clisops Documentation

Release 0.8.0

Elle Smith

Feb 02, 2022

CONTENTS:

CHAPTER

ONE

QUICK GUIDE

1.1 clisops - climate simulation operations

The `clisops` package (pronounced "clie-sops") provides a python library for running *data-reduction* operations on Xarray data sets or files that can be interpreted by Xarray. These basic operations (subsetting, averaging and regridding) are likely to work where data structures are NetCDF-centric, such as those found in ESGF data sets.

`clisops` is employed by the daops library to perform its basic operations once `daops` has applied any necessary *fixes* to data in order to remove irregularities/anomalies. Users are recommended to investigate using `daops` directly in order to access these *fixes* which may affect the scientific credibility of the results.

`clisops` can be used stand-alone to read individual, or groups of, NetCDF files directly.

- Free software: BSD
- Documentation: https://clisops.readthedocs.io.

1.1.1 Features

The package provides the following operations:

- subset
- average
- regrid

1.1.2 Online Demo

You can try clisops online using Binder (just click on the binder link below), or view the notebooks on NBViewer.

1.2 Credits

This package was created with `Cookiecutter` and the `audreyr/cookiecutter-pypackage` project template.

- Cookiecutter: https://github.com/audreyr/cookiecutter
- cookiecutter-pypackage: https://github.com/audreyr/cookiecutter-pypackage

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install clisops, run this command in your terminal:

```
$ pip install clisops
```

This is the preferred method to install clisops, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

Warning: Some average operations (*clisops.core.average_shape*) require at least version 0.5.2 of the *xESMF* package. Unfortunately, this package is not available on pypi at the time these lines were written and it also depends on packages (ESMF, ESMpy) unavailable on windows. It can still be installed on osx/linux through *conda* or directly [from source](https://github.com/pangeo-data/xESMF/).

Another optional dependency is [pygeos](https://pygeos.readthedocs.io/en/latest/index.html). If installed, the performance of *core.subset.create_mask* and *core.subset.subset_shape* are greatly improved.

2.2 From sources

The sources for clisops can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/roocs/clisops
```

Create Conda environment named *clisops*.

Warning: For windows users, you might need to edit the `environment.yml` file and remove the *xESMF* package.

```
$ conda env create -f environment.yml
$ source activate clisops
```

Install clisops in development mode:

```
$ pip install -r requirements.txt
$ pip install -r requirements_dev.txt
$ python setup.py develop
```

Run tests:

```
$ pytest -v tests/
```

CHAPTER

THREE

USAGE

To use clisops in a project:

```
import clisops
```

For information on the configuration options available in clisops, see: https://roocs-utils.readthedocs.io/en/latest/configuration.html#clisops

CHAPTER

FOUR

API

4.1 Core subset functionality

Subset module.

clisops.core.subset.**create_mask**(*, *x_dim: xarray.core.dataarray.DataArray*, *y_dim: xarray.core.dataarray.DataArray*, *poly: geopandas.geodataframe.GeoDataFrame*, *wrap_lons: bool = False*, *check_overlap: bool = False*)

Create a mask with values corresponding to the features in a GeoDataFrame using vectorize methods.

The returned mask's points have the value of the first geometry of *poly* they fall in.

Parameters

- **x_dim** (*xarray.DataArray*) – X or longitudinal dimension of xarray object. Can also be given through *ds_in*.
- **y_dim** (*xarray.DataArray*) – Y or latitudinal dimension of xarray object. Can also be given through *ds_in*.
- **poly** (*gpd.GeoDataFrame*) – GeoDataFrame used to create the xarray.DataArray mask. If its index doesn't have a integer dtype, it will be reset to integers, which will be used in the mask.
- **wrap_lons** (*bool*) – Shift vector longitudes by -180,180 degrees to 0,360 degrees; Default = False
- **check_overlap** (*bool*) – Perform a check to verify if shapes contain overlapping geometries.

Returns *xarray.DataArray*

Examples

```
>>> import geopandas as gpd
>>> import xarray as xr
>>> from clisops.core.subset import create_mask
>>> ds = xr.open_dataset(path_to_tasmin_file)
>>> polys = gpd.read_file(path_to_multi_shape_file)
...
# Get a mask from all polygons in the shape file
>>> mask = create_mask(x_dim=ds.lon, y_dim=ds.lat, poly=polys)
>>> ds = ds.assign_coords(regions=mask)
...
```

(continues on next page)

(continued from previous page)

```
# Operations can be applied to each regions with  `groupby`. Ex:
>>> ds = ds.groupby('regions').mean()
...
# Extra step to retrieve the names of those polygons stored in another column (here
↪"id")
>>> region_names = xr.DataArray(polys.id, dims=('regions',))
>>> ds = ds.assign_coords(regions_names=region_names)
```

clisops.core.subset.**distance**(*da: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, ***, *lon: Union[float, Sequence[float], xarray.core.dataarray.DataArray]*, *lat: Union[float, Sequence[float], xarray.core.dataarray.DataArray]*)

Return distance to a point in meters.

Parameters

- **da** (*Union[xarray.DataArray, xarray.Dataset]*) – Input data.
- **lon** (*Union[float, Sequence[float], xarray.DataArray]*) – Longitude coordinate.
- **lat** (*Union[float, Sequence[float], xarray.DataArray]*) – Latitude coordinate.

Returns *xarray.DataArray* – Distance in meters to point.

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import distance
...
To get the indices from closest point, use:
>>> da = xr.open_dataset(path_to_pr_file).pr
>>> d = distance(da, lon=-75, lat=45)
>>> k = d.argmin()
>>> i, j, _ = np.unravel_index(k, d.shape)
```

