RCAT

Release 1.0

Nov 02, 2022

Contents

RCAT is an analysis tool primarily developed for analysis of regional climate models, but may also be used for global models as well. It's purely written in Python where we aim for a modular code in a functional style. The purpose is to get an efficient and structured way of collaboration within our model developers but also for non pythonists that want to use the tool for standard climate data analysis.

The tool is adapted to new demands in regional climate modeling, where the data amounts are large and analysis often is carried out on HPC systems.

Tutorials **- Start here**

Instructions for new RCAT users on how to install the software and making your first plot.

How-to guides

Hands on guides with code examples.

Development

Reference

Reference material (APIs)

CHAPTER 1

Tutorials

Here you can find information on how to get started. How to install RCAT and set up configuration for your High Performance Computing (HPC).

The more experienced user can follow short instructions in the *How-To Guides* for various topics.

- *Installation*
- *RCAT Configuration*
- *RCAT Statistics*

1.1 Installation

RCAT is a python3 based tool and the easiest way to get started is by using the Conda framework. If you don't have conda installed follow the installation guide from start. Otherwise you can follow from *RCAT environment* .

N.B. conda-forge channel need to be added in $HOME/.condarc

1.1.1 Miniconda

Install Miniconda

Add conda-forge channel

- create $HOME/.condarc file

```
channels:
    - conda-forge
    - defaults
```

Update conda

- conda update conda

1.1.2 RCAT environment

Create environment

- conda create -n rcat

Activate environment

- conda activate rcat

Install dependencies

It's important that you follow the order of the installation list below due to a bug in the esmpy module.

- conda install esmpy
- conda install xesmf dask
- conda install netcdf4 dask-jobqueue matplotlib basemap

1.2 RCAT Configuration

The main set up is done in the **<path-to-RCAT>/config/config_main.ini** configuration file. In this file you will set up paths to model data, which variables to analyze and how (define statistics), which observations to compare with etc. In other words, this is your starting point when applying RCAT.

1. **Setup folder structure** If you don't want to pollute your cloned git repository we suggest you to create a new folder for your analysis and copy the main RCAT configuration file to the new folder.

```
mkdir -p $HOME/rcat_analysis/test
cd $HOME/rcat_analysis/test
cp <path-to-RCAT>/config/config_main.ini .
```

2. **Configure settings in config_main.ini** A configuration .ini file has a specific structure based on sections, properties and values. The RCAT config_main.ini file consists of a handful of these sections, for example **MODELS**, under which you specify certain properties or values. The latter may in some cases be common structures used in python like lists or dictionaries. Below follows a description of each of the sections needed to setup the analysis.

   - **MODELS** Here you specify the path to model data. At the moment a specific folder structure is anticipated, with sub-folders under fpath according to output frequency; fpath/day, fpath/6H, fpath/15Min, etc. Names of these sub-folders are inherited from the freq property set under variables in the **SETTINGS** section.

```
model = {
   'fpath': '/path/to/model/data',
   'grid type': 'reg', 'grid name': 'FPS-ALPS3',
   'start year': 1998, 'end year': 2000, 'months': [1,2,3,4,5,6,7,8,9,
↪10,11,12]
}
```

     Here you also set a couple of grid specifications - namely *grid type* and *grid name*. Grid type defines the type of grid the model data is currently on; it can be set to either *rot* or *reg*. The former means that model data is on a rotated grid and the latter that it is on a non-rotated grid (i.e. regular, rectilinear, curvilinear). If data is on rotated grid RCAT will de-rotate the grid. However, it requires that model files include coordinate variables in accordance with CF conventions - *rlon/rlat* for longitudes and latitudes as well as the variable *rotated_pole* with attributes *grid_north_pole_longitude* and *grid_north_pole_latitude*. Grid name is a user defined label for

the grid. If data is to be remapped to this model grid, the output filenames from RCAT analysis will include this specified grid name.

Here's another example comparing two models:

```
model_his = {
   'fpath': '/path/to/model_1/data',
   'grid type': 'reg', 'grid name': 'FPS-ALPS3',
   'start year': 1985, 'end year': 2005, 'months': [1,2,3,4,5,6,7,8,9,
↪10,11,12]
}
model_scn = {
   'fpath': '/path/to/model_2/data',
   'grid type': 'reg', 'grid name': 'FPS-ALPS3',
   'start year': 2080, 'end year': 2100, 'months': [1,2,3,4,5,6,7,8,9,
↪10,11,12]
}
```

Two different periods is set here because a simulation of historic period will be compared with a simulation of future climate. More models can be added to the section, but note that the first model (e.g. model_his in the above example) will be the reference model. That is, if validation plot is True, and no obs data is specified, the difference plots will use the first specified model in section as reference data.

Note: If you want to see how RCAT uses defined file paths and other information to retrieve lists of model data files, see the *get_mod_data* function in *<path-to-RCAT/rcat/runtime/RCAT_main.py*.

- **OBS** If observation data is to be used in the analysis, you will need to specify a meta data file by setting the full path to *observations_metadata_NN.py* (located under <path-to-RCAT>/config). *NN* is any label that signifies the observation meta data for a specific location or system (for example a HPC system). If such a specific meta data file does not exist, it should be created (SAMPLE_observations_metadata.py can be used as a template) and modified. **N.B.** Change only the *obs_data* function – where observations are specified.

  In addition, in this section one will specify the time period and months for obs data. The same time period will be used for all observations. Which specific observations to include in the analysis is not defined here, but in the **SETTINGS** section, in the variables property.

- **SETTINGS output dir**: The path for the output (statistics files, plots). If you re-run the analysis with the same output directory, you will prompted to say whether to overwrite existing output. "overwrite" does not mean that existing folder will be completely overwritten (deleted and created again). The existing folder structure will be kept intact together with output files. However, potentially some output (statistics/figure files) with same names will be overwritten.

  variables: One of the key settings in the configuration file. The value of this property is represented by a dictionary; the keys are strings of variable names ('pr', 'tas', ...) and the value of each key (variable) is another dictionary consisting of a number of specific settings:

```
variables = {
 'tas': {
    'freq': 'day',
    'units': 'K',
    'scale factor': None,
    'accumulated': False,
    'obs': 'ERA5',
```

(continues on next page)

(continued from previous page)

```
    'obs scale factor': None,
    'var names': {'model_1': {'prfx': 'tas', 'vname': 'var167'}},
    'regrid to': 'ERA5',
    'regrid method': 'bilinear'},
 'psl': {
    'freq': '3hr',
    'units': 'hPa',
    'scale factor': 0.01,
    'accumulated': False,
    'obs': None,
    'obs scale factor': None,
    'var names': None,
    'regrid to': None,
    'regrid method': 'bilinear'},
 'pr': {
    'freq': '1hr',
    'units': 'mm',
    'scale factor': 3600,
    'accumulated': False,
    'obs': 'EOBS20',
    'obs scale factor': 86400,
    'var names': None,
    'regrid to': {'name': 'NORCP12', 'file': '/nobackup/rossby20/sm_
↪petli/data/grids/grid_norcp_ald12.nc'},
    'regrid method': 'conservative'},
    }
```

- *freq*: A string of the time resolution of input model data. The string should match any of the sub-folders under the path to model data, e.g. 'day', '1hr', '3hr'. In effect, you may choose different time resolutions for different variables in the analysis.
- *units*: The units of the variable data (which will appear in figures created in RCAT, and thus should reflect the units after data have been manipulated through the analysis).
- *scale factor*: A numeric factor (integer/float) that model data is multiplied with, to convert to desired units (e.g. from J/m2 to W/m2) and to ensure that all data (model and observations) have the same units. If no scaling is to be done, set value to None. An arithmetic expression is not allowed; for example if data is to be divided by 10 you cannot define factor as 1/10, it must then be 0.1. It is assumed that all model data will use the same factor..
- *accumulated*: Boolean switch identifying variable data as accumulated fields or not. If the former (True), then data will be de-accumulated "on the fly" when opening files of data.
- *obs*: String or list of strings with acronyms of observations to be included in the analysis (for the variable of choice, and therefore different observations can be chosen for different variables). Available observations, and their acronyms, are specified in the <path-to-RCAT>/config/observations_metadata_NN.py file. In this file you can also add new observational data sets.
- *obs scale factor*: As scale factor above but for observations. If multiple observations are defined, some of which would need different scale factors, a list of factors can be provided. However, if the same factor should be used for all observations, it is enough to just specify a single factor.
- *var names*: Variable names specified in the top key of *variables* usually refers to common names defined in CF conventions. However, there might be cases where either the variable name specified in the file name or of the variable in the file differ from these conventions. Var names provides an option to account for this; it is specified as a dictionary with keys *prfx*

and *vname* for the file name prefix and variable name respectively. If file formats follows the conventions, and thus have same prefix and name as the top key variable name, *var names* should be set to *None*. See code snippet above for examples of both types of settings.

- *regrid to*: If data is to be remapped to a common grid, you specify either the name (model name or observation acronym) of a model defined under **MODELS** section or an observation defined under *obs* key. Or, if an external grid should be used, it can be set to a dictionary with the *name* and *file* keys. *name* has the same purpose as *grid name* in the **MODELS** section above. The value of *file* must be the full path to a netcdf file that at least contains *lon* and *lat* variables defining the target grid. If no remapping is to be done, set *regrid to* to None.
- *regrid method*: String defining the interpolation method: 'conservative' or 'bilinear'.

regions: A list of strings with region names, defining geographical areas data will be extracted from. If set, 2D statistical fields calculated by RCAT will be cropped over these regions, and in line plots produced in RCAT mean statistical values will calculated and plotted for each of the regions. If the pool data option in statistics configuration (see below) is set to True, then data over regions will be pooled together before statistical calculations. If no cropping of data is wanted, set this property to None. Read more about how to handle regions and polygons in RCAT *here*.

- **STATISTICS** Another main section of the analysis configuration. Therefore, the description of this segment is given separately, see *RCAT Statistics*
- **PLOTTING** This section is intended for the case you want to perform a general evaluation/validation of the model. This means that (for the moment) a set of standards plots (maps and line plots) can be done by RCAT for a set of standard statistical output: annual, seasonal and diurnal cycles, pdf's, percentiles and ASoP analysis. If plotting procedures for other statistics is wished for, they need to be implemented in the RCAT *plotting module*.

  validation plot: If validation plot is set to True, standard plots will be produced for the defined statistics. Otherwise, plotting can be done elsewhere using the statistical output files (netcdf format) created by RCAT.

  map configure: In this property you can change/add key value pairs that control for example map projection ('proj') and resolution ('res') as well as the dimensions of the map; 'zoom' can be set to 'crnrs' if corners of model grid is to be used, or 'geom' if you want to specify width and height (in meters) of the map. In the latter case you need to set 'zoom_geom' [width, height]. Note that these settings refers to the reference model in the analysis which is the first model data set specified in the **MODELS** section.

```
map configure = {'proj': 'stere', 'res': 'l', 'zoom': 'geom', 'zoom_
↪geom': [1700000, 2100000], 'lon_0': 16.5, 'lat_0': 63}
```

  For more settings, see the map_setup function in the *plots module*.

  map grid setup: Settings for the map plot configuration, for example whether to use a colorbar or not (cbar_mode) and where to put it and the padding between panels. For more info, see the *image_grid_setup* function in the *plots module*.

```
map grid setup = {'axes_pad': 0.5, 'cbar_mode': 'each', 'cbar_location
↪': 'right', 'cbar_size': '5%%', 'cbar_pad': 0.03}
```

  map kwargs: Additional keyword arguments to be added in the matplotlib contour plot call, see the make_map_plot function in the *plotting module*.

  line plot settings: Likewise, settings for line plots can be made, e.g. line widths and styles as well as axes configurations. There are a number of functions in the *plotting module* that handles line/scatter/box plots, see for example the fig_grid_setup and make_line_plot functions.

```
line grid setup = {'axes_pad': (11., 6.)}
line kwargs = {'lw': 2.5}
```

- **CLUSTER** The last section control the cluster type. You can choose between local pc and SLURM at the moment.

  cluster type: choose "local" for running on you local pc and "slurm" if you want to run RCAT on a HPC with a SLURM job scheduler and read information below. For local pc no other settings need to be made in this section.

  SLURM RCAT uses Dask to perform file managing and statistical analysis in an efficient way through parallelization. When applying Dask on queuing systems like PBS or Slurm, Dask-Jobqueue provides an excellent interface for handling such work flow. It is used in RCAT and to properly use Dask and Dask-Jobqueue on an HPC system you need to provide some information about that system and how you plan to use it. By default, when Dask-Jobqueue is first imported a configuration file is placed in ~/.config/dask/jobqueue.yaml. What is set in this file are the default settings being used. On Bi/NSC we have set up a default configuration file as below.

```
jobqueue:
    slurm:
    name: dask-worker

    # Dask worker options
    cores: 16
    memory: "64 GB"
    processes: 1

    interface: ib0
    death-timeout: 60
    local-directory: $SNIC_TMP

    # SLURM resource manager options
    queue: null
    project: null
    walltime: '01:00:00'
    job-extra: ['--exclusive']
```

  When default settings have been set up, the main properties that you usually want to change in the **CLUSTER** section are the number of nodes to use and wall time:

```
nodes = 15
slurm kwargs = {'walltime': '02:00:00', 'memory': '256GB', 'job_extra
↪': ['-C fat']}
```

  nodes: Sometimes you might need more memory on the nodes, and on Bi/NSC there are fat nodes available. If you want to use fat nodes, you can specify this through

```
slurm kwargs = {'walltime': '02:00:00', 'memory': '256GB', 'job_extra
↪': ['-C fat']}
```

3. **Run RCAT**

   When you have done your configuration and saved config_main.ini you can start the analysis step. The main program is located in the *rcat* directory and called RCAT_main.py. See point 1: *Setup folder structure* and run main RCAT_main.py from your analysis folder.

```
python <path-to-RCAT>/rcat/runtime/RCAT_main.py -c config_main.ini
```

Note: Don't forget to set $PYTHONPATH to your RCAT directory (<path-to-RCAT>).

