cmipdata Documentation

Release 0.6

Neil Swart

April 12, 2016

Contents

cmipdata is a python package for preprocessing large ensembles of climate model data in standardized NetCDF files, such as those used in the Coupled Model Intercomparison Project (CMIP). The primary usage is to process the raw netCDF data from many models/realizations/experiments into a useful form for further analysis (e.g. by time-joining or slicing, remapping, averaging etc). **cmipdata** is the python wrapper that intelligently interfaces with the ensemble of model data, while the underlying data processing is done efficiently and transparently using Climate Data Operators (cdo). Limited functionality for loading processed data into numpy arrays and making basic plots is also provided.

Contents:

CHAPTER 1

Quickstart

1.1 Installing

You can install cmipdata with pip from pypi:

```
pip install cmipdata
```

or the latest version directly from github

```
pip install git+https://github.com/swartn/cmipdata.git
```

cmipdata has primarily been developed and tested within the anaconda python distribution on Linux x86/x64 and Mac OSX. Windows is not supported.

You can (should) do this inside a virtual environment. In that case it will work without root privileges. If you are using anaconda see http://conda.pydata.org/docs/faq.html#env.

Dependencies

The external package Climate Data Operators (cdo) v1.6 or later is required. Python dependencies are handled by pip.

1.2 Using cmipdata

After a successful installation, you can import cmipdata as you would any other package in python:

```
import cmipdata as cd
```

The next step is to create an `Ensemble`. `Ensemble` objects are the structures used in cmipdata to organize climate model data. The assumption is that you have some CMIP-like netCDF model data on your local disk (cmipdata does not facillitate downloading data). For this example we have several hundred CMIP5 sea-ice concentration files in our directory, downloaded from the ESGF. To create an ensemble object, simple use `mkensemble()`, specifing the filepattern to match:

```
   In [2]: ens = cd.mkensemble('sic_OImon*')
   This ensemble contains:
    49 models
    49 realizations
    1 experiments
    1 variables
    279 associated files

For more details use ens.fulldetails()
```

The printout tells us some details about the data in our new ensemble. It consists of 49 models, 49 realizations, 1 experiment and 1 variable, all of which are also objects in the organizational paradigm of cmipdata (see *The cmipdata API*). There are many more files than model or realizations, because for some models the experiment is broken up into multiple files, each representing a time-slice. In this example there is only one experiment (historical), but there could be many (if we had files for the RCPs experiments too). cmipdata provided the ability to join these multiple time-slice files together, and perform a host of other elaborate processing.

CHAPTER 2

Examples

2.1 Concatenate model time-slices

In this example we have the surface temperature files for HadCM3 in our directory, which are provided in six time-slices for the historical experiment, as the files:

- ts_Amon_HadCM3_historical_r1i1p1_185912-188411.nc
- ts_Amon_HadCM3_historical_r1i1p1_188412-190911.nc
- ts_Amon_HadCM3_historical_r1i1p1_190912-193411.nc
- ts_Amon_HadCM3_historical_r1i1p1_193412-195911.nc
- ts_Amon_HadCM3_historical_r1i1p1_195912-198411.nc
- ts_Amon_HadCM3_historical_r1i1p1_198412-200512.nc

To join them we can use the `cat_exp_slices()` function from cmipdata:

```
import cmipdata as cd
ens = cd.mkensemble('ts_Amon_HadCM3*')
ens = cd.cat_exp_slices(ens)
```

The result is one unified file in our directory, which has been appropriately named. By default the individual time-slice fields will be deleted, and only the joined file is left in our directory. We can change this by passing the `delete=False` option to `cat_exp_slices()`. We were also returned an updated ensemble object, the structure of which we can view as follows:

```
ens.fulldetails()

HadCM3:
    historical
            r1i1p1
                    ts
                            ts_Amon_HadCM3_historical_r1i1p1_185912-200512.nc
```

Note that the joined file has been named in such a way that the start and end dates cover the full range of the input files.