clisops.core.subset.**subset_bbox**(*da: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *lon_bnds: Union[numpy.array, Tuple[Optional[float], Optional[float]]] = None*, *lat_bnds: Union[numpy.array, Tuple[Optional[float], Optional[float]]] = None*, *start_date: Optional[str] = None*, *end_date: Optional[str] = None*, *first_level: Optional[Union[int, float]] = None*, *last_level: Optional[Union[int, float]] = None*, *time_values: Optional[Sequence[str]] = None*, *level_values: Optional[Union[Sequence[float], Sequence[int]]] = None*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Subset a DataArray or Dataset spatially (and temporally) using a lat lon bounding box and date selection.

Return a subset of a DataArray or Dataset for grid points falling within a spatial bounding box defined by longitude and latitudinal bounds and for dates falling within provided bounds.

TODO: returns the what? In the case of a lat-lon rectilinear grid, this simply returns the

Parameters

- **da** (*Union[xarray.DataArray, xarray.Dataset]*) – Input data.
- **lon_bnds** (*Union[np.array, Tuple[Optional[float], Optional[float]]]*) – List of minimum and maximum longitudinal bounds. Optional. Defaults to all longitudes in original data-array.

- **lat_bnds** (*Union[np.array, Tuple[Optional[float], Optional[float]]]*) – List of minimum and maximum latitudinal bounds. Optional. Defaults to all latitudes in original data-array.
- **start_date** (*Optional[str]*) – Start date of the subset. Date string format – can be year ("%Y"), year-month ("%Y-%m") or year-month-day("%Y-%m-%d"). Defaults to first day of input data-array.
- **end_date** (*Optional[str]*) – End date of the subset. Date string format – can be year ("%Y"), year-month ("%Y-%m") or year-month-day("%Y-%m-%d"). Defaults to last day of input data-array.
- **first_level** (*Optional[Union[int, float]]*) – First level of the subset. Can be either an integer or float. Defaults to first level of input data-array.
- **last_level** (*Optional[Union[int, float]]*) – Last level of the subset. Can be either an integer or float. Defaults to last level of input data-array.
- **time_values** (*Optional[Sequence[str]]*) – A list of datetime strings to subset.
- **level_values** (*Optional[Union[Sequence[float], Sequence[int]]]*) – A list of level values to select.

Returns *Union[xarray.DataArray, xarray.Dataset]* – Subsetted xarray.DataArray or xarray.Dataset

---

Note: subset_bbox expects the lower and upper bounds to be provided in ascending order. If the actual coordinate values are descending then this will be detected and your selection reversed before the data subset is returned.

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import subset_bbox
>>> ds = xr.open_dataset(path_to_pr_file)
...
# Subset lat lon
>>> prSub = subset_bbox(ds.pr, lon_bnds=[-75, -70], lat_bnds=[40, 45])
```

clisops.core.subset.**subset_gridpoint**(*da: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *lon: Optional[Union[float, Sequence[float], xarray.core.dataarray.DataArray]] = None*, *lat: Optional[Union[float, Sequence[float], xarray.core.dataarray.DataArray]] = None*, *start_date: Optional[str] = None*, *end_date: Optional[str] = None*, *first_level: Optional[Union[int, float]] = None*, *last_level: Optional[Union[int, float]] = None*, *tolerance: Optional[float] = None*, *add_distance: bool = False*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Extract one or more nearest gridpoint(s) from datarray based on lat lon coordinate(s).

Return a subsetted data array (or Dataset) for the grid point(s) falling nearest the input longitude and latitude coordinates. Optionally subset the data array for years falling within provided date bounds. Time series can optionally be subsetted by dates. If 1D sequences of coordinates are given, the gridpoints will be concatenated along the new dimension "site".

Parameters

- **da** (*Union[xarray.DataArray, xarray.Dataset]*) – Input data.
- **lon** (*Optional[Union[float, Sequence[float], xarray.DataArray]]*) – Longitude coordinate(s). Must be of the same length as lat.
- **lat** (*Optional[Union[float, Sequence[float], xarray.DataArray]]*) – Latitude coordinate(s). Must be of the same length as lon.
- **start_date** (*Optional[str]*) – Start date of the subset. Date string format – can be year ("%Y"), year-month ("%Y-%m") or year-month-day("%Y-%m-%d"). Defaults to first day of input data-array.
- **end_date** (*Optional[str]*) – End date of the subset. Date string format – can be year ("%Y"), year-month ("%Y-%m") or year-month-day("%Y-%m-%d"). Defaults to last day of input data-array.
- **first_level** (*Optional[Union[int, float]]*) – First level of the subset. Can be either an integer or float. Defaults to first level of input data-array.
- **last_level** (*Optional[Union[int, float]]*) – Last level of the subset. Can be either an integer or float. Defaults to last level of input data-array.
- **tolerance** (*Optional[float]*) – Masks values if the distance to the nearest gridpoint is larger than tolerance in meters.
- **add_distance** (*bool*)

Returns *Union[xarray.DataArray, xarray.Dataset]* – Subsetted xarray.DataArray or xarray.Dataset

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import subset_gridpoint
>>> ds = xr.open_dataset(path_to_pr_file)
...
# Subset lat lon point
>>> prSub = subset_gridpoint(ds.pr, lon=-75,lat=45)
...
# Subset multiple variables in a single dataset
>>> ds = xr.open_mfdataset([path_to_tasmax_file, path_to_tasmin_file])
>>> dsSub = subset_gridpoint(ds, lon=-75, lat=45)
```

clisops.core.subset.**subset_level**(*da: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *first_level: Optional[Union[int, float]] = None*, *last_level: Optional[Union[int, float]] = None*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Subset input DataArray or Dataset based on first and last levels. Return a subset of a DataArray or Dataset for levels falling within the provided bounds.

