Climate Change Analysis using Multi-Model outputs and CliMAF - Documentation

Release 1.0

Stéphane Sénési

Jul 12, 2021

Contents

Climate Change Analysis using Multi-Model outputs and CliMAF

CHAPTER 1

Contents:

1.1 Introduction

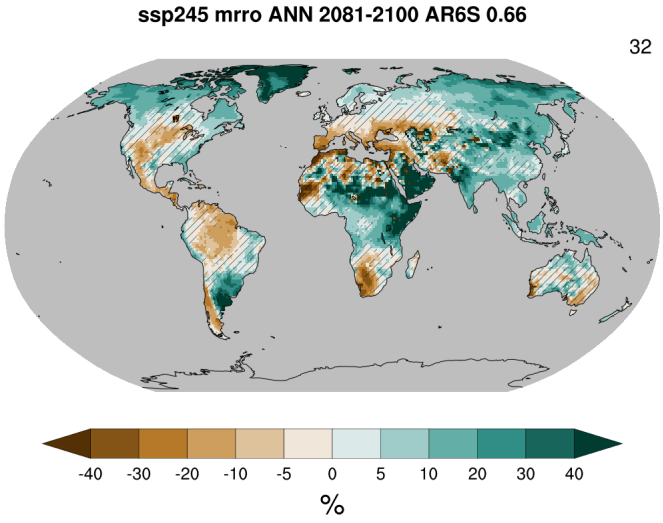
This is both the README page of CAMMAC GitHub repository, and the introduction page of the CAMMAC doc which can be reached on ReadTheDocs. However it doesn't render fully on GitHub

CAMMAC is designed for analyzing climate variables change in a CMIP6 multi-model context, assuming that CMIP6 data is locally available.

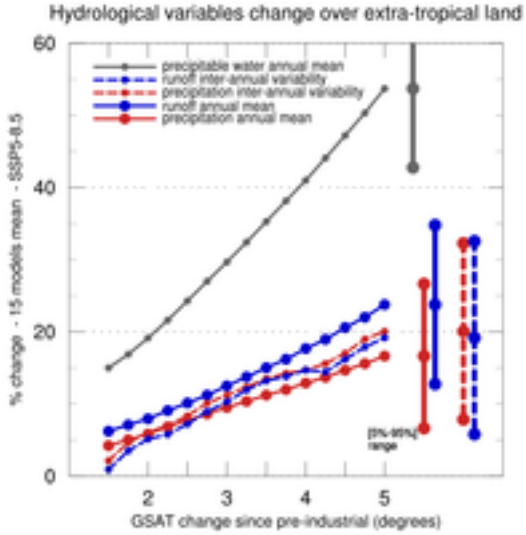
It was originally developped and used for Météo-France's contribution to IPCC Assessment Report #6 (AR6) WG1 Chapter 8 (that chapter deals with hydrological aspects of Climate Change). This is why it applies a 'one model / one vote' rule (but see *One model / one vote*)

The typical analyses allowed by CAMMAC are :

- create multi-panel **figures showing global maps of the multi-model change** for some physical variables (or derived variables) between two periods (chosen in two CMIP6 experiments), or for a given global warming level. What varies between panels can be the projection experiment, or the variable, or the season... Various *change definitions* are implemented. The maps also include hatching and or stippling for representing *confidence schemes* according to AR5 or AR6 schemes



- create **plots of the dependance of relative change with warming level** for some physical variable; the change is evaluated after averaging on basins or seasons, or even hybrid seasons (such as 'global winter', a composite of austral JJA and boreal DJF)



The workflow has three possible modes :

- launch batch jobs using e.g. one of the provided bash scripts (which is the easiest way) (see *Batch execution of notebooks*)
- interactively run one of the *provided IPython notebooks* (which provides more flexiblility)
- build your own Python program using the *CAMMAClib library*

The first mode, batch jobs, makes use of the notebooks quoted for the second mode; in both cases, one sets *a number of parameters* which allow to tune the processing.

For all three modes **a data versions dictonnary is needed**, as a basis for selecting the dataset to process (details *here*)

A few notebooks dealing with data availability, quality control and pre-processing are also provided and *described here*

While CAMMAC has been developped and used on IPSL's ESPRI <https://en.aeris-data.fr/espri-2/>'_ platform, it is fully portable and usable on any machine with CMIP6 data and e.g. conda-enabled environments (see *Sofware*

requirements)

The initial development was funded by Météo-France's CNRM, and done by Stéphane Sénési

