Py-ART Cheat Sheet

Learn More About Py-ART[1] at https://arm-doe.github.io/pyart/

Py-ART Introduction

The Python ARM Radar Toolkit, Py-ART, is a Python module containing a collection of weather radar algorithms and utilities. Py-ART is used by the Atmospheric Radiation Measurement (ARM) Climate Research Facility for working with data from a number of its precipitation and cloud radars, but has been designed so that it can be used by others in the radar and atmospheric communities. Py-ART has the ability to ingest (read) from a number of common weather radar formats. Radar data can be written to NetCDF files which conform to the CF/Radial convention. Py-ART also contains routines which can produce common radar plots including PPIs and RHIs. Algorithms in the module are able to performs a number of corrections on the radar moment data in antenna coordinate. A sophisticated mapping routines is able to efficiently create uniform Cartesian grids of radar fields from one or more radars.

Installation

The recommended way to install Py-ART is by installing Anaconda or Miniconda, then create an environment and activate it:

- Then create a conda environment:
  $ conda create -n pyart python=3.9

- Activate the Py-ART environment:
  $ conda activate pyart

- Then install Py-ART:
  $ conda install -c conda-forge arm_pyart

For the most recent Py-ART developments, you need to get the latest main branch from github.com

- Clone the Py-ART repository and go into the directory:
  $ git clone git@github.com:ARM-DOE/pyart.git
  $ python setup.py install

Contact Information

GitHub Discussions:
https://github.com/ARM-DOE/pyart/discussions

Email:
scollis@anl.gov
zsherman@anl.gov
mgrover@anl.gov

Getting Started

Cookbooks for Getting Started:
https://cookbooks.projectpythia.org/radar-cookbook/README.html

>>> import pyart — To import Py-ART.
>>> print(pyart.__version__) — Check version.

Reading and Writing Data

Reading Data

>>> radar = pyart.io.read(filename)
- Read a file supported by RSL.
- Read a MDV file.
- Read a Sigmet (IRIS) product file.
- Read a Cfradial netCDF file.
- Read a CSU-CHILL CHL file.
- Read a NEXRAD Level 2 Archive file.
- Read a Common Data Model NEXRAD 2 file.
- Read a NEXRAD Level 3 product.
- Read a UF File.

>>> radar = pyart.aux_io.read_d3r_gcpex_nc(file)
- Read a D3R GCPEX netCDF file.

>>> radar = pyart.aux_io.read_gamic(filename)
- Read a GAMIC hdf5 file.

Reading and Writing Data

>>> radar = pyart.aux_io.read_kazr(filename)
- Read K-band ARM Zenith Radar (KAZR) data.

>>> radar = pyart.aux_io.read_noxp_iphex_nc(file)
- Read a NOXP IPHEX netCDF file.

>>> radar = pyart.aux_io.read_odim_h5(filename)
- Read a ODIM_H5 file.

>>> radar = pyart.aux_io.read_radx(filename)
- Read a radx file by using RadxConvert.

>>> radar = pyart.aux_io.read_pattern(filename)
- Read PATTERN project X-band radar file.

>>> radar = pyart.aux_io.read_rainbow_wrl(file)
- Read a RAINBOW file.

Writing Radar Data

>>> pyart.io.write_cfradial(filename, radar)
>>> pyart.io.write_uf(filename, radar)

Reading Grid Data

>>> grid = pyart.io.read_grid(filename)
>>> grid = pyart.io.read_grid_mdv(filename)

Writing Grid Data

>>> pyart.io.write_grid(filename, grid)
>>> pyart.io.write_grid_mdv(filename, grid)
>>> pyart.io.write_grid_geotiff(grid, filename)

Reading Sonde Data

>>> sonde = pyart.io.read_sonde(filename)
>>> sonde = pyart.io.read_sonde_vap(
    filename[, radar, . . .])

Graphing Data

Radar Data

>>> pyart.graph.RadarDisplay(radar)
- Display object to plot data from a radar object.