Parameters

- **da** (*Union[xarray.DataArray, xarray.Dataset]*) – Input data.
- **first_level** (*Optional[Union[int, float]]*) – First level of the subset (specified as the value, not the index). Can be either an integer or float. Defaults to first level of input data-array.
- **last_level** (*Optional[Union[int, float]]*) – Last level of the subset (specified as the value, not the index). Can be either an integer or float. Defaults to last level of input data-array.

Returns *Union[xarray.DataArray, xarray.Dataset]* – Subsetted xarray.DataArray or xarray.Dataset

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import subset_level
>>> ds = xr.open_dataset(path_to_pr_file)
...
# Subset complete levels
>>> prSub = subset_level(ds.pr,first_level=0,last_level=30)
...
# Subset single level
>>> prSub = subset_level(ds.pr,first_level=1000,last_level=1000)
...
# Subset multiple variables in a single dataset
>>> ds = xr.open_mfdataset([path_to_tasmax_file, path_to_tasmin_file])
>>> dsSub = subset_time(ds,first_level=1000.0,last_level=850.0)
```

Notes

TBA

clisops.core.subset.**subset_level_by_values**(*da: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *level_values: Optional[Union[Sequence[float], Sequence[int]]] = None*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Subset input DataArray or Dataset based on a sequence of vertical level values. Return a subset of a DataArray or Dataset for levels matching those requested.

Parameters

- **da** (*Union[xarray.DataArray, xarray.Dataset]*) – Input data.
- **level_values** (*Optional[Union[Sequence[float], Sequence[int]]]*) – A list of level values to select.

Returns *Union[xarray.DataArray, xarray.Dataset]* – Subsetted xarray.DataArray or xarray.Dataset

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import subset_level_by_values
>>> ds = xr.open_dataset(path_to_pr_file)
...
# Subset a selection of levels
>>> levels = [1000., 850., 250., 100.]
>>> prSub = subset_level_by_values(ds.pr, level_values=levels)
```

Notes

If any levels are not found, a ValueError will be raised. The requested levels will automatically be re-ordered to match the order in the input dataset.

clisops.core.subset.**subset_shape**(*ds: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *shape: Union[str, pathlib.Path, geopandas.geodataframe.GeoDataFrame]*, *raster_crs: Optional[Union[str, int]] = None*, *shape_crs: Optional[Union[str, int]] = None*, *buffer: Optional[Union[int, float]] = None*, *start_date: Optional[str] = None*, *end_date: Optional[str] = None*, *first_level: Optional[Union[int, float]] = None*, *last_level: Optional[Union[int, float]] = None*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Subset a DataArray or Dataset spatially (and temporally) using a vector shape and date selection.

Return a subset of a DataArray or Dataset for grid points falling within the area of a Polygon and/or MultiPolygon shape, or grid points along the path of a LineString and/or MultiLineString. If the shape consists of several disjoint polygons, the output is cut to the smallest bbox including all polygons.

Parameters

- **ds** (*Union[xarray.DataArray, xarray.Dataset]*) – Input values.
- **shape** (*Union[str, Path, gpd.GeoDataFrame]*) – Path to shape file, or directly a geodataframe. Supports formats compatible with geopandas.
- **raster_crs** (*Optional[Union[str, int]]*) – EPSG number or PROJ4 string.
- **shape_crs** (*Optional[Union[str, int]]*) – EPSG number or PROJ4 string.
- **buffer** (*Optional[Union[int, float]]*) – Buffer the shape in order to select a larger region stemming from it. Units are based on the shape degrees/metres.
- **start_date** (*Optional[str]*) – Start date of the subset. Date string format – can be year ("%Y"), year-month ("%Y-%m") or year-month-day("%Y-%m-%d"). Defaults to first day of input data-array.
- **end_date** (*Optional[str]*) – End date of the subset. Date string format – can be year ("%Y"), year-month ("%Y-%m") or year-month-day("%Y-%m-%d"). Defaults to last day of input data-array.
- **first_level** (*Optional[Union[int, float]]*) – First level of the subset. Can be either an integer or float. Defaults to first level of input data-array.
- **last_level** (*Optional[Union[int, float]]*) – Last level of the subset. Can be either an integer or float. Defaults to last level of input data-array.

Returns *Union[xarray.DataArray, xarray.Dataset]* – A subset of *ds*

Notes

If no CRS is found in the shape provided (e.g. RFC-7946 GeoJSON, https://en.wikipedia.org/wiki/GeoJSON), assumes a decimal degree datum (CRS84). Be advised that EPSG:4326 and OGC:CRS84 are not identical as axis order of lat and long differs between the two (for more information, see: https://github.com/OSGeo/gdal/issues/2035).

