

Lidar Radar Open Software Environment LROSE

and

the Python ARM Radar Toolkit Py-ART

Joe VanAndel and Mike Dixon
Earth Observing Laboratory (EOL)
National Center for Atmospheric Research (NCAR)

Scott Collis and Jonathan Helmus
Environmental Sciences Division (EVS)
Argonne National Laboratory



Radar 101: What does the data look like?

RAdio Detection And Ranging

- Radars are a type of remote sensor, put simply they emit a discrete pulse of radiation and gate the receiver in order spatially “range” returns.
- *Scanning* radars also move the antenna (dish) in elevation and azimuth in order to scan out a volume of space. So each **ray** data has an associated azimuth and elevation and each **gate** within the **ray** has an associated range.
- Wide range of binary formats available. Until recently difficult to read without (some times commercial) software. Evolving community standards.



Lidar 101: What does the data look like?

LIght Detection And Ranging

- Lidars are a type of remote sensor, put simply they emit a discrete pulse of radiation and gate the receiver in order spatially “range” returns.
- *Scanning* lidars also move the beam in elevation and azimuth in order to scan out a volume of space. So each **ray** data has an associated azimuth and elevation and each **gate** within the **ray** has an associated range.
- Wide range of binary formats available. Until recently difficult to read without (some times commercial) software. Evolving community standards.



HSRL & HCR



S-Pol-Ka and Profilers



Radar Software Challenge

- Large Variety of Instruments
- Different Platforms – Airborne, Mobile, Fixed
- Many different analysis/display tools
- Users want to use C++, Java, Fortran, Matlab, IDL, Numeric Python
- Lack of standardization of data formats.



Radar Software Challenge(2)

- Large code base, including aging legacy applications
- Scientific Community has needs not supported by current software
- Inherited data formats that are not optimal for scientific data exchange.



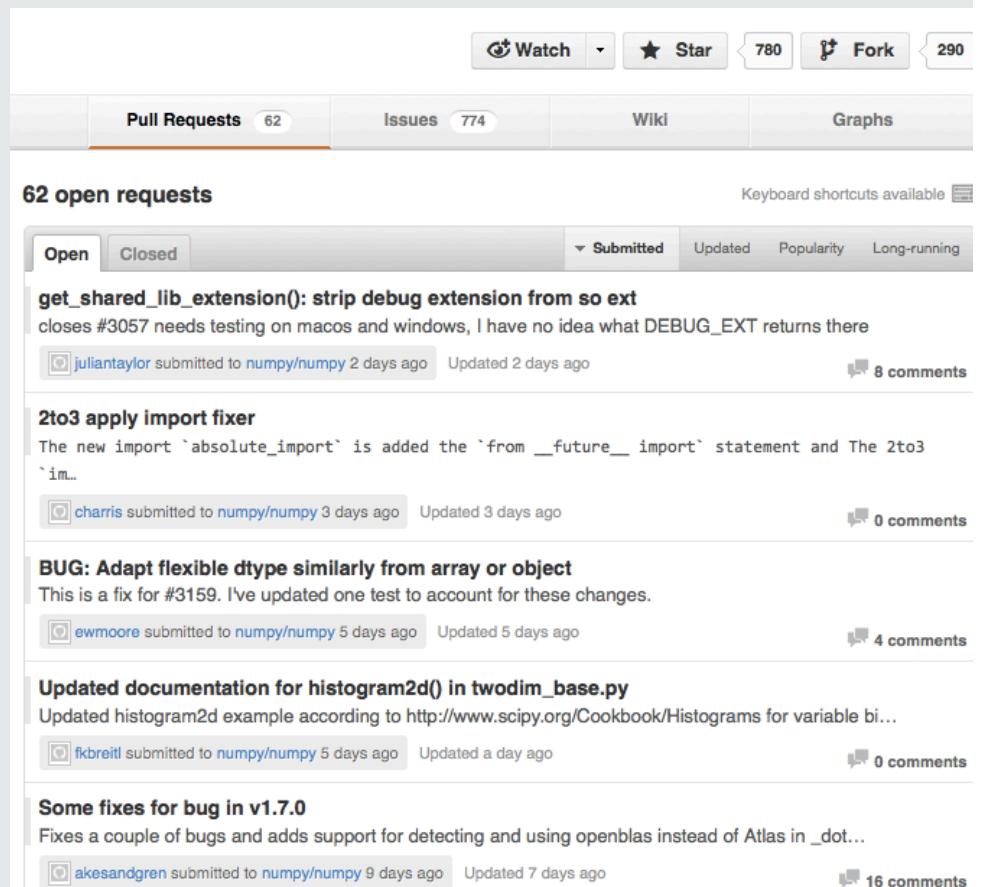
The open source community software paradigm

- Open source is exactly what it says, the source is open.. This is apart from “freeware”, open source goes further.. The source is open although there are various clauses on the use of the source and developers need to be aware of the differences and cross compatibility between various licenses..
- Community software goes even one step further.. By using various social coding platforms community projects can grow to large multi-contributor platforms.