>>> pyart.graph.RadarMapDisplay(radar)
- Plots data to a geographic map using Cartopy.

>>> pyart.graph.AirborneRadarDisplay(radar)
- Plots data from a airborne radar object.

After defining the display objects above, each have specific functions for plots. For Example:

>>> display.plot(field[, sweep])
>>> display.plot_ppi(field[, sweep, vmin, . . .])
>>> display.plot_ppi_map(field[, sweep, . . .])
>>> display.plot_rhi(field[, sweep, vmin, . . .])
>>> display.plot_vpt(field[, vmin, vmax, . . .])

Grid Data

>>> pyart.graph.GridMapDisplay(grid)
>>> display.plot_grid(field[, level, vmin, vmax, . . .])
>>> display.plot_latitudinal_level(
    field, y_index[, ...])
>>> display.plot_longitudinal_level(
    field, x_index[, ...])

Mapping Data

>>> grid = pyart.map.grid_from_radars(
    radars, grid_shape, grid_limits)
- Map one or more radars to a Cartesian grid returning a grid object.

>>> grids = pyart.map.map_to_grid(
    radars, grid_shape, grid_limits)
- Map one or more radars to a Cartesian grid.

>>> grids = pyart.map.map_gates_to_grid(
    radars, grid_shape, grid_limits)
- Map gates from radar(s) to a Cartesian grid.

Core Radar and Grid

Radar

The radar class and functions within for handling the radar class meta data.

>>> radar = pyart.io.read(filename)
>>> radar.info()
- Prints information on the radar.

>>> radar.add_field(field_name, dict[, ...])
- Adds a new field to the radar object or replaces an existing one.

>>> radar_sweep = radar.extract_sweeps(sweeps)
- Extracts a sweep and returns a radar object for that sweep.

>>> x, y, z = radar.get_gate_x_y_z(sweep[, ...])
- Returns the x, y and z gate locations for a given sweep.

Grid

The grid class and functions within for handling the grid class meta data.

>>> grid = pyart.io.read_grid(filename)
>>> grid.to_xarray()
- Returns the grid object in xarray.

>>> grid.write(filename[, ...])
- Writes the grid to a NetCDF file.

>>> grid.add_field(field_name, dict[, ...])
- Adds a new field to the grid object or replaces an existing one.

>>> grid.get_point_longitude_latitude(level, [, ...])
- Returns the latitude and longitude values for the level at a given height of grid values.

Wind Profile

Creates a Horizontal Wind Profile.

>>> profile = pyart.core.HorizontalWindProfile(
    height, speed, direction)
>>> u_wind = profile.u_wind()
- U component of horizontal wind in meters per second.

>>> v_wind = profile.v_wind()
- V component of horizontal wind in meters per second.

Transforms

Transformation between coordinate systems.

>>> x, y, z = pyart.core.antenna_to_cartesian(
    ranges, azimuths, elevations)
- Return Cartesian coordinates from antenna coordinates.

>>> x, y, z = pyart.core.geographic_to_cartesian(
    lon, lat, projparams)
- Geographic to Cartesian coordinate transform.

>>> x, y = pyart.core.geographic_to_cartesian_aeqd(
    lon, lat, lon_0, lat_0, R=6370997)
- Azimuthal equidistant geographic to Cartesian coordinate transform.

>>> lo, la = pyart.core.cartesian_to_geographic_aeqd(
    x, y, lon_0, lat_0, R=6370997)
- Azimuthal equidistant Cartestian to geographic coordinate transform.

>>> lo, la = pyart.core.cartesian_to_geographic(
    x, y, projparams)
- Cartesian to Geographic coordinate transform.

>>> x, y, z = (
    pyart.core.antenna_to_cartesian_aircraft_relative(
    ranges, rot, tilt)
- Calculate aircraft-relative Cartesian coordinates from radar coordinates.