1.2 Short cut

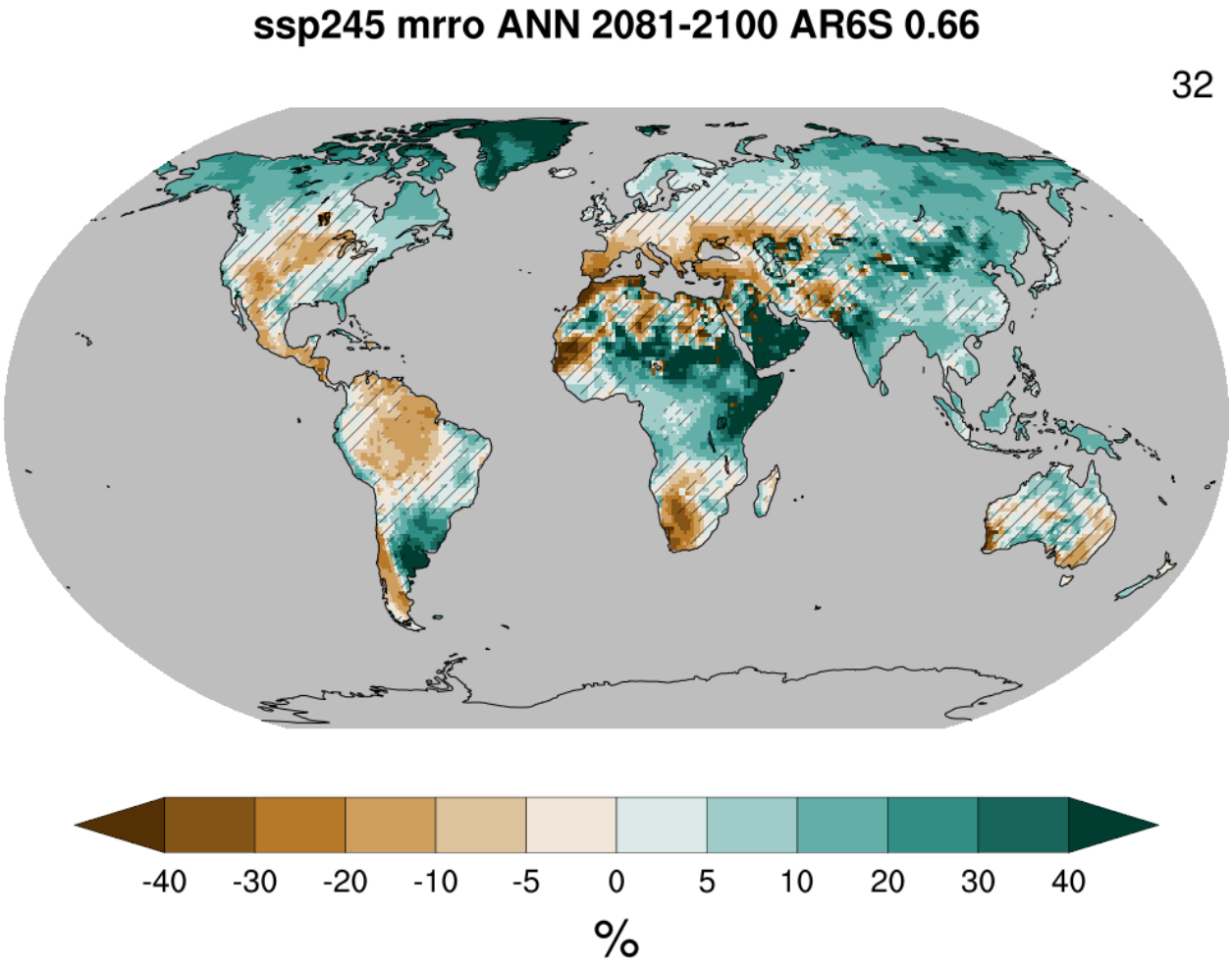
If you are in the lucky case where :

- you have access to the ESPRI platform
- that platform still hosts directory '/data/ssenesi/CAMMAC'
- you have a simple need for a change map

you can create a map of the change in variable `mrro` for season `ANN` in projection `ssp245` between periods 1995-2014 and 2081-2100, using the confidence/hatching scheme `KS` (which refers to the Knutti-Sedlacek robustness index) with threshold value 0.9 by typing :

```
export CLIMAF=/home/ssenesi/climaf_installs/climaf_running
export CAMMAC=/data/ssenesi/CAMMAC
$CAMMAC/jobs/basic.sh ssp585 JJAS KS 0.9 pr
```

This uses notebook `basic` and should create a sub-directory 'basic_mrro_ssp245_ANN_AR6S_0.66' hosting this figure :



You can also copy and edit script `basic.sh` for changing more driving parameters.

Note that parameters included in `$CAMMAC/jobs/common_parameters.yaml` are take into account, but with a lower priority than those set in the script itself

If directory '/data/ssenesi/CAMMAC' has been removed, you should get the same results by first getting CAMMAC source using only the *code cloning instructions* of the installation section

If you have other needs, have may a look at *Notebooks for figures/tables generation* and *Batch execution of notebooks*

If you don't have access to the ESPRI platform, you will need :

- to have CMIP6 data locally available
- possibly, to tell CliMAF how to access it (see *Using CAMMAC outside the ESPRI platform*)
- to go through *Installation*

1.3 CAMMAC principles for managing multi-model data

1.3.1 One model / one vote

CAMMAC has been designed for AR6 analysis, which included the rule that only one variant (or simulation) must be used per experiment and model.

This rule is implemented in function `mips_et_al.models_for_experiments()`; because this function actually returns a list of pairs (model,variant), it should be possible to change its behaviour to return the result of any other rule, and have a consistent computation downstream. This has however not be tested

1.3.2 Rationale

For using multi-model output, one needs to choose some model ensemble, which implies at some stage to make a list of available versions of data for each model. Further, in order to implement traceable science, each input dataset should be fully qualified by its metadata, which include the simulation variant index (e.g. `r1i1p1f2` for CMIP6), the version and the grid of the dataset. **The way CAMMAC handles these points is through reading a dictionnary of available data versions** (in json format). This is hereafter called a 'data versions dictionnary'

