Jackson, R, et al. 2020 PyDDA: A Pythonic Direct Data Assimilation Framework for Wind Retrievals. *Journal of Open Research Software,* 8: 20. DOI: https://doi.org/10.5334/jors.264

SOFTWARE METAPAPER

PyDDA: A Pythonic Direct Data Assimilation Framework for Wind Retrievals

Robert Jackson[1], Scott Collis[1], Timothy Lang[2], Corey Potvin[3] and Todd Munson[1]

[1] Argonne National Laboratory, Argonne, IL, US

[2] NASA Marshall Space Flight Center, Huntsville, AL, US

[3] NOAA/OAR National Severe Storms Laboratory, and Cooperative Institute for Mesoscale Meteorological Studies, University of Oklahoma, Norman, OK, US

Corresponding author: Robert Jackson (rjackson@anl.gov)

This software assimilates data from an arbitrary number of weather radars together with other spatial wind fields (eg numerical weather forecasting model data) in order to retrieve high resolution three dimensional wind fields. PyDDA uses NumPy and SciPy's optimization techniques combined with the Python Atmospheric Radiation Measurement (ARM) Radar Toolkit (Py-ART) in order to create wind fields using the 3D variational technique (3DVAR). PyDDA is hosted and distributed on GitHub at https://github.com/openradar/PyDDA. PyDDA has the potential to be used by the atmospheric science community to develop high resolution wind retrievals from radar networks. These retrievals can be used for the evaluation of numerical weather forecasting models and plume modelling. This paper shows how wind fields from 2 NEXt generation RADar (NEXRAD) WSR-88D radars and the High Resolution Rapid Refresh can be assimilated together using PyDDA to create a high resolution wind field inside Hurricane Florence.

Keywords: winds; radar; weather; high resolution; doppler

Funding statement: The development of this software is supported by the Climate Model Development and Validation (CMDV) activity which is funded by the Office of Biological and Environmental Research in the US Department of Energy Office of Science.

(1) Overview

Introduction

High resolution 3D wind retrievals have a wide range of practical applications. For example, 3D wind retrievals are frequently used to gain insight on the kinematic processes inside thunderstorms, hurricanes, and tornadoes (i.e. [1, 2]). Furthermore, 3D wind fields are critical for dispersion modelling [3] and for predicting potential hazards to infrastructure. These wind retrievals are commonly created from data from networks of scanning weather radars such as those from the NEXt generation RADar (NEXRAD) network of WSR-88Ds installed by the National Oceanic and Atmospheric Adminstration (NOAA) all over the United States that can detect the velocity of particles in the direction of the radar beam over a volume, commonly called the radial or Doppler velocity [4, 5]. The NEXRAD network samples volumes of approximately 250 km by 250 km by 20 km every 5 to 10 minutes. While other wind measurements such as profilers and anemometers provide the most accurate wind measurements, they measure a much more limited volume than a scanning radar being limited to the column and single point in space respectively. Numerical weather forecasting models provide 3D winds with a greater volume coverage than the NEXRAD network, but they are affected by uncertainties in the model prediction and assimilation and generally provide data at coarser spatial and temporal scales than wind observations made by scanning radars. Therefore a solution for integrating wind observations from many different sources operating at differing spatial and temporal scales has the best capability of providing a complete picture of the spatial and temporal evolution of the 3D wind field.

Since each sensor and model produces winds at differing spatial and temporal scales, retrieving the 3D winds from them is a nontrivial task. For a single radar, deriving a 3D wind field amounts to solving one equation with three unknowns since only the wind velocity in the direction relative to the radar beam is known. Therefore the problem of deriving 3D winds from one radar requires additional constraints. For more than one radar, we can increase the number of equations corresponding to each radar. However, even in the multi-radar scenario, there are measurement uncertainties related to the differing sampling strategies and sensitivities of each radar. Therefore extra assumptions must be made to retrieve the 3D wind field from velocities recorded network of radars and produced by models. Two frameworks have typically been used to retrieve the 3D winds from a network of radars. One is the framework that imposes a strong constraint by integrating the mass continuity equation from the surface to the top of the atmosphere [6, 7]. The other, is the 3D variational (3DVAR) framework that minimizes a sum of cost functions related to radar measurements, equations of motion, balloon based profiles of wind measurements and high resolution weather forecasting models [8, 9, 10, 11, 12]. The 3DVAR framework has the advantage in that it is easier to integrate observations from extra sensors and models than the strong constraint framework and is less sensitive to initial and boundary conditions [9, 12, 13]. Therefore, the current standard for 3D wind retrievals from radars is the 3DVAR framework.

Past software applied to 3D wind retrievals includes Custom Editing and Display of Reduced Information (CEDRIC) [13] and MultiDop [9, 11, 14]. CEDRIC is based off of the traditional mass continuity integration technique which, again, limits the ability of the user to account for measurement uncertainties than 3DVAR. While ground-breaking at the time, CEDRIC is difficult to use as it involves learning a separate scripting language just to use it. Multidop is a Python wrapper around a C program implementing the 3DVAR technique called Dual Doppler Analysis (DDA). Multidop bought 3DVAR winds to the open source community for the first time. However, it is based off of legacy code for the minimization of the cost function, can be difficult to compile, currently only supports up to 3 radars, is challenging to extend, and is not thread-safe for mass processing of winds on a multiple computational cores. Pythonic Direct Data Assimilation (PyDDA) was developed in order to make a fully Pythonic, open source solution for implementing the 3DVAR framework for 3D wind retrievals where data from an arbitrary number of radars and models can be added. PyDDA is fully written in Python and is built on standard packages in the Scientific Python ecosystem including NumPy [15], SciPy [16], Cartopy [17], and matplotlib [18] as well as the Python ARM Radar Toolkit [19].