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import subset_shape
>>> pr = xr.open_dataset(path_to_pr_file).pr
...
# Subset data array by shape
>>> prSub = subset_shape(pr, shape=path_to_shape_file)
...
# Subset data array by shape and single year
>>> prSub = subset_shape(pr, shape=path_to_shape_file, start_date='1990-01-01', end_
↪date='1990-12-31')
...
# Subset multiple variables in a single dataset
>>> ds = xr.open_mfdataset([path_to_tasmin_file, path_to_tasmax_file])
>>> dsSub = subset_shape(ds, shape=path_to_shape_file)
```

clisops.core.subset.**subset_time**(*da: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *start_date: Optional[str] = None*, *end_date: Optional[str] = None*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Subset input DataArray or Dataset based on start and end years. Return a subset of a DataArray or Dataset for dates falling within the provided bounds.

Parameters

- **da** (*Union[xarray.DataArray, xarray.Dataset]*) – Input data.
- **start_date** (*Optional[str]*) – Start date of the subset. Date string format – can be year ("%Y"), year-month ("%Y-%m") or year-month-day("%Y-%m-%d"). Defaults to first day of input data-array.
- **end_date** (*Optional[str]*) – End date of the subset. Date string format – can be year ("%Y"), year-month ("%Y-%m") or year-month-day("%Y-%m-%d"). Defaults to last day of input data-array.

Returns *Union[xarray.DataArray, xarray.Dataset]* – Subsetted xarray.DataArray or xarray.Dataset

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import subset_time
>>> ds = xr.open_dataset(path_to_pr_file)
...
# Subset complete years
>>> prSub = subset_time(ds.pr,start_date='1990',end_date='1999')
...
# Subset single complete year
```

(continues on next page)

(continued from previous page)

```
>>> prSub = subset_time(ds.pr,start_date='1990',end_date='1990')
...
# Subset multiple variables in a single dataset
>>> ds = xr.open_mfdataset([path_to_tasmax_file, path_to_tasmin_file])
>>> dsSub = subset_time(ds,start_date='1990',end_date='1999')
...
# Subset with year-month precision - Example subset 1990-03-01 to 1999-08-31␣
↪inclusively
>>> txSub = subset_time(ds.tasmax,start_date='1990-03',end_date='1999-08')
...
# Subset with specific start_dates and end_dates
>>> tnSub = subset_time(ds.tasmin,start_date='1990-03-13',end_date='1990-08-17')
```

Notes

TODO add notes about different calendar types. Avoid "%Y-%m-31". If you want complete month use only "%Y-%m".

clisops.core.subset.**subset_time_by_components**(*da: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, ***, *time_components: Optional[Dict] = None*)

Subsets by one or more time components (year, month, day etc).

Parameters

- **da** (*Union[xarray.DataArray, xarray.Dataset]*) – Input data.
- **time_components** (*Union[Dict, None] = None*)

Returns *xarray.DataArray*

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import subset_time_by_components
...
To select all Winter months (Dec, Jan, Feb) or [12, 1, 2]:
>>> da = xr.open_dataset(path_to_file).pr
>>> winter_dict = {"month": [12, 1, 2]}
>>> res = subset_time_by_components(da, time_components=winter_dict)
```

clisops.core.subset.**subset_time_by_values**(*da: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *time_values: Optional[Sequence[str]] = None*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Subset input DataArray or Dataset based on a sequence of datetime strings. Return a subset of a DataArray or Dataset for datetimes matching those requested.

Parameters

- **da** (*Union[xarray.DataArray, xarray.Dataset]*) – Input data.
- **time_values** (*Optional[Sequence[str]] = None*)

Returns *Union[xarray.DataArray, xarray.Dataset]* – Subsetted xarray.DataArray or xarray.Dataset

Examples

```
>>> import xarray as xr
>>> from clisops.core.subset import subset_time_by_values
>>> ds = xr.open_dataset(path_to_pr_file)
...
# Subset a selection of datetimes
>>> times = ["2015-01-01", "2018-12-05", "2021-06-06"]
>>> prSub = subset_time_by_values(ds.pr, time_values=times)
```

Notes

If any datetimes are not found, a ValueError will be raised. The requested datetimes will automatically be re-ordered to match the order in the input dataset.

4.2 Core average functionality

Average module.

clisops.core.average.**average_over_dims**(*ds: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *dims: Optional[Tuple[str]] = None*, *ignore_undetected_dims: bool = False*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Average a DataArray or Dataset over the dimensions specified.

Parameters

- **ds** (*Union[xr.DataArray, xr.Dataset]*) – Input values.
- **dims** (*Tuple[str] = None*) – The dimensions over which to apply the average. If None, none of the dimensions are averaged over. Dimensions must be one of ["time", "level", "latitude", "longitude"].
- **ignore_undetected_dims** (*bool*) – If the dimensions specified are not found in the dataset, an Exception will be raised if set to True. If False, an exception will not be raised and the other dimensions will be averaged over. Default = False

Returns *Union[xr.DataArray, xr.Dataset]* – New Dataset or DataArray object averaged over the indicated dimensions. The indicated dimensions will have been removed.

Examples

