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: e25, DOI: http://dx.doi.org/10.5334/jors.119

SOFTWARE METAPAPER

The Python ARM Radar Toolkit (Py-ART), a Library for Working with Weather Radar Data in the Python Programming Language

Jonathan J. Helmus and Scott M. Collis
Environmental Science Division, Argonne National Laboratory, Argonne, IL
Corresponding author: Jonathan J. Helmus (jhelmus@anl.gov)

The Python ARM Radar Toolkit is a package for reading, visualizing, correcting and analysing data from weather radars. Development began to meet the needs of the Atmospheric Radiation Measurement Climate Research Facility and has since expanded to provide a general-purpose framework for working with data from weather radars in the Python programming language. The toolkit is built on top of libraries in the Scientific Python ecosystem including NumPy, SciPy, and matplotlib, and makes use of Cython for interfacing with existing radar libraries written in C and to speed up computationally demanding algorithms. The source code for the toolkit is available on GitHub and is distributed under a BSD license.

Keywords: Python; radar; weather; weather radar; Doppler velocity
Funding statement: This paper has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. This research was supported by the Office of Biological and Environmental Research of the U.S. Department of Energy as part of the Atmospheric Radiation Measurement Climate Research Facility.

(1) Overview

Introduction

Weather radars are ideally suited to provide remotely sensed data of the state of the atmosphere over a wide geographic area. These instruments are capable of providing measurements related to the size, concentration, shape and motion of cloud and precipitation droplets. At frequencies below 4 GHz, these instruments are capable of detecting air motion during clear sky conditions by probing the differences in air density. Unlike other sensors, weather radars operate and provide meaningful measurements at all times and in most conditions. Additionally, radar measurements are not severely affected by clouds and dense precipitation, which limits the range of optical instruments.

The data provided by weather radars are often used for the near real-time forecasting of weather events by meteorologists, but can also be used to initialize and direct numerical weather predictions, provide key insights on atmospheric processes, serve as input for hydrological models, and develop climatological statistics. Given the wide uses of data from weather radar, it is not surprising that there has been considerable investment in the development and operation of these instruments worldwide. Networks of radars cover much of the inhabited regions of North America and Europe, providing updated views of the atmosphere for these continents multiple times per hour.

Measurements made by weather radars contain a tremendous amount of information, yet considerable effort must be made to extract out scientifically meaningful parameters. The raw voltage measurements must be converted into spectra from which moments are extracted using signal-processing routines. Quality control and correction routines are typically applied to flag and remove known artifacts from these radar measurements after which retrieval algorithms can be applied to derive geophysical parameters such as precipitation type and amount, wind speeds and directions, and cloud type [1, 2].

These corrected radar moments and retrieved properties can be analysed in a variety of ways to provide additional insight. For example, methods can be used to map how conditions change over time allowing the development of models that explain and predict atmospheric processes. Routines that identify and categorize features in the data can be run over large amounts of radar data to provide statistical information on weather and climate patterns.

Processing, correcting and analysing weather radar data covers a wide range of computational disciplines and fields. The instrument itself accomplishes some of this computational work, typically the signal processing steps and a limited amount of quality control and correction of moments. Other software must be used for additional processing and analysis. Given the wide scope of information that weather radars can provide, having flexible and extendable software is key. Users of such software will have a wide range of needs from basic visualization of the data to the development of complex processing pipelines. Ideally software should be easy to use for common tasks yet allow advanced users access to more powerful features.

In this paper, we describe the Python ARM Radar Toolkit, hereafter Py-ART, an open source Python package built on top of modules in the scientific Python ecosystem that can be used to visualize, correct and analyse radar data in a number of formats. This package provides a framework for working with radar data in Python from which sophisticated workflows can be created for end-to-end analysis.

A short history of the package will be given followed by a discussion of the features of the package. The next section provides details on the implementation of the toolkit as well as information on software engineering practices and tools used to develop and insure the software is of high quality. This is followed by information on the availability of the software and a short section on how portions of the whole package can be reused in future radar software.