In this paper, we will first introduce how the 3DVAR framework in PyDDA was implemented using the Scientific Python ecosystem and how contributors can expand PyDDA to include their data. After that, an example case of using the software to retrieve winds from NEXRAD observations and High Resolution Rapid Refresh (HRRR) model runs in Hurricane Florence is presented. Finally, details on how PyDDA is quality controlled are presented.

Implementation and architecture

3DVAR framework

Let $\mathbf{v}(x, y, z) = (u(x, y, z), v(x, y, z), w(x, y, z)) \in \Re^3$ be the analysis wind field over a Cartesian grid. The $\mathbf{v}(x, y, z)$ are defined over a discrete Cartesian grid of m by n by o points provided as $(x_{ijk}, y_{ijk}, z_{ijk})$ for $i \in [0, m)$, $j \in [0, n)$, and $k \in [0, o)$. PyDDA derives $\mathbf{v}(x_{ijk}, y_{ijk}, z_{ijk})$ for $i \in [0, m)$, $j \in [0, n)$ by finding the $\mathbf{v}(x_{ijk}, y_{ijk}, z_{ijk})$ that minimizes the cost function in Equation (1).

$$J(\mathbf{v}) = C_o J_o + C_{mass} J_{mass} + C_v J_v + C_r J_r + C_s J_s + C_{model} J_{model} + C_{point} J_{point} \quad (1)$$

with a gradient $\nabla J(\mathbf{v})$ that is the sum of the gradients $C_n \nabla J_n$ of each term in Equation 1. All of the J_n, ∇J_n terms are defined as in **Tables 1** and **2**. Detailed formulas of each function in **Tables 1** and **2** are available in the source code of the `cost_functions` module and in [9, 10, 11]. Equation (1) is written in Python using the NumPy and SciPy libraries.

The C_n terms are user adjustable weighting coefficients for each constraint. Therefore, the importance of a particular measurement or model derived data point to the constraint can be adjusted. This is useful for accounting for the uncertainties in each constraint. For example, the retrieval can be more weakly constrained by model data that has higher uncertainty than the radar observations. Furthermore, since Equation (1) is a sum of cost functions and their gradients, adding more terms to Equation (1) is a simple way to expand this framework. All of the functions in **Tables 1** and **2** are currently implemented in the `cost_functions` module of PyDDA.

Optimization problem

Finding the $\mathbf{v}(x_{ijk}, y_{ijk}, z_{ijk})$ that minimizes $J(\mathbf{v})$ involves the minimization of a cost function of the order of a million ($3mno$) dimensions, three dimensions for each $(x_{ijk}, y_{ijk}, z_{ijk})$ for $i \in [0, m)$, $j \in [0, n)$, and $k \in [0, o)$. SciPy's implementation of the Limited Memory Broyden-Fletcher-Goldfarb-Shanno Bounded (L-BFGS-B) optimization technique is suited for

Table 1: List of cost functions currently implemented in PyDDA.

Cost Function	Symbol	Routine
Total	$J(\mathbf{v})$	`J_function`
Radar observations	J_o	`calculate_radial_vel_cost_function`
Mass continuity	J_{mass}	`calculate_mass_continuity`
Vertical vorticity	J_v	`calculate_vertical_vorticity_cost`
Rawinsonde	J_r	`calculate_background_cost`
Smoothness	J_s	`calculate_smoothness_cost`
Model	J_{model}	`calculate_model_cost`
Point	J_{point}	`calculate_point_cost`

Table 2: List of cost function gradients currently implemented in PyDDA.

Cost Function	Symbol	Gradient
Total	$\nabla J(\mathbf{v})$	`grad_J`
Radar observations	∇J_o	`calculate_grad_radial_vel`
Mass continuity	∇J_{mass}	`calculate_mass_continuity_gradient`
Vertical vorticity	∇J_v	`calculate_vertical_vorticity_gradient`
Rawinsonde	∇J_r	`calculate_background_gradient`
Smoothness	∇J_s	`calculate_smoothness_gradient`
Model	∇J_{model}	`calculate_model_gradient`
Point	∇J_{point}	`calculate_point_cost`

Table 3: List of current initialization options in PyDDA.

Initialization	Routine name
Constant wind field	`make_constant_wind_field`
Py-ART `HorizontalWindProfile` object	`make_wind_field_from_profile`
Weather Research and Forecasting (WRF)	`make_background_from_wrf`
High Resolution Rapid Refresh (HRRR)	`make_initialization_from_hrrr`

minimizing such a cost function [16, 20]. The L-BFGS-B technique needs the cost function $J(\mathbf{v})$, its gradient $\nabla J(\mathbf{v})$, the bounds of $\mathbf{v}(x_{ijk}, y_{ijk}, z_{ijk})$ and initial guess of $\mathbf{v}(x_{ijk}, y_{ijk}, z_{ijk})$ for $i \in [0, m)$, $j \in [0, n)$, and $k \in [0, o)$, all of which are obtainable since $J(\mathbf{v})$ is twice differentiable. L-BFGS-B is suited for minimizing $J(\mathbf{v})$ as it is designed to conserve memory which is important when the dimensionality of $J(\mathbf{v})$ is on the order of millions. Furthermore, since L-BFGS-B minimizes $J(\mathbf{v})$ over a bounded domain, this ensures that convergence to a physically realistic solution is reached. In addition, SciPy's implementation of L-BFGS-B takes advantage of as many cores that are present in a single machine allowing for fast convergence.

