Python Disdrometer Processing: From Prototyping to Library Development Using Open Source Tooling

Joseph C. Hardin, V. Chandrasekar

Colorado State University

jhardin@engr.colostate.edu

April 7th 2014
UCAR SEA 2014

Acknowledgements

I would like to thank:

- Haonan Chen for being the first test subject for a lot of this code(and a few bugfixes).
- Jussi Leinonen for developing the PyTMatrix Library
- Scott Collins and Jonathan Helmus for the Py-ART library(As well as dealing with a deluge of questions I always seem to have).

The Question

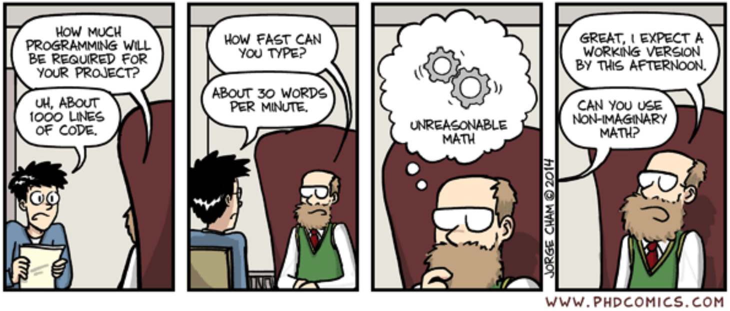
How do you create scientific software for scientists...without any "developers"

You try to turn scientists into developers.

UH, I'M NOT A VERY GOOD PROGRAMMER, PROF. SMITH.
THAT'S OK.
PART OF GETTING YOUR PH.D. IS LEARNING TO LEARN WHATEVER YOU NEED TO LEARN TO MAKE YOUR RESEARCH WORK.
IT'S ABOUT STEPPING OUT OF YOUR COMFORT ZONE AND LEARNING NEW SKILLS!
ALSO, YOU'RE CHEAPER THAN HIRING A REAL PROGRAMMER.
SO I'VE LEARNED.
JORGE CHAM © 2014
WWW.PHDCOMICS.COM

Graduate Students

Additionally, what do you do when you have unreal constraints on your time(As all scientists and engineers do!)


HOW MUCH PROGRAMMING WILL BE REQUIRED FOR YOUR PROJECT?
UH, ABOUT 1000 LINES OF CODE.
HOW FAST CAN YOU TYPE?
ABOUT 30 WORDS PER MINUTE.
UNREASONABLE MATH
JORGE CHAM © 2014
GREAT, I EXPECT A WORKING VERSION BY THIS AFTERNOON.
CAN YOU USE NON-IMAGINARY MATH?
WWW.PHDCOMICS.COM

You start trying to reuse available software, and code in such a way that you get the most results for your time!

What is the Point of This Talk?

As part of my research, I recently found myself needing to do some analysis on disdrometer data. I ran into a few problems along the way.

- Very little existing code
- You have to e-mail to get that code.
- To process the code you use a different language than reading it.

I wanted to make the work I was doing usable to others.

This talk focuses on the script to library process, and the tools that help. I'm not an expert, but these are some of the things I've learned from participating in a few libraries.

Disdrometer

A disdrometer is a ground instrument that measures how many raindrops are falling and their sizes and velocities. This gives us a drop size distribution.

Figure : (a) Parsivel Disdrometer , (b) 2D-Video Disdrometer

Drop Size Distribution

The disdrometer counts how many drops are in each size bin. We can derive rainfall, reflectivities, and other parameters.

