Python-based scientific analysis and visualization of precipitation systems at NASA Marshall Space Flight Center

Timothy J. Lang

Acknowledgments: Scott Collis, Brody Fuchs, Nick Guy, Paul Hein, Jonathan Helmus, Brent Roberts

Talk Overview

1. Motivation

2. Three Python Modules

- Python Advanced Microwave Precipitation Radiometer Data Toolkit (**PyAMPR**)
- Marshall MRMS Mosaic Python Toolkit (**MMM-Py**)
- Python Turbulence Detection Algorithm (**PyTDA**)

3. Current and Future Paths

4. Summary

Python Advanced Microwave Precipitation Radiometer Data Toolkit (PyAMPR)

Motivation

- AMPR is a polarimetric, multi-frequency, cross-track scanning airborne passive microwave radiometer managed by NASA MSFC
- Nearly 25-year scientific legacy, flown in ~15 airborne missions
- In operation today – MC3E (2011), IPHEx (2014), OLYMPEX (2015)

Dataset available here: **http://ghrc.msfc.nasa.gov**

Problems

- Obscure, user-unfriendly ASCII format with hundreds of columns
- Dataset format has changed over years, from project to project
- Polarimetric upgrade (2010) added new channels to dataset
- Legacy data ingest and visualization software uses outdated or commercially licensed languages

Goal – Modernize AMPR data management and visualization

PyAMPR Software Structure

Python 2.x – NumPy, matplotlib, Basemap, time, datetime, calendar, gzip, codecs
Other dependencies – simplekml, auxiliary code for custom colormap and Google Earth output (from http://ocefpaf.github.io/python4oceanographers/blog/2014/03/10/gearth/)

Class AmprTb

Attributes:

TBs from all channels (10, 19, 37, 85 GHz – Both A & B), Nav & GeoLocation info, Terrain info (elevation & land fraction) – All for a single flight

Methods:

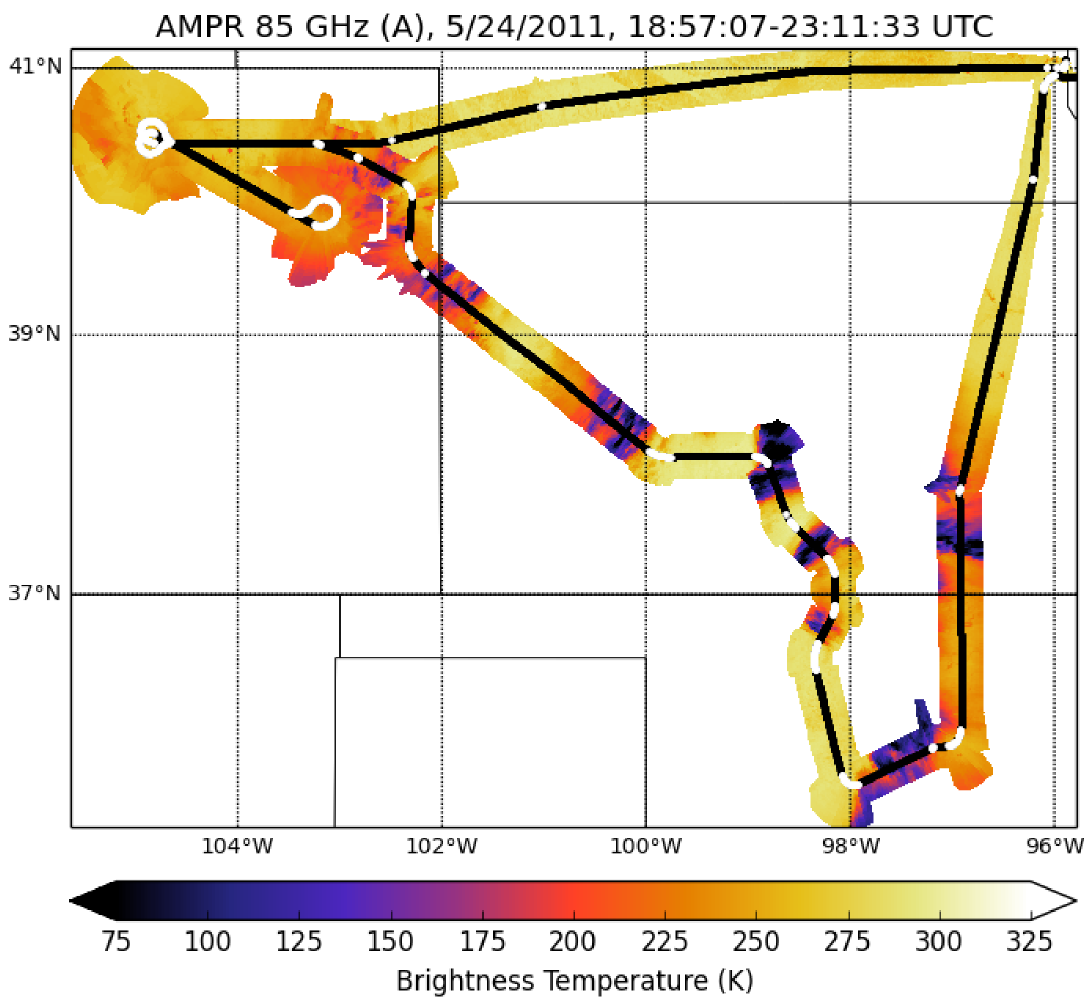
__init__, read_ampr_tb_level1b, help, plot_ampr_track, plot_ampr_channels, calc_polarization, write_ampr_kmz

Common data model regardless of flight/project
(missing variables get bad data values)

Getting Started

iphex_data = pyampr.AmprTb('iphex_data_file.txt', project='IPHEX')

pyampr.AmprTb.plot_ampr_track()