The `retrieval` module contains the code that implements the 3DVAR technique using L-BFGS-B. In particular, the code is contained within the `get_dd_wind_field` procedure which takes in input from an arbitrary number of Py-ART `Grid` objects representing an arbitrary number of radars and automatically enters in the optimal inputs to L-BFGS-B for the user. `get_dd_wind_field` will run iterations of L-BFGS-B until convergence is reached. Convergence is reached when either $|\nabla(J(\mathbf{v}))| < 10^{-3}$ or when the maximum change in w over 10 iterations is less than 0.2 $m\ s^{-1}$.

Another component of the optimization problem is that the final $\mathbf{v}$ may be sensitive to the initial guess for $\mathbf{v}$. Therefore, PyDDA has an `initalization` module that provides users with several options for initial guesses listed in **Table 3**. All of these functions return the initial $\mathbf{v}$ as a 3-tuple of NumPy arrays on the Py-ART `Grid` objects' grid. This standardized form for outputs in the `initialization` module allows for the addition of custom initializations.

Having both the initialization and retrieval contained within PyDDA reduces the wind retrieval technique to just a few lines of code as shown below.

```
import pyart
import pydda

berr_grid = pyart.io.read_grid("grid1.nc")
cpol_grid = pyart.io.read_grid("grid2.nc")
sounding = pyart.io.read_arm_sonde("arm_sounding.cdf")

# Load sounding data and insert as an intialization
u_init, v_init, w_init = pydda.initialization.make_wind_field_from_profile(
    cpol_grid, sounding)

# Start the wind retrieval. This example only uses the mass
# continuity and radar observational constraints.
Grids = pydda.retrieval.get_dd_wind_field([berr_grid, cpol_grid],
    u_init, v_init, w_init, Co=10.0, Cm=1500.0)
```

In addition to constraining against radar observations, PyDDA includes support for constraining against model wind data interpolated to the analysis grid. Currently, the `constraints` module contains procedures that interpolate WRF and HRRR data to PyDDA's analysis grid using SciPy's interpolation module. These modules add model fields to Py-ART Grid objects using the NetCDF4 and cfgrib modules for reading the model data. `get_dd_wind_field` takes in any number of wind fields with any name, adding support for more constraints from various data.

Data visualization

In addition to calculation, PyDDA comes with a basic `visualization` module that supports the plotting of wind barbs and streamlines over gridded radar reflectivity fields for easy viewing of the results. `visualization` was written in Python and uses the matplotlib and cartopy libraries for visualization of the data. **Figure 1** shows an example cross section through a thunderstorm sampled by 2 Doppler radars in Darwin, Australia. In addition to plotting over the analysis grid, plotting over geographic maps is supported through the use of the cartopy package [17], with an example streamline plot shown in **Figure 2**.

Software Quality control

Using the GitHub issue tracker, users can request for help with PyDDA as well as point out potential issues with the software. This provides a forum for contributors to make various improvements to PyDDA. When a contributor makes a modification to PyDDA, the contributor submits a pull request to the master branch of `openradar/PyDDA` on GitHub. For each pull request, `pytest` runs a suite of tests on each module of PyDDA to assure that the quality of the retrievals are maintained. In addition, whenever a user submits a pull request on GitHub, Travis CI builds PyDDA in the test environment and runs pytest whenever a pull request is made to the master branch of PyDDA. Travis CI runs these tests on Python 3.6, 3.7, and 3.8 environments on Linux. If the build from Travis CI fails, the contributor is asked to resolve the issues with the code until the continuous integration completes successfully. In addition, each pull request has to be approved by the main developer. Therefore, the main developer determines if the contributor should write a unit test for their added module. Furthermore, PyDDA is also released on the conda-forge repository which requires that Circle CI, Travis CI, and AppVeyor all run PyDDA's unit tests before the release is published on conda-forge.

Unit tests also need to validate the results of PyDDA against basic dynamical principles. For example, the unit tests that tests the `get_dd_wind_field` function will perform the retrieval on the example in **Figure 2** against previous 3DVAR retrievals of this same dataset by [21]. While there are differences between the retrieval used by [21] and in PyDDA, both [21] and PyDDA should show resolve a strong updraft ($w > 1$) in the location given in **Figure 2b**. Therefore, the test checks for the presence of this updraft. In order to test individual cost functions, properties of the wind field that would be expected from physical or mathematical principles are checked. For example, in order to test the $J_{mass} = \nabla \cdot \mathbf{v}$, the test check to see if J_{mass} is zero in a constant wind field, but negative in a convergent wind field and positive in a divergent wind field.