```
>>> from clisops.core.average import average_over_dims
>>> pr = xr.open_dataset(path_to_pr_file).pr
...
# Average data array over latitude and longitude
>>> prAvg = average_over_dims(pr, dims=['latitude', 'longitude'], ignore_undetected_
↪dims=True)
```

clisops.core.average.**average_shape**(*ds: xarray.core.dataset.Dataset*, *shape: Union[str, pathlib.Path, geopandas.geodataframe.GeoDataFrame]*, *variable: Optional[Union[str, Sequence[str]]] = None*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Average a DataArray or Dataset spatially using vector shapes.

Return a DataArray or Dataset averaged over each Polygon given. Requires xESMF >= 0.5.0.

Parameters

- **ds** (*xarray.Dataset*) – Input values, coordinate attributes must be CF-compliant.
- **shape** (*Union[str, Path, gpd.GeoDataFrame]*) – Path to shape file, or directly a geodataframe. Supports formats compatible with geopandas. Will be converted to EPSG:4326 if needed.
- **variable** (*Union[str, Sequence[str], None]*) – Variables to average. If None, average over all data variables.

Returns *Union[xarray.DataArray, xarray.Dataset]* – *ds* spatially-averaged over the polygon(s) in *shape*. Has a new *geom* dimension corresponding to the index of the input geodataframe. Non-geometry columns of the geodataframe are copied as auxiliary coordinates.

Notes

The spatial weights are computed with ESMF, which uses corners given in lat/lon format (EPSG:4326), the input dataset *ds* must provide those. In opposition to *subset.subset_shape*, the weights computed here take partial overlaps and holes into account.

As xESMF computes the weight masks only once, skipping missing values is not really feasible. Thus, all NaNs propagate when performing the average.

Examples

```
>>> import xarray as xr
>>> from clisops.core.average import average_shape
>>> pr = xr.open_dataset(path_to_pr_file).pr
...
# Average data array over shape
>>> prAvg = average_shape(pr, shape=path_to_shape_file)
...
# Average multiple variables in a single dataset
>>> ds = xr.open_mfdataset([path_to_tasmin_file, path_to_tasmax_file])
>>> dsAvg = average_shape(ds, shape=path_to_shape_file)
```

4.3 Subset operation

class clisops.ops.subset.**Subset**(*ds*, *file_namer='standard'*, *split_method='time:auto'*, *output_dir=None*, *output_type='netcdf'*, ***params*)

Bases: clisops.ops.base_operation.Operation

4.4 Average operation

clisops.ops.average.**average_over_dims**(*ds*, *dims: Optional[Union[Tuple[str], roocs_utils.parameter.dimension_parameter.DimensionParameter]] = None*, *ignore_undetected_dims: bool = False*, *output_dir: Optional[Union[str, pathlib.Path]] = None*, *output_type='netcdf'*, *split_method='time:auto'*, *file_namer='standard'*) → List[Union[xarray.core.dataset.Dataset, str]]

Parameters

- **ds** (*Union[xr.Dataset, str]*)
- **dims** (*Optional[Union[Tuple[str], DimensionParameter]]*) – The dimensions over which to apply the average. If None, none of the dimensions are averaged over. Dimensions must be one of ["time", "level", "latitude", "longitude"].
- **ignore_undetected_dims** (*bool*) – If the dimensions specified are not found in the dataset, an Exception will be raised if set to True. If False, an exception will not be raised and the other dimensions will be averaged over. Default = False
- **output_dir** (*Optional[Union[str, Path]] = None*)
- **output_type** (*{"netcdf", "nc", "zarr", "xarray"}*)
- **split_method** (*{"time:auto"}*)
- **file_namer** (*{"standard", "simple"}*)

Returns

- *List[Union[xr.Dataset, str]]*
- *A list of the outputs in the format selected; str corresponds to file paths if the*
- *output format selected is a file.*

Examples

ds: xarray Dataset or "cmip5.output1.MOHC.HadGEM2-ES.rcp85.mon.atmos.Amon.r1i1p1.latest.tas"
dims: ['latitude', 'longitude']
ignore_undetected_dims: False
output_dir: "/cache/wps/procs/req0111"
output_type: "netcdf"
split_method: "time:auto"
file_namer: "standard"

4.5 Common functions

clisops.utils.common.**expand_wildcards**(*paths: Union[str, pathlib.Path]*) → list
Expand the wildcards that may be present in Paths.

4.6 Output Utilities

class clisops.utils.output_utils.**FileLock**(*fpath*)
Bases: object

From https://github.com/cedadev/cmip6-object-store/cmip6_zarr/file_lock.py

acquire(*timeout=10*)

release()

clisops.utils.output_utils.**check_format**(*fmt*)
Checks requested format exists.

clisops.utils.output_utils.**filter_times_within**(*times*, *start=None*, *end=None*)
Takes an array of datetimes, returning a reduced array if start or end times are defined and are within the main array.

clisops.utils.output_utils.**get_chunk_length**(*da*)
Calculate the chunk length to use when chunking xarray datasets.

Based on memory limit provided in config and the size of the dataset.

clisops.utils.output_utils.**get_da**(*ds*)
Returns xr.DataArray when format of ds may be either xr.Dataset or xr.DataArray.

clisops.utils.output_utils.**get_format_extension**(*fmt*)
Finds the extension for the requested output format.

clisops.utils.output_utils.**get_format_writer**(*fmt*)
Finds the output method for the requested output format.

clisops.utils.output_utils.**get_output**(*ds*, *output_type*, *output_dir*, *namer*)
Return output after applying chunking and determining the output format and chunking

clisops.utils.output_utils.**get_time_slices**(*ds: Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]*, *split_method*, *start=None*, *end=None*, *file_size_limit: Optional[str] = None*) → List[Tuple[str, str]]
Take an xarray Dataset or DataArray, assume it can be split on the time axis into a sequence of slices. Optionally, take a start and end date to specify a sub-slice of the main time axis.

Use the prescribed file size limit to generate a list of ("YYYY-MM-DD", "YYYY-MM-DD") slices so that the output files do not (significantly) exceed the file size limit.

Parameters

- **ds** (*Union[xr.Dataset, xr.DataArray]*)
- **split_method**
- **start**
- **end**
- **file_size_limit** (*str*) – a string specifying "<number><units>".

Returns *List[Tuple[str, str]]*

4.7 File Namers

class clisops.utils.file_namers.**SimpleFileNamer**(*replace=None*, *extra=''*)
Bases: clisops.utils.file_namers._BaseFileNamer

Simple file namer class.

Generates numbered file names.

class clisops.utils.file_namers.**StandardFileNamer**(*replace=None*, *extra=''*)
Bases: clisops.utils.file_namers.SimpleFileNamer

Standard file namer class.

Generates file names based on input dataset.

get_file_name(*ds*, *fmt='nc'*)
Constructs file name.

clisops.utils.file_namers.**get_file_namer**(*name*)
Returns the correct filenamer from the provided name

4.8 Dataset Utilities

clisops.utils.dataset_utils.**adjust_date_to_calendar**(*da*, *date*, *direction='backwards'*)
Check that the date specified exists in the calendar type of the dataset. If not, change the date a day at a time (up to a maximum of 5 times) to find a date that does exist.

The direction to change the date by is indicated by 'direction'.

Parameters

- **da** – xarray.Dataset or xarray.DataArray
- **date** – The date to check, as a string.
- **direction** – The direction to move in days to find a date that does exist. 'backwards' means the search will go backwards in time until an existing date is found. 'forwards' means the search will go forwards in time.

  The default is 'backwards'.

Returns (str) The next possible existing date in the calendar of the dataset.

clisops.utils.dataset_utils.**calculate_offset**(*lon*, *first_element_value*)
Calculate the number of elements to roll the dataset by in order to have longitude from within requested bounds.

Parameters

- **lon** – longitude coordinate of xarray dataset.
- **first_element_value** – the value of the first element of the longitude array to roll to.

clisops.utils.dataset_utils.**check_lon_alignment**(*ds*, *lon_bnds*)
Check whether the longitude subset requested is within the bounds of the dataset. If not try to roll the dataset so that the request is. Raise an exception if rolling is not possible.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/roocs/clisops/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

clisops could always use more documentation, whether as part of the official clisops docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/roocs/clisops/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *clisops* for local development.

1. Fork the *clisops* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:roocs/clisops.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
# For virtualenv environments:
$ mkvirtualenv clisops

# For Anaconda/Miniconda environments:
$ conda create -n clisops python=3.7

$ cd clisops/
$ pip install -e .
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally!

5. When you're done making changes, check that you verify your changes with *black* and run the tests, including testing other Python versions with *tox*:

