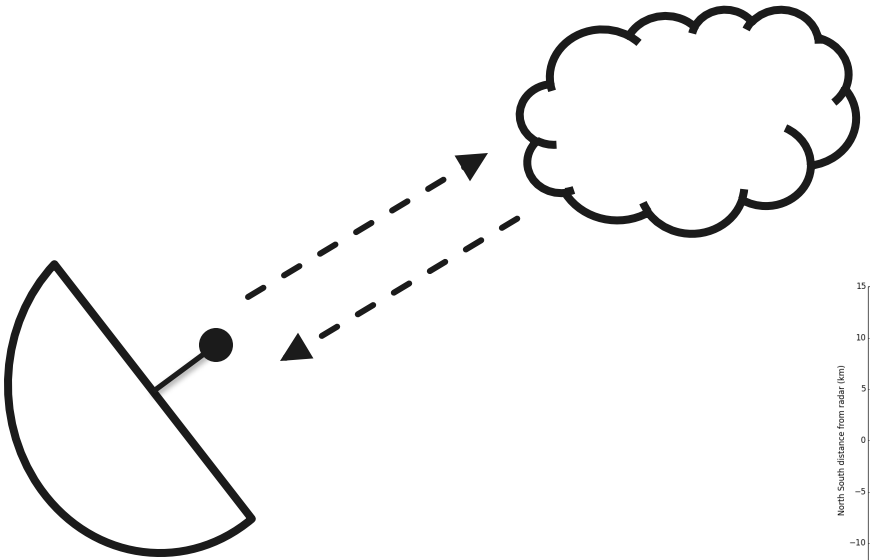
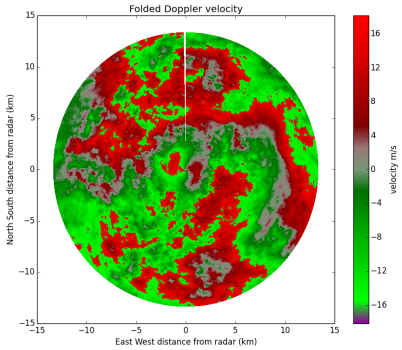
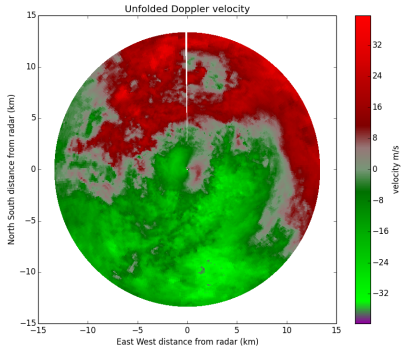
Profiling Python code to improve memory usage and execution time

Jonathan Helmus, Argonne National Laboratory







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 visualizing, correcting and analyzing **weather radar data** using packages from the scientific Python stack.

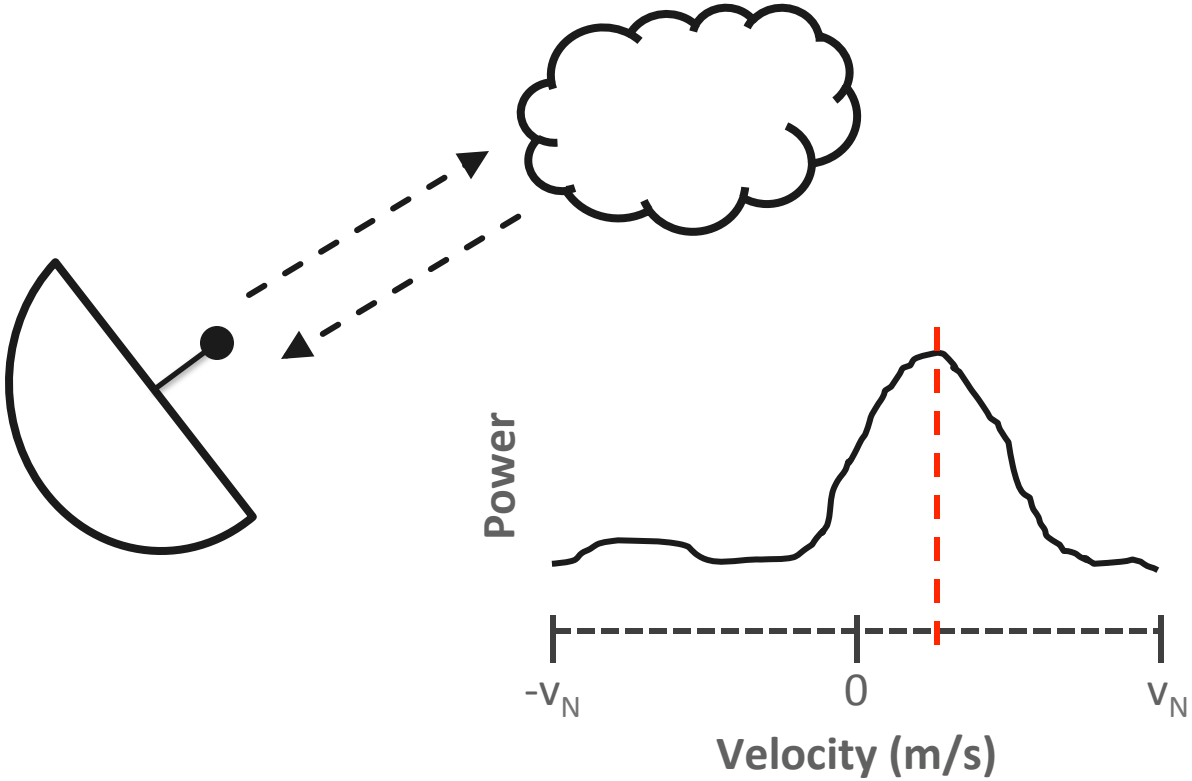
- Development began to address the needs of the **ARM** program with the acquisition of **multiple scanning cloud and precipitation radars**.

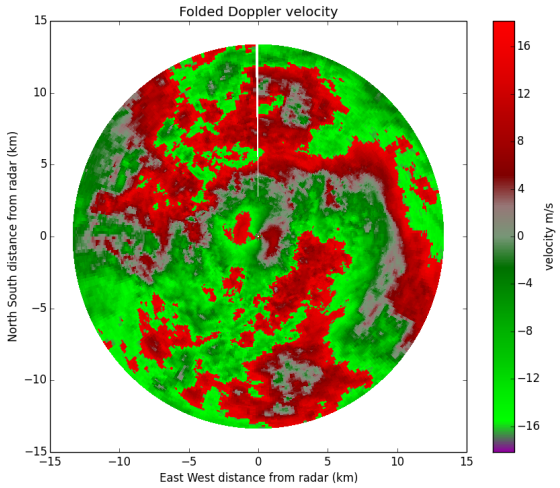
- The project has since been expanded to work with a **variety of weather radars,** including NEXRAD and TDWR radars, and a wide user base including radar researchers, weather enthusiast and climate modelers.

- Available on GitHub as **open source software** under a BSD license, arm-doe.github.io/pyart/.

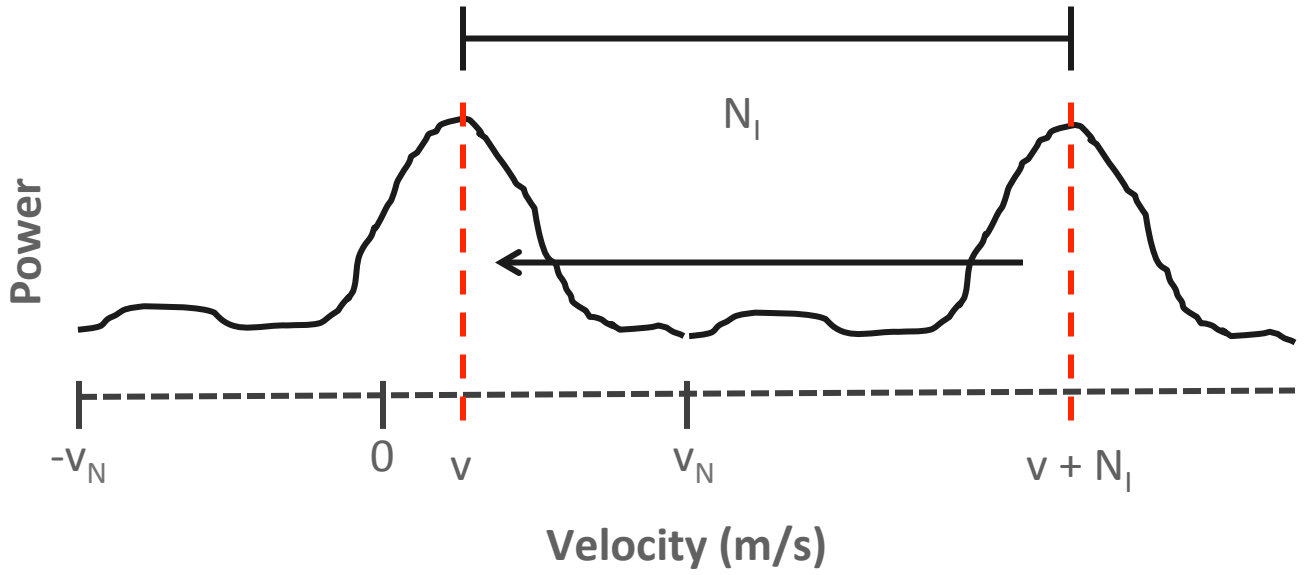
- Conda packages are available at binstar.org/jjhelmus for Windows and OS X.