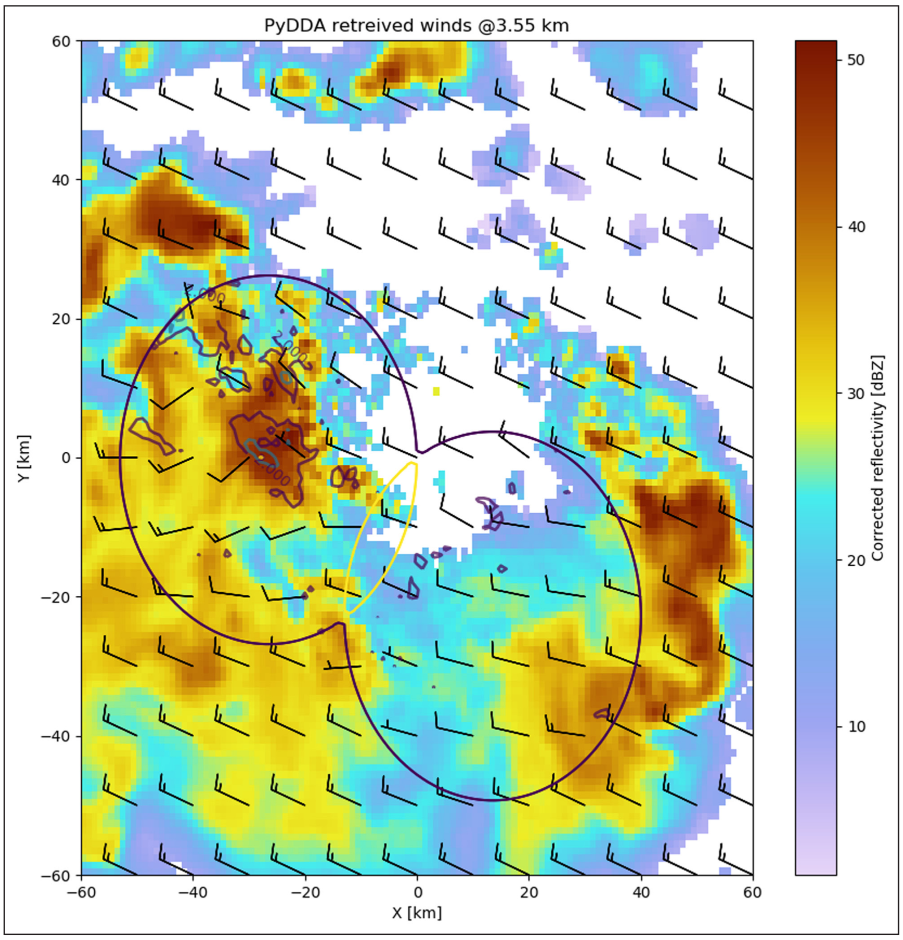

Figure 1: An example wind barb plot overlaid on radar reflectivity at 3.5 km altitude for winds retrieved in thunderstorms sampled by 2 radars over Darwin. Contours represent the presence of updrafts with given velocities at the 3.5 km height level. The area inside the two circles indicate where the wind retrieval is most reliable.

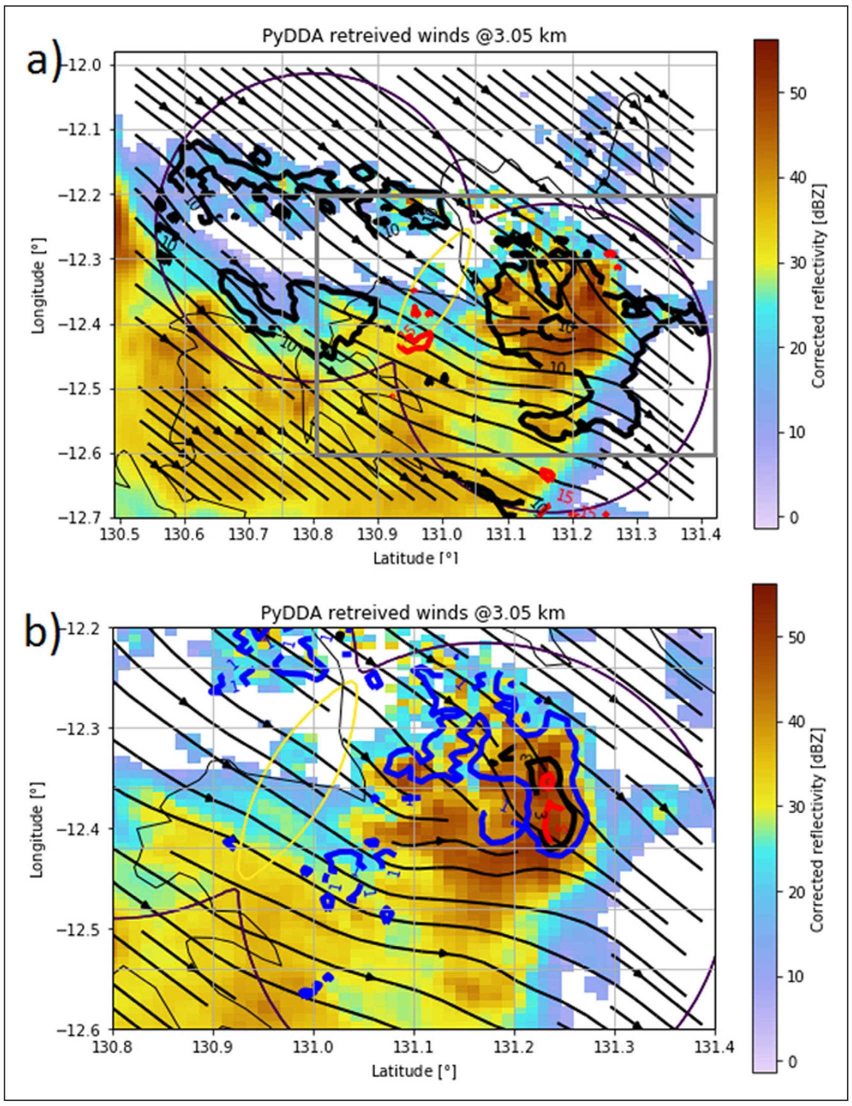
a)
PyDDA retreived winds @3.05 km
Longitude [°]
Latitude [°]
Corrected reflectivity [dBZ]
b)
PyDDA retreived winds @3.05 km
Longitude [°]
Latitude [°]
Corrected reflectivity [dBZ]

Figure 2: (a) An example streamline plot overlaid on radar reflectivity at 3 km altitude observed from 2 radars over Darwin, Australia. Contours represent horizontal wind speed. **(b)** as **(a)**, but zoomed into the region enclosed by the grey box. Contours represent vertical wind speed.

