**Python Tracking Read Me**

**Purpose:** Version of tracking algorithm developed by Sally McFarlane and Zhe Feng that has been modified for Python by Hannah C Barnes.

* Two options for tracking
  + *MCS* – Starts by identifying clouds using satellite data, then uses radar data to identify “robust MCSs”
  + *Shallow Clouds* – Identifies clouds using liquid water path from an LES model

**Algorithm Details:**

Code written using Python 2.6; but has been updated to work in Python 3.

Parallelized version requires pool method in Python’s multiprocessing module

* <https://docs.python.org/2/library/multiprocessing.html>
* Multiprocessing was chosen since allows the code to automatically transition from using only one process to running multiprocesses at once.
* Drawbacks of multiprocessing include the fact that its setup requires non-trivial modifications to configuration files and it only uses one node.
* I think you could modify the code for ipyparallel, but would need to run the code in three sequential batches (1: idclouds and tracksingle; 2: gettracks, trackstats, indentifymcs/cell, matchpf, robustmcs; 3: mapmcs/cell.
* There is a flag that allows the code to run serially rather than in parallel.

Required Python Modules: (Module names italized, aliases and functions in regular font)

* + *Numpy* as np
  + *os*
  + *sys*
  + *fnmatch*
  + *time*
  + *warnings*
  + *datetime*
  + *calendar*
  + utc from *pytz*
  + Dataset, stringtochar, num2date from *netCDF4*
  + *xarray* as xr
  + Pool from *Multiprocessing*
  + *pandas* as pd
  + label, binary\_dilation, filters, generate\_binary\_structure from *scipy.ndimage*
  + RectBivariateSpline from *scipy.interpolate*
  + pi from *math*
  + regionprops from *skimage.measure*
  + *gc*
  + skew from *scipy.stats*

Data is loaded using Dataset from netCDF4 and written using xarray. I tried using xarray to read and write all data. However, that was a very bad idea. It very significantly slowed the code.

**Algorithm Outline:**

run\_mcsdata\_allradarmatch.py and run\_LESdata.py are the master scripts that run the entire code. All variables and options are set in the first portion of these codes. No modifications should be made to any of the other codes.

1. idclouds.py
   * Identifies and classifies clouds in the raw data based on preset thresholds referred to as the cold core, cold anvil, and warm anvil.
   * Generates a netcdf file for each raw data file that contain maps of the cloud classifications and their unique number. Vectors with basic statistics about each cloud is also generated.
     + Files located in tracking directory
   * subroutine\_idclouds.py
     + Contains two methods for identifying and classifying clouds
       - Futyan3 – Identifies cold core and cold anvil, then expands to generate warm anvil
       - Futyan4 – identifies cold core, then expands to generate cold and warm anvils.
     + Both methods expand by one grid box in a diamond shape.
     + Same general idea used for both data sets. However, have two separate codes so output slightly more tailored to each case.
   * Parallelized
2. tracksingle.py
   * Compares consecutive files and identifies clouds that overlap forward and/or backward in time.
   * Generates one file for each pair of cloud files that tell which clouds in the reference and new files are linked. (reference = time0, new = time0 + dt)
     + Files located in tracking directory
   * Parallelized
     + Run on either type of data.
3. gettracks.py
   * + Takes the data about the overlapping clouds from the previous step and tracks clouds throughout the duration of the data
     + At the end, very short tracks are flagged and filled with fill values. This is the slow part of this code.
     + Generates one file that is organized by file and cloud number. For each cloud in a file it gives data that describes what track that cloud is associated with as well as its status. It also describes which clouds merge and split from larger tracks.
       - Located in stats directory
       - Run on both data sets
       - Not parallelized
       - One season takes ~2 hours
       - Status descriptions below.
4. trackstats.py
   * + Takes each track from the previous step and calculates statistics about that track. Removes short tracks and renumbers to tracks, split, and merge numbers appropriately.
     + Generates one file that has statistical data about each track. These statistics include the cloud number associated with each cloud in that track, the cloud number of any splits or mergers, the location of each cloud, geometric characteristics of each cloud, and the time of each cloud.
       - File located in stats directory
     + Same general idea used for both data sets. However, have two separate codes so output slightly more tailored to each case.
   * Not parallelized
   * Slow step, one season takes ~1 hour
5. identifymcs.py / identifycell.py
   * + This applies preset thresholds to identify which tracks are MCSs or cell tracks.
     + Generates one file that contains statistics about each track at each time. These statistics include the cloud number associated with each cloud in that track, the cloud number of any splits or mergers, the location of each cloud, geometric characteristics of each cloud, and the time of each cloud.
       - File located in stats directory
     + Same general idea used for both data sets. However, have two separate codes so output slightly more tailored to each case.
     + Not parallelized.
     + One version of codes read in using netCDF4 and one uses xarray. netCDF4 version is much faster. One season for the netCDF4 version takes a few minutes.
6. matchpf.py
   * + Takes the satellite defined MCS from the previous step and matches those storms with radar and rain accumulation data.
     + Identifies cores and precipitation features, which are radar defined regions of precipitation within the MCS cloud shields. Cores must be classified as convective. Precipitation features must be classified as convective or stratiform and exceed a preset rain rate. Multiple cores and precipitation features can exist on one satellite defined MCS.
     + Generates one file that contains statistics about each track. Contains many of the statistics from the previous file and adds statistics about the cores and precipitation features including their echo top heights, rain rates, geometric characteristics. The statistics about individual cores or precipitation features are sorted by descending area.
       - File located in stats directory
     + Only applies to MCS classification.
     + Not parallelized
     + Takes approximately 1 hour
7. robustmcs.py
   * + Using the radar statistics from the previous file it refines the list of MCSs so it only contains MCSs that satisfy preset radar thresholds.
     + Generates one file that contains statistics about each track. Those statistics are similar to those in the previous file but refined so only radar defined MCSs are contained.
       - File located in stats directory
     + Only applies to MCS classification.
     + Not parallelized
     + Very fast, just a few minutes
8. mapmcs.py / mapcell.py
   * Using the statistics files generated in previous file this code plots maps of the clouds associated with the MCS or cell tracks.
   * Generates a file for each time step of the data. This file includes maps of the raw data and the cell track numbers.
     + Files located in msctracking or celltracking directories
     + Same general idea used for both data sets. However, have two separate codes so output slightly more tailored to each case.
     + Parallelized.
     + Fast enough to run in debug.
     + Has the ability to create maps of all tracks, regardless of whether it is associated with a MCS or cell. This adds a lot of time to the code. Just need to uncomment appropriate sections.