Doppler Velocity Measurements by Radar



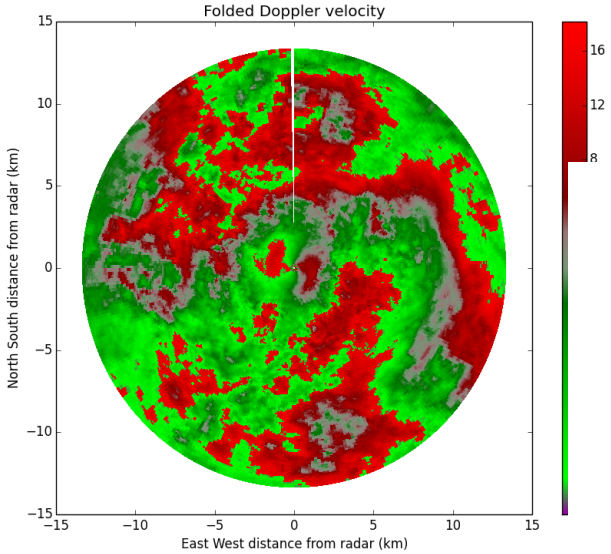


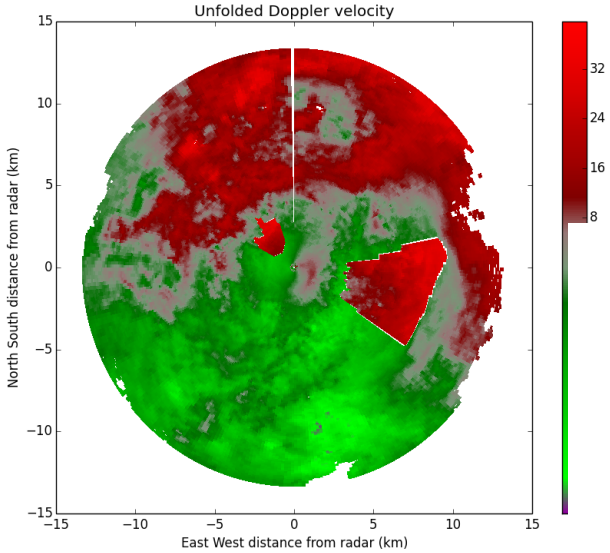
Doppler Velocity Aliasing

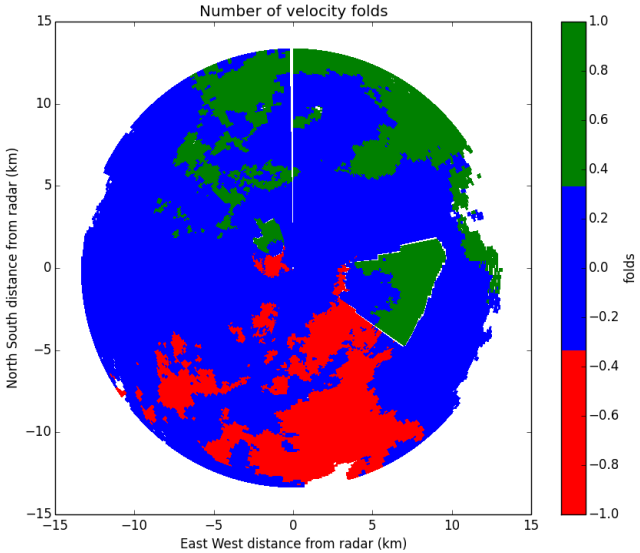


$$v' = v + n \times N_I \qquad \text{where } n = 0, \pm 1, \ldots$$

Four-Dimensional Doppler Dealiasing







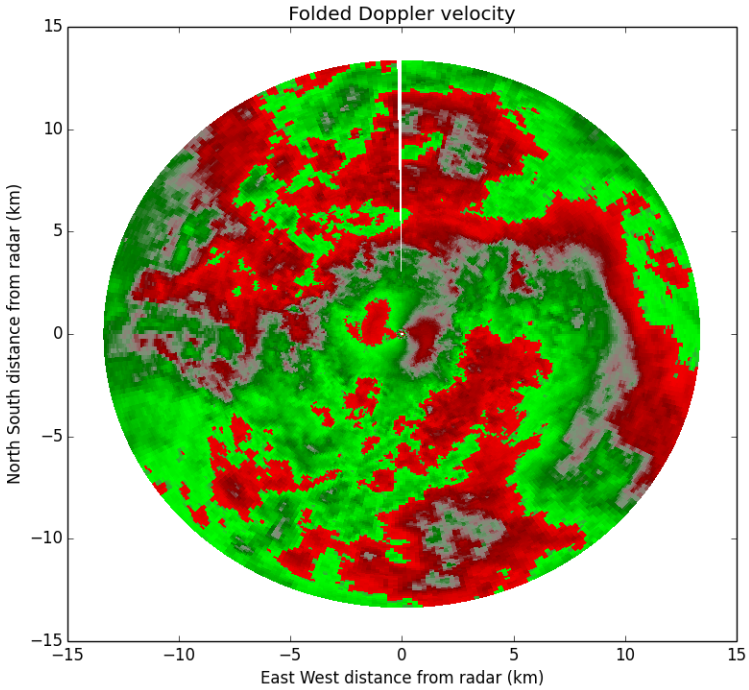
A Real-Time Four-Dimensional Doppler Dealiasing Scheme

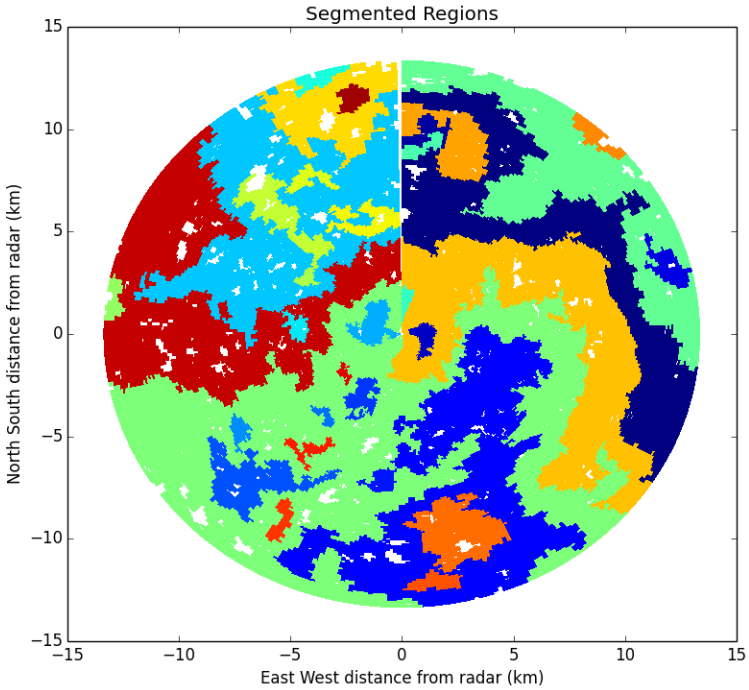
CURTIS N. JAMES* AND ROBERT A. HOUZE JR.

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

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

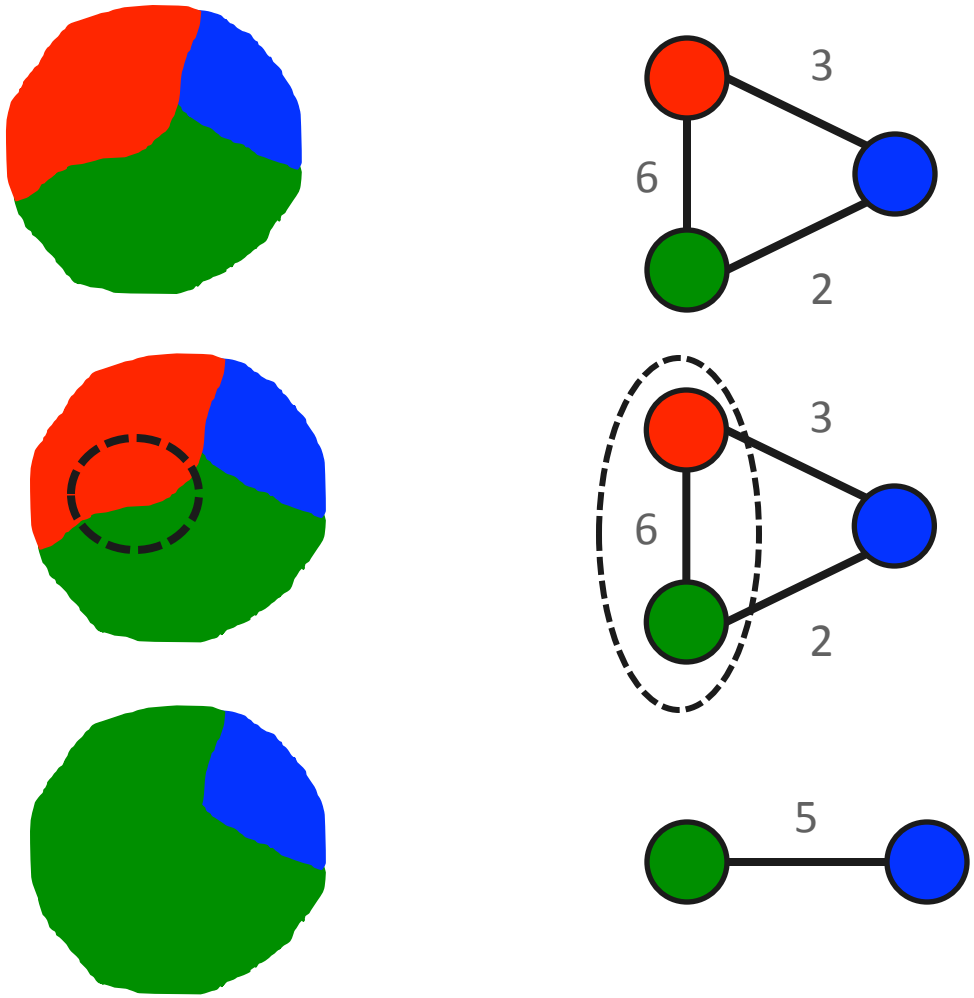
Region Based Dealiasing : Algorithm Design I



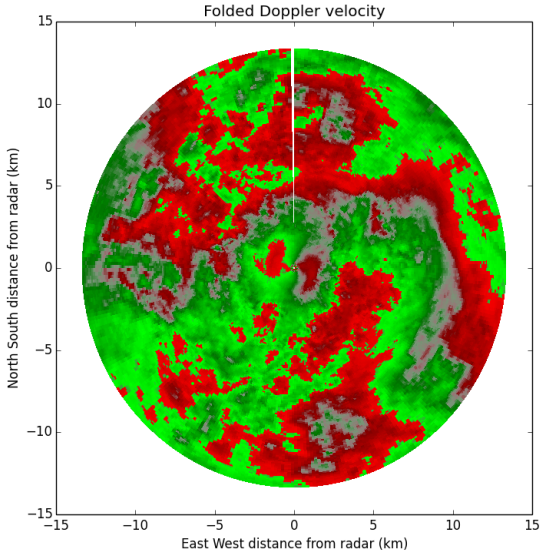


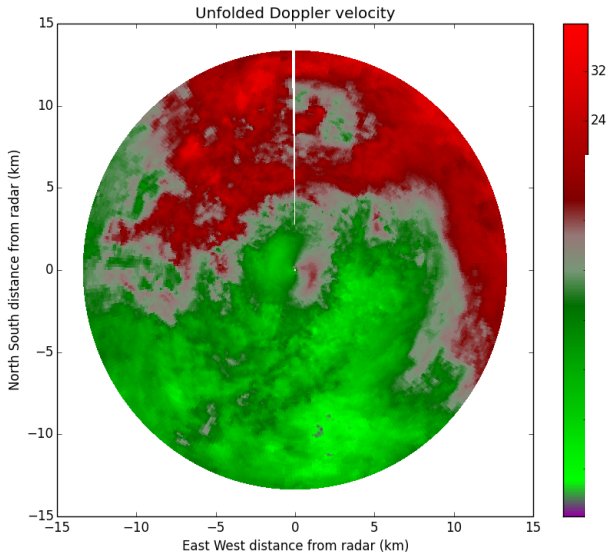
- Segmentation into regions based upon folded velocities.
- Uses scipy.ndimage module.