1.3.3 Building a data versions dictionnary

CAMMAC includes a notebook for helping with buiding a data versions dictionnary, `data_selection` in directory select_data_versions; it has to be launched once before invoking any computation script or notebook, and this should be redone when available datasets do change (this is actually mandatory if some dataset is withdrawn,e xcempt using notebook's parameter `exclude_models`). Helper script `$CAMMAC/jobs/select_data.sh` can be used

This notebook is further described in *the section for data related notebooks*

1.3.4 Use of data versions dictionnary by processing notebooks

Data versions dictionnaries are an input for most CAMMAC processing notebooks : they define the list of models and simulation variant they actually use by reading it and then using a CAMMAClib function for computing the intersection of available models across the experiments they are dealing with (typically an historical + a projection for assessing a change, and the control experiment for assessing the variability, if needed) for the variable they are analyzing. This also include choosing a variant when needed. Notebooks parameters also generally allows to restrict models used to an explicit list, and also allows to explicitly exclude some models; ths is handy for small-size tests

The processing notebooks also use this detailed data version info to generate the data documentation for each figure, formated according to the AR6/WGI/TSU guidelines, and stored in a file named after the figure name and with suffix "_md.txt"

The notebooks will usually insert the dictionnary 'tag' (or label) as a suffix in the generated figure filename, in order to ensure traceability.

1.3.5 Reference data versions dictionnary

A reference data versions dictionnary is provided with the software in directory *data* , and is usable for CMIP6 data on the ESPRI platform. It is based on the data available there as of february 1st, 2021; and its the one configured by default in all notebooks

At the time of writing, it can be used on the ESPRI platform, and this will carry on as long as no data is withdrawn; after some withdrawal occurs, that dictionnary will still be usable if corresponding models are indicated as excluded using the relevant notebooks parameter ('excluded_models')

1.4 Software organization, requirements and installation

1.4.1 Software organization and directory structure

The directory structure of the software is composed of 5 directories :

- **CAMMAClib/**, a library of python (2.7) modules providing shared functions, and documented *there*
- **notebooks/**, a series of parameterized IPython notebooks, which can be seen as "main programs" using CAMMAClib; each notebook was inspired by the needs of one of the multi-panel AR6/WG1/Chapter8 figures, but designed to be more generic than these needs. See the notebook_gallery
- **jobs/**, a series of "job scripts", where each script is dedicated to launch a job for actually computing one single AR6 chapter 8 figure or table (or a Technical Summary figure or panel). These jobs allow for changing notebook parameters. These "job scripts" are, at the time of writing, not yet publicly available; they will became available when the AR6/WGI report is released, in the course of 2021. A few jobs are yet provided as examples or for ancillary tasks.
- **select_data_versions/** : notebooks for handling the selection of the set of CMIP6 dataset versions used by the figure notebooks for the case of AR6 report, and for data check and pre-processing
- **data/** :
  - Data_versions_selection_20210201.json : a dictionnary of CMIP6 data versions available on the ESPRI platform on 1st february 2021, suited for use with the notebooks
  - fixed_fields/, a series of fixed fields representing the land fraction for all models used has been gathered in a specific directory, in order to alleviate for the lack of those fields for some CMIP6 experiments for some models;
  - basins/ , a series of datafiles defining AR6 monsoon regions (courtesy of Sabin Thazhe Purayil) and a file describing world hydrological basins (courtesy of B.Decharme - Decharme et al. 2019)
  - colomaps/ , the series of AR6 defined Ncl colormaps

1.4.2 Sofware requirements

CAMMAC main programs are notebooks and so need Jupyter. However, using CAMMAClib in similar python main programs is also possible

CAMMAClib and notebooks heavily rely on CliMAF with a version >= 2.0.1[1] for implementing data access, for computations (mainly using CDO behind the curtain) and for ploting figures (using Ncl behind the curtain).

Because CliMAF has a built-in knowledge of CMIP6 data organization on the ESPRI platform, a slight adaptation has to be done for using CAMMAC on other platforms (see *Using CAMMAC outside the ESPRI platform*)

The job_pm.sh utility make use of papermill for launching batch executions of notebooks, while allowing for changing their parameters.

Only quite common python packages are needed; they include numpy and xarray (and requests when using the notebook queryin the ESGF errata system). TBD : give a detailed list of actually required packages

1.4.3 Installation

Once *required softwares* are installed, installing CAMMAC is as simple as :

- cloning the github repository (which amounts to less the 40 Mbytes) by

```
mkdir <my_cammac_install>
cd <my_cammac_install>
git clone https://github.com/senesis/CAMMAC
```

- create a file similar to `jobs/job_env_ciclad.sh`, which will allow to set up the environment:
  - either automatically for jobs launched by jobs/job_pm.sh or
  - manually before launching jupyter for interactive use of CAMMAC notebooks
- change the symbolic link jobs/job_env.sh to designate that new file
- adapt the content of jobs/common_parameters.yaml, replacing all occurrences of /data/ssenesi/CAMMAC by the full path of your CAMMAC install directory
- adapt the code of `jobs/job_pm.sh` to your batch command if command 'qsub' is not available
- insert this code either in your profile or before using CAMMAC scripts or notebooks :

```
export CLIMAF=<a CliMAF directory with version > 2.0>
export CAMMAC=<my_cammac_install>/CAMMAC  # Must be a full path
```

1.5 Notebooks for data registering, control and pre-processing

These notebooks are stored in directory select_data_versions

1.5.1 Creating a data versions dictionnary

Notebook `data_selection` analyzes, for each interesting variable, each experiment of interest, which are the available models, variants and versions at the host computing/data center

This in line with what is described in *CAMMAC principles for managing multi-model data*.