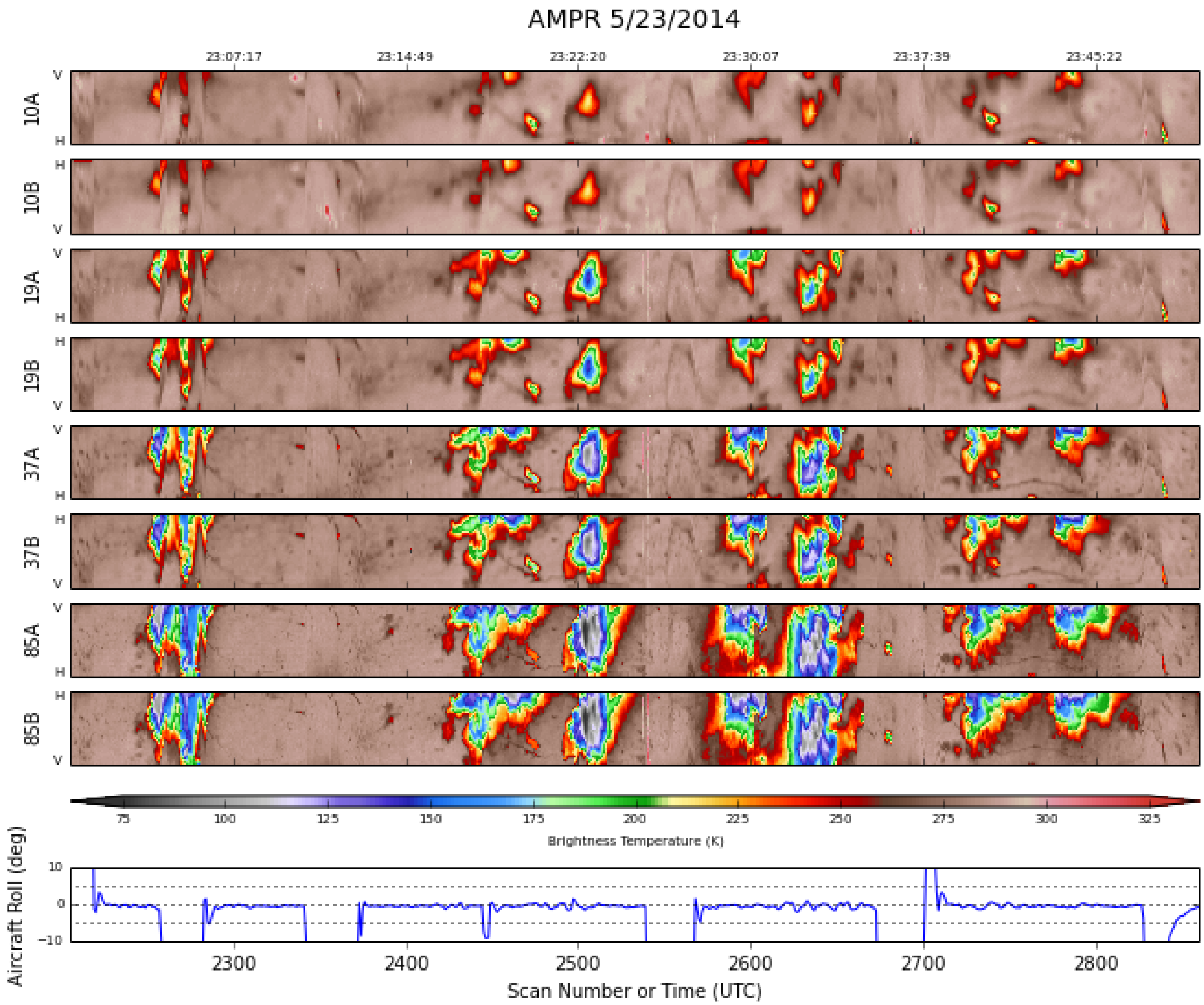


pyplot.pcolormesh + Basemap backbone

<u>Options</u>

- Lat/Lon range adjustment
- Time/Scan range adjustment
- Aircraft Track flag
- Gridline adjustments
- Image file save
- Custom/default figure titles
- Custom color tables, levels
- Manage poor geolocations
- Aircraft maneuver suppression
- Figure, axis object return

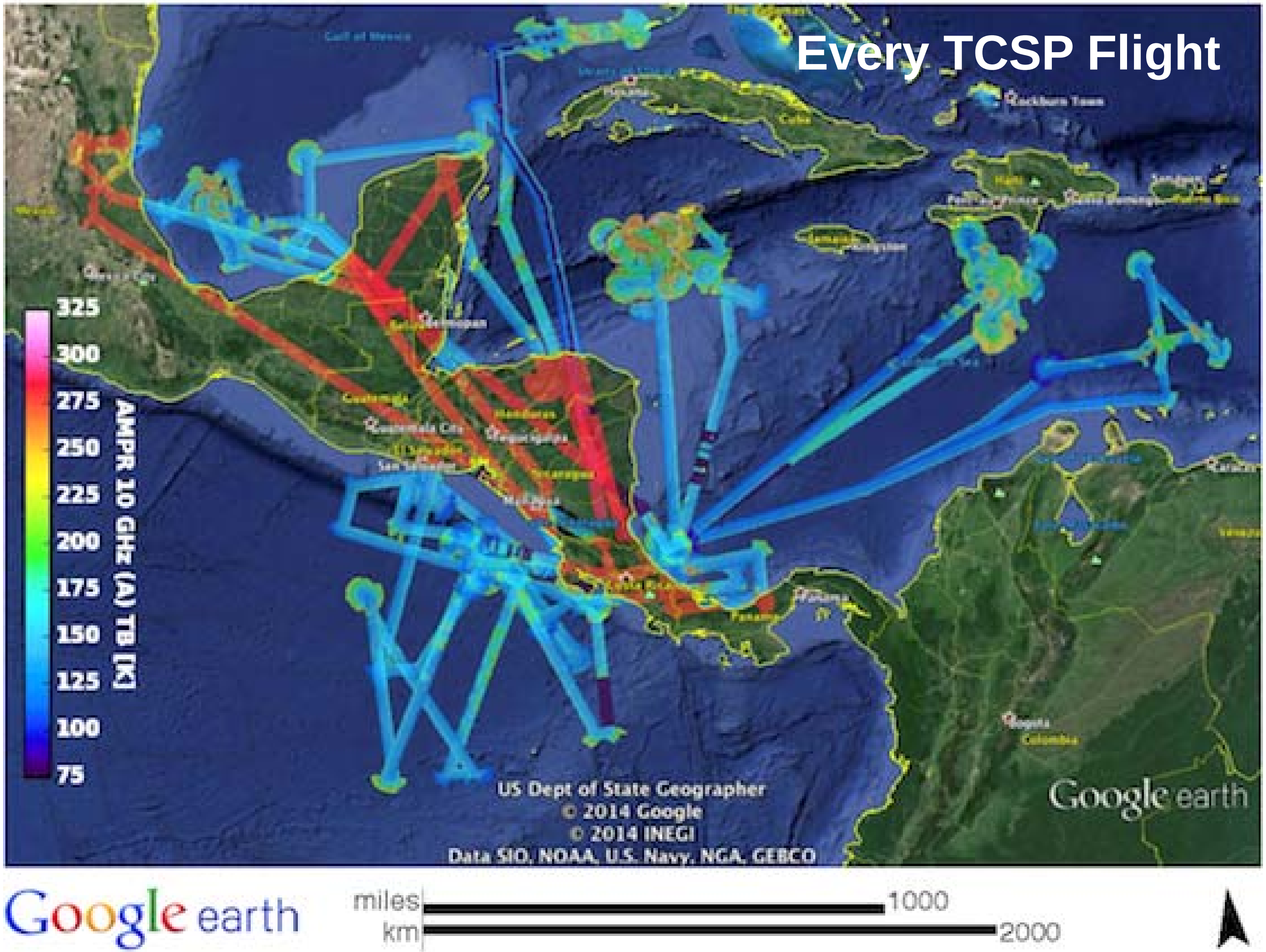
pyampr.AmprTb.plot_ampr_channels()