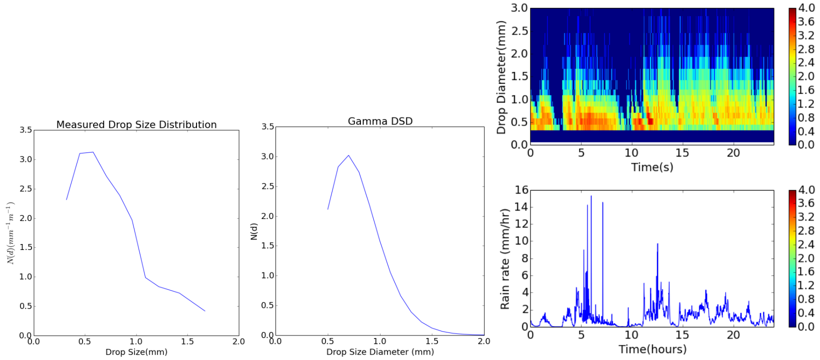


Figure : (a) Measured DSD (b) Analytical DSD (c) Measured DSD and rain rate for a day

Uses for Disdrometers

What do we use disdrometers for? Quite a lot actually. This includes:

- Rainfall Rate Measurement
- Radar Verification
- Microphysical Estimation
- Hydrology
- Drop Morphology

During the NASA GPM campaigns, a large number(25+) of disdrometers are deployed to investigate the nature of rainfall and test radar estimation of rainfall.

Radar Verification

Given a drop size distribution $N(D)$ we can make a few assumptions, and calculate what a radar would see looking at these variables using the T-Matrix method.

$$\zeta = \frac{\lambda^4}{\pi^5 |K_p|^2} \int_D \sigma_b(D) N(D) dD \tag{1}$$

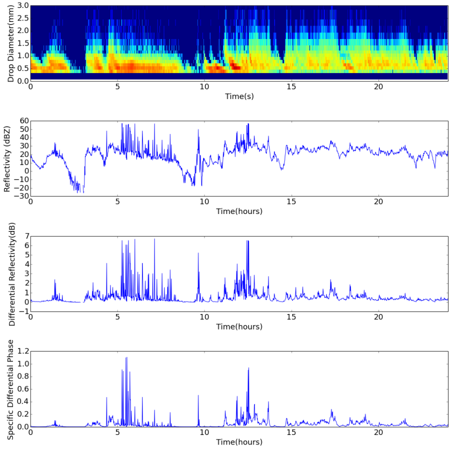
$$\zeta_{DR} = \frac{\int_D |S_{hh}(r,D)|^2 N(D) dD}{\int_D |S_{vv}(r,D)|^2 N(D) dD} \tag{2}$$

$$K_{dp} = \frac{2\pi}{k_0} \Re \int_D N(D) \left[\hat{h} \cdot \vec{f}(r,D) - \hat{v} \cdot \vec{f}(r,D) \right] dD \tag{3}$$

We can then use these simulated parameters to figure out the relationship between rain rate and the radar measured parameters, Although these are not the only useful parameters.

Radar Moments

As an example here we see a drop size distribution and it's scattered radar parameters over the course of a day.



So what's the problem?

There are several problems today when working with disdrometer data. A few of these are:

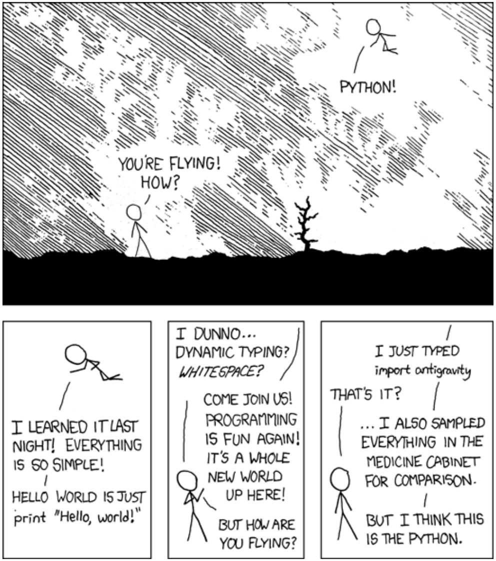
- Multiple Different Data Formats, for even the same disdrometer.
    - Some organizations preprocess data
    - Each vendor has it's own format
    - Each format provides different information
- Multiple languages are needed for the toolchain
    - Scattering in Fortran
    - File Parsing in...IDL!
    - Radar data processed in Matlab
- Multiple tools must be strung together, often very fragile.

Why a Python Library?

We turn to Python to write a library for disdrometers for a few reasons.