This allows to provide to computing notebooks a list of all datasets available, for letting them build an ensemble. It includes checking that the data period is consistent with the definition of the experiment (or with a minimum duration for the control experiment)

[1] CliMAF 2.0.0 is OK except for using the hatching confidence scheme based on Knutti and Sedlacek robustness index, in notebool `basic`

The result is stored as a json file; such a file is provided with the software (see *Reference data versions dictionnary*); it is named after the pattern *Data_versions_selection_<some_tag>.json*; it is actually a dictionnary of dictionnaries of ... organized that way :

```
>>> data_versions[experiment][variable][table][model][variant]=(grid,version,data_
↪period)
```

In its present version, and only for performance purpose, that notebook code is slighlty dependent on data organization used on the ESPRI platform; however, its data inspection mechanism mainly relies on CliMAF data management and should work anywhere after a slight adapatation (in the first few cells : search for '/bdd')

1.5.2 Creating a 'hand-made' data versions dictionnary

Notebook `handmade_data_selection` is an example of how to create a data versions dictionnary in a hard-coded mode. However, that example is based on an old structure of such dictionnaries and would have to be slightly reworked to match the *actual structure*

1.5.3 Checking ESGF errata

Notebook `Check_errata` is intended to automatically verify a subset of those datasets that are registered in a data versions dictionnary, against the ESGF errata system.

It uses a service point of this system which, at the time of CAMMAC development, was not yet fully stabilized, and which may have change and break the logic. The errata system provides mesages which have to be manually interpreted. For helping with that, the notebook organizes its output (in printed and json format) by grouping the error messages by variable, then by severity and then by error message text.

1.5.4 Checking data ranges

Notebook `Check_ranges` prints user-chosen field statistics ot user-chose time statistics for a series of variables and experiments.

It allows to detect e.g. those models which don't use the common units. The field and time statistics are specified in CDO argument syntax, and allow to elaborate complex operations thanks to CDO operators piping syntax

1.5.5 Checking that data available on ESGF is locally available

Notebook `Check_errata` queries the ESGF for latest version for a series of variables and experiments and checks its availability on the local file system;

At the time of writing, this notebook is tune for the ESPRI computing system and makes use of the file hierarchy known on this system. It can be run automatically, e.g. using job `datasets_stat.sh`. It prints its results and send them to a list of email adresses

1.5.6 Pre-processing data for generating derived variables

Notebook `create_derived_variable` allows to create datafiles for some derived variables, for a selection of the experiments (using those datasets described in a data versions dictionnary). These derived variables are defined using CDO operators piping syntax and can have any frequency

Note: There are other, on-the-fly, ways to create derived variables; see *Variable derivation*

The default settings allow to derive the annual number of dry days and the average daily rain amount (or non-dry days), from the daily precipitation data.

In order to allow for incremental processing of numerous datasets, a setting allows to avoid recomputing already existing derived data.

The notebook produces a version of the dataset versions dictionnary which is extended with the description of the derived variables; it stores the output data at a location and with a file naming convention which is fully configurable. This information on derived variables location and organization can be provided to CAMMAC by some CliMAF call such as

```
>>> derived_variables_pattern  = "/data/ssenesi/CMIP6_derived_variables/${variable}"
>>> derived_variables_pattern += "/${variable}_${table}_${model}_${experiment}_$
↪{realization}_${grid}_${version}_${PERIOD}.nc"
>>> derived_variable_table='yr'
>>> climaf.dataloc.dataloc(project='CMIP6', organization='generic', url=derived_
↪variables_pattern, table=derived_variable_table)
```

This is actually the case, by default : the first three commands are included in (relevant) notebooks parameters setting cell, and the last one in all notebooks as described *there*.

1.6 Notebooks for figures/tables generation

This section describe figure and tables creation notebooks, while notebooks related to creating a data versions dictionnary and to data management and processing are described *there*

The figures/tables creation notebooks are stored in directory 'notebooks';they are provided with expressive names; some details of their design show below, together with a link to their html rendering.

Most notebooks include both a compute phase and a figure plot phase, which can generally be activated separately

1.6.1 Notebooks for global change maps

Next notebooks all produce a multi-panel figure, except for the first two. Excpet for the first one, they all basically call function `changes.change_figure_with_caching`, which documentation is helpful for understanding details of the computation. The plot itself is done using function `figures.change_figure()`

- `basic` : create a single map of the change in a variable between a reference period and a projection period. A hatching is superimposed, either after Knutti and Sedlacek 2013 robustness index (https://doi.org/10.1038/NCLIMATE1716) or AR6 'simple' scheme
- `change_map_simple` : same as basic, but allows for another set of hatching schemes: AR5, AR6 comprehensive and AR6 simple 'simple' scheme
- those notebooks use a single projection experiment, but multiple variable or seasons:
  - `change_map_4seasons` : compute a 4 panels-figure for one variable where each panel shows the map of the change for a season and a single SSP. Used for AR6/WGI/fig8.14
  - `change_map_1SSP_4vars` : compute a 9 panels-figure where each panel shows the map of the change of one variable and a single SSP. Used for AR6/WGI/TS7-fig1
  - `change_map_1SSP_9vars` : compute a 9 panels-figure where each panel shows the map of the change of one variable and a single SSP. Used in AR6/WGI for a preliminary TS figure