History of the package

The Atmospheric Radiation Measurement Climate Research facility (ARM) is a US Department of Energy program, which provides *in situ* and remote sensing observatories to the climate research community [3, 4]. The program's mission is to improve the understanding and representation of clouds and aerosols. Since the early 1990's, the program has been collecting and providing data to users from a wide array of instruments which probe the atmosphere. With investments from the American Recovery Act, the program acquired a number of scanning cloud and precipitation radars and upgraded existing profiling systems. The Python ARM Radar Toolkit (Py-ART) was created in order to work with the data from these new radars when existing software was not found to meet the needs of the program.

As development of Py-ART progressed, it was recognized that many of the processes and algorithms contained within the software were not unique to the ARM program and would be of use to the wider radar community. In 2013, Py-ART was released as open source software and the scope of the project expanded to address the needs of the nascent open source radar community. New features were added to the toolkit such as the ability to read files from non-ARM radars, and a community of developers and users both within the ARM program but also outside of the program formed around the software. Today, Py-ART contains contributions from developers at universities, government programs, and radar enthusiasts from around the globe. The user base is continuously growing as users learn of the package through short courses and presentations at conferences, the project's online documentation, and word of mouth from other users.

Implementation and architecture

Features and Use

Py-ART is a Python package that provides a variety of routines for reading, processing, analysing and visualizing data from weather radars. The package is organized in a number of sub-packages roughly separated by the type of functionality they provide.

The *core* sub-packages contain the **Radar** and **Grid** classes. These are the primary objects used throughout Py-ART to store moments, pointing information and metadata from weather radars in the native elevation, azimuth and range coordinates or Cartesian coordinates. Other functions within Py-ART create, modify or visualize data within these classes. The **Radar** class stores data from a single radar volume in memory using a layout which is based upon the structures used to store radar volumes on disk as recommended by the CF/Radial convention [5].

The *io* sub-package provides the ability to read in radar data from a number of common radar file formats into **Radar** and **Grid** objects. This sub-package also contains routines for writing data from **Radar** and **Grid** objects out to disk as NetCDF files. Support for reading data from additional file formats is provided in the *aux_io* sub-package. Formats supported in this sub-package are more limited in scope than those in the *io* sub-package. A summary of the file formats supported by Py-ART is given in **Table 1**.

The *graph* sub-package contains classes and routines for visualizing radar data. The **RadarDisplay** class can be used to plot data from radar volumes made up of plan position indicator (PPI) or range-height indicator (RHI) sweeps as well as from vertically pointing instruments. **Figure 1** shows an example plot created using the **RadarDisplay** class to plot an RHI sweep from an X-band radar operated by the ARM program at its Southern Great Plains site. Data from PPI scans or Cartesian grids can be visualized on cartographic maps using the **RadarMapDisplay** or **GridMapDisplay** classes. **Figure 2** plots PPI scans from an ARM C-Band radar which was created using the **RadarMapDisplay** class. The sub-modules also contain a **RadarDisplay_Airborne** class for plotting data from radars attached to aircraft as well as a number of colormaps appropriate for plotting various radar moments.

Format	Reading	Writing
CfRadial	X	X
CSU Chill	X	
GAMIC	Partial	
MDV	X	
NEXRAD Level II	X	
NEXRAD Level III	X	
ODIM H5	Partial	
Sigmet	X	
UF	X	X

Table 1: Radar file formats supported by Py-ART.