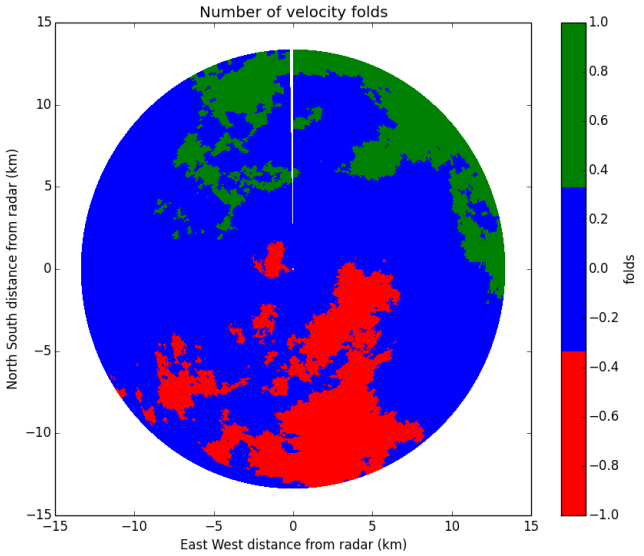
Region Based Dealiasing : Algorithm Design II



Region Based Dealiasing : Initial results







Memory usage reduction with *memory_profiler*

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)
```

Memory usage reduction with *memory_profiler*

```
$ 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
```

Memory usage reduction with *memory_profiler*

```
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
```

Run time optimization with *cProfile*

```
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)
```

Run time optimization with *cProfile*

```
$ 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}
```

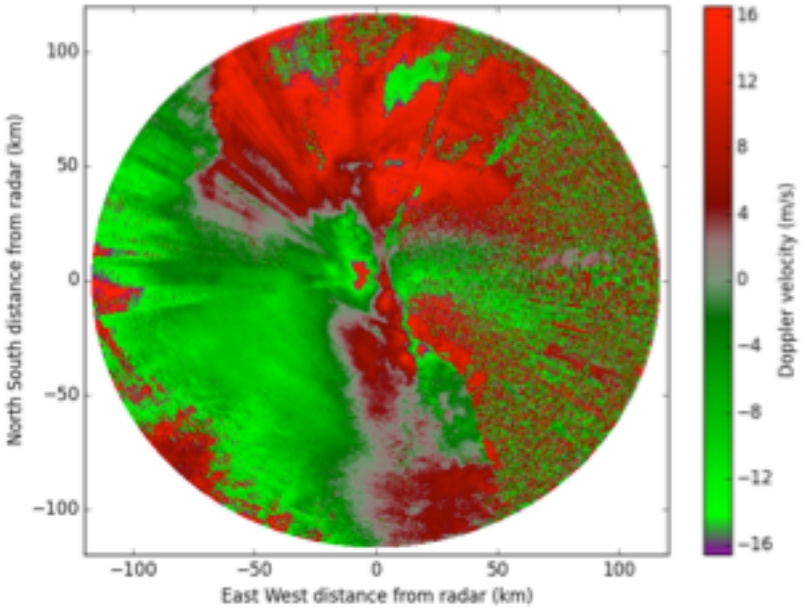
Run time optimization with *cProfile*

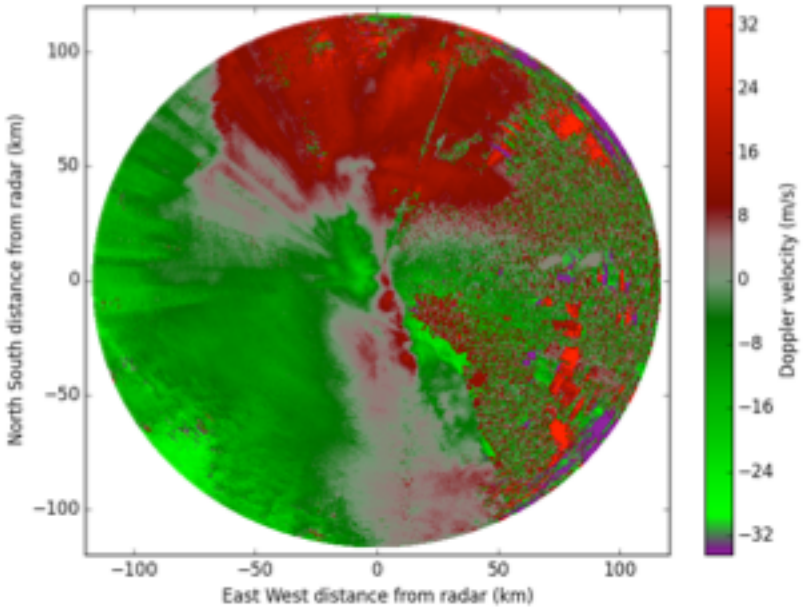
```
import dealias_lib
from trackers import RegionTracker, EdgeTracker
import pyart

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')
    …
```

Run time optimization with *cProfile*

```
$ 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)
```

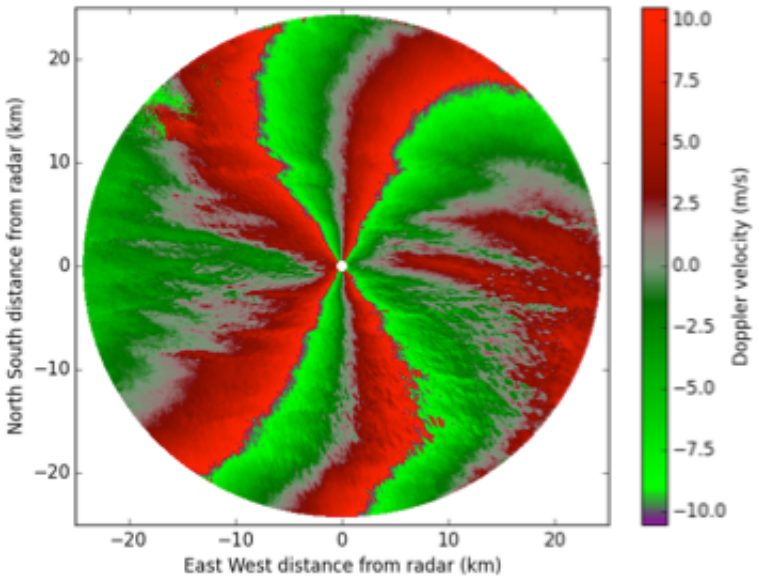
Sample results: ARM CSAPR

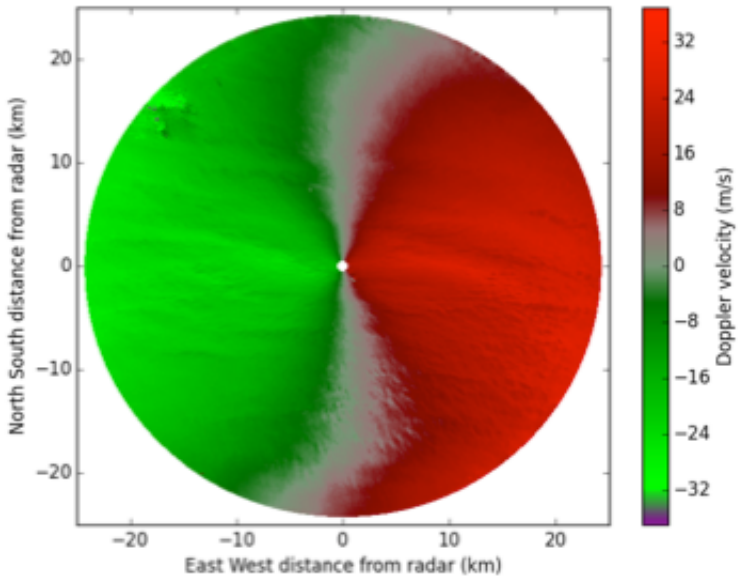




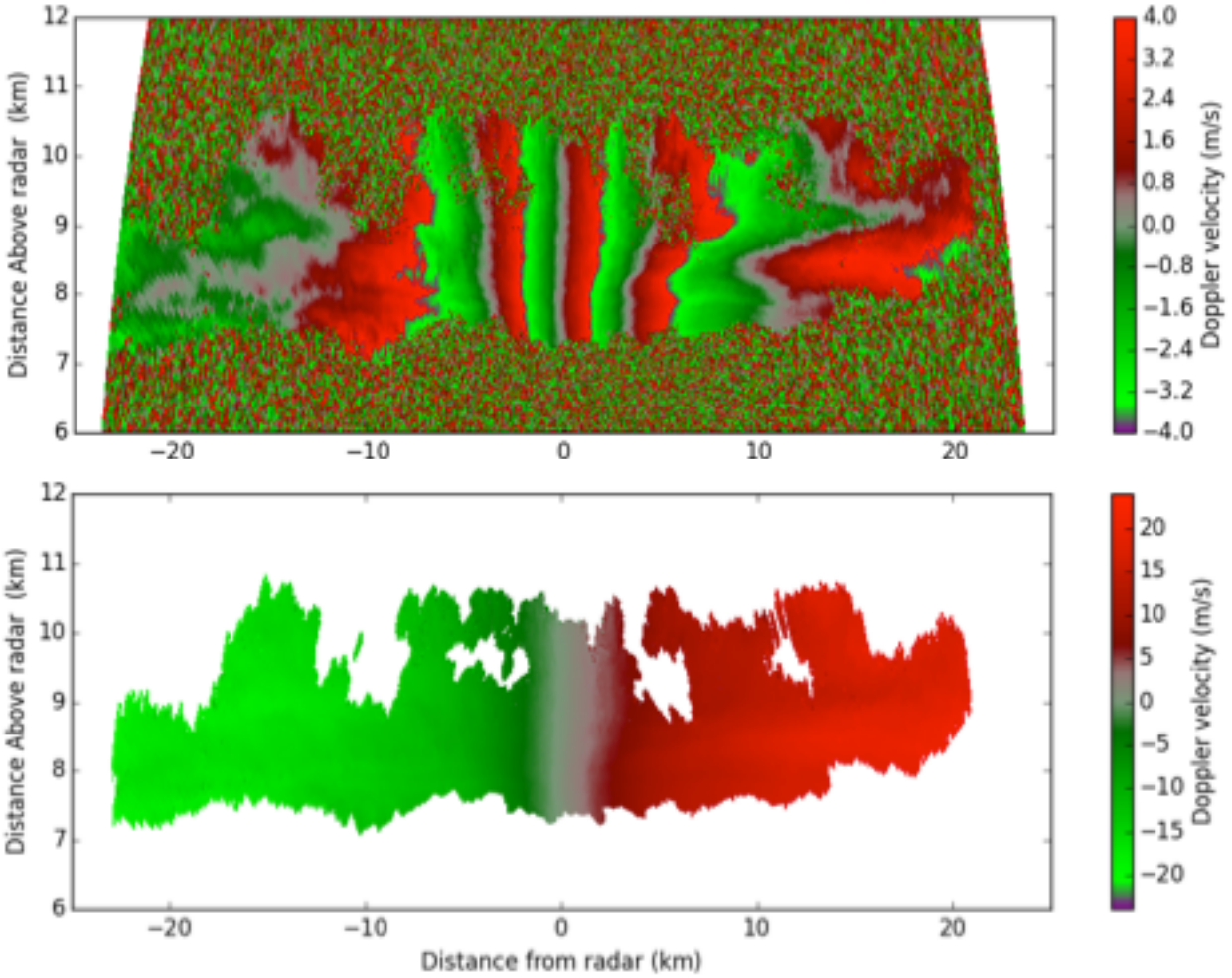
Time required to dealias a full radar volume: ~1.5 minutes

Sample results: ARM KaSACR





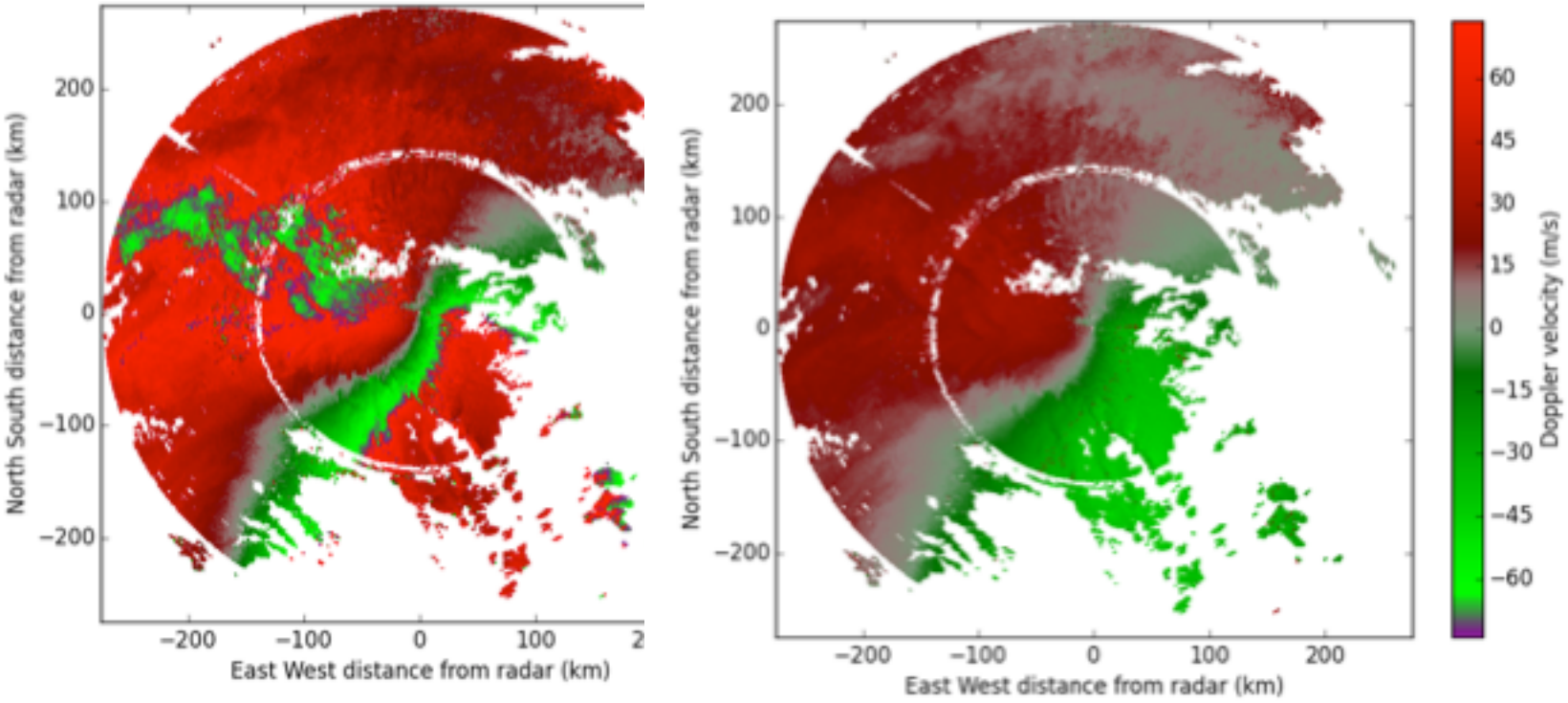
Sample results: ARM WSACR



Profiling Python code, Jonathan Helmus, SEA Conference 2015, Boulder, CO

ARM CLIMATE RESEARCH FACILITY

Sample results: NEXRAD



Profiling Python code, Jonathan Helmus, SEA Conference 2015, Boulder, CO

ARM CLIMATE RESEARCH FACILITY

Run time optimization with *line_profiler*

https://pypi.python.org/pypi/line_profiler/

Source code