```
# For virtualenv environments:
$ pip install black pytest tox

# For Anaconda/Miniconda environments:
$ conda install -c conda-forge black pytest tox

$ black --target-python py36 clisops tests
$ python setup.py test
$ tox
```

6. Before committing your changes, we ask that you install *pre-commit* in your virtualenv. *Pre-commit* runs git hooks that ensure that your code resembles that of the project and catches and corrects any small errors or inconsistencies when you *git commit*:

```
# For virtualenv environments:
$ pip install pre-commit

# For Anaconda/Miniconda environments:
$ conda install -c conda-forge pre_commit

$ pre-commit install
```

7. Commit your changes and push your branch to GitHub:

```
$ git add *

$ git commit -m "Your detailed description of your changes."
# `pre-commit` will run checks at this point:
# if no errors are found, changes will be committed.
# if errors are found, modifications will be made. Simply `git commit` again.

$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, please follow these guidelines:

1. Open an *issue* on our GitHub repository with your issue that you'd like to fix or feature that you'd like to implement.
2. Perform the changes, commit and push them either to new a branch within roocs/clisops or to your personal fork of clisops.

Warning: Try to keep your contributions within the scope of the issue that you are addressing. While it might be tempting to fix other aspects of the library as it comes up, it's better to simply to flag the problems in case others are already working on it.

Consider adding a "**# TODO:**" comment if the need arises.

3. Pull requests should raise test coverage for the clisops library. Code coverage is an indicator of how extensively tested the library is. If you are adding a new set of functions, they **must be tested** and **coverage percentage should not significantly decrease.**
4. If the pull request adds functionality, your functions should include docstring explanations. So long as the docstrings are syntactically correct, sphinx-autodoc will be able to automatically parse the information. Please ensure that the docstrings adhere to one of the following standards:
   - numpydoc
   - reStructuredText (ReST)
5. The pull request should work for Python 3.7 as well as raise test coverage. Pull requests are also checked for documentation build status and for PEP8 compliance.