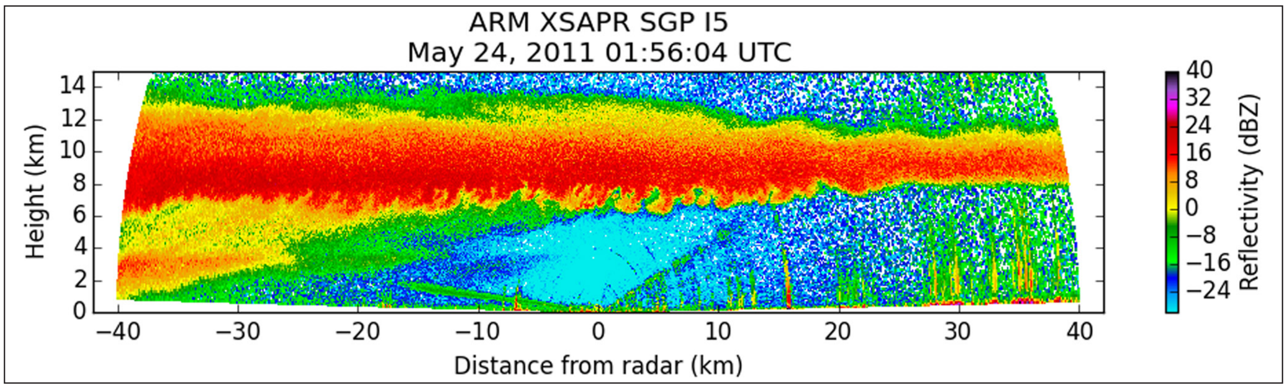

Figure 1: A plot of an RHI scan from an X-band radar operated by ARM at their Southern Great Plains site created using Py-ART.

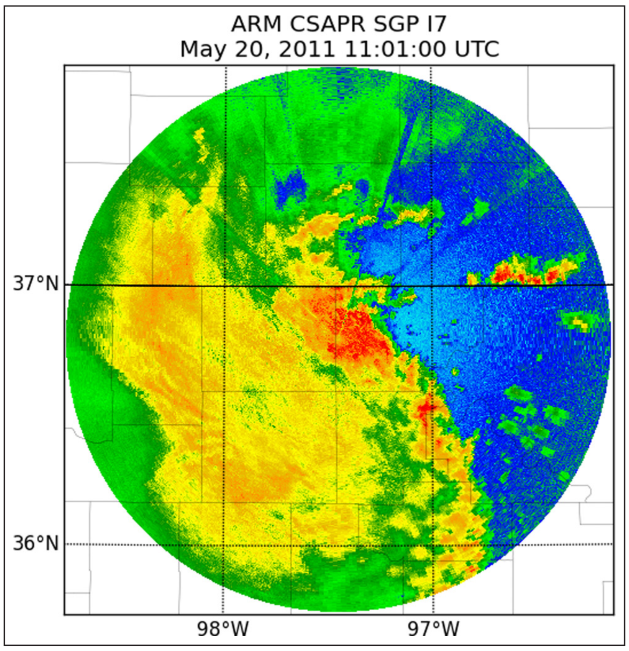

Figure 2: A plot of a PPI scan from an ARM C-band radar at the Southern Great Plains site created using the **RadarMapDisplay** class in Py-ART.

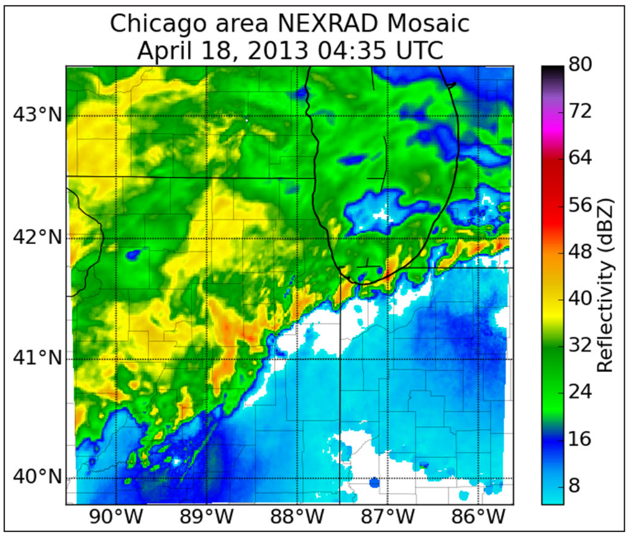

Figure 3: A mosaic of radar reflectivity from five NEXRAD radars near the Chicago, IL area during an intense rainfall event created using the gridding and visualization features in Py-ART.