This example shows only one model, but the method works equally well for a large ensemble consisting of multple models, each with multiple realizations and hundreds of files. For example, using:

```
ens = cd.mkensemble('ts_Amon_*')
```

would build an ensemble consisting of all files in the present directory starting with ts_Amon, and join_exp_slices would, on a per-realization basis, do the joining, when necessary.

2.2 Concatenate experiments

The CMIP5 experimental design consisted of various experiments. Sometimes it is desirable to time-join experiments, for example, the historical and RCP4.5 experiments can be join together for each realization from each model to provide a continuous time-series that runs from 1871 to 2100. The `cat_experiments()` function provides this functionality. In this example we join the historical and RCP4.5 sea-ice extent fields for many models and realizations:

```
import cmipdata as cd

ens = cd.mkensemble('sic_OImon*')

    This ensemble contains:
        49 models
       369 realizations
         2 experiments
         1 variables
      1642 associated files

    For more details use ens.fulldetails()

ens_joined = cd.cat_experiments(ens, 'sic', 'historical', 'rcp45')
```

This operation takes some time, prints out progress along the way, and returns an updated ensemble. By default the input files will be deleted, and only the joined files will remain in our directory. `cat_experiments()` takes care of doing joining of multiple time-slices within experiments, as well as joining the two experiments together. After this operation the number of files have been reduced, due to joining. The number of models or realizations in the returned ensemble may also be lower, because only models/realizations that have both a historical and RCP4.5 experiment available are retained. prints this information out for us:

```
Models deleted from ensemble (missing one experiment completely):

        Model   Experiment

        CESM1-FASTCHEM          historical
        CESM1-CAM5-1-FV2        historical
        CNRM-CM5-2      historical
        MPI-ESM-P       historical
        CMCC-CESM       historical
        MRI-ESM1        historical


Realizations deleted (missing from one experiment):

        Model   Realizations

        IPSL-CM5A-LR    r6i1p1 r5i1p1
        bcc-csm1-1-m    r2i1p1 r3i1p1
        GFDL-CM3        r2i1p1 r4i1p1
        ACCESS1-0       r2i1p1
        CNRM-CM5        r7i1p1 r9i1p1 r2i1p1 r10i1p1 r4i1p1 r3i1p1 r6i1p1 r5i1p1
                        r8i1p1
        GISS-E2-H       r6i1p1 r6i1p3
        FGOALS-g2       r2i1p1 r4i1p1 r3i1p1 r5i1p1
```

```
        ACCESS1-3       r2i1p1 r3i1p1
        CCSM4   r1i2p1 r1i2p2
        HadGEM2-ES      r5i1p1
        GISS-E2-R       r1i1p128 r1i1p122 r1i1p121 r1i1p126 r1i1p127 r1i1p124
                        r1i1p125 r6i1p2
        HadGEM2-CC      r2i1p1 r3i1p1
        MIROC-ESM       r2i1p1 r3i1p1
        EC-EARTH        r5i1p1
        MRI-CGCM3       r4i1p2 r2i1p1 r3i1p1 r5i1p2
        NorESM1-M       r2i1p1 r3i1p1
        bcc-csm1-1      r2i1p1 r3i1p1
        CESM1-WACCM     r1i1p1
        IPSL-CM5A-MR    r2i1p1 r3i1p1
```

We can use the `sinfo()` command to get an update on what the returned ensmble looks like:

```
ens_joined.sinfo()
This ensemble contains:
 43 models
 153 realizations
 1 experiments
 1 variables
 153 associated files
```

After the joining we can now see that there is one file per realization.

2.3 remap

To remap of all the sea ice concentration (sic) files in our directory to a common one-degree by one-degree grid, simply do:

```
import cmipdata as cd

ens = cd.mkensemble('sic_OImon*')
ens = cd.remap(ens, remap='r360x180')
```

The string given to remap is any valid remapping option that can be given to cdo. For example, you could give the name of a target grid file to remap to. The `remap()` function also allows you to choose the remapping method, but distance weighted remapping is used by default.

