Community Convective cloud Model Evaluation Toolkit (CoCoMET) User Guide

Travis Hahn[1], Dié Wang[2], Hershel Weiner[3], Calvin Brooks[4], Jie Xi Li[5], and Siddhant Gupta[6]

[1]Department of Statistics, The Pennsylvania State University
[2]Environmental and Climate Sciences Department, Brookhaven National Laboratory
[3]Physics and Astronomy Department, University of Hawaii
[4]Physics, Applied Physics, and Astronomy Department, Rensselaer Polytechnic Institute
[5]Applied Mathematics & Statistics, Stony Brook University
[6]Environmental Sciences Division, Argonne National Laboratory

January 2025

A toolkit of the Advanced Study of Cloud and Environment iNTerations (ASCENT) program.

Contents

1 Introduction to CoCoMET

CoCoMET is an open-source Python package centered around the unification of model data, observation types, trackers, and variable calculation methodology. CoCoMET works by internally handling all the data transformations and tuning parameters required to prepare and run various trackers in addition to running the trackers themselves in a parallelized environment and calculating common tracked object properties. The package can interface with multiple model outputs including the Weather Research and Forecast (WRF) model, the non-hydrostatic mesoscale atmospheric model (MesoNH), and the Regional Atmospheric Modeling System (RAMS) in conjunction with observational data such as spectral radiances from the Geostationary Operational Environmental Satellite (GOES) sytem and radar products from the Next Generation Weather Radar (NEXRAD). A separate module links Eulerian observations with the Lagrangian tracking results. In this module, we use observations from the U.S. Department of Energy's Atmospheric Radiation Measurement (ARM) Climate Research Facility at various sites across the globe. Additionally, we introduce CoCoMET-US (CoCoMET-Unified Specification), a standardized specification designed to allow the easy expansion of ANTE-TRACE to future models and tracking software. Therefore, CoCoMET has the capability to quickly and efficiently examine convective activity across models and observations, a crucial task for model assessment or model intercomparison studies.

1.1 Current Capabilities

We first categorize by the input datatype followed by a section dedicated to the analysis of tracked features.

- **WRF**
  - *Tracking and object-based analysis of clouds (tobac)* tracking of variables:
    - Reflectivity
    - Brightness Temperature
    - Updraft Velocity
    - Precipitation Rate
    - Any Gridded WRF Variable (Case Sensitive)
  - *Multi-Object Analysis of Atmospheric Phenomena (MOAAP)* tracking of Mesoscale Convective Systems and Cloud Shields
  - *Tracking Algorithm for Mesoscale convective Systems (TAMS)* tracking of Mesoscale Convective Systems
- **MesoNH**
  - *tobac* tracking of variables:
    - Reflectivity
    - Brightness Temperature
    - Updraft Velocity
    - Precipitation Rate

    - Any Gridded MesoNH Variable (Case Sensitive)
  - *MOAAP* tracking of Mesoscale Convective Systems and Cloud Shields
  - *TAMS* tracking of Mesoscale Convective Systems
- **RAMS**
  - *tobac* tracking of variables:
    - Reflectivity
    - Brightness Temperature
    - Updraft Velocity
    - Precipitation Rate
    - Any Gridded RAMS Variable (Case Sensitive)
  - *MOAAP* tracking of Mesoscale Convective Systems and Cloud Shields
  - *TAMS* tracking of Mesoscale Convective Systems
- **NEXRAD**
  - Automatically grid archival radar data
  - Combine multiple radar sites into one grid
  - *tobac* tracking of variables:
    - Reflectivity
- **Standardized Radar Grids**
  - Use any radar data which can be gridded in accordance with the CoCoMET-US Section S1.1
  - *tobac* tracking of variables:
    - Reflectivity
- **GOES Cloud and Moisture Imagery (CMI)**
  - *tobac* tracking of variables:
    - Brightness Temperature
- **Analysis Functions**
  - *ARM* Functions:
    - *arm* Link any 2D or 3D ARM variables to tracks
    - *convective_indices* Calculates convective indices from INTERPSONDE data (CAPE, CIN, etc.)
  - *var_max_height* Calculates the highest point at which a variable threshold is satisfied. Analogous to the echo top height for reflectivity data.
  - *max_intensity* The maximum value of a variable over a tracked feature's segmented area.