Numerical python: A model open source community

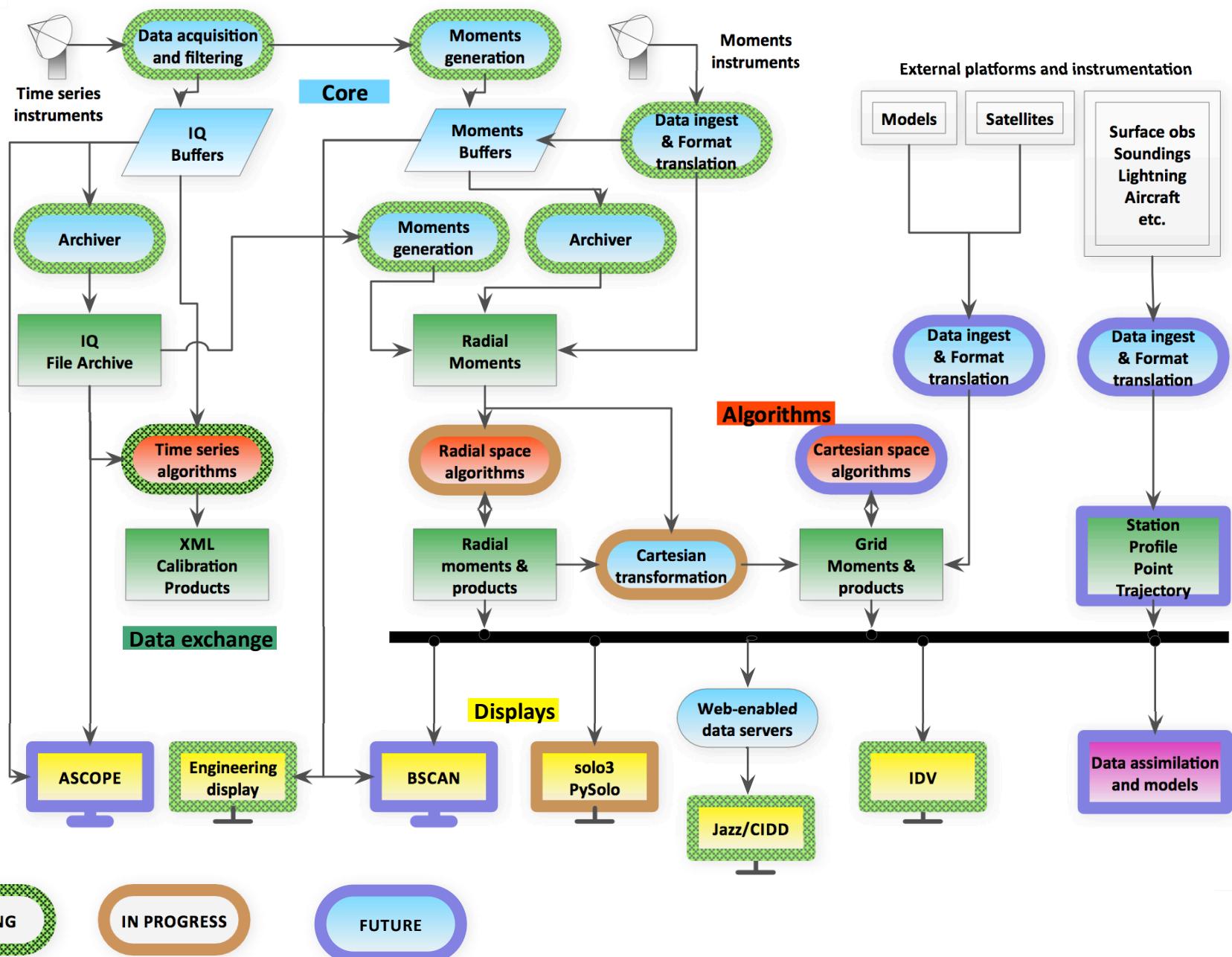
- Numerical Python, or Numpy, is a poster child for a community open source project.
- Originally conceived by Travis Oliphant it has progressed FAR beyond its initial concept...
- It has spawned companies, consultancies and organizations including iPython which has recently received funding from the Sloan foundation.
- As a group the atmospheric science must adopt a community project like this...



The screenshot shows the GitHub interface for the numpy repository. At the top, there are buttons for Watch (with a dropdown), Star, Fork, and 290 contributors. Below that is a navigation bar with Pull Requests (62), Issues (774), Wiki, and Graphs. The main area displays 62 open pull requests. The first few are listed:

- get_shared_lib_extension(): strip debug extension from so ext**
closes #3057 needs testing on macos and windows, I have no idea what DEBUG_EXT returns there
Submitted 2 days ago Updated 2 days ago by juliantaylor 8 comments
- 2to3 apply import fixer**
The new import `absolute_import` is added the `from __future__ import` statement and The 2to3 `im...
Submitted 3 days ago Updated 3 days ago by charris 0 comments
- BUG: Adapt flexible dtype similarly from array or object**
This is a fix for #3159. I've updated one test to account for these changes.
Submitted 5 days ago Updated 5 days ago by ewmoore 4 comments
- Updated documentation for histogram2d() in twodim_base.py**
Updated histogram2d example according to http://www.scipy.org/Cookbook/Histograms for variable bi...
Submitted 5 days ago Updated a day ago by fkbreitl 0 comments
- Some fixes for bug in v1.7.0**
Fixes a couple of bugs and adds support for detecting and using openblas instead of Atlas in _dot...
Submitted 9 days ago Updated 7 days ago by akesandgren 16 comments





LROSE

- CfRadial Data Exchange Format
- Radx C++ Libraries to read/write open file standards
- Algorithms & Analysis tools
- Core applications
- Display Tools – Visualization & Editing



CfRadial

- NetCDF using Climate and Forecasting conventions (CF)
- Readable by Linux, Windows, OS X.
- Readable from Radx C++ library, Matlab, IDL, Numeric Python
- Readable by weather models.



Radx library

Format	What/Who	Read Access	Write Access
CfRadial	NCAR/EOL/UNIDATA	Yes	Yes
DORADE	NCAR/EOL	Yes	Yes
UF	Universal Format	Yes	Yes
Foray-1	NCAR/EOL (Legacy)	Yes	Yes
DOE ARM netcdf	Precedes CfRadial	Yes	No
NEXRAD msg32	Level 2 Archive	Yes	Yes
NEXRAD msg1	Level 2 Archive	Yes	No
SIGMET –raw format	Vaisala (Sigmet)	Yes	No
RAPIC	BOM Australia	Yes	No
LEOSPHERE LIDAR	ASCII format	Yes	No



Displays

- Solo3 – C++
- Jazz – Java
- CIDD – C++
- VCHILL - Java
- IDV – Java
- ProfilerDisplay – IDL



Displays(2)