   The build statuses and build errors for pull requests can be found at: https://travis-ci.org/roocs/clisops/pull_requests

Warning: PEP8 and Black is strongly enforced. Ensure that your changes pass **Flake8** and **Black** tests prior to pushing your final commits to your branch. Code formatting errors are treated as build errors and will block your pull request from being accepted.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_clisops
```

5.5 Versioning

In order to update and release the library to PyPI, it's good to use a semantic versioning scheme. The method we use is as follows:

```
major.minor.patch-release
```

Major releases denote major changes resulting in a stable API;

Minor is to be used when adding a module, process or set of components;

Patch should be used for bug fixes and optimizations;

Release is a keyword used to specify the degree of production readiness (*beta* [, and optionally, *gamma*])

> An increment to the Major or Minor will reset the Release to *beta*. When a build is promoted above *beta* (ie: release-ready), it's a good idea to push this version towards PyPi.

5.6 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (**including an entry in HISTORY.rst**). Then run:

```
$ bumpversion <option> # possible options: major / minor / patch / release
$ git push
$ git push --tags
```

5.7 Packaging

When test coverage and stability is adequate, maintainers should update the pip-installable package (wheel) on PyPI. In order to do this, you will need the following libraries installed:

- twine
- setuptools
- wheel

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Ag Stephens ag.stephens@stfc.ac.uk

6.2 Co-Developers

- Eleanor Smith eleanor.smith@stfc.ac.uk @ellesmith88
- Carsten Ehbrecht ehbrecht@dkrz.de
- Trevor James Smith smith.trevorj@ouranos.ca @Zeitsperre

6.3 Contributors

- Pascal Bourgault bourgault.pascal@ouranos.ca @aulemahal
- David Huard huard.david@ouranos.ca @huard
- Martin Schupfner schupfner@dkrz.de **`@sol1105 <https://github.com/sol1105`_**

CHAPTER

SEVEN

VERSION HISTORY

7.1 v0.8.0 (2022-01-13)

7.1.1 New Features

- `clisops.core.average.average_shape` copies the global and variable attributes from the input data to the results.

7.1.2 Bug fixes

- `average_shape` and `create_weight_masks` were adapted to work with xESMF 0.6.2, while maintaining compatibility with earlier versions.
- Fix added to remove `_FillValue` added to coordinate variables and bounds by xarray when outputting to netCDF.

7.1.3 Other Changes

- Passing `DataArray` objects to `clisops.core.average.average_shape` is now deprecated. Averaging requires grid cell boundaries, which are not `DataArray` coordinates, but independent `Dataset` variables. Please pass `Dataset` objects and an optional list of variables to average.
- `average_shape` performs an initial subset over the averaging region, before computing the weights, to reduce memory usage.
- Minimum xesmf version set to 0.6.2.
- Minimum pygeos version set to 0.9.
- Replace `cascaded_union` by `unary_union` to anticipate a *shapely* deprecation.

7.2 v0.7.0 (2021-10-26)

7.2.1 Breaking Changes

- `time` input for `time` in `ops.subset.subset` but now be one of [<class 'roocs_utils.parameter.param_utils.Interval'>, <class 'roocs_utils.parameter.param_utils.Series'>, <class 'NoneType'>, <class 'str'>].
- `level` input for `level` in `ops.subset.subset` but now be one of [<class 'roocs_utils.parameter.param_utils.Interval'>, <class 'roocs_utils.parameter.param_utils.Series'>, <class 'NoneType'>, <class 'str'>].
- `roocs-utils`>= 0.5.0 required.

7.2.2 New Features

- `time_values` and `level_values` arguments added to `core.subset.subset_bbox` which allows the user to provide a list of time/level values to select.
- `subset_time_by_values` and `subset_level_by_values` added to `core.subset.subset_bbox`. These allow subsetting on sequence of datetimes or levels.
- `subset_time_by_components` added to `core.subset.subset_bbox`. This allows subsetting by time components - year, month, day etc.
- `check_levels_exist` and `check_datetimes_exist` function checkers added in `core.subset` to check requested levels and datetimes exist. An exception is raised if they do not exist in the dataset.
- `time_components` argument added to `ops.subset` to allowing subsetting by time components such as year, month, day etc.

7.2.3 Other Changes

- Python 3.6 no longer tested in GitHub actions.

7.3 v0.6.5 (2021-06-10)

7.3.1 New Features

- New optional dependency `PyGEOS`, when installed the performance of `core.subset.create_mask` and `cure.subset.subset_shape` are greatly improved.

7.4 v0.6.4 (2021-05-17)

7.4.1 Breaking Changes

- Exception raised in `core.average.average_over_dims` when dims is None.
- Exception raised in `core.average.average_over_shape` when grid and polygon have no overlapping values.

7.4.2 New Features

- `ops.subset.subset` now ensures all latitude and longitude bounds are in ascending order before passing to `core.subset.subset_bbox`
- `core.subset.subset_level` now checks that the order of the bounds matches the order of the level data.
- `core.subset._check_desc_coords` now checks the bounds provided are ascending before flipping them.

7.4.3 Other Changes

- clisops logging no longer disables other loggers.
- GitHub CI now leverages `tox` for testing as well as tests averaging functions via a conda-based build.
- Added a CI build to run against xarray@master that is allowed to fail.

7.5 v0.6.3 (2021-03-30)

7.5.1 Breaking Changes

- Raise an exception in `core.subset.subset_bbox` when there are no data points in the result.
- `roocs-utils`>=0.3.0 required.

7.5.2 Bug Fixes

- In `core.subset.check_start_end_dates` check if start and end date requested exist in the calendar of the dataset. If not, nudge the date forward if start date or backwards if end date.