- while those one show 3 projection experiments:
  - `change_map_3SSPs_2seasons`: compute a 6 panels-figure for change for one variable and 2 seasons (columns) and 3 SSPs (rows). Used for AR6/WGI/fig 8.15
  - `change_map_3SSPs_2vars`: compute a 6 panels-figure for change for two variables (columns) and 3 SSPs (rows). Used for AR6/WGI/fig 8.17 and 8.18
  - `change_map_3SSPs_3horizons` : compute a 9 panels-figure for change for one variable, three time horizons (columns) and 3 SSPs (rows). Used in AR6/WGI for a preliminary TS figure
  - `change_map_3SSPs_plus_ref` : compute a 4 panels-figure for one variable with the reference (top left) and the change for 3 SSPs. Used for AR6/WGI/Box8.2 fig 1
- and these one deal with warming levels :
  - `change_map_1var_at_WL_1SSP_with_clim` : compute a single panel figure showing the map of a single variable change for a given warming level and a single SSP, with superimposition of a few contours of the climatology of this variable in e.g. a control experiment. Used in AR6/WGI/fig8.21.
  - `change_map_path_dependance` : compute a 6 panels-figure showing the changes in some (raw or transformed) variable at two levels of warming, for a series of projection experiments, and their diff, for 2 seasons. Used for AR6/WGI/fig 8.25

1.6.2 Notebooks for plots/tables of rate of change vs warming level

- A series of notebooks name 'change_hybrid. . . .' allow for computing changes over regions, integrated over seasons and hybrid_seasons. What is called an hybrid season here is the union of pairs (region,season), which allow to define e.g. a 'global winter' by (DFJ, northern hemisphere) + (JJA, souther hemisphere). A number of regions are knwon by keyword (globe, land, NH, SH . . .)

  The change are computed with their direct or parametric dependance to the global warming level :

  - direct dependance means that, for each desired warming level, one computes for each model which is the central year corresponding to the warming level and then what is the change for that year in that model. These change values for the same warming level are then e.g. averaged across the models.
  - parametric dependance means that, for a given set of time periods, one computes for each model, on one side the global warming which is then averaged across models, and on the other side the change value, which is also avreaged across models; this provide a parametric dependency of the change to the global warming, where the paremeter is the time period

  In the course of incremental CAMMAC development, the following redundant notebooks were successively developped :

  - notebook `change_hybrid_seasons` only implements the parametric dependance scheme (it allows to compute changes for a series of time horizons) and has a companion notebooks `change_hybrid_seasons_figure` for creating a plot of the change time series. It was used for AR6/WGI figure 8.16.
  - notebook `change_hybrid_seasons_dT` is derived form previous notebook, but implements both schemes (so, it also allows to compute changes for a series of warming levels); it was actualy tested only using the direct dependance scheme (so, for warming levels). With its companion notebook `change_hybrid_seasons_dT_figure`, it was used for producing AR6/WGI figure Box TS 12; with its other companion notebook `change_hybrid_seasons_dT_table`, which allows to filter out models that do not reach a givel warming level, it was used for producing two panels for AR6/WGI figure Box TS X f3
  - notebook `change_hybrid_seasons_must` is derived from previous notebook but can also produces results in a tabular form; it was used in AR6/WGI for producing the data for tables 8.1 and 8.2, in CSV and

text mode; it is the best basis for replacong the other computation notebooks but its output dictionnary may have a some differences with what is expected by the change_hybrid. . . _figure figure creation notebboks

- `change_rate_basins` : compute a 6-to-9-panels figure of time evolution for three ensemble-statistics (e.g. mean and two percentiles) for two variables integrated over three basins, and for three SSPs. The variables are a combination of a geopysical variable (e.g. "mrro") and a time statistics ("mean" or "std"). Few common notebooks parameters apply (see documentation in notebook itself). There are two companion Ncl scripts, automatically called for ploting the results, one for the case of three basins and two statistics (mean an standard deviation), the other for up to 9 basins and only the time mean (`change_rate_basins_1var` and `change_rate_basins_2vars`)

1.6.3 Meridional profiles of zonal means

- `change_zonal_mean` : compute a 6-panels figure of zonal mean for statistics of two variables (rows) and three SSPs (columns). The statistics are : ensemble mean and 5% percentiles, ensemble mean on land, and ensemble 5% percentiles of internal variability. Graphs have a color code matching the SSPs. There is a companion Ncl script for ploting the figure, `change_zonal_mean.ncl`, which is automaticallty called by the notebook

1.7 Common notebooks behaviour and parameters

1.7.1 Parameters setting principles

All notebooks have their parameters grouped in a single notebook cell which is the first code cell. **The parameters are generally documented in that first cell, otherwise their meaning is described hereafter.**

A further notebook cell usually supersedes the value of some parameters, provided the "do_test\ parameter is set to True in first cell (which is the default). This allows that the default behaviour is to test the notebook on a small-size problem

This first cell also has a Jupyter metadata wich name is 'parameters' and which allows papermill to supersede first cell's parameters values when it runs the notebook (such as in `batch mode`; this is actually the way it knows after which cell it should include the parameters which it is provided with

Note that the combination of both features implies that `do_test` should be set to False when launching in batch mode, except when purposely wishing to activate the test paramaters set.