2.4 Zonal mean

To create a zonal mean of all the sea ice concentration (sic) files, simply do:

```
import cmipdata as cd

ens = cd.mkensemble('sic_OImon*')
ens = cd.zonmean(ens,delete=False)
```

2.5 Time slice

Choose a common time slice out of all the CMIP5 sea ice concentration files in our directory:

```
import cmipdata as cd
ens = cd.mkensemble('sic_OImon*')
ens = cd.time_slice(ens, start_date='1979-01-01', end_date='2013-12-31')
```

where `start_date` and `end_date` are given in YYYY-MM-DD format. This will create a new set of files covering the chosen period, and by default will delete the original input files (to prevent this specify `delete=False`). Any realizations that do not contain the full requested date range will be dropped from the ensemble (and deleted by default).

2.6 Custom cdo command

You can apply any valid cdo command to the whole ensemble using the `my_operator()` function. Chained cdo commands are allowed. In this example we will carry out several chained operations. Our objective is to get a time-anomalies of sea-ice for the period 1979 to 2013 relative to a base-period of 1991 to 2000. We also want the result to be zonally meaned, and remapped onto a 1-degree-latitude grid:

```
import cmipdata as cd

ens = cd.mkensemble('sic_OImon*')

my_cdo_str = 'cdo remapdis,r1x180 -zonmean -seldate,1979-01-01,2013-12-31' +
             '-sub {infile} -timmean -seldate,1991-01-01,2000-12-31 {infile}' +
             '{outfile}'

ens = cd.my_operator(ens, my_cdo_str, output_prefix='test_', delete=False)
```

The result is a set of files that begin with the prefix ‘‘test_‘, and an updated ensemble.

CHAPTER 3

The cmipdata API

This section describes the **cmipdata** Application Programming Interface (API). It contains a list of classes and functions, within the three core cmipdata modules: *classes*, *preprocessing_tools* and *loading_tools*. The little developed plotting_tools is also described.

3.1 classes

The classes module provides one classes and three functions. The class: DataNode

The core functionality of cmipdata is to organize a large number of model output files into a logical structure so that further processing can be done. Data is organized into a tree-like structure using the class DataNode as the nodes of a tree. The entire tree structure will be referred to as an ensemble. At each level of the tree the level is specified by the genre attribute.

Various methods exist to interact with the ensemble, and its constituent elements.

The *mkensemble()* function is used to create Ensemble objects, while match_ensembles() finds models common to two ensembles and match_reliazations() matches realizations between two ensembles. Once created, an ensemble can be used to harness the power of the *preprocessing_tools* to apply systematic operations to all files.

class classes.**DataNode**(*genre*, *name*, *parent=None*, ***kwargs*)

Bases: object

Defines a cmipdata DataNode.

Attributes

genre	(string) The attribute of DataNode
name	(string) The name of the particular genre
children	(list) List of DataNodees of genre beneath the current DataNode
parent	(DataNode) for genre 'ensemble' the parent is None
start_date	(string) for genre 'file'
end_date	(string) for genre 'file'
realm	(string) for genre 'variable' contains the realm of the varaible

Methods

add(*child*)
Add DataNode to children

Parameters **child** : DataNode

delete(*child*)
Delete DataNode from children

Parameters **child** : DataNode

fulldetails()
prints information about the number of models, experiments, variables and files ina DataNode tree.

fulldetails_tofile(*fi*)
prints information about the number of models, experiments, variables and files ina DataNode tree.

getChild(*input_name*)

Returns DataNode given the name of the DataNode if it is in children

Parameters **input_name** : string

Returns **DataNode** : Returns None if the DataNode is not in children

getDictionary()
Returns a dictionary which has the genres and their names for all the ancestors of the DataNode

getNameWithoutDates()
Return string name with the dates removed if present

Returns string

lister(*genre*, *unique=True*)
Returns a list of names of a particular genre

Parameters **genre** : string

the genre of returned list

unique: boolean

if True removes duplicates from the list

Return

——

list of strings

mer()

Returns a generator containing lists of length 3

with the DataNode genre:'realization' the DataNode genre:'experiment' string model-experiment-realization

Returns generator

objects(*genre*)
Returns a generator for a DataNode of a particular genre

Parameters **genre** : string

the genre of returned generator

parentobject(*genre*)
Returns the parent DataNode of a particular genre

Parameters **genre** : string

the genre of returned DataNode

sinfo(*listOfGenres=['variable', 'model', 'experiment', 'realization', 'ncfile']*)
Returns the number of models, experiments, realizations, variables and files in the DataNode

squeeze()
Remove any empty elements from the ensemble

classes.**match_models**(*ens1*, *ens2*, *delete=False*)
Find common models between two ensembles.

Parameters **ens1** : cmipdata ensemble

ens2 : cmipdata ensemble

the two cmipdata ensembles to compare.

Returns **ens1** : cmipdata ensemble

ens2 : cmipdata ensemble

two ensembles with matching models.

classes.**match_realizations**(*ens1*, *ens2*, *delete=False*)
Find common realizations between two ensembles.

Parameters **ens1** : cmipdata ensemble

ens2 : cmipdata ensemble

the two cmipdata ensembles to compare.

Returns **ens1** : cmipdata ensemble

ens2 : cmipdata ensemble

two ensembles with matching realizations.

classes.**mkensemble**(*filepattern*, *experiment='*'*, *prefix=''*, *kwargs=''*)
Creates and returns a cmipdata ensemble from a list of filenames matching filepattern.

Optionally specifying prefix will remove prefix from each filename before the parsing is done. This is useful, for example, to remove pre-pended paths used in filepattern (see example 2).

Once the list of matching filenames is derived, the model, experiment, realization, variable, start_date and end_date fields are extracted by parsing the filnames against a specified file naming convention. By default this is the CMIP5 convention, which is:

```
variable_realm_model_experiment_realization_startdate-enddate.nc
```

If the default CMIP5 naming convention is not used by your files, an arbitary naming convention for the parsing may be specified by the dicionary kwargs (see example 3).

Parameters **filepattern** : string

A string that by default is matched against all files in the current directory. But filepattern could include a full path to reference files not in the current directory, and can also include wildcards.

prefix : string

A pattern occuring in filepattern before the start of the official filename, as defined by the file naming converntion. For instance, a path preceeding the filename.

Examples

1. Create ensemble of all sea-level pressure files from the historical experiment in the current directory:

```
ens = mkensemble('psl*historical*.nc')
```

2. Create ensemble of all sea-level pressure files from all experiments in a non-local directory:

```
ens = mkensemble('/home/ncs/ra40/cmip5/sam/c5_slp/psl*'
                 , prefix='/home/ncs/ra40/cmip5/sam/c5_slp/')
```

3.Create ensemble defining a custom file naming convention:

```
kwargs = {'separator':'_', 'variable':0, 'realm':1, 'model':2, 'experiment':3,
          'realization':4, 'dates':5}

ens = mkensemble('psl*.nc', **kwargs)
```

3.2 preprocessing_tools

The preprocessing_tools module of cmipdata is a set of functions which use os.system calls to Climate Data Operators (cdo) to systematically apply a given processing on multiple NetCDF files, which are listed in cmipdata ensemble objects.

preprocessing_tools.**areaint** (*ensemble*, *delete=True*, *output_prefix=''*)

Calculate the area weighted integral for each file in ens.

The output files are prepended with 'area-integral'. The original the input files are removed if delete=True (default). An updated ensemble object is also returned.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the processing.

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory.

Examples

1.Compute the area integral for all files in ens:

```
ens = cd.areaint(ens)
```

preprocessing_tools.**areamean**(*ensemble*, *delete=True*, *output_prefix=''*)

Calculate the area mean for each file in ens.

The output files are prepended with 'area-mean'. The original the input files are removed if delete=True (default). An updated ensemble object is also returned.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the processing.

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory.