1. Python can wrap other languages(In this case, Fortran and C)
2. Python is high level, and easy to write scientific code in.
3. Python is becoming the standard in atmospheric sciences.
4. The recent influx of useful Python Packages.
5. IPython Notebook
6. I was writing this code for my research..why not share it with the world in a usable form?

There is an XKCD for everything.


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

The rest of the talk

For the rest of this talk, we're going to discuss the workflow I went through to start the PyDisdrometer library. This will cover:

- Data Exploration
- Algorithm Development
- Incorporating other libraries
- Developing a usable package(instead of a collection of scripts)
- Packaging
- Project Dissemination

The neat part about this is, every step along the way stayed within Python, and within fully open source tools. This means no learning multiple languages, and that we can publish the entire workflow so that others can contribute.

Ipython Notebook: Data Exploration

The first step in any kind of close to data library development is to actually work with the data. We are going to cover a very interesting tool here that makes this kind of exploration very natural, the IPython Notebook.
Features:

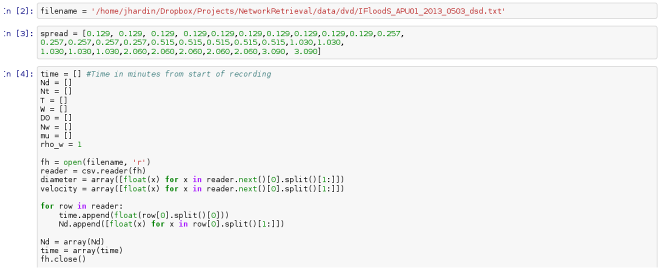
- Web Interface to Python Shell
- Inline Plotting
- Amazing Collaboration Tool
- Reproducible Research

Ipython Notebook: Data Exploration Pt. 2

So we start by loading a few example datasets, plotting them and just generally examining what they contain.

Algorithm Development

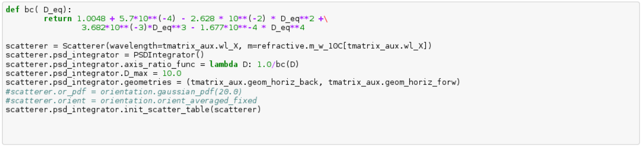
Next we need to we start writing code to solve our problems. In my case this means processing the data, generating radar parameters etc.
After some digging through file specifications we finally get a simple set of code that can read a parsivel file.



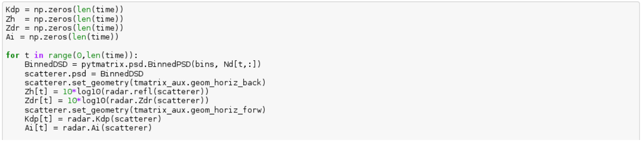
```
:n [2]: filename = '/home/jhardin/Dropbox/Projects/NetworkRetrieval/data/dvd/IFloodS_APU01_2013_0503_dsd.txt'

:n [3]: spread = [0.129, 0.129, 0.129, 0.129,0.129,0.129,0.129,0.129,0.129,0.129,0.257,
        0.257,0.257,0.257,0.257,0.515,0.515,0.515,0.515,0.515,1.030,1.030,
        1.030,1.030,1.030,2.060,2.060,2.060,2.060,2.060,3.090, 3.090]

:n [4]: time = [] #Time in minutes from start of recording
        Nd = []
        Nt = []
        T = []
        W = []
        D0 = []
        Nw = []
        mu = []
        rho_w = 1

        fh = open(filename, 'r')
        reader = csv.reader(fh)
        diameter = array([float(x) for x in reader.next()[0].split()[1:]])
        velocity = array([float(x) for x in reader.next()[0].split()[1:]])

        for row in reader:
            time.append(float(row[0].split()[0]))
            Nd.append([float(x) for x in row[0].split()[1:]])

        Nd = array(Nd)
        time = array(time)
        fh.close()
```

