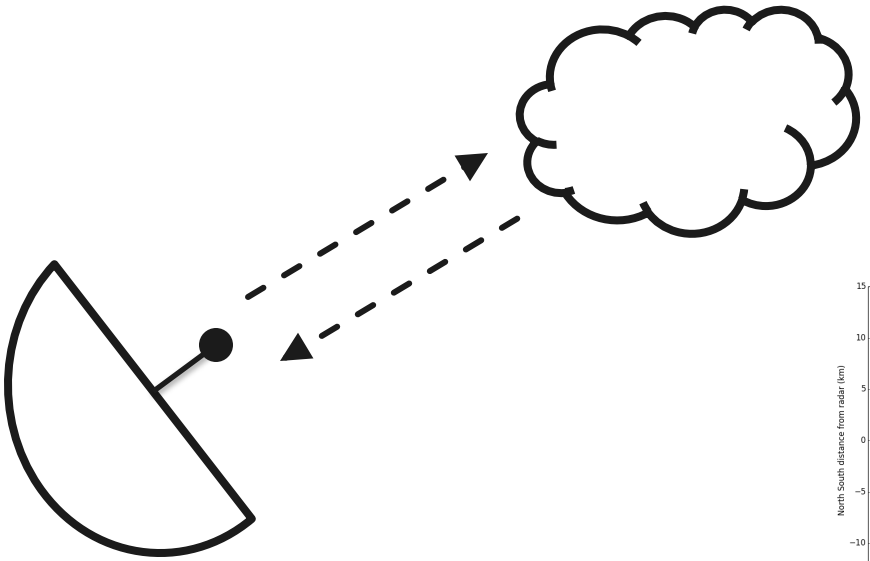
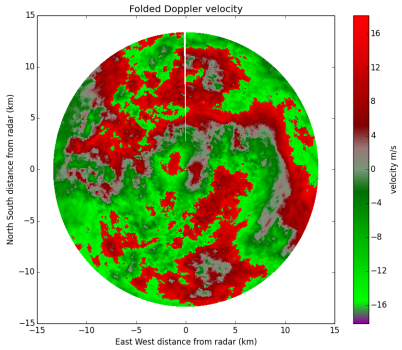
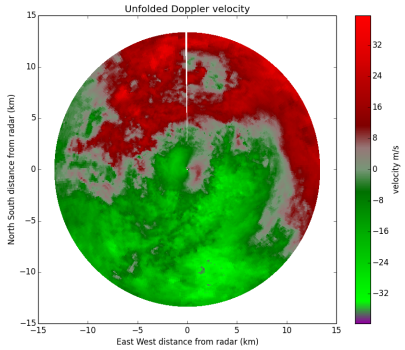
# Designing and implementing weather radar algorithms in Python

**Jonathan Helmus**[1], Kirk North[2], Scott Giangrande[3], Scott Collis[1]

[1]Argonne National Laboratory, USA
[2]McGill University, Montreal, Canada
[3]Brookhaven National Laboratory, USA






This presentation 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.

# The ARM Climate Research Facility

- The U.S. Department of Energy's Atmospheric Radiation Measurement (ARM) Climate Research facility provides *in situ* and remote sensing data with a mission to **improve climate and earth systems models**.

- The program operates a **large number of instruments** at three **fixed sites** as well as two **mobile facilities** that can be deployed in support of field campaigns around the globe.

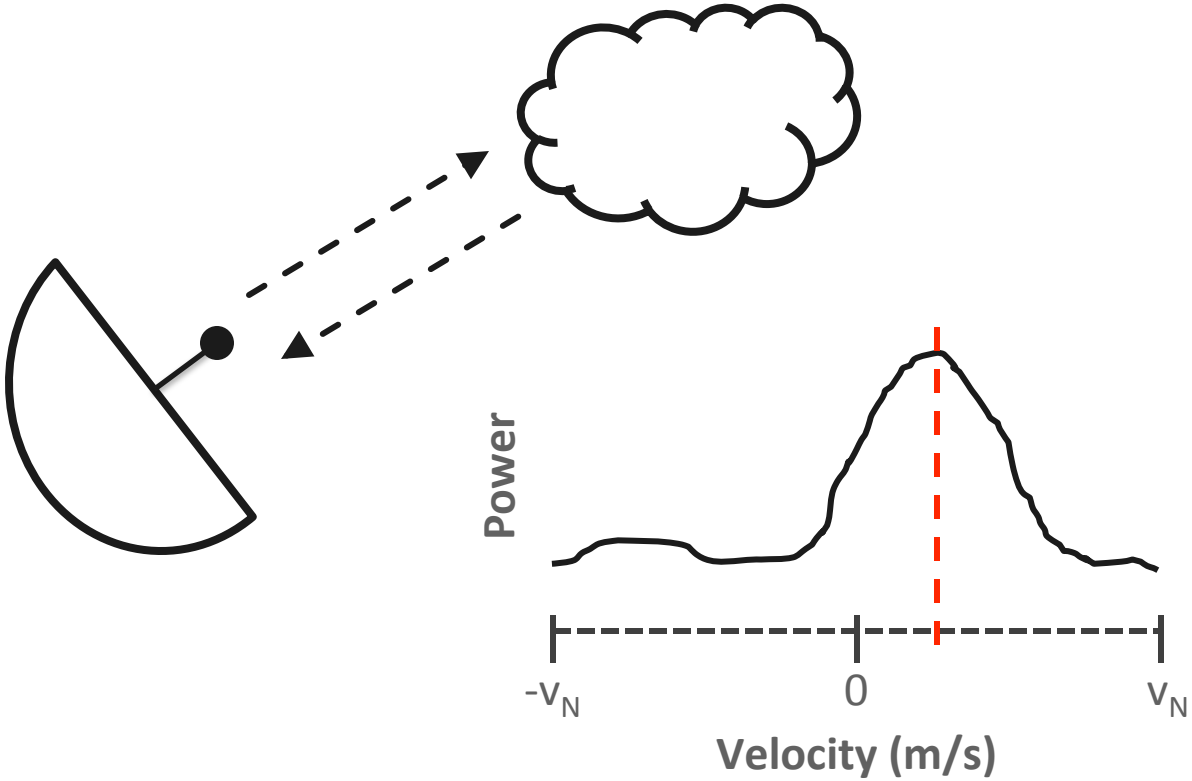
- This instrumentation includes a number of scanning **cloud and precipitation radars** which were acquired with funding from the American Recovery Act.

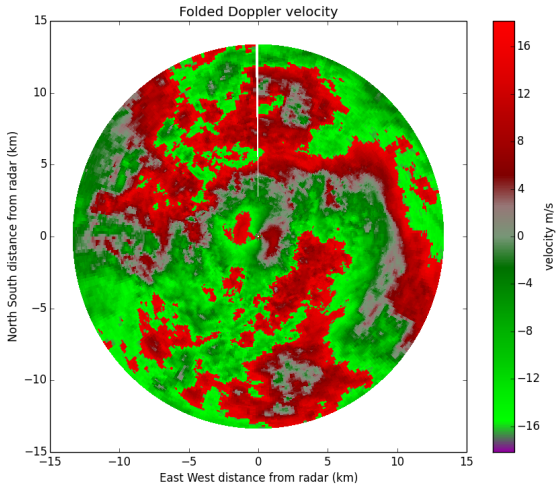
- The program is the process of preparing **two new scanning radars** for deployment in 2015.

# The Python ARM Radar Toolkit: Py-ART

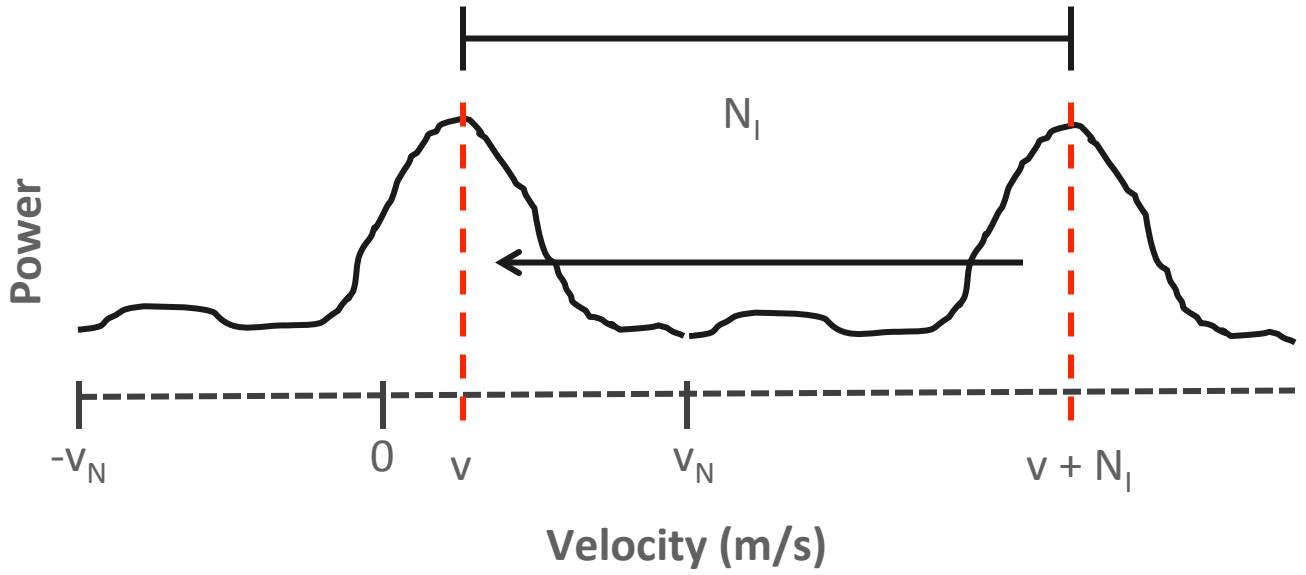
- Py-ART is a module for plotting, correcting and analyzing weather radar data.
- Development began to address the needs of ARM with the acquisition of multiple scanning cloud and precipitation radars.
- The project has since been expanded to work with a variety of weather radars and a wider user base including radar researchers and climate modelers.
- Available on GitHub as open source software, http://arm-doe.github.io/pyart/
- Highlights from 2014: BAMS article on open source weather radar software; ERAD short course; Convective/stratiform echo classification routine; improvement to the linear programming based phase processing; support for reading files from the CSU CHILL and NOAA airborne radars.

# Doppler Velocity Measurements by Radar



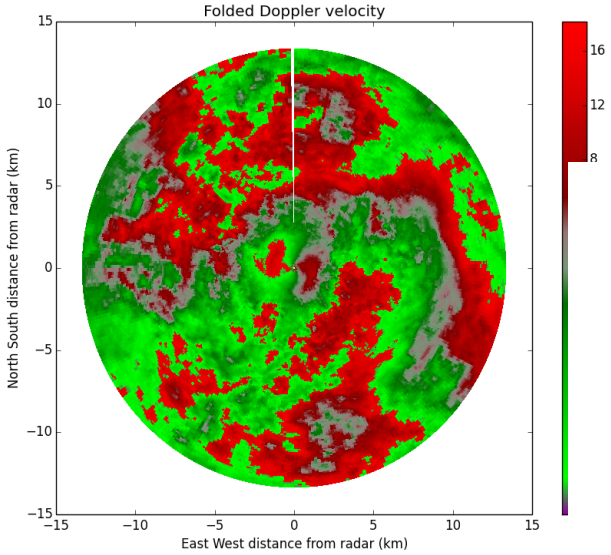



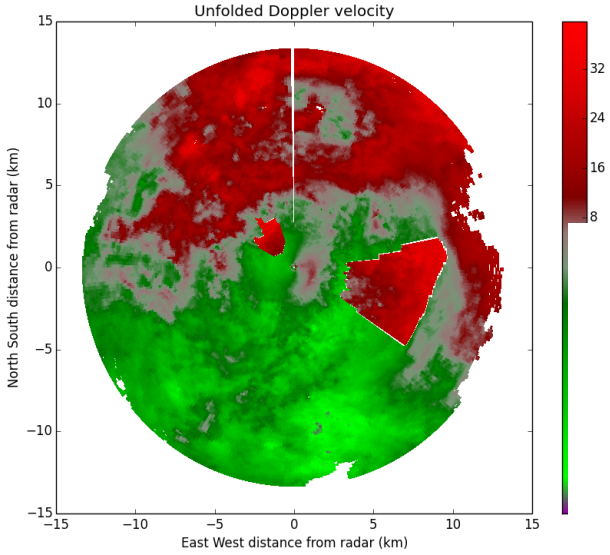
# Doppler Velocity Aliasing

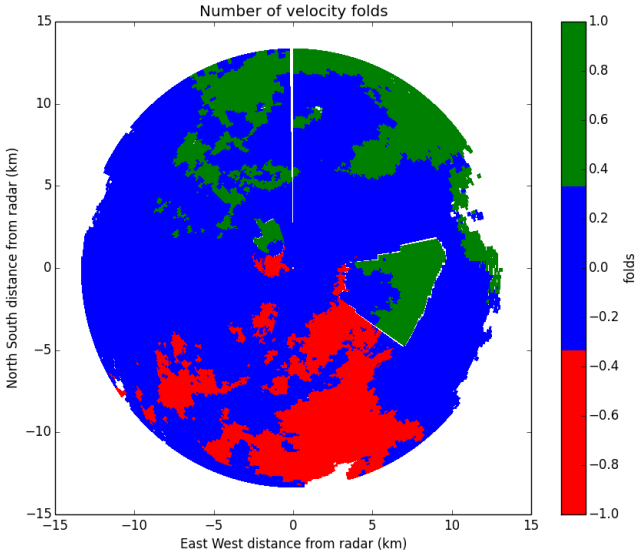



$$v' = v + n \times N_I \qquad \text{where } n = 0, \pm 1, \ldots$$

# Four-Dimensional Doppler Dealiasing




**A Real-Time Four-Dimensional Doppler Dealiasing Scheme**

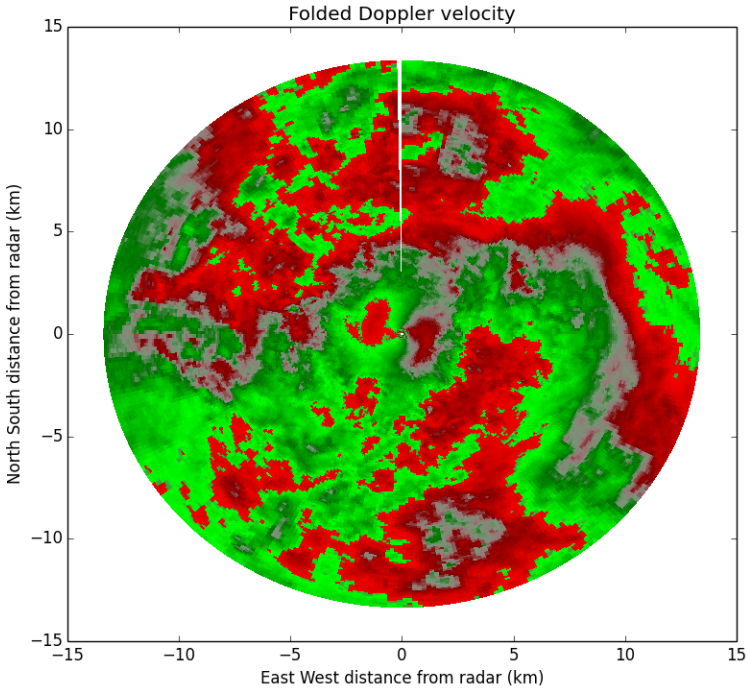

CURTIS N. JAMES* AND ROBERT A. HOUZE JR.

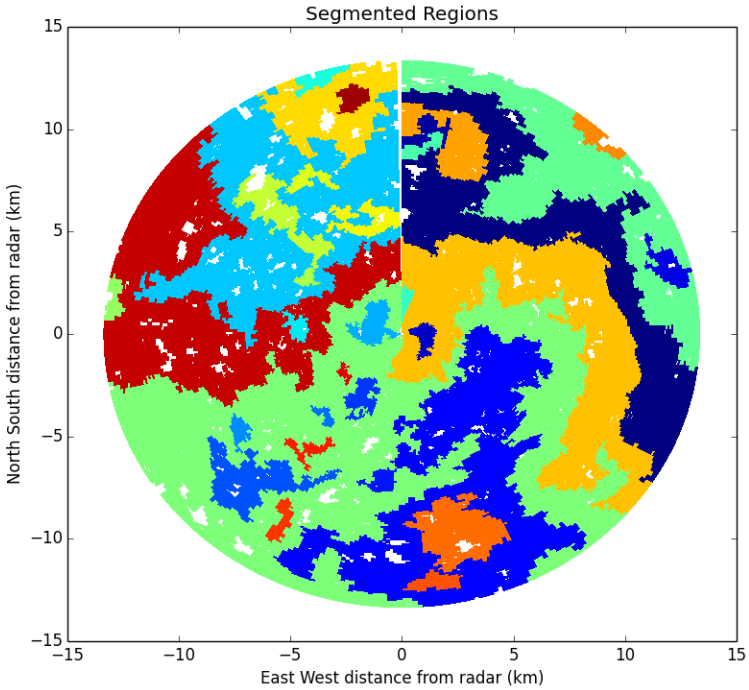
*Department of Atmospheric Sciences, University of Washington, Seattle, Washington*



J Tech, Vol 18, **2001**, pg. 1674.

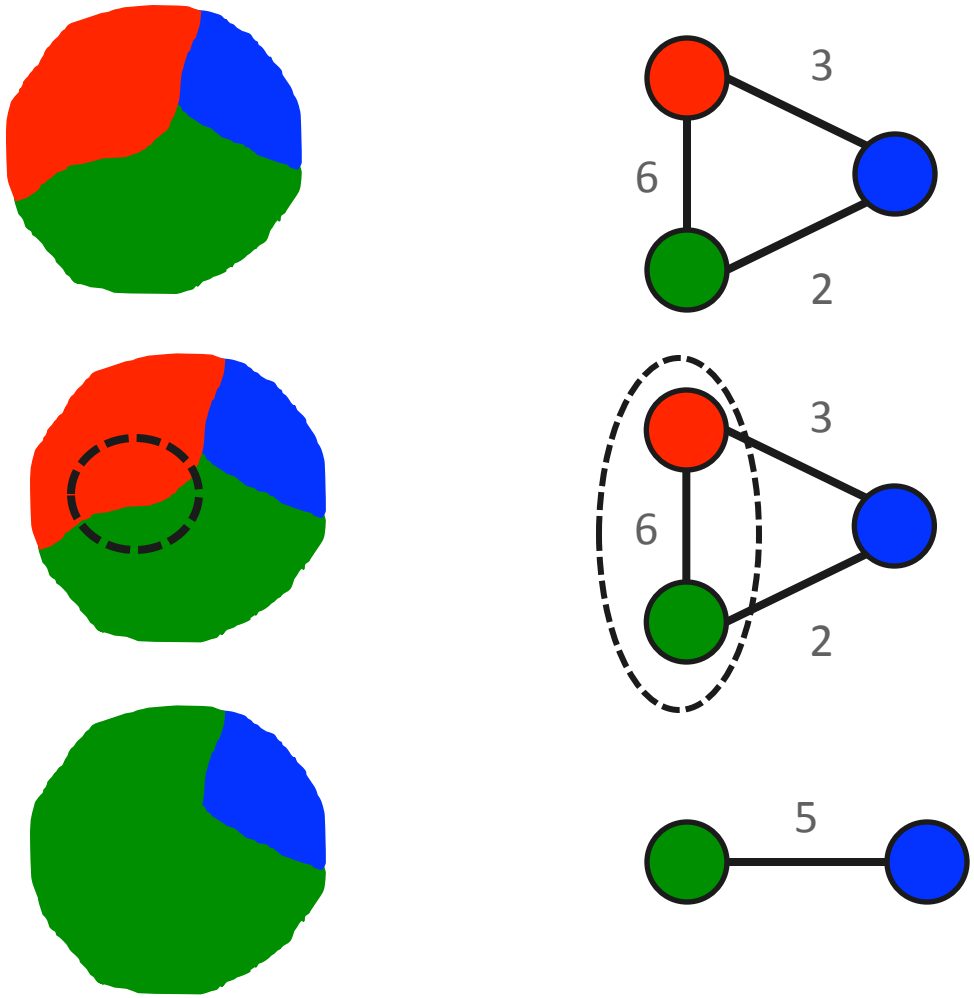
# Region Based Dealiasing : Algorithm Design I



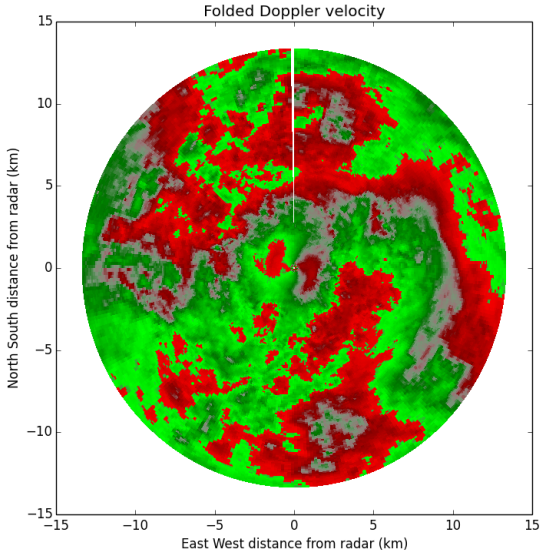



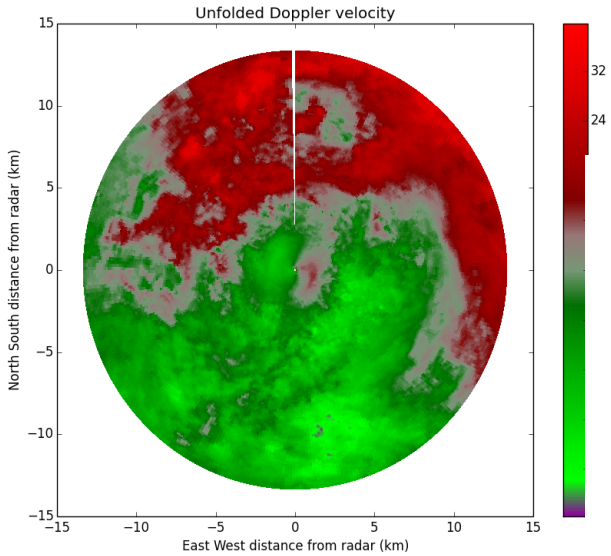

- Segmentation into regions based upon folded velocities.
- Uses scipy.ndimage module.

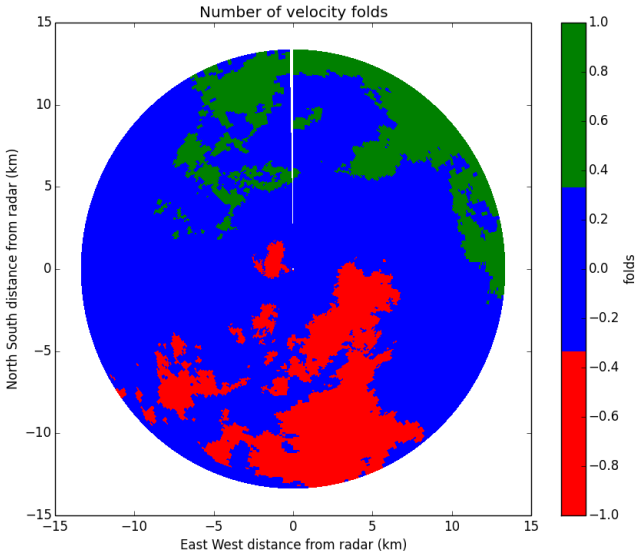
# Region Based Dealiasing : Algorithm Design II



# Region Based Dealiasing : Initial results



# Algorithm Optimization : Memory

https://pypi.python.org/pypi/memory_profiler

```
import dealias_lib
import numpy as np
import pyart

@profile
def dealias(original_data, nyquist):
    dealias_data = original_data.copy()
    labels, nfeatures = dealias_lib.find_segments(original_data)
    segment_sizes = dealias_lib.segment_sizes(labels, nfeatures)
    edge_sum, edge_count = dealias_lib.edge_sum_and_count(
        labels, nfeatures, original_data)

    upper_indices = np.triu_indices_from(edge_count, 1)
    for i in xrange(4):
        n = np.argmax(edge_count[upper_indices])
        x, y = upper_indices[0][n], upper_indices[1][n]
        if edge_count[x,y] == 0:
            break
        dealias_lib.combine_segments(
            x, y, labels, segment_sizes, edge_sum,
            edge_count, nyquist, dealias_data, step=None)

    return

if __name__ == "__main__":
    radar = pyart.io.read('105235.mdv')
    original_data = radar.fields['velocity']['data'].copy()
    original_data = original_data[:100]
    nyquist = radar.instrument_parameters['nyquist_velocity']['data'][0]
    dealias(original_data, nyquist)
```

# Algorithm Optimization : Memory

```
$ python -m memory_profiler script.py
Filename: script.py

Line #    Mem usage    Increment   Line Contents
================================================
     6  627.469 MiB    0.000 MiB   @profile
     7                             def dealias(original_data, nyquist):
     8  627.844 MiB    0.375 MiB       dealias_data = original_data.copy()
     9  516.098 MiB -111.746 MiB       labels, nfeatures = dealias_lib.find_segments(original_data)
    10  493.902 MiB  -22.195 MiB       segment_sizes = dealias_lib.segment_sizes(labels, nfeatures)
    11  493.902 MiB    0.000 MiB       edge_sum, edge_count = dealias_lib.edge_sum_and_count(
    12  776.426 MiB  282.523 MiB           labels, nfeatures, original_data)
    13
    14 2264.855 MiB 1488.430 MiB       upper_indices = np.triu_indices_from(edge_count, 1)
    15 2916.219 MiB  651.363 MiB       for i in xrange(4):
    16 2854.086 MiB  -62.133 MiB           n = np.argmax(edge_count[upper_indices])
    17 2854.086 MiB    0.000 MiB           x, y = upper_indices[0][n], upper_indices[1][n]
    18 2854.086 MiB    0.000 MiB           if edge_count[x,y] == 0:
    19                                         break
    20 2854.086 MiB    0.000 MiB           dealias_lib.combine_segments(
    21 2854.086 MiB    0.000 MiB               x, y, labels, segment_sizes, edge_sum,
    22 2916.219 MiB   62.133 MiB               edge_count, nyquist, dealias_data, step=None)
    23
    24 2916.219 MiB    0.000 MiB       return
```

# Algorithm Optimization : Memory

```
Line #    Mem usage    Increment   Line Contents
================================================
     6  627.492 MiB    0.000 MiB   @profile
     7                             def dealias(original_data, nyquist):
     8  627.867 MiB    0.375 MiB       dealias_data = original_data.copy()
     9  516.406 MiB -111.461 MiB       labels, nfeatures = dealias_lib.find_segments(original_data)
    10  494.211 MiB  -22.195 MiB       segment_sizes = dealias_lib.segment_sizes(labels, nfeatures)
    11  494.211 MiB    0.000 MiB       edge_sum, edge_count = dealias_lib.edge_sum_and_count(
    12  311.348 MiB -182.863 MiB           labels, nfeatures, original_data)
    13  302.684 MiB   -8.664 MiB       edge_count.setdiag(0)
    14
    15  313.363 MiB   10.680 MiB       for i in xrange(4):
    16  310.547 MiB   -2.816 MiB           argmax_idx = edge_count.data.argmax()
    17  310.547 MiB    0.000 MiB           x = np.nonzero(
    18  310.547 MiB    0.000 MiB                   edge_count.indptr <= argmax_idx)[0][-1]
    19  310.547 MiB    0.000 MiB           y = edge_count.indices[argmax_idx]
    20  310.547 MiB    0.000 MiB           if x > y:
    21                                         x,y = y,x
    22  310.547 MiB    0.000 MiB           if edge_count[x,y] == 0:
    23                                         break
    24  310.547 MiB    0.000 MiB           dealias_lib.combine_segments(
    25  310.547 MiB    0.000 MiB               x, y, labels, segment_sizes, edge_sum,
    26  313.363 MiB    2.816 MiB               edge_count, nyquist, dealias_data, step=None)
    27
    28  313.363 MiB    0.000 MiB       return
```

# Algorithm Optimization : Speed

```
import dealias_lib
import numpy as np
import pyart

def dealias(original_data, nyquist):
    ...
    ...
    ...
    return

if __name__ == "__main__":

    radar = pyart.io.read('single_folded_sweep.nc')
    original_data = radar.fields['velocity']['data'].copy()
    nyquist = radar.instrument_parameters['nyquist_velocity']['data'][0]

    # profile
    import cProfile
    cProfile.run("dealias(original_data, nyquist)",
                 "Profile.prof")

    # print out stats
    import pstats
    s = pstats.Stats("Profile.prof")
    s.strip_dirs().sort_stats("time").print_stats(20)
```

# Algorithm Optimization : Speed

```
$ python profile_script.py
Thu Jan  1 21:59:03 2015    Profile.prof

         7344341 function calls (7342262 primitive calls) in 75.729 seconds

   Ordered by: internal time
   List reduced from 192 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      693   48.491    0.070   48.491    0.070 {scipy.sparse._sparsetools.csr_sample_values}
     2772    5.209    0.002   15.494    0.006 compressed.py:702(_insert_many)
   494518    4.260    0.000    4.260    0.000 {numpy.core.multiarray.concatenate}
   483430    4.173    0.000    6.971    0.000 arraysetops.py:93(unique)
     6920    2.386    0.000    2.386    0.000 {scipy.sparse._sparsetools.get_csr_submatrix}
    55388    1.642    0.000    1.642    0.000 {method 'reduce' of 'numpy.ufunc' objects}
     8304    1.425    0.000    1.425    0.000 {scipy.sparse._sparsetools.csr_sample_offsets}
     8304    1.301    0.000   18.600    0.002 compressed.py:649(_set_many)
   483430    0.564    0.000    0.564    0.000 {method 'flatten' of 'numpy.ndarray' objects}
   486202    0.540    0.000    0.540    0.000 {method 'argsort' of 'numpy.ndarray' objects}
    17998    0.437    0.000    0.938    0.000 compressed.py:126(check_format)
    42919    0.362    0.000    0.362    0.000 {method 'astype' of 'numpy.ndarray' objects}
      692    0.291    0.000   24.949    0.036 dealias_lib.py:156(combine_segments)
   282452    0.279    0.000    0.279    0.000 {numpy.core.multiarray.array}
    44308    0.225    0.000    1.317    0.000 sputils.py:132(get_index_dtype)
    16608    0.224    0.000    0.590    0.000 stride_tricks.py:36(broadcast_arrays)
    47080    0.206    0.000    0.206    0.000 getlimits.py:244(__init__)
    22144    0.181    0.000    0.283    0.000 stride_tricks.py:22(as_strided)
        1    0.155    0.155    0.214    0.214 dealias_lib.py:97(edge_sum_and_count)
  1966244    0.153    0.000    0.153    0.000 {method 'append' of 'list' objects}
```

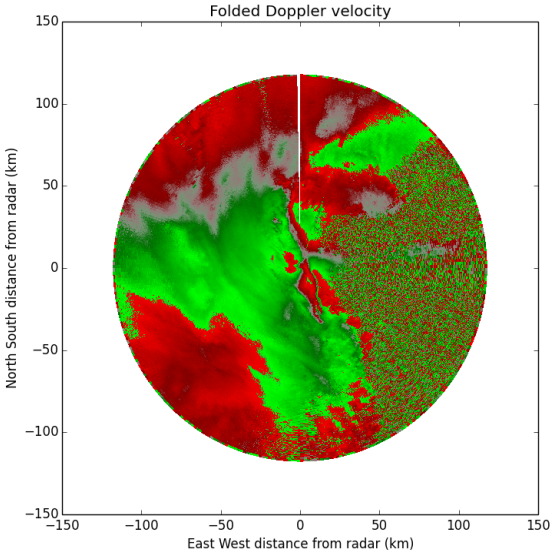
# Algorithm Optimization : Speed

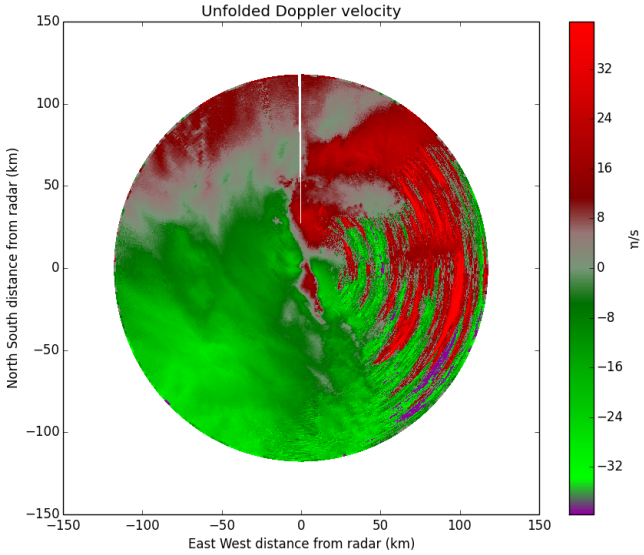
```
import dealias_lib
from trackers import RegionTracker, EdgeTracker
import pyart

def dealias(original_data, nyquist):

    labels, nfeatures = dealias_lib.find_segments(original_data)
    segment_sizes = dealias_lib.segment_sizes(labels, nfeatures)
    edge_sum, edge_count = dealias_lib.edge_sum_and_count(
        labels, nfeatures, original_data)

    region_tracker = RegionTracker(segment_sizes)
    edge_tracker = EdgeTracker(edge_sum, edge_count, nyquist)
    for i in xrange(80000):
        if dealias_lib.combine_segments(region_tracker, edge_tracker):
            break

    dealias_data = original_data.copy()
    for i in range(nfeatures+1):
        nwrap = region_tracker.unwrap_number[i]
        dealias_data[labels == i] += nwrap * nyquist
    return dealias_data

if __name__ == "__main__":

    radar = pyart.io.read('single_folded_sweep.nc')
    …
```

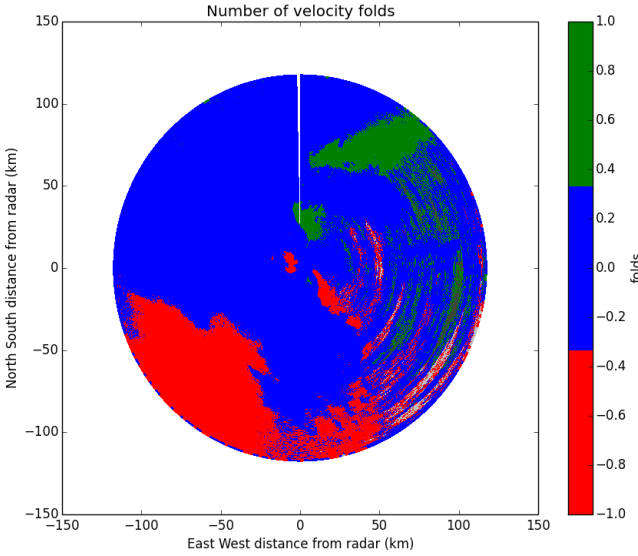
# Algorithm Optimization : Speed

```
$ python profile_script.py
Thu Jan  1 22:04:25 2015    Profile.prof

         199063 function calls in 0.626 seconds

   Ordered by: internal time
   List reduced from 124 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.129    0.129    0.626    0.626 profile_script_2.py:6(dealias)
     2505    0.117    0.000    0.144    0.000 compressed.py:772(_get_single_element)
        1    0.112    0.112    0.147    0.147 dealias_lib.py:97(edge_sum_and_count)
     1670    0.060    0.000    0.060    0.000 {method 'remove' of 'list' objects}
     5013    0.038    0.000    0.045    0.000 sputils.py:188(isintlike)
    54001    0.025    0.000    0.033    0.000 index_tricks.py:490(__next__)
     2526    0.014    0.000    0.014    0.000 {method 'reduce' of 'numpy.ufunc' objects}
     2505    0.012    0.000    0.031    0.000 sputils.py:244(_unpack_index)
      693    0.010    0.000    0.012    0.000 trackers.py:206(pop_edge)
    27583    0.009    0.000    0.009    0.000 {isinstance}
      692    0.009    0.000    0.073    0.000 trackers.py:102(merge_nodes)
    54001    0.008    0.000    0.008    0.000 {next}
     2505    0.008    0.000    0.012    0.000 sputils.py:310(_check_boolean)
      304    0.008    0.000    0.008    0.000 trackers.py:193(unwrap_node)
        1    0.007    0.007    0.234    0.234 trackers.py:58(__init__)
     2505    0.007    0.000    0.226    0.000 csr.py:197(__getitem__)
     2505    0.005    0.000    0.005    0.000 {method 'compress' of 'numpy.ndarray' objects}
      304    0.005    0.000    0.005    0.000 trackers.py:42(unwrap_node)
      693    0.004    0.000    0.107    0.000 dealias_lib.py:159(combine_segments)
     2505    0.004    0.000    0.004    0.000 sputils.py:272(_check_ellipsis)
```

# Region Based Dealiasing : Final results




Time required to dealiasing a full sweep: ~2.5 minutes

# Conclusions

- We have developed and implemented in Python a novel new algorithm for Doppler velocity dealiasing based upon unfolding regions of similar velocities.
- The *memory_profiler* module provides a list of memory usage within a Python function. https://pypi.python.org/pypi/memory_profiler
- The *cProfile* module in the Python standard library provides statistics on how often and how long portions of a Python program are executed.
- By optimizing the region based dealiasing algorithm for speed and memory usage, we are able to dealias complete radar sweeps in approximately 2.5 minutes.

**Thank you for your attention.**

**Questions?**