- Xprof – IDL
- Emerald – Matlab
- PySolo - Python



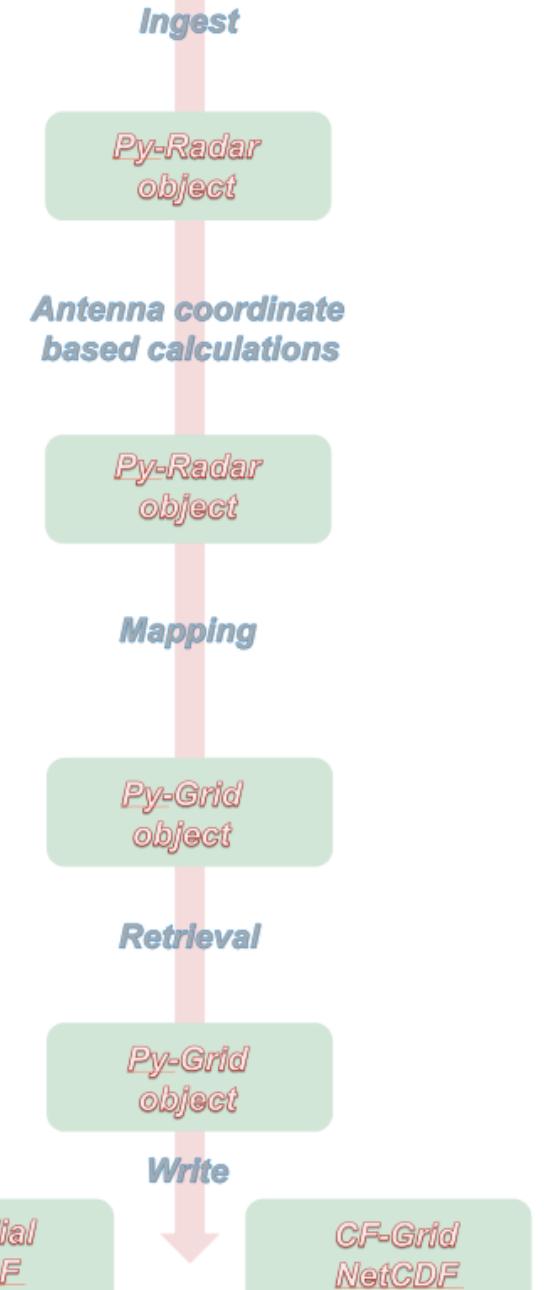
The Python-ARM Radar Toolkit: Py-ART

- ARM DoE has funded Argonne to produce a suite of radar products from the ARM network of X and C band radars.
- DoE also funds PIs to do science with the radar output.
- Py-ART was born out of the desire for a open flexible architecture to allow easy data access and to empower PIs to contribute.



Py-ART and the Py-Radar object

- So how best to represent the data at radar gates and associated metadata?
- Also need to align the object to a community standard format for easy mapping.
- NASA use a hierarchical memory object (volume -> sweeps -> rays) very flexible but poorly suited to array based operations.
- The Py-Radar object uses dictionaries for moments and metadata. Closely mirrors the CF-Radial format.
- All radial based methods in Py-ART (Python ARM Radar Toolkit) work on Py-Radar objects.



KISS Example, in the iPython Notebook.

```
In [2]: import netCDF4
basedir='/Users/scollis/local_modules/'
import sys
sys.path.append(basedir)
from pyart.io import radar
from pylab import *

rsl library found  /Users/scollis/local_modules/pyart/io/lib/librsl.dylib

In [3]: netcdf_object=netCDF4.Dataset('/data/sgpcsaprsurcmacI7.c0.20110520.095801.nc')
myradar=Radar(netcdf_object)

...

```

```
In [4]: print myradar.fields.keys()

[u'corrected_reflectivity_horizontal', u'reflectivity_horizontal', u'recalculated_diff_phase', u'specific_attenuation',
u'unf_dp_phase_shift', u'mean_doppler_velocity', u'diff_phase', u'norm_coherent_power', u'dp_phase_shift',
u'doppler_spectral_width', u'diff_reflectivity', u'proc_dp_phase_shift', u'copol_coeff']

In [5]: print myradar.fields['corrected_reflectivity_horizontal']['data'].shape

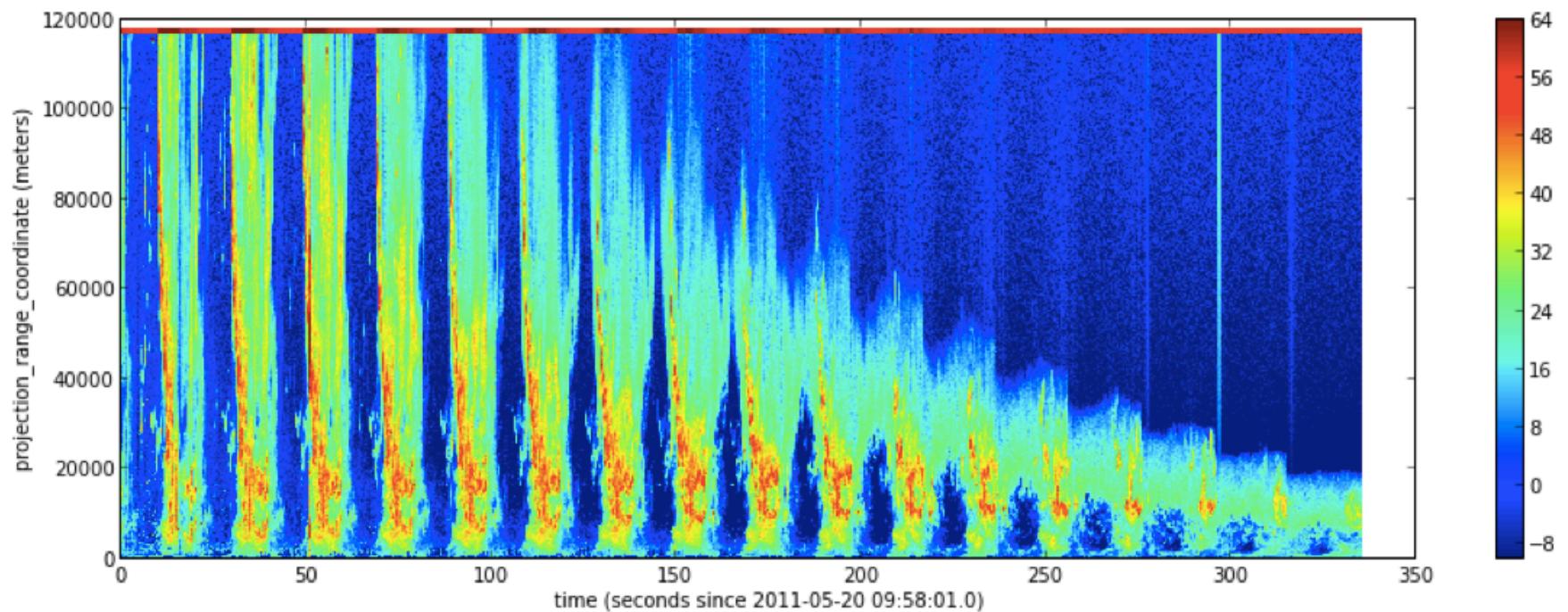
(6120, 983)
```



KISS Example, in the iPython Notebook.