The *correct* sub-module contains algorithms for correcting artifacts found in radar data. Multiple routines are included for unfolding or dealiasing radial Doppler velocities based upon the FourDD algorithm [6], multidimensional phase unwrapping [7, 8] and a novel region-based algorithm. The sub-module also contains routines for correcting attenuation using polarametric variables [9] and processing differential phase using a linear programming method [10]. A limited number of radar retrievals are available in the *retrieve* sub-module.

The *map* sub-module provides routines for creating regular Cartesian grids from data from one or multiple radars. Grid points are found by interpolation of all radar gates within a specified radius of influence using Cressman [11] or Barnes [12] weighting. **Figure 3** shows a mosaic over the Chicago, IL region created from radar volumes from five NEXRAD radars in the area.

Py-ART is written predominately in the Python programming language [13]. Python is a high-level, interpreted programming language that is known for its expressive, concise and easy to read syntax. Python is a general purpose programming language with a large standard library and comprehensive repository of third-party packages that includes a rich set of open source libraries designed for scientific computing.

Py-ART makes use of many of these scientific Python libraries. The multi-dimensional array data structure from the NumPy library [14] is used as an effective means of storing numerical radar data in memory. Routines in NumPy as well as from the SciPy library [15] are used within Py-ART to operate upon these arrays at speeds that approach those of compiled programming languages. Visualization in Py-ART makes use of matplotlib [16], another Python library widely used in the scientific community. Increasingly, data used in the atmospheric sciences community is stored and shared in NetCDF (Network Common Data Format [17]) files. Many weather radars offer an option to output data in this format. In order to read and write from this format, Py-ART uses the netcdf4-python library [18].

Portions of Py-ART are written in Cython, a superset of Python that translates to C that is compiled to create a Python extension module. Cython allows Python code to interface with C and C++ routines and is used in Py-ART to interface with the TRMM RSL library [19] as well as to create a Python wrapper around the FourDD [6] dealiasing algorithm. Py-ART also uses Cython to speed up numerical Python routines by adding static types to some of the variables. This process is only used when these routines cannot be efficiently written using NumPy or SciPy and the performance improvements obtained by using Cython are significant. A small portion of Py-ART is written in Fortran and uses f2py [20] to interface with Python.

Development and project hosting of Py-ART are done on GitHub. This service allows Py-ART to make use of a hosted git version control system for tracking changes to the software's source code. In addition, GitHub provides a bug tracker, feature request system, wiki, and documentation hosting for the project. Contributions to Py-ART are typically submitted as a "pull request" through GitHub. All of the contributions by core developers follow this process.

Quality control

To insure that high quality, working software is released, Py-ART includes, maintains and runs a large suite of unit tests, which exercise the majority of the functionality of the package. The nose testing framework [21] is used to design and run tests in Py-ART. Plug-ins to this package provide details on the code coverage of the test suite, which is monitored and used to find regions of code that require additional units tests.

To insure that new additions to Py-ART work and do not break existing functionality, a continuous integration platform run by Travis CI builds the package from source and runs the unit tests upon each commit to the repository. These same checks are run on pull requests submitted by contributors. If these tests fail, the developers are notified so that modifications to the code can be made to fix these issues prior to merging the changes into Py-ART.

To aid end users and developers in using and understanding Py-ART, a comprehensive set of documentation is available. This includes both a User and Developer Reference Manual, which provide a listing of all functions and classes in the packages aimed at either users or developers of the package. A number of examples are also available which provide pre-written scripts that demonstrate some of the common use cases of Py-ART.

This documentation is built using the Sphinx documentation generator [22] that creates the HTML documentation, which is hosted on GitHub. This website is automatically updated by Travis CI whenever new code is added to ensure that the documentation and code stay in sync.

Documentation of the functions, classes and methods is included directly in the source code as comments formatted as reStructuredText following the standards used in NumPy and SciPy. These "docstrings" are extracted by Sphinx from the source code and formatted nicely using the numpydoc Sphinx extension [23]. The examples included in the documentation are also generated using Sphinx using an extension originally developed by the scikit-learn [24] project. These examples execute code on real radar data and serve as a limited set of functional tests for Py-ART.

(2) Availability

Operating system

Linux, OS X, and Windows.

Programming language

Python 2.6 and 2.7, 3.3, 3.4. and 3.5

Additional system requirements

None.

Dependencies

NumPy 1.6+, SciPy 0.11+, matplotlib 1.1+, netcdf4-python 1.0.2+. Additional optional dependencies required for certain features.

List of contributors

Jonathan Helmus, Argonne National Laboratory
Scott Collis, Argonne National Laboratory
Anderson Gama, Universität Stuttgart
Kirk North, McGill University
Joseph Hardin, Pacific Northwest National Laboratory
Nick Guy, University of Wyoming
Kai Muehlbaer, University of Bonn
Tim Lang, Marshall Space Flight Center

Software location

Archive (e.g. institutional repository, general repository) (required)

Name: Figshare
Persistent identifier: https://dx.doi.org/10.6084/m9.figshare.2202553.v1
Licence: BSD
Publisher: UChicago Argonne LLC
Version published: 1.6.0
Date published: 02/15/16

Code repository (e.g. SourceForge, GitHub etc.) (required)

Name: GitHub
Identifier: https://github.com/ARM-DOE/pyart
Licence: BSD
Date published: 02/01/16

Language

English

(3) Reuse potential

Despite being a relatively new software package, Py-ART is already seeing use within the meteorology and weather radar research communities. Py-ART drives the back end of a number of websites that provide plots of data from weather radars. Py-ART is used at universities and research institutes to visualize and process data from weather radars for both educational and research activities including a recent publication where Py-ART was used to detect radar signatures of deep convection [25].

Often Py-ART is one component in a workflow that includes a number of radar software packages. This includes other open source weather radar libraries including wradlib [26], RadX [27], and BALTRAD [28]. Members from these packages along with the authors recently worked together to provide a vision for open source radar software [19] and built a Virtual Machine which includes these software packages for use in a short course on open source radar software [29].

Py-ART has been designed to allow the reuse of many of its individual components. When possible, the implementation of an algorithm or routine was separated from the architecture of the package. For example, to read in data from a NEXRAD Level III file in Py-ART, a user would use the *read_nexradl_level3* function that returns a **Radar** instance. The **NEXRADLevel3File** class allows for low-level access to NEXRAD files that does not use the **Radar** class data model. In fact, the module that contains the **NEXRADLevel3File** is not coupled to any portions of Py-ART and can be used independently from the rest of the package. Many of the other functions which read in data from files follow this setup, with a class providing low-level access to the file content which can be reused outside of Py-ART. Many of Py-ART's processing and analysis routines are also designed in a two-layer manner where the top layer uses Py-ART specific classes and objects, and the lower layer uses more basic variables.

In addition to being re-usable, Py-ART is also extendable. The classes and functions within Py-ART can serve as a base upon which additional functionality can be built. Already, members of the community have written the ARM Radar Toolkit Viewer, ARTView [30], an interactive radar viewing browser using Py-ART as a base. Tim Lang from NASA's Marshall Space Flight Center has released a number of packages that extend Py-ART to provide additional processing methods including PyTDA for detecting Turbulence and PyBlock for estimating beam blockage [31]. Colorado State University has released a CSU_RadarTools package that contains Python tools for polarimetric radar retrievals [32]. The documentation for these packages uses Py-ART for data loading and visualization.

Acknowledgements

The authors wish to thank the users of Py-ART for their encouragement and suggestions as well as the contributors to the various scientific Python packages for creating such a powerful programming environment. Finally, the authors wish to thank Argonne National Laboratory and the Atmospheric Radiation Measurement Climate Research Facility for supporting the development of Py-ART.

Competing interests

The authors declare that they have no competing interests.