```
@profile
def dealias_region_based(
        radar, interval_splits=3, interval_limits=None,
        skip_between_rays=2, skip_along_ray=2, centered=True,
        nyquist_vel=None, gatefilter=None, rays_wrap_around=None,
        keep_original=True, vel_field=None, corr_vel_field=None, **kwargs):
    """
    Dealias Doppler velocities using a region based algorithm.
...
```

Script

```
import pyart
radar = pyart.io.read('105235.mdv')
pyart.correct.dealias_region_based(radar, keep_original=False,
                                   skip_along_ray=100, skip_between_rays=100)
```

Run time optimization with *line_profiler*

```
$ kernprof.py -l -v script.py
Timer unit: 1e-06 s

File: /home/jhelmus/dev/pyart/pyart/correct/region_dealias.py
Function: dealias_region_based at line 57
Total time: 46.1975 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    57                                           @profile
    58                                           def dealias_region_based(
...
   156                                               # extract sweep data
   157        17         3116    183.3      0.0          sdata = vdata[sweep_slice].copy()   # is a copy needed here?
   158        17           38      2.2      0.0          scorr = data[sweep_slice]
   159        17           28      1.6      0.0          sfilter = gfilter[sweep_slice]
   160
   161                                               # find regions in original data
   162        17       278528  16384.0      0.6          labels, nfeatures = _find_regions(sdata, sfilter, interval_limits)
   163        17           40      2.4      0.0          edge_sum, edge_count, region_sizes = _edge_sum_and_count(
   164        17           21      1.2      0.0              labels, nfeatures, sdata, rays_wrap_around, skip_between_rays,
   165        17       163290   9605.3      0.4              skip_along_ray)
   166
   167                                               # find the number of folds in the regions
   168        17        57276   3369.2      0.1          region_tracker = _RegionTracker(region_sizes)
   169        17     33134917 1949112.8     71.7          edge_tracker = _EdgeTracker(edge_sum, edge_count, nyquist_interval)
   170     48314        56051      1.2      0.1          while True:
   171     48314      6810804    141.0     14.7              if _combine_regions(region_tracker, edge_tracker):
   172        17           25      1.5      0.0                  break
```

Run time optimization with *line_profiler*

```
$ kernprof.py -l -v script.py

Total time: 13.4539 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
  ...
  171
  172                                           # find the number of folds in the regions
  173        17        57249   3367.6      0.4  region_tracker = _RegionTracker(region_sizes)
  174        17           30      1.8      0.0  edge_tracker = _EdgeTracker(indices, edge_count, velos,
  175        17       625889  36817.0      4.7                              nyquist_interval, nfeatures+1)
  176     48314         57803      1.2      0.4  while True:
  177     48314      6622303    137.1     49.2      if _combine_regions(region_tracker, edge_tracker):
  178        17           24      1.4      0.0          break
  179
  180                                           # center sweep if requested, determine a global sweep unfold number
  181                                           # so that the average number of gate folds is zero.
  182        17           17      1.0      0.0  if centered:
  183        17          329     19.4      0.0      gates_dealiased = region_sizes.sum()
  184        17           26      1.5      0.0      total_folds = np.sum(
  185        17          564     33.2      0.0          region_sizes * region_tracker.unwrap_number[1:])
  186        17           71      4.2      0.0      sweep_offset = int(round(float(total_folds) / gates_dealiased))
  187        17           22      1.3      0.0      if sweep_offset != 0:
  188         3           22      7.3      0.0          region_tracker.unwrap_number -= sweep_offset
  189
  190                                           # dealias the data using the fold numbers
  191                                           # start from label 1 to skip masked region
  192     48334        58536      1.2      0.4  for i in range(1, nfeatures+1):
  193     48317        68051      1.4      0.5      nwrap = region_tracker.unwrap_number[i]
  194     48317       182673      3.8      1.4      if nwrap != 0:
  195     16432      5252213    319.6     39.0          scorr[labels == i] += nwrap * nyquist_interval
```

Run time optimization with *line_profiler*