- *area* Calculates the areas of tracked features in square kilometers.
- *volume* Calculates the volumes of tracked features in cubic kilometers.
- *velocity* Calculates the velocities of tracked features in meters per second.
- *perimeter* Calculates the 2D perimeters of tracked features.
- *surface_area* Calculates the 3D perimeters of tracked features.
- *cell_growth* Calculates the rate of growth / decay of tracked features based on volume or area.
- *irregularity* Calculates how "irregular" each tracked feature is based on convexity or sphericity.
- *merge_split* Calculates 2D or 3D mergers and splitters.

1.2 Planned Capabilities

- Post-processing functions
  - Location / Time-based Filters
  - Remove Edge Cells
- Radar filters / corrections.

2 Installing CoCoMET

CoCoMET can be installed in two ways: with *pip* or with *conda*, but both version do necessitate *conda* to function properly.

To install it with *pip*, you first need to goto the CoCoMET github page and download the most recent *.whl* file from the releases. Then you can create a *conda* environment with Python version 3.12 and install the *cf-units* package with *conda*. Installation of *cf-units* via *conda* is important as *pip* tends to fail compiling the package. Finally, you can install CoCoMET by doing *pip install CoMET-ver-*.whl* where ver refers to the version number of the version you have.

To install via *conda*, you can simply do *conda install -c conda-forge cocomet* which should resolve all dependencies automatically. ** NOT YET IMPLEMENTED **

3 Running CoCoMET

Once installed, you can run CoCoMET by first importing it via: *import CoCoMET* or *from CoCoMET import CoCoMET_start*, and then you only require the running of a single function and a CONFIG file. Within a Python file or terminal, you can just run the start function with *CoCoMET.CoCoMET_start("./path_to_config.yml")* or *CoCoMET_start("./path_to_config.yml")* based off of how you imported CoCoMET. Or, in the case where you have already defined a Python dictionary with the CONFIG information, you can run the start function with *CoCoMET_start(CONFIG=user_dict)*. This function will return a single Python object containing all of the tracking and analysis information calculated. The details of the output are laid out in the Understanding CoCoMET Output section below.

4 Setting up your CONFIG

As mentioned in the above section, CoCoMET primarily relies on a single, user-defined CONFIG file for controlling all of the functionality. There are two ways to define the CONFIG, mainly by creating an external YAML file (a text file with special formatting) or a Python dictionary. For the purposes of this document, we will be describing how to create the YAML file.

4.1 Boilerplate Structure

The CONFIG file is, in general, laid out in this way:

```
Setup Options

Observation Type:
        path_to_data
        additional_observation_parameters

        tracker:
                tracker_params

                analysis:
                        analysis_variables

Observation Type:
        path_to_data
        .
        .
        .
```