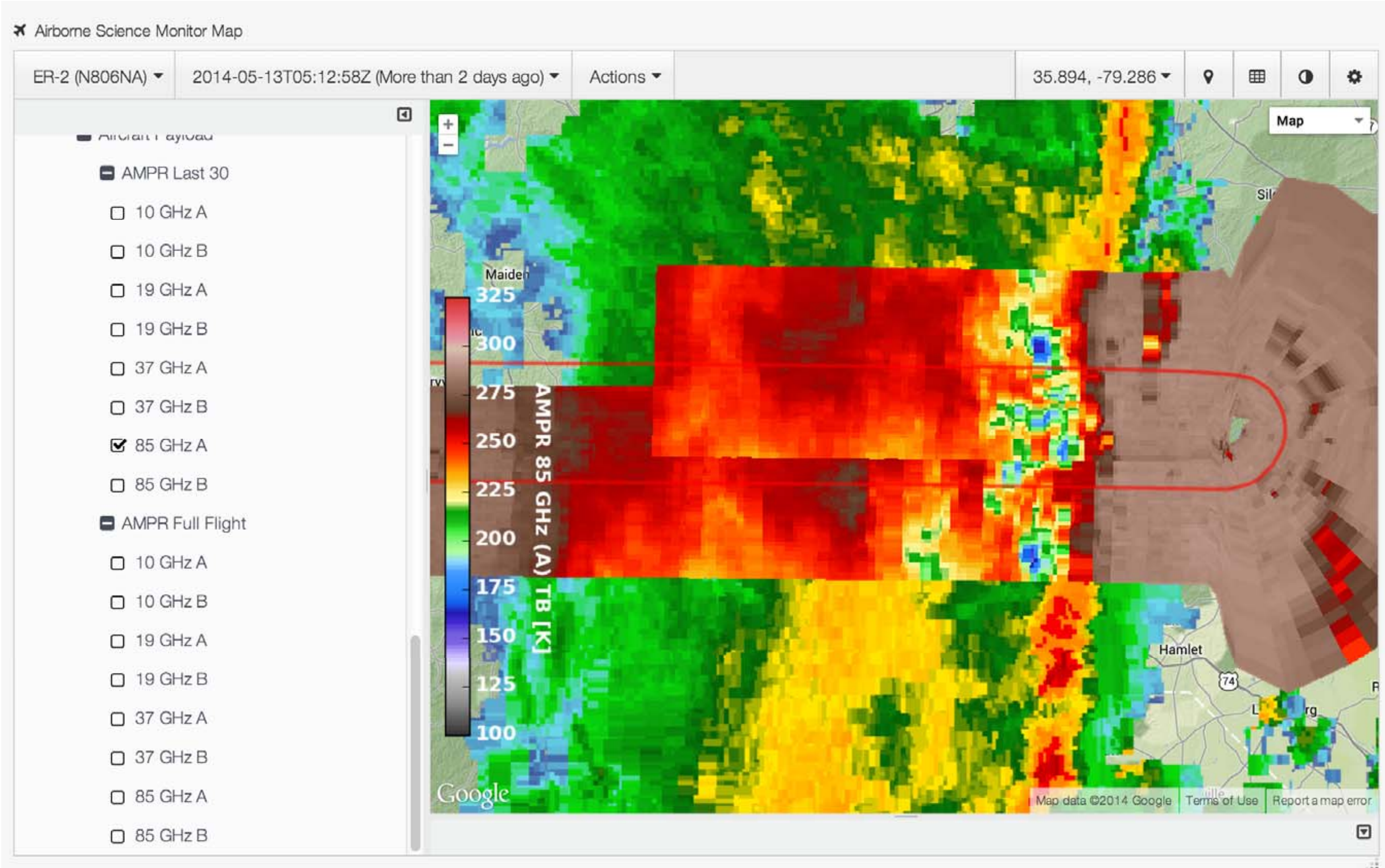


pcolormesh + plot backbone

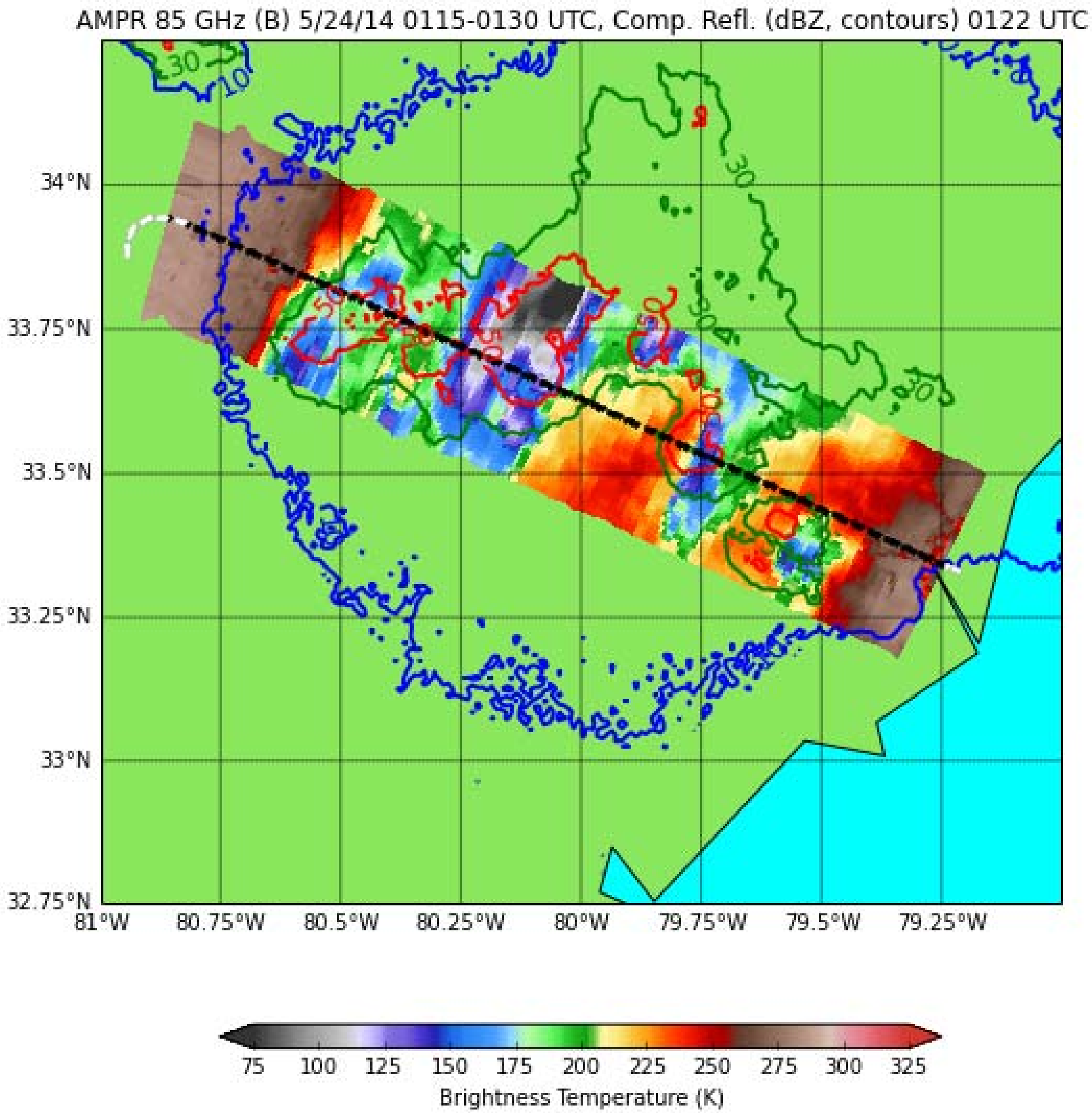
Similar options to plot_ampr_track(), minus geolocation stuff

Support for polarization deconvolution via calc_polarization method pyampr.AmprTb.write_ampr_kmz()





- AMPR was the only scanning instrument on ER-2 to provide real-time imagery during IPHEx campaign
- Real-time data transmission enabled continual production of KMZs (powered by PyAMPR), which were ingested into NASA Mission Tool Suite



AMPR + NEXRAD

<u>Key feature of PyAMPR</u>

Strong canned imagery creation, but figure/axis return enables endless customization

Marshall MRMS Mosaic Python Toolkit (MMM-Py)

Motivation

- NOAA MRMS mosaics provide 3D NEXRAD radar reflectivity on a 2-minute, 0.01° national grid (formerly 5-minute)
- Mosaics a godsend for multiple applications and research projects, including my own interests (large MCSs that produce sprites)

Problems

- Data distributed as regional tiles in custom binary format