1.3 RCAT Statistics

In the **STATISTICS** section in the file config_main.ini you specify the statistics to be done, in a Python dictionary structure.

```
stats = {
  'annual cycle': 'default',
  'seasonal cycle': {'stat method': 'mean', 'thr': {'pr': 1.0}},
  'percentile': {'resample resolution': ['day', 'max'], 'pctls': [90, 95, 99, 99.9]},
  'pdf': {'thr': {'pr': 1.0}, 'normalized': True, 'bins': {'pr': (0, 50, 1), 'tas':␣
↪(265, 290, 5)}}
  'diurnal cycle': {'dcycle stat': 'amount', 'stat method': 'percentile 95'}
  'moments': {'moment stat': {'pr': ['Y', 'max'], 'tas': ['D', 'mean']}},
}
```

The keys in stats are the statistical measures and values provides the settings/configurations to be applied to the specific statistical calculation. The statistics that are currently available in RCAT and their default settings are given in the *RCAT Statistics <stats_control_functions>* module. Some related information is also found in the API-reference *Statistics*. In particular, the default_stats_config function in that module specifies the statistics possible to calculate along with their properties. Many of the properties (or settings) are common for each of the measures, for example resample resolution, thr or chunk dimension, while others may be specific for the kind of statistics.

If you set *default* as the key value in stats, as is the case for *annual cycle* in the code snippet above, then (obviously) the default settings will be used. To modify the settings, the key value should be set to a dictionary containing the particular properties to be modified as keys and values with the modified item values.

1. **Common settings and properties** Here we list the most common settings and give some information of them.

   vars: String *variable_name* or list of strings *[variable_name_1, variable_name_2, . . .]* representing which variables (as defined in variables under **SETTINGS** in the main configuration file). If set to empty list [] the calculation will be performed for all defined variables. Some statistics are specifically for a certain variable and then this variable is set as default.

   resample resolution: If you want to resample input data (model or obs) to another time resolution before statistical computation, you can set this property to a list with two items; the first defines the temporal resolution (e.g. 3 hours, day, month, etc) and the second the method resampling used (taking the mean or sum for example). The xarray resample function is applied here which builds on the similar function in the pandas package. For example, resampling to 6 hourly data, taking the sum over intervening time steps would be defined as follows in the configuration file:

   ```
   resample resolution': ['6H', 'sum']
   ```

   The documentation of and available options for the resampling function can be found here (xarray) and here (pandas) (see DateOffset Objects section for frequency string options).

   chunk dimension: An important feature behind Dask parallelization is the use of *blocked algorithms*; to divide the data into chunks and then run computations on each block of data in parallel (see Dask documentation for more information). When working with multiple dimension data arrays in Dask you can chunk the data along different dimensions, depending on what kind of calculations you may want to

do. For example, when computing seasonal means of data with dimensions (time, lat, lon) it doesn't really matter along which dimension the data is chunked along. However, if you want to calculate time series percentiles for each grid point then chunking should be done in space ('lat', 'lon'). The chunk dimension property has two options; time/space. For example to chunk along time, set

```
'chunk dimension': 'time'
```

pool data: Boolean switch (True/False). Set to True if statistics should be done on pooled data, i.e. assembling all the grid points and time steps and then perform calculations. If you want the pooling to be done over certain sub-regions, then you need to specify these in the regions property in the main configuration file, *<path-to-RCAT>/config/config_main.ini*.

thr: Thresholding of data. The value of this (None is the default) should be a dictionary with keys defining variables and values an integer of float; e.g.

```
'thr': {'pr': 0.1, 'tas': 273}
```

2. **Specific settings and properties** Here we list more specific settings and give some information of them.

   stat method: In many of the available statistical calculations, computations can be done using various methods or moments (e.g. mean, sum, std, etc). For example, if calculating the diurnal cycle, one could compute the mean of all values for each time unit in the cycle or another measure such as a percentile value. This can be specified with this property. Default value is mean. To use a percentile, set (for 95th percentile);

   ```
   'stat method': 'percentile 95'
   ```

   dcycle stat: In the computation of the diurnal cycle (including harmonic fit) the *dcycle stat* defines whether to compute magnitudes or frequency of occurrences. For the former set it to 'amount', for the latter to 'frequency'. When calculating frequencies you must also set the 'thr' option, so for each unit of time in the cycle the occurrence above this threshold is calculated.

   hours (in diurnal cycle): The value of this property is a list of hours that should be used in the diurnal cycle computation. It might be changed if you want to compare data sets with different temporal resolution (this can also be achieved with the *resample resolution* option).

   normalized (in pdf): Boolean switch. With normalization, the normalized contribution (by the total mean) from each bin interval in the pdf (or frequency intensity distribution) is computed.

   normalized (in Rxx): In the Rxx function (see *statistics <statistics_functions>* module) the counts above the threshold is normalized by the total number of values if this property is set to True.

   moment stat: The moment statistical calculation involve a basic calculation on the data, such as means, sums or standard deviations. It is basically the same as the resample resolution property and the *moment stat* is set the same way. For example, if you want to calculate the annual maximum of the input data set.

   ```
   'moment stat': ['Y', 'max']
   ```

3. How do you add new statistical methods to RCAT? The code in RCAT is heavily based on xarray as well as dask. Xarray has been interfaced closely with dask applications so much of the things that can be done in xarray, like many (basic) statistical calculations, are already dask compliant and therefore relatively easy to implement in RCAT. If you would like to include any new such feature, have a look in the *RCAT Statistics* module, for example how the implementation of 'seasonal cycle' has been done.

   For more elaborate statistics, using for example functions created by the user (using standard numpy/python code), it may be a bit more complex. Xarray has a function called apply_ufunc which allows repeatedly applying a user function to xarray objects containing Dask arrays in an automatic way. See here for_some more information.

CHAPTER 2

How-To Guides

The purpose of this section is to provide a new user with an overview of RCAT – an easy-to-follow guide of RCAT applications, in order to quickly and easily get started with the tool. To this end you will find below a few rather simple instructions or 'recipes' that will allow you to do some basic analysis of model output and make simple visualizations of the results.

2.1 Preparation

To get started with the application examples it is expected that the RCAT environment has been installed. If not, go ahead and *do so* before continuation. The main configuration and setup of RCAT is done in *<path-to-RCAT>/config/config_main.ini*. It is therefore encouraged to read through the *configuration* and *statistics* sections before (or along with) going through the examples below.

Note: The Use Cases (and some parts of the source code) are in a few apects very specific to the HPC system at the National Supercomputer Centre (NSC) in Sweden; for example, available observation data sets and folder structure of model output. In future updates of RCAT, we strive to make it more general and flexible.

2.2 Use Cases

1. *The Annual and Seasonal Cycles*
2. *PDF's on different time scales*
3. *Diurnal Variations*

2.3 RCAT polygons

- *How to create and plot polygons*

2.3.1 Use Case 1: Annual & and Seasonal Cycles

In this example annual and seasonal statistics will be calculated and in most steps, changes are made in different sections of the configuration file, *<path-to-RCAT>/config/config_main.ini*. It is recommended to copy this file to your experiment folder. Detailed information of what can be configured and how can be found in *RCAT Configuration*.

How to calculate monthly and seasonal mean statistics?

Daily data from two models is used as input for calculation of monthly and seasonal means at each point of their respective (native) grids. Statistical results are written to disk in netcdf files; a separate file for each model and statistic (and sub-region if specified). No plotting is done here.

STEP 1: Data input

Under section **MODELS** you'll specify the path to model data. Configure for two models – *arome* and *aladin* – using the same time period and months. Since the annual and seasonal cycles will be calculated, all months are chosen.

```
arome = {
     'fpath': '/nobackup/rossby21/rossby/joint_exp/norcp/NorCP_AROME_ERAI_ALADIN_1997_
↪2017/netcdf',
     'grid type': 'reg', 'grid name': 'NEU-3',
     'start year': 1998, 'end year': 2002, 'months': [1,2,3,4,5,6,7,8,9,10,11,12]
     }
aladin = {
     'fpath': '/nobackup/rossby21/rossby/joint_exp/norcp/NorCP_ALADIN_ERAI_1997_2017/
↪netcdf',
     'grid type': 'reg', 'grid name': 'NEU-12',
     'start year': 1998, 'end year': 2002, 'months': [1,2,3,4,5,6,7,8,9,10,11,12]
     }
```

In this example we will not use any observations so no modifications are needed in the **OBS** section.

STEP 2: Variables

Under **SETTINGS** the full path to output directory should be defined. If folder doesn't exist already it will be created by RCAT.

```
output dir = /nobackup/rossby22/sm_petli/analysis/test_analysis
```

The key *variables* defines which variables to analyze along with some options regarding that particular variable. Since only models will be analyzed here, *'obs'* is set to None. Further, models will be kept at their respective grids, thus *'regrid to'* is also set to None. Statistics is configured for T2m (*tas*) and precipitation (*pr*) with daily data as input (*'freq'* set to *'day'*).

```
variables = {
    'pr': {'freq': 'day',
           'units': 'mm',
           'scale factor': None,
           'accumulated': True,
           'obs': None,
           'var names': None,
           'regrid to': None},
    'tas': {'freq': 'day',
```

(continues on next page)

(continued from previous page)

```
            'units': 'K',
            'scale factor': None,
            'accumulated': False,
            'obs': None,
            'var names': None,
            'regrid to': None},
    }
```

```
regions = ['Fenno-Scandinavia']
```

regions is a list of pre-defined regions – see available regions in *<path-to-RCAT>/rcat/utils/polygon_files* folder (see also *Polygons* module). Statistics will be selected for the specified sub-regions.

STEP 3: Select statistics

Under **STATISTICS** seasonal and annual cycles are chosen.

```
stats = {
    'annual cycle': 'default',
    'seasonal cycle': {'thr': {'pr': 1.0}},
    }
```

The *'default'* property means that default options for the particular statistic are used. All default options can be seen in the *default_stats_config* function in *RCAT Statistics*. For seasonal cycle, we choose to use a threshold for precipitation of *1.0* and so calculation is only based on wet days.

STEP 4: No plotting

Set validation plot to false – no plotting done in this example.

```
validation plot = False
```

STEP 5: Configure cluster

Under the **CLUSTER** section one should specify which type of cluster to use. Here, it is configured for a SLURM cluster. *nodes* specify the number of nodes to be used. In *cluster kwargs* a number of different options can be set (here specific for SLURM), for example walltime which is set to 2 hours.

```
cluster type = slurm
nodes = 10
cluster kwargs = {'walltime': '02:00:00'}
```

STEP 6: Run RCAT

To run the analysis run from terminal (see *Run RCAT* in *RCAT Configuration*):

```
python <path-to-RCAT>/rcat/runtime/RCAT_main.py -c config_main.ini
```

If successfully completed, output statistics netcdf files will be located in the sub-folder *stats* under the user-defined output directory. An *img* folder is also created, however, it will be empty as no plotting have been done.

Adding comparison to observations and visualize results

In order to include observations and visualize the end results, follow the procedure as in the previous example with the following changes introduced:

1. Under **OBS** section, choose same years and months as models

```
start year = 1998
end year = 2002
months = [1,2,3,4,5,6,7,8,9,10,11,12]
```

2. The *variables* property in **SETTINGS** section shall be modified:

   - Include observations; *'obs': ['EOBS20', 'ERA5']*. Also, scale factors are now included for observations as well.
   - Since models and observations will be compared, taking differences, the data must be on the same grid. Therefore, set *'regrid to': 'ERA5'*. This means that all data will be interpolated to the *ERA5* grid. Further, the *'regrid method'* needs to be set – *bilinear* for T2m and *conservative* for pr.

```
variables = {
    'pr': {'freq': 'day',
           'units': 'mm',
           'scale factor': None,
           'accumulated': True,
           'obs': ['EOBS20', 'ERA5'],
           'obs scale factor': [86400, 86400],
           'var names': None,
           'regrid to': 'ERA5',
           'regrid method': 'conservative'},
    'tas': {'freq': 'day',
            'units': 'K',
            'scale factor': None,
            'accumulated': False,
            'obs': ['EOBS20', 'ERA5'],
            'obs scale factor': None,
            'var names': None,
            'regrid to': 'ERA5',
            'regrid method': 'bilinear'},
    }
```

3. Under **PLOTTING**, *validation plot* should be set to *True* to enable plotting. It is possible to configure the visualization in different ways, for example various map configurations in map plots or the looks of line plots. However, for simplicity here, the default configurations will be used, which means setting all properties to an empty dictionary ({}).

```
validation plot = True

map configure = {}
map grid setup = {}
map kwargs = {}

line grid setup = {}
line kwargs = {}
```

With these modifications in place, run RCAT again (STEP 6 above).

