PyCINRAD

Release 1.7.1

Puyuan Du

May 05, 2022

CONTENTS:

PyCINRAD

CHAPTER

ONE

API REFERENCE

1.1 Subpackages

1.1.1 cinrad.io package

cinrad.io.base module

class cinrad.io.base.**RadarBase**

Bases: abc.ABC

Base class for readers in *cinrad.io*. Only used when subclassed

available_product(*tilt: int*) → list

Get all available products in given tilt

cinrad.io.level2 module

class cinrad.io.level2.**CinradReader**(*file: Any*, *radar_type: Optional[str] = None*)

Bases: *cinrad.io.base.RadarBase*

Class reading old-version CINRAD data.

Args: file (str, IO): Path points to the file or a file object.

radar_type (str): Type of the radar.

file_name (str): Name of the file, only used when *file* argument is a file object.

get_azimuth_angles(*scans: Optional[int] = None*) → numpy.ndarray

Get index of input azimuth angle (radian)

get_data(*tilt: int*, *drange: Union[int, float]*, *dtype: str*) → xarray.core.dataset.Dataset

Get radar data with extra information

Args: tilt (int): Index of elevation angle starting from zero.

drange (float): Radius of data.

dtype (str): Type of product (REF, VEL, etc.)

Returns: xarray.Dataset: Data.

get_nrays(*scan: int*) → int

Get number of radials in certain scan

get_raw(*tilt: int*, *drange: Union[int, float]*, *dtype: str*) → Union[numpy.ndarray, tuple]

Get radar raw data

Args: tilt (int): Index of elevation angle starting from zero.

drange (float): Radius of data.

dtype (str): Type of product (REF, VEL, etc.)

Returns: numpy.ndarray or tuple of numpy.ndarray: Raw data

projection(*reso: float*, *h_offset: bool = False*) → tuple

Calculate the geographic coordinates of the requested data range.

class cinrad.io.level2.**PhasedArrayData**(*file*)

Bases: *cinrad.io.base.RadarBase*

get_data(*tilt: int*, *drange: Union[int, float]*, *dtype: str*) → xarray.core.dataset.Dataset

Get radar data with extra information

Args: tilt (int): Index of elevation angle starting from zero.

drange (float): Radius of data.

dtype (str): Type of product (REF, VEL, etc.)

Returns: xarray.Dataset: Data.

get_raw(*tilt: int*, *drange: Union[int, float]*, *dtype: str*) → Union[numpy.ndarray, tuple]

Get radar raw data

Args: tilt (int): Index of elevation angle starting from zero.

drange (float): Radius of data.

dtype (str): Type of product (REF, VEL, etc.)

Returns: numpy.ndarray or tuple of numpy.ndarray: Raw data

class cinrad.io.level2.**StandardData**(*file: Any*)

Bases: *cinrad.io.base.RadarBase*

Class reading data in standard format.

Args: file (str, IO): Path points to the file or a file object.

available_tilt(*product: str*) → List[int]

Get all available tilts for given product

get_data(*tilt: int*, *drange: Union[int, float]*, *dtype: str*) → xarray.core.dataset.Dataset

Get radar data with extra information

Args: tilt (int): Index of elevation angle starting from zero.

drange (float): Radius of data.

dtype (str): Type of product (REF, VEL, etc.)

Returns: xarray.Dataset: Data.

get_raw(*tilt: int*, *drange: Union[int, float]*, *dtype: str*) → Union[numpy.ndarray, tuple]

Get radar raw data

Args: tilt (int): Index of elevation angle starting from zero.

drange (float): Radius of data.

dtype (str): Type of product (REF, VEL, etc.)

Returns: numpy.ndarray or tuple of numpy.ndarray: Raw data

classmethod merge(*files: List[str]*, *output: str*)

Merge single-tilt standard data into a volumetric scan

Args: files (List[str]): List of path of data to be merged

output (str): The file path to store the merged data

cinrad.io.level3 module

class cinrad.io.level3.**PUP**(*file: Any*)

Bases: *cinrad.io.base.RadarBase*

Class handling PUP data (Nexrad Level III data)

Args: file (str, IO): Path points to the file or a file object.

get_data() → xarray.core.dataset.Dataset

Get radar data with extra information.