- Major change to tile format in 2013 (including NMQ to MRMS name change)
- No well-known, widely distributed software for detailed analyses

Goal – Read into a common data model and display any MRMS file (binary or netCDF) and be able to merge regional tiles as needed

MMM-Py Software Structure

Python 2.x – NumPy, matplotlib, Basemap, scipy, time, calendar, gzip, os, struct

Class MosaicTile

Attributes:

reflectivity (mrefl3d, mrefl3d_comp), grid info, Time, Duration, Version, Filename, Variables (support for future dual-pol)

Methods:

__init__, read_mosaic_binary, read_mosaic_netcdf, get_comp, diag, help, plot_horiz, plot_vert, three_panel_plot, write_mosaic_binary, subsection, output_composite

Class MosaicStitch

Child class of MosaicTile, read methods disabled, added stitch_ns() & stitch_we()

Function stitch_mosaic_tiles

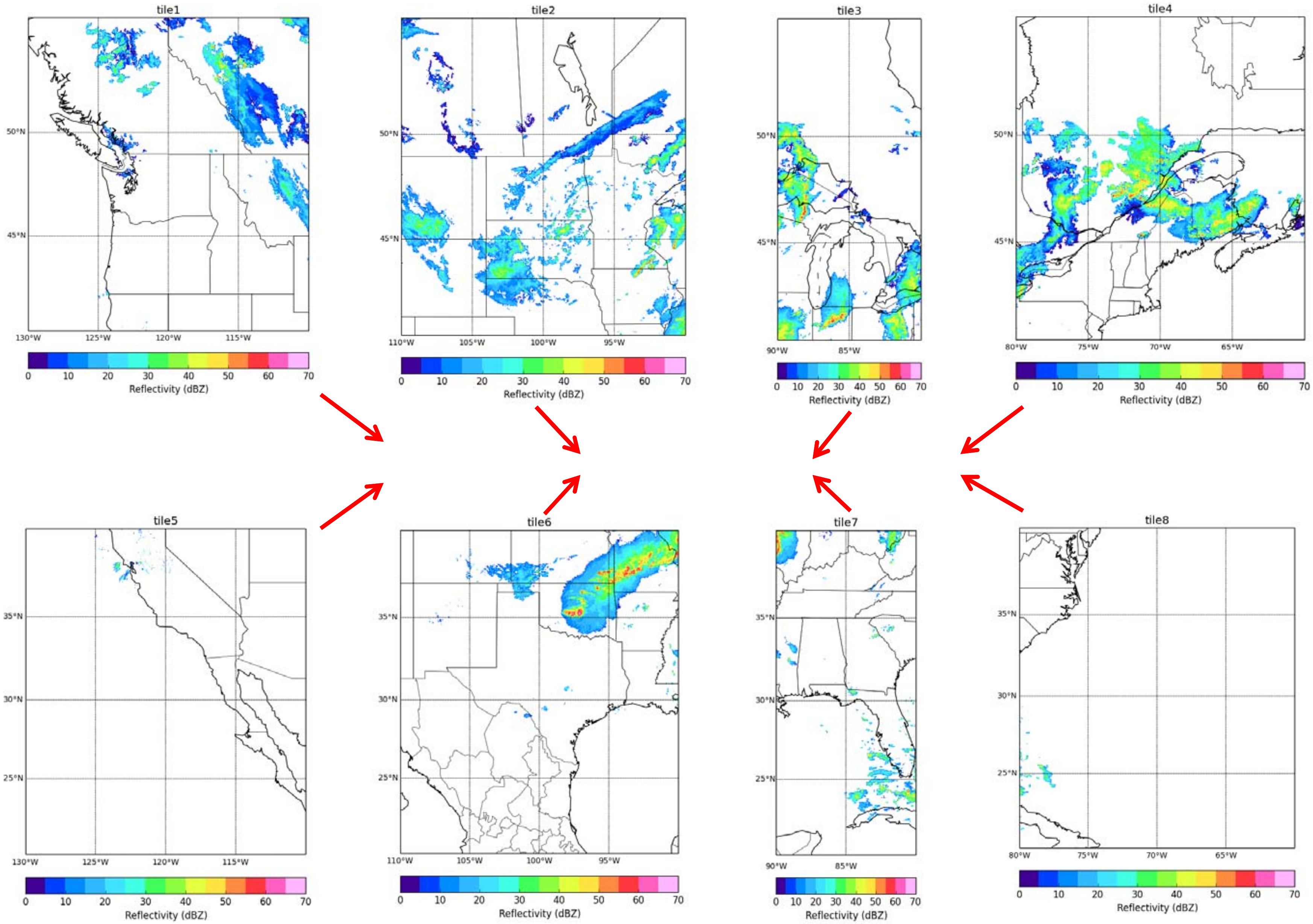
Arguments: Array of MosaicTiles/Stitches, direction of stitching (if 1D array)