2.3.2 Use Case 2: Probability distributions

In the following RCAT is applied to calculate standard empirical probability distribution functions. Similar to *Use Case 1* most changes will be done in the configuration file, *<path-to-RCAT>/config/config_main.ini*.

Create hourly PDF statistics and visualize the results

In the first example PDF's based on hourly data for historical and scenario simulations will be calculated for precipitation and T2m. Output statistics are then compared in line plots for specified regions.

STEP 1: Data input

Under **MODELS** section configure for two *arome* simulations – historic (*arome_his*) and future scenario (*arome_scn*). Thus, different years are specified, however, in the example here months 6,7,8 are specified so that only data for June, July and August is extracted.

```
arome_his = {
     'fpath': '/nobackup/rossby21/rossby/joint_exp/norcp/NorCP_AROME_ECE_ALADIN_1985_
↪2005/netcdf',
     'grid type': 'reg', 'grid name': 'NEU-3',
     'start year': 1990, 'end year': 1994, 'months': [6,7,8]
     }
arome_scn = {
     'fpath': '/nobackup/rossby21/rossby/joint_exp/norcp/NorCP_AROME_ECE_ALADIN_RCP85_
↪2080_2100/netcdf',
     'grid type': 'reg', 'grid name': 'NEU-3',
     'start year': 2090, 'end year': 2094, 'months': [6,7,8]
     }
```

We're looking at climate change signal in the model, so the **OBS** section can be left as is.

STEP 2: Variables

Set output directory under the **SETTINGS** section.

The key *variables* defines which variables to analyze along with some options regarding that particular variable. Since only models will be analyzed here, *obs* is set to None. Further, models will be kept at their respective grids, thus *regrid to* is also set to None. Statistics is configured for T2m (*tas*) and precipitation (*pr*) with hourly data as input (*freq* set to *1H*).

Specify regions the *regions* key for which statistics will be selected for.

```
output dir = /nobackup/rossby22/sm_petli/analysis/test_pdf_analysis

variables = {
    'pr': {'freq': '1H',
           'units': 'mm',
           'scale factor': None,
           'accumulated': True,
           'obs': None,
           'var names': None,
           'regrid to': None},
    'tas': {'freq': '1H',
            'units': 'K',
```

(continues on next page)

(continued from previous page)

```
            'scale factor': None,
            'accumulated': False,
            'obs': None,
            'var names': None,
            'regrid to': None},
    }

regions = ['Scandinavia']
```

STEP 3: Select statistics

Under **STATISTICS** *pdf* will be specified. A list of bins will be used in the pdf. If not given here these bins will be defined automatically in RCAT by taking minimum and maximum of input data. This can be quite crude and not so representative, so it is suggested to define them here under the *pdf* key. They are specified in a dictionary where the keys are input variables and the values are the respective bin definitions. The bin definition is a list/tuple with start, stop and step values. For example, for precpitation a list of bins starting with 0 and ending with 50 using a step of 1 is defined here.

```
stats = {
   'pdf': {'bins': {'pr': (0, 50, 1), 'tas': (264, 312, 1)}}
    }
```

See the *default_stats_config* function in *RCAT Statistics* module for the default options for pdf.

STEP 4: Plotting

- Under **PLOTTING**, *validation plot* should be set to *True* to enable plotting. Plotting of pdf's will be line plots only (regions should therefore be specified). We only specify linewidths to be 2.5.

```
validation plot = True

map configure = {}
map grid setup = {}
map kwargs = {}

line grid setup = {}
line kwargs = {'lw': 2.5}
```

STEP 5: Configure cluster

The number of nodes to be used in the selected SLURM cluster is set to 20 (increase if needed) and a walltime of 2 hours.

```
cluster type = slurm
nodes = 20
cluster kwargs = {'walltime': '02:00:00'}
```

STEP 6: Run RCAT

To run the analysis run from terminal (see *Run RCAT* in *RCAT Configuration*):

```
python <path-to-RCAT>/rcat/runtime/RCAT_main.py -c config_main.ini
```

Output statistics files will be located in the sub-folder *stats* under the user-defined output directory.

Calculate PDF's for daily maximum values instead

Imagine one would like to do the same kind of statistical analysis as above, however, with a different temporal resolution and/or time statistic on the input data. For example, let's assume that pdf's should be calculated for daily maxmimum data instead. How can this be achieved?

This can be done during RCAT runtime, using an option in the *stats* property (under **SETTINGS**) called *resample resolution*. It is specified by a list/tuple with two locations; the first index represents the time resolution sought after and the second location the statistic used for each sample in the resampling. In the example here data is resampled into daily maximum values:

```
stats = {
   'pdf': {'bins': {'pr': (0, 50, 1), 'tas': (264, 312, 1)}, 'resample resolution': [
↪'D', 'max']}
    }
```

When set, run RCAT again.

2.3.3 Use Case 3: Diurnal Cycles

One main advantage of RCAT is that it can run analyses of large data sets through the use of parallelization (using the *dask* module). In principle it involves splitting the data into several "chunks" (in the space and/or time dimensions) and then run a set of operations on each of the chunk in parallel. Read more about it on dask homepage.

Depending on the analysis you want to run on your data, you might consider chunking your data differently. If, for example, you would like to calculate a quantile value for the data over all time steps then you should do the chunking in space only so that each chunk has all time steps available. Here, RCAT will be applied to calculate diurnal cycles of some model output using different statistical measures and how the splitting/chunking of data matters.

Similar to *Use Case 1* most changes will be done in the configuration file, *<path-to-RCAT>/config/config_main.ini*.

Calculate diurnal cycles of mean CAPE and plot the results

STEP 1: Data input

Under section **MODELS** specify the path to model data and set start and end years as well as months to analyze.

```
arome = {
     'fpath': '/nobackup/rossby21/rossby/joint_exp/norcp/NorCP_AROME_ERAI_ALADIN_1997_
↪2017/netcdf',
     'grid type': 'reg', 'grid name': 'NEU-3',
     'start year': 1998, 'end year': 2002, 'months': [5,6,7,8,9]
     }
aladin = {
     'fpath': '/nobackup/rossby21/rossby/joint_exp/norcp/NorCP_ALADIN_ERAI_1997_2017/
↪netcdf',
     'grid type': 'reg', 'grid name': 'NEU-12',
     'start year': 1998, 'end year': 2002, 'months': [5,6,7,8,9]
     }
```

If you would like to include observations as well, set accordingly in the **OBS** section.

STEP 2: Variables

Set output directory under the **SETTINGS** section.

In the key *variables* we specify in this example *pcape* (a specific model version of CAPE) available on 3 hourly time resolution. If only models will be analyzed set *'obs'* to None. *'regrid to'* is set to the coarser grid of the two models and data is interpolated using the *bilinear* method.

Specify region(s) in the *regions* key for which statistics will be selected, and finally plotted, for.

```
output dir = /nobackup/rossby22/sm_petli/analysis/test_dcycle_analysis

variables = {
    'pcape': {'freq': '3H',
              'units': 'J/kg',
              'scale factor': None,
              'accumulated': False,
              'obs': None,
              'var names': None,
              'regrid to': 'aladin',
              'regrid method': 'bilinear'},
    }

regions = ['Sweden', 'Denmark', 'Norway', 'Finland']
```

STEP 3: Select statistics

The statistics, *diurnal cycle*, is specified under the *stats* key in the **STATISTICS** section. Default options for diurnal cycle is found in the *default_stats_config* function in *RCAT Statistics*. In default settings, *hours* is set to all 24 hours in a day. Since the data here is on 3 hourly resolution we specify these hours. The *stat method* (the statistical measure) for each hour is *mean* in default and it is kept here, and the data is chunked in the time dimension (also default so not specified here).

```
stats = {
   'diurnal cycle': {'hours': [0, 3, 6, 9, 12, 15, 18, 21]}
    }
```

STEP 4: Plotting

Under **PLOTTING**, *validation plot* should be set to *True* to enable plotting. Plotting of diurnal cycles will be both maps (for each hour) and line plots for specified regions.

```
validation plot = True

map configure = {}
map grid setup = {}
map kwargs = {}

line grid setup = {}
line kwargs = {'lw': 2.5}
```

STEP 5: Configure cluster

The number of nodes to be used in the selected SLURM cluster is set to 10 (increase if needed) and a walltime of 2 hours.

```
cluster type = slurm
nodes = 10
cluster kwargs = {'walltime': '02:00:00'}
```

STEP 6: Run RCAT

To run the analysis run from terminal (see *Run RCAT* in *RCAT Configuration*):

```
python <path-to-RCAT>/rcat/runtime/RCAT_main.py -c config_main.ini
```

Output statistics and image files will be located under the user-defined output directory in the *stats* and *imgs* sub-folders respectively

Calculate diurnal cycles of 99th percentile CAPE values

Instead of the mean value for each hour in the diurnal cycle (at any grid point in the domain) it could be meaningful to use another statistical measure, for example the 99th percentile. To do this, in addition to changing the *stat method* value, one will need to have all time steps available for the calculation and thus the *chunk dimension* should be changed from *'time'* (default) to *'space'*:

```
stats = {
   'diurnal cycle': {'hours': [0, 3, 6, 9, 12, 15, 18, 21], 'stat method':
↪'percentile 99', 'chunk dimension': 'space'}
    }
```

When set, run RCAT again.

2.3.4 Polygons in RCAT

– How to plot them and create new ones

Polygons are used in RCAT to extract or select data and statistics for specified sub-regions or areas. These consist of text files containing information of latitudes and longitudes for the area and are stored under *<path-to-RCAT>/rcat/utils/polygon_files/*. The *Polygons* module use these polygons to do the extraction (with the *mask_region* function) and it also has a number of different help functions to create new polygons (for use in RCAT or elsewhere) and plot them conveniently.

The following tutorial will describe some of this functionality along with a few examples.

Plot a polygon

The *Polygons* module has a function called *plot_polygon* which allows you to plot one of the existing polygons on map. There are a couple of ways to apply the function – either from within python (or in a script) where the module is imported and plotting function calls can be made, or the function call can be made directly from command line:

```
>> import rcat.utils.polygons as pg
>> pg.plot_polygon?
Signature: pg.plot_polygon(polygon, savefig=False, figpath=None)
Docstring:
Plot polygon on map.

Parameters
----------
polygon: string or list
    Name of polygon as defined by poly_dict dictionary in 'polygons'
    function, or list with polygon coordinates [[lon1, lat1], [lon2, lat2],
    ..., [lon1, lat1]].
savefig: boolean
    If True, figure is saved to 'figpath' location ('figpath' must be set!).
    If false, figure is displayed on screen.
figpath: string
    Path to folder for saved polygon figure.
```

The *plot_polygon* function takes a polygon name as input chosen from the collection of existing polygons in *<path-to-RCAT>/rcat/utils/polygon_files/*. [Note that the polygon name is the text file name without '.txt' and underscores replaced by white space.] Also new polygons can be plotted if providing a list with polygon coordinates (not possible when executing function from command line!). The keyword arguments allows to save the polygon plot, if not it is just displayed.

If you do not know which polygons are available, you can easily print them:

```
>> pg.polygons(poly_print=True)

Available polygons/regions:

    Alps
    Black Sea
    British Isles
    Central Europe
    Denmark
    East Europe
    Fenno-Scandinavia
    Finland
    France
    Germany
    Iberian peninsula
    Mediterranean Sea
    Netherlands
    Norway
    South-East Europe
    Spain
    Sweden
    Switzerland
    United Kingdom
    West Europe
```

To plot the polygon of British Isles:

```
>> pg.plot_polygon('British Isles')
```

This function call can be made directly from the command line. Run the *<path-to-RCAT>/rcat/utils/polygons.py* script (make sure it is executable) providing appropriate arguments:

```
./polygons.py --help
./polygons.py -p plot -a "British Isles" --save True --figpath $HOME
```

Create a new polygon

New regions/polygons can be created using the *create_polygon* function, either from within python or from the command line. The call involves a set of instructions where the user is continuously prompted for information and actions.

From command line:

```
./polygons.py -p create
```

The creation part is made by clicking pointer on a displayed map. If you want to save selected polygon to RCAT, make sure to provide correct folder path and an appropriate polygon name. Once saved it will automatically be ready for RCAT – check for example by printing available polygons:

```
./polygons.py -p printareas
```

CHAPTER 3

Development

RCAT is written and maintained mainly by Petter Lind and David Lindstedt for analysis of high resolution regional climate output. It's now an open source project and as such, anyone is welcome to contribute. Please look through the page on how to do improve the software.