Returns: xarray.Dataset: Data.

class cinrad.io.level3.**SWAN**(*file: Any*, *product: Optional[str] = None*)

Bases: object

Class reading SWAN grid data.

Args: file (str, IO): Path points to the file or a file object.

get_data(*level: int = 0*) → xarray.core.dataset.Dataset

Get radar data with extra information

Args: level (int): The level of reflectivity data. Only used in *3DREF* data.

Returns: xarray.Dataset: Data.

class cinrad.io.level3.**StandardPUP**(*file*)

Bases: *cinrad.io.base.RadarBase*

cinrad.io.export module

1.1.2 cinrad.visualize package

cinrad.visualize.ppi module

class cinrad.visualize.ppi.**PPI**(*data: xarray.core.dataset.Dataset*, *fig: Optional[Any] = None*, *norm: Optional[Any] = None*, *cmap: Optional[Any] = None*, *nlabel: Optional[int] = None*, *label: Optional[List[str]] = None*, *dpi: Union[int, float] = 350*, *highlight: Optional[Union[str, List[str]]] = None*, *coastline: bool = False*, *extent: Optional[List[Union[int, float]]] = None*, *section: Optional[xarray.core.dataset.Dataset] = None*, *style: str = 'black'*, *add_city_names: bool = False*, *plot_labels: bool = True*, ***kwargs*)

Bases: `object`

Create a figure plotting plan position indicator

By default, norm, cmap, and colorbar labels will be determined by the data type.

Args: data (xarray.Dataset): The data to be plotted.

fig (matplotlib.figure.Figure): The figure to plot on. Optional.

norm (matplotlib.colors.Normalize): Customized norm data. Optional.

cmap (matplotlib.colors.Colormap): Customized colormap. Optional.

nlabel (int): Number of labels on the colorbar. Optional.

dpi (int): DPI of the figure. Optional.

highlight (str, list(str)): Areas to be highlighted. Optional.

coastline (bool): Plot coastline on the figure if set to True. Default False.

extent (list(float)): The extent of figure. Optional.

add_city_names (bool): Label city names on the figure if set to True. Default True.

plot_labels (bool): Text scan information on the side of the plot. Default True.

gridlines(*draw_labels: bool = True*, *linewidth: Union[int, float] = 0*, ***kwargs*)

Draw grid lines on cartopy axes

plot_cross_section(*data: xarray.core.dataset.Dataset*, *ymax: Optional[int] = None*, *linecolor: Optional[str] = None*, *interpolate: bool = True*)

Plot cross section data below the PPI plot.

plot_range_rings(*_range: Union[int, float, list]*, *color: str = 'white'*, *linewidth: Union[int, float] = 0.5*, ***kwargs*)

Plot range rings on PPI plot.

storm_track_info(*filepath: str*)

Add storm tracks from Nexrad Level III (PUP) STI product file

cinrad.visualize.rhi module

1.2 Submodules

1.3 cinrad.calc module

class cinrad.calc.**GridMapper**(*fields: List[xarray.core.dataset.Dataset]*, *max_dist: Union[int, float] = 0.1*)

This class can merge scans from different radars to a single cartesian grid.

Args: fields (list(xarray.Dataset)): Lists of scans to be merged.

max_dist (int, float): The maximum distance in kdtree searching.

Example:

```
>>> gm = GridMapper([r1, r2, r3])
>>> grid = gm(0.1)
```

class cinrad.calc.**VCS**(*r_list: List[xarray.core.dataset.Dataset]*)

Class performing vertical cross-section calculation

Args: r_list (list(xarray.Dataset)): The whole volume scan.

get_section(*start_polar: Optional[Tuple[float, float]] = None*, *end_polar: Optional[Tuple[float, float]] = None*, *start_cart: Optional[Tuple[float, float]] = None*, *end_cart: Optional[Tuple[float, float]] = None*, *spacing: int = 500*) → xarray.core.dataset.Dataset

Get cross-section data from input points

Args: start_polar (tuple): polar coordinates of start point i.e.(distance, azimuth)

end_polar (tuple): polar coordinates of end point i.e.(distance, azimuth)

start_cart (tuple): geographic coordinates of start point i.e.(longitude, latitude)

end_cart (tuple): geographic coordinates of end point i.e.(longitude, latitude)