Supporting Modules: PyTMatrix

PyTMatrix is a Python library written by Jussi Leinonen that wraps T-Matrix scattering code written in Fortran by Mishenko, and allows very easy access from Python. We will use this to convert a DSD to radar measured parameters.
https://github.com/jleinonen/pytmatrix



```
def bc( D_eq):
        return 1.0048 + 5.7*10**(-4) - 2.628 * 10**(-2) * D_eq**2 +\
                3.682*10**(-3)*D_eq**3 - 1.677*10**-4 * D_eq**4

scatterer = Scatterer(wavelength=tmatrix_aux.wl_X, m=refractive.m_w_10C[tmatrix_aux.wl_X])
scatterer.psd_integrator = PSDIntegrator()
scatterer.psd_integrator.axis_ratio_func = lambda D: 1.0/bc(D)
scatterer.psd_integrator.D_max = 10.0
scatterer.psd_integrator.geometries = (tmatrix_aux.geom_horiz_back, tmatrix_aux.geom_horiz_forw)
#scatterer.or_pdf = orientation.gaussian_pdf(20.0)
#scatterer.orient = orientation.orient_averaged_fixed
scatterer.psd_integrator.init_scatter_table(scatterer)
```



```
Kdp = np.zeros(len(time))
Zh  = np.zeros(len(time))
Zdr = np.zeros(len(time))
Ai = np.zeros(len(time))

for t in range(0,len(time)):
    BinnedDSD = pytmatrix.psd.BinnedPSD(bins, Nd[t,:])
    scatterer.psd = BinnedDSD
    scatterer.set_geometry(tmatrix_aux.geom_horiz_back)
    Zh[t] = 10*log10(radar.refl(scatterer))
    Zdr[t] = 10*log10(radar.Zdr(scatterer))
    scatterer.set_geometry(tmatrix_aux.geom_horiz_forw)
    Kdp[t] = radar.Kdp(scatterer)
    Ai[t] = radar.Ai(scatterer)
```

Turning Scripts into a Public Library

At this point we've been playing with code and doing a lot of analysis for papers and have a collection of scripts.
We need to take these scripts that are a little specific and generalize them and prepare them to be used by others. This means:

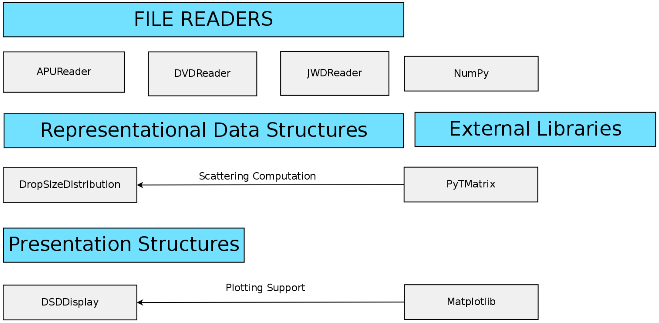
- Making sure code works together.
- Packaging the code up into classes
- Make the code easy to deploy
- Testing the code on other computers/platforms/architectures
- Making the code easy to interface with
- Making sure our unit tests are comprehensive(You do write unit tests right?)

Transitioning the Codebase to a Library: Generating the API

One of the best ways to make your code easy to interface with is to encapsulate it into classes that work well with each other. In our case this means we broke the code up as follows:

- Each filetype gets a class that knows how to read it. This should only require a filename.
- These readers return a drop size distribution object. This object forms the core of the library.
- This object knows how to run scattering computations on itself and some basic processing.
- Additional code operates on this dsd object.

General Layout



Make code "Just Work"

A few things can be done to make the library more intuitive.

- Choose sane defaults.
- Require as little effort to use as possible.
- Automate common workflows.

For instance, the scattering computations require a drop shape relationship, a wavelength, and a temperature. If the user does not explicitly provide these, we automatically set:

1. DSR(Beard and Chuang)
2. Frequency(X-Band)
3. Temperature(10C)

These are all user customizable, but by providing sane defaults, we allow a user to get up to speed with the library very quickly, and then refine things as they learn more.

Distributing the Code: Github

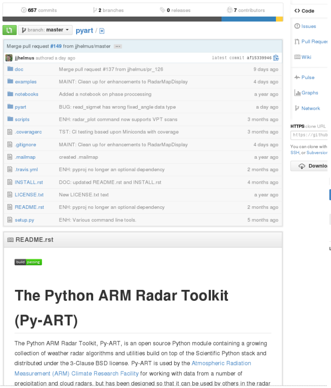
It is important to:

- Minimize effort needed to contribute
- Version Control
- Bug Tracking
- Automatically download code
- Social is nice

We have chosen to use GitHub. This lets us take our existing version controlled repository, and put it online for the world to see.

Github pt2

An example from the Pyart library



657 commits · 2 branches · 0 releases · 7 contributors

branch: master · pyart /

Merge pull request #149 from jjhelmus/master

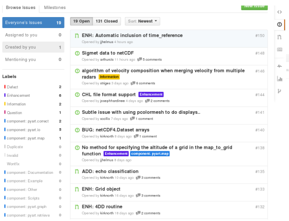
jjhelmus authored a day ago · latest commit af15339946

doc	Merge pull request #137 from jjhelmus/pr_126	9 days ago
examples	MAINT: Clean up for enhancements to RadarMapDisplay	4 days ago
notebooks	Added a notebook on phase proccessing	a year ago
pyart	BUG: read_sigmet has wrong fixed_angle data type	a day ago
scripts	ENH: radar_plot command now supports VPT scans	3 months ago
.coveragerc	TST: CI testing based upon Miniconda with coverage	3 months ago
.gitignore	MAINT: Clean up for enhancements to RadarMapDisplay	4 days ago
.mailmap	created .mailmap	a year ago
.travis.yml	ENH: pyproj no longer an optional dependency	2 months ago
INSTALL.rst	DOC: updated README.rst and INSTALL.rst	4 months ago
LICENSE.txt	New LICENSE.txt text	a year ago
README.rst	ENH: pyproj no longer an optional dependency	2 months ago
setup.py	ENH: Various command line tools.	5 months ago

README.rst

The Python ARM Radar Toolkit (Py-ART)

The Python ARM Radar Toolkit, Py-ART, is an open source Python module containing a growing collection of weather radar algorithms and utilities build on top of the Scientific Python stack and distributed under the 3-Clause BSD license. Py-ART is used by the Atmospheric Radiation Measurement (ARM) Climate Research Facility for working with data from a number of precipitation and cloud radars, but has been designed so that it can be used by others in the radar



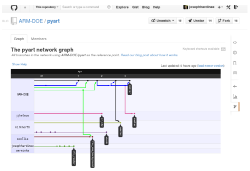
Everyone's Issues	19
Assigned to you	0
Created by you	1
Mentioning you	0

19 Open · 131 Closed

Labels

Defect	2
Enhancement	6
Information	2
Question	1
component: pyart.correct	2
component: pyart.io	5
component: pyart.map	1
Duplicate	0
Invalid	0
Wontfix	0
component: Documentation	0
component: Example	0
component: Other	0
component: Scripts	0
component: pyart.graph	0
component: pyart.retrieve	0

- ENH: Automatic inclusion of time_reference — Opened by jjhelmus 4 hours ago — #150
- Sigmet data to netCDF — Opened by arthurcts 11 hours ago — 5 comments — #148
- algorithm of velocity composition when merging velocity from multiple radars [Information] — Opened by striges 3 days ago — 6 comments — #146
- CHL file format support [Enhancement] — Opened by josephhardinee 4 days ago — 2 comments — #144
- Subtle issue with using pcolormesh to do displays.. — Opened by scollis 7 days ago — 1 comment — #141
- BUG: netCDF4.Dataset arrays — Opened by kirknorth 8 days ago — 1 comment — #140
- No method for specifying the altitude of a grid in the map_to_grid function [Enhancement] [component: pyart.map] — Opened by jjhelmus 8 days ago — #138
- ADD: echo classification — Opened by kirknorth 10 days ago — 3 comments — #135
- ENH: Grid object — Opened by kirknorth 16 days ago — 3 comments — #133
- ENH: 4DD routine — Opened by kirknorth 18 days ago — 2 comments — #132



The pyart network graph

All branches in the network using ARM-DOE/pyart as the reference point. Read our blog post about how it works.

Last updated: 9 hours ago (load newer version)

Packaging Python Code

To get our code to users we have two methods:

1. Download from Github
2. Use Python Packaging Program(PIP)

You add some information about your package, then python will list your code on PyPI. To install code users just do

```
pip install pydisdrometer
```

Python then installs all the necessary dependencies.

Using PIP

The boilerplate to use pip for simple projects is quite easy.

```
from distutils.core import setup

setup(
    name='PyDisdrometer',
    version='0.1.0',
    author='Joseph C. Hardin',
    author_email='josephhardinee@gmail.com',
    packages=['pydisdrometer'],
    url='http://pypi.python.org/pypi/PyDisdrometer/',
    license='LICENSE.txt',
    description='Pyton Disdrometer Processing',
    long_description=open('README.txt').read(),
)
```

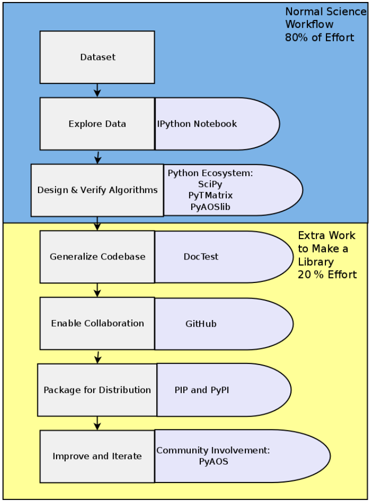
PyAOS and AMS

The American Meteorological Society recognized the usefulness of Python early on and started a Python in the Atmospheric Sciences track at the general meeting.
In addition, a PyAOS community has come together and started writing large amounts of code in Python that makes end to end development much more enjoyable. To list a few of the projects that I use in particular:

- PyArt(Python Radar toolkit from the ARM Project)
- PyTMatrix(Python T-Matrix Scattering)
- PyAOSlib: Toolkit of many different atmospheric science routines

Quick Summary of Tooling

So to recap on the process.


Normal Science Workflow 80% of Effort
Dataset
Explore Data
IPython Notebook
Design & Verify Algorithms
Python Ecosystem: SciPy PyTMatrix PyAOSlib
Extra Work to Make a Library 20 % Effort
Generalize Codebase
DocTest
Enable Collaboration
GitHub
Package for Distribution
PIP and PyPI
Improve and Iterate
Community Involvement: PyAOS

Demonstrating the Use of the Library

A simple(and common) use case. We want the $R(Z_H)$ relationship for a precipitation case from a parsivel disdrometer.

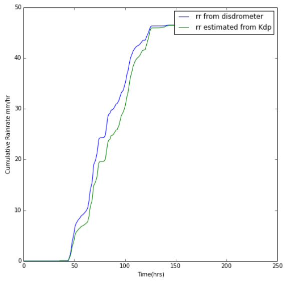
```
import pydisdrometer
dsd = read_parsivel(filename)
dsd.calc_radar_parameters()
relation, covariance = dsd.calc_R_Zh_relationship()
print(relation)
```

```
    (array([ 0.2112311,  0.4427024 ])
```

Demonstrating the Use of the Library Pt. 2

Let's estimate rainfall based on K_{DP} and look at how well the estimator fits.

```
rel_kd, covariance = dsd.calc_R_Kdp_relationship()
plot(dsd.time/360.0, np.cumsum(dsd.rain_rate)/360.0)
plot(dsd.time/360.0,
    np.cumsum(rel_kd[0]*(np.power(dsd.Kdp,rel_kd[1])))/360
```



What's next?

This library is very new, so there is a lot to add still.

- Method of Moments estimators for gamma parameters
- Integration with Pyart to do radar verification
- More disdrometer filetypes
- Statistical Analysis

How can you contribute?

How can you contribute to this package, or Python in the atmospheric sciences in general?

- Fork this project, contribute back code
- Use this library, submit bug reports
- Contribute to supporting packages

Questions?

Questions?