3.1 Contribution guidelines

- PEP 8
- Modular
- Writing tests
- Code review

CHAPTER 4

API-reference

4.1 User API

4.1.1 Visualization

Color routines

The routines below provides access to predefined (Matplotlib) or self-produced colors and color maps.

rcat.plot.colors.getcolormap(cmap_name[, custom])	Function to retrieve colormap, either customized (custom=True) or available through Matplotlib predefined colormaps.
rcat.plot.colors.getsinglecolor(color_name)	Function to retrieve single custom color.
rcat.plot.colors.norm_colors(bounds, ncolors)	In addition to min and max of levels, this function takes as arguments boundaries between which data is to be mapped.

rcat.plot.colors.getcolormap

rcat.plot.colors.**getcolormap**(*cmap_name*, *custom=False*)

Function to retrieve colormap, either customized (custom=True) or available through Matplotlib predefined colormaps.

Parameters

- **cmap_name** (*string*) – String, giving name of colormap to be retrieved.
- **custom** (*Boolean*) – Logical indicating self-produced (custom=True) or Matplotlib colormap.

Returns **cmap**

Return type Matplotlib colormap object

rcat.plot.colors.getsinglecolor

rcat.plot.colors.**getsinglecolor**(*color_name*)

Function to retrieve single custom color.

Parameters **color_name** (*string*) – String, giving name of color to be retrieved.

Returns color

Return type Matplotlib color object

rcat.plot.colors.norm_colors

rcat.plot.colors.**norm_colors**(*bounds*, *ncolors*, *clip=False*)

In addition to min and max of levels, this function takes as arguments boundaries between which data is to be mapped. The colors are then linearly distributed between these 'bounds'.

Graphic routines

Functions for different plots such as scatterplots and mapplots,initiation and configuration of figure objects etc

rcat.plot.plots.figure_init([plottype, ...])	Setting up a figure object
rcat.plot.plots.image_grid_setup([figsize, ...])	Set up the plot axes using mpl_toolkits.axes_grid1.ImageGrid Used primarily when plotting maps or for image analysis For more information on available settings: https://doc.ebichu.cc/matplotlib/mpl_toolkits/axes_grid1/overview.html
rcat.plot.plots.get_nrow_ncol(npanels)	Return number of rows and columns from a given total no of panels for a grid
rcat.plot.plots.fig_grid_setup([figsize, ...])	Set up the plot axes using pyplot.subplots
rcat.plot.plots.axes_settings(ax[, ...])	Configuration of axes; titles and labels
rcat.plot.plots.map_axes_settings(fig, axs)	Settings for map plot axes
rcat.plot.plots.make_scatter_plot(grid, ...)	Create a scatter plot
rcat.plot.plots.make_raster_plot(data[, ...])	Create a raster plot
rcat.plot.plots.make_line_plot(grid, ydata)	Create a line plot
rcat.plot.plots.make_box_plot(grid, data[, ...])	Create a box plot
rcat.plot.plots.custom_legend(colors, labels)	Creates a list of matplotlib Patch objects that can be passed to the legend(...) function to create a custom legend.
rcat.plot.plots.gen_clevels(data, nsteps[, ...])	Create contour levels based on min and max of input data

Continued on next page

Table 2 – continued from previous page

rcat.plot.plots.map_setup(grid, lats, lons)	Create a basemap object to be used in the overlaying of 2d plots.
rcat.plot.plots.make_map_plot(data[, grid, ...])	Producing map plots
rcat.plot.plots.plot_map(m, x, y, data, ...)	Producing a map plot
rcat.plot.plots.image_colorbar(cs, cbaxs[, ...])	Add colobar to map plot

rcat.plot.plots.figure_init

rcat.plot.plots.**figure_init**(*plottype='line'*, *printtypes=False*)

Setting up a figure object

Parameters

- **plottype** (*string*) – Type of plot to make
- **printtypes** (*boolean*) – If available plottypes should be printed on screen

rcat.plot.plots.image_grid_setup

rcat.plot.plots.**image_grid_setup**(*figsize=(12, 12)*, *fshape=(1, 1)*, ***grid_kwargs*)

Set up the plot axes using mpl_toolkits.axes_grid1.ImageGrid Used primarily when plotting maps or for image analysis For more information on available settings: https://doc.ebichu.cc/matplotlib/mpl_toolkits/axes_grid1/overview.html

Parameters

- **figsize** (*tuple*) – Size of figure in inches; (width, height)
- **fshape** (*tuple*) – setting the shape of figure (nrow, ncol)
- ****grid_kwargs** (*Additional keyword arguments*) –

Returns

- **fig** (*Figure object*)
- **grid** (*AxesGrid object*)

Examples of **kwargs with default values: direction="row", axes_pad=0.02, add_all=True, share_all=False, label_mode="L", aspect=True, cbar_mode=None, cbar_location="right", cbar_size="5%", cbar_pad=None,

rcat.plot.plots.get_nrow_ncol

rcat.plot.plots.**get_nrow_ncol**(*npanels*)

Return number of rows and columns from a given total no of panels for a grid

rcat.plot.plots.fig_grid_setup

rcat.plot.plots.**fig_grid_setup**(*figsize=(12, 12)*, *fshape=(1, 1)*, *direction='row'*, *axes_pad=(None, None)*, ***grid_kwargs*)

Set up the plot axes using pyplot.subplots

Parameters

- **figsize** (*tuple*) – Size of figure in inches; (width, height)
- **fshape** (*tuple*) – setting the shape of figure (nrow, ncol)
- **direction** (*string*) – 'row' or 'col'; rowwise or columnwise order of axes instances
- **axes_pad** (*tuple*) – padding (height,width) between edges of adjacent subplots
- ****grid_kwargs** (*Additional keyword arguments*) –

Returns

- **fig** (*Figure object*)
- **grid** (*List with axes instances*)

rcat.plot.plots.axes_settings

rcat.plot.plots.**axes_settings**(*ax*, *figtitle=None*, *xlabel=None*, *ylabel=None*, *xtlabels=None*, *ytlabels=None*, *xticks=None*, *yticks=None*, *xlim=None*, *ylim=None*, *color='k'*, *fontsize='xx-large'*, *fontsize_lbls='xx-large'*, *fontsize_title='xx-large'*, *ftitle_location='center'*)

Configuration of axes; titles and labels

Parameters

- **ax** (*axes object*) – Axes object assoicated with figure
- **figtitle** (*string*) – The headtitle
- **ylabel** (*xlabel,*) – Labels of the axes
- **yticks** (*xticks,*) – If set, location of ticks
- **ytlabels** (*xtlabels,*) – Contains tick labels (corresponding to ticks location)
- **ylim** (*xlim,*) – Limits of the axes
- **color** (*str*) – Color for the xtick labels
- **fontsize/fontsize_lbls/fontsize_title** (*string or int*) – Size of font for axes labels, ticklabels, and figure title respectively
- **ftitle_location** (*str*) – Horizontal alignment of figure title; center (default), right or left.

rcat.plot.plots.map_axes_settings

rcat.plot.plots.**map_axes_settings**(*fig*, *axs*, *figtitle=None*, *headtitle=None*, *time_mean=None*, *time_units=None*, *fontsize='x-large'*, *fontsize_htitle='xx-large'*)

Settings for map plot axes

Parameters

- **fig** (*Object handle*) – Figure handle
- **axs** (*Axes objects*) – Single object or list with axes from map plot
- **figtitle** (*Strings*) – Single title or list of titles for each plot in figure
- **headtitle** (*String*) – Head title of figure

- **time_mean** (*string*) – If maps should be labeled according to time averages; 'season'/'month'/'hour'
- **time_units** (*list/array*) – Optional. If time_mean is set, time_units is a list of seasons, months or hours to be used in the labeling.
- **fontsize** (*string or int*) – Size of font for figure title
- **fontsize_htitle** (*string or int*) – Size of font for suptitle

rcat.plot.plots.make_scatter_plot

rcat.plot.plots.**make_scatter_plot**(*grid*, *xdata*, *ydata*, *sdata=None*, *fcolors=None*, *ecolors=None*, *lbl_fontsize='large'*, *axis_type='linear'*, *labels=None*, ***sc_kwargs*)

Create a scatter plot

Parameters

- **grid** (*AxesGrid object*) – returned from the 'fig_grid_setup' function
- **xdata/ydata** (*Array/list*) – 1D array or list of 1D arrays with data for x/y axis
- **fcolors** (*array/list*) – List of colors to be used for each data set in input data. This is separate from 'color'/'c' option available from matplotlib.scatter call (and set in sc_kwargs) where all individual input data sets will have that specific color/s. If set, then ecolors need also be supplied.
- **ecolors** (*array/list*) – List of edge colors to be used for each data set in input data. This is separate from 'edgecolors'/'ec' option available from matplotlib.scatter call (and set in sc_kwargs) where all individual input data sets will have that specific color/s. If set, then fcolors need also be supplied.
- **lbl_fontsize** (*string/float*) – Fontsize for legend labels
- **axis_type** (*str*) – Linear or log axes: 'linear' (defualt), 'logx'/'logy'/'logxy' (log x, y or both axes).
- **labels** (*String/List*) – String or list of strings with legend labels
- ****sc_kwargs** (*keyword arguments*) – arguments (key=value) that can be used in pyplot.scatter See matplotlib.org for more information

Returns **axs** – The axes objects created for each plot

Return type Axes objects

rcat.plot.plots.make_raster_plot

rcat.plot.plots.**make_raster_plot**(*data*, *grid=None*, *clevs=None*, *norm=None*, *cmap='viridis'*, ***rs_kwargs*)

Create a raster plot

Parameters

- **grid** (*AxesGrid object*) – returned from the 'image_grid_setup' function
- **data** (*List*) – List with 2D array(s) of data
- **cmap** (*string/list*) – String or list with strings of predefined Matplotlib colormaps. Defaults to 'viridis'

- **clevs** (*Iterable data structure*) – Consisting of lists with defined contour levels; e.g. (np.arange(1,10,2), [0,2,4,6,8]), [np.arange(100,step=5)]*3
- **norm** (*BoundaryNorm object*) – Object generated from matplotlib.colors.BoundaryNorm function. Generate a colormap index based on discrete intervals.
- ****rs_kwargs** (*keyword arguments*) – arguments (key=value) that can be used in pyplot.imshow See matplotlib.org for more information

Returns

- **axs** (*Axes objects*) – The axes objects created for each plot
- **rasters** (*Plot objects*) – The raster plot objects created for each plot

rcat.plot.plots.make_line_plot

rcat.plot.plots.**make_line_plot**(*grid*, *ydata*, *xdata=None*, *labels=None*, *lbl_fontsize='x-large'*, *axis_type='linear'*, ***lp_kwargs*)

Create a line plot

Parameters

- **grid** (*Axis object*) – returned from the 'fig_grid_setup' function
- **ydata** (*array/list*) – Required. 1D array or list of 1D arrays with data for y axis
- **xdata** (*array/list*) – Optional. 1D array or list of 1D arrays with data for x axis
- **labels** (*string/list*) – String or list of strings with line labels
- **lbl_fontsize** (*string/float*) – Fontsize for legend labels
- **axis_type** (*str*) – Linear or log axes: 'linear' (defualt), 'logx'/'logy'/'logxy' (log x, y or both axes).
- ****lp_kwargs** (*keyword arguments*) – arguments (key=value) that can be used in pyplot.line See matplotlib.org for more information

Returns **axs** – The axes objects created for each plot

Return type Axes objects

rcat.plot.plots.make_box_plot

rcat.plot.plots.**make_box_plot**(*grid*, *data*, *labels=None*, *leg_labels=None*, *grouped=False*, *box_colors=None*, ***box_kwargs*)

Create a box plot

Parameters

- **grid** (*AxesGrid object*) – returned from the 'fig_grid_setup' function
- **data** (*List/Array*) – 1d array or list of 1d arrays with data for boxplot one box.
- **labels** (*str/list*) – String or a list of strings with xtick labels (for each box/group of boxes)
- **leg_labels** (*str/list*) – String or a list of strings with legend labels (mostly used for grouped boxplots).

- **grouped** (*boolean*) – Whether to plot grouped boxplot. If True, input data must be a dictionary. See _grouped_boxplot function for more info.
- **box_colors** (*array/list*) – Optional list of colors to be used for the boxes.
- ****box_kwargs** (*keyword arguments*) – arguments (key=value) that can be used in pyplot.boxplot See matplotlib.org for more information

Returns

- **axs** (*list*) – Axes objects for each plot
- **bps** (*list*) – Each item in list is a dictionary mapping each component of the boxplot to a list of the *.Line2D* instances created.

rcat.plot.plots.custom_legend

rcat.plot.plots.**custom_legend**(*colors*, *labels*, *linestyles=None*)

Creates a list of matplotlib Patch objects that can be passed to the legend(...) function to create a custom legend.

Parameters

- **colors** (*list*) – A list of colors, one for each entry in the legend. You can also include a linestyle, for example: 'k–'
- **labels** (*list*) – A list of labels, one for each entry in the legend.

rcat.plot.plots.gen_clevels

rcat.plot.plots.**gen_clevels**(*data*, *nsteps*, *robust=None*)

Create contour levels based on min and max of input data

Parameters

- **data** (*array*) – data array
- **nsteps** (*int*) – number of levels to be produced
- **robust** (*None or string*) – If to soften the max/min limits due to extreme values; "top"/"bottom"/"both", which end(s) to soften.

rcat.plot.plots.map_setup

rcat.plot.plots.**map_setup**(*grid, lats, lons, proj='stere', lon_0=None, lat_0=None, lat_1=None, res='l', zoom='crnrs', crnr_vals=None, zoom_geom=[None, None], **mkwargs*)

Create a basemap object to be used in the overlaying of 2d plots.