4.2 Detailed Descriptions

Here we describe, in detail, all possible CoCoMET CONFIG inputs / options. Note that all the following are case-sensitive.

```
verbose: True # [bool] Whether to use verbose output (loading bars, etc.).
parallel_processing: True # [bool] Whether or not to use parallel processing for certain
     tasks.
max_cores: 32 # [int] Number of cores to use if parallel_processing==True. Enter None
    for unlimited.

# Here we list the possible model inputs
(wrf, mesonh, rams):
        path_to_data: "wrfout_d02*" # [str] Glob-like path to input data. Required.

        id_idealized: False # [bool] True/False boolean if input data is idealized or not
             . This corrects time errors arising from idealized setting with some models
        min_frame_index: 10 # [int] 0-based indexing, inclusive. If you want to select
            only a subset of the input data. A frame is a single input file. Optional.
        max_frame_index: 70 # [int] 0-based indexing, inclusive. Optional.

        feature_tracking_var: "DBZ" # [str] DBZ, TB, WA, PR, Or other WRF Standard
            variable name (Case-Sensitive). Variable you want to track features on.
        segmentation_var: "DBZ" # [str] Variable you want to do the segmentation on. Same
             options as feature_tracking_var.
```

```
tobac:
       feature_id: # This takes any argument that tobac takes as a feature
           detection parameter. Available \href{https://tobac.readthedocs.io/en/
           latest/threshold_detection_parameters.html}{here}:
              threshold: [30,40,50,60] # [Float or Arr[Floar]]
              target: "maximum" # [str]
              .
              .
              .

       linking: # This takes any argument that tobac takes as a linking parameter
           . Available \href{https://tobac.readthedocs.io/en/latest/linking.html
           }{here}:
              method_linking: "predict" # [str]
              v_max: 20 # [float]
              .
              .
              .

       segmentation_2d: # This takes any argument that tobac takes as a
           segmentation parmater \href{https://tobac.readthedocs.io/en/latest/
           segmentation_parameters.html}{here} AND the addition of the height
           parameter
              height: 2 # [float] Approximate height, in kilometers, at which to
                   calculate the segmented area of the tracked features.
                  Required
              method: "watershed" # [str]
              threshold: 15 # [float]
              .
              .
              .

       segmentation_3d: # Takes the same argument as above but without the height
            argument
              method: "watershed" # [str]
              threshold: 15 # [float]

       analysis:
              arm: {
                     path_to_files: "/ARM/VDISQUANTS/*", # [str] A glob-like
                         path to the ARM product output. Required.
                     variable_name: "precipitation_rate", # [str or Arr[str]]
                         Case sensitive name of variable you want to extract
                         from the ARM data. Can be either a string or array of
                         strings such as ["precipitation_rate", "
                         precipitation_total"]. Required.
              } # Returns an xarray Dataset with the following: frame,
                  tracking_time, arm_time, time_delta, closest_feature_id (km),
                  variable_names.

              convective_indices: {
```

```
        path_to_files: "/ARM/INTERPSONDE/*", # [str] A glob-like
            path to the INTERPSONDE ARM product output. Required.
        parcel: 3, # [int] Whether to use 1 - Surface, 2 - Mixed-
            Layer, or 3 - Most Unstable parcel. The default is 3.
        ml_depth: 482.0, # [float] Depth of mixed-layer in meters (
            when parcel = 2). The default is 482.0.
        mu_depth: 700.0, # [float] Look below this pressure level,
            in hPa, for most unstable parcel (when parcel = 3). The
             default is 700.0.
        start: 0, # [int] Index to start at for determining parcel
            properties (e.g. 1 = z(1)). The default is 0.
        flag_heat: 2, # [int] Latent heating only due to liquid or
            liquid and ice? 1 - Liquid only / 2 - Liquid and Ice.
            The default is 2.
        flag_convert: 0, # [int] Convert pre-existing cloud water
            to cloud ice? 0 = No / 1 - Yes (using linear function
            of temperature) Requires flag_adiabat = 2 below (so we
            have hydrometeors to freeze). The default is 0.
        flag_adiabat: 1 # [int] Reversible or Irreversible parcel
            path? 1 - Pseudoadiabatic (Irreversible) / 2 - Moist
            Adiabatic (Reversible). The default is 1.
} # Returns an xarray Dataset with the following: sonde_time,
    cell_id, cape, ncape, cin, lnb, lfc, lcl, wind_shear, low_rh,
    mid_rh, richardson, elr_0_3.

var_max_height: {
        variable: "DBZ", # [str] The variable from the input
            segmentation_xarray which should be used for
            calculating var_max_height. Required.
        threshold: 15, # [float] The value which needs to be
            exceeded to count towards the var top height. I.e. 15
            for reflectivity. Required.
        cell_footprint_height: 2, # [float] The height, in
            kilometers, used to calculate the cell area to
            determine where to calculate var_max_heights. The
            default is 2km.
        quantile: 0.95 # [float<=1] The percentile of calculated
            max heights to return. The default is 0.95.
} # Returns a pandas dataframe with the following rows: frame,
    feature_id, cell_id, max_height in km.

max_intensity: {
        variable: "DBZ", # [str] The variable from the input
            segmentation_xarray which should be used for
            calculating var_max_height. Required.
        cell_footprint_height: 2, # [float] The height, in
            kilometers, used to calculate the cell area to
            determine where to calculate the max intensity. The
            default is 2km.
        quantile: 0.95. # [float<=1] The percentile of calculated