Support

Documentation

Documentation is provided at http://openradarscience.org/PyDDA. It is automatically updated by Travis CI when pull request to the master branch is approved on GitHub. Sphinx automatically generates documentation from the docstrings at the top of each procedure that are written in reStructuredText. Travis CI, using doctr, will run Sphinx every time a pull request to the master branch is approved in order to update the documentation. Therefore, contributors must provide documentation in docstrings at the top of their procedures. Three examples on how to use PyDDA are provided in the documentation. Any user can download and run them to check if the PyDDA is working.

Issues

Developing wind retrievals from radars is typically a complicated task. Therefore, PyDDA has support methods available for users. The primary method of support is through PyDDA's issue tracker on GitHub. On the issue tracker, users are encouraged to ask questions as well as point out potential bugs in PyDDA. In addition, users can provide input for features they would like to see as well as develop their own contributions. The PyDDA developers will examine each issue and provide support. If the issue is related to a bug in PyDDA, either the developer or the contributor will work on resolving the bug.

(2) Availability

Operating system

Windows, Mac OS X, and Linux

Programming language

Python 3.6+

Additional system requirements

We recommend:

1. 8+ GB RAM
2. 1 GB+ of hard drive space.
3. An Intel x86 compatible CPU with 4+ cores.

Dependencies
NumPy, SciPy, matplotlib, Py-ART, cartopy are required. cfgrib is needed for reading HRRR data, but is currently optional as cfgrib does not work on Windows.

List of contributors
- Robert Jackson, Argonne National Laboratory
- Scott Collis, Argonne National Laboratory
- Todd Munson, Argonne National Laboratory
- Zach Sherman, Argonne National Laboratory
- Timothy Lang, NASA Marshall Space Flight Center,
- Corey Potvin, Cooperative Institute for Mesoscale Meteorological Studies/University of Oklahoma

Software location
Archive http://openradarscience.org/PyDDA
Name: PyDDA
Persistent identifier: DOI: 10.5281/zenodo.3942686
Licence: BSD 3-clause license
Publisher: Robert Jackson
Version published: 0.5.2
Date published: 07/13/2020

Code repository GitHub
Name: PyDDA
Persistent identifier: https://github.com/openradar/PyDDA
Licence: BSD 3-clause license
Date published: 07/13/2020

Language
English

(3) Reuse potential
PyDDA has many applications within meteorology as in fields such as air quality modelling and civil engineering. As of the writing of this article, the authors are collaborating with Dan Chavez at Purdue University to use PyDDA's wind fields to create databases of 3D winds in Hurricane Florence and Michael for improving hurricane prediction. In a project funded by the Department of Energy's Office of Energy Efficiency & Renewable Energy, PyDDA will produce wind fields for assessing the effects of damaging winds on the electric grid. In addition, 3D wind retrievals are vital to understanding thunderstorm dynamics so an open source solution for such retrievals has great potential to be used by the meteorology community.

A synergy of radar observations and models in Hurricane Florence
Using PyDDA, combining observations from high resolution forecasting models and radar observations to create a more complete picture of the 3D wind field inside a hurricane is simple. For this example, we will show how the winds inside Hurricane Florence can be retrieved using PyDDA and show how using radar and model data together demonstrates the improvement in capabilities that PyDDA provides compared to previous software.

On 14 September 2018, Hurricane Florence was within range of 2 radars from the NEXRAD network: KMHX stationed in Newport, NC and KLTX stationed in Wilmington, NC. Both radars provided continuous 3D observations of radial velocities inside Hurricane Florence as it made landfall over North Carolina. In addition, the High Resolution Rapid Refresh (HRRR) model was run at every hour, providing model reanalysis of observational data at an hourly temporal resolution and a 3.7 km spatial resolution. The archived HRRR data that is supported by PyDDA can be downloaded from the University of Utah's HRRR archive [22, 23].

As one can see, in **Figure 3** we have an incomplete picture of the wind field inside Hurricane Florence as the 2 NEXRAD radars only provide partial coverage of the winds at 0.5 km altitude. Therefore, we also constrain