Output: MosaicStitch

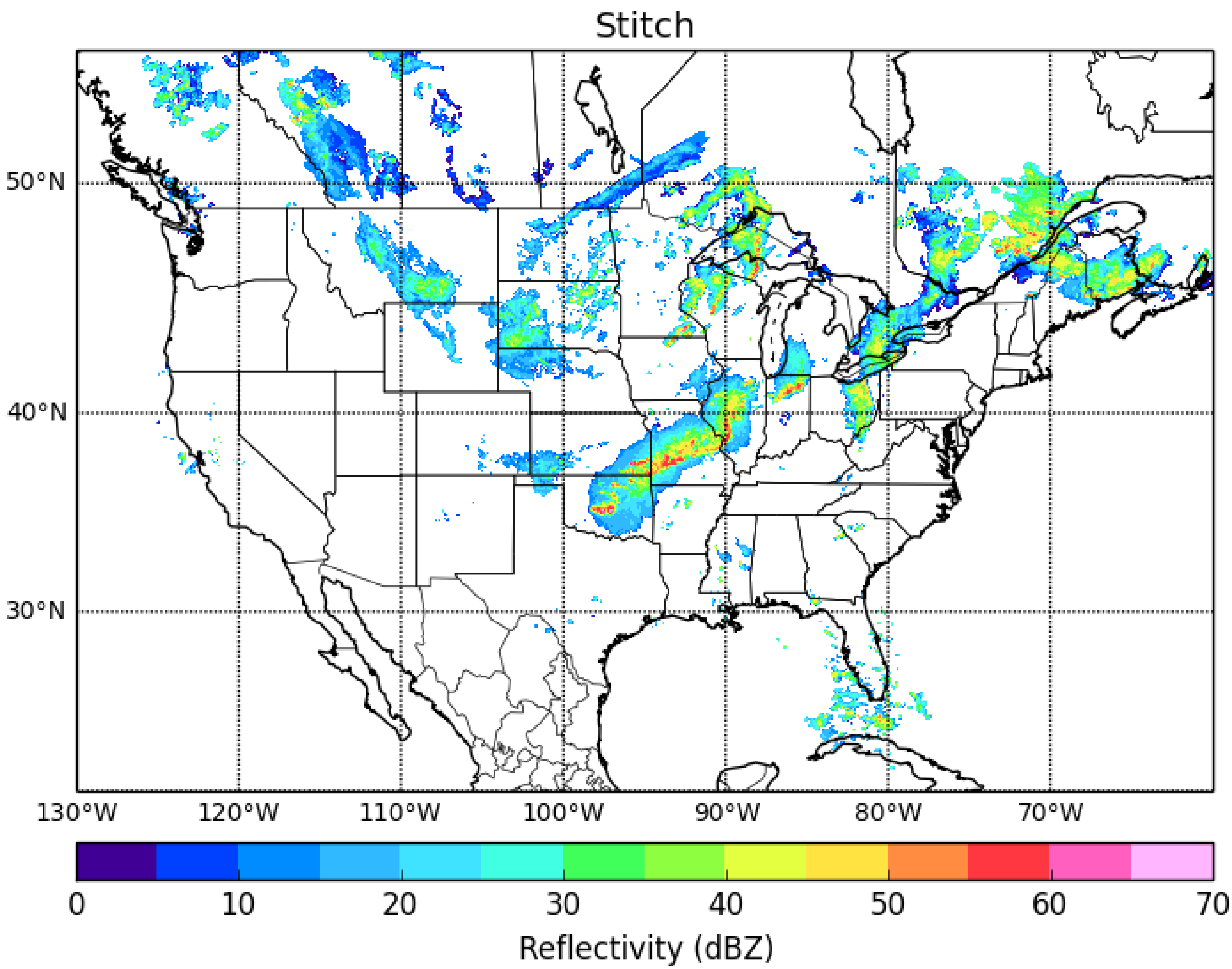
Getting Started

mosaic_tile = mmmpy.MosaicTile('mosaic_tile.bin.gz')

mmmpy.stitch_mosaic_tiles() in action – Going from multiple MosaicTiles …



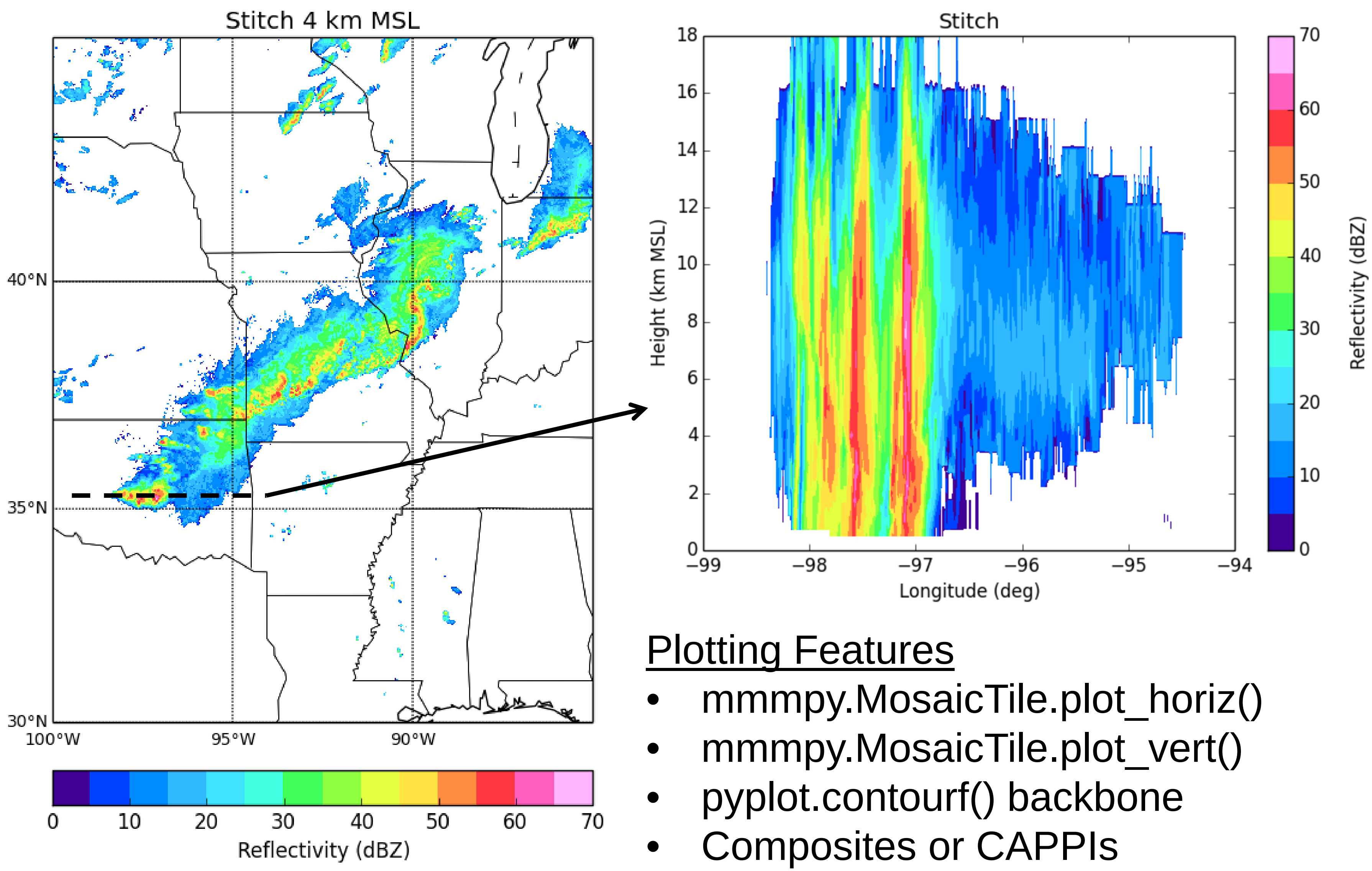
new_stitch = mmmpy.stitch_mosaic_tiles(map_array=[[tile1, tile2, tile3, tile4], [tile5, tile6, tile7, tile8]])



… to a MosaicStitch

numpy.append() is backbone

Iterative calls to mmmpy.MosaicStitch.stitch_ns/we()

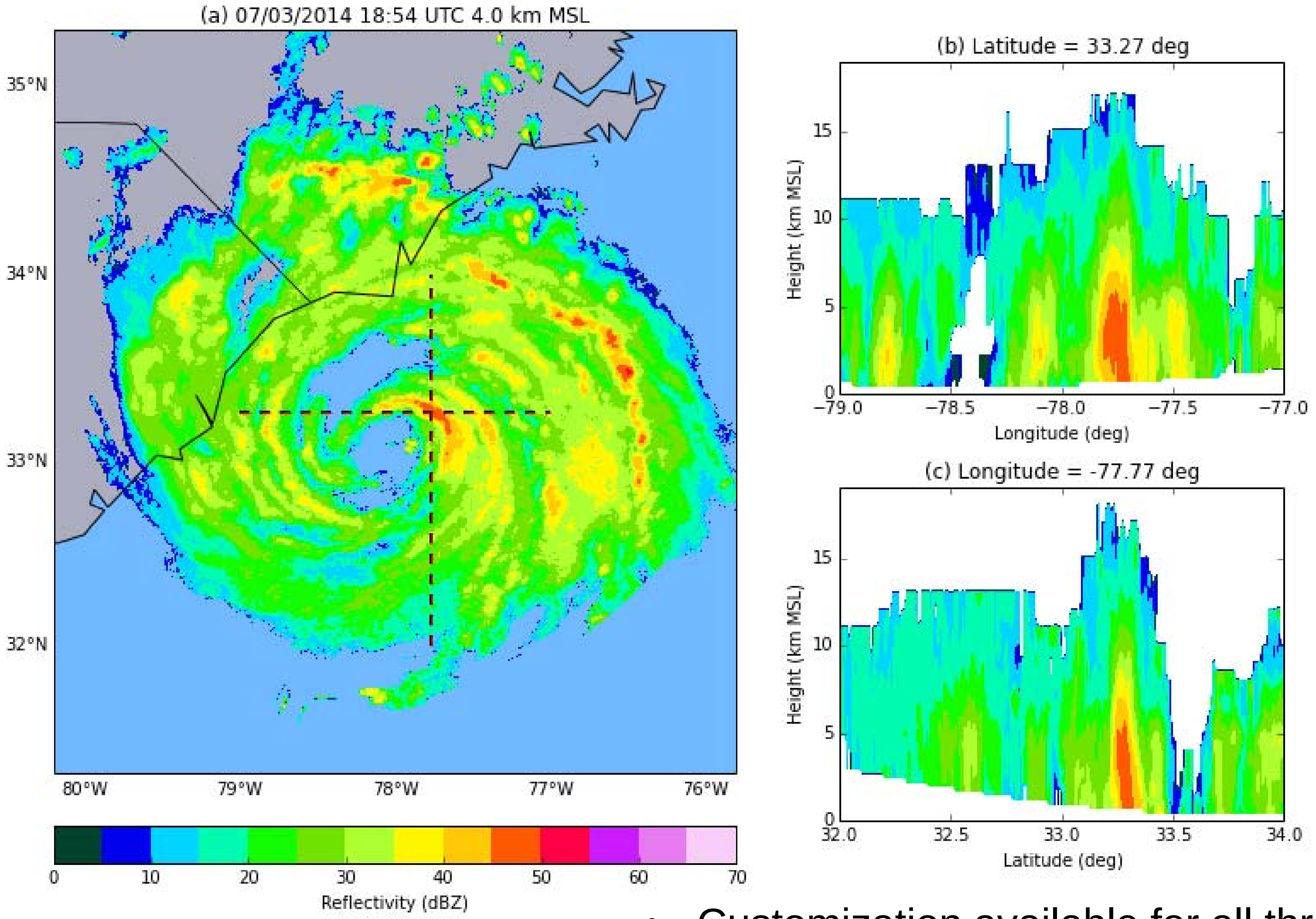


<u>Plotting Features</u>

- mmmpy.MosaicTile.plot_horiz()
- mmmpy.MosaicTile.plot_vert()
- pyplot.contourf() backbone
- Composites or CAPPIs
- Custom lat/lon/vertical range
- Custom gridlines & Basemap res.
- Custom color map, levels, titles

mmmpy.MosaicTile.three_panel_plot()

Example: Hurricane Arthur



- Customization available for all three subplots, including figure/axis return
- Once again – Canned yet flexible figure creation

Python Turbulence Detection Algorithm (PyTDA)

Motivation

- Turbulence (specifically, eddy dissipation rate or EDR) long known to be retrievable via Doppler spectrum width analysis
- NCAR Turbulence Detection Algorithm (NTDA, Williams et al. 2006) uses fuzzy-logic techniques to correct for artifacts and smooth data for accurate retrievals

Problems

- Spectrum width noisy, needs smoothing/QC before simply inverting to EDR
- NTDA aimed at NEXRAD & US Gov. use (aviation forecasting over entire US)
- Need turbulence retrieval for generic radars & case studies (i.e., the little guy!)

Goal – Rapid, efficient retrieval of turbulence from Doppler radar data

PyTDA Software Structure

Python 2.x – numpy, scipy, sklearn, time, Py-ART

Function calc_turb_sweep()
Arguments: Py-ART radar object, sweep number, split-cut flag, NTDA flag, NTDA search radius, x/y limits, radar beamwidth and gate spacing
Output: 2D turbulence sweep, plus lat/lon of each gate

Approach:

- NTDA does weighted average of interest fields (C_{pr}, C_{zh}, C_{snr}, C_{swv}, C_{rng}) in user-defined disc surrounding gate (usually R = 2 km)
- Invert spectrum width using long- & short-range equations (e.g., Labitt 1981) - scipy.special.gamma() & scipy.special.hyp2f1()
- One-dimensionalize and reduce data using ravel() and masks, respectively
- sklearn.neighbors.BallTree for nearest neighbor search (NTDA search radius & spectrum width variance)
- numpy function broadcasting whenever possible, Cython when not

Function calc_turb_vol()
Arguments: Similar to above, iteratively calls calc_turb_sweep() over volume
Output: Py-ART radar object with masked turbulence field ($EDR^{1/3}$) added

PyTDA Example

```
In [2]: for iii in xrange(len(files)):
            fname = os.path.basename(files[iii])
            print fname
            radar = pyart.io.read(files[iii])
            pytda.calc_turb_vol(radar, radius=2.0, split_cut=False,
                                verbose=False, xran=xran, yran=yran)
            pyart.io.write_cfradial(fname+'.nc',radar)

20101026_130417_KGWX_v257_SUR.uf
20101026_130910_KGWX_v258_SUR.uf
20101026_131404_KGWX_v259_SUR.uf
20101026_131857_KGWX_v260_SUR.uf
20101026_132349_KGWX_v261_SUR.uf
20101026_132843_KGWX_v262_SUR.uf
20101026_133434_KGWX_v263_SUR.uf
20101026_133928_KGWX_v264_SUR.uf
```

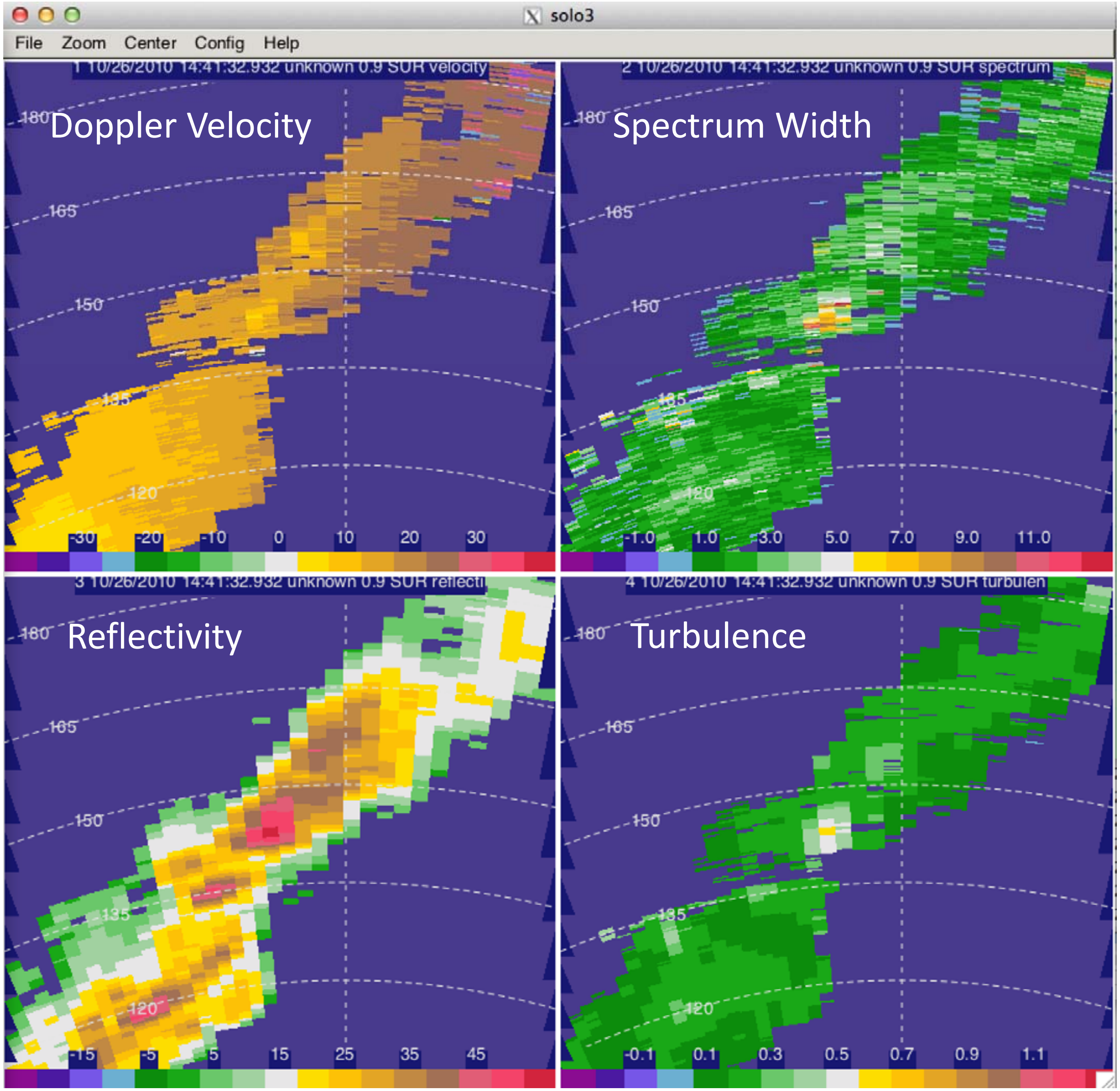
My goal with all of these modules has been to turn complex tasks into one line of code

Retrieval of turbulence ($EDR^{1/3}$) using Python variant of NCAR Turbulence Detection Algorithm (PyTDA)

Software interfaces seamlessly with Py-ART radar objects but is its own standalone package

Turbulence field can be viewed using Py-ART RadarDisplay object, or saved to radar file, viewed in NCAR solo, gridded, etc.

NB: No correction for wind shear yet



Current/Future Work

Python Implementation of Single-Doppler Retrieval of 2D Winds via 2DVAR

(*Xu et al. 2006*)

- Similar approach to PyTDA (Standalone module w/ Py-ART backbone)
- Statistical interpolation using radial velocity to correct a background field
- (Background can be assumed 0, or estimated via sounding/VAD/scatterometer)
- Essentially simplified data assimilation, Extension of VAD
- Capable of retrieving mesoscale fronts/boundaries
- Similar errors to more advanced, model-based methods (*Fast et al. 2008*)
- Solve matrix equations using numpy.linalg module

2900 Q. XU *et al.*

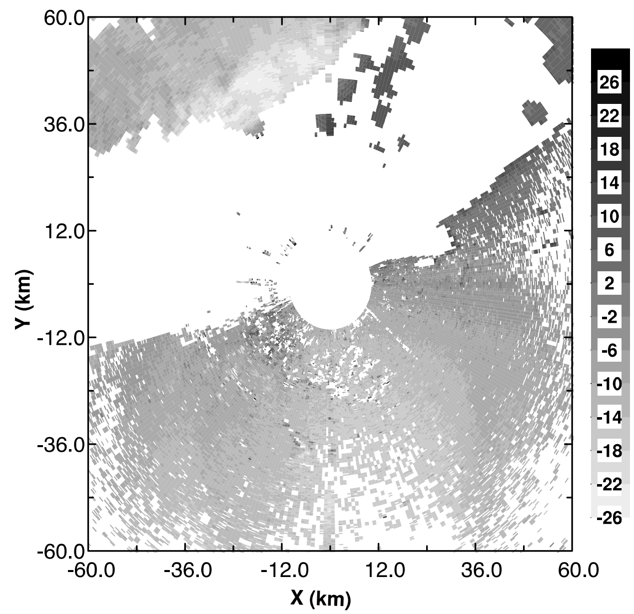
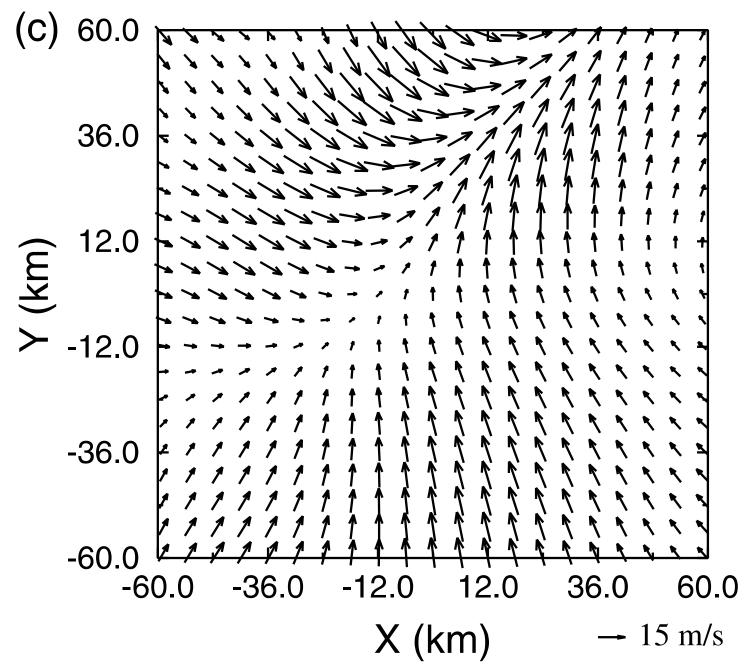


Figure 6. De-aliased radial velocities on the conical surface of 0.5 degree elevation angle over the area of 120 × 120 km^2 centred at the radar. The raw radial-velocity data were collected by the Oklahoma City WSR-88D (KTLX) radar 0040 UTC on16 June 2002 for a surface cold front over the area of Oklahoma State.



Summary

1. **PyAMPR** – Read, analyze, visualize AMPR data
2. **MMM-Py** – Read, analyze, visualize MRMS 3D radar mosaics
3. **PyTDA** – Retrieve turbulence from Doppler radar data

All to be hosted at:
https://github.com/tjlang (Have patience with NASA!)

Other MSFC Precip Python Work At 95th AMS

- Poster 427, "A high-resolution merged wind dataset for DYNAMO: Progress and future plans" – Py-ART used to correct and visualize Doppler radar data (Hall 4, Monday-Tuesday, 5-6 Jan, 3MJOSYMP)
- Talk 6.5, "Investigation of the electrification of pyrocumulus clouds" – Py-ART used to analyze pyroCb electrification (Rm 225AB, 2:30p Tuesday 6 Jan, 7LIGHTNING)