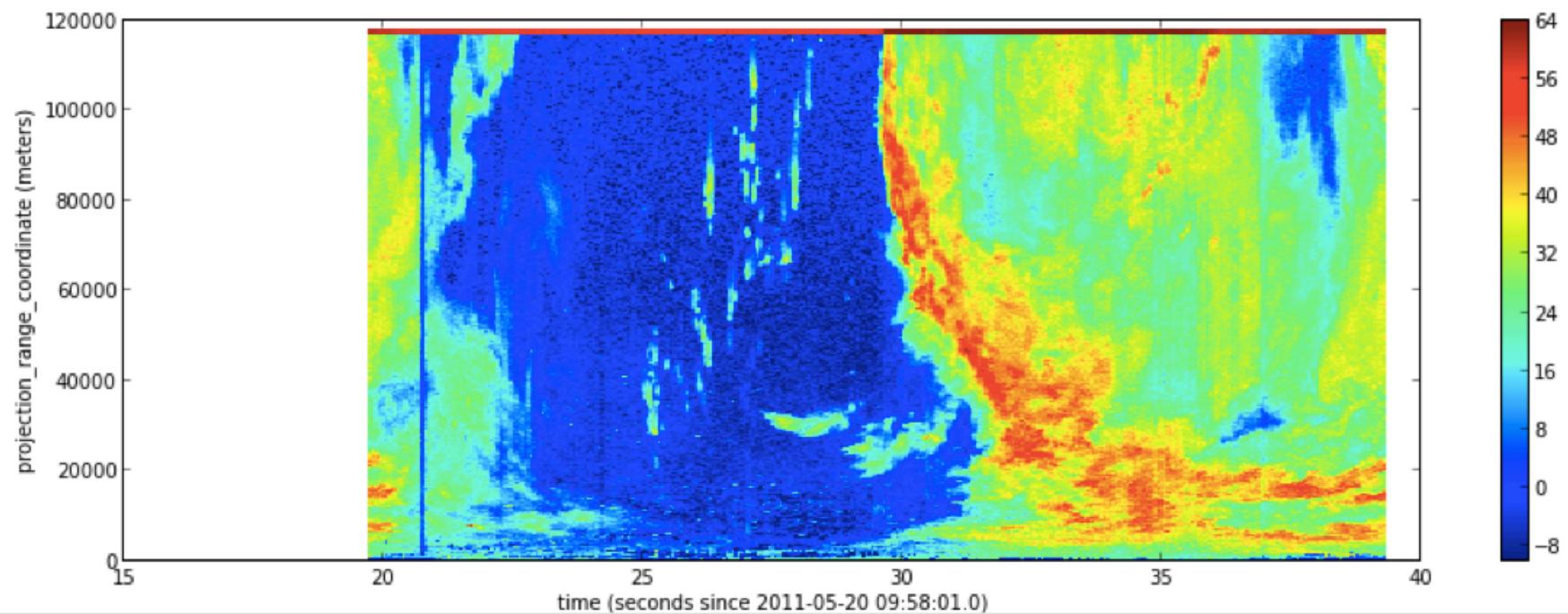
```
In [12]: figure(figsize=[15,5])
pcolormesh(myradar.time['data'], myradar.range['data'],
    myradar.fields['corrected_reflectivity_horizontal']['data'].transpose(),
    vmin=-10, vmax=64)
xlabel(myradar.time['standard_name']+ ('+'+myradar.time['units']+')'))
ylabel(myradar.range['standard_name']+ ('+'+myradar.range['units']+')'))
colorbar()
```

```
Out[12]: <matplotlib.colorbar.Colorbar instance at 0x10a5f07a0>
```



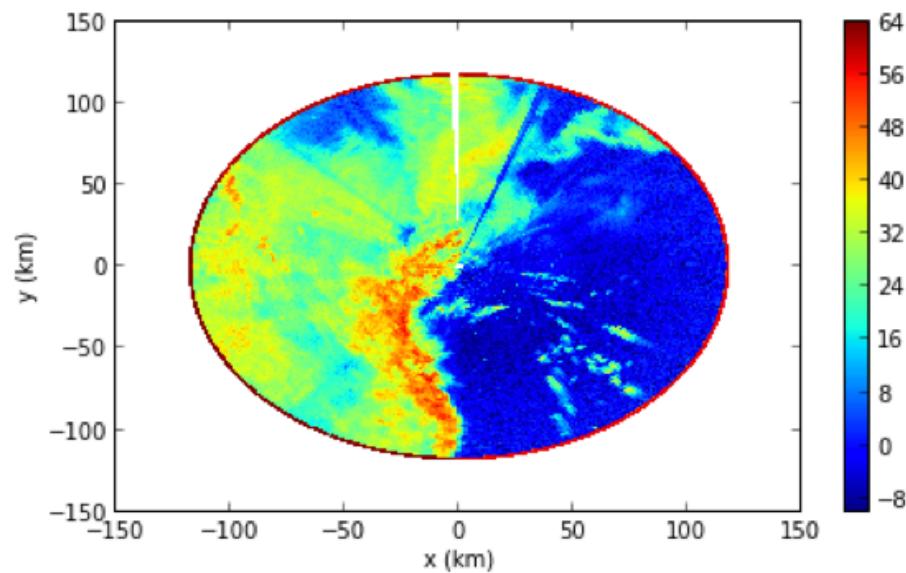
```
In [17]: figure(figsize=[15,5])
sweep_num=1
i1=myradar.sweep_info['sweep_start_ray_index']['data'][sweep_num]
i2=myradar.sweep_info['sweep_end_ray_index']['data'][sweep_num]
pcolormesh(myradar.time['data'][i1:i2], myradar.range['data'],
            myradar.fields['corrected_reflectivity_horizontal']['data'][i1:i2,:,:].transpose(),
            vmin=-10, vmax=64)
xlabel(myradar.time['standard_name']+ ('+'+myradar.time['units']+')'))
ylabel(myradar.range['standard_name']+ ('+'+myradar.range['units']+')'))
colorbar()
```