1.7.2 How to include own python code in all notebooks

All notebooks execute (after parameters setting) the code they may find in a file named 'user_python_settings.py' located in the directory designated by environment variable CAMMAC_USER_PYTHON_CODE_DIR (or the current execution directory if the variable is not set). This allows to avoid code duplication among numerous notebooks. Batch launcher ensures, by default, for setting this variable to a proper value (i.e. directory jobs) for accessing this version of file 'user_python_settings.py' : (see also *Batch execution of notebooks*)

```
from climaf.api import *

# Fix some model errors
calias('CMIP6','evspsbl',scale=-1,\
    conditions={"model":["CAMS-CSM1-0","EC-Earth3","EC-Earth3-Veg","EC-Earth3-LR","EC-
↪Earth3-Veg-LR"]})
calias('CMIP6','pr',scale=1000.,conditions={"model" : "CIESM"})
```

(continues on next page)

(continued from previous page)

```
calias('CMIP6','mrso',scale=1000.,conditions={"model" : "CIESM"})
calias('CMIP6','mrsos',scale=100.,conditions={"model" : "FGOALS-f3-L"})

# Define P-E for CMIP6 variables
derive('CMIP6', 'P-E','minus','pr','evspsbl')

# Define location of fixed fields
dataloc(project='CMIP6', organization='generic',
     url=default_fixed_fields_dir+"/${variable}_${table}_${model}_*_*${grid}.nc")

# Define location of derived variables if needed (e.g. yearly stats of daily precip)
if derived_variables_pattern is not None :
            climaf.dataloc.dataloc(project='CMIP6', organization='generic',
                                   url=derived_variables_pattern, table=derived_
↪variable_table)
```

For all notebooks where it makes sense, the following applies :

1.7.3 Parameters related to inputs

- parameters `data_versions_tag` and `data_versions_dir` are mandatory; the tag is included in output filenames
- parameter `excluded_models` allows to explicitly exclude a list of models from the ensemble derived from the daat versions dictionnary; this is handy when some issue occurs with data quality; it is ususally a list od model identifiers, but for some notebooks it is a dictionnary of such lists, the dictionnary keys being variable names or experiment name
- parameter `included_models` allows to explicitly limit the to a given list of models from the ensemble derived from the data versions dictionnary; this is handy for e.g. small-size notebook tests
- parameters `derived_variables_pattern` and `derived_variables_table` allow to tell the notebook how to reach those input data which are not in the standard data directory. See *Pre-processing data for generating derived variables*

1.7.4 Main parameters related to computation

- if the figure is about one single experiment, it is designated by parameter `experiment`, otherwise by a list in parameter `experiments`
- the period of interest for the projection is provided through parameter `proj_period` which is a string matching CliMAF syntax for periods, as e.g. "1900-1910"
- if any figure panel is about a change or a reference, parameters `ref_period` and `ref_experiment` are used for the reference experiment and period
- parameter `season` allows to specify the season for averaging the variables; tested values are "DJF" and "JJA", and also, for most notebooks, "ANN" (this is documented in the notebook)
- for panels showing a map of change, parameter `field_type` allows to choose which type of change is shown : `mean_change`, `mean_rchange`, `mean_schange`, `median_change`, `median_rchange`, `median_schange; where `mean` / `median` represent the ensemble operation`, `change` represents the plain change, `rchange` represents the relative change, and `schange` represents the change standardized by the internal variability

- parameter `scheme` allows to call for superimposition on maps of confidence information (related to robustness and significance) according to three different schemes :
  - value `AR5` means method a) in AR5 report's Box 12.1, so with superimposition of :
    - hatching for locations of lack of agreement across models and
    - stippling for locations with both agreement across models and exceedance of a threshold linked to internal variability
  - values `AR6` and `AR6S` mean respectively the comprehensive and simple scheme defined for AR6 (see its Cross Chapter Box Alas 1). Parameter `sign_threshold` represenets the threshold used on the eprcent of models which should agree on sign for this method (which is by default 0.66 or 0.9 for the simple method, depending on the AR6 chapter). For method AR6, parameter `confidence_factor` represents the multiplicative factor applied to control variability for deciding a change is significant (besides sqrt(2)) (default value is 1.645 in AR method)
  - and, only for notebook basic.sh, value `KS` refers to the method of Knutti and Sedlacek 2013, which admits an additional `threshold` parameter

1.7.5 Parameters related to outputs

- `custom_plot` allows to provide a dictionnary of plot options for superseding some CAMMAC default options; it has to comply with CLiMAF plot arguments semantics. See also the documentation for figure ploting function `figures.change_figure`
- the generated figure has a filename which is either completely defined by parameter `figure_filename` (if set), or built by concatenating enough parameters information for getting non-ambiguous naming;
- for large figures, a small size figures (half size) is also generated, with suffix "_small"
- in the case of automatic figure filename, parameter `version` is systematically a suffix for the filename, and parameter `outdir` is also used
- the figure title is either set using parameter `manual_title` or automatic