Parameters

- **grid** (*list*) – List with axes objects for each plot in figure
- **lats** (*array(s)*) – array or list of arrays with latitudes
- **lons** (*array(s)*) – array or list of arrays with longitudes
- **proj** (*string*) – the projection to be used (so far only two proj's: sterographic and lambert conformal)
- **lat_0/lon_0** (*float*) – centre latitude and longitude location of map

- **lat_1** (*float*) – first standard parallell. if proj='lcc', lat_1 must be given.
- **res** (*string*) – resolution (low 'l', intermediate 'i' or high 'h')
- **zoom** (*string*) – "cornrs" or "geom", how to set the extent of map
- **crnr_vals** (*1D list/array*) – List of values for corners given as [llcrnrlat, llcrnrlon, urcrnrlat, urcrnrlon]
- **zoom_geom** (*1D list/array*) – Integers setting map geometry [width, height] in meters.
- ****mkwargs** (*key word arguments*) – Additional arguments provided to Basemap call. See http://matplotlib.org/basemap/# for more info.

Returns

- **m** (*Basemap object*) – map object
- **coords** (*list*) – list containing x,y arrays of the transformed longitudes to map coordinates

rcat.plot.plots.make_map_plot

rcat.plot.plots.**make_map_plot**(*data*, *grid=None*, *map_obj=None*, *coords=None*, *lats=None*, *lons=None*, *cmap=None*, *clevs=None*, *robust=None*, *norm=None*, *mesh=False*, *filled=True*, ***map_kwargs*)

Producing map plots

Parameters

- **data** (*array/list/tuple*) – Array or list/tuple of 2D data array(s) to plot
- **grid** (*AxesGrid object*) – returned from the 'image_grid_setup' function If not provided it is generated using number of data arrays as input.
- **map_obj** (*tuple*) – List with Basemap objects returned from 'map_setup' function. If not provided, it is generated within this function.
- **coords** (*tuple*) – List with arrays of map coordinates; returned from 'map_setup' function
- **lons** (*lats,*) – If map_obj is not provided, latitudes and longitudes need to be provided to setup a Basemap object.
- **cmap** (*string/list*) – String or list with strings of predefined Matplotlib colormaps. For filled contour plots it defaults to 'viridis'.
- **clevs** (*Iterable data structure*) – Consisting of lists with defined contour levels; e.g. (np.arange(1,10,2), [0,2,4,6,8]), [np.arange(100,step=5)]*3
- **robust** (*string*) – See gen_clevs function for info
- **norm** (*BoundaryNorm object*) – Object generated from matplotlib.colors.BoundaryNorm function. Generate a colormap index based on discrete intervals.
- **mesh** (*boolean*) – Whether to plot data as mesh. If false (default), contour plot is made.
- **filled** (*boolean*) – Whether to color fill between contours or not. Defaults to True
- ****map_kwargs** (*keyword arguments*) – arguments (key=value) that can be used in pyplot.contour(f) See http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.contour

Returns **mplots** – List with map plot instances

Return type List

rcat.plot.plots.plot_map

rcat.plot.plots.**plot_map**(*m*, *x*, *y*, *data*, *clevs*, *cmap*, *norm*, *mesh*, *filled*, ***map_kwargs*)

Producing a map plot

Parameters

- **m** (*Basemap object*) – Handle for the map object returned from 'map_setup' function
- **data** (*numpy array*) – 2D data array to plot
- **x,y** (*numpy arrays*) – Arrays of map coordinates; returned from 'map_setup' function
- **clevs** (*List/array*) – Contour levels
- **cmap** (*string*) – Color map. See http://matplotlib.org/users/colormaps.html for more information.
- **norm** (*Boundary norm object*) – Normalize data to [0,1] to use for mapping colors
- **mesh** (*boolean*) – Whether to plot data as mesh. If false (default), contour plot is made.
- **filled** (*Boolean*) – Whether to color fill between contours or not. Defaults to True
- ****map_kwargs** (*keyword arguments*) – arguments (key=value) that can be used in pyplot.contour(f) See http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.contour

Returns cs

Return type Contour plot object

rcat.plot.plots.image_colorbar

rcat.plot.plots.**image_colorbar**(*cs*, *cbaxs*, *title=None*, *labelspacing=1*, *labelsize='x-large'*, *formatter='{:.2f}'*, ***cbar_kwargs*)

Add colobar to map plot

Parameters

- **cs** (*Plot object*) – Such as an image (imshow) or a contour set (with contourf)
- **cbaxs** (*cbar axis object/list*) – Colorbar axis object or list with axes objects for each plot in figure
- **title** (*list*) – list of strings with colorbar titles
- **labelspacing** (*int*) – Label spacing; the integer value represents the number of steps between each label. 1 show each label (default), 2 every second, etc.
- **labelsize** (*str/int*) – Size of labels; integer value or a string (e.g. 'large')
- ****cbar_kwargs** (*keyword arguments*) – arguments (key=value) that can be used in pyplot.colobar See http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.colorbar

4.1.2 Statistics

Arithmetic routines

Functions for various arithmetic calculations.

rcat.stats.arithmetics.run_mean(x, N[, mode])	Calculate running mean

rcat.stats.arithmetics.run_mean

rcat.stats.arithmetics.**run_mean**(*x*, *N*, *mode='valid'*)

Calculate running mean

Return running mean of data vector x where N is the window size. mode key word argument describes how the edges should be handled. See numpy.convolve for more information.

ASoP

Analyzing Scales of Precipitation.

rcat.stats.ASoP.asop(data[, keepdims, axis, ...])	Calculate ASoP parameters.
rcat.stats.ASoP.bins_calc(n[, bintype])	Calculates bins with edges according to Eq.

rcat.stats.ASoP.asop

rcat.stats.ASoP.**asop**(*data*, *keepdims=False*, *axis=0*, *bins=None*, *thr=None*, *return_bins=False*)

Calculate ASoP parameters.

Parameters

- **data** (*array*) – 2D or 1D array of data. All data points are collectively used in the asop calculation unless 'keepdims' is True. Then calculation is performed along zeroth axis (expected time dimension).
- **keepdims** (*boolean*) – If data is 2d (time in third dimesion) and keepdims is set to True, calculation is applied to the dimension defined by axis argument (default 0) and returns a 2d array of asop components. If set to False (default) all values are collectively assembled before calculation.
- **axis** (*int*) – The axis over which to apply the calculation if keepdims is set to True. Default is 0.
- **bins** (*list/array*) – Defines the bin edges, including the rightmost edge, allowing for non-uniform bin widths. If bins is set to 'None' they will be automatically calculated using Klingaman bins; function bins_calc in this module.
- **thr** (*float*) – Value of threshold if thresholding data. Default None.
- **return_bins** (*boolean*) – If set to True (default False), bins that have been used in the calculation are returned.

Returns

- **Cfactor** (*array*) – data array with relative contribution per bin to the total mean.
- **FCfactor** (*array*) – data array with relative contribution per bin independent of the total mean.
- **bins_ret** (*array*) – If return_bins is True, the array of bin edges is returned.

rcat.stats.ASoP.bins_calc

rcat.stats.ASoP.**bins_calc**(*n*, *bintype='Klingaman'*)

Calculates bins with edges according to Eq. 1 in Klingaman et al. (2017); https://www.geosci-model-dev.net/10/57/2017/

Parameters

- **n** (*array/list*) – 1D array or list with bin numbers
- **bintype** (*str*) – The type of bins to be calculated; 'Klingaman' (see reference) or 'exponential' for exponential bins.

Returns **bn** – 1D array of bin edges

Return type array

Bootstrapping

Routines for bootstrap calculations.

rcat.stats.bootstrap.block_bootstr(data[, ...])	Calculate block bootstrap samples.

rcat.stats.bootstrap.block_bootstr

rcat.stats.bootstrap.**block_bootstr**(*data*, *block=5*, *nrep=500*, *nproc=1*)

Calculate block bootstrap samples.

This is a block boostrap function, converted from R into python, based on: http://stat.wharton.upenn.edu/~buja/STAT-541/time-series-bootstrap.R

Parameters

- **data** (*list/array*) – 1D data array on which to perform the block bootstrap.
- **block** (*int*) – the block length to be used. Default is 5.
- **nrep** (*int*) – the number of resamples produced in the bootstrap. Default is 500.
- **nproc** (*int*) – Number of processors, default 1. If larger than 1, multiple processors are used in parallell using the multiprocessing module.

Returns **arrBt** – 2D array with bootstrap samples; rows are the samples, columns the values.

Return type Array

Climate indices

Routines for various climate index calculations.

rcat.stats.climateindex.hotdays_calc(data, ...)	Calculate number of hotdays.
rcat.stats.climateindex.extr_hotdays_calc(...)	Calculate number of extreme hotdays.

Continued on next page

Table 6 – continued from previous page

rcat.stats.climateindex.tropnights_calc(data)	Calculate number of tropical nights.
rcat.stats.climateindex.ehi(data, thr_95[, ...])	Calculate Excessive Heat Index (EHI).
rcat.stats.climateindex.cdd_calc	
rcat.stats.climateindex.Rxx(data[, thr, ...])	Rxx mm, count of any time units (days, hours, etc) when precipitation xx mm: Let RRij be the precipitation amount on time unit i in period j.
rcat.stats.climateindex.RRpX(data, percentile)	RRpX mm, total amount of precipitation above the percentile threshold pX; RR pX mm: Let RRij be the daily precipitation amount on day i in period j.
rcat.stats.climateindex.RRtX(data, thr[, ...])	RRtX mm, total amount of precipitation above the threshold 'thr'.
rcat.stats.climateindex.SDII(data[, thr, ...])	SDII, Simple pricipitation intensity index: Let RRwj be the daily precipitation amount on wet days, w (RR 1mm) in period j.

rcat.stats.climateindex.hotdays_calc

rcat.stats.climateindex.**hotdays_calc**(*data*, *thr_p75*)

Calculate number of hotdays.

Return days with mean temperature above the 75th percentile of climatology.

Parameters

- **data** (*array*) – 1D-array of temperature input timeseries
- **thr_p75** (*float*) – 75th percentile daily mean value from climatology

rcat.stats.climateindex.extr_hotdays_calc

rcat.stats.climateindex.**extr_hotdays_calc**(*data*, *thr_p95*)

Calculate number of extreme hotdays.

Return days with mean temperature above the 95th percentile of climatology.

Parameters

- **data** (*array*) – 1D-array of temperature input timeseries
- **thr_p95** (*float*) – 95th percentile daily mean value from climatology

rcat.stats.climateindex.tropnights_calc

rcat.stats.climateindex.**tropnights_calc**(*data*)

Calculate number of tropical nights.

Return days with minimum temperature not below 20 degrees C.

Parameters **data** (*array*) – 1D-array of minimum temperatures timeseries in degrees Kelvin

rcat.stats.climateindex.ehi

rcat.stats.climateindex.**ehi**(*data*, *thr_95*, *axis=0*, *keepdims=False*)

Calculate Excessive Heat Index (EHI).

Parameters

- **data** (*list/array*) – 1D/2D array of daily temperature timeseries
- **thr_95** (*float*) – 95th percentile daily mean value from climatology
- **axis** (*int*) – The axis along which the calculation is applied (default 0).
- **keepdims** (*boolean*) – If data is 2d (time in third dimesion) and keepdims is set to True, calculation is applied to the zeroth axis (time) and returns a 2d array of calculated statistics. If set to False (default) all values are collectively assembled before calculation.

Returns **EHI** – Excessive heat index

Return type float

rcat.stats.climateindex.Rxx

rcat.stats.climateindex.**Rxx**(*data*, *thr=1.0*, *axis=0*, *normalize=False*, *keepdims=False*)

Rxx mm, count of any time units (days, hours, etc) when precipitation xx mm: Let RRij be the precipitation amount on time unit i in period j. Count the number of days where RRij xx mm.

Parameters

- **data** (*array*) – 1D/2D data array, with time steps on the zeroth axis (axis=0).
- **thr** (*float/int*) – Threshold to be used; eg 10 for R10, 20 R20 etc. Default 1.0.
- **axis** (*int*) – Along which axis to calculate Rxx. Defaults to 0
- **normalize** (*boolean*) – If True (default False) the counts are normalized by total number of time units in each array/grid point. Returned values will then be fractions.
- **keepdims** (*boolean*) – If False (default) calculation is performed on all data collectively, otherwise for each timeseries on each point in 2d space. 'Axis' then defines along which axis the timeseries are located.

Returns **Rxx** – 1D/2D array with calculated Rxx indices.

Return type list/array

rcat.stats.climateindex.RRpX

rcat.stats.climateindex.**RRpX**(*data*, *percentile*, *thr=None*, *axis=0*, *keepdims=False*)

RRpX mm, total amount of precipitation above the percentile threshold pX; RR pX mm: Let RRij be the daily precipitation amount on day i in period j. Sum the precipitation for all days where RRij pX mm.