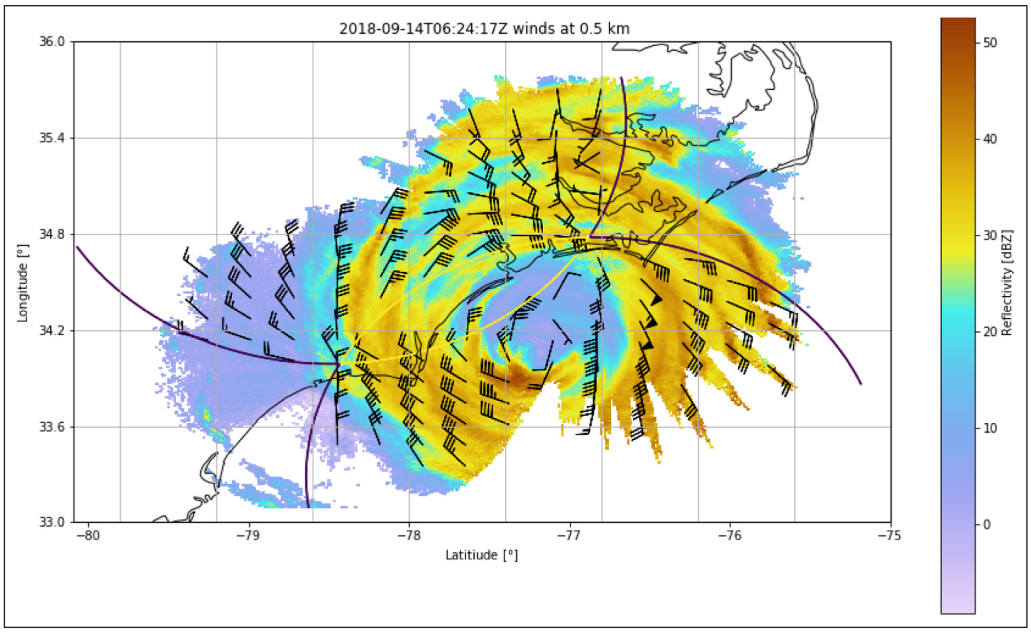

Figure 3: Winds at 0.5 km retrieved by PyDDA using only data from the 2 NEXRAD radars overlaid on a reflectivity mosaic generated from the two NEXRAD radars. Barbs are in $m\,s^{-1}$.

against the HRRR calculated winds in **Figure 4**, which gives us the retrieval in **Figure 5**. As you can see, a much more complete picture of Hurricane Florence is provided, showing that Eastern North Carolina is indeed having winds much higher than the 10 m s^{-1} inferred by **Figure 3** and also showing the cyclonic circulation that would be expected in a hurricane. This shows how the integration of data from multiple sources enhances the applicability and expandability of PyDDA compared to other software, as Multidop and CEDRIC did not support the integration of model data.

Expanding PyDDA

PyDDA was designed so that contributors can easily add cost functions and custom initializations to PyDDA. In particular, since Equation (1) is a sum of cost functions, adding terms to Equation (1) expands the framework. The `cost_functions` module contains all of the cost functions that are used by PyDDA. Expanding the code to incorporate more cost functions involves implementing the cost function, its gradient, and then adding the cost function to the `J_function` in the `cost_functions` module and the gradient to `grad_J` in

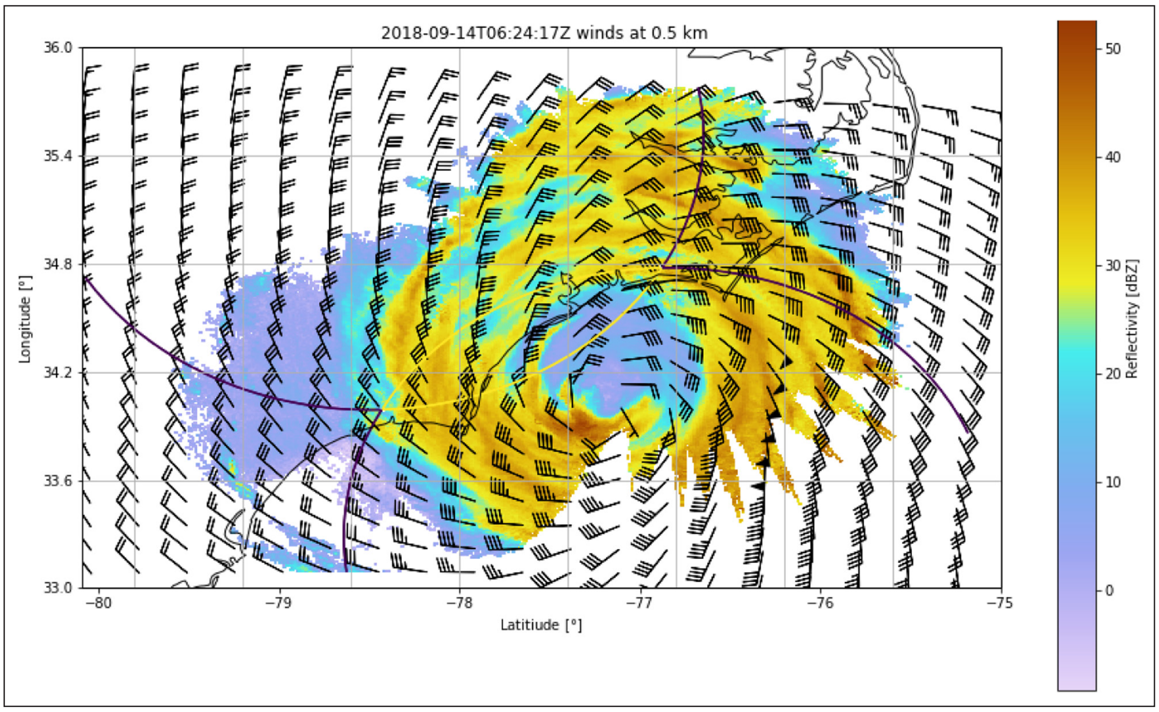

Figure 4: As Figure 3, but only using the HRRR model as a constraint.