1.7.6 Other parameters

- a number of notebooks use a caching mechanism (atop of CliMAF one) for final fields ; in that case, they use a directory designated by `cachedir` , which default value is "./cache"; a parameter `use_cached_proj_fields` (which defaults to True) allows to (de-)activate this feature;
- multi-model maps are based on reprojecting time averaged fields on a grid common to all models; it is designated by paramterer `common_grid` with CDO syntax (see paragraph 1.3.2 of CDO doc, and defaults to a 1° regular latlon grid. The regridding algorithm used is a CDO's conservative reamp scheme, execpt when CDO does not support the input grid for this algorithm. This is fine tuned through function `choose_regrid_option()`
- when the figure needs a computation of internal multi-decennal variability of some variable, parameter `variab_sampling_args`, which value is a dictionary, allows :
  - to possibly change the parameters for multi-decennal variability computation in AR5 and AR6 methods :
    - "shift", the length of the begin of the control period which is discarded (if there is a long enough data period for that)
    - "size", the size of the samples (in years)
    - "number", the number of samples
    - "detrend", which drives a detrending before computing time variance (defaults to True)

  - to drive some internals :
    * "compute" (True/False) : should function variability_AR5 ask CliMAF to perform the actual computation or to just return the CliMAF object representing the variability field (with a deferred computation); this parameter has no actual effect when executing a notebook
    * "house_keeping" : should intermediate fields be erased from CliMAF cache after computing variability; beacuse of the length of the data periods for control experiment, this can have a strong impact on CliMAF cache size, which can be an issue with respect to file resources; re-running a notebook with house_keeping=True can be a way to save on file ressources

- `same_model_for_var` is a logical toggle which indicates whether, when computing internal variability on an ensemble of models, the chosen ensemble should be the same that the ensemble used for computing the change; if this is the case, the list of models used for both computations is restricted to the set of models providing data for the control and the reference and the projection experiments. This parameter is not implemented in all notebooks, in which case it takes value 'False'. Implementing it is quite easy for notebooks producing chane maps by calling function `changes.change_figure_with_caching()`

All notebooks producing maps do use CAMMAClib function `change_figure_with_caching()` which embedded documentation can also be helpful

1.8 Batch execution of notebooks

The way to use notebooks for batch execution is illustrated by the set of scripts for AR6 figures and other scripts in directory $CAMMAC/jobs.

Note: In short : these scripts can be launched from any directory, they allow to change notebook's parameters, their output will go in a sub-directory named after the scrips's name, and they take care of setting the execution environment defined at install stage. You can create your own version of such scripts, see *Designing own scripts for batch*

The scripts can be launched after setting the minimal environment, e.g. :

```
export CLIMAF=/home/ssenesi/climaf_installs/climaf_running
export CAMMAC=/data/ssenesi/CAMMAC
```

1.8.1 Provided scripts for batch jobs

Unfortunately, before the AR6 report goes public, the full set of AR6 scripts cannot be disclosed to the public, so the examples provided are limited to:

- `basic.sh` which allows to create a singe-panel figure with the map of a variable change by launching notebook basic; it takes arguments for variable, projection experiment, season, hatching scheme and its threshold.
- `select_data.sh` which allows to build a data versions dictionnary by launching notebook data_selection
- `check_ranges.sh` which allows to launch notebook Chek_ranges
- `datasets_stats.sh` which allows to launch notebook Chek_ESGF_lists_on_bdd
- `pr_day_stat.sh` which allows to derive daily precipitations statistics by launching notebook create_derived_variable

1.8.2 Recommended script design

These scripts all follow the recommended structure of a job script :

- create, in current directory, a working directory named with the script basename
- prepare a file for changing notebook parameters (note : parameter `do_test` should usually be set to False, there, except if you intend to execute the cell setting parameters to test values)
- execute jobs/job_pm.sh (`download to see auto-doc at top`) which launches through qsub a job that :
  - initalize some environment variables (using by default source file jobs/job_env.sh, which has been set at *CAMMAC install phase*, or the one designated by environment varable ENV_PM)
  - include further parameter settings through a file common_parameters.yaml (either from current directory, or from the arguments, of from directory 'jobs'); these parameters have **lower priority** than the ones above
  - exports environemnt variable CAMMAC_USER_PYTHON_CODE_DIR, setting it to the 'jobs' directory if not set upstream
  - uses papermill for executing the notebooks with parameters values which supersedes the one in notebook's first cell
  - traces the execution in a notebook named with script's basename, created in the working directory;

If not requested otherwise, the figure is created in sub-directory 'figures', named with script basename

1.8.3 Yaml syntax for setting parameters in scripts

The syntax used for changing notebooks parameters in job scripts is Yaml, because papermill requests it. Here is a short primer for the correspondance between Python and Yaml syntax. You may also refer to Learn Yaml in minutes

Python	Yaml
experiments = ["ssp126", "ssp245", "ssp585"]	experiments: [ssp126, ssp245, ssp585]
variability_sampling_args = { "nyears": 20 }	**variability_sampling_args:** nyears: 20
another_dict = { "nyears": 20 }	another_dict : { "nyears": 20 }
a_number_as_a_string = "1"	a_number_as_a_string: "1"

1.9 Designing own scripts for batch

You can design your own script for launching a notebook in batch mode, taking e.g. script `basic.sh` as an example. You should adopt the *recommended structure described here*. You may need to know a bit of *Yaml syntax*

You should better have your own version of file `common_parameters.yaml`, and either put in in the directory you are launching your script from, or explicitly provide its path as 5th argument of the call to job_pm.sh. Otherwise, the file installed with CAMMAC (in dir jobs/) will be used (see `common_parameters.yaml`

Anyway, you can supersede the settings of common_parameters.yaml by similar settings in your script

Remember also that some python code from file `cammac_user_settings.py` will be executed in one of the few first cells of the notebook. By default, this file is sought in $CAMMAC/jobs/ (see `cammac_user_settings.py`. If you want some control over its content, put your own copy somewhere and set and export environment variable `CAMMAC_USER_PYTHON_CODE_DIR` accordingly