Out[17]: <matplotlib.colorbar.Colorbar instance at 0x10cce2950>



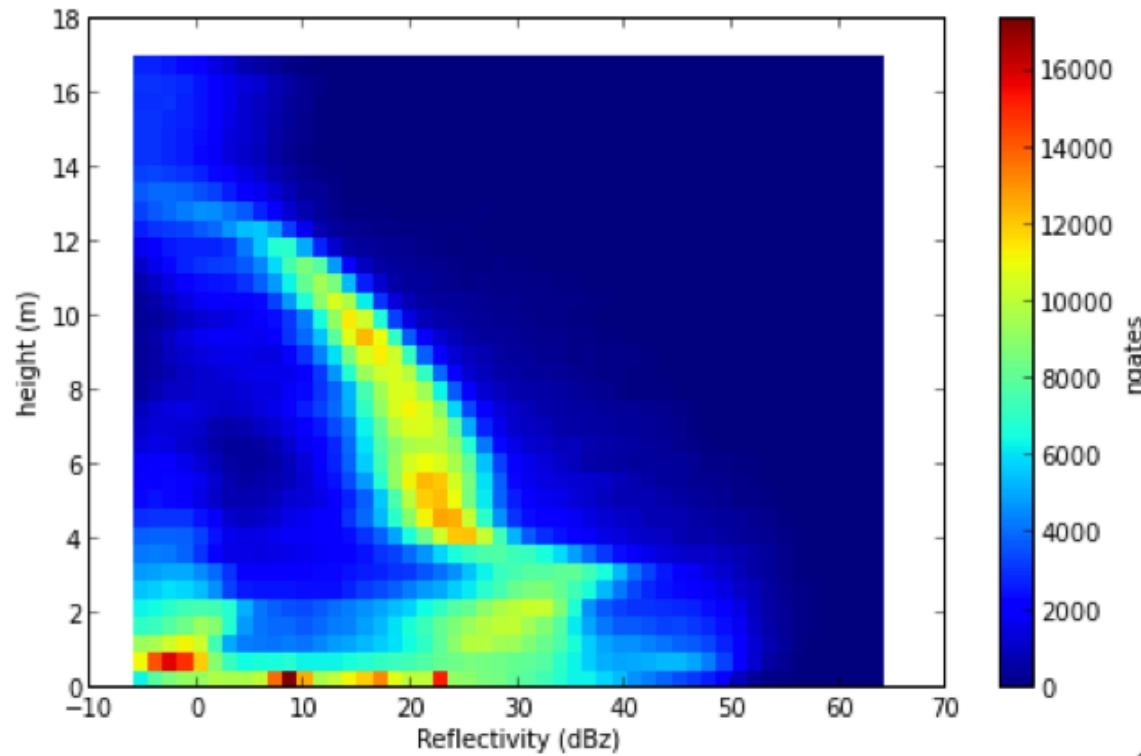
```
In [27]: rg,azg=meshgrid(myradar.range['data'],myradar.azimuth['data'])
rg,eleg=meshgrid(myradar.range['data'],myradar.elevation['data'])
x,y,z=radar_coords_to_cart(rg,azg, eleg) #appending carts
figure(figsize=[7,4])
sweep_num=1
i1=myradar.sweep_info['sweep_start_ray_index']['data'][sweep_num]
i2=myradar.sweep_info['sweep_end_ray_index']['data'][sweep_num]
pcolormesh(x[i1:i2,:]/1000.0, y[i1:i2,:]/1000.0,
            myradar.fields['corrected_reflectivity_horizontal']['data'][i1:i2,:], vmin=-10, vmax=64)
xlabel('x (km)'); ylabel('y (km)') ; colorbar()
```

```
Out[27]: <matplotlib.colorbar.Colorbar instance at
0x11b36f7a0>
```



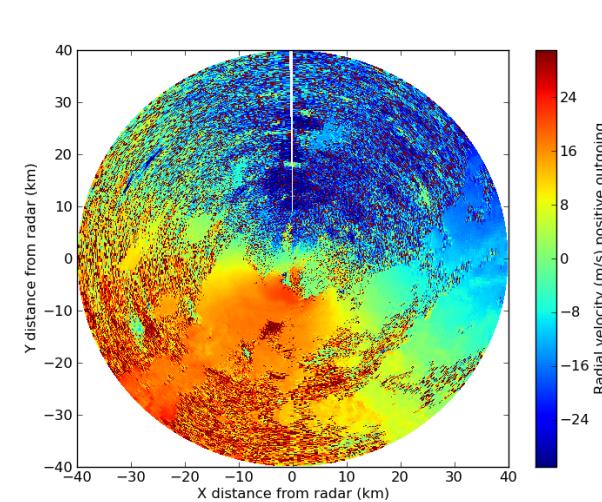
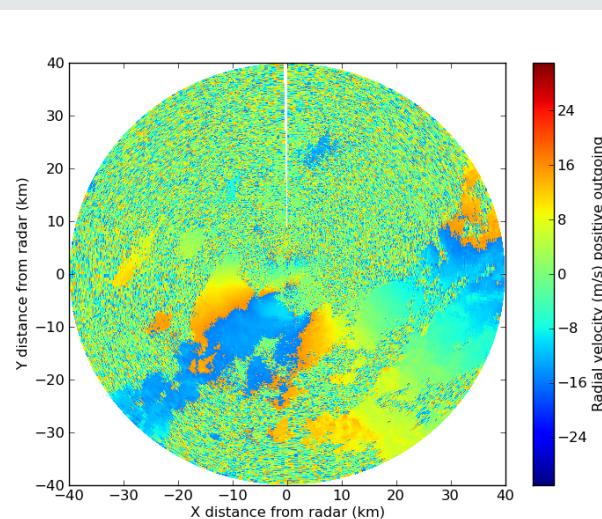
```
In [48]: H, xedges, yedges = np.histogram2d(z[:,0:-10].flatten()/1000.0,
                                         myradar.fields['corrected_reflectivity_horizontal']['data'][:,0:-10].flatten(),
                                         bins=(35, 50), range=([0,17], [-6,64]))
```

```
In [52]: figure(figsize=[8,5])
pcolormesh(yedges, xedges, H)
xlabel('Reflectivity (dBz)'); ylabel('height (m)'); cb=colorbar()
cb.set_label('ngates')
```