Py-ART Cheat Sheet

Learn More About Py-ART[1] at https://arm-doe.github.io/pyart/

Corrections

GateFilters

A class for building a boolean arrays for filtering gates based on a set of condition in a radar field.

```
>>> gatefilter = pyart.correct.GateFilter(radar)
>>> gatefilter.exclude_all()
>>> gatefilter.exclude_below(field, 10)
>>> gatefilter.exclude_masked(field)
```

Velocity Unfolding

```
>>> corr_vel = pyart.correct.dealias_fourdd(
        radar[, ...])
```

- Dealias Doppler velocities using 4DD algorithm [2].

```
>>> corr_vel = pyart.correct.dealias_region_based(
        radar[, ...])
```

- Dealias velocities using a region based algorithm.

```
>>> corr_vel = pyart.correct.dealias_unwrap_phase(
        radar[, ...])
```

- Dealias Doppler velocities using multi-dimensional phase unwrapping.

Other Corrections

```
>>> atten, co_z = pyart.correct.calculate_attenuation(
        radar, z_offset[, ...])
```

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

```
>>> proc_kdp, re_kdp = pyart.correct.phase_proc_lp(
        radar, offset[, ...])
```

- Phase process using a LP method [3].

```
>>> filter = pyart.correct.despeckle_field(
        radar, field[, ...])
```

- Despeckle a radar volume by identifying small objects in each scan and masking them out.

```
>>> rhohv = pyart.correct.correct_noise_rhohv(
        radar[, ...])
```

- Corrects RhoHV for noise [4].

Retrievals

```
>>> qvp = pyart.retrieve.quasi_vertical_profile(
        radar[, ...])
```

- Creates a quasi vertical profile object. based on Ryzhkov et al [12].

```
>>> vad = pyart.retrieve.velocity_azimuth_display(
        radar[, ...])
```

- Creates a velocity azimuth display object. containing U Wind, V Wind and Height. based on Michelson et al [13].

```
>>> kdp, phif, phir = pyart.retrieve.kdp_maesaka(
        radar[, ...])
```

- Computes the specific differential phase (KDP) based on Maesaka et al [5].

```
>>> snr = pyart.retrieve.compute_snr(radar[, ...])
```

- Computes SNR from a reflectivity field and the noise in dBZ.

```
>>> L = pyart.retrieve.compute_l(radar[, ...])
```

- Computes Rhohv in logarithmic scale according to L=-log10(1-RhoHV)

```
>>> cdr = pyart.retrieve.compute_cdr(radar[, ...])
```

- Computes the Circular Depolarization Ratio.

```
>>> eclass = pyart.retrieve.steiner_conv_strat(
        grid[, ...])
```

- Partition reflectivity into convective-stratiform using the Steiner et al. [6].

```
>>> hy =
pyart.retrieve.hydroclass_semisupervised(
        radar[, ...])
```

- Classifies precipitation echoes following the approach by Besic et al. [7].

```
>>> tex = pyart.retrieve.texture_of_complex_phase(
        radar[, ...])
```

- Calculate the texture of the differential phase [8].

Retrievals

```
>>> rain = pyart.retrieve.est_rain_rate_z(
        radar[, alpha, beta, ...])
```

- Estimates rainfall rate from reflectivity using a power law.

```
>>> rain = pyart.retrieve.est_rain_rate_a(
        radar[, alpha, beta, ...])
```

- Estimates rainfall rate from specific attenuation using alpha power law [9], [10].

```
>>> rain = pyart.retrieve.est_rain_rate_kdp(
        radar[, alpha, beta, ...])
```

- Estimates rainfall from kdp using alpha power.

Utilities

Direction Statistics

```
>>> mean = pyart.util.angular_mean(angles)
```

- Compute the mean of a distribution of angles in radians.

```
>>> std_dev = pyart.util.angular_std(angles)
```

- Compute the standard deviation of a distribution of angles in radians.

```
>>> mean = pyart.util.angular_mean_deg(angles)
```

- Compute the mean of a distribution of angles in degrees.

```
>>> std_dev = pyart.util.angular_std_deg(angles)
```

- Compute the standard deviation of a distribution of angles in degrees.

```
>>> mean = pyart.util.interval_mean(
        dist, interval_min, interval_max)
```

- Compute the mean of a distribution within an interval.

```
>>> std_dev = pyart.util.interval_std(
        dist, interval_min, interval_max)
```

- Compute the standard deviation of a distribution within an interval.