7.5.3 Other Changes

- Error message improved to include longitude bounds of the dataset when the bounds requested in `ops.subset.subset` are not within range and rolling could not be completed.

7.6 v0.6.2 (2021-03-22)

7.6.1 Bug Fixes

- Better support for disjoint shapes in `subset_shape`.
- Identify latitude and longitude using `cf-xarray` rather than by "lat" and "lon"

7.6.2 New Features

- Add `output_staging_dir` option in *etc/roocs.ini*, to write files to initially before moving them to the requested output_dir.
- Notebook of examples for average over dims operation added.

7.7 v0.6.1 (2021-02-23)

7.7.1 Bug Fixes

- Add `cf-xarray` as dependency. This is a dependency of `roocs-utils`>=0.2.1 so is not a breaking change.
- Remove `python-dateutil`, `fiona` and `geojson` as dependencies, no longer needed.

7.8 v0.6.0 (2021-02-22)

7.8.1 Breaking Changes

- New dev dependency: ``GitPython``==3.1.12
- `roocs-utils`>=0.2.1 required.

7.8.2 New Features

- `average_over_dims` added into `average.core` and `average.ops`
- New `core.average.average_shape` + `core.subset.subset_create_weight_masks`. Depends on *xESMF* >= 0.5.2, which is a new optional dependency.

7.8.3 Bug Fixes

- Fixed issue where the temporal subset was ignored if level subset selected.
- Roll dataset used in subsetting when the requested longitude bounds are not within those of the dataset.
- Fixed issue with subsetting grid lon and lat coordinates that are in descending order for `core.subset.subset_bbox`.

7.8.4 Other Changes

- Changes to allow datasets without a time dimension to be processed without issues.
- Use `DatasetMapper` from `roocs-utils` to ensure all datasets are mapped to file paths correctly.
- Using file caching to gather `mini-esgf-data` test data.
- Added a `dev` recipe for pip installations (*pip install clisops[dev]*).
- Updated pre-commit and pre-commit hooks to newest versions.
- Migrated linux-based integration builds to GitHub CI.
- Added functionality to `core.subset.create_mask` so it can accept `GeoDataFrames` with non-integer indexes.
- `clisops.utils.file_namers` adjusted to allow values to be overwritten and extras to be added to the end before the file extension.

7.9 v0.5.1 (2021-01-11)

7.9.1 Breaking Changes

- Reverting breaking changes made by the change to `core.subset.create_mask`. This change introduces a second evaluation for shapes touching grid-points.

7.9.2 Other Changes

- Using file caching to gather `xclim` test data.
- Change made to `core.subset.subset_bbox._check_desc_coords` to cope with subsetting when only one latitude or longitude exists in the input dataset

7.10 v0.5.0 (2020-12-17)

7.10.1 Breaking Changes

- Moved `core.subset.create_mask_vectorize` to `core.subset.create_mask`. The old spatial join option was removed.
- `core.subset.subset_shape` lost its `vectorize` kwarg, as it is now default.
- `roocs-utils`>0.1.5 used

7.10.2 Other Changes

- udunits2>=2.2 removed as a requirement to make clisops completely pip installable.
- rtee and libspatialindex removed as requirements, making it easier to install through pip.
- Static types updated to include missing but permitted types.
- Better handling for paths in ops.subset allowing windows build to be fixed.

7.11 v0.4.0 (2020-11-10)

Adding new features, updating doc strings and documentation and inclusion of static type support.

7.11.1 Breaking Changes

- clisops now requires udunits2>=2.2.
- roocs-utils>=0.1.4 is now required.
- space parameter of clisops.ops.subset renamed to area.
- chunk_rules parameter of clisops.ops.subset renamed to split_method.
- filenamer parameter of clisops.ops.subset renamed to file_namer.

7.11.2 New Features

- subset_level added.
- PR template.
- Config file now exists at clisops.etc.roocs.ini. This can be overwritten by setting the environment variable ROOCS_CONFIG to the file path of a config file.
- Static typing added to subset operation function.
- info and debugging are now logged rather than printed.
- Notebook of examples for subset operation added.
- split_method implemented to split output files by if they exceed the memory limit provided in clisops.etc.roocs.ini named file_size_limit. Currently only the time:auto exists which splits evenly on time ranges.
- file_namer implemented in clisops.ops.subset. This has simple and standard options. simple numbers output files whereas standard names them according to the input dataset.
- Memory usage when completing the subsetting operation is now managed using dask chunking. The memory limit for memory usage for this process is set in clisops.etc.roocs.ini under chunk_memory_limit.

7.11.3 Bug Fixes

- Nudging time values to nearest available in dataset to fix a bug where subsetting failed when the exact date did not exist in the dataset.

7.11.4 Other Changes

- `cfunits` dependency removed - not needed.
- requirements.txt and environment.yml synced.
- Documentation updated to include API.
- Read the docs build now tested in CI pipeline.
- md files changed to rst.
- tests now use `mini-esgf-data` by default.

7.12 v0.3.1 (2020-08-04)

7.12.1 Other Changes

- Add missing `rtree` dependency to ensure correct spatial indexing.

7.13 v0.3.0 (2020-07-23)

7.13.1 Other Changes

- Update testdata and subset module (#34).

7.14 v0.2.1 (2020-07-08)

7.14.1 Other Changes

- Fixed docs version (#25).

7.15 v0.2.0 (2020-06-19)

7.15.1 New Features

- Integration of xclim subset module in `clisops.core.subset`.
- Added jupyter notebook with and example for subsetting from xclim.

7.15.2 Other Changes

- Fixed RTD doc build.
- Updated travis CI according to xclim requirements.
- Now employing PEP8 + Black compatible autoformatting.
- Pre-commit is now used to launch code formatting inspections for local development.

7.16 v0.1.0 (2020-04-22)

- First release.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX