Journal of
open research software

Figueras i Ventura, J, et al. 2020 Pyrad: A Real-Time Weather Radar Data Processing Framework Based on Py-ART. *Journal of Open Research Software,* 8: 28. DOI: https://doi.org/10.5334/jors.330

SOFTWARE METAPAPER

Pyrad: A Real-Time Weather Radar Data Processing Framework Based on Py-ART

Jordi Figueras i Ventura, Martin Lainer, Zaira Schauwecker, Jacopo Grazioli and Urs Germann

MeteoSwiss, Locarno, CH

Corresponding author: Jordi Figueras i Ventura (jordi.figuerasiventura@meteoswiss.ch)

Pyrad is a real-time data processing framework developed by MeteoSwiss. The framework is aimed at reading, processing and visualizing polar data from individual weather radars as well as composite Cartesian products both off-line and in real time. The processing flow is controlled by three simple configuration files. This allows the construction of reproducible data processing chains. In the off-line mode, data from multiple radars can be ingested. It is written in the Python programming language. Most of the signal processing and part of the data visualization is performed by a MeteoSwiss-developed version of the Py-ART radar toolkit, which contains enhanced features. Thanks to the broad types of input files accepted and its flexibility it can be easily adapted and used by any member of the weather radar community. The source code is available on GitHub. Compiled versions are also available on PyPI and conda-forge. They are distributed under a BSD license.

Keywords: Python; radar; weather radar; quality control; real-time; visualization; data processing

(1) Overview

Introduction

Due to their wide coverage, 24/7 automatic operation and the capability to provide useful information in almost any condition, weather radars have an ever-expanding range of applications (see for example [1] and [2] for an overview). Direct applications include severe weather detection (e.g. [3]), cloud and precipitation type identification (e.g. [4]), quantitative precipitation estimation (e.g. [5]) and wind speed and direction retrievals (e.g. [6]) as well as in microphysical studies of clouds and precipitation. Radar data is also widely used in precipitation nowcasting (e.g. [7]) and can substantially improve weather forecasts if properly assimilated ([8]). Furthermore, radar data has multiple applications in the field of hydrology and is also essential in issuing automatic weather warnings (e.g. [9]).

The extraction of meaningful information from radar returns is a complex task and quite often depends on the desired application. Moreover, the data processing can be viewed as a chain of individual algorithms. Oftentimes how the data is processed at the earliest stages of the processing chain has a significant impact on the end result. Hence, even if a well-documented algorithm is implemented, the outcome may vary significantly from the reference due to divergences in the previous processing steps. Further enhancing the problem of reproducibility, many scientists are at least partially dependent on commercial software which, in the first place, is usually a black box and, secondly, makes the reproducibility of the results by other scientists complicated if they do not have the same software.

In the last years, a number of initiatives towards a more open weather radar science have been produced [10]. A pioneer among them was BALTRAD [11], which was designed primarily for the efficient exchange of data but that contains a processing toolbox with a wide range of functionality. BALTRAD contains many submodules which are written in a mixture of C/C++, Python and Java. Roughly at the same time of BALTRAD, wradlib was released [12]. wradlib is essentially a Python-based library containing a growing collection of functions to read, correct and transform radar data. Since the focus is primarily in data processing, it is easier to integrate in an existing processing chain than BALTRAD. Shortly after wradlib, Py-ART was released [13]. Py-ART was developed in the context of the Atmospheric Radiation Measurement (ARM) programme [14]. Py-ART is also a python-based library as wradlib. The main difference between the two is that Py-ART is based on a data model that closely follows the standards of CfRadial whereas wradlib is mostly data agnostic. wradlib and Py-ART are very complementary and indeed some features of Py-ART already use wradlib. Py-ART has had some success with extensions for interactive visualization of data such as ARTView [15] and specific projects built around the Py-ART radar object such as PyTDA [16] which estimates turbulence from Doppler

radar data. Another important contribution to the open source ecosystem is LROSE, which builds on the legacy left by TITAN [17]. Similar to Py-ART and wradlib, the idea behind LROSE is to provide building blocks, written in C/C++, taking care of the routine tasks while scientists can focus on the development of algorithms. Finally, another noteworthy software package in the field of radar meteorology is pySTEPS [18]. This package is a library with a set of methods used for probabilistic precipitation nowcasting.

The aim of Pyrad is to provide users with a tool to easily construct real-time and offline weather radar data processing chains in a transparent and reproducible manner. The program controls the data flow and the datasets and products to be created. We use here dataset as a broad concept. It can be anything from a secondary radar field (e.g. a rainfall rate field) to a different data object such a time series of data in a point in space. The behavior of the processing chain is controlled by three simple user-defined configuration files so there is no need for the user to actually program anything. The exact data processing can be reproduced by simply sharing the configuration files. Pyrad is built on top of Py-ART, and in fact it can be seen as an extension of it. As such, it can make full use of the Py-ART capabilities in a structured manner. In the process of developing Pyrad, we built an enhanced version of Py-ART (hereby called MCH Py-ART) with many additional capabilities. Indeed, to our knowledge, Pyrad using the MeteoSwiss Py-ART is the first open source project that offers basic IQ and spectral processing capabilities. Pyrad is compatible with both the MCH Py-ART and ARM Py-ART. If used with the ARM Py-ART, simply some of the Pyrad capabilities will not be available. However we regularly contribute to the ARM Py-ART features of MCH Py-ART which we consider mature and of interest for the broader weather radar community. Hence MCH Py-ART can be regarded as a bleeding edge version of the ARM Py-ART.

This papers aims to present the main features and the current state of the Pyrad software. First a history of the genesis of the package will be given. It will follow a discussion on the main features and how they were implemented. The subsequent section will provide a description of the way quality is ensured. It follows information on the repository of the source code of the software (from the repository compiled packages stored in PyPI and conda-forge can be accessed) and concludes with a discussion on how the package can be used for radar applications.

We struggle to make Pyrad as simply to use and user-friendly as possible. Still, it is a software aimed at people that have at least a basic understanding of weather radar. There are numerous books that can help providing a good understanding of the topic (e.g. [2, 19, 20 and 21]).