```
$ kernprof.py -l -v script.py

Total time: 8.20599 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
  ...
  171
  172                                           # find the number of folds in the regions
  173        17        60067   3533.4      0.7          region_tracker = _RegionTracker(region_sizes)
  174        17           32      1.9      0.0          edge_tracker = _EdgeTracker(indices, edge_count, velos,
  175        17       640319  37665.8      7.8                                      nyquist_interval, nfeatures+1)
  176     48314        60980      1.3      0.7          while True:
  177     48314      6872862    142.3     83.8              if _combine_regions(region_tracker, edge_tracker):
  178        17           21      1.2      0.0                  break
  179
  180                                           # center sweep if requested, determine a global sweep unfold number
  181                                           # so that the average number of gate folds is zero.
  182        17           21      1.2      0.0          if centered:
  183        17          376     22.1      0.0              gates_dealiased = region_sizes.sum()
  184        17           29      1.7      0.0              total_folds = np.sum(
  185        17          687     40.4      0.0                  region_sizes * region_tracker.unwrap_number[1:])
  186        17           76      4.5      0.0              sweep_offset = int(round(float(total_folds) / gates_dealiased))
  187        17           25      1.5      0.0              if sweep_offset != 0:
  188         3           23      7.7      0.0                  region_tracker.unwrap_number -= sweep_offset
  189
  190                                           # dealias the data using the fold numbers
  191        17        14154    832.6      0.2          nwrap = np.take(region_tracker.unwrap_number, labels)
  192        17        13471    792.4      0.2          scorr += nwrap * nyquist_interval
```

Run time optimization using *Cython*

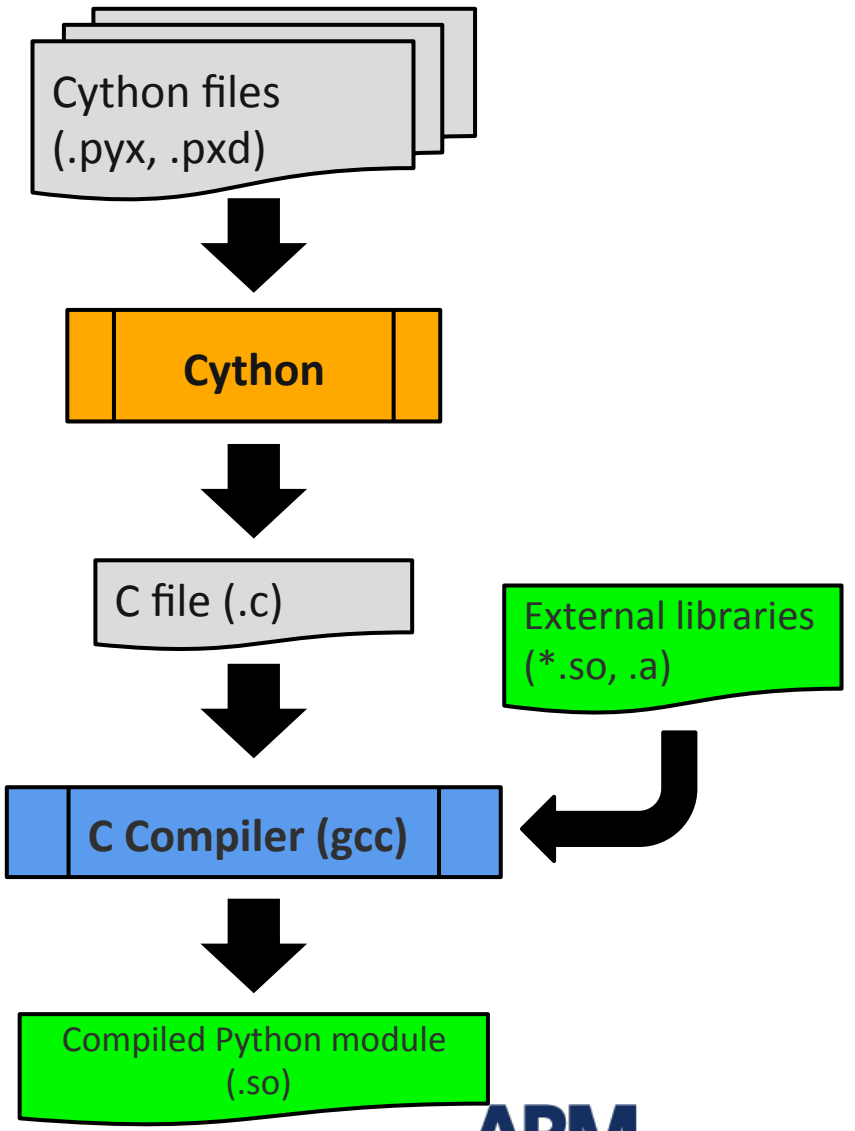
```
$ kernprof.py -l -v script.py

Total time: 153.644 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
  ...
   159                                           # extract sweep data
   160        17         3170    186.5      0.0  sdata = vdata[sweep_slice].copy()   # is a copy needed here?
   161        17           38      2.2      0.0  scorr = data[sweep_slice]
   162        17           28      1.6      0.0  sfilter = gfilter[sweep_slice]
   163
   164                                           # find regions in original data
   165        17       284246  16720.4      0.2  labels, nfeatures = _find_regions(sdata, sfilter, interval_limits)
   166        17           43      2.5      0.0  edge_sum, edge_count, region_sizes = _edge_sum_and_count(
   167        17           19      1.1      0.0      labels, nfeatures, sdata, rays_wrap_around, skip_between_rays,
   168        17    107524384 6324963.8     70.0       skip_along_ray)
   169
   170                                           # find the number of folds in the regions
   171        17        58564   3444.9      0.0  region_tracker = _RegionTracker(region_sizes)
   172        17     33254618 1956154.0     21.6   edge_tracker = _EdgeTracker(edge_sum, edge_count, nyquist_interval)
   173     48314        56613      1.2      0.0  while True:
   174     48314      6727992    139.3      4.4      if _combine_regions(region_tracker, edge_tracker):
   175        17           19      1.1      0.0          break
```

Run time optimization using *Cython*

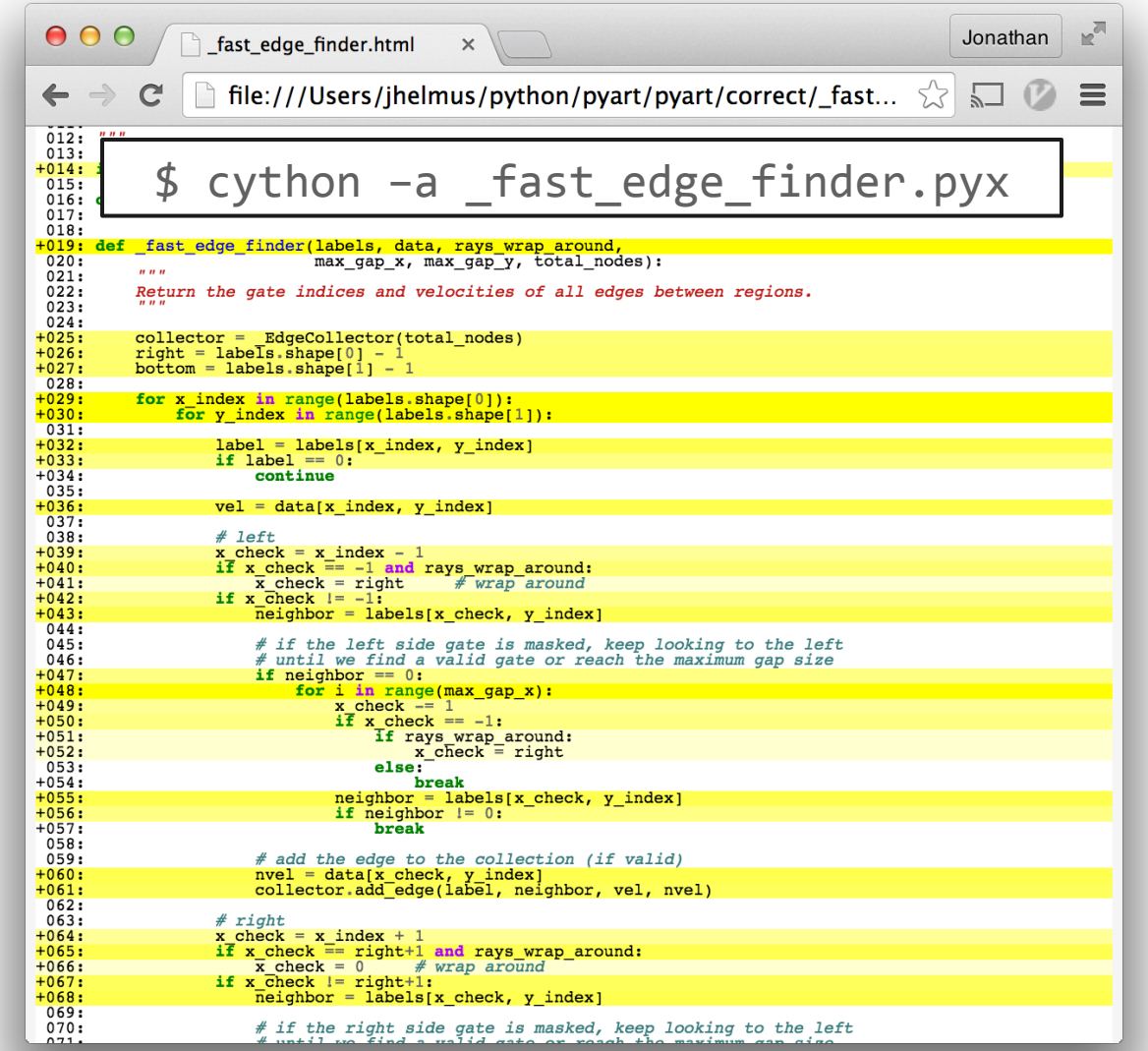
Python code, especially when it contains nested loops, can be slow. One method to speed up the run time is to use Cython.

Cython

- Python to C code translator.
- Generates a Python extension module.
- Can be used to speed up Python code by adding static type information.
- Also can be used to interact with C/C++ function and classes in external libraries.

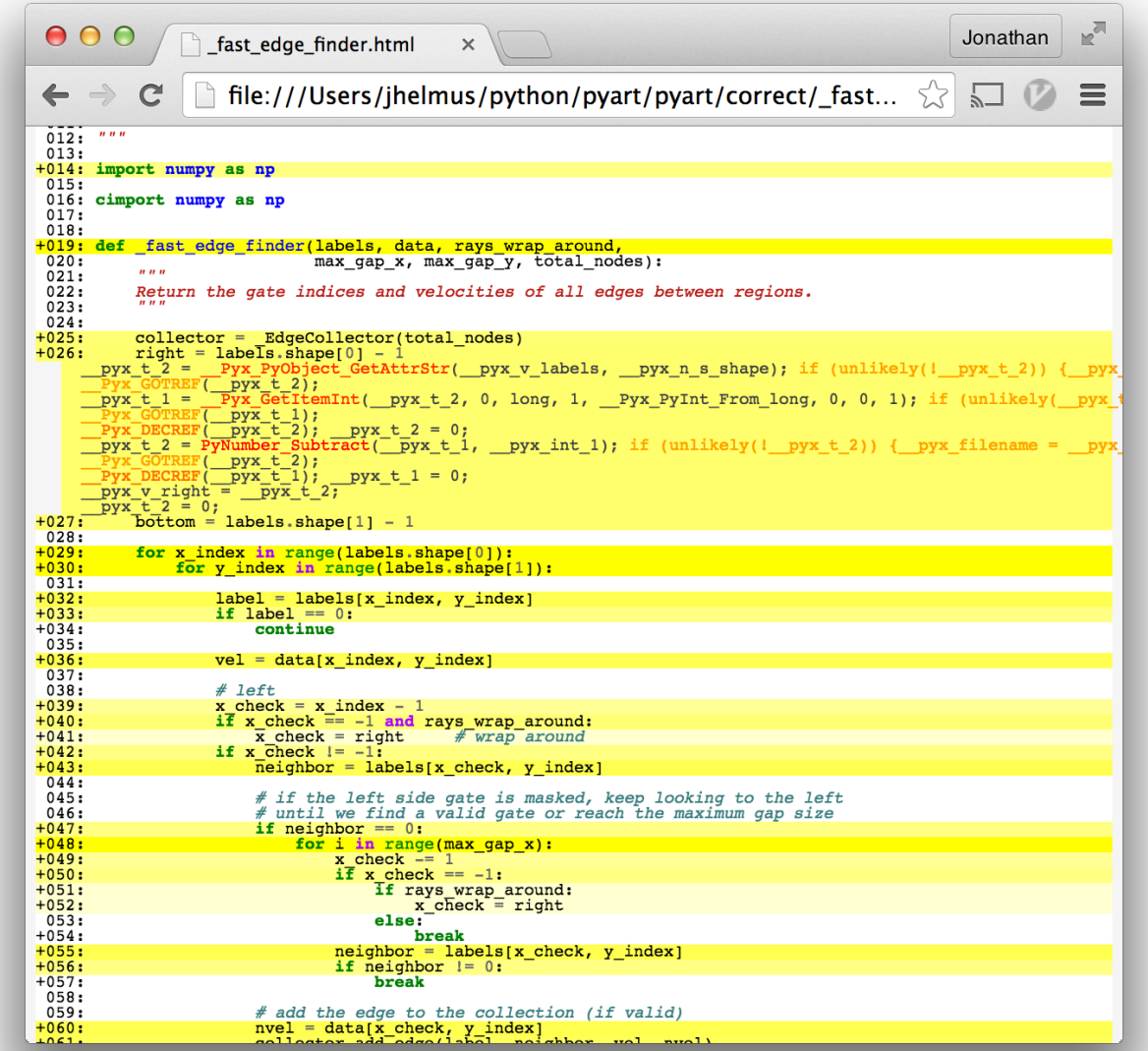


Optimizing *Cython* code

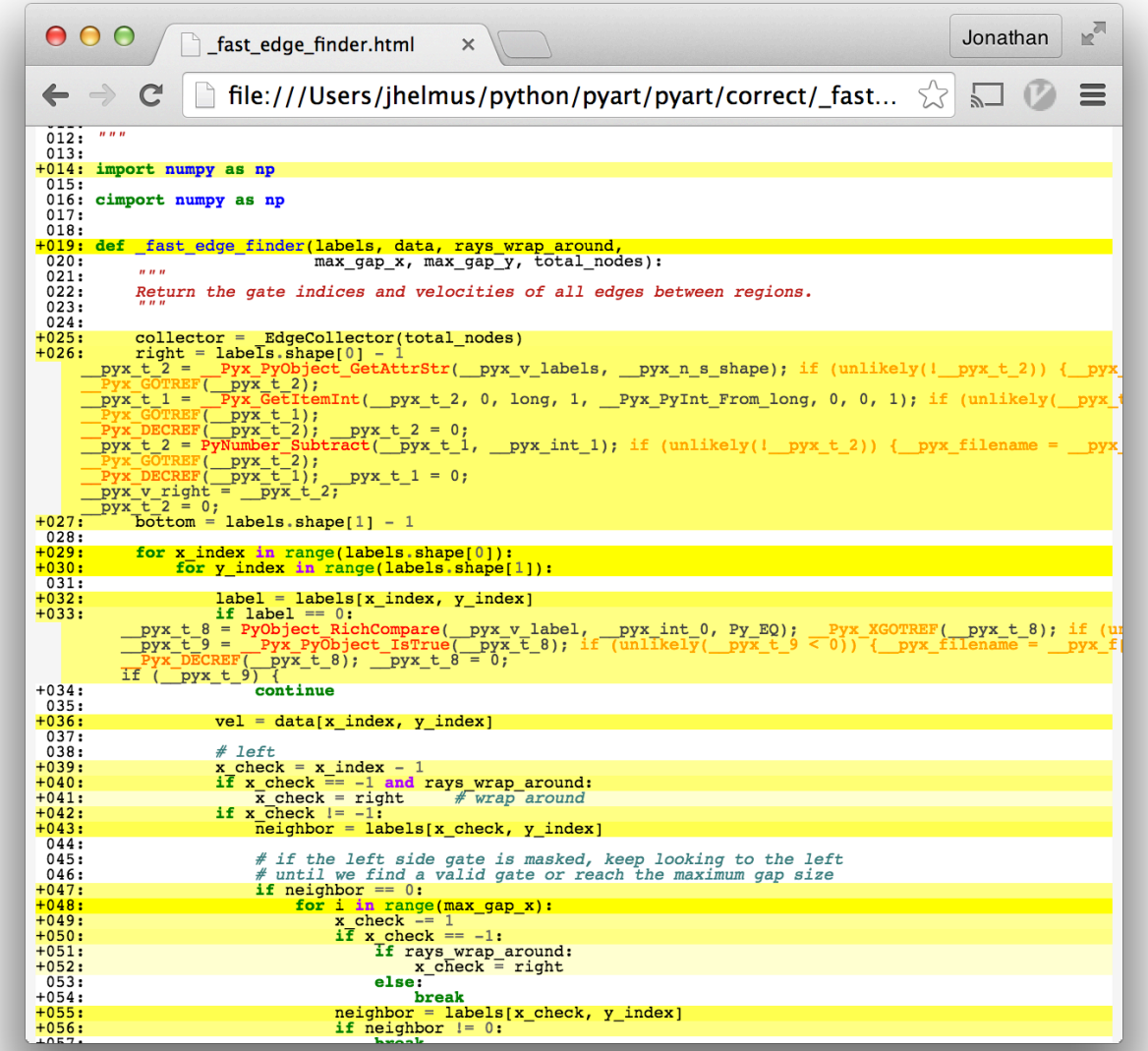