**Cloud Status Method and Classifications:**

The code loops through each file. While the code is looping it fills two sets matrices. One set describes how the cloud changes in time. The other set describes what track a given cloud merges into or splits from.

* referencetrackstatus – describes what is the status of the cloud at the reference /current time (options include merging, simple continuation, stopping, or being a cloud that splits at the next time step)
* newtrackstatus – describes what is the status of the associated clouds at the new/next time step (this describes which new clouds form from a split)
* trackmergenumber – describes which track that cloud merges into. This is stored at the current/reference time.
* tracksplitnumber – describes which track this cloud split from. This is stored at the new/next time step

The referencetrackstatus and newtrackstatus matrices use the following classification methodology:

* The reference matrix describes when the cloud stops, starts, continues, or merges. There is also a flag that indicates that this cloud splits into two clouds at the next time step.
  + 0=stop, 1=continue, 2=big cloud of merger, 21=small cloud of merger, 15=big cloud in merger that splits at the next time step, 34=small cloud in a merger that splits at the next time step, 13=This cloud splits at the next time step
* The new matrix describes when the cloud splits.
  + 3=bigger cloud in a split, 31=smaller cloud in a split

When the code is done looping the referencetrackstatus and newtrackstatus matrices are added together and the resulting values are listed below.

\* The cloud status numbers purely given to inform the user, the code has been modified so these values are not actually used.

|  |  |
| --- | --- |
| *Status Number* | *Description* |
| 0 | Track stops |
| 1 | Simple track continuation |
| 2 | This is the bigger cloud in simple merger |
| 3 | This is the bigger cloud from a simple split that stops at this time |
| 4 | This is the bigger cloud from a split and this cloud continues to the next time |
| 5 | This is the bigger cloud from a split that subsequently is the big cloud in a merger |
| 13 | This cloud splits at the next time step |
| 15 | This cloud is the bigger cloud in a merge that then splits at the next time step |
| 16 | This is the bigger cloud in a split that then splits at the next time step |
| 18 | Merge-split at same time (big merge, splitter, and big split) |
| 21 | This is the smaller cloud in a simple merger. |
| 24 | This is the bigger cloud of a split that is then the small cloud in a merger |
| 31 | This is the smaller cloud in a simple split that stops |
| 32 | This is a small split that continues onto the next time step |
| 33 | This is a small split that then is the bigger cloud in a merger |
| 34 | This is the small cloud in a merger that then splits at the next time step |
| 37 | Merge-split at same time (small merge, splitter, big split) |
| 44 | This is the smaller cloud in a split that is smaller cloud in a merger at the next time step |
| 46 | Merge-split at same time (big merge, splitter, small split) |
| 52 | This is the smaller cloud in a split that is smaller cloud in a merger at the next time step |
| 65 | Merge-split at same time (smaller merge, splitter, small split) |