Miscellaneous Utilities

```
>>> radar_column = pyart.util.get_column_rays(
        radar, azimuths[, ...])
```

- Given the location (in latitude,longitude) of a target, return the rays that correspond to radar column above the target.

```
>>> radar_rhi = pyart.util.cross_section_ppi(
        radar, target_azimuths[, ...])
```

- Extract cross sections from a PPI volume along one or more azimuth angles.

```
>>> radar_ppi = pyart.util.cross_section_rhi(
        radar, target_elevations)
```

- Extract cross sections from an RHI volume along one or more elevation angles.

```
>>> mean, ther, var, noise = (
        pyart.util.estimate_noise_hs74(
                spectrum[, navg])
```

- Estimate noise parameters of a Doppler spectrum [11].

```
>>> pyart.util.to_vpt(radar[, single_scan])
```

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

```
>>> radar = pyart.util.join_radar(radar1, radar2)
```

- Combine two radar instances into one.

```
>>> sim_vel = pyart.util.simulated_vel_from_profile(
        radar, profile[, ...])
```

- Create simulated radial velocities from a profile of horizontal winds.

```
>>> texture = pyart.util.texture_along_ray(
        radar, var[, wind_size])
```

- Compute field texture along ray using a user specified window size.

```
>>> win = pyart.util.rolling_window(a, window)
```

- Create a rolling window object for application of functions.

```
>>> std_dev = pyart.util.angular_texture_2d(
        image, N, interval)
```

- Compute the angular texture of an image.

References

[1] Helmus, J.J. Collis, S.M., 2016: The Python ARM Radar Toolkit (Py-ART), a Library for Working with Weather Radar Data in the Python Programming Language. *Journal of Open Research Software*. **4(1)**, p.e25.

[2] C. N. James and R. A Houze Jr, 2001: A Real-Time Four-Dimensional Doppler Dealising Scheme. *Journal of Atmospheric and Oceanic Technology*, **18**, 1674.

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

[4] Gourley et al., 2006: Data Quality of the Meteo-France C-Band Polarimetric Radar. *JAOT*, **23**, 1340-1356

[5] 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*.

[6] 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.

[7] Besic, N., Figueras i Ventura, J., Grazioli, J., Gabella, M., Germann, U., and Berne, A., 2016: Hydrometeor classification through statistical clustering of polarimetric radar measurements: a semi-supervised approach. *Atmos. Meas. Tech.*, **9**, 4425-4445

[8] Gourley, J. J., P. Tabary, and J. Parent du Chatelet, 2007: A fuzzy logic algorithm for the separation of precipitating from nonprecipitating echoes using polarimetric radar observations. *Journal of Atmospheric and Oceanic Technology* **24(8)**, 1439-1451

[9] Diederich M., Ryzhkov A., Simmer C., Zhang P. and Tromel S., 2015: Use of Specific Attenuation for Rainfall Measurement at X-Band Radar Wavelenghts. Part I: Radar Calibration and Partial Beam Blockage Estimation. *Journal of Hydrometeorology*, **16**, 487-502.

[10] 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.

[11] Hildebrand P.H., and R. S. Sekhon, 1974: Objective Determination of the Noise Level in Doppler Spectra. *Journal of Applied Meteorology*, **13**, 808-811.

[12] 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. *Journal Atmospheric and Oceanic Technology*

[13] Michelson, D. B., Andersson, T., Koistinen, J., Collier, C. G., Riedl, J., Szturc, J., Gjertsen, U., Nielsen, A. and Overgaard, S. 2000: BALTEX Radar Data Centre Products and their Methodologies. In SMHI Reports. *Meteorology and Climatology*, Swedish Meteorological and Hydrological Institute, Norrkoping.