ESMValTool User's and Developer's Guide

Release 2.0.0

ESMValTool Development Team

Aug 05, 2020

ESMVALTOOL

Part I

Introduction

ABOUT

The Earth System Model Evaluation Tool (ESMValTool) is a community-development that aims at improving diagnosing and understanding of the causes and effects of model biases and inter-model spread. The ESMValTool is open to both users and developers encouraging open exchange of diagnostic source code and evaluation results from the Coupled Model Intercomparison Project (CMIP) ensemble. This will facilitate and improve ESM evaluation beyond the state-of-the-art and aims at supporting the activities within CMIP and at individual modelling centers. We envisage running the ESMValTool routinely on the CMIP model output utilizing observations available through the Earth System Grid Federation (ESGF) in standard formats (obs4MIPs) or made available at ESGF nodes.

The goal is to develop a benchmarking and evaluation tool that produces well-established analyses as soon as model output from CMIP simulations becomes available, e.g., at one of the central repositories of the ESGF. This is realized through standard recipes that reproduce a certain set of diagnostics and performance metrics that have demonstrated its importance in benchmarking Earth System Models (ESMs) in a paper or assessment report, such as Chapter 9 of the Intergovernmental Panel on Climate Change (IPCC) Fifth Assessment Report (AR5) (Flato et al., 2013). The expectation is that in this way a routine and systematic evaluation of model results can be made more efficient, thereby enabling scientists to focus on developing more innovative methods of analysis rather than constantly having to "reinvent the wheel".

In parallel to standardization of model output, the ESGF also hosts observations for Model Intercomparison Projects (obs4MIPs) and reanalyses data (ana4MIPs). obs4MIPs provides open access data sets of satellite data that are comparable in terms of variables, temporal and spatial frequency, and periods to CMIP model output (Taylor et al., 2012). The ESMValTool utilizes these observations and reanalyses from ana4MIPs plus additionally available observations in order to evaluate the models performance. In many diagnostics and metrics, more than one observational data set or meteorological reanalysis is used to assess uncertainties in observations.

The main idea of the ESMValTool is to provide a broad suite of diagnostics which can be performed easily when new model simulations are run. The suite of diagnostics needs to be broad enough to reflect the diversity and complexity of Earth System Models, but must also be robust enough to be run routinely or semi-operationally. In order the address these challenging objectives the ESMValTool is conceived as a framework which allows community contributions to be bound into a coherent framework.

TWO

LICENSE

The ESMValTool is released under the Apache License, version 2.0. Citation of the ESMValTool paper ("Software Documentation Paper") is kindly requested upon use, alongside with the software DOI for ESMValTool (doi:10.5281/zenodo.3401363) and ESMValCore (doi:10.5281/zenodo.3387139) and version number:

- Righi, M., Andela, B., Eyring, V., Lauer, A., Predoi, V., Schlund, M., Vegas-Regidor, J., Bock, L., Brötz, B., de Mora, L., Diblen, F., Dreyer, L., Drost, N., Earnshaw, P., Hassler, B., Koldunov, N., Little, B., Loosveldt Tomas, S., and Zimmermann, K.: Earth System Model Evaluation Tool (ESMValTool) v2.0 – technical overview, Geosci. Model Dev., 13, 1179–1199, https://doi.org/10.5194/gmd-13-1179-2020, 2020.

Besides the above citation, users are kindly asked to register any journal articles (or other scientific documents) that use the software at the ESMValTool webpage (http://www.esmvaltool.org/). Citing the Software Documentation Paper and registering your paper(s) will serve to document the scientific impact of the Software, which is of vital importance for securing future funding. You should consider this an obligation if you have taken advantage of the ESMValTool, which represents the end product of considerable effort by the development team.

WHAT ESMVALTOOL CAN DO FOR YOU

The ESMValTool applies a great variety of standard diagnostics and metrics, and produces a collection of netCDF and graphical files (plots). Thus, the tool needs a certain amount of input from the user so that it can:

- establish the correct input and output parameters and the structured workflow;

- acquire the correct data;

- execute the workflow; and

- output the desired collective data and media.

To facilitate these four steps, the user has control over the tool via two main input files: the *user configuration file* and the recipe. The configuration file sets user and site-specific parameters (like input and output paths, desired output graphical formats, logging level, etc.), whereas the recipe file sets data, preprocessing and diagnostic-specific parameters (data parameters grouped in the datasets sections, preprocessing steps for various preprocessors sections, variables' parameters and diagnostic-specific instructions grouped in the diagnostics sections). The configuration file may be used for a very large number of runs with very minimal changes since most of the parameters it sets are recyclable; the recipe file can be used for a large number of applications, since it may include as many datasets, preprocessors and diagnostics sections as the user deems useful.

Once the user configuration files and the recipe are at hand, the user can start the tool. A schematic overview of the ESMValTool workflow is depited in the figure below.

For a generalized run scenario, the tool will perform the following ordered procedures.

3.1 Data finding

- read the data requirements from the datasets section of the recipe and assemble the data request to locate the data;

- find the data using the specified root paths and DRS types in the configuration file (note the flexibility allowed by the data finder);
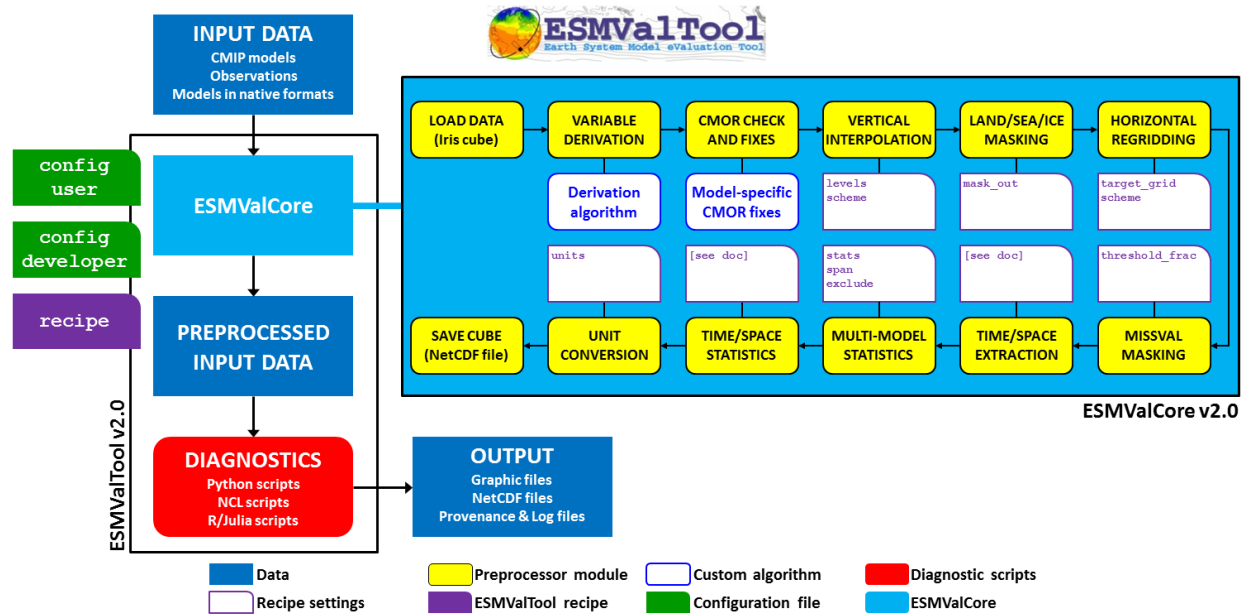
Fig. 1: Schematic of the system architecture.

3.2 Data selection

- data selection is performed using the parameters specified in the datasets section (including e.g. type of experiment, type of ensemble, time boundaries etc); data will be retrieved and selected for each variable that is specified in the diagnostics section of the recipe;

3.3 Data fixing

- the ESMValTool requires data to be in CMOR format; since errors in the data formatting are not uncommon, the ESMValTool performs checks against the CMOR library and fixes small irregularities (note that the degree of leniency is not very high).

3.4 Variable derivation

- variable derivation (in the case of non CMOR-standard variables, most likely associated with observational datasets) is performed automatically before running the preprocessor;

- if the variable definitions are already in the database then the user will just have to specify the variableto be derived in the diagnostics section (as any other standard variable, just setting `derive:  true`).

3.5 Run the preprocessor

- if any preprocessor section is specified in the recipe file, then data will be loaded in memory as iris cubes and passed through the preprocessing steps required by the user and specified in the preprocessor section, using the specific preprocessing step parameters provided by the user as keys (for the parameter name) and values (for the paramater value); the preprocessing order is very imprtant since a number of steps depend on prior excution of other steps (e.g. multimodel statistics can not be computed unless all models are on a common grid, hence a prior regridding on a common grid is necessary); the preprocessor steps order can be set by the user as custom or the default order can be used;

- once preprocessing has finished, the tool writes the data output to disk as netCDF files so that the diagnostics can pick it up and use it; the user will also be provided with a metadata file containing a summary of the preprocessing and pointers to its output. Note that writing data to disk between the preprocessing and the diagnostic phase is required to ensure multi-language support for the latter.

3.6 Run the diagnostics

- the last and most important phase can now be run: using output files from the preprocessor, the diagnostic scripts are executed using the provided diagnostics parameters.

Part II

Getting started

INSTALLATION

ESMValTool 2.0 requires a Unix(-like) operating system and Python 3.6+.

The ESMValTool supports three different installation methods:

- Installation through Conda package manager (see https://www.continuum.io);
- Deployment through a Docker container (see https://www.docker.com);
- From the source code available at https://github.com/ESMValGroup/ESMValTool.

The next sections will detail the procedure to install ESMValTool for each of this methods.

4.1 Conda installation

In order to install the Conda package, you will need both conda and Julia pre-installed, this is because Julia cannot be installed from conda. For a minimal conda installation go to https://conda.io/miniconda.html. Installation instructions for Julia can be found on the Julia download page.

Once you have installed the above prerequisites, you can install ESMValTool by running:

```
conda install esmvaltool -c esmvalgroup -c conda-forge
```

Here `conda` is the executable calling the Conda package manager to install `esmvaltool` and the `-c` flag specifies the software channels in which `esmvaltool` and its dependencies can be found.

4.1.1 Installation of subpackages

The diagnostics bundled in ESMValTool are scripts in four different programming languages: Python, NCL, R, and Julia.

There are four language specific packages available:

- `esmvaltool-julia`
- `esmvaltool-ncl`
- `esmvaltool-python`
- `esmvaltool-r`

The main `esmvaltool` package contains all four subpackages listed above.

If you only need to run a recipe with diagnostics in some of these languages, it is possible to install only the dependencies needed to do just that.

- The diagnostic script(s) used in each recipe, are documented in *Recipes*. The extension of the diagnostic script can be used to see in which language a diagnostic script is written.

- Some of the CMORization scripts are written in Python, while others are written in NCL. Therefore, both `esmvaltool-pyhon` and `esmvaltool-ncl` need to be installed in order to be able to run all CMORization scripts.

For example, to only install support for diagnostics written in Python and NCL, run

```
conda install esmvaltool-python esmvaltool-ncl -c esmvalgroup -c conda-forge
```

Note that it is only necessary to install Julia prior to the conda installation if you are going to install the `esmvaltool-julia` package.

Note that the ESMValTool source code is contained in the `esmvaltool-python` package, so this package will always be installed as a dependency if you install one or more of the packages for other languages.

4.2 Docker installation

ESMValTool is also provided through DockerHub in the form of docker containers. See https://docs.docker.com for more information about docker containers and how to run them.

You can get the latest release with

```
docker pull esmvalgroup/esmvaltool:stable
```

If you want to use the current master branch, use

```
docker pull esmvalgroup/esmvaltool:latest
```

To run a container using those images, use:

```
docker run esmvalgroup/esmvaltool:stable --help
```

Note that the container does not see the data or environmental variables available in the host by default. You can make data available with `-v /path:/path/in/container` and environmental variables with `-e VARNAME`.

For example, the following command would run a recipe

```
docker run -e HOME -v "$HOME":"$HOME" -v /data:/data esmvalgroup/esmvaltool:stable␣
→run examples/recipe_python.yml
```

with the environmental variable `$HOME` available inside the container and the data in the directories `$HOME` and `/data`, so these can be used to find the configuration file, recipe, and data.

It might be useful to define a bash alias or script to abbreviate the above command, for example

```
alias esmvaltool="docker run -e HOME -v $HOME:$HOME -v /data:/data esmvalgroup/
→esmvaltool:stable"
```

would allow using the `esmvaltool` command without even noticing that the tool is running inside a Docker container.

4.3 Singularity installation

Docker is usually forbidden in clusters due to security reasons. However, there is a more secure alternative to run containers that is usually available on them: Singularity.

Singularity can use docker containers directly from DockerHub with the following command

```
singularity run docker://esmvalgroup/esmvaltool:stable run examples/recipe_python.yml
```

Note that the container does not see the data available in the host by default. You can make host data available with `-B /path:/path/in/container`.

It might be useful to define a bash alias or script to abbreviate the above command, for example

```
alias esmvaltool="singularity run -B $HOME:$HOME -B /data:/data docker://esmvalgroup/
→esmvaltool:stable"
```

would allow using the `esmvaltool` command without even noticing that the tool is running inside a Singularity container.

Some clusters may not allow to connect to external services, in those cases you can first create a singularity image locally:

```
singularity build esmvaltool.sif docker://esmvalgroup/esmvaltool:stable
```

and then upload the image file `esmvaltool.sif` to the cluster. To run the container using the image file `esmvaltool.sif` use:

```
singularity run esmvaltool.sif run examples/recipe_python.yml
```

4.4 Install from source

4.4.1 Obtaining the source code

The ESMValTool source code is available on a public GitHub repository: https://github.com/ESMValGroup/ESMValTool

The easiest way to obtain it is to clone the repository using git (see https://git-scm.com/). To clone the public repository:

```
git clone https://github.com/ESMValGroup/ESMValTool.git
```

It is also possible to work in one of the ESMValTool private repositories, e.g.:

```
git clone https://github.com/ESMValGroup/ESMValTool-private.git
```

By default, this command will create a folder called ESMValTool containing the source code of the tool.

GitHub also allows one to download the source code in as a tar.gz or zip file. If you choose to use this option, download the compressed file and extract its contents at the desired location.

4.4.2 Prerequisites

It is recommended to use conda to manage ESMValTool dependencies. For a minimal conda installation go to https://conda.io/miniconda.html. To simplify the installation process, an environment definition file is provided in the repository (`environment.yml` in the root folder).

> **Attention:** Some systems provide a preinstalled version of conda (e.g., via the module environment). However, several users reported problems when installing NCL with such versions. It is therefore preferable to use a local, fully user-controlled conda installation. Using an older version of conda can also be a source of problems, so if you have conda installed already, make sure it is up to date by running `conda update -n base conda`.

To enable the `conda` command, please source the appropriate configuration file from your `~/.bashrc` file:

```
source <prefix>/etc/profile.d/conda.sh
```

or `~/.cshrc`/`~/.tcshrc` file:

```
source <prefix>/etc/profile.d/conda.csh
```

where `<prefix>` is the install location of your anaconda or miniconda (e.g. `/home/$USER/anaconda3` or `/home/$USER/miniconda3`).

> **Note:** Note that during the installation, conda will ask you if you want the installation to be automatically sourced from your `.bashrc` or `.bash-profile` files; if you answered yes, then conda will write bash directives to those files and everytime you get to your shell, you will automatically be inside conda's `(base)` environment. To deactivate this feature, look for the `# >>> conda initialize >>>` code block in your `.bashrc` or `.bash-profile` and comment the whole block out.

The ESMValTool conda environment file can also be used as a requirements list for those cases in which a conda installation is not possible or advisable. From now on, we will assume that the installation is going to be done through conda.

Ideally, you should create a conda environment for ESMValTool, so it is independent from any other Python tools present in the system.

Note that it is advisable to update conda to the latest version before installing ESMValTool, using the command (as mentioned above)

```
conda update --name base conda
```

To create an environment, go to the directory containing the ESMValTool source code (called ESMValTool if you did not choose a different name) and run

```
conda env create --name esmvaltool --file environment.yml
```

This installs the ESMValCore package from conda as a dependency.

The environment is called `esmvaltool` by default, but it is possible to use the option `--name SOME_ENVIRONMENT_NAME` to define a custom name. You should then activate the environment using the command:

```
conda activate esmvaltool
```

It is also possible to update an existing environment from the environment file. This may be useful when updating an older installation of ESMValTool:

```
conda env update --name esmvaltool --file environment.yml
```

but if you run into trouble, please try creating a new environment.

> **Attention:** From now on, we assume that the conda environment for ESMValTool is activated.

4.4.3 Software installation

Once all prerequisites are fulfilled, ESMValTool can be installed by running the following commands in the directory containing the ESMValTool source code (called ESMValTool if you did not choose a different name):

```
pip install -e '.[develop]'
```

If you would like to run Julia diagnostic scripts, you will also need to install Julia and the Julia dependencies:

```
esmvaltool install Julia
```

If you would like to run R diagnostic scripts, you will also need to install the R dependencies. Install the R dependency packages:

```
esmvaltool install R
```

The next step is to check that the installation works properly. To do this, run the tool with:

```
esmvaltool --help
```

If everything was installed properly, ESMValTool should have printed a help message to the console.

For a more complete installation verification, run the automated tests and confirm that no errors are reported:

```
python setup.py test
```

CONFIGURATION

The `esmvaltool` command is provided by the ESMValCore package, the documentation on configuring ESMVal-Core can be found here. In particular, it is recommended to read the section on the User configuration file and the section on Finding data.

To install the default configuration file in the default location, run

```
esmvaltool config get_config_user
```

Note that this file needs to be customized using the instructions above, so the `esmvaltool` command can find the data on your system, before it can run a recipe.

RUNNING

ESMValTool is mostly used as a command line tool. Whenever your conda environment for ESMValTool is active, you can just run the command `esmvaltool`. See running esmvaltool in the ESMValCore documentation for a short introduction.

6.1 Running a recipe

An example recipe is available in the ESMValTool installation folder as `examples/recipe_python.yml`.

This recipe finds data from CanESM2 and MPI-ESM-LR for 2000 - 2002, extracts a single level (850 hPa), regrids it to a 1x1 degree mesh and runs a diagnostic script that creates some plots of Air temperature and precipitation flux. You can download the recipe from github and save it in your project directory as (e.g.) `recipe_python.yml` and then run ESMValTool with

```
esmvaltool run recipe_python.yml --synda-download
```

The `--synda-download` option tells ESMValTool to use Synda to search for and download the necessary datasets.

ESMValTool will also find recipes that are stored in its installation directory. A copy of the example recipe is shipped with ESMValTool as: `/path/to/installation/esmvaltool/recipes/examples/recipe_python.yml`. Thus, the following also works:

```
esmvaltool run examples/recipe_python.yml
```

Note that this command does not call Synda. The required data should thus be located in the directories specified in your user configuration file. Recall that the chapter *Configuring ESMValTool* provides an explanation of how to create your own config-user.yml file.

To get help on additional commands, please use

```
esmvaltool --help
```

It is also possible to get help on specific commands, e.g.

```
esmvaltool run --help
```

will display the help message with all options for the `run` command.

6.2 Available diagnostics and metrics

See Section *Recipes* for a description of all available recipes.

To see a list of installed recipes run

```
esmvaltool recipes list
```

6.3 Running multiple recipes

It is possible to run more tha one recipe in one go: currently this relies on the user having access to a HPC that has `rose` and `cylc` installed since the procedure involves installing and submitting a Rose suite. the utility that allows you to do this is `esmvaltool/utils/rose-cylc/esmvt_rose_wrapper.py`.

6.3.1 Base suite:

The base suite to run esmvaltool via rose-cylc is *u-bd684*; you can find this suite in the Met Office Rose repository at:

https://code.metoffice.gov.uk/svn/roses-u/b/d/6/8/4/trunk/

When `rose` will be working with python3.x, this location will become default and the pipeline will aceess it independently of user, unless, of course the user will specify `-s $SUITE_LOCATION`; until then the user needs to grab a copy of it in `$HOME` or specify the default location via `-s` option.

6.3.2 Environment:

We will move to a unified and centrally-installed esmvaltool environment; until then, the user will have to alter the env_setup script:

```
u-bd684/app/esmvaltool/env_setup
```

with the correct pointers to esmvaltool installation, if desired.

To be able to submit to cylc, you need to have the */metomi/* suite in path AND use a *python2.7* environment. Use the Jasmin-example below for guidance.

6.3.3 Jasmin-example:

This shows how to interact with rose-cylc and run esmvaltool under cylc using this script:

```
export PATH=/apps/contrib/metomi/bin:$PATH
export PATH=/home/users/valeriu/miniconda2/bin:$PATH
mkdir esmvaltool_rose
cd esmvaltool_rose
cp ESMValTool/esmvaltool/utils/rose-cylc/esmvt_rose_wrapper.py .
svn checkout https://code.metoffice.gov.uk/svn/roses-u/b/d/6/8/4/trunk/ ~/u-bd684
[enter Met Office password]
[configure ~/u-bd684/rose_suite.conf]
[configure ~/u-bd684/app/esmvaltool/env_setup]
python esmvt_rose_wrapper.py -c config-user.yml \
-r recipe_autoassess_stratosphere.yml recipe_OceanPhysics.yml \
-d $HOME/esmvaltool_rose
rose suite-run u-bd684
```

Note that you need to pass FULL PATHS to cylc, no . or .. because all operations are done remotely on different nodes.

A practical actual example of running the tool can be found on JASMIN: `/home/users/valeriu/esmvaltool_rose`. There you will find the run shell: `run_example`, as well as an example how to set the configuration file. If you don't have Met Office credentials, a copy of *u-bd684* is always located in `/home/users/valeriu/roses/u-bd684` on Jasmin.

OUTPUT

ESMValTool automatically generates a new output directory with every run. The location is determined by the output_dir option in the config-user.yml file, the recipe name, and the date and time, using the the format: YYYYMMDD_HHMMSS.

For instance, a typical output location would be: output_directory/recipe_ocean_amoc_20190118_1027/

This is effectively produced by the combination: output_dir/recipe_name_YYYYMMDD_HHMMSS/

This directory will contain 4 further subdirectories:

1. *Diagnostic output* (work): A place for any diagnostic script results that are not plots, e.g. files in NetCDF format (depends on the diagnostics).

2. *Plots*: The location for all the plots, split by individual diagnostics and fields.

3. *Run*: This directory includes all log files, a copy of the recipe, a summary of the resource usage, and the *settings.yml* interface files and temporary files created by the diagnostic scripts.