References

1. **Doviak, R** and **Zrnic, D** 2006 Doppler Radar and Weather Observations: Second Edition, Dover Publications.
2. **Bringi, V** and **Chandrasekar, V** 2005 Polarimetric Doppler Weather Radar: Principles and Applications, Cambridge University Press.
3. **Ackerman, T** and **Stokes, G** 2003 'The Atmospheric Radiation Measurement Program,' *Phys. Today*, 56 (1), 38–44. DOI: http://dx.doi.org/10.1063/1.1554135
4. **Mather, J** and **Voyles, J** 2012 'The Arm Climate Research Facility: A Review of Structure and Capabilities,' *Bull. Am. Meteorol. Soc.*, 94 (3), 377–392. DOI: http://dx.doi.org/10.1175/BAMS-D-11-00218.1
5. **Dixon, M, Lee, W, Rilling, B, Burghard, C** and **Van Andel, J** 2015 'CfRadial data file format: Proposed CF-compliant netCDF format for moments data for RADAR and LIDAR in radial coordinates.' URL: http://www.ral.ucar.edu/projects/titan/docs/radial_formats/CfRadialDoc.v1.3.20130701.pdf.
6. **James, C** and **Houze Jr, R** 2001 'A real-time four-dimensional Doppler dealiasing scheme,' *J. Atmospheric Ocean. Technol.,* 18 (10), 1674–1683. DOI: http://dx.doi.org/10.1175/1520-0426(2001)018<1674:ARTFDD>2.0.CO;2
7. **Herráez, M, Burton, D, Lalor, M** and **Gdeisat, M** 2002 'Fast two-dimensional phase-unwrapping algorithm based on sorting by reliability following a non-continuous path,' *Appl. Opt.,* 41 (35), 7437–7444. DOI: http://dx.doi.org/10.1364/AO.41.007437
8. **Abdul-Rahman, H, Gdeisat, M, Burton, D** and **Lalor, M** 2005 'Fast three-dimensional phase-unwrapping algorithm based on sorting by reliability following a non-continuous path,' *SPIE Proceedings,* 5856, 32–40. DOI: http://dx.doi.org/10.1117/12.611415
9. **Gu, J, Ryzhkov, A, Zhang, P, Neilley, P, Knight, P, Wolf, B** and **Lee, D** 2011 'Polarimetric Attenuation Correction in Heavy Rain at C Band,' *J. Appl. Meteorol. Clim.*, 50 (1), 39–58. DOI: http://dx.doi.org/10.1175/2010JAMC2258.1
10. **Giangrande, S, McGraw, R** and **Lei, L** 2013 'An Application of Linear Programming to Polarimetric Radar Differential Phase Processing,' *J. Atmospheric Ocean. Technol.*, 30 (8), 1716–1729, Aug. 2013. DOI: http://dx.doi.org/10.1175/JTECH-D-12-00147.1
11. **Cressman, G** 1959 'An operational objective analysis system,' *Mon. Weather Rev.*, 87 (10), 367–374. DOI: http://dx.doi.org/10.1175/1520-0493(1959)087<0367:AOOAS>2.0.CO;2
12. **Barnes, S** 1964, 'A Technique for Maximizing Details in Numerical Weather Map Analysis,' *J. Appl. Meteorol.*, 3 (4), 396–409. DOI: http://dx.doi.org/10.1175/1520-0450(1964)003<0396:ATFMDI>2.0.CO;2
13. **van Rossum, G** 1995 'Python Tutorial, Technical Report CS-R9526,' Centrum voor Wiskunde en Informatica (CWI).
14. **Oliphant, T** 2007 "Python for Scientific Computing," *Comput. Sci. Eng.*, 9 (3), 10–20. DOI: http://dx.doi.org/10.1109/MCSE.2007.58