History of the package

Since 2012, MeteoSwiss operates a mobile X-band Doppler polarimetric weather radar. The system is used primarily for measurement campaigns tailored for specific customers. When there is no specific request from such customers, it is also used as a research and development platform in order to test new algorithms and scanning modes. Quite often, the customer has specific needs that are non-conventional in weather radar data and therefore not easily tackled using existing commercial software. Consequently we have developed a considerable amount of software to control the data flow, process the data and display it in a convenient manner. A first processing and interactive analysis framework was created in the nineties in the Interactive Data Language (IDL), which at the time was one of the officially approved programming languages at MeteoSwiss, to process data from the 3rd generation operational weather radar network. Since there was a considerable in-house know-how, the real-time processing framework for the X-band radar was also developed in IDL. However, over the years it became clear that IDL had some major limitations: 1) The fact that requires a license to execute the code limits the re-use and sharing of the code by external partners, 2) IDL is a relatively old language, with many issues that makes it a somehow rigid language and difficult to write code with by non-experts, such as for example, the single namespace. 3) The number of IDL users in meteorological applications is dwindling and consequently the number of readily available libraries and online support is limited compared with other programming languages.

Due to the perceived limitations of the IDL language, in 2016 it was decided to abandon new developments of the IDL processing framework and explore other options. The driving criteria that the new option had to fulfill were identified as the following: 1) The programming language should not be subject to license fees, in order to facilitate its use and sharing by external partners. 2) It should be flexible and easily programmable by non-experts, yet at the same time fast enough for real-time execution. 3) It should have a sizable user-base to ensure extensive support, both in the form of readily usable software libraries and online support.

Very quickly it became clear that Python was the best choice available. Factors on the choice included its wide-spread use by the scientific community, the multitude of open source scientific packages available (e.g. the SciPy ecosystem [22]) and the vibrant online community. Additional factors were the easiness with which software documentation can be produced by using docstrings (by using sphinx [23] for example) and the existence of software quality control checkers such as Pylint [24]. Two other factors contributed to the choice: the fact that at that time Python was officially approved as a usable programming language by MeteoSwiss and, more importantly, the level of maturity that at that stage had reached open source radar data processing packages such as wradlib and Py-ART.

Over the last years, MeteoSwiss and, more specifically the Radar, Satellite and Nowcasting division of MeteoSwiss has established numerous formal and informal partnerships with Swiss research institutions, e.g. EPFL, ETH Zurich, the University of Bern, etc. Within the weather radar community, and the scientific community in general, there is an increased perception that collaboration has

to extend beyond the knowledge and data exchange to include also software. Pyrad was hence created to fulfill not only the specific MeteoSwiss needs towards our customers but also as an open platform to exchange software with our partners. The idea behind is that the framework takes care of the data flow and visualization and therefore the developer needs only to implement its specific algorithm.

It was also evident that, with two mature python-based weather radar processing toolkits available at the time (wradlib and Py-ART), it would have been absurd to start a new open source project from scratch. Py-ART has a data model in the form of the radar object whereas wradlib is data agnostic to a certain extent. It was decided that having a data model would emulate better the functioning of the IDL framework and hence Pyrad was built as a "wrapper" of Py-ART, with wradlib providing support for some specific functions. This setup had also the advantage that Py-ART was starting to be used by one of the main partners of MeteoSwiss, i.e. EPFL, and that would facilitate the collaboration between the two groups.

The first phase of the development was to quickly implement a data flow control, similar to that implemented in the previous IDL version but with increased flexibility. A second phase consisted in implementing the functionality (file reading, data processing and visualization) in order to perform the core duties of the team in charge of the mobile X-band radar. It was immediately apparent that, regarding data processing and visualization, we had developed many more functionalities in IDL than what was readily available in Py-ART at the time. Hence we setup our own version of Py-ART. We were constrained to favor fast implementation rather than robustness so we developed a lot of code in a short time, testing mostly its overall functionality rather than individual functions, but we always had in mind to pull relevant functions to the original ARM Py-ART. The fact that at the core of Pyrad lies Py-ART and that we already had a processing framework in IDL greatly sped up the development phase. In few months we had a working prototype and in less than one year we were able to substitute the IDL framework entirely for 90% of our operational needs regarding the mobile X-band radar.

In a third phase we expanded the functionality in order to allow the data processing and visualization of our C-band radars and the mobile X-band radar owned by EPFL, hence the software became truly a platform for the processing of all Swiss weather radars. Additionally we implemented reading and visualization routines for all the Cartesian products generated from the C-band operational radar to facilitate its use by MeteoSwiss partners. More recently we have added the capability to ingest ODIM_H5 [25] and Nexrad level II [26] file formats. In addition to the native CfRadial V1 [27] file format used by Py-ART, it makes it a truly globally usable software package. The most recent software developments have been the capability to process spectral and IQ data. Hence the software can be used to process radar data from IQ to Cartesian composite products. Nowadays, Pyrad has reached a mature stage, it has multiple capabilities and it is used both operationally and for research purposes. At this point we are working in three directions: 1) Facilitate its use by external partners. A main step in this direction was creating PyPI and conda-forge packages which are easily downloadable by the user. We are also working constantly to improve the documentation and provide configuration file examples that can be easily interpreted. 2) Consolidate the software by increasing the coverage of the unit-tests and 3) Contribute back to Py-ART in a more systematic way so that the software developed by MeteoSwiss can serve a broader audience.

Implementation and architecture

Pyrad is a Python package that provides a variety of routines for reading, processing, analyzing and visualizing data from weather radars and, up to a certain extent, from other active remote sensing instruments such as lidars. It can be used as a toolbox of individual functions but it is most powerful when used as a processing framework. In such case, the full functionality of Pyrad can be used by writing 3 simple configuration files. Pyrad is built around the data models defined in the "core" module of MCH Py-ART. Currently there are 3 classes in this module: Radar, Grid and RadarSpectra. Radar and Grid are also defined in the ARM Py-ART while RadarSpectra is, at the moment, only implemented in the MeteoSwiss version. Those classes are used to store the data and the metadata necessary for the processing. The Radar object stores radar polar moments in two-dimensional matrices (ray, range), the Grid object stores gridded data whereas the RadarSpectra is a class based on Radar that stores IQ and complex spectra data in a three-dimensional matrix (ray, range, npulses_max). npulses_max is the maximum number of pulses per ray collected during a scan.