1.10 Interactive mode

The interactive execution of notebooks provides extended flexibility for testing and modifying the code. The preparation steps for an interactive session are :

- to export variables `CAMMAC` and `CLIMAF` :

```
export CAMMAC=/data/ssenesi/CAMMAC
export CLIMAF=/home/ssenesi/climaf_installs/V2.0.1
```

- to source file `jobs/job_env.sh`

```
source $CAMMAC/jobs/job_env.sh
```

- if you wish to systematically load some of your code in CAMMAC notebooks, to export environment variable `CAMMAC_USER_PYTHON_CODE_DIR` with a value set to a directory containing a suitable cammac_user_settings.py file (otherwise, the one in $CAMMAC/jobs will be used)

and then to launch jupyter e.g. on a copy of some notebook

1.11 Variable derivation

For deriving new variables to be plotted, and besides the way which involves pre-processing (see *Pre-processing data for generating derived variables*), CAMMAC includes a mechanics for use by its change compute function `changes.change_fields`, which allows on-the-fly transformation of a variable using any CliMAF operator, providing it with arguments.

This is based on a dictionnary (`changes.derivations`) defined as shown below, and which links a derivation name (e.g. `dry`) to a CliMAF operator and its arguments (e.g. `ccdo` and `:{"operator" :"yearsum -ltc, "+one_mm_per_day }`). That exemple derives the annual number of dry days from the daily precipitation values

How this information is used is explained with function `changes.change_fields`

Key value `plain` simply means : no derivation/transformation An advanced use of this mechanics also allows to compute the inter-annual variaility of some variable using key `iav`

```
one_mm_per_day="%g"%(1./(24.*3600.)) # in S.I.

derivations={
     # Default : no transformation of the variable
     "plain"         : {},

     # Annual count of number of dry days (when applied to daily precip by change_
↪fields)
     "dry"           : { "operator" : ccdo,
                         "operator_args" :{"operator" :"yearsum -ltc,"+one_mm_per_day }
↪},

     # Annual average daily rain for non-dry days (when applied to daily precip  by␣
↪change_fields)
     "drain"         : { "operator" : ccdo,
                         "operator_args" : {"operator" :"yearmean -setrtomiss,-1,"+one_
↪mm_per_day}},

     # Inter-annual variability (when applied to monthly data  by change_fields)
     "iav"           : { "post_operator" : inter_annual_variability,
```

(continues on next page)

(continued from previous page)

```
                    "post_operator_args" :{"house_keeping" : True, "compute":␣
↪True}},

    # Gini index on annual values (when applied to monthly data by change_fields)
    "gini"          : { "post_operator" : gini},

    # Walsh seasonnality index (when applied to monthly precip by change_fields)
    "seasonality"   : { "operator" : walsh_seasonality },
}
```

1.12 Advanced use : CAMMAClib

CAMMAClib modules are presented below by a summary and a link which gives access to the functions documentation and source code

These links to function-level documentation of all functions and modules in CAMMAClib are also much helpful: Functions index / Modules index

Modules :

- /lib/mod_changes
- /lib/mod_variability
- /lib/mod_figures

  A single function for plotting maps the AR6 way, possibly with hatching

- /lib/mod_mips_et_al
- /lib/mod_ancillary

  Ancillary functions for :

  - managing dictionnaries of dictionnaries of dictionnaries of . . .
  - converting acronyms to nice labels
  - handling labelbars (using ImageMagick's convert) : extract, assemble

- /lib/mod_cancillary
- /lib/mod_esgf_query

  Two functions for querying the ESGF errata service and summarizing its results

1.13 Extended use

1.13.1 Extending CAMMAC use beyond CMIP6

CAMMAC is yet validated only with data for project CMIP6, and this project name is implicit in some functions. However, a former version of CAMMAC was used to process CMIP5 data, and restoring this possibility could be considered. For other projects, with a less strict output data organization, an analysis would be necessary for assessing CAMMAC hard needs for processsing it. It would also be advisable to test the *derived_variable parameters* for reaching data in other projects

1.13.2 Using CAMMAC outside the ESPRI platform

CAMMAC was developped on the ESPRI platform. Because CliMAF has a built-in knowledge of CMIP6 data organization on that platform (and a few others), this allows to avoid describing this data oragnization in CAMMAC. For using CAMMAC on other platforms with CMIP6 data, two cases arise :

- if the data is organized according to CMIP6 Data Reference Syntax, i.e. with that kind of file structure

```
<some root>/
  <mip_era>/
    <activity_id>/
      <institution_id>/
        <source_id>/
          <experiment_id>/
            <member_id>/
              <table_id>/
                <variable_id>/
                  <grid_label>/
                    <version>/
                      <variable_id>_<table_id>_<source_id>_<experiment_id >_
↪<member_id>_<grid_label>[_<time_range>].nc
```

  the only need is to declare the CMIP6 data root directory to CliMAF, by :

```
>>> cdef('root','/my/CMIP6/data_root',project="CMIP6")
```

- otherwise, one has to duplicate the declaration of climaf module climaf.project.cmip6 with the needed changes regarding base_patterns (toward the end of the module).
- for using the data inspection notebook (see *Notebooks for data registering, control and pre-processing*), a slight change is necessary to adapt to local data organization (see *there*).

Python Module Index

Index