15. **Jones, E, Oliphant, T** and **Peterson, P** 2001 'SciPy: Open source scientific tools for Python,' URL: http://www.scipy.org/
16. **Hunter, J** 2007 'Matplotlib: A 2D Graphics Environment,' *Comput. Sci. Eng.*, 9 (3), 90–95. DOI: http://dx.doi.org/10.1109/MCSE.2007.55
17. **Unidata** 'Network Common Data Form (netCDF)'. DOI: http://doi.org/10.5065/D6H70CW6
18. **Whitaker, J** 2015 'netCDF4 API documentation'. URL: http://unidata.github.io/netcdf4-python/
19. **Heistermann, M, Collis, S, Dixon, M, Giangrande, S, Helmus, J, Kelley, B, Koistinen, J, Michelson, D, Peura, M, Pfaff, T** and **Wolff, D** 2014 'The Emergence of Open-Source Software for the Weather Radar Community,' *Bull. Am. Meteorol. Soc.*, 96 (1), 117–128. DOI: http://dx.doi.org/10.1175/BAMS-D-13-00240.1
20. **Peterson, P** 2009 'F2PY: a tool for connecting Fortran and Python programs,' *Int. J. Comput. Sci. Eng.*, 4 (4), 296. DOI: http://dx.doi.org/10.1504/IJCSE.2009.029165
21. **Pellerin, J** 2009 'nose'. URL: https://nose.readthedocs.org/en/latest/
22. **Sphinx: Python Documentation Generator** 2015 URL: http://sphinx-doc.org/
23. **numpydoc – Numpy's Sphinx extensions** 2015 URL: https://github.com/numpy/numpydoc
24. **Pedregosa, F, Varoquaux, G, Gramfort, A, Michel, V, Thirion, B, Grisel, O, Blondel, M, Prettenhofer, P, Weiss, R, Dubourg, V, Vanderplas, J, Passos, A, Cournapeau, D, Brucher, M, Perrot, M** and **Duchesnay, É** 2011 'Scikit-learn: Machine Learning in Python,' *J. Mach. Learn. Res.*, 12, 2825–2830.
25. **Lier-Walqui, M, Fridlind, A, Ackerman, A, Collis, S, Helmus, J, MacGorman, D, North, K, Kollias, P** and **Posselt, D** 2015 'On Polarimetric Radar Signatures of Deep Convection for Model Evaluation: Columns of Specific Differential Phase Observed during MC3E', *Monthly Weather Review*, 144 (2), 737–758. DOI: http://dx.doi.org/10.1175/MWR-D-15-0100.s1
26. **Heistermann, M, Jacobi, S** and **Pfaff, T** 2013 'Technical Note: An open source library for processing weather radar data (*wradlib*),' *Hydrol. Earth Syst. Sci.*, 17 (2), 863–871. DOI: http://dx.doi.org/10.5194/hess-17-863-2013
27. **Dixon, M** 2015 'RADX.' URL: http://www.ral.ucar.edu/projects/titan/docs/radial_formats/radx.html.
28. **Michelson, D, Koistinen, J, Peltonen, T, Szturc, J** and **Rasmussen, M** 2012 'Advanced weather radar networking with BALTRAD+,' *Proceedings of ERAD 2012*, Toulouse, France.
29. **Heistermann, M, Collis, S, Dixon, M, Helmus, J, Henja, A, Michelson, D** and **Pfaff, T** 2015 'An Open Virtual Machine for Cross-Platform Weather Radar Science,' *Bull. Am. Meteorol. Soc.*, 96, 1641–1645. DOI: http://dx.doi.org/10.1175/BAMS-D-14-00220.1
30. **Guy, N, Gama, A, Lang, T** and **Hein, P** 2015 'artview: ARTview release 1.0' DOI: http://dx.doi.org/10.5281/zenodo.27358
31. **Lang, T** 2015 'Python-based scientific analysis and visualization of precipitation systems at NASA Marshall Space Flight Center', *95th American Meteorological Society Annual Meeting*. URL: https://ams.confex.com/ams/95Annual/webprogram/Paper262779.html.
32. **Lang, T** 2015 'CSU_RadarTools,'' URL: https://github.com/CSU-Radarmet/CSU_RadarTools.

How to cite this article: 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: e25, DOI: http://dx.doi.org/10.5334/jors.119

Submitted: 15 February 2016 **Accepted:** 01 July 2016 **Published:** 18 July 2016