Parameters

- **data** (*array*) – 1D/2D data array, with time steps on the zeroth axis (axis=0).
- **percentile** (*int*) – Percentile that defines the threshold.
- **thr** (*float/int*) – Pre-thresholding of data to do calculation for wet days/hours only.

- **keepdims** (*boolean*) – If False (default) calculation is performed on all data collectively, otherwise for each timeseries on each point in 2d space. 'Axis' then defines along which axis the timeseries are located.

Returns **RRpx** – 1D/2D array with calculated RRpXX indices.

Return type list/array

rcat.stats.climateindex.RRtX

rcat.stats.climateindex.**RRtX**(*data*, *thr*, *axis=0*, *keepdims=False*)

RRtX mm, total amount of precipitation above the threshold 'thr'.

Parameters

- **data** (*array*) – 1D/2D data array, with time steps on the zeroth axis (axis=0).
- **thr** (*int*) – Threshold that defines the threshold above which data is summed.
- **keepdims** (*boolean*) – If False (default) calculation is performed on all data collectively, otherwise for each timeseries on each point in 2d space. 'Axis' then defines along which axis the timeseries are located.

Returns **RRtx** – 1D/2D array with calculated RRtXX indices.

Return type list/array

rcat.stats.climateindex.SDII

rcat.stats.climateindex.**SDII**(*data*, *thr=1.0*, *axis=0*, *keepdims=False*)

SDII, Simple pricipitation intensity index: Let RRwj be the daily precipitation amount on wet days, w (RR 1mm) in period j.

Parameters

- **data** (*list/array*) – 2D array.
- **thr** (*float/int*) – threshold for wet events (wet days/hours etc)
- **axis** (*int*) – The axis along which the calculation is applied (default 0).
- **keepdims** (*boolean*) – If data is 2d (time in third dimesion) and keepdims is set to True, calculation is applied to the zeroth axis (time) and returns a 2d array of freq-int dists. If set to False (default) all values are collectively assembled before calculation.

Convolution

This module includes functions to perform convolution, for example image smoothing, using scipy's convolution routines.

rcat.stats.convolve.kernel_gen(n[, ktype, kfun])	Function to create a kernel, i.e.
rcat.stats.convolve.convolve2Dfunc	
rcat.stats.convolve.fft_prep(array, kernel, ...)	Prepare data array and kernel for fft computation.

Continued on next page

Table 7 – continued from previous page

rcat.stats.convolve.convolve_fft(array, kernel)	Convolve an ndarray with an nd-kernel.

rcat.stats.convolve.kernel_gen

rcat.stats.convolve.**kernel_gen**(*n*, *ktype='square'*, *kfun='mean'*)

Function to create a kernel, i.e. a moving window (box or disk) with side/radius equal to 'n'.

Parameters

- **n** (*int*) – Side/radius of square/disk of smoothening window.
- **ktype** (*string*) – The type of box; 'square' (default) or 'disk'.
- **kfun** (*string*) – The function 'kfun' applied to each sub-sample within the moving window. Either 'mean' (default) or 'sum'.

rcat.stats.convolve.fft_prep

rcat.stats.convolve.**fft_prep**(*array*, *kernel*, *fill_value*, *boundary='fill'*, *psf_pad=False*, *fft_pad=True*)

Prepare data array and kernel for fft computation.

Parameters

- **boundary** (*{'fill', 'wrap'}, optional*) – A flag indicating how to handle boundaries:
  - 'fill': set values outside the array boundary to fill_value (default)
  - 'wrap': periodic boundary
- **fft_pad** (*bool, optional*) – Default on. Zero-pad image to the nearest 2^n
- **psf_pad** (*bool, optional*) – Default off. Zero-pad image to be at least the sum of the image sizes (in order to avoid edge-wrapping when smoothing)

rcat.stats.convolve.convolve_fft

rcat.stats.convolve.**convolve_fft**(*array*, *kernel*, *boundary='fill'*, *fill_value=0*, *crop=True*, *return_fft=False*, *fft_pad=True*, *psf_pad=False*, *interpolate_nan=False*, *quiet=False*, *ignore_edge_zeros=False*, *min_wt=0.0*, *normalize_kernel=False*, *allow_huge=True*, *fftn=<function fftn>*, *ifftn=<function ifftn>*)

Convolve an ndarray with an nd-kernel. Returns a convolved image with shape = array.shape. Assumes kernel is centered.

convolve_fft differs from *scipy.signal.fftconvolve* in a few ways:

- It can treat `NaN` values as zeros or interpolate over them.
- `inf` values are treated as `NaN`
- (optionally) It pads to the nearest 2^n size to improve FFT speed.
- Its only valid `mode` is 'same' (i.e. the same shape array is returned)

- It lets you use your own fft, e.g., *pyFFTW <http://pypi.python.org/pypi/pyFFTW>* or *pyFFTW3 <http://pypi.python.org/pypi/PyFFTW3/0.2.1>*, which can lead to performance improvements, depending on your system configuration. pyFFTW3 is threaded, and therefore may yield significant performance benefits on multi-core machines at the cost of greater memory requirements. Specify the `fftn` and `ifftn` keywords to override the default, which is *numpy.fft.fft* and *numpy.fft.ifft*.

  Parameters

  - **`array`** (*numpy.ndarray*) – Array to be convolved with `kernel`
  - **`kernel`** (*numpy.ndarray*) – Will be normalized if `normalize_kernel` is set. Assumed to be centered (i.e., shifts may result if your kernel is asymmetric)
  - **`boundary`** (*`{'fill', 'wrap'}`, `optional`*) – A flag indicating how to handle boundaries: * 'fill': set values outside the array boundary to fill_value (default) * 'wrap': periodic boundary
  - **`interpolate_nan`** (*`bool`, `optional`*) – The convolution will be re-weighted assuming `NaN` values are meant to be ignored, not treated as zero. If this is off, all `NaN` values will be treated as zero.
  - **`ignore_edge_zeros`** (*`bool`, `optional`*) – Ignore the zero-pad-created zeros. This will effectively decrease the kernel area on the edges but will not re-normalize the kernel. This parameter may result in 'edge-brightening' effects if you're using a normalized kernel
  - **`min_wt`** (*`float`, `optional`*) – If ignoring `NaN` / zeros, force all grid points with a weight less than this value to `NaN` (the weight of a grid point with *no* ignored neighbors is 1.0). If `min_wt` is zero, then all zero-weight points will be set to zero instead of `NaN` (which they would be otherwise, because 1/0 = nan). See the examples below
  - **`normalize_kernel`** (*`function or boolean`, `optional`*) – If specified, this is the function to divide kernel by to normalize it. e.g., `normalize_kernel=np.sum` means that kernel will be modified to be: `kernel = kernel / np.sum(kernel)`. If True, defaults to `normalize_kernel = np.sum`.

  Other Parameters

  - **fft_pad** (*bool, optional*) – Default on. Zero-pad image to the nearest 2^n
  - **psf_pad** (*bool, optional*) – Default off. Zero-pad image to be at least the sum of the image sizes (in order to avoid edge-wrapping when smoothing)
  - **crop** (*bool, optional*) – Default on. Return an image of the size of the largest input image. If the images are asymmetric in opposite directions, will return the largest image in both directions. For example, if an input image has shape [100,3] but a kernel with shape [6,6] is used, the output will be [100,6].
  - **return_fft** (*bool, optional*) – Return the fft(image)*fft(kernel) instead of the convolution (which is ifft(fft(image)*fft(kernel))). Useful for making PSDs.
  - **fftn, ifftn** (*functions, optional*) – The fft and inverse fft functions. Can be overridden to use your own ffts, e.g. an fftw3 wrapper or scipy's fftn, e.g. `fftn=scipy.fftpack.fftn`
  - **complex_dtype** (*np.complex, optional*) – Which complex dtype to use. *numpy* has a range of options, from 64 to 256.
  - **quiet** (*bool, optional*) – Silence warning message about NaN interpolation
  - **allow_huge** (*bool, optional*) – Allow huge arrays in the FFT? If False, will raise an exception if the array or kernel size is >1 GB

Raises ValueError: – If the array is bigger than 1 GB after padding, will raise this exception unless allow_huge is True

See also:

`convolve()` Convolve is a non-fft version of this code. It is more memory efficient and for small kernels can be faster.

Returns **default** – **array** convolved with `kernel`. If `return_fft` is set, returns fft(**array**) * fft(`kernel`). If crop is not set, returns the image, but with the fft-padded size instead of the input size

Return type ndarray

Probability distributions

rcat.stats.pdf.freq_int_dist(data[, ...])	Calculate frequency - instensity distriutions.
rcat.stats.pdf.prob_of_exceed(data[, ...])	Calculates probability of exceedance distriutions.
rcat.stats.pdf.perkins_skill_score(p_mod, p_obs)	Calculate the Perkins Skill Score (PSS).

rcat.stats.pdf.freq_int_dist

rcat.stats.pdf.**freq_int_dist**(*data*, *keepdims=False*, *axis=0*, *bins=10*, *thr=None*, *density=True*, *ci=False*, *bootstrap=False*, *nmc=500*, *block=1*, *ci_level=95*, *nproc=1*)

Calculate frequency - instensity distriutions.

Parameters

- **`data`** (*`array`*) – 2D or 1D array of data. All data points are collectively used in the freq-instensity calculation unless 'keepdims' is True. Then calculation is performed along the dimension defined by axis argument (default 0).
- **`keepdims`** (*`boolean`*) – If data is 2d (time in third dimesion) and keepdims is set to True, calculation is applied to the dimension defined by axis argument (default 0) and returns a 2d array of freq-int dists. If set to False (default) all values are collectively assembled before calculation.
- **`axis`** (*`int`*) – The axis over which to apply the calculation if keepdims is set to True. Default is 0.
- **`bins`** (*`int/list/array`*) – If an int, it defines the number of equal-width bins in the given range (10, by default). If a sequence, it defines the bin edges, including the rightmost edge, allowing for non-uniform bin widths. If bins is set to 'None' they will be automatically calculated.
- **`thr`** (*`float`*) – Value of threshold if thresholding data. Default None.
- **`density`** (*`boolean`*) – If True (default) then the value of the probability density function at each bin is returned, otherwise the number of samples per bin.
- **`bootstrap`** (*`boolean`*) – If to use block bootstrap to calculate confidence interval.
- **`nmc`** (*`int/float`*) – Number of bootstrap samples to use.
- **`block`** (*`int/float`*) – Size of block to use in block bootstrap

- **ci_level** (*int/float*) – The confidence interval level to use (eg 95, 99 representing 95%, 99% levels)
- **nproc** (*int*) – Number of processes to use in bootstrap calculation. Default 1.

Returns

- **pdf** (*array*) – data array with size len(bins)-1 with counts/probabilities
- **ci** (*dict*) – data dictionary with confidence level for each bin; keys 'min_levels'/'max_levels' with corresponding values. If bootstrap is False, then None values are returned.

rcat.stats.pdf.prob_of_exceed

rcat.stats.pdf.**prob_of_exceed**(*data*, *keepdims=False*, *axis=0*, *thr=None*, *pctls_levels=None*)

Calculates probability of exceedance distriutions.

Parameters

- **data** (*array*) – 2D or 1D array of data. All data points are collectively used in the freq-instensity calculation unless 'keepdims' is True. Then calculation is performed along zeroth axis.
- **pctls_levels** (*'default', None or array/list*) – If set to 'default', probability levels of exceedance are defined by a set of percentiles ranging from 0-100 and calculated from input data. If an array or list, these levels (between 0 and 100) will be used instead. Default is None in which case input data is merely ranked from 0 to 1.
- **keepdims** (*boolean*) – If data is 2d (time in third dimesion) and keepdims is set to True, calculation is applied to the zeroth axis (time) and returns a 2d array of freq-int dists. If set to False (default) all values are collectively assembled before calculation.
- **axis** (*int*) – The axis over which to apply the calculation if keepdims is set to True. Default is 0.
- **thr** (*float*) – Value of threshold if thresholding data. Default None.

Returns **eop** – exceedance of probability array.

Return type array

rcat.stats.pdf.perkins_skill_score

rcat.stats.pdf.**perkins_skill_score**(*p_mod*, *p_obs*, *axis=0*)

Calculate the Perkins Skill Score (PSS).

Parameters

- **p_obs** (*p_mod,*) – 1d or 2d arrays with frequency of values (probability) in a given bin from the model and observations respectively. Make sure that the sum of probabilities over all the bins should be equal to one. This depends on how the pdf was calculated. Bins with unity width gives total prob of one.
- **axis** (*int*) – If data is 2d, the PSS will be calculated along this axis. Default axis is zero.

Returns **pss** – Returns Perkins Skill Score, single float or array with floats.

Return type float/array

SAL module

Routines for calculation of SAL statistics.

rcat.stats.sal.A_stat(data, refdata)	Calculate the amplitude component (A).
rcat.stats.sal.S_stat(data, data_label, ...)	Function to calculate the structure component (S).
rcat.stats.sal.L_stat(data, data_label, ...)	Function to determine the location component (L).
rcat.stats.sal.threshold(data, thr_type, value)	Function to calculate the threshold to be used to identify objects.
rcat.stats.sal.distfunc(x)	Calculate distances
rcat.stats.sal.remove_large_objects(...)	Remove large objects based on the maximum size limit defined by 'max_size'.
rcat.stats.sal.sal_calc(tstep, data, ...[, ...])	Perform the SAL calculation using the S, A, L functions.
rcat.stats.sal.write_to_disk(ddict, nt, ...)	
rcat.stats.sal.run_sal_analysis(data, ...[, ...])	Run the SAL analysis on the two data sets 'data' and 'refdata', where the latter is supposed to represent the 'truth'.

rcat.stats.sal.A_stat

rcat.stats.sal.**A_stat**(*data*, *refdata*)

Calculate the amplitude component (A).

Parameters **refdata** (*data,*) – 2D data arrays to be compared, where refdata is the reference data e.g. observations.

Returns **A** – The calculated amplitude component

Return type float

rcat.stats.sal.S_stat

rcat.stats.sal.**S_stat**(*data*, *data_label*, *refdata*, *refdata_label*, *obj_prop=True*, *lsmask=None*)

Function to calculate the structure component (S). The basic idea is to compare the volume of the normalized precipitation objects. This property captures information of the geometrical characteristics (size and shape) and how they differ between model (M) and reference (O).

Parameters

- **refdata** (*data,*) – 2D data arrays to be compared, where refdata is the reference data e.g. observations.
- **refdata_label** (*data_label,*) – Arrays with labeled objects. Returned from the label() function.
- **obj_prop** (*boolean*) – If True, individual object (rain fall area) properties are calculated and returned.
- **lsmask** (*array*) – Land/sea mask (2d boolean array) to characterize identified objects as land, ocean or coastal objects.

Returns

- **S** (*float*) – The calculated structure component.

- **area_props/ref_area_props** (*dictionary*) – Dictionary containing properties of identified objects in data and refdata respectively.

rcat.stats.sal.L_stat

rcat.stats.sal.**L_stat**(*data*, *data_label*, *refdata*, *refdata_label*)

Function to determine the location component (L). It consists of two components, L1 and L2. L1: measures the normalized distance between the centers of mass of the modelled and observed fields. L2: The second considers the averaged distance between the center of mass of the total field and individual field objects.

Parameters

- **refdata** (*data,*) – 2D data arrays to be compared, where refdata is the reference data e.g. observations.
- **refdata_label** (*data_label,*) – Arrays with labeled objects. Returned from the label() function.

Returns **L1, L2, L** – Dictionary with the calculated location components L1 and L2 as well as its composite L (L1 + L2).

Return type dictionary

rcat.stats.sal.threshold

rcat.stats.sal.**threshold**(*data*, *thr_type*, *value*)

Function to calculate the threshold to be used to identify objects.

Parameters

- **data** (*array*) – 2D data array.
- **thr_type** (*string*) – Type of threshold. Can be either "S" for specified (any absolute value), "F" for a fraction (between 0 and 1) of the maximum value, and "P" for a percentile (95 for the 95th percentile etc).
- **value** (*int/float*) – The corresponding value based on the chosen threhold type.

rcat.stats.sal.distfunc

rcat.stats.sal.**distfunc**(*x*)

Calculate distances

rcat.stats.sal.remove_large_objects

rcat.stats.sal.**remove_large_objects**(*segments*, *max_size*)

Remove large objects based on the maximum size limit defined by 'max_size'.

Parameters

- **segments** (*array*) – Array with labeled objects. Returned from the label() function.
- **max_size** (*int*) – Maximum size (number of grid points)

Returns **out** – The segments array with too large objects removed.

Return type array

rcat.stats.sal.sal_calc

rcat.stats.sal.**sal_calc**(*tstep*, *data*, *refdata*, *thr_t*, *thr_v*, *obj_prop=True*, *olsl=None*, *ousl=None*, *smlvl=None*, *land_sea_mask=None*)

Perform the SAL calculation using the S, A, L functions.

rcat.stats.sal.write_to_disk

rcat.stats.sal.**write_to_disk**(*ddict*, *nt*, *fname*, *attrs*)

rcat.stats.sal.run_sal_analysis

rcat.stats.sal.**run_sal_analysis**(*data*, *refdata*, *thr_type*, *thr_val*, *obj_prop=True*, *obj_lower_size_limit=None*, *obj_upper_size_limit=None*, *smoothening_data_level=None*, *land_sea_mask=None*, *write_to_file=False*, *filename=None*, *nproc=1*)

Run the SAL analysis on the two data sets 'data' and 'refdata', where the latter is supposed to represent the 'truth'.

Parameters

- **data/refdata** (*arrays*) – 2D data arrays with zeroth dimension representing time steps. Both data sets must have the same dimension sizes, i.e. both in time and space.
- **thr_type** (*string*) – Type of threshold to use. See 'threshold' function for more information.
- **thr_val** (*float/int*) – Value of threshold.
- **obj_prop** (*boolean*) – If True (default), a number of object area properties are returned for each of the identified objects. See 'S_stat' function for more information.
- **obj_lower_size_limit** (*int*) – If set, all objects with an area (number of connected grid points) lower than the value set is removed from analysis.
- **obj_upper_size_limit** (*int*) – If set, all objects with an area (number of connected grid points) greater than the value set is removed from analysis.
- **smoothening_data_level** (*int*) – If set, the number represents the # of grid points of the side of a moving window used to smooth the data arrays. Mean value within window is calculated.
- **land_sea_mask** (*array/None*) – If set, land_sea_mask must be a 2d boolean array with same dimension as input data. The land/sea-mask is then used to identify objects as either land (1), ocean (0) or coastal (2) in the object properties dictionary. Thus, mask only used if obj_prop=True. N.B. Mask must have True for ocean points and False for land points.
- **write_to_file** (*boolean*) – Whether to write results to disk.
- **filename** (*str*) – Name of file for writing to disk.
- **nproc** (*int*) – Number of processors to use in calculation. If larger than 1 (default), the multiprocessing module is used to distribute the calculation in the time dimension.

Returns

- **out_dict** (*dictionary*) – Dictionary with calculated SAL statistics and area properties (if obj_prop is set to True).

- **nc** (*file*) – If 'write_to_file' is True, results are written to disk in a netcdf file.

4.1.3 Utilities

Atmospheric physics

Routines for calculations of various physical properties

rcat.utils.atmosphys.rh2sh(rh, T[, P])	Convert relative humidity to specific humidity Code from: https://github.com/PecanProject/pecan/blob/master/modules/data.atmosphere/R/metutils.R Equation for sh from standard literature, e.g.
rcat.utils.atmosphys.td2sh(Td, P)	Convert dew point temperature to specific humidity https://github.com/PecanProject/pecan/blob/master/modules/data.atmosphere/R/metutils.R
rcat.utils.atmosphys.sh2td(sh, p)	Returns dew point temperature (K) at mixing ratio sh (kg/kg) and pressure p (Pa).
rcat.utils.atmosphys.es	
rcat.utils.atmosphys.e	
rcat.utils.atmosphys.td(e)	Returns dew point temperature (C) at vapor pressure e (Pa) Insert Td in 2.17 in Rogers&Yau and solve for Td
rcat.utils.atmosphys.wind2uv(ws, wd[, dir_unit])	Converts wind speed and direction to u and v components.
rcat.utils.atmosphys.uv2wind(u, v)	Converts u and v components to wind speed and direction.
rcat.utils.atmosphys.calc_vaisala(ddict, ...)	Calculate Brunt-Vaisala frequency in layer bounded by two pressure levels.

rcat.utils.atmosphys.rh2sh

rcat.utils.atmosphys.**rh2sh**(*rh*, *T*, *P=1013.25*)

Convert relative humidity to specific humidity Code from: https://github.com/PecanProject/pecan/blob/master/modules/data.atmosphere/R/metutils.R Equation for sh from standard literature, e.g. K. Emanuel (1994; eq. 4.1.4) Reference: * Emanuel, K. A. (1994): Atmospheric Convection. New York, NY:

Oxford University Press, 580 pp.

Parameters

- **float/array of floats** (*P,*) – Relative humidity in proportion, not percent
- **float/array of floats** – Absolute temperature in Kelvin
- **float/array of floats** – Pressure in hPa (mb)

Returns Specific humidity in kg/kg

Return type sh,

rcat.utils.atmosphys.td2sh

rcat.utils.atmosphys.**td2sh**(*Td*, *P*)

Convert dew point temperature to specific humidity https://github.com/PecanProject/pecan/blob/master/

modules/data.atmosphere/R/metutils.R

Parameters

- **float/array of floats** (*P,*) – Absolute dew point temperature in Kelvin
- **float/array of floats** – Pressure in mb

Returns Specific humidity in g/kg

Return type p, float/array of floats

rcat.utils.atmosphys.sh2td

rcat.utils.atmosphys.**sh2td**(*sh*, *p*)

Returns dew point temperature (K) at mixing ratio sh (kg/kg) and pressure p (Pa). Insert Td in 2.17 in Rogers&Yau and solve for Td

rcat.utils.atmosphys.td

rcat.utils.atmosphys.**td**(*e*)

Returns dew point temperature (C) at vapor pressure e (Pa) Insert Td in 2.17 in Rogers&Yau and solve for Td

rcat.utils.atmosphys.wind2uv

rcat.utils.atmosphys.**wind2uv**(*ws*, *wd*, *dir_unit='rad'*)

Converts wind speed and direction to u and v components.

Parameters

- **float/array of floats** (*wd,*) – Wind speed data
- **float/array of floats** – Wind direction data
- **string** (*dir_deg,*) – Units of the wind direction data; 'rad' (default) or 'deg'.

Returns u and v wind components

Return type (u, v), floats/arrays of floats

rcat.utils.atmosphys.uv2wind

rcat.utils.atmosphys.**uv2wind**(*u*, *v*)

Converts u and v components to wind speed and direction.

Parameters

- **float/array of floats** (*v,*) – east/west wind component
- **float/array of floats** – north/south wind component

Returns wind speed and wind direction respectively.

Return type (ws, wd), floats/arrays of floats

rcat.utils.atmosphys.calc_vaisala

rcat.utils.atmosphys.**calc_vaisala**(*ddict*, *model*, *lower_plevel*, *upper_plevel*)

Calculate Brunt-Vaisala frequency in layer bounded by two pressure levels. Pressure must be given in hPa, temperature (T) in Kelvin and specific humidity (q) in kg/kg.

Parameters

- **ddict** (*dictionary*) – Data dictionary
- **model** (*str*) – model data to use
- **lower_plevel** (*float*) – lower pressure surface
- **upper_plevel** (*float*) – upper pressure surface

Returns **N2** – The squared Brunt-Vaisala frequency

Return type array with floats

IO handling

This module provides routines representing tools to read and write NetCDF files.

rcat.utils.file_io.ncdump(nc_fid[, verb])	ncdump outputs dimensions, variables and their attribute information.
rcat.utils.file_io.openFile(filename)	Function to open netcdf file.
rcat.utils.file_io.getDimensions(nc[, close])	Function to retrieve the dimensions of a netcdf file nc: Netcdf object opened with function "openFile" close: set True if you want the file to be closed after retrieval.
rcat.utils.file_io.getParams(nc, params[, close])	Function to retrieve variables from a netcdf file nc: Netcdf object opened with function "openFile" params: A list of strings with the parameters to be retrieved close: set True if you want the file to be closed after retrieval.
rcat.utils.file_io.fracday2datetime(tdata)	Takes an array of dates given in %Y%m%d.%f format and returns a corresponding datetime object
rcat.utils.file_io.write2netcdf(filename, ...)	Opens a new NetCDF file to write the input data to.

rcat.utils.file_io.ncdump

rcat.utils.file_io.**ncdump**(*nc_fid*, *verb=True*)

ncdump outputs dimensions, variables and their attribute information. The information is similar to that of NCAR's ncdump utility. ncdump requires a valid instance of Dataset.

Parameters

- **nc_fid** (*netCDF4.Dataset*) – A netCDF4 dateset object
- **verb** (*Boolean*) – whether or not nc_attrs, nc_dims, and nc_vars are printed

Returns

- **nc_attrs** (*list*) – A Python list of the NetCDF file global attributes
- **nc_dims** (*list*) – A Python list of the NetCDF file dimensions

- **nc_vars** (*list*) – A Python list of the NetCDF file variables

rcat.utils.file_io.openFile

rcat.utils.file_io.**openFile**(*filename*)
Function to open netcdf file. filename: string with full path to file

rcat.utils.file_io.getDimensions

rcat.utils.file_io.**getDimensions**(*nc*, *close=False*)
Function to retrieve the dimensions of a netcdf file nc: Netcdf object opened with function "openFile" close: set True if you want the file to be closed after retrieval. Returns lons and lats, time as well as gridsize Nx,Ny

rcat.utils.file_io.getParams

rcat.utils.file_io.**getParams**(*nc*, *params*, *close=False*)
Function to retrieve variables from a netcdf file nc: Netcdf object opened with function "openFile" params: A list of strings with the parameters to be retrieved close: set True if you want the file to be closed after retrieval. Returns a list with the given parameters.

rcat.utils.file_io.fracday2datetime

rcat.utils.file_io.**fracday2datetime**(*tdata*)
Takes an array of dates given in %Y%m%d.%f format and returns a corresponding datetime object

rcat.utils.file_io.write2netcdf

rcat.utils.file_io.**write2netcdf**(*filename*, *filedir*, *dim*, *variables*, *global_attr=None*, *nc_format='NETCDF4'*, *compress=False*, *complevel=4*)
Opens a new NetCDF file to write the input data to. For nc_format, you can choose from 'NETCDF3_CLASSIC', 'NETCDF3_64BIT', 'NETCDF4_CLASSIC', and 'NETCDF4' (default)

Parameters

- **filename** (*str*) – name of netcdf file to write to
- **filedir** (*str*) – directory path to put the file
- **dim** (*dict*) – dimensions to be used
- **variables** (*dict*) – variables with their values and attributes
- **global_attr** (*dict*) – global attributes (optional)
- **nc_format** (*str*) – Specify netCDF format
- **compress** (*boolean*) – Whether to compress (using 'zlib=True' in the write call).
- **complevel** (*int*) – An integer between 1-9 representing the degree of compression to be used.

The dictionaries should be structured as described by the examples below:

dims_dict = {} dims_dict['x'] = 154 dims_dict['y'] = 192 dims_dict['nv'] = 4 dims_dict['time'] = None

vars_dict = {} vars_dict = {'lon': {'values': lons, 'dims': ('y', 'x'),

'attributes': {'long_name': 'longitude', 'standard_name': 'longitude', 'units': 'degrees_east', '_CoordinateAxisType': 'Lon'}},

'lat': {'values': lats, 'dims': ('y', 'x'),

'attributes': {'long_name': 'latitude', 'standard_name': 'latitude', 'units': 'degrees_north', '_CoordinateAxisType': 'Lat'}},

'pr': {'values': pr, 'dims': ('time', 'y', 'x'),

'attributes': {'long_name': 'precipitation',

'standard_name': 'precipitation flux', 'units': 'kg m-2 s-1', 'coordinates': 'lon lat', '_FillValue': -9999.}}}

glob_attr = {'description': 'some description of file', 'history': 'Created ' + time.ctime(time.time()), 'experiment': 'Fractions Skill Score analysis', 'contact': 'petter.lind@smhi.se', 'references': 'http://journals.ametsoc.org/doi/abs/ 10.1175/2007MWR2123.1'}

Grid applications

Routines to remap data given source and target grids.

rcat.utils.grids.fnCellCorners(rgrLon, rgrLat)	File name: fnCellBoundaries Author: Andreas Prein E-mail: prein@ucar.edu Date created: 20.03.2015 Date last modified: 20.03.2015
rcat.utils.grids.calc_vertices(lons, lats[, ...])	Estimate the cell boundaries from the cell location of regular grids
rcat.utils.grids.fnRemapConOperator(rgrLonS, ...)	File name: fnRemapConOperator Author: Andreas Prein E-mail: prein@ucar.edu Date created: 26.05.2017 Date last modified: 26.05.2017
rcat.utils.grids.fnRemapCon(rgrLonS, ...)	File name: fnRemapCon Author: Andreas Prein E-mail: prein@ucar.edu Date created: 13.06.2017 Date last modified: 13.06.2017
rcat.utils.grids.add_matrix_NaNs(regridder)	Replace zero values of cells in the new grid that are outside the old grid's domain with NaN's.

rcat.utils.grids.fnCellCorners

rcat.utils.grids.**fnCellCorners**(*rgrLon*, *rgrLat*)

File name: fnCellBoundaries Author: Andreas Prein E-mail: prein@ucar.edu Date created: 20.03.2015 Date last modified: 20.03.2015

Estimate the cell boundaries from the cell location of regular grids

returns: rgrLonBND & rgrLatBND –> arrays of dimension [nlon,nlat] containing the cell boundaries of each gridcell in rgrlon and rgrlat

rcat.utils.grids.calc_vertices

rcat.utils.grids.**calc_vertices**(*lons*, *lats*, *write_to_file=False*, *filename=None*)

Estimate the cell boundaries from the cell location of regular grids

Parameters

- **lats** (*lons,*) – Longitude and latitude values
- **write_to_file** (*bool*) – If True lat/lon information, including vertices, is written to file following the structure given by cdo commmand 'griddes'
- **filename** (*str*) – Name of text file for the grid information. Only used if write_to_file is True. If not provided, a default name will be used.

Returns **lon_bnds, lat_bnds** – Arrays of dimension [4, nlat, nlon] containing cell boundaries of each gridcell in lons and lats

Return type arrays

rcat.utils.grids.fnRemapConOperator

rcat.utils.grids.**fnRemapConOperator**(*rgrLonS*, *rgrLatS*, *rgrLonT*, *rgrLatT*, *rgrLonSBNDS=None*, *rgrLatSBNDS=None*, *rgrLonTBNDS=None*, *rgrLatTBNDS=None*)

File name: fnRemapConOperator Author: Andreas Prein E-mail: prein@ucar.edu Date created: 26.05.2017 Date last modified: 26.05.2017

Generates an opperator to coservatively remapp data from a source rectangular grid to an target rectangular grid.

Parameters

- **rgrLonS,rgrLatS** (*arrays*) – Source grid longitude and latitude values
- **rgrLonT,rgrLatT** (*arrays*) – Target grid longitude and latitude values
- **rgrLonSBNDS,rgrLatSBNDS** (*arrays*) – Source grid longitude and latitude grid point boundaries (corners). These must be given in the structure (lat, lon, vertices) where vertices are the four corners of each grid point. If not provided (default) then corners are calculated using fnCellCorners.
- **rgrLonTBNDS,rgrLatTBNDS** (*arrays*) – Target grid longitude and latitude grid point boundaries (corners). See above for more info.

Returns **grConRemapOp** – opperator that contains the grid cells and their wheights of the source grid for each target grid cell

Return type dictionary

rcat.utils.grids.fnRemapCon

rcat.utils.grids.**fnRemapCon**(*rgrLonS*, *rgrLatS*, *rgrLonT*, *rgrLatT*, *grOperator*, *rgrData*)

File name: fnRemapCon Author: Andreas Prein E-mail: prein@ucar.edu Date created: 13.06.2017 Date last modified: 13.06.2017

Uses the remap operator generated by the function fnRemapConOperator to remap data to a target grid conservatively

Parameters

- **rgrLonS,rgrLatS** (*arrays*) – Source grid longitude and latitude values
- **rgrLonT,rgrLatT** (*arrays*) – Target grid longitude and latitude values
- **grOperator** (*dictionary*) – Remapping operator returned from fnRemapConOperator.

- **rgrData** (*3D/4D array*) – Data to be regridded, structured as (time, lat, lon) or (time, variables, lat, lon).

Returns **rgrTarData** – Remapped data matrix

Return type array

rcat.utils.grids.add_matrix_NaNs

rcat.utils.grids.**add_matrix_NaNs**(*regridder*)

Replace zero values of cells in the new grid that are outside the old grid's domain with NaN's.

Parameters **regridder** (*Object from xESMF Regridder function*) –

Returns Modified regridder where zero valued cells (outside source grid) has been replaced with NaN's.

Return type regridder

Config reader module

Creates and return a dictionary built from a config file.

rcat.utils.ini_reader.get_config_dict(ini_file)	Create a dictionary from then input config file.

rcat.utils.ini_reader.get_config_dict

rcat.utils.ini_reader.**get_config_dict**(*ini_file*)

Create a dictionary from then input config file. PARAMETERS: config (.ini) file RETURNS : Dictionary

Polygons

Mask polygons

Routine for Masking Data with Polygons.

rcat.utils.polygons.polygons
rcat.utils.polygons.mask_region
rcat.utils.polygons.create_polygon
rcat.utils.polygons.plot_polygon
rcat.utils.polygons.topo_mask
rcat.utils.polygons.find_geo_indices

Draw polygon

Draw a simple polygon using matplotlib with mouse event handling.

rcat.utils.draw_polygon.Canvas.set_location(event)
rcat.utils.draw_polygon.Canvas.update_path(event)

rcat.utils.draw_polygon.Canvas.set_location

Canvas.**set_location**(*event*)

rcat.utils.draw_polygon.Canvas.update_path

Canvas.**update_path**(*event*)

4.2 Internal API

4.2.1 RCAT Plots

Module script for plotting

rcat.runtime.RCAT_plots.plot_main
rcat.runtime.RCAT_plots.get_clevs
rcat.runtime.RCAT_plots.map_season
rcat.runtime.RCAT_plots.map_ann_cycle
rcat.runtime.RCAT_plots.line_ann_cycle
rcat.runtime.RCAT_plots.map_pctls
rcat.runtime.RCAT_plots.map_diurnal_cycle
rcat.runtime.RCAT_plots.line_diurnal_cycle
rcat.runtime.RCAT_plots.pdf_plot
rcat.runtime.RCAT_plots.map_asop
rcat.runtime.RCAT_plots.line_asop

4.2.2 RCAT Statistics

Functions controlling statistical calculations

rcat.runtime.RCAT_stats.default_stats_config(stats)	The function returns a dictionary with default statistics configurations for a selection of statistics given by input stats.
rcat.runtime.RCAT_stats.mod_stats_config(...)	Get the configuration for the input statistics 'requested_stats'.
rcat.runtime.RCAT_stats.calc_statistics(...)	Calculate statistics 'stat' according to configuration in 'stat_config'.

rcat.runtime.RCAT_stats.default_stats_config

rcat.runtime.RCAT_stats.**default_stats_config**(*stats*)
The function returns a dictionary with default statistics configurations for a selection of statistics given by input stats.

rcat.runtime.RCAT_stats.mod_stats_config

rcat.runtime.RCAT_stats.**mod_stats_config**(*requested_stats*)
Get the configuration for the input statistics 'requested_stats'. The returned configuration is a dictionary.

rcat.runtime.RCAT_stats.calc_statistics

rcat.runtime.RCAT_stats.**calc_statistics**(*data*, *var*, *stat*, *stat_config*)
Calculate statistics 'stat' according to configuration in 'stat_config'. This function calls the respective stat function (defined in _stats).

Statistical functions

rcat.runtime.RCAT_stats.moments(data, var, ...)	Calculate standard moment statistics: avg, median, std, max/min
rcat.runtime.RCAT_stats.seasonal_cycle(data, ...)	Calculate seasonal cycle
rcat.runtime.RCAT_stats.annual_cycle(data, ...)	Calculate annual cycle
rcat.runtime.RCAT_stats.diurnal_cycle(data, ...)	Calculate diurnal cycle
rcat.runtime.RCAT_stats.dcycle_harmonic_fit(...)	Calculate diurnal cycle with Harmonic oscillation fit
rcat.runtime.RCAT_stats.percentile(data, ...)	Calculate percentiles
rcat.runtime.RCAT_stats.freq_int_dist(data, ...)	Calculate frequency intensity distributions
rcat.runtime.RCAT_stats.asop(data, var, ...)	Calculate ASoP components for precipitation
rcat.runtime.RCAT_stats.Rxx(data, var, stat, ...)	Count of any time units (days, hours, etc) when precipitation xx mm.

rcat.runtime.RCAT_stats.moments

rcat.runtime.RCAT_stats.**moments**(*data*, *var*, *stat*, *stat_config*)
Calculate standard moment statistics: avg, median, std, max/min

rcat.runtime.RCAT_stats.seasonal_cycle

rcat.runtime.RCAT_stats.**seasonal_cycle**(*data*, *var*, *stat*, *stat_config*)
Calculate seasonal cycle

rcat.runtime.RCAT_stats.annual_cycle

rcat.runtime.RCAT_stats.**annual_cycle**(*data*, *var*, *stat*, *stat_config*)
Calculate annual cycle

rcat.runtime.RCAT_stats.diurnal_cycle

rcat.runtime.RCAT_stats.**diurnal_cycle**(*data*, *var*, *stat*, *stat_config*)
Calculate diurnal cycle

rcat.runtime.RCAT_stats.dcycle_harmonic_fit

rcat.runtime.RCAT_stats.**dcycle_harmonic_fit**(*data*, *var*, *stat*, *stat_config*)
Calculate diurnal cycle with Harmonic oscillation fit

rcat.runtime.RCAT_stats.percentile

rcat.runtime.RCAT_stats.**percentile**(*data*, *var*, *stat*, *stat_config*)
Calculate percentiles

rcat.runtime.RCAT_stats.freq_int_dist

rcat.runtime.RCAT_stats.**freq_int_dist**(*data*, *var*, *stat*, *stat_config*)
Calculate frequency intensity distributions

rcat.runtime.RCAT_stats.asop

rcat.runtime.RCAT_stats.**asop**(*data*, *var*, *stat*, *stat_config*)
Calculate ASoP components for precipitation

rcat.runtime.RCAT_stats.Rxx

rcat.runtime.RCAT_stats.**Rxx**(*data*, *var*, *stat*, *stat_config*)
Count of any time units (days, hours, etc) when precipitation xx mm.

CHAPTER 5

Release notes

v1.0.0 - First online collaborative version

Index

N

O

P

R

S

T

U

W