                     201
_set_az_rhi_title() (pyart.graph.RadarMapDisplayBasemap
                     method), 216
_set_ray_title() (pyart.graph.AirborneRadarDisplay
                  method), 156
_set_ray_title() (pyart.graph.RadarDisplay
                  method), 188
_set_ray_title() (pyart.graph.RadarMapDisplay
                  method), 201
_set_ray_title() (pyart.graph.RadarMapDisplayBasemap
                  method), 216
_set_title() (pyart.graph.AirborneRadarDisplay
              method), 156

```

---

`_set_title()` (*pyart.graph.RadarDisplay* method), 188  
`_set_title()` (*pyart.graph.RadarMapDisplay* method), 201  
`_set_title()` (*pyart.graph.RadarMapDisplayBasemap* method), 216  
`_set_vpt_time_axis()` (*pyart.graph.AirborneRadarDisplay* static method), 157  
`_set_vpt_time_axis()` (*pyart.graph.RadarDisplay* static method), 188  
`_set_vpt_time_axis()` (*pyart.graph.RadarMapDisplay* static method), 201  
`_set_vpt_time_axis()` (*pyart.graph.RadarMapDisplayBasemap* static method), 216  
`_set_vpt_title()` (*pyart.graph.AirborneRadarDisplay* method), 157  
`_set_vpt_title()` (*pyart.graph.RadarDisplay* method), 188  
`_set_vpt_title()` (*pyart.graph.RadarMapDisplay* method), 201  
`_set_vpt_title()` (*pyart.graph.RadarMapDisplayBasemap* method), 216

## A

`add_field()` (*pyart.core.Grid* method), 38  
`add_field()` (*pyart.core.Radar* method), 46  
`add_field()` (*pyart.core.RadarSpectra* method), 54  
`add_field_like()` (*pyart.core.Radar* method), 46  
`add_field_like()` (*pyart.core.RadarSpectra* method), 54  
`AirborneRadarDisplay` (class in *pyart.graph*), 153  
`angular_mean()` (in module *pyart.util*), 230  
`angular_mean_deg()` (in module *pyart.util*), 230  
`angular_std()` (in module *pyart.util*), 230  
`angular_std_deg()` (in module *pyart.util*), 231  
`angular_texture_2d()` (in module *pyart.util*), 231  
`antenna_to_cartesian()` (in module *pyart.core*), 58  
`antenna_vectors_to_cartesian()` (in module *pyart.core*), 58  
`atmospheric_gas_att()` (in module *pyart.retrieve*), 114

## B

`birds_gate_filter()` (in module *pyart.filters*), 71

## C

`calculate_attenuation()` (in module *pyart.correct*), 84

`calculate_attenuation_philinear()` (in module *pyart.correct*), 85  
`calculate_attenuation_zphi()` (in module *pyart.correct*), 86  
`calculate_snr_from_reflectivity()` (in module *pyart.retrieve*), 114  
`calculate_velocity_texture()` (in module *pyart.retrieve*), 114  
`cartesian_to_antenna()` (in module *pyart.core*), 59  
`cartesian_to_geographic()` (in module *pyart.core*), 59  
`cartesian_to_geographic_aeqd()` (in module *pyart.core*), 59  
`cartesian_vectors_to_geographic()` (in module *pyart.core*), 60  
`cartopy_coastlines()` (*pyart.graph.GridMapDisplay* method), 170  
`cartopy_political_boundaries()` (*pyart.graph.GridMapDisplay* method), 170  
`cartopy_states()` (*pyart.graph.GridMapDisplay* method), 170  
`check_field_exists()` (*pyart.core.Radar* method), 47  
`check_field_exists()` (*pyart.core.RadarSpectra* method), 55  
`class_based_gate_filter()` (in module *pyart.filters*), 72  
`cleanup()` (*pyart.testing.InTemporaryDirectory* method), 241  
`colocated_gates()` (in module *pyart.util*), 231  
`compute_bird_density()` (in module *pyart.retrieve*), 116  
`compute_ccor()` (in module *pyart.retrieve*), 116  
`compute_cdr()` (in module *pyart.retrieve*), 116  
`compute_differential_phase()` (in module *pyart.retrieve*), 116  
`compute_differential_phase_iq()` (in module *pyart.retrieve*), 117  
`compute_differential_reflectivity()` (in module *pyart.retrieve*), 117  
`compute_differential_reflectivity_iq()` (in module *pyart.retrieve*), 117  
`compute_Doppler_velocity()` (in module *pyart.retrieve*), 115  
`compute_Doppler_velocity_iq()` (in module *pyart.retrieve*), 115  
`compute_Doppler_width()` (in module *pyart.retrieve*), 115  
`compute_Doppler_width_iq()` (in module *pyart.retrieve*), 115  
`compute_evp()` (in module *pyart.retrieve*), 117

- [compute\\_iq\(\)](#) (in module [pyart.retrieve](#)), 118  
[compute\\_l\(\)](#) (in module [pyart.retrieve](#)), 118  
[compute\\_mean\\_phase\\_iq\(\)](#) (in module [pyart.retrieve](#)), 119  
[compute\\_noise\\_power\(\)](#) (in module [pyart.retrieve](#)), 119  
[compute\\_noisedBZ\(\)](#) (in module [pyart.retrieve](#)), 119  
[compute\\_pol\\_variables\(\)](#) (in module [pyart.retrieve](#)), 119  
[compute\\_pol\\_variables\\_iq\(\)](#) (in module [pyart.retrieve](#)), 120  
[compute\\_qvp\(\)](#) (in module [pyart.retrieve](#)), 120  
[compute\\_radial\\_noise\\_hs\(\)](#) (in module [pyart.retrieve](#)), 121  
[compute\\_radial\\_noise\\_ivic\(\)](#) (in module [pyart.retrieve](#)), 122  
[compute\\_rcs\(\)](#) (in module [pyart.retrieve](#)), 122  
[compute\\_rcs\\_from\\_pr\(\)](#) (in module [pyart.retrieve](#)), 123  
[compute\\_reflectivity\(\)](#) (in module [pyart.retrieve](#)), 123  
[compute\\_reflectivity\\_iq\(\)](#) (in module [pyart.retrieve](#)), 123  
[compute\\_rhohv\(\)](#) (in module [pyart.retrieve](#)), 124  
[compute\\_rhohv\\_iq\(\)](#) (in module [pyart.retrieve](#)), 124  
[compute\\_rqvp\(\)](#) (in module [pyart.retrieve](#)), 124  
[compute\\_signal\\_power\(\)](#) (in module [pyart.retrieve](#)), 125  
[compute\\_snr\(\)](#) (in module [pyart.retrieve](#)), 125  
[compute\\_spectra\(\)](#) (in module [pyart.retrieve](#)), 125  
[compute\\_spectral\\_differential\\_phase\(\)](#) (in module [pyart.retrieve](#)), 126  
[compute\\_spectral\\_differential\\_reflectivity\(\)](#) (in module [pyart.retrieve](#)), 126  
[compute\\_spectral\\_noise\(\)](#) (in module [pyart.retrieve](#)), 127  
[compute\\_spectral\\_phase\(\)](#) (in module [pyart.retrieve](#)), 127  
[compute\\_spectral\\_power\(\)](#) (in module [pyart.retrieve](#)), 127  
[compute\\_spectral\\_reflectivity\(\)](#) (in module [pyart.retrieve](#)), 128  
[compute\\_spectral\\_rhohv\(\)](#) (in module [pyart.retrieve](#)), 128  
[compute\\_st1\\_iq\(\)](#) (in module [pyart.retrieve](#)), 128  
[compute\\_st2\\_iq\(\)](#) (in module [pyart.retrieve](#)), 128  
[compute\\_svp\(\)](#) (in module [pyart.retrieve](#)), 129  
[compute\\_ts\\_along\\_coord\(\)](#) (in module [pyart.retrieve](#)), 129  
[compute\\_vol\\_refl\(\)](#) (in module [pyart.retrieve](#)), 130  
[compute\\_vp\(\)](#) (in module [pyart.retrieve](#)), 130  
[compute\\_wbn\\_iq\(\)](#) (in module [pyart.retrieve](#)), 131  
[convert\\_data\(\)](#) (in module [pyart.aux\\_io](#)), 20  
[copy\(\)](#) ([pyart.correct.GateFilter](#) method), 81  
[copy\(\)](#) ([pyart.filters.GateFilter](#) method), 68  
[correct\\_bias\(\)](#) (in module [pyart.correct](#)), 88  
[correct\\_noise\\_rhohv\(\)](#) (in module [pyart.correct](#)), 88  
[correct\\_sys\\_phase\(\)](#) (in module [pyart.correct](#)), 89  
[correct\\_visibility\(\)](#) (in module [pyart.correct](#)), 89  
[cross\\_section\\_ppi\(\)](#) (in module [pyart.util](#)), 231  
[cross\\_section\\_rhi\(\)](#) (in module [pyart.util](#)), 232  
[cut\\_radar\(\)](#) (in module [pyart.util](#)), 232  
[cut\\_radar\\_spectra\(\)](#) (in module [pyart.util](#)), 232
- ## D
- [datetime\\_from\\_dataset\(\)](#) (in module [pyart.util](#)), 233  
[datetime\\_from\\_grid\(\)](#) (in module [pyart.util](#)), 233  
[datetime\\_from\\_radar\(\)](#) (in module [pyart.util](#)), 233  
[datetimes\\_from\\_dataset\(\)](#) (in module [pyart.util](#)), 233  
[datetimes\\_from\\_radar\(\)](#) (in module [pyart.util](#)), 233  
[dealias\\_fourdd\(\)](#) (in module [pyart.correct](#)), 89  
[dealias\\_region\\_based\(\)](#) (in module [pyart.correct](#)), 92  
[dealias\\_unwrap\\_phase\(\)](#) (in module [pyart.correct](#)), 93  
[despeckle\\_field\(\)](#) (in module [pyart.correct](#)), 94  
[det\\_sys\\_phase\\_ray\(\)](#) (in module [pyart.correct](#)), 94  
[detect\\_ml\(\)](#) (in module [pyart.retrieve](#)), 131
- ## E
- [est\\_rain\\_rate\\_a\(\)](#) (in module [pyart.retrieve](#)), 131  
[est\\_rain\\_rate\\_hydro\(\)](#) (in module [pyart.retrieve](#)), 132  
[est\\_rain\\_rate\\_kdp\(\)](#) (in module [pyart.retrieve](#)), 132  
[est\\_rain\\_rate\\_z\(\)](#) (in module [pyart.retrieve](#)), 133  
[est\\_rain\\_rate\\_za\(\)](#) (in module [pyart.retrieve](#)), 133  
[est\\_rain\\_rate\\_zkdp\(\)](#) (in module [pyart.retrieve](#)), 133  
[est\\_rain\\_rate\\_zpoly\(\)](#) (in module [pyart.retrieve](#)), 134  
[est\\_rhohv\\_rain\(\)](#) (in module [pyart.correct](#)), 95  
[est\\_vertical\\_windshear\(\)](#) (in module [pyart.retrieve](#)), 134  
[est\\_wind\\_profile\(\)](#) (in module [pyart.retrieve](#)), 134



`est_wind_vel()` (in module `pyart.retrieve`), 135  
`est_zdr_precip()` (in module `pyart.correct`), 95  
`est_zdr_snow()` (in module `pyart.correct`), 96  
`estimate_noise_hs74()` (in module `pyart.util`), 233  
`estimate_noise_ivic13()` (in module `pyart.util`), 233  
`example_roi_func_constant()` (in module `pyart.map`), 147  
`example_roi_func_dist()` (in module `pyart.map`), 147  
`example_roi_func_dist_beam()` (in module `pyart.map`), 147  
`exclude_above()` (`pyart.correct.GateFilter` method), 81  
`exclude_above()` (`pyart.filters.GateFilter` method), 68  
`exclude_all()` (`pyart.correct.GateFilter` method), 81  
`exclude_all()` (`pyart.filters.GateFilter` method), 68  
`exclude_below()` (`pyart.correct.GateFilter` method), 81  
`exclude_below()` (`pyart.filters.GateFilter` method), 68  
`exclude_equal()` (`pyart.correct.GateFilter` method), 82  
`exclude_equal()` (`pyart.filters.GateFilter` method), 69  
`exclude_gates()` (`pyart.correct.GateFilter` method), 82  
`exclude_gates()` (`pyart.filters.GateFilter` method), 69  
`exclude_inside()` (`pyart.correct.GateFilter` method), 82  
`exclude_inside()` (`pyart.filters.GateFilter` method), 69  
`exclude_invalid()` (`pyart.correct.GateFilter` method), 82  
`exclude_invalid()` (`pyart.filters.GateFilter` method), 69  
`exclude_masked()` (`pyart.correct.GateFilter` method), 82  
`exclude_masked()` (`pyart.filters.GateFilter` method), 69  
`exclude_none()` (`pyart.correct.GateFilter` method), 82  
`exclude_none()` (`pyart.filters.GateFilter` method), 69  
`exclude_not_equal()` (`pyart.correct.GateFilter` method), 82  
`exclude_not_equal()` (`pyart.filters.GateFilter` method), 69  
`exclude_outside()` (`pyart.correct.GateFilter` method), 82  
`exclude_outside()` (`pyart.filters.GateFilter` method), 69

`exclude_transition()` (`pyart.correct.GateFilter` method), 82  
`exclude_transition()` (`pyart.filters.GateFilter` method), 69  
`extract_sweeps()` (`pyart.core.Radar` method), 47  
`extract_sweeps()` (`pyart.core.RadarSpectra` method), 55

## F

`fetch_radar_time_profile()` (in module `pyart.retrieve`), 136  
`find_objects()` (in module `pyart.correct`), 97  
`from_u_and_v()` (`pyart.core.HorizontalWindProfile` class method), 41

## G

`gate_excluded` (`pyart.correct.GateFilter` attribute), 83  
`gate_excluded` (`pyart.filters.GateFilter` attribute), 70  
`gate_included` (`pyart.correct.GateFilter` attribute), 83  
`gate_included` (`pyart.filters.GateFilter` attribute), 70  
`GateFilter` (class in `pyart.correct`), 78  
`GateFilter` (class in `pyart.filters`), 65  
`generate_az_rhi_title()` (`pyart.graph.AirborneRadarDisplay` method), 157  
`generate_az_rhi_title()` (`pyart.graph.RadarDisplay` method), 188  
`generate_az_rhi_title()` (`pyart.graph.RadarMapDisplay` method), 201  
`generate_az_rhi_title()` (`pyart.graph.RadarMapDisplayBasemap` method), 216  
`generate_filename()` (`pyart.graph.AirborneRadarDisplay` method), 157  
`generate_filename()` (`pyart.graph.GridMapDisplay` method), 170  
`generate_filename()` (`pyart.graph.GridMapDisplayBasemap` method), 180  
`generate_filename()` (`pyart.graph.RadarDisplay` method), 188  
`generate_filename()` (`pyart.graph.RadarMapDisplay` method), 201  
`generate_filename()` (`pyart.graph.RadarMapDisplayBasemap` method), 217  
`generate_grid_title()` (`pyart.graph.GridMapDisplay` method),

170  
generate\_grid\_title() (pyart.graph.GridMapDisplayBasemap method), 180  
generate\_latitudinal\_level\_title() (pyart.graph.GridMapDisplay method), 171  
generate\_latitudinal\_level\_title() (pyart.graph.GridMapDisplayBasemap method), 180  
generate\_longitudinal\_level\_title() (pyart.graph.GridMapDisplay method), 171  
generate\_longitudinal\_level\_title() (pyart.graph.GridMapDisplayBasemap method), 180  
generate\_ray\_title() (pyart.graph.AirborneRadarDisplay method), 157  
generate\_ray\_title() (pyart.graph.RadarDisplay method), 188  
generate\_ray\_title() (pyart.graph.RadarMapDisplay method), 202  
generate\_ray\_title() (pyart.graph.RadarMapDisplayBasemap method), 217  
generate\_title() (pyart.graph.AirborneRadarDisplay method), 157  
generate\_title() (pyart.graph.RadarDisplay method), 189  
generate\_title() (pyart.graph.RadarMapDisplay method), 202  
generate\_title() (pyart.graph.RadarMapDisplayBasemap method), 217  
generate\_vpt\_title() (pyart.graph.AirborneRadarDisplay method), 158  
generate\_vpt\_title() (pyart.graph.RadarDisplay method), 189  
generate\_vpt\_title() (pyart.graph.RadarMapDisplay method), 202  
generate\_vpt\_title() (pyart.graph.RadarMapDisplayBasemap method), 217  
geographic\_to\_cartesian() (in module pyart.core), 60  
geographic\_to\_cartesian\_aeqd() (in module pyart.core), 61  
get\_azimuth() (pyart.core.Radar method), 47  
get\_azimuth() (pyart.core.RadarSpectra method), 55  
get\_basemap() (pyart.graph.GridMapDisplayBasemap method), 180  
get\_coeff\_attg() (in module pyart.retrieve), 136  
get\_dataset() (pyart.graph.GridMapDisplay method), 171  
get\_earth\_radius() (in module pyart.map), 147  
get\_elevation() (pyart.core.Radar method), 47  
get\_elevation() (pyart.core.RadarSpectra method), 55  
get\_end() (pyart.core.Radar method), 47  
get\_end() (pyart.core.RadarSpectra method), 56  
get\_field() (pyart.core.Radar method), 47  
get\_field() (pyart.core.RadarSpectra method), 56  
get\_freq\_band() (in module pyart.retrieve), 136  
get\_gate\_lat\_lon\_alt() (pyart.core.Radar method), 48  
get\_gate\_lat\_lon\_alt() (pyart.core.RadarSpectra method), 56  
get\_gate\_x\_y\_z() (pyart.core.Radar method), 48  
get\_gate\_x\_y\_z() (pyart.core.RadarSpectra method), 56  
get\_ivic\_flat\_reg\_var\_max() (in module pyart.util), 234  
get\_ivic\_pct() (in module pyart.util), 234  
get\_ivic\_snr\_thr() (in module pyart.util), 235  
get\_library() (in module pyart.aux\_io), 20  
get\_mask\_fzl() (in module pyart.correct), 97  
get\_nyquist\_vel() (pyart.core.Radar method), 49  
get\_nyquist\_vel() (pyart.core.RadarSpectra method), 57  
get\_point\_longitude\_latitude() (pyart.core.Grid method), 38  
get\_projparams() (pyart.core.Grid method), 39  
get\_slice() (pyart.core.Radar method), 49  
get\_slice() (pyart.core.RadarSpectra method), 57  
get\_start() (pyart.core.Radar method), 49  
get\_start() (pyart.core.RadarSpectra method), 57  
get\_start\_end() (pyart.core.Radar method), 49  
get\_start\_end() (pyart.core.RadarSpectra method), 57  
get\_sun\_hits() (in module pyart.correct), 98  
get\_sun\_hits\_ivic() (in module pyart.correct), 98  
Grid (class in pyart.core), 35  
grid\_displacement\_pc() (in module pyart.retrieve), 136  
grid\_from\_radars() (in module pyart.map), 148  
grid\_shift() (in module pyart.retrieve), 137  
GridMapDisplay (class in pyart.graph), 168  
GridMapDisplayBasemap (class in pyart.graph), 177

## H

HorizontalWindProfile (class in pyart.core), 39  
hydroclass\_semisupervised() (in module pyart.retrieve), 137

I

`include_above()` (*pyart.correct.GateFilter method*), 83

`include_above()` (*pyart.filters.GateFilter method*), 70

`include_all()` (*pyart.correct.GateFilter method*), 83

`include_all()` (*pyart.filters.GateFilter method*), 70

`include_below()` (*pyart.correct.GateFilter method*), 83

`include_below()` (*pyart.filters.GateFilter method*), 70

`include_equal()` (*pyart.correct.GateFilter method*), 83

`include_equal()` (*pyart.filters.GateFilter method*), 70

`include_gates()` (*pyart.correct.GateFilter method*), 83

`include_gates()` (*pyart.filters.GateFilter method*), 70

`include_inside()` (*pyart.correct.GateFilter method*), 83

`include_inside()` (*pyart.filters.GateFilter method*), 70

`include_none()` (*pyart.correct.GateFilter method*), 83

`include_none()` (*pyart.filters.GateFilter method*), 70

`include_not_equal()` (*pyart.correct.GateFilter method*), 83

`include_not_equal()` (*pyart.filters.GateFilter method*), 70

`include_not_masked()` (*pyart.correct.GateFilter method*), 83

`include_not_masked()` (*pyart.filters.GateFilter method*), 70

`include_not_transition()` (*pyart.correct.GateFilter method*), 83

`include_not_transition()` (*pyart.filters.GateFilter method*), 70

`include_outside()` (*pyart.correct.GateFilter method*), 84

`include_outside()` (*pyart.filters.GateFilter method*), 71

`include_valid()` (*pyart.correct.GateFilter method*), 84

`include_valid()` (*pyart.filters.GateFilter method*), 71

`info()` (*pyart.core.Radar method*), 49

`info()` (*pyart.core.RadarSpectra method*), 57

`init_gate_altitude()` (*pyart.core.Radar method*), 49

`init_gate_altitude()` (*pyart.core.RadarSpectra method*), 57

`init_gate_longitude_latitude()` (*pyart.core.Radar method*), 49

`init_gate_longitude_latitude()` (*pyart.core.RadarSpectra method*), 57

`init_gate_x_y_z()` (*pyart.core.Radar method*), 49

`init_gate_x_y_z()` (*pyart.core.RadarSpectra method*), 57

`init_point_altitude()` (*pyart.core.Grid method*), 39

`init_point_longitude_latitude()` (*pyart.core.Grid method*), 39

`init_point_x_y_z()` (*pyart.core.Grid method*), 39

`init_rays_per_sweep()` (*pyart.core.Radar method*), 49

`init_rays_per_sweep()` (*pyart.core.RadarSpectra method*), 58

`InTemporaryDirectory` (class in *pyart.testing*), 239

`interpol_spectra()` (in module *pyart.util*), 235

`intersection()` (in module *pyart.util*), 235

`interval_mean()` (in module *pyart.util*), 235

`interval_std()` (in module *pyart.util*), 236

`is_vpt()` (in module *pyart.util*), 236

`iso0_based_gate_filter()` (in module *pyart.filters*), 72

`iter_azimuth()` (*pyart.core.Radar method*), 49

`iter_azimuth()` (*pyart.core.RadarSpectra method*), 58

`iter_elevation()` (*pyart.core.Radar method*), 49

`iter_elevation()` (*pyart.core.RadarSpectra method*), 58

`iter_end()` (*pyart.core.Radar method*), 49

`iter_end()` (*pyart.core.RadarSpectra method*), 58

`iter_field()` (*pyart.core.Radar method*), 49

`iter_field()` (*pyart.core.RadarSpectra method*), 58

`iter_slice()` (*pyart.core.Radar method*), 49

`iter_slice()` (*pyart.core.RadarSpectra method*), 58

`iter_start()` (*pyart.core.Radar method*), 50

`iter_start()` (*pyart.core.RadarSpectra method*), 58

`iter_start_end()` (*pyart.core.Radar method*), 50

`iter_start_end()` (*pyart.core.RadarSpectra method*), 58

`ivic_flat_reg_var_max_table()` (in module *pyart.util*), 236

`ivic_pct_table()` (in module *pyart.util*), 236

`ivic_snr_thr_table()` (in module *pyart.util*), 237

J

`join_radar()` (in module *pyart.util*), 237

`join_spectra()` (in module *pyart.util*), 237

K

`kdp_leastsquare_double_window()` (in module *pyart.retrieve*), 138

`kdp_leastsquare_single_window()` (in module *pyart.retrieve*), 138



kdp\_maesaka() (in module *pyart.retrieve*), 139  
 kdp\_schneebeli() (in module *pyart.retrieve*), 140  
 kdp\_vulpiani() (in module *pyart.retrieve*), 141

## L

label\_xaxis\_r() (*pyart.graph.AirborneRadarDisplay* method), 158  
 label\_xaxis\_r() (*pyart.graph.RadarDisplay* method), 189  
 label\_xaxis\_r() (*pyart.graph.RadarMapDisplay* method), 202  
 label\_xaxis\_r() (*pyart.graph.RadarMapDisplayBasemap* method), 218  
 label\_xaxis\_rays() (*pyart.graph.AirborneRadarDisplay* static method), 158  
 label\_xaxis\_rays() (*pyart.graph.RadarDisplay* static method), 189  
 label\_xaxis\_rays() (*pyart.graph.RadarMapDisplay* static method), 202  
 label\_xaxis\_rays() (*pyart.graph.RadarMapDisplayBasemap* static method), 218  
 label\_xaxis\_time() (*pyart.graph.AirborneRadarDisplay* static method), 158  
 label\_xaxis\_time() (*pyart.graph.RadarDisplay* static method), 189  
 label\_xaxis\_time() (*pyart.graph.RadarMapDisplay* static method), 202  
 label\_xaxis\_time() (*pyart.graph.RadarMapDisplayBasemap* static method), 218  
 label\_xaxis\_x() (*pyart.graph.AirborneRadarDisplay* method), 158  
 label\_xaxis\_x() (*pyart.graph.RadarDisplay* method), 189  
 label\_xaxis\_x() (*pyart.graph.RadarMapDisplay* method), 202  
 label\_xaxis\_x() (*pyart.graph.RadarMapDisplayBasemap* method), 218  
 label\_yaxis\_field() (*pyart.graph.AirborneRadarDisplay* method), 158  
 label\_yaxis\_field() (*pyart.graph.RadarDisplay* method), 189  
 label\_yaxis\_field() (*pyart.graph.RadarMapDisplay* method), 202  
 label\_yaxis\_field() (*pyart.graph.RadarMapDisplayBasemap* method), 218

label\_yaxis\_y() (*pyart.graph.AirborneRadarDisplay* method), 158  
 label\_yaxis\_y() (*pyart.graph.RadarDisplay* method), 189  
 label\_yaxis\_y() (*pyart.graph.RadarMapDisplay* method), 202  
 label\_yaxis\_y() (*pyart.graph.RadarMapDisplayBasemap* method), 218  
 label\_yaxis\_z() (*pyart.graph.AirborneRadarDisplay* method), 158  
 label\_yaxis\_z() (*pyart.graph.RadarDisplay* method), 189  
 label\_yaxis\_z() (*pyart.graph.RadarMapDisplay* method), 202  
 label\_yaxis\_z() (*pyart.graph.RadarMapDisplayBasemap* method), 218

## M

make\_empty\_grid() (in module *pyart.testing*), 241  
 make\_empty\_ppi\_radar() (in module *pyart.testing*), 241  
 make\_empty\_rhi\_radar() (in module *pyart.testing*), 241  
 make\_normal\_storm() (in module *pyart.testing*), 242  
 make\_single\_ray\_radar() (in module *pyart.testing*), 242  
 make\_storm\_grid() (in module *pyart.testing*), 242  
 make\_target\_grid() (in module *pyart.testing*), 242  
 make\_target\_radar() (in module *pyart.testing*), 242  
 make\_time\_unit\_str() (in module *pyart.io*), 4  
 make\_velocity\_aliased\_radar() (in module *pyart.testing*), 242  
 make\_velocity\_aliased\_rhi\_radar() (in module *pyart.testing*), 242  
 map\_gates\_to\_grid() (in module *pyart.map*), 148  
 map\_profile\_to\_gates() (in module *pyart.retrieve*), 142  
 map\_to\_grid() (in module *pyart.map*), 149  
 mean\_of\_two\_angles() (in module *pyart.util*), 237  
 mean\_of\_two\_angles\_deg() (in module *pyart.util*), 237  
 melting\_layer\_giangrande() (in module *pyart.retrieve*), 142  
 melting\_layer\_hydroclass() (in module *pyart.retrieve*), 144  
 moment\_and\_texture\_based\_gate\_filter() (in module *pyart.filters*), 72  
 moment\_based\_gate\_filter() (in module *pyart.correct*), 99  
 moment\_based\_gate\_filter() (in module *pyart.filters*), 73

## P

- `phase_proc_lp()` (in module `pyart.correct`), 100
- `phase_proc_lp_gf()` (in module `pyart.correct`), 101
- `plot()` (`pyart.graph.AirborneRadarDisplay` method), 158
- `plot()` (`pyart.graph.RadarDisplay` method), 189
- `plot()` (`pyart.graph.RadarMapDisplay` method), 203
- `plot()` (`pyart.graph.RadarMapDisplayBasemap` method), 218
- `plot_azimuth_to_rhi()` (`pyart.graph.AirborneRadarDisplay` method), 158
- `plot_azimuth_to_rhi()` (`pyart.graph.RadarDisplay` method), 190
- `plot_azimuth_to_rhi()` (`pyart.graph.RadarMapDisplay` method), 203
- `plot_azimuth_to_rhi()` (`pyart.graph.RadarMapDisplayBasemap` method), 218
- `plot_basemap()` (`pyart.graph.GridMapDisplayBasemap` method), 180
- `plot_colorbar()` (`pyart.graph.AirborneRadarDisplay` method), 160
- `plot_colorbar()` (`pyart.graph.GridMapDisplay` method), 171
- `plot_colorbar()` (`pyart.graph.GridMapDisplayBasemap` method), 181
- `plot_colorbar()` (`pyart.graph.RadarDisplay` method), 191
- `plot_colorbar()` (`pyart.graph.RadarMapDisplay` method), 204
- `plot_colorbar()` (`pyart.graph.RadarMapDisplayBasemap` method), 219
- `plot_cross_hair()` (`pyart.graph.AirborneRadarDisplay` static method), 160
- `plot_cross_hair()` (`pyart.graph.RadarDisplay` static method), 191
- `plot_cross_hair()` (`pyart.graph.RadarMapDisplay` static method), 204
- `plot_cross_hair()` (`pyart.graph.RadarMapDisplayBasemap` static method), 220
- `plot_crosshairs()` (`pyart.graph.GridMapDisplay` method), 171
- `plot_crosshairs()` (`pyart.graph.GridMapDisplayBasemap` method), 181
- `plot_grid()` (`pyart.graph.GridMapDisplay` method), 172
- `plot_grid()` (`pyart.graph.GridMapDisplayBasemap` method), 181
- `plot_grid_contour()` (`pyart.graph.GridMapDisplay` method), 173
- `plot_grid_lines()` (`pyart.graph.AirborneRadarDisplay` static method), 160
- `plot_grid_lines()` (`pyart.graph.RadarDisplay` static method), 191
- `plot_grid_lines()` (`pyart.graph.RadarMapDisplay` static method), 205
- `plot_grid_lines()` (`pyart.graph.RadarMapDisplayBasemap` static method), 220
- `plot_label()` (`pyart.graph.AirborneRadarDisplay` method), 160
- `plot_label()` (`pyart.graph.RadarDisplay` method), 192
- `plot_label()` (`pyart.graph.RadarMapDisplay` method), 205
- `plot_label()` (`pyart.graph.RadarMapDisplayBasemap` method), 220
- `plot_labels()` (`pyart.graph.AirborneRadarDisplay` method), 161
- `plot_labels()` (`pyart.graph.RadarDisplay` method), 192
- `plot_labels()` (`pyart.graph.RadarMapDisplay` method), 205
- `plot_labels()` (`pyart.graph.RadarMapDisplayBasemap` method), 220
- `plot_latitude_slice()` (`pyart.graph.GridMapDisplay` method), 174
- `plot_latitude_slice()` (`pyart.graph.GridMapDisplayBasemap` method), 182
- `plot_latitudinal_level()` (`pyart.graph.GridMapDisplay` method), 174
- `plot_latitudinal_level()` (`pyart.graph.GridMapDisplayBasemap` method), 183
- `plot_latlon_level()` (`pyart.graph.GridMapDisplay` method), 175
- `plot_latlon_slice()` (`pyart.graph.GridMapDisplay` method), 176
- `plot_line_geo()` (`pyart.graph.RadarMapDisplay` method), 205
- `plot_line_geo()` (`pyart.graph.RadarMapDisplayBasemap` method), 221
- `plot_line_xy()` (`pyart.graph.RadarMapDisplay` method), 205

`plot_line_xy()` (*pyart.graph.RadarMapDisplayBasemap method*), 221  
`plot_longitude_slice()` (*pyart.graph.GridMapDisplay method*), 176  
`plot_longitude_slice()` (*pyart.graph.GridMapDisplayBasemap method*), 183  
`plot_longitudinal_level()` (*pyart.graph.GridMapDisplay method*), 176  
`plot_longitudinal_level()` (*pyart.graph.GridMapDisplayBasemap method*), 184  
`plot_point()` (*pyart.graph.RadarMapDisplay method*), 206  
`plot_point()` (*pyart.graph.RadarMapDisplayBasemap method*), 221  
`plot_ppi()` (*pyart.graph.AirborneRadarDisplay method*), 161  
`plot_ppi()` (*pyart.graph.RadarDisplay method*), 192  
`plot_ppi()` (*pyart.graph.RadarMapDisplay method*), 206  
`plot_ppi()` (*pyart.graph.RadarMapDisplayBasemap method*), 221  
`plot_ppi_map()` (*pyart.graph.RadarMapDisplay method*), 207  
`plot_ppi_map()` (*pyart.graph.RadarMapDisplayBasemap method*), 223  
`plot_range_ring()` (*pyart.graph.AirborneRadarDisplay static method*), 162  
`plot_range_ring()` (*pyart.graph.RadarDisplay static method*), 194  
`plot_range_ring()` (*pyart.graph.RadarMapDisplay method*), 209  
`plot_range_ring()` (*pyart.graph.RadarMapDisplayBasemap method*), 224  
`plot_range_rings()` (*pyart.graph.AirborneRadarDisplay method*), 162  
`plot_range_rings()` (*pyart.graph.RadarDisplay method*), 194  
`plot_range_rings()` (*pyart.graph.RadarMapDisplay method*), 209  
`plot_range_rings()` (*pyart.graph.RadarMapDisplayBasemap method*), 225  
`plot_ray()` (*pyart.graph.AirborneRadarDisplay method*), 163  
`plot_ray()` (*pyart.graph.RadarDisplay method*), 194  
`plot_ray()` (*pyart.graph.RadarMapDisplay method*), 209  
`plot_ray()` (*pyart.graph.RadarMapDisplayBasemap method*), 225  
`plot_rhi()` (*pyart.graph.AirborneRadarDisplay method*), 163  
`plot_rhi()` (*pyart.graph.RadarDisplay method*), 195  
`plot_rhi()` (*pyart.graph.RadarMapDisplay method*), 210  
`plot_rhi()` (*pyart.graph.RadarMapDisplayBasemap method*), 225  
`plot_sweep_grid()` (*pyart.graph.AirborneRadarDisplay method*), 165  
`plot_vpt()` (*pyart.graph.AirborneRadarDisplay method*), 166  
`plot_vpt()` (*pyart.graph.RadarDisplay method*), 196  
`plot_vpt()` (*pyart.graph.RadarMapDisplay method*), 211  
`plot_vpt()` (*pyart.graph.RadarMapDisplayBasemap method*), 227  
`polar_to_cartesian()` (*in module pyart.map*), 151  
`prepare_for_read()` (*in module pyart.io*), 4  
`projection_proj` (*pyart.core.Grid attribute*), 39  
`ptoa_to_sf()` (*in module pyart.correct*), 102  
`pyart.aux_io` (*module*), 18  
`pyart.bridge` (*module*), 61  
`pyart.core` (*module*), 34  
`pyart.correct` (*module*), 75  
`pyart.filters` (*module*), 63  
`pyart.graph` (*module*), 151  
`pyart.io` (*module*), 1  
`pyart.map` (*module*), 146  
`pyart.retrieve` (*module*), 110  
`pyart.testing` (*module*), 238  
`pyart.util` (*module*), 228

## Q

`quasi_vertical_profile()` (*in module pyart.retrieve*), 145

## R

`Radar` (*class in pyart.core*), 41  
`radar_from_spectra()` (*in module pyart.util*), 237  
`RadarDisplay` (*class in pyart.graph*), 185  
`RadarMapDisplay` (*class in pyart.graph*), 197  
`RadarMapDisplayBasemap` (*class in pyart.graph*), 213  
`RadarSpectra` (*class in pyart.core*), 50  
`read()` (*in module pyart.io*), 4  
`read_arm_sonde()` (*in module pyart.io*), 5  
`read_arm_sonde_vap()` (*in module pyart.io*), 5  
`read_bin()` (*in module pyart.aux\_io*), 20

read\_cartesian\_metrane() (in module *pyart.aux\_io*), 20  
 read\_cfl() (in module *pyart.aux\_io*), 21  
 read\_cfradial() (in module *pyart.io*), 5  
 read\_cfradial2() (in module *pyart.io*), 6  
 read\_chl() (in module *pyart.io*), 7  
 read\_d3r\_gcpx\_nc() (in module *pyart.aux\_io*), 21  
 read\_edge\_netcdf() (in module *pyart.aux\_io*), 22  
 read\_file\_c() (in module *pyart.aux\_io*), 22  
 read\_file\_py() (in module *pyart.aux\_io*), 22  
 read\_gamic() (in module *pyart.aux\_io*), 23  
 read\_gif() (in module *pyart.aux\_io*), 23  
 read\_grid() (in module *pyart.io*), 7  
 read\_grid\_mdv() (in module *pyart.io*), 8  
 read\_iq() (in module *pyart.aux\_io*), 24  
 read\_kazr() (in module *pyart.aux\_io*), 25  
 read\_mdv() (in module *pyart.io*), 8  
 read\_metrane() (in module *pyart.aux\_io*), 25  
 read\_nexrad\_archive() (in module *pyart.io*), 9  
 read\_nexrad\_cdm() (in module *pyart.io*), 10  
 read\_nexrad\_level3() (in module *pyart.io*), 11  
 read\_noxp\_iphex\_nc() (in module *pyart.aux\_io*), 26  
 read\_odim\_h5() (in module *pyart.aux\_io*), 26  
 read\_pattern() (in module *pyart.aux\_io*), 27  
 read\_product\_c() (in module *pyart.aux\_io*), 27  
 read\_product\_py() (in module *pyart.aux\_io*), 27  
 read\_psr\_cpi\_header() (in module *pyart.aux\_io*), 28  
 read\_psr\_header() (in module *pyart.aux\_io*), 28  
 read\_psr\_spectra() (in module *pyart.aux\_io*), 28  
 read\_radx() (in module *pyart.aux\_io*), 28  
 read\_rainbow\_psr() (in module *pyart.aux\_io*), 28  
 read\_rainbow\_psr\_spectra() (in module *pyart.aux\_io*), 29  
 read\_rainbow\_wrl() (in module *pyart.aux\_io*), 30  
 read\_rsl() (in module *pyart.io*), 12  
 read\_sigmet() (in module *pyart.io*), 12  
 read\_sinarame\_h5() (in module *pyart.aux\_io*), 31  
 read\_spectra() (in module *pyart.aux\_io*), 31  
 read\_uf() (in module *pyart.io*), 14  
 rolling\_window() (in module *pyart.util*), 238

## S

scanning\_losses() (in module *pyart.correct*), 103  
 selfconsistency\_bias() (in module *pyart.correct*), 103  
 selfconsistency\_bias2() (in module *pyart.correct*), 104  
 selfconsistency\_kdp\_phidp() (in module *pyart.correct*), 106  
 set\_aspect\_ratio() (*pyart.graph.AirborneRadarDisplay* static method), 167

set\_aspect\_ratio() (*pyart.graph.RadarDisplay* static method), 197  
 set\_aspect\_ratio() (*pyart.graph.RadarMapDisplay* static method), 213  
 set\_aspect\_ratio() (*pyart.graph.RadarMapDisplayBasemap* static method), 228  
 set\_limits() (*pyart.graph.AirborneRadarDisplay* static method), 167  
 set\_limits() (*pyart.graph.RadarDisplay* static method), 197  
 set\_limits() (*pyart.graph.RadarMapDisplay* static method), 213  
 set\_limits() (*pyart.graph.RadarMapDisplayBasemap* static method), 228  
 simulated\_vel\_from\_profile() (in module *pyart.util*), 238  
 smooth\_masked() (in module *pyart.correct*), 107  
 smooth\_phidp\_double\_window() (in module *pyart.correct*), 107  
 smooth\_phidp\_single\_window() (in module *pyart.correct*), 108  
 snr\_based\_gate\_filter() (in module *pyart.filters*), 74  
 solar\_flux\_lookup() (in module *pyart.correct*), 108  
 steiner\_conv\_strat() (in module *pyart.retrieve*), 145  
 sun\_position\_mfr() (in module *pyart.correct*), 108  
 sun\_position\_pysolar() (in module *pyart.correct*), 109  
 sun\_power() (in module *pyart.correct*), 109  
 sun\_retrieval() (in module *pyart.correct*), 109

## T

temp\_based\_gate\_filter() (in module *pyart.filters*), 75  
 texture() (in module *pyart.util*), 238  
 texture\_along\_ray() (in module *pyart.util*), 238  
 to\_vpt() (in module *pyart.util*), 238  
 to\_xarray() (*pyart.core.Grid* method), 39

## U

u\_wind(*pyart.core.HorizontalWindProfile* attribute), 41

## V

v\_wind(*pyart.core.HorizontalWindProfile* attribute), 41  
 velocity\_azimuth\_display() (in module *pyart.retrieve*), 146  
 visibility\_based\_gate\_filter() (in module *pyart.filters*), 75

## W

wgs84\_to\_swissCH1903() (*in module pyart.core*),  
61  
write() (*pyart.core.Grid method*), 39  
write\_cfradial() (*in module pyart.io*), 14  
write\_grid() (*in module pyart.io*), 15  
write\_grid\_geotiff() (*in module pyart.io*), 16  
write\_grid\_mdv() (*in module pyart.io*), 17  
write\_odim\_h5() (*in module pyart.aux\_io*), 32  
write\_sinaramc\_cfradial() (*in module  
pyart.aux\_io*), 33  
write\_spectra() (*in module pyart.aux\_io*), 33  
write\_uf() (*in module pyart.io*), 18