At its most basic, the processing (see **Figure 1**) consists in sequentially ingesting one volume of radar data in the adequate object, performing a data manipulation on that object to create a new dataset (usually using a function implemented in Py-ART) and generating products out of the new dataset. The production of a dataset or product may require the use of auxiliary data. For example, the solar flux from the Dominion Radio Astrophysical Observatory (DRAO) research facility observatory in Canada is used to estimate the receiver bias in the sun monitoring function. Another example is a function that compares rainfall rate estimates obtained from radar data to data from SwissMetNet rain gauges or from disdrometers. The newly-generated dataset may be added as a set of additional data fields (if it is an object with similar characteristics as the input, e.g. a rainfall rate field obtained from a reflectivity field) or make it to entirely replace the initial object (e.g. polarimetric moments computed from IQ data), allowing subsequent levels of processing on the same data. Once all levels of processing have been performed, a new radar object is ingested.

Each dataset to be generated has a dictionary structure called dataset configuration (dscfg) attached that stores the parametrization used to produce it. One of the keywords in dscfg is global_data. This keyword is used to store any information that needs to be persistent in time, i.e. usable beyond the current radar scan. For example,

there is a Pyrad function that computes temporal averages. In this function the global_data variable is used to store a variable with the starting date and time of the averaging and a Radar object containing a field with the cumulative sum of values at each range gate and another field with the number of time steps used. At each time step, the difference between the starting time of the averaging and the current scan time is computed and if the desired averaging time is reached the average can simply be computed by dividing the field with the cumulative sum by the number of points. In the offline mode, once all radar volumes within the processing period specified have been consumed, post-processing of the data can be performed, e.g. output of global statistics, time series plots, etc. Here by radar volume we mean all the data that is obtained in the course of an individual radar scan.

The processing is controlled using 3 simple configuration files. The main one specifies the base paths where the input data is stored, the output should be located and the path to the other two configuration files. All configuration files can have any arbitrary name. The second configuration file, so-called location configuration file, contains general specification such as the name of the radar(s), its location (if not in the file metadata), general specifications about the graphic output format (dimensions, dpi, etc.) and radar metadata that is not readable from the radar file or that for some reason needs to be overridden. The third configuration file, the so called product configuration file, contains an ordered list of datasets to generate. For each dataset, a series of keywords specify which are the required dataset inputs and whatever parametrization is necessary to obtain the dataset. If the keyword "MAKE_GLOBAL" is set, the dataset will be made available for the next processing level. Within the dataset, the keyword "products", if specified, contains which products should be generated out of the dataset, e.g. (pseudo) Plan Position Indicators (PPIs) i.e. displays of data in the horizontal plane, (pseudo) Range Height Indicators (RHIs) i.e. displays of data in the vertical plane, etc. We opted for separating the configuration in 3 files so that they can be easily re-used. For example, if we process data from the same data and in the same way in two different computers with different storing configuration we only need to have one main configuration file for each machine that point to the same location and product files.

Modules

Internally, Pyrad is organized in 6 sub-packages according to the type of functionality that they provide: "flow", "io", "proc", "prod", "graph" and "util" (see **Figure 2**). The "flow" sub-package contains the functions that control the data flow. This package contains the "main" and "main_rt" functions which perform the off-line and real-time processing respectively. The "io" sub-package provides the ability to read in all the data used by Pyrad: individual radar data (IQ, spectra and polarimetric moments), radar

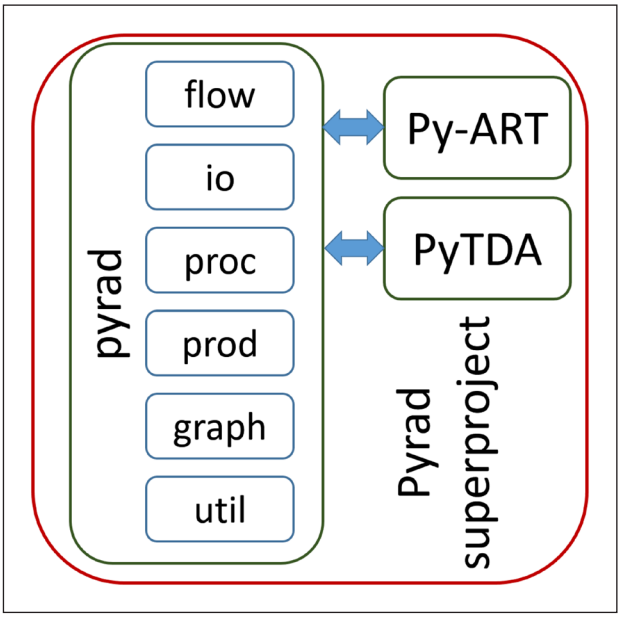

Figure 2: Structure of the Pyrad superproject software.