Returns: xarray.Dataset: Cross-section data

cinrad.calc.**hydro_class**(*z: xarray.core.dataset.Dataset*, *zdr: xarray.core.dataset.Dataset*, *rho: xarray.core.dataset.Dataset*, *kdp: xarray.core.dataset.Dataset*, *band: str = 'S'*) → xarray.core.dataset.Dataset

Hydrometeor classification

Args: z (xarray.Dataset): Reflectivity data.

zdr (xarray.Dataset): Differential reflectivity data.

rho (xarray.Dataset): Cross-correlation coefficient data.

kdp (xarray.Dataset): Specific differential phase data.

band (str): Band of the radar, default to S.

Returns: xarray.Dataset: Classification result.

cinrad.calc.**quick_cr**(*r_list: List[xarray.core.dataset.Dataset]*, *resolution: tuple = (1000, 1000)*) → xarray.core.dataset.Dataset

Calculate composite reflectivity

Args: r_list (list(xarray.Dataset)): Reflectivity data.

Returns: xarray.Dataset: composite reflectivity

cinrad.calc.**quick_et**(*r_list: List[xarray.core.dataset.Dataset]*) → xarray.core.dataset.Dataset

Calculate echo tops

Args: r_list (list(xarray.Dataset)): Reflectivity data.

Returns: xarray.Dataset: echo tops

cinrad.calc.**quick_vil**(*r_list: List[xarray.core.dataset.Dataset]*) → xarray.core.dataset.Dataset

Calculate vertically integrated liquid.

This algorithm process data in polar coordinates, which avoids the loss of data. By default, this function calls low-level function *vert_integrated_liquid* in C-extension. If the C-extension is not available, the python version will be used instead but with much slower speed.

Args: r_list (list(xarray.Dataset)): Reflectivity data.

Returns: xarray.Dataset: vertically integrated liquid

cinrad.calc.**quick_vild**(*r_list: List[xarray.core.dataset.Dataset]*) → xarray.core.dataset.Dataset

Calculate vertically integrated liquid density.

By default, this function calls low-level function *vert_integrated_liquid* in C-extension. If the C-extension is not available, the python version will be used instead but with much slower speed.

Args: r_list (list(xarray.Dataset)): Reflectivity data.

Returns: xarray.Dataset: Vertically integrated liquid

1.4 cinrad.common module

1.5 cinrad.grid module

cinrad.grid.**grid_2d**(*data: numpy.ndarray*, *x: numpy.ndarray*, *y: numpy.ndarray*, *x_out: Optional[numpy.ndarray] = None*, *y_out: Optional[numpy.ndarray] = None*, *resolution: tuple = (1000, 1000)*) → tuple

Interpolate data in polar coordinates into geographic coordinates

Args: data (numpy.ndarray): Original radial data.

x (numpy.ndarray): Original longitude data arranged in radials.

y (numpy.ndarray): Original latitude data arranged in radials.

resolution (tuple): The size of output.

Returns: numpy.ndarray: Interpolated data in grid.

numpy.ndarray: Interpolated longitude in grid.

numpy.ndarray: Interpolated latitude in grid.

cinrad.grid.**resample**(*data: numpy.ndarray*, *distance: numpy.ndarray*, *azimuth: numpy.ndarray*, *d_reso: Union[int, float]*, *a_reso: int*) → tuple

Resample radar radial data which have different number of radials in one scan into that of 360 radials

Args: data (numpy.ndarray): Radar radial data.

distance (numpy.ndarray): Original distance.

azimuth (numpy.ndarray): Original azimuth.

Returns: numpy.ndarray: Resampled radial data.

numpy.ndarray: Resampled distance.

numpy.ndarray: Resampled azimuth.

1.6 cinrad.utils module

cinrad.utils.**potential_maximum_gust**(*et: numpy.ndarray*, *vil: numpy.ndarray*) → numpy.ndarray

Estimate the potential maximum gust with a descending downdraft by Stewart's formula

- modindex
- genindex

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

R

S

V