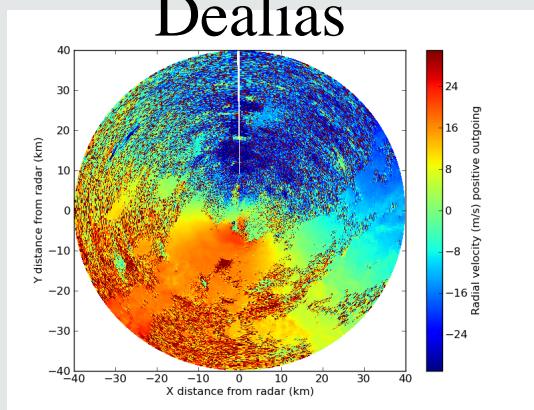
Abstract the interface, use the power of extensibility to improve performance

- The Py-Radar object gives a standard object to work on, if you build code around this object allows the building of interoperable modules.
- But there is lots of legacy code that works on other objects and in other languages.
- In addition optimization may require building extensions in C, C++ and Fortran.
- This does not matter, data can be taken from the Py-Radar object, transcribed, external module called and the data is then filed back in the radar object.
- Py-ART dealias module transcribes into a RSL container.

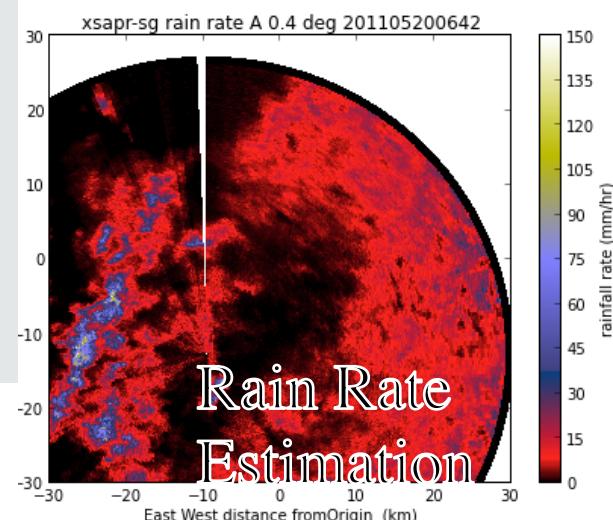


Current module list

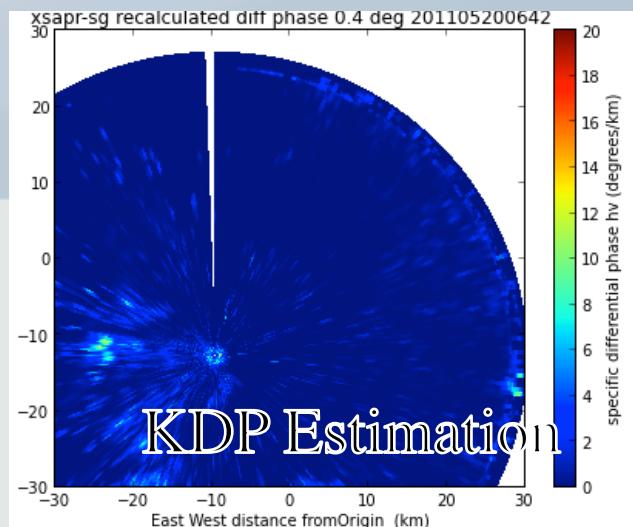
Dealias



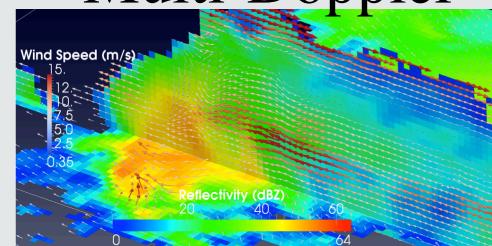
xsapr-sg rain rate A 0.4 deg 201105200642



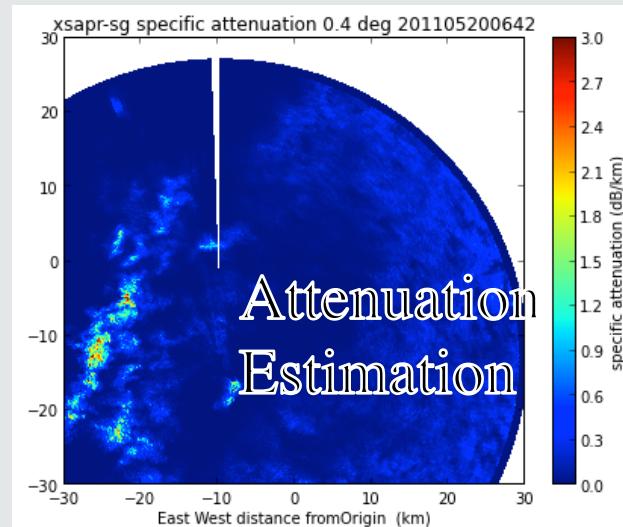
KDP Estimation



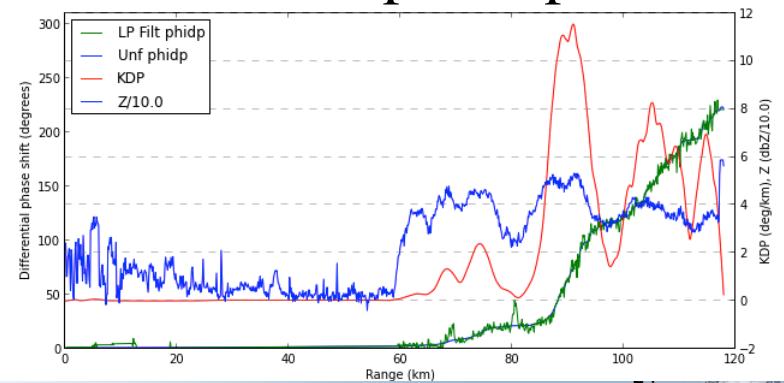
Multi-Doppler



Attenuation
Estimation



GLPK based phase processing



Generalized abstract radar mapping interface

- What we want is an object (in our favorite environment) to take one cloud of points and use objective analysis to map to another cloud of points.
- Eg:
 - `myobject=map((x,y,z), data)`
 - `Mymappeddata=myobject((xn,yn,zn))`
- This could, for example map from a sector of RHI scans to an aircraft track, nested PPIs to a volume, or even multiple data sources (of the same observable) to a single grid.
- We have implemented* this in Python using a ball tree.

***implemented=prototype^a**
^a**prototype=slow^c**
^c**slow=to be optimized soon!**

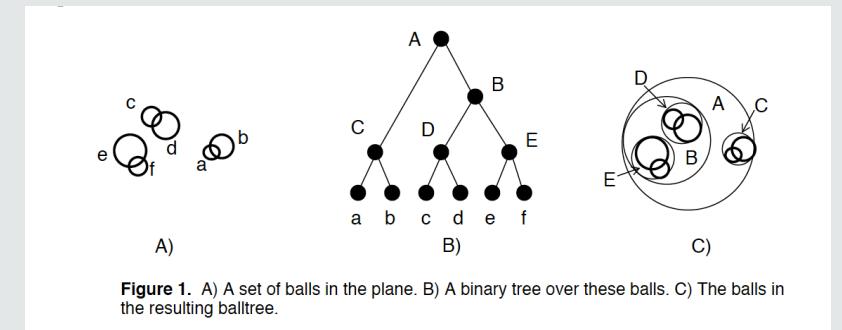
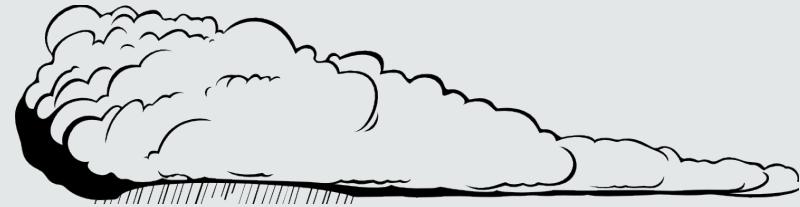
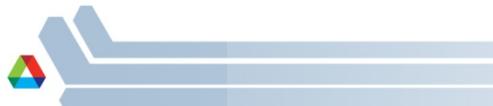
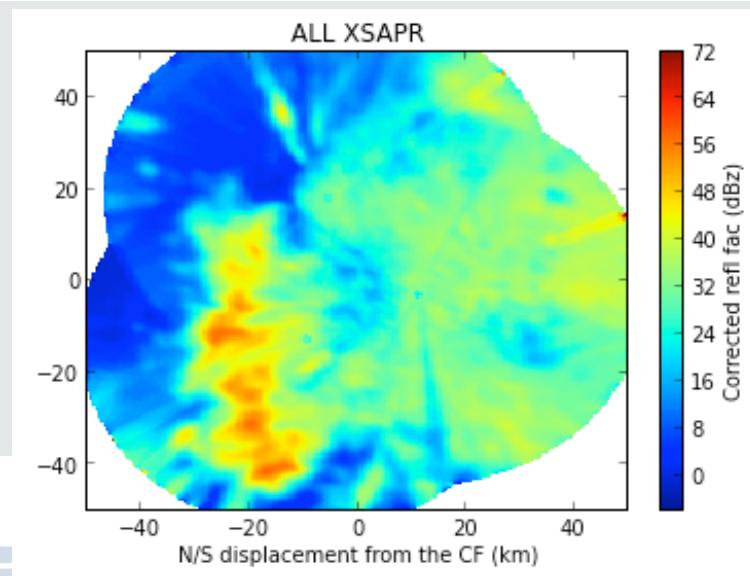
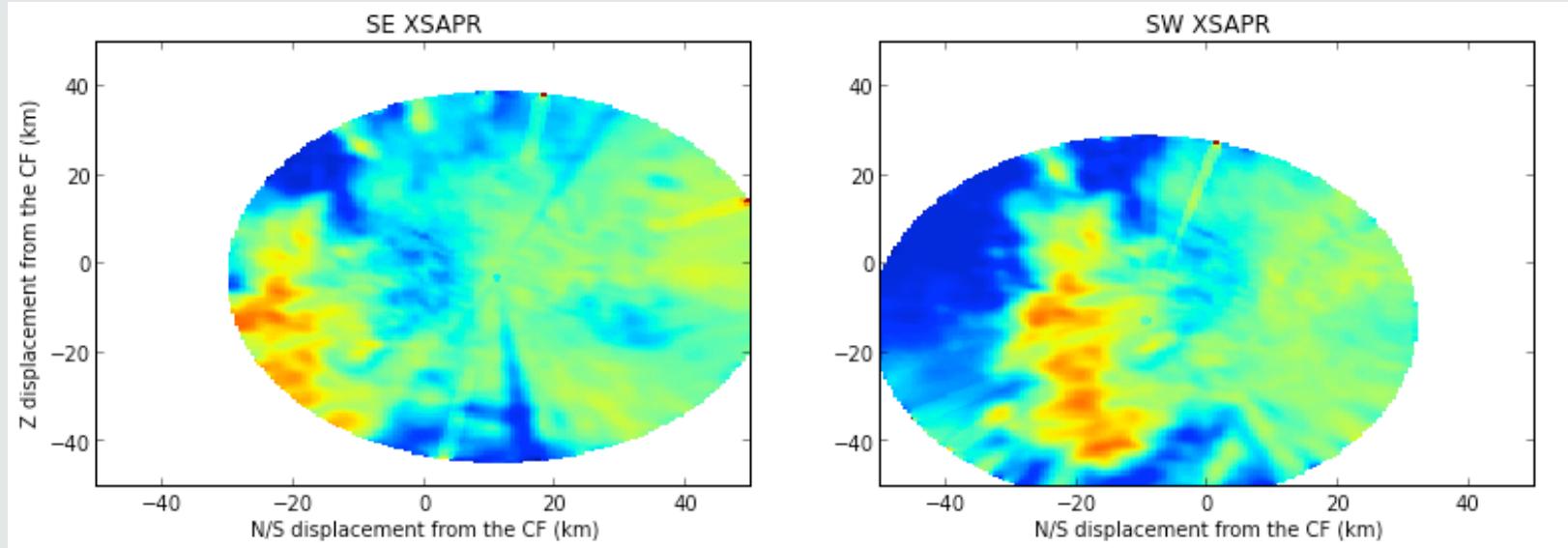


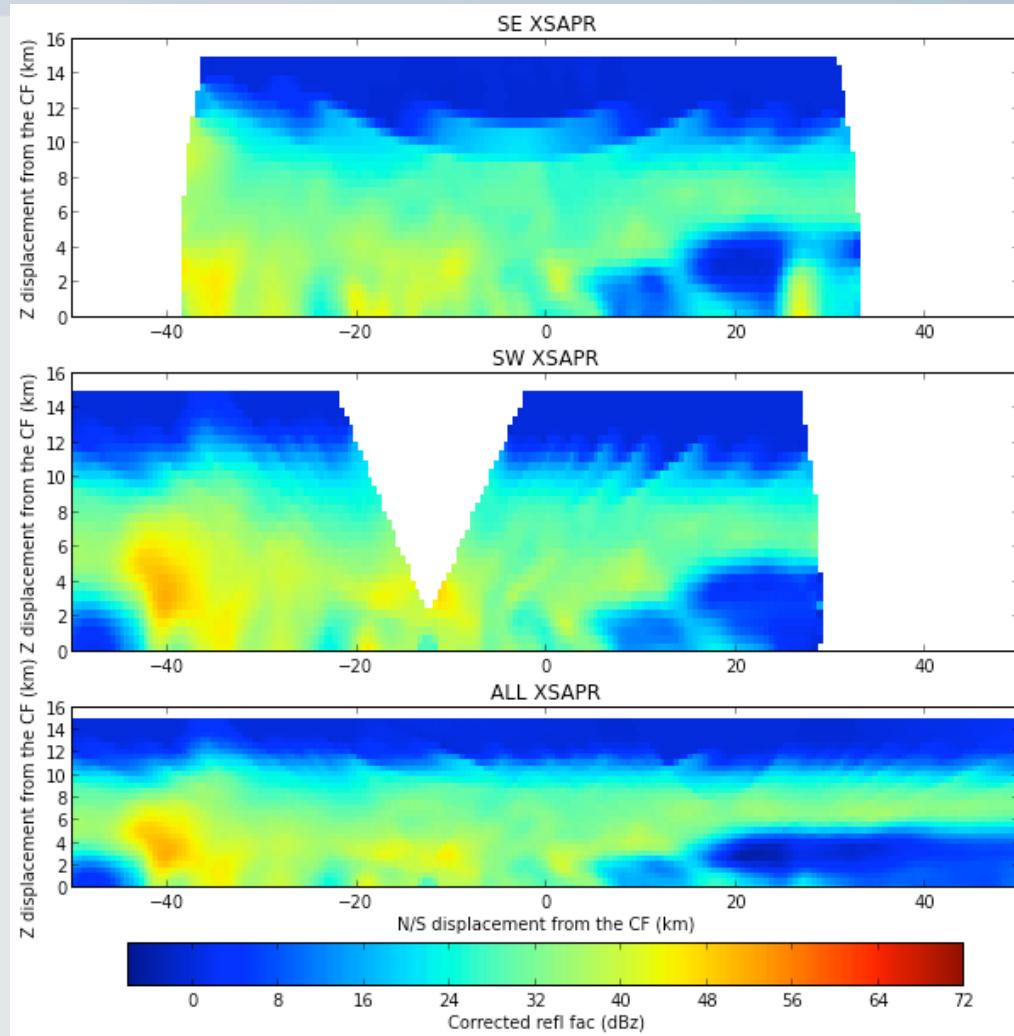
Figure 1. A) A set of balls in the plane. B) A binary tree over these balls. C) The balls in the resulting balltree.



Generalized abstract radar mapping interface



Generalized abstract radar mapping interface



Py-ART and LROSE

LROSE: Application stack performance

- A cloud of command line driven routines for data analysis and interactive display.
- Performance driven.
- All the way from instrument to application.
- Uses community standard file formats.

Py-ART: Interactive environment flexibility

- A library of python importable modules.
- From saved radial data (looking at radar spectra support).
- Based on common python objects.
- Flexibility and ease driven.



Synergies and commonalities

- The two projects have common goals: reading, correcting, mapping, displaying and analyzing radial data.
- Even though we have two different approaches we are building a strong collaboration about building a core set of libraries that can be used in either environment.
- The extensibility of Python can be leveraged to bring in C code from LROSE.
- LROSE can leverage Python tools to build displays (eg Py-SOLO)



Synergies and commonalities

- The two projects have common goals: reading, correcting, mapping, displaying and analyzing radial data.
- Even though we have two different approaches we are building a strong collaboration about building a core set of libraries that can be used in either environment.
- The extensibility of Python can be leveraged to bring in C code from LROSE.
- LROSE can leverage Python tools to build displays (eg Py-SOLO)

**Our goals are simple:
Build community software to lower
the barriers to entry for radar science**



Future directions in (li/ra)dar software

- Radar/Lidar systems are producing more data with greater bit depth and sampling.
- This data is a classic example of “big data” not due, necessarily, to the size but also due to the geometric complexity.
- Lowering barriers to access is essential for progress.
- For large radar datasets a barrier is also compute power and IO. Need to take code to the data.
- The good news is that radial data is particularly amenable to parallelization.



Future directions in (li/ra)dar software

- Radar/Lidar systems are producing more data with greater bit depth and sampling.
- This data is a classic example of “big data” not due, necessarily, to the size but also due to the geometric complexity.
- Lowering barriers to access is essential for progress.
- For large radar datasets a barrier is also compute power and IO. Need to take code to the data.
- The good news is that radial data is particularly amenable to parallelization.

**The NEXRAD radar record is over 0.8PB
and growing (faster!).**



Conclusions

- There is a need for community radial software projects.
- LROSE and Py-ART represent two philosophies for going forward working in close collaboration.
- Aspects of LROSE are available and Py-ART will be released later in April.

Fork me on GitHub

Coming Soon: <https://github.com/ARM-DOE/pyart>

<http://www.eol.ucar.edu/>

http://www.ral.ucar.edu/projects/titan/docs/radial_formats/