Examples

1.Compute the area mean for all files in ens:

```
area_mean_ens = cd.areamean(ens)
```

preprocessing_tools.**cat_exp_slices**(*ensemble*, *delete=True*, *output_prefix=''*)

Concatenate multiple time-slice files per experiment.

For all models in ens which divide their output into multiple files per experiment (time-slices), cat_exp_slices concatenates the files into one unified file, and deletes the individual slices, unless delete=False. The input ensemble can contain multiple models, experiments, realizations and variables, which cat_exp_slices will process independently. In other words, files are joined per-model, per-experiment, per-realization, per-variable. For example, if the ensemble contains two experiments for many models/realizations for variable psl, two unified files will be produced per realization: one for the historical and one for the rcp45 experiment. To join files over experiments (e.g. to concatenate historical and rcp45) see cat_experiments.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the concatenation.

delete : boolean

If delete=True, delete the individual time-slice files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly concatenated files.

The concatenated files are written to present working directory.

See also:

cat_experiments Concatenate the files for two experiments.

Examples

For a simple ensemble comprized of only 1 model, 1 experiment and one realization.:

```
    # Look at the ensemble structure before the concatenation
    ens.fulldetails()
    HadCM3:
        historical
                r1i1p1
                        ts
                              ts_Amon_HadCM3_historical_r1i1p1_185912-188411.nc
                              ts_Amon_HadCM3_historical_r1i1p1_188412-190911.nc
                              ts_Amon_HadCM3_historical_r1i1p1_190912-193411.nc
                              ts_Amon_HadCM3_historical_r1i1p1_193412-195911.nc
                              ts_Amon_HadCM3_historical_r1i1p1_195912-198411.nc
                              ts_Amon_HadCM3_historical_r1i1p1_198412-200512.nc

    # Do the concantenation
    ens = cd.cat_exp_slices(ens)

    # Look at the ensemble structure after the concatenation
    ens.fulldetails()
    HadCM3:
        historical
                r1i1p1
                        ts
                              ts_Amon_HadCM3_historical_r1i1p1_185912-200512.nc
```

preprocessing_tools.**cat_experiments**(*ensemble*, *variable_name*, *exp1_name*, *exp2_name*, *delete=True*, *output_prefix=''*)

Concatenate the files for two experiments.

Experiments exp1 and exp2 are concatenated into a single file for each realization of each model listed in ens. For each realization, the concatenated file for variable variable_name is written to the current working directory and the input files are deleted by default, unless delete=False.

The concatenation occurs for each realization for which input files exist for both exp1 and exp2. If no match is found for the realization in exp1 (i.e. there is no corresponding realization in exp2), then the files for both experiments are deleted from the path (unless delete=False) and the realization is removed from ens. Similarly if exp2 is missing for a given model, that model is deleted from ens.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the concatenation.

variable_name : str

The name of the variable to be concatenated.

exp1_name : str

The name of the first experiment to be concatenated (e.g. 'historical').

exp2_name : str

The name of the second experiment to be concatenated (e.g. 'rcp45').

delete : boolean

If delete=True, delete the individual time-slice files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly concatenated files.

The concatenated files are written to present working directory.

Examples

1.Join the historical and rcp45 simulations for variable ts in ens:

```
ens = cd.cat_experiments(ens, 'ts', exp1_name='historical', exp2_name='rcp45')
```

preprocessing_tools.**climatology**(*ensemble*, *delete=True*, *output_prefix=''*)

Compute the monthly climatology for each file in ens.

The climatology is calculated over the full file-length using cdo ymonmean, and the output files are prepended with '**climatology_**'. The original the input files are removed if delete=True (default). An updated ensemble object is also returned.

If you want to compute the climatology over a specific time slice, use time_slice before compute the climatology.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the remapping.

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory.

Examples

1.Compute the climatology:

```
climatology_ens = cd.climatology(ens)
```

preprocessing_tools.**del_ens_files**(*ensem*)

delete from disk all files listed in ensemble ens

preprocessing_tools.**ens_stats**(*ens*, *variable_name*, *output_prefix=''*)

Compute the ensemble mean and standard deviation.

The ensemble mean and standard deviation is computed over all models-realizations and experiments for variable variable_name in ens, such that each model has a weight of one. An output file is written containing the ensemble mean and another file is written with the standard deviation, containing the names '_ENS-**MEAN_**' and '_ENS-**STD_**' in the place of the model-name. If the ensemble contains multiple experiments, files are written for each experiment.

The ensemble in ens must be homogenous. That is to say all files must be on the same grid and span the same time-frame, within each experiment (see remap, and time_slice for more). Additionally, variable_name should have only one filename per realization and experiment. That is, join_exp_slice should have been applied.

The calculation is done by, first computing the mean over all realizations for each model; then for the ensemble, calculating the mean over all models. The standard deviation is calculated across models using the realization mean for each model.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the concatenation.

variable_name : str

The name of the variable to be concatenated.

Returns A tuple of lists containing the names of the mean and standard deviation files created

The ENS-MEAN and ENS-STD files are written to present working directory.

Examples

1.Compute the statistics for the ts variable:

```
>>cd.ens_stats(ens, 'ts')
```

experiment_list = ens.lister('experiment') for exname in experiment_list:

files_to_mean = [] for model in ens.objects('model'):

experiment = model.getChild(exname) if experiment != None:

modfilesall = [] for realization in experiment.children:

realization modfilesall.append(realization.getChild(variable_name).children)

preprocessing_tools.**my_operator**(*ensemble*, *my_cdo_str=''*, *output_prefix='processed_'*, *delete=False*)

Apply a customized cdo operation to all files in ens.

For each file in ens the command in my_cdo_str is applied and an output file appended by 'output_prefix' is created.

Optionally delete the original input files if delete=True.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the processing.

my_cdo_str : str

The (chain) of cdo commands to apply. Defined variables which can be used in my_cdo_str are: model, experiment, realization, variable, infile, outfile

output_prefix : str

The string to prepend to the processed filenames.

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory.

Examples

1.Do an annual mean:

```
my_cdo_str = 'cdo -yearmean {infile} {outfile}'
my_ens = cd.my_operator(ens, my_cdo_str, output_prefix='annual_')
```

2.Do a date selection and time mean:

```
        my_cdo_str = 'cdo sub {infile} -timmean -seldate,1991-01-01,2000-12-31 {infile} {outfile}'
        my_ens = cd.my_operator(ens, my_cdo_str, output_prefix='test_')
```

preprocessing_tools.**remap**(*ensemble*, *remap='r360x180'*, *method='remapdis'*, *delete=True*, *output_prefix=''*)

Remap files to a specified resolution.

For each file in ens, remap to resolution remap='**r_nlon_x_nlat_**', where _nlon_, _nlat_ are the number of lat-lon points to use. Removal of the original input files occurs if delete=True (default). An updated ensemble object is also returned.

By default the distance weighted remapping is used, but any valid cdo remapping method can be used by specifying the option argument 'method', e.g. method='remapdis'.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the remapping.

remap : str

The resolution to remap to, e.g. for a 1-degree grid remap='r360x180'

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory.

preprocessing_tools.**time_anomaly**(*ensemble*, *start_date*, *end_date*, *delete=False*, *output_prefix=''*)

Compute the anomaly relative the period between start_date and end_date, for each file in ens.

The resulting output is written to file with the prefix '**anomaly_**', and the original input files are deleted if delete=True.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the processing.

start_date : str

Start date for the base period with format: YYYY-MM-DD

end_date : str

End date for the base period with format: YYYY-MM-DD

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory.

Examples

1.Compute the anomaly relative to the base period 1980 to 2010:

```
ens = cd.time_anomaly(ens, start_date='1980-01-01', end_date='2010-12-31')
```

preprocessing_tools.**time_slice**(*ensemble*, *start_date*, *end_date*, *delete=True*, *output_prefix=''*)

Limit the data to the period between start_date and end_date, for each file in ens.

The resulting output is written to file, named with with the correct date range, and the original input files are deleted if delete=True.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the processing.

start_date : str

Start date for the output file with format: YYYY-MM-DD

end_date : str

End date for the output file with format: YYYY-MM-DD

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory.

Examples

1. Select data between 1 January 1980 and 31 December 2013:

```
ens = cd.time_slice(ens, start_date='1979-01-01', end_date='2013-12-31')
```

preprocessing_tools.**trends**(*ensemble*, *start_date*, *end_date*, *delete=False*)

Compute linear trends over the period between start_date and end_date, for each file in ens.

The resulting output is written to file, named with with the correct date range, and the original input files are deleted if delete=True.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the processing.

start_date : str

Start date for the output file with format: YYYY-MM-DD

end_date : str

End date for the output file with format: YYYY-MM-DD

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory,

and begin with "**slope_**" and "**intercept_**".

Examples

1.Select data between 1 January 1980 and 31 December 2013:

```
ens = cd.trends(ens, start_date='1979-01-01', end_date='2013-12-31')
```

preprocessing_tools.**zonmean**(*ensemble*, *delete=True*, *output_prefix=''*)

Calculate the zonal mean for each file in ens.

The output files are prepended with 'zonal-mean'. The original the input files are removed if delete=True (default). An updated ensemble object is also returned.

Parameters **ens** : cmipdata Ensemble

The ensemble on which to do the processing.

delete : boolean

If delete=True, delete the original input files.

Returns **ens** : cmipdata Ensemble

An updated ensemble object, containing the names of the newly processed files.

The processed files are also written to present working directory.

Examples

1.Compute the zonal mean for all files in ens:

```
zonal_mean_ens = cd.zonmean(ens)
```

3.3 loading_tools

The loading_tools module of cmipdata is a set of functions which use the cdo python bindings and NetCDF4 to load data from input NetCDF files listed in a cmipdata ensemble object into python numpy arrays. Some processing can optionally be done during the loading, specifically remapping, time-slicing, time-averaging and zonal-averaging.

loading_tools.**get_dimensions**(*ifile*, *varname*, *toDatetime=False*)

Returns the dimensions of variable varname in file ifile as a dictionary. If one of the dimensions begins with lat (Lat, Latitude and Latitudes), it will be returned with a key of lat, and similarly for lon. If toDatetime=True, the time dimension is converted to a datetime.

loading_tools.**get_models**(*files*)

loading_tools.**get_realizations**(*files*)

loading_tools.**loadfiles**(*ens*, *varname*, *toDatetime=False*, ***kwargs*)

Load a variable "varname" from all files in ens, and load it into a matrix where the zeroth dimensions represents an input file and dimensions 1 to n are the dimensions of the input variable. Variable "varname" must have the same shape in all ifiles. Keyword argument toDatetime (defaults to False) will be passed as a keyword argument to get_dimensions(). Optionally specify any kwargs valid for loadvar.

Requires netCDF4, cdo bindings and numpy

Returns dictionary with keys data and dimensions

data maps to a numpy array containing the data dimensions has keys; models, realizations,

and possibly lat, lon, and time

loading_tools.**loadvar**(*ifile*, *varname*, *cdostr=None*, ***kwargs*)

Load variables from a NetCDF file with optional pre-processing.

Load a CMIP5 netcdf variable "varname" from "ifile" and an optional cdo string for preprocessing the data from the netCDF files. Requires netCDF4, CDO and CDO python bindings. Returns a masked array, var.

CHAPTER 4

Contributors

Neil Swart, CCCma, Environment Canada: Neil.Swart@canada.ca

David Fallis: davidwfallis@gmail.com

The source code is at https://github.com/swartn/cmipdata. Pull requests and comments are welcome.

CHAPTER 5

LICENSE

See the LICENSE.txt file in the cmipdata package. cmipdata is distributed under the GNU General Public License version 2, and the Open Government License - Canada (http://data.gc.ca/eng/open-government-licence-canada)

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

c

l

p