```
012: """
013:
+014: 
015:
016: 
017:
018:
+019: def _fast_edge_finder(labels, data, rays_wrap_around,
020:                       max_gap_x, max_gap_y, total_nodes):
021:     """
022:     Return the gate indices and velocities of all edges between regions.
023:     """
024:
+025:     collector = _EdgeCollector(total_nodes)
+026:     right = labels.shape[0] - 1
+027:     bottom = labels.shape[1] - 1
028:
+029:     for x_index in range(labels.shape[0]):
+030:         for y_index in range(labels.shape[1]):
031:
+032:             label = labels[x_index, y_index]
+033:             if label == 0:
+034:                 continue
035:
+036:             vel = data[x_index, y_index]
037:
038:             # left
+039:             x_check = x_index - 1
+040:             if x_check == -1 and rays_wrap_around:
+041:                 x_check = right     # wrap around
+042:             if x_check != -1:
+043:                 neighbor = labels[x_check, y_index]
044:
045:                 # if the left side gate is masked, keep looking to the left
046:                 # until we find a valid gate or reach the maximum gap size
+047:                 if neighbor == 0:
+048:                     for i in range(max_gap_x):
+049:                         x_check -= 1
+050:                         if x_check == -1:
+051:                             if rays_wrap_around:
+052:                                 x_check = right
053:                             else:
+054:                                 break
+055:                         neighbor = labels[x_check, y_index]
+056:                         if neighbor != 0:
+057:                             break
058:
059:                 # add the edge to the collection (if valid)
+060:                 nvel = data[x_check, y_index]
+061:                 collector.add_edge(label, neighbor, vel, nvel)
062:
063:             # right
+064:             x_check = x_index + 1
+065:             if x_check == right+1 and rays_wrap_around:
+066:                 x_check = 0     # wrap around
+067:             if x_check != right+1:
+068:                 neighbor = labels[x_check, y_index]
069:
070:                 # if the right side gate is masked, keep looking to the left
```

Optimizing *Cython* code

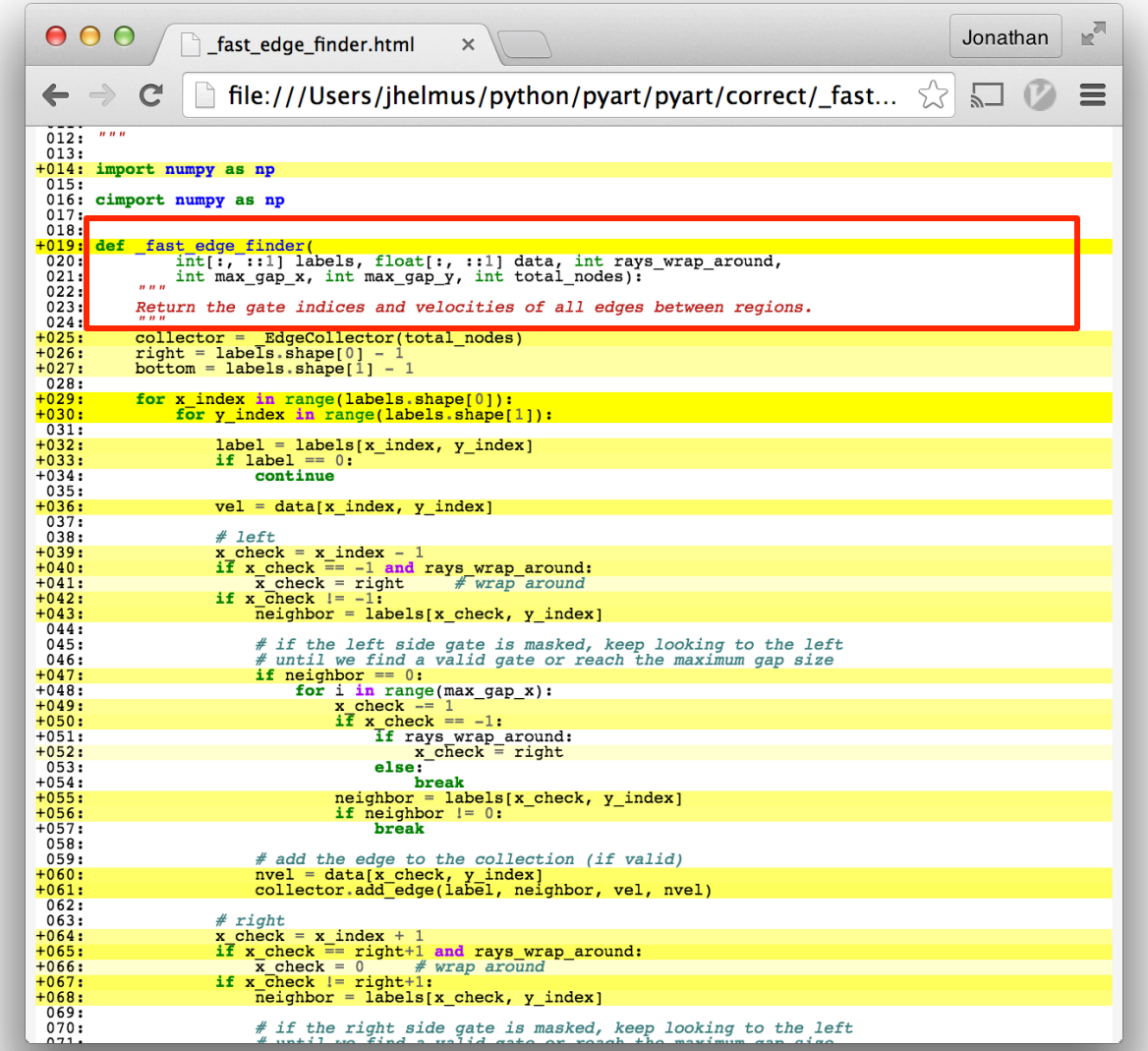


Optimizing *Cython* code



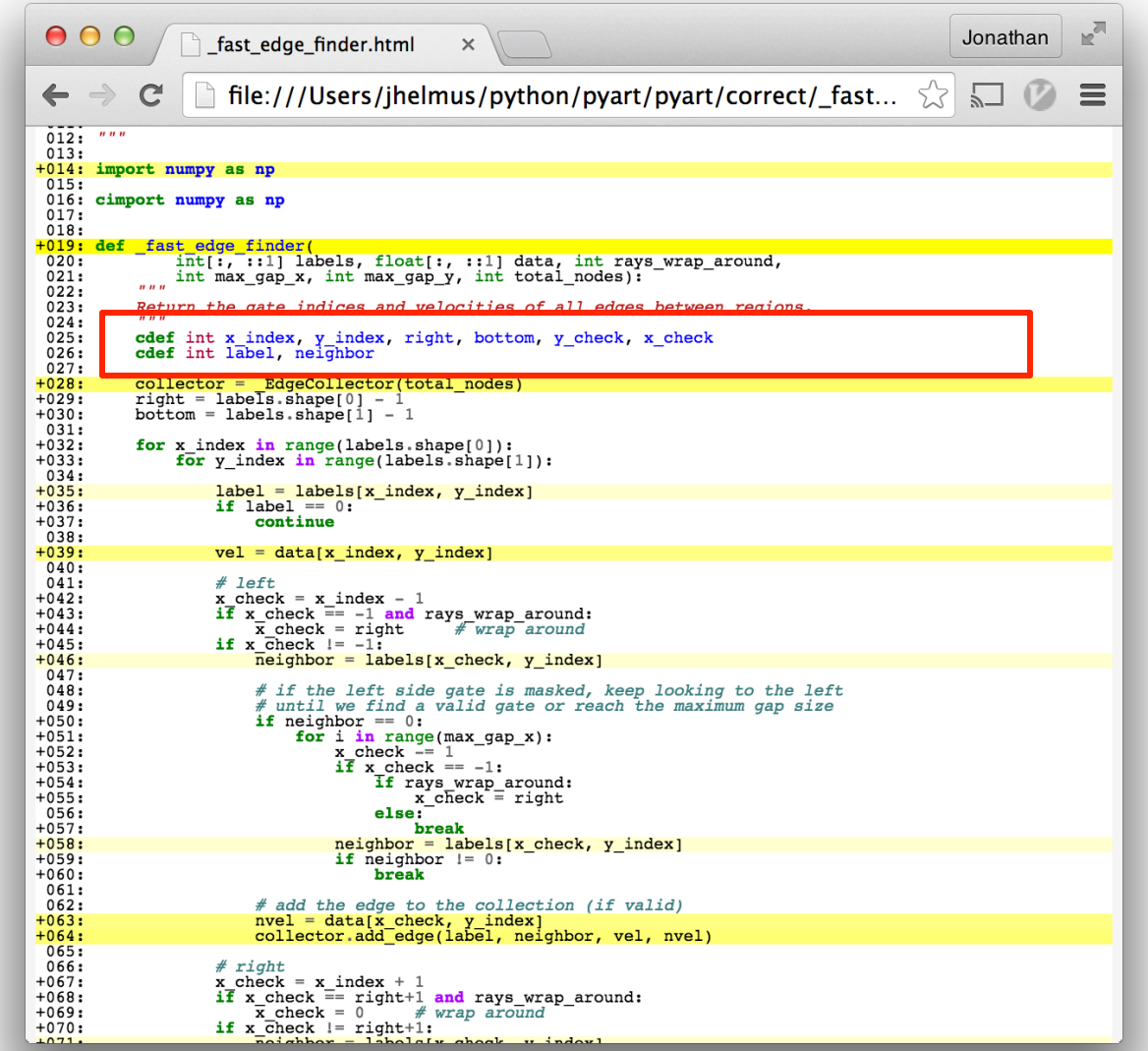

```
 012: """
 013:
+014: import numpy as np
 015:
 016: cimport numpy as np
 017:
 018:
+019: def _fast_edge_finder(labels, data, rays_wrap_around,
 020:                       max_gap_x, max_gap_y, total_nodes):
 021:     """
 022:     Return the gate indices and velocities of all edges between regions.
 023:     """
 024:
+025:     collector = _EdgeCollector(total_nodes)
+026:     right = labels.shape[0] - 1
    __pyx_t_2 = __Pyx_PyObject_GetAttrStr(__pyx_v_labels, __pyx_n_s_shape); if (unlikely(!__pyx_t_2)) {__pyx_
    __Pyx_GOTREF(__pyx_t_2);
    __pyx_t_1 = __Pyx_GetItemInt(__pyx_t_2, 0, long, 1, __Pyx_PyInt_From_long, 0, 0, 1); if (unlikely(__pyx_t
    __Pyx_GOTREF(__pyx_t_1);
    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
    __pyx_t_2 = PyNumber_Subtract(__pyx_t_1, __pyx_int_1); if (unlikely(!__pyx_t_2)) {__pyx_filename = __pyx_
    __Pyx_GOTREF(__pyx_t_2);
    __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
    __pyx_v_right = __pyx_t_2;
    __pyx_t_2 = 0;
+027:     bottom = labels.shape[1] - 1
 028:
+029:     for x_index in range(labels.shape[0]):
+030:         for y_index in range(labels.shape[1]):
 031:
+032:             label = labels[x_index, y_index]
+033:             if label == 0:
      __pyx_t_8 = PyObject_RichCompare(__pyx_v_label, __pyx_int_0, Py_EQ); __Pyx_XGOTREF(__pyx_t_8); if (ur
      __pyx_t_9 = __Pyx_PyObject_IsTrue(__pyx_t_8); if (unlikely(__pyx_t_9 < 0)) {__pyx_filename = __pyx_f
      __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
      if (__pyx_t_9) {
+034:                 continue
 035:
+036:             vel = data[x_index, y_index]
 037:
 038:             # left
+039:             x_check = x_index - 1
+040:             if x_check == -1 and rays_wrap_around:
+041:                 x_check = right     # wrap around
+042:             if x_check != -1:
+043:                 neighbor = labels[x_check, y_index]
 044:
 045:                 # if the left side gate is masked, keep looking to the left
 046:                 # until we find a valid gate or reach the maximum gap size
+047:                 if neighbor == 0:
+048:                     for i in range(max_gap_x):
+049:                         x_check -= 1
+050:                         if x_check == -1:
+051:                             if rays_wrap_around:
+052:                                 x_check = right
 053:                             else:
+054:                                 break
+055:                         neighbor = labels[x_check, y_index]
+056:                         if neighbor != 0:
```

Optimizing *Cython* code



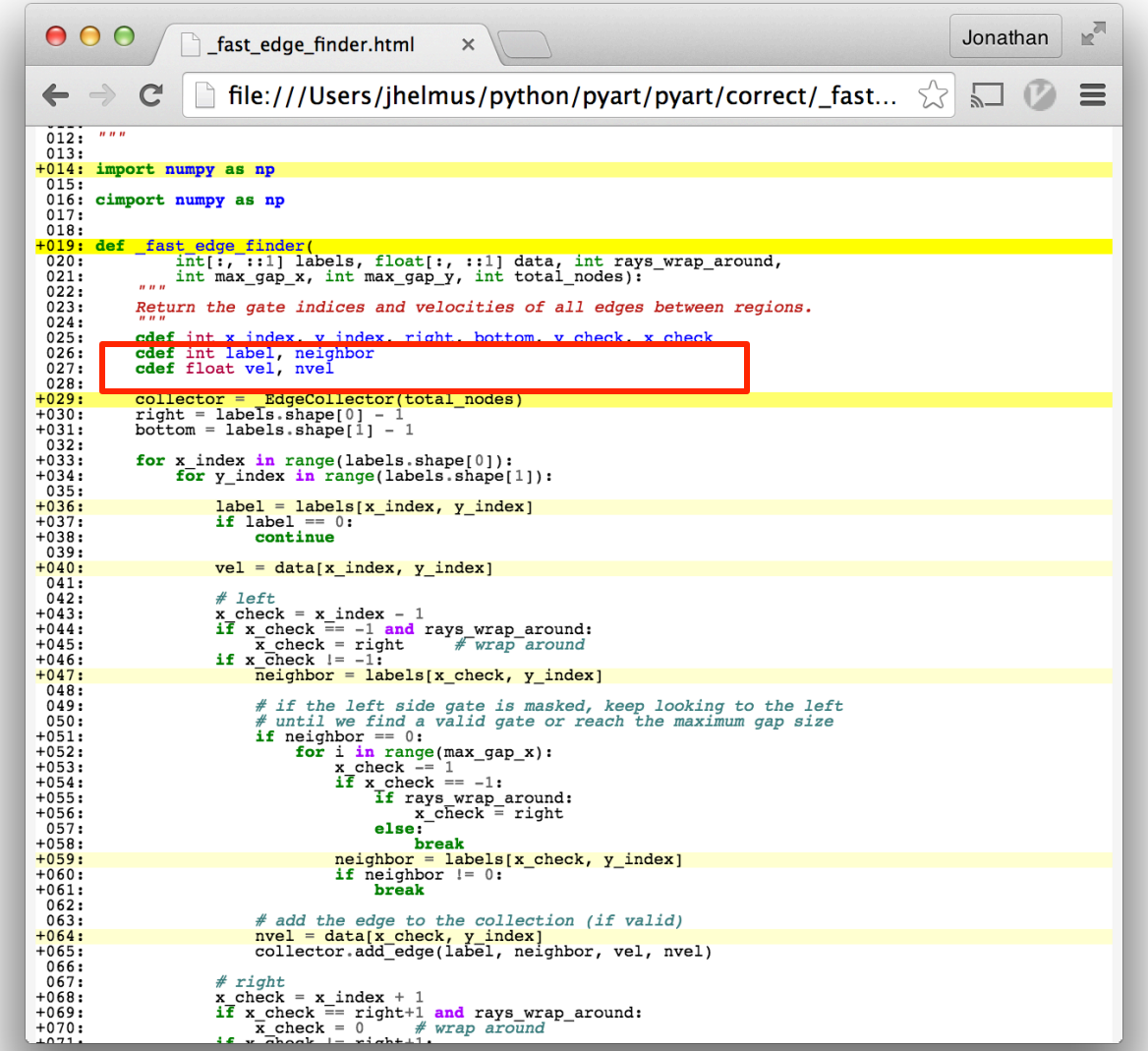

```
012: """
013:
+014: import numpy as np
015:
016: cimport numpy as np
017:
018:
+019: def _fast_edge_finder(
020:         int[:, ::1] labels, float[:, ::1] data, int rays_wrap_around,
021:         int max_gap_x, int max_gap_y, int total_nodes):
022:     """
023:     Return the gate indices and velocities of all edges between regions.
024:     """
+025:     collector = _EdgeCollector(total_nodes)
+026:     right = labels.shape[0] - 1
+027:     bottom = labels.shape[1] - 1
028:
+029:     for x_index in range(labels.shape[0]):
+030:         for y_index in range(labels.shape[1]):
031:
+032:             label = labels[x_index, y_index]
+033:             if label == 0:
+034:                 continue
035:
+036:             vel = data[x_index, y_index]
037:
038:             # left
+039:             x_check = x_index - 1
+040:             if x_check == -1 and rays_wrap_around:
+041:                 x_check = right     # wrap around
+042:             if x_check != -1:
+043:                 neighbor = labels[x_check, y_index]
044:
045:                 # if the left side gate is masked, keep looking to the left
046:                 # until we find a valid gate or reach the maximum gap size
+047:                 if neighbor == 0:
+048:                     for i in range(max_gap_x):
+049:                         x_check -= 1
+050:                         if x_check == -1:
+051:                             if rays_wrap_around:
+052:                                 x_check = right
053:                             else:
+054:                                 break
+055:                         neighbor = labels[x_check, y_index]
+056:                         if neighbor != 0:
+057:                             break
058:
059:                 # add the edge to the collection (if valid)
+060:                 nvel = data[x_check, y_index]
+061:                 collector.add_edge(label, neighbor, vel, nvel)
062:
063:             # right
+064:             x_check = x_index + 1
+065:             if x_check == right+1 and rays_wrap_around:
+066:                 x_check = 0     # wrap around
+067:             if x_check != right+1:
+068:                 neighbor = labels[x_check, y_index]
069:
070:                 # if the right side gate is masked, keep looking to the left
```

Optimizing *Cython* code

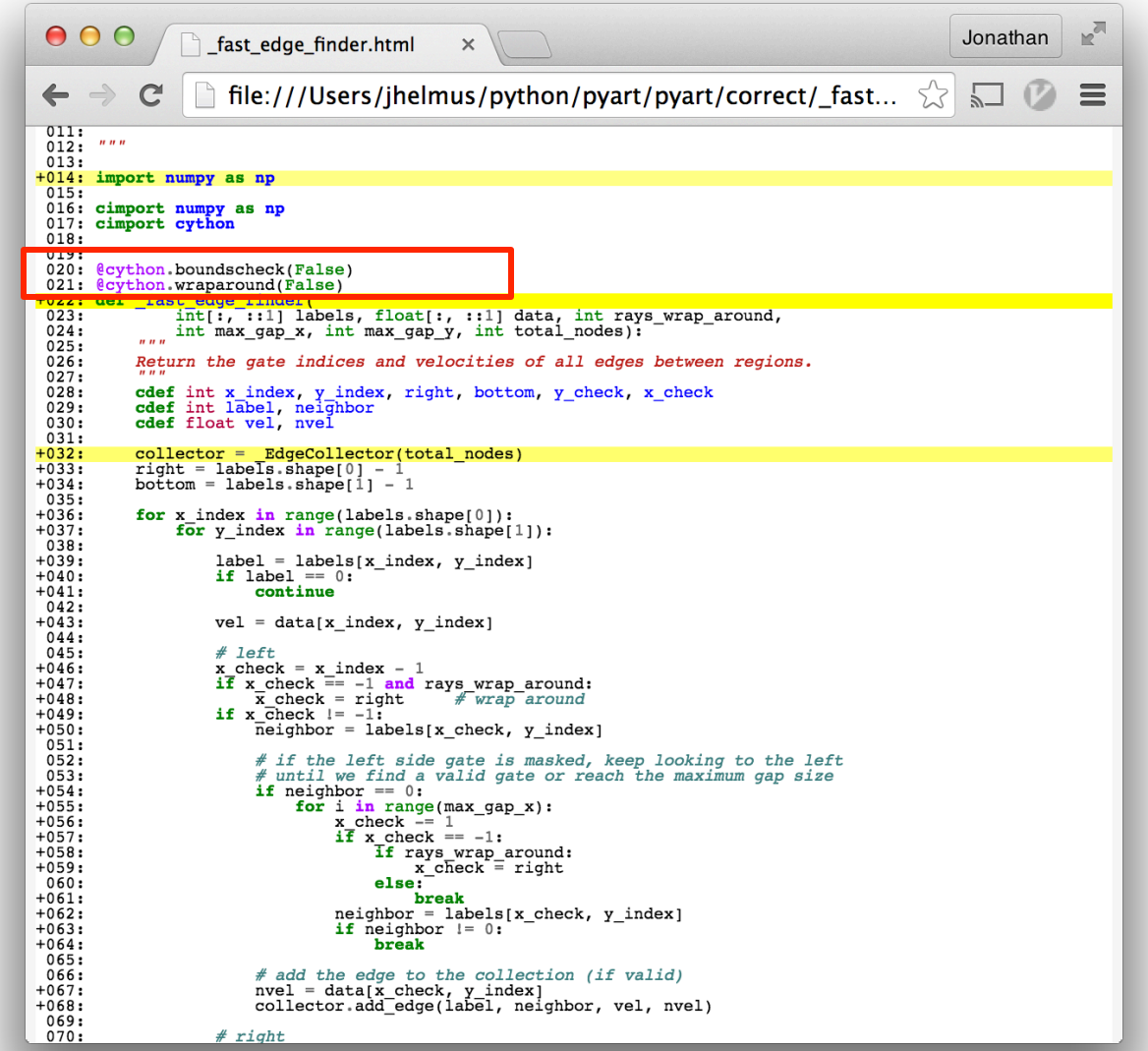



```
012: """
013:
+014: import numpy as np
015:
016: cimport numpy as np
017:
018:
+019: def _fast_edge_finder(
020:         int[:, ::1] labels, float[:, ::1] data, int rays_wrap_around,
021:         int max_gap_x, int max_gap_y, int total_nodes):
022:     """
023:     Return the gate indices and velocities of all edges between regions.
024:     """
025:     cdef int x_index, y_index, right, bottom, y_check, x_check
026:     cdef int label, neighbor
027:
+028:     collector = _EdgeCollector(total_nodes)
+029:     right = labels.shape[0] - 1
+030:     bottom = labels.shape[1] - 1
031:
+032:     for x_index in range(labels.shape[0]):
+033:         for y_index in range(labels.shape[1]):
034:
+035:             label = labels[x_index, y_index]
+036:             if label == 0:
+037:                 continue
038:
+039:             vel = data[x_index, y_index]
040:
041:             # left
+042:             x_check = x_index - 1
+043:             if x_check == -1 and rays_wrap_around:
+044:                 x_check = right     # wrap around
+045:             if x_check != -1:
+046:                 neighbor = labels[x_check, y_index]
047:
048:                 # if the left side gate is masked, keep looking to the left
049:                 # until we find a valid gate or reach the maximum gap size
+050:                 if neighbor == 0:
+051:                     for i in range(max_gap_x):
+052:                         x_check -= 1
+053:                         if x_check == -1:
+054:                             if rays_wrap_around:
+055:                                 x_check = right
056:                             else:
+057:                                 break
+058:                         neighbor = labels[x_check, y_index]
+059:                         if neighbor != 0:
+060:                             break
061:
062:                 # add the edge to the collection (if valid)
+063:                 nvel = data[x_check, y_index]
+064:                 collector.add_edge(label, neighbor, vel, nvel)
065:
066:             # right
+067:             x_check = x_index + 1
+068:             if x_check == right+1 and rays_wrap_around:
+069:                 x_check = 0     # wrap around
+070:             if x_check != right+1:
```

Optimizing *Cython* code



Optimizing *Cython* code




```
011:
012: """
013:
+014: import numpy as np
015:
016: cimport numpy as np
017: cimport cython
018:
019:
020: @cython.boundscheck(False)
021: @cython.wraparound(False)
+022: def _fast_edge_finder(
023:         int[:, ::1] labels, float[:, ::1] data, int rays_wrap_around,
024:         int max_gap_x, int max_gap_y, int total_nodes):
025:     """
026:     Return the gate indices and velocities of all edges between regions.
027:     """
028:     cdef int x_index, y_index, right, bottom, y_check, x_check
029:     cdef int label, neighbor
030:     cdef float vel, nvel
031:
+032:     collector = _EdgeCollector(total_nodes)
+033:     right = labels.shape[0] - 1
+034:     bottom = labels.shape[1] - 1
035:
+036:     for x_index in range(labels.shape[0]):
+037:         for y_index in range(labels.shape[1]):
038:
+039:             label = labels[x_index, y_index]
+040:             if label == 0:
+041:                 continue
042:
+043:             vel = data[x_index, y_index]
044:
045:             # left
+046:             x_check = x_index - 1
+047:             if x_check == -1 and rays_wrap_around:
+048:                 x_check = right      # wrap around
+049:             if x_check != -1:
+050:                 neighbor = labels[x_check, y_index]
051:
052:                 # if the left side gate is masked, keep looking to the left
053:                 # until we find a valid gate or reach the maximum gap size
+054:                 if neighbor == 0:
+055:                     for i in range(max_gap_x):
+056:                         x_check -= 1
+057:                         if x_check == -1:
+058:                             if rays_wrap_around:
+059:                                 x_check = right
060:                             else:
+061:                                 break
+062:                         neighbor = labels[x_check, y_index]
+063:                         if neighbor != 0:
+064:                             break
065:
066:                 # add the edge to the collection (if valid)
+067:                 nvel = data[x_check, y_index]
+068:                 collector.add_edge(label, neighbor, vel, nvel)
069:
070:             # right
```

Run time optimization using *Cython*

```
$ kernprof.py -l –v script.py

Total time: 155.869 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   ...
   164                                           # find regions in original data
   165        17       347781  20457.7      0.2  labels, nfeatures = _find_regions(sdata, sfilter, interval_limits)
   166        17           46      2.7      0.0  edge_sum, edge_count, region_sizes = _edge_sum_and_count(
   167        17           20      1.2      0.0      labels, nfeatures, sdata, rays_wrap_around, skip_between_rays,
   168        17    107326070 6313298.2     68.9       skip_along_ray)


Total time: 49.5179 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   ...
   160                                           # find regions in original data
   161        17       348933  20525.5      0.7  labels, nfeatures = _find_regions(sdata, sfilter, interval_limits)
   162        17           46      2.7      0.0  edge_sum, edge_count, region_sizes = _edge_sum_and_count(
   163        17           22      1.3      0.0      labels, nfeatures, sdata, rays_wrap_around, skip_between_rays,
   164        17       234857  13815.1      0.5      skip_along_ray)
```

107 seconds vs. 0.234 seconds, x450 performance improvement

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 and can be used as a guide to reduce memory usage.

- The ***cProfile*** and ***line_profiler*** modules provide statistics on execution time, either by function or line-by-line. This information can be used to optimize runtime.

- ***Cython*** is a Python to C translator that can be used to speed up the execution time of Python code by adding static type information.

Modification	Run time
Original	94 sec
Cython edge finding	35.7 sec
Edge tracker refactor	14.0 sec
Vectorized unwrapping	8.8 sec