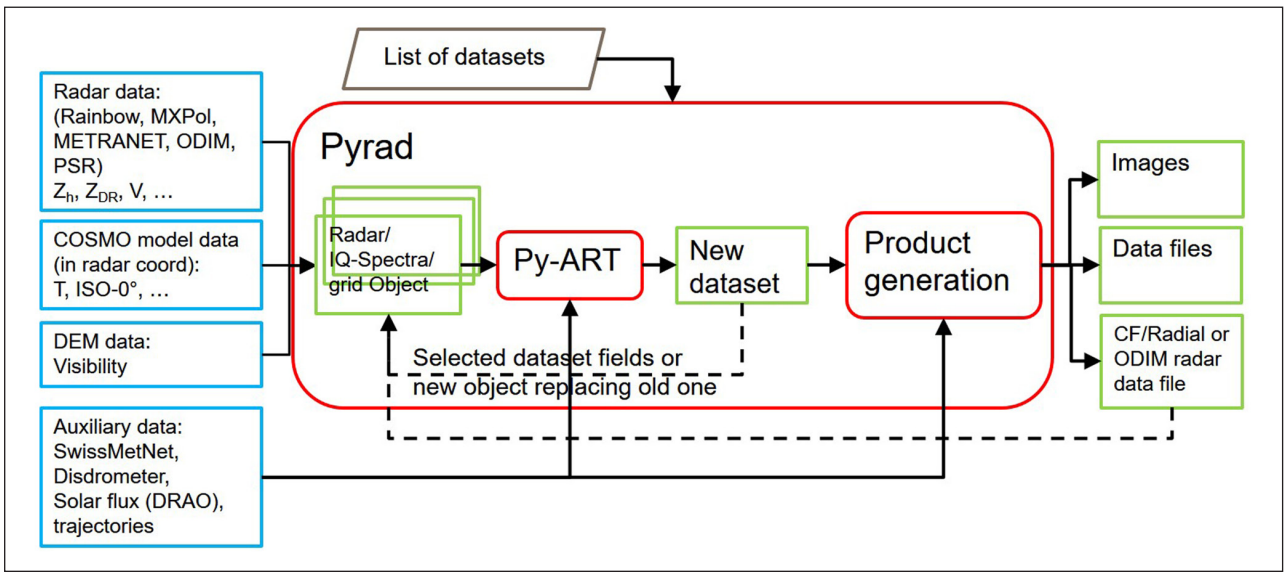

Figure 1: Pyrad flow diagram.

composite data, e.g. QPE products but also auxiliary data such as data from NWP models (temperature, wind, etc.), radar visibility, disdrometer data, rain gauges, plane trajectories and many more. The "io" module calls functions contained in the "io" and "aux_io" modules in Py-ART to read radar and gridded data. This module also manages the writing of outputs. Radar data can be stored in the ODIM_H5 file format or in CfRadial. Most of the other data outputs are written as .csv files. The "proc" sub-package contains all functions to process the datasets. Most of these functions are mere wrappers to Py-ART functions that perform some sort of data processing. The "prod" sub-package contains all functions to generate products out of the datasets. Datasets are organized in categories. For example, there are categories for volumetric data, inter-comparison data, etc. Each data processing function has assigned a dataset category which defines which type of products can be generated. The "graph" module implements visualization functions. Some of these functions internally call visualization functions implemented in the graph sub-package in Py-ART while others generate specialized visualization and are implemented directly in the module. Examples of the latter are time-series plots or sun hits plots. Finally, the "util" module contains a collection of auxiliary functions to manipulate the data, such as functions to get data along a particular range, to compute histograms, etc.

As an experimental feature, off-line Pyrad allows for parallel processing using dask [28]. It can be parallelized the generation of products out of a dataset or the generation of datasets in the same processing level or eventually both.

Scripts

Pyrad provides users with 3 main scripts: main_process_data_rt.py, main_process_data_period.py and main_process_data.py. main_process_data_rt.py is used for real-time processing. It internally calls the function main_rt. As positional argument, it requires a coma-separated list of main configuration files which covers all the data processing that should be performed. Thus, multiple scan modes or multiple radars can be processed using a single instance. Starting and ending times of the processing can optionally be specified. The real-time processing can be performed either continuously keeping main_rt in "listening" mode or by periodically calling main_rt, which would then process all new data arrived in the interval between calls. main_process_data_period.py is used to process data obtained over several days off-line. As positional argument it expects the name of the main config file and the starting and ending dates to process. Keyword arguments that may be specified include the start time and end time of the processing each day. This script will process all the radar volumes, obtained within the specified time interval, of one day, make a post-processing at the end of the interval and repeat sequentially this process until the end date. The obvious application of such script is the data quality monitoring, where statistics are performed on a daily basis. Finally main_process_data.py is used for off-line processing. As a positional argument requires the name of a main configuration file. The most obvious mode of operation is to specify a date and time to start (keyword "startime") and end (keyword "endtime") the processing. The script will then look for all the data within the period and process it sequentially. The behavior of the script changes if instead of the time keywords a trajectory file is specified with the keyword "trajfile". There are three different trajectory types (specified with keyword "trajtype"): plane, lightning or proc_periods. The "plane" type is used to get time series of radar data which is co-located to the position of a moving object, e.g. a plane. It is very useful to match data obtained in-situ by a plane with radar data. The "lightning" type is similar to the plane type but it was designed to co-locate lightning data obtained from Lightning Mapping Arrays (LMAs) to radar data. It has the particularity that it can be used to obtain data matching a specific flash (through the keyword "flashnr") or data matching all flashes in a lightning data file (when "flashnr" is 0). **Figure 3** shows an example of the type of product that can be obtained using this script. Finally, the proc_periods type expects a file with a list of start and end processing dates and times. It is meant to be able to process (and eventually post-process) disjoined periods of data with the utmost flexibility. For example, if one is interested in getting statistics of radar data over periods where a certain condition was met (processing only rainy periods for example) this function could be used.

Capabilities

Since the functionalities of Pyrad are continuously expanding there is no point in explaining them in detail here. At the moment there are 16 different dataset groups, more than 80 different processing functions and more than 40 different product types that can be generated. Most processing capability is used for polarimetric moments processing (see **Figure 4** for a list of functionalities available for polar moments). Products that can be generated from Radar objects can be categorized as visualization products or data file products. Visualization products include all the standard weather radar visualization products (e.g. (pseudo)PPI, (pseudo) RHI, B-scope, CAPPI, cross-sections, etc.) but also non-standard products such as data histograms, time series at points of interest, time-range plots, scatter plots, etc. Radar objects can be saved in CfRadial or ODIM_H5 convention while other data is typically saved as CSV files.

IQ data capability includes the computation of polarimetric and Doppler moments in the time-domain (using lag-N estimators) and the transformation into spectral data using windowed FFTs. Spectral data processing currently includes 0-Doppler filtering, spectral co-polar correlation coefficient filtering, spectral noise filtering (spectral clipping), computation of spectral polarimetric moments, noise estimation from the spectra, computation of polarimetric and Doppler moments and obtaining data in points and regions of interest. Both IQ and spectral data can be visualized in Range/Angle/Time-Doppler/Pulses plots (see **Figure 5** for an example of visualization) and saved in a netcdf file.

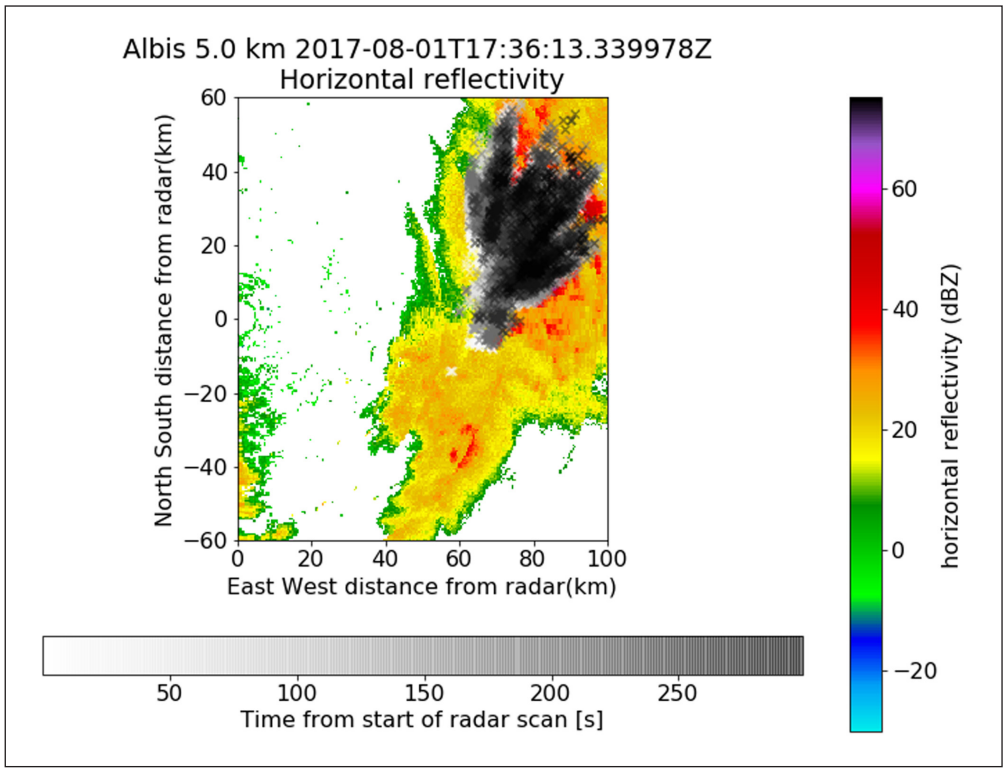

Figure 3: Example of lightning trajectory product. It shows a Constant Altitude Plan Position Indicator (CAPPI) at 5000 m MSL altitude with crosses at the location of flash sources detected by an LMA superposed. The crosses are color-coded as a function of time since the starting of the radar volume scan. Data was obtained by the MeteoSwiss C-band network radar Albis on 2017-08-01 at 17:45 UTC.

Echo classification & filtering	ψ_{DP} processing and attenuation correction	Monitoring, calibration & noise corr	Retrievals	Special functions
• Clutter ID and filtering • SNR filter • Visibility filter • Outlier filter • Hydrometeor classification (semi-supervised cluster)	• Φ_{DP0} correction • Φ_{DP} smoothing (1, 2 windows) • Least square K_{DP} retrieval (1, 2 wind) • Φ_{DP}, K_{DP} retrieval Maesaka • Linear Programming Φ_{DP}, K_{DP} retrieval • Φ_{DP}, K_{DP} retrieval Vulpiani • ZPhi & PhiLinear att corr	• Bias correction • ρ_{HV} noise correction • ρ_{HV} in rain estimation • Z_{dr} in moderate rain estimation • Z_{dr} in snow estimation • Z_{dr} in birdbath scan • Self-consistency Z_h bias estimation • Time averaging • Ground clutter monitoring • Radar inter-comparison • Sun signal monitoring	• Signal power • SNR • Radial noise power • Clutter Correction ratio • L parameter • CDR • RCS • Melting layer detection • Wind velocity • Wind shear • Various rainrate algorithms • Rainfall accumulation • Velocity dealiasing • Bird density • Velocity dealias • VAD	• Volume cutting • Gridding • Trajectory • Point of interest • Data gridding • Cumulative distribution functions • Quasi Vertical Profiles • Temporal statistics • Fixed range/fixed range span data

Figure 4: Processing capabilities for polarimetric moments.

The capabilities implemented so far for the gridded data include obtaining data at points and regions of interest and performing temporal statistics. Gridded data can be visualized as images and contours over a map, cross-sections (see an example in **Figure 6**) or histograms and saved in a netcdf file.

Quality control

The core of Pyrad is based on Py-ART. Py-ART includes a large suit of unit tests for individual functions. At the moment Pyrad has two levels of testing. A first level automatically makes sure that new code does not break the software. With each commit, a continuous integration platform run by Travis CI build, builds the package from source. If the package cannot be imported correctly we get notified and can check where the issue is. The second level of testing is focused on functionality. Both Pyrad and MCH Py-ART have a development branch where new functionality is first implemented and thoroughly tested using MeteoSwiss radar data. Only once the new functions have proved to work as expected they are merged into the master branches. This process has not yet

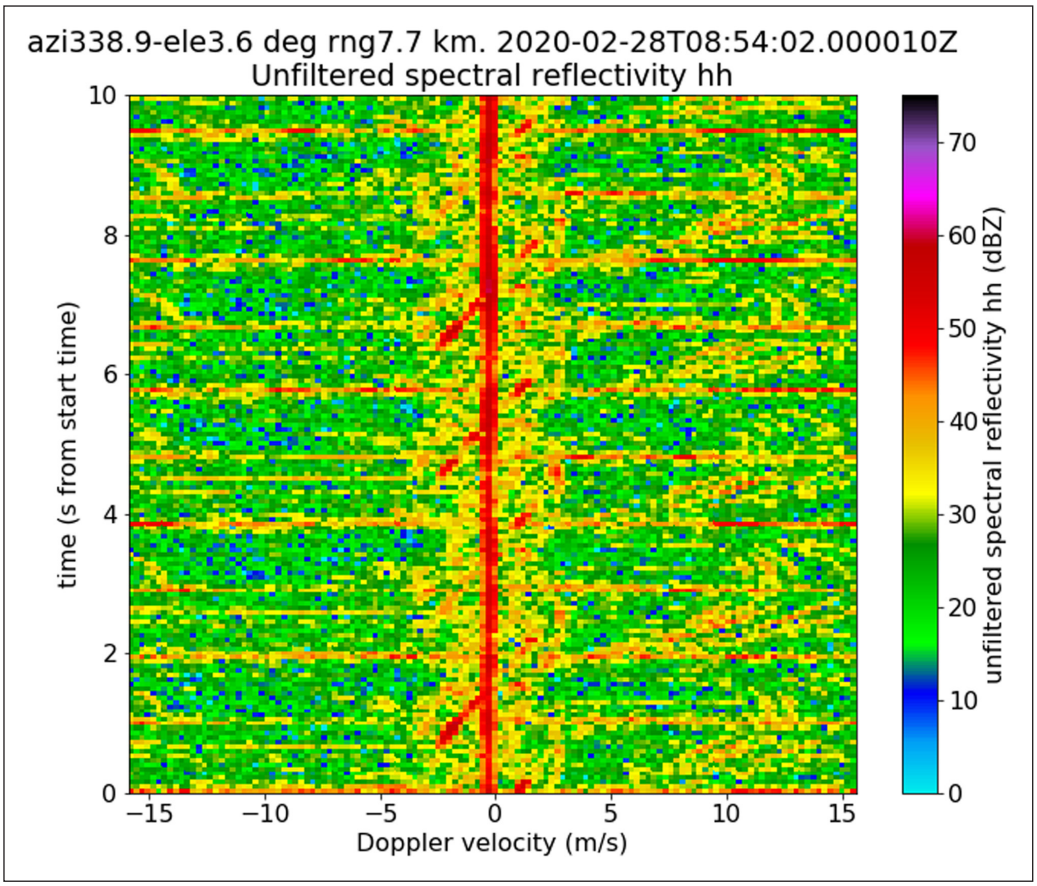

Figure 5: Example of time-Doppler product featuring the spectral reflectivity. Data was obtained by a mobile X-band radar operated by MeteoSwiss on 2020-02-28. The radar was operating in staring mode and pointing towards a windmill.

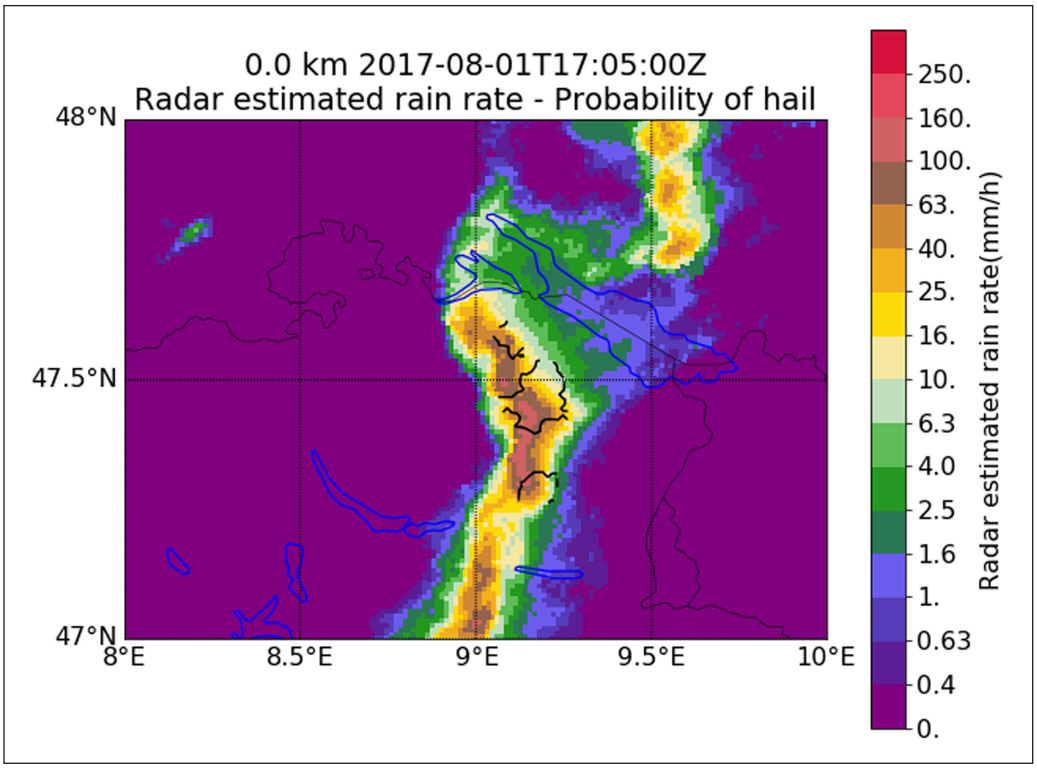

Figure 6: Example of product obtained from a Cartesian grid. It features the MeteoSwiss radar rainfall rate composite with the 80% probability of hail contours overplotted. Lakes and international borders are also shown. Data was obtained on 2017-08-01 at 17:05UTC.

been automatized. We plan to add unit tests also for Pyrad testing in the near future.

A similar approach is used for the software package releases in PyPI and conda-forge. Users can easily understand whether the software was properly installed simply by importing Pyrad in a python command shell. Furthermore, they can use some of the MeteoSwiss sample data and configuration files to check the main Pyrad functionalities. The data samples and example of configuration files are contained in the following github repository: https://github.com/meteoswiss-mdr/pyrad-examples. The repository also contains the expected output for each configuration file.

(2) Availability

Operating system
Linux and OS X

Programming language
Python 3. (3.6 and 3.7 tested, 3.8 planned)

Additional system requirements
None.

Dependencies
Py-ART (preferably MCH Py-ART) and all its dependencies. MCH Py-ART has the following additional optional dependencies respect to ARM Py-ART: imageio and pysolar. Pyrad has as optional dependencies pandas, shapely, dask, bokeh and h5py. If PyTDA is used it requires also scikit-learn.

List of contributors
- Floortje Elisabeth Maria van den Heuvel and Daniel Wolfensberger, EPFL, Lausanne, Switzerland
- Jordi Figueras i Ventura, Martin Lainer, Jacopo Grazioli and Daniele Nerini, MeteoSwiss, Locarno, Switzerland
- Andreas Leuenberger, MeteoSwiss, Locarno, Switzerland now at Palindrome Remote Sensing, Landquart, Switzerland

Software location

Archive
Name: GitHub
Persistent identifier: https://github.com/meteoswiss-mdr/pyrad
Licence: BSD
Publisher: MeteoSwiss
Version published: Latest tag 0.4.4
Date published: First commit 25/08/16

Code repository
Name: conda-forge
Persistent identifier: https://anaconda.org/conda-forge/pyrad_mch
Licence: BSD
Date published: 02/03/20

Language
English

(3) Reuse potential

Pyrad is currently used on an operational basis at MeteoSwiss in various domains: 1) Real-time processing of the mobile X-band polarimetric moments, 2) Post-processing of the mobile X-band data, 3) Polarimetric data quality monitoring of both the C-band operational network and the mobile X-band radar. It is also used extensively as development platform for new algorithms by MeteoSwiss and some of its partners. Moreover, it is widely used to read and visualize MeteoSwiss radar data.

In principle, anybody wishing to process polar weather radar data in the Rainbow, ODIM_H5, NEXRAD or CfRadial file formats can use Pyrad as it is, without the need to program anything, by simply preparing the 3 configuration files needed. Most of the standard weather radar data processing functionalities (i.e. clutter detection and suppression, attenuation correction, hydrometeor classification, rainfall rate estimation, etc.) are readily available in Pyrad. Users can also opt to integrate the software package to their existing processing chain and use individual components for their needs, such as specialized plotting routines or IQ data processing. In fact, data of any scanning or fixed pointing device can potentially be processed by Pyrad and it has already successfully been used to visualize data from scanning lidars and cloud radars.

Due to its modular nature, Pyrad can also easily be extended. For example, as long as it can comply with the data model, a user can easily add its own data reader and, by adding just a few keywords, be able to use the whole functionality of the package. Furthermore, new algorithms can be easily implemented without the burden of having to think about the data flow, inputs and outputs. Since Pyrad is compatible with Py-ART and wradlib, developments on those libraries are continuously integrated in the Pyrad environment. To our knowledge, Pyrad may be the only open source weather radar software package providing functions for IQ and spectral processing. It can, therefore, be used to double-check the output of commercial weather radar signal processors, which tipically are a black box for users. Again, it can be used as a development platform for more sophisticated algorithms since most of the typical visualization functions are already implemented in it.

Since it is used for some of its operational needs, Pyrad will be continuously developed and used at MeteoSwiss in the foreseeable future. Hence, the stability and continuous support of the package is not dependent on transient money or a time-limited project, which should be a guarantee for users. There are various ways in which external users get support. The first is through the online reference documentation, automatically generated from the Python docstrings. Secondly, there is a detailed user manual, in the Pyrad repository, with instructions on how to install, develop, make pull requests and request support. Bugs and code expansion requests can also be signaled via the Github issues page. Pull requests from external partners will be considered. We are very willing to expand the package with functions that may serve to

a broader community. Finally, the authors can be directly contacted by email.

Pyrad can be installed from source from its github repository or as a PyPI package (https://pypi.org/project/pyrad-mch/) or conda package. For regular users we recommend to install it as a conda package. Detailed installation instructions are provided in the Pyrad user manual which can be downloaded from the Pyrad repository (https://github.com/meteoswiss-mdr/pyrad/blob/master/doc/pyrad_user_manual.pdf). Examples of configuration files with sample data and expected output can be found in https://github.com/meteoswiss-mdr/pyrad-examples.

Acknowledgements

The authors wish to thank the users of Pyrad for their encouragement and suggestions. Many thanks also to the creators and mantainers of Py-ART, on which the code is founded, and to all the contributors to the various scientific Python packages. They also wish to thank MeteoSwiss and armasuisse for supporting the development of the software.

Competing Interests

The authors have no competing interests to declare.

References

1. **Bringi, V N** and **Chandrasekar, V** 2001 *Polarimetric Doppler Weather Radar: Principles and Applications*. Cambridge University Press. DOI: https://doi.org/10.1017/CBO9780511541094
2. **Ryzhkov, A V** and **Zrnic, D S** 2019 Radar Polarimetry for Weather Observations. *Springer Atmospheric Sciences*. Springer International Publishing. ISBN 9783030050924. URL https://books.google.ch/books?id=d0uYvQEACAAJ. DOI: https://doi.org/10.1007/978-3-030-05093-1
3. **Tanamachi, R L** and **Heinselman, P L** 2016 Rapid-scan, polarimetric observations of Central Oklahoma severe storms on 31 may 2013. *Weather and Forecasting*, 31(1): 19–42. DOI: https://doi.org/10.1175/WAF-D-15-0111.1
4. **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. *Atmospheric Measurement Techniques*, 9(9): 4425–4445. URL https://www.atmos-meas-tech.net/9/4425/2016/. DOI: https://doi.org/10.5194/amt-9-4425-2016
5. **Figueras i Ventura, J** and **Tabary, P** 2013 The new french operational polarimetric radar rainfall rate product. *Journal of Applied Meteorology and Climatology*, 52(8): 1817–1835. DOI: https://doi.org/10.1175/JAMC-D-12-0179.1
6. **Bousquet, O** and **Chong, M** 1998 A Multiple-Doppler Synthesis and Continuity Adjustment Technique (MUSCAT) to recover wind components from Doppler radar measurements. *J. Atmos. Oceanic Technol.* 15(2): 343–359. DOI: https://doi.org/10.1175/1520-0426(1998)015<0343:AMDSAC>2.0.CO;2
7. **Sideris, I V, Foresti, L, Nerini, D** and **Germann, U** 2020 Nowprecip: localized precipitation nowcasting in the complex terrain of switzerland. *Quarterly Journal of the Royal Meteorological Society*, 146(729): 1768–1800. URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3766. DOI: https://doi.org/10.1002/qj.3766
8. **Sun, J** and **Wilson, J W** 2003 The assimilation of radar data for weather prediction. *Meteorological Monographs*, 52: 175–198. DOI: https://doi.org/10.1175/0065-9401(2003)030<0175:TAORDF>2.0.CO;2
9. **Hering, A M, Nisi, L, della Bruna, G, Gaia, M, Nerini, D, Ambrosetti, P, Hamann, U, Trefalt, S** and **Germann, U** 2015 Fully automated thunderstorm warnings and operational nowcasting at meteoswiss. In: *8th European Conference on Severe Storms ECSS 2015*, Vienna, Austria.
10. **Heistermann, M, Collis, S, Dixon, M J, Giangrande, S, Helmus, J J, Kelley, B, Koistinen, J, Michelson, D B, Peura, M, Pfaff, T** and **Wolff, D B** 2015 The emergence of open-source software for the weather radar community. *Bulletin of the American Meteorological Society*, 96(1): 117–128. DOI: https://doi.org/10.1175/BAMS-D-13-00240.1
11. **Michelson, D, Henja, A, Ernes, S, Haase, G, Koistinen, J, Ośródka, K, Peltonen, T, Szewczykowski, M** and **Szturc, J** 2018 BALTRAD advanced weather radar networking. *Journal of Open Research Software*, 6. DOI: https://doi.org/10.5334/jors.193
12. **Heistermann, M, Jacobi, S** and **Pfaff, T** 2013 Technical note: An open source library for processing weather radar data (wradlib). *Hydrology and Earth System Sciences*, 17(2): 863–871. URL https://www.hydrol-earth-syst-sci.net/17/863/2013/. DOI: https://doi.org/10.5194/hess-17-863-2013
13. **Helmus, J J** and **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. DOI: https://doi.org/10.5334/jors.119
14. **Mather, J H** and **Voyles, J W** 2013 The ARM climate research facility: A review of structure and capabilities. *Bulletin of the American Meteorological Society*, 94(3): 377–392. DOI: https://doi.org/10.1175/BAMS-D-11-00218.1
15. **Anderson, N G, Lang, T, Helmus, J J, Check your git settings!** and **Nesbitt, S.** nguy/artview: Artview release v1.3, August 2017. DOI: https://doi.org/10.5281/zenodo.853317
16. **Lang, T** 2015 Python-based scientific analysis and visualization of precipitation systems at NASA Marshall Space Flight Center. In: *5th Symposium on Advances in Modeling and Analysis Using Python*. URL https://ams.confex.com/ams/95Annual/webprogram/Paper262779.html.
17. **Dixon, M** and **Wiener, G** 1993 TITAN: Thunderstorm Identification, Tracking, Analysis, and Nowcasting-a

radar-based methodology. *Journal of Atmospheric and Oceanic Technology*, 10(6): 785–797. DOI: https://doi.org/10.1175/1520-0426(1993)010<0785:TTITAA>2.0.CO;2

18. **Pulkkinen, S, Nerini, D, Pérez Hortal, A A, Velasco-Forero, C, Seed, A, Germann, U** and **Foresti, L** 2019 Pysteps: An open-source python library for probabilistic precipitation nowcasting (v1.0). *Geoscientific Model Development*, 12(10): 4185–4219. URL https://www.geosci-model-dev.net/12/4185/2019/. DOI: https://doi.org/10.5194/gmd-12-4185-2019
19. **Fabry, F** 2015 *Radar Meteorology: Principles and Practice*. Cambridge University Press. DOI: https://doi.org/10.1017/CBO9781107707405
20. **Bringi, V N** and **Chandrasekar, V** 2001 *Polarimetric Doppler Weather Radar: Principles and Applications*. Cambridge University Press. ISBN 9780521623841. URL https://books.google.ch/books?id=KvJvfP9t5Y8C. DOI: https://doi.org/10.1017/CBO9780511541094
21. **Doviak, R J** and **Zrnic, D S** 2006 Doppler Radar and Weather Observations. *Dover Books on Engineering Series*. Dover Publications. ISBN 9780486450605. URL https://books.google.ch/books?id=ispLkPX9n2UC.
22. **Millman, K J** and **Aivazis, M** 2011 Python for scientists and engineers. *Computing in Science & Engineering*, 13(2): 9–12. URL https://aip.scitation.org/doi/abs/10.1109/MCSE.2011.36. DOI: https://doi.org/10.1109/MCSE.2011.36
23. **SPHINX contributors** 2020 Sphinx: Python documentation generator. URL https://www.sphinx-doc.org. [Online; accessed 10-March-2020].
24. **Pylint contributors** 2020 Pylint: Star your pyton code! URL https://www.pylint.org/. [Online; accessed 10-March-2020].
25. **Michelson, D B, Lewandowski, R, Szewczykowski, M, Beekhuis, H, Haase, G, Mammen, T, Faure, D, Simpson, M, Leijnse, H** and **Johnson, D** 2019 EUMETNET OPERA weather radar information model for implementation with the HDF5 file format version 2.3. *Technical report, EUMETNET OPERA*. URL https://www.eumetnet.eu/activities/observations-programme/current-activities/opera/.
26. **NOAA National Weather Service (NWS) Radar Operations Center** 1991 Noaa next generation radar (nexrad) level 2 base data. *Technical report, NOAA National Centers for Environmental Information*.
27. **Dixon, M** and **Lee, W-C** 2016 CfRadial data file format. *Technical report, EOL, NCAR*. URL https://github.com/NCAR/CfRadial/tree/master/docs.
28. **Rocklin, M** 2015 Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In: Huff, K and Bergstra, J (eds.), *Proceedings of the 14th Python in Science Conference*, 126–132. DOI: https://doi.org/10.25080/Majora-7b98e3ed-013

How to cite this article: Figueras i Ventura, J, Lainer, M, Schauwecker, Z, Grazioli, J and Germann, U 2020 Pyrad: A Real-Time Weather Radar Data Processing Framework Based on Py-ART. *Journal of Open Research Software,* 8: 28. DOI: https://doi.org/10.5334/jors.330

Submitted: 01 April 2020 **Accepted:** 16 September 2020 **Published:** 08 October 2020