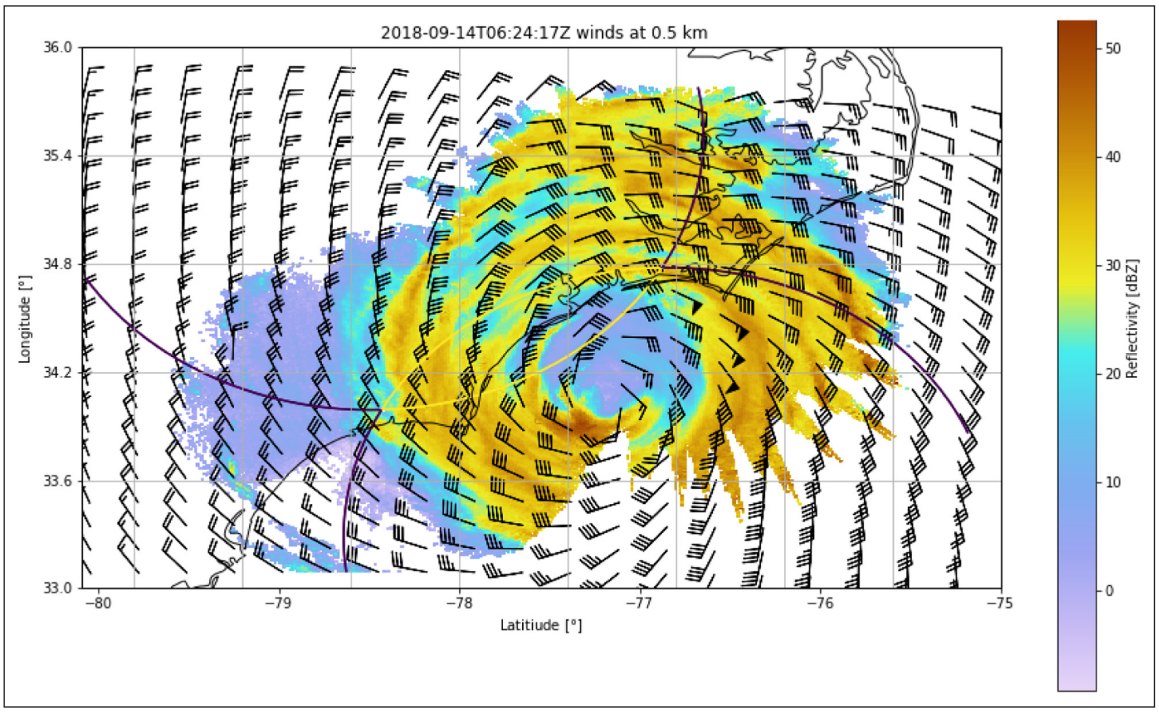

Figure 5: As Figure 3, but using both the radar and HRRR winds as constraints.

the `cost_functions` module. Since a cost function will typically be the integral of a function over a grid, commonly called a functional, the gradients are derived by taking the functional derivative of the cost function. For more information on functional derivatives, see [24]. See [10] for formulas for the gradients to the mass continuity and radar observational cost functions currently used in PyDDA. Since the functional derivatives and the cost functions have explicit forms, they are all implemented into PyDDA using NumPy [14] and SciPy [15].

The `constraints` module can be expanded to include more functions that interpolate data from various datasets to the analysis grid. Currently, it has two examples from HRRR and WRF data. The `get_dd_wind_field` already is able to take in any number of models with any field name, so any function that interpolates a dataset can be added to the `constraints` module.

Another way that PyDDA can be expanded is by adding in custom initializations to the 3DVAR technique. Since Py-DDA takes in a 3-tuple of NumPy arrays representing **v** with the same shape as the analysis grid as an initialization, any function that returns this 3-tuple can be added to the `initialization` module. Example initializations consisting of a constant wind field, a sounding, and HRRR and WRF outputs are already in PyDDA.

The PyDDA development team is currently seeking collaborators for expanding the functionality of PyDDA. If a user needs help in expanding PyDDA to include their own sensor or model, they are encouraged to contact the main developer currently supporting PyDDA, Robert Jackson, at rjackson@anl.gov for collaboration on this effort. There is a contributor's guide in the documentation that shows the code style, code of conduct, license, and documentation standards that must be followed for each contribution. In addition, we have published a roadmap of the eventual goals of PyDDA, available at https://github.com/openradar/PyDDA/blob/master/ROADMAP.md that show what contributions to PyDDA are useful.

Acknowledgements
We would like to thank Alain Protat and Valentin Louf for processing the C-band Polarization Radar data. We would also like to thank Zach Sherman for reorganizing the software for proper packaging.

Competing Interests
The authors have no competing interests to declare.