            max intensities to return. The default is 0.95. NOT
            IMPLEMENTED.
```

```
} # Returns a pandas dataframe with the following rows: frame,
    feature_id, cell_id, max_intensity

area: {
       height: 2, # [float] The height, in kilometers, which is
           used to calculate the area of cells. The default is 2km
           .
       variable: None, # [str] Variable to which we should apply
           the threshold if desired. The default is None.
       threshold: None # [float] Value of selected variable which
           the area should be greater than. E.g. only want to look
            at grid cells where the rain mixing ratio is greater
           than some number. The default is None.
} # Returns a pandas dataframe with the following rows: frame,
    feature_id, cell_id, area where area is in km^2.

volume: {
       variable: None # [str] Variable to which we should apply
           the threshold if desired. The default is None.
       threshold: None # [float] Value of selected background
           variable which the volume should be greater than. E.g.
           only want to look at grid cells where the rain mixing
           ratio is greater than some number. The default is None.
} # Returns a pandas dataframe with the following rows: frame,
    feature_id, cell_id, volume where area is in km^3.

velocity: {} # Returns a pandas dataframe with the following rows:
     frame, feature_id, cell_id, velocity where velocity is in m/s
     or m/frame.

perimeter: {
       variable: None # [str] Variable to which we should apply
           the threshold if desired. The default is None.
       threshold: None # [float] Value of selected background
           variable which the perimeter grid cells should be
           greater than. E.g. only want to look at grid cells
           where the rain mixing ratio is greater than some number
           . The default is None.
} # Returns a pandas dataframe with the following rows: frame,
    feature_id, cell_id, perimeter where perimeter is in km^(
    segmentation_dim - 1).

cell_growth: {} # Returns a pandas dataframe with the following
    rows: frame, feature_id, cell_id, cell_growth where
    cell_growth is in m^3 / s.

irregularity: {
       irregularity_metrics: ["sphericity", "convexity"], # [str
           or Arr[str]] A string or list of strings for which
           irregularity metric to calculate. Options are "
           convexity" or "sphericity". Required.
       segmentation_type: "2d" # [str] Whether to calculate 2d or
           3d convexity if convexity in irregularity_matrics. The
```

```
                default is "3d".
        } # Returns a pandas dataframe with the following rows: frame,
            feature_id, cell_id, irregularity where irregularity values
            are floats from 0 to 1.

        merge_split: { # Default setup is for 2D merge / split tracking on
             reflectivity
               segmentation_type: "2d", # [str] Type of merging and
                   splitting to analyze, either 2d or 3d. The default is
                   "2d".
               varible: "DBZ", # [str] The variable which defines the
                   background field--i.e. what we want to track mergers
                   and splitters for. Required.
               invert: False, # [bool] If we care about tracking lower
                   values such as brightness temperature (i.e. we want to
                   flood fill stuff less than a threshold), we need to
                   invert the data so we can use the same algorithm. Will
                   also invert background value. The default is False.
               cell_footprint_height: 2, # [float] The height at which we
                   want to find the newly calculated cell areas in
                   kilometers. The default is 2km.
               touching_threshold: 0.20, # [0<float<=1] The percentage of
                   cell border which must be shared by two cells. The
                   default is 0.20.
               flood_background: 20, # [float] The minimum background
                   value for area calculation. The default is 20.
               score_threshold: 1, # [float] The minimum value of the
                   score function to be considered a part of the cell. The
                    default is 0.
               score_weight_1: 1, # [float] The weighting of the relative
                   strength of the cell. The default is 1.
               score_weight_2: 1, # [float] The weighting of the relative
                   distance of the cell. The default is 1.
               radius_multiplyer: 0.1, # [float] The multiplyer to
                   increase the search radius by. Is equal to 1 +
                   radius_multiplyer. So a value of 0.1 increase the
                   search radius by 10%. The default is 0.1.
               overlap_threshold: 0.5, # [0<float<=1] The amount two cells
                    need to overlap [0 to 1]. The default is 0.5.
               steps_forward_back: 2 # [int] The number of steps to look
                   forward or back to detect mergers/splitters. The
                   default is 2.
        }

moaap: # This takes any argument that MOAAP accepts \href{https://github.com/
    AndreasPrein/MOAAP/wiki}{here}.
       tracking_save_path: "/data/moaap_output/" # [str] Location to save MOAAP
           tracking output to
       MinTimePR: 0.5 # [float]
       MinAreaPR: 10 # [float]
       .
       .
       .
```

```
        analysis_type: "MCS" # [str] Either "MCS" or "cloud" if we want to analyze
             tracked MCSs or cloud shields.

        analysis:
               # Same options as tobac

 tams:
        ctt_threshold: 235 # [float]
        ctt_core_threshold: 219 # [float]
        u_projection: 0 [int]
        parallel: False # [bool] Whether to use TAMS parallelization, tends to not
             work when using CoCoMET parallelization as well

        analysis_type: "cloud" # [str]

        analysis:
               # Same options as tobac

# Here we list possible NEXRAD, i.e. gridded, radar options
(nexrad, multi_nexrad):
```

4.3 Example CONFIG

5 Understanding CoCoMET Output