References

1. **Kosiba, K, Wurman, J, Richardson, Y, Markowski, P, Robinson, P** and **Marquis, J** 2013 Genesis of the Goshen County, Wyoming, Tornado on 5 June 2009 during VORTEX2. *Monthly Weather Review*, 141: 1157–1181. DOI: https://doi.org/10.1175/MWR-D-12-00056.1
2. **Kosiba, K A** and **Wurman, J** 2014 Finescale Dual-Doppler Analysis of Hurricane Boundary Layer Structures in Hurricane Frances (2004) at Landfall. *Monthly Weather Review*, 142: 1874–1891. DOI: https://doi.org/10.1175/MWR-D-13-00178.1
3. **Fast, J D, Newsom, R K, Allwine, K J, Xu, Q, Zhang, P, Copeland, J** and **Sun, J** 2008 An Evaluation of Two NEXRAD Wind Retrieval Methodologies and Their Use in Atmospheric Dispersion Models. *J. Appl. Meteor. Climatol*, 47: 2351–2371. DOI: https://doi.org/10.1175/1520-0477(1993)074<1669:TWATWO>2.0.CO;2
4. **Crum, T D** and **Alberty, R L** 1993 The WSR-88D and the WSR-88D Operational Support Facility. *Bull. Amer. Meteor. Soc.*, 74: 1669–1688, DOI: https://doi.org/10.1175/1520-0477(1993)074¡1669:TWATWO¿2.0.CO;2
5. **NOAA National Weather Service (NWS) Radar Operations Center** 1991 *NOAA Next Generation Radar (NEXRAD) Level 2 Base Data*. [KLTX, KMHX reflectivity, Doppler velocity]. NOAA National Centers for Environmental Information. DOI: https://doi.org/10.7289/V5W9574V [30 Oct 2018].
6. **Gal-Chen, T** 1978 A Method for the Initialization of the Anelastic Equations: Implications for Matching Models with Observations. *Mon. Wea. Rev.*, 106: 587–606. DOI: https://doi.org/10.1175/1520-0493(1978)106<0587:AMFTIO>2.0.CO;2
7. **Gal-Chen, T** and **Kropfli, R A** 1984 Buoyancy and Pressure Perturbations Derived from Dual-Doppler Radar Observations of the Planetary Boundary Layer: Applications for Matching Models with Observations. *J. Atmos. Sci.*, 41: 3007–3020. DOI: https://doi.org/10.1175/1520-0469(1984)041<3007:BAPPDF>2.0.CO;2
8. **Lorenc, A C** 1986 Analysis methods for numerical weather prediction. *Quart. J. Roy. Meteor. Soc.*, 112: 1177–1194. DOI: https://doi.org/10.1002/qj.49711247414
9. **Shapiro, A, Potvin, C K** and **Gao, J** 2009 Use of a vertical vorticity equation in variational dual-Doppler wind analysis. *J. Atmos. Oceanic Technol.*, 26: 2089–2106. DOI: https://doi.org/10.1175/2009JTECHA1256.1
10. **Gao, J, Xue, M, Shapiro, A** and **Droegemeier, K K** 1999 A Variational Method for the Analysis of Three-Dimensional Wind Fields from Two Doppler Radars. *Mon. Wea. Rev.*, 127: 2128–2142. DOI: https://doi.org/10.1175/1520-0493(1999)127<2128:AVMFTA>2.0.CO;2
11. **Potvin, C K, Shapiro, A** and **Xue, M** 2012 Impact of a Vertical Vorticity Constraint in Variational Dual-Doppler Wind Analysis: Tests with Real and Simulated Supercell Data. *J. Atmos. Oceanic Technol.*, 29: 32–49. DOI: https://doi.org/10.1175/JTECH-D-11-00019.1
12. **Potvin, C K, Betten, D, Wicker, L J, Elmore, K L** and **Biggerstaff, M I** 2012 3DVAR versus Traditional Dual-Doppler Wind Retrievals of a Simulated Supercell Thunderstorm. *Mon. Wea. Rev.*, 140: 3487–3494. DOI: https://doi.org/10.1175/MWR-D-12-00063.1
13. **Miller, L J** and **Fredrick, S M** 1998 Custom Editing and Display of Reduced Information in Cartesian space (CEDRIC) manual. *National Center for Atmospheric Research, Mesoscale and Microscale Meteorology Division.* Boulder, CO, 1–130.

14. **Lang, T, Souto, M, Khobahi, S** and **Jackson, B** 2017 nasa/MultiDop: MultiDop v0.3. *Version v0.3. Zenodo*. DOI: https://doi.org/10.5281/zenodo.1035904
15. **Oliphant, T E** 2006 *A guide to NumPy*. USA: Trelgol Publishing.
16. **Oliphant, T E, Peterson, P,** et al. 2001 *SciPy: Open Source Scientific Tools for Python*. Available at: http://www.scipy.org/ [Online; accessed 2018-10-19].
17. **Met Office** 2015 *Cartopy: a cartographic python library with a matplotlib interface*. Exeter, Devon. Available at: http://scitools.org.uk/cartopy.
18. **Hunter, J D** 2007 Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3): 90–95. DOI: https://doi.org/10.1109/MCSE.2007.55
19. **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(1): e25. DOI: https://doi.org/10.5334/jors.119
20. **Liu, D C** and **Nocedal, J** 1989 On the Limited Memory Method for Large Scale Optimization. *Mathematical Programming B*, 45(3): 503–528. DOI: https://doi.org/10.1007/BF01589116
21. **Collis, S, Protat, A, May, P T** and **Williams, C** 2013 Statistics of Storm Updraft Velocities from TWP-ICE Including Verification with Profiling Measurements. *J. Appl. Meteor. Climatol.*, 52: 1909–1922. DOI: https://doi.org/10.1175/JAMC-D-12-0230.1
22. **Blaylock, B K, Horel, J D** and **Liston, S T** 2017 Cloud archiving and data mining of High-Resolution Rapid Refresh forecast model output. *Computers & Geosciences*, 109: 43–50. DOI: https://doi.org/10.1016/j.cageo.2017.08.005
23. **Horel, J** and **Blaylock, B** 2018 Archive of the High Resolution Rapid Refresh Model. *Dataset*. University of Utah. DOI: https://doi.org/10.7278/S5JQ0Z5B
24. **Roubicek, T, Calculus of variations** 2014 Chap.17 In: Grinfield, M (ed.), *Mathematical Tools for Physicists*, 551–588. Weinheim: J. Wiley.

How to cite this article: Jackson, R, Collis, S, Lang, T, Potvin, C and Munson, T 2020 PyDDA: A Pythonic Direct Data Assimilation Framework for Wind Retrievals. *Journal of Open Research Software,* 8: 20. DOI: https://doi.org/10.5334/jors.264

Submitted: 28 February 2019 **Accepted:** 16 September 2020 **Published:** 07 October 2020