4. *Preprocessed datasets* (preproc): This directory contains all the preprocessed netcdfs data and the *metadata.yml* interface files. Note that by default this directory will be deleted after each run, because most users will only need the results from the diagnostic scripts.

7.1 Preprocessed datasets

The preprocessed datasets will be stored to the preproc/ directory. Each variable in each diagnostic will have its own the *metadata.yml* interface files saved in the preproc directory.

If the option `save_intermediary_cubes` is set to `true` in the config-user.yml file, then the intermediary cubes will also be saved here. This option is set to false in the default `config-user.yml` file.

If the option `remove_preproc_dir` is set to `true` in the config-user.yml file, then the preproc directory will be deleted after the run completes. This option is set to true in the default `config-user.yml` file.

7.2 Run

The log files in the run directory are automatically generated by ESMValTool and create a record of the output messages produced by ESMValTool and they are saved in the run directory. They can be helpful for debugging or monitoring the job, but also allow a record of the job output to screen after the job has been completed.

The run directory will also contain a copy of the recipe and the *settings.yml* file, described below. The run directory is also where the diagnostics are executed, and may also contain several temporary files while diagnostics are running.

7.3 Diagnostic output

The work/ directory will contain all files that are output at the diagnostic stage. Ie, the model data is preprocessed by ESMValTool and stored in the preproc/ directory. These files are opened by the diagnostic script, then some processing is applied. Once the diagnostic level processing has been applied, the results should be saved to the work directory.

7.4 Plots

The plots directory is where diagnostics save their output figures. These plots are saved in the format requested by the option *output_file_type* in the config-user.yml file.

7.5 Settings.yml

The settings.yml file is automatically generated by ESMValCore. For each diagnostic, a unique settings.yml file will be produced.

The settings.yml file passes several global level keys to diagnostic scripts. This includes several flags from the config-user.yml file (such as 'write_netcdf', 'write_plots', etc. . .), several paths which are specific to the diagnostic being run (such as 'plot_dir' and 'run_dir') and the location on disk of the metadata.yml file (described below).

```
input_files:[[...]recipe_ocean_bgc_20190118_134855/preproc/diag_timeseries_scalars/
↪mfo/metadata.yml]
log_level: debug
output_file_type: png
plot_dir: [...]recipe_ocean_bgc_20190118_134855/plots/diag_timeseries_scalars/Scalar_
↪timeseries
profile_diagnostic: false
recipe: recipe_ocean_bgc.yml
run_dir: [...]recipe_ocean_bgc_20190118_134855/run/diag_timeseries_scalars/Scalar_
↪timeseries
script: Scalar_timeseries
version: 2.0a1
work_dir: [...]recipe_ocean_bgc_20190118_134855/work/diag_timeseries_scalars/Scalar_
↪timeseries
write_netcdf: true
write_plots: true
```

The first item in the settings file will be a list of *Metadata.yml* files. There is a metadata.yml file generated for each field in each diagnostic.

7.6 Metadata.yml

The metadata.yml files is automatically generated by ESMValTool. Along with the settings.yml file, it passes all the paths, boolean flags, and additional arguments that your diagnostic needs to know in order to run.

The metadata is loaded from cfg as a dictionary object in python diagnostics.

Here is an example metadata.yml file:

```
?
  [...]/recipe_ocean_bgc_20190118_134855/preproc/diag_timeseries_scalars/mfo/CMIP5_
↪HadGEM2-ES_Omon_historical_r1i1p1_TO0M_mfo_2002-2004.nc
  : cmor_table: CMIP5
  dataset: HadGEM2-ES
  diagnostic: diag_timeseries_scalars
  end_year: 2004
  ensemble: r1i1p1
  exp: historical
  field: TO0M
  filename: [...]recipe_ocean_bgc_20190118_134855/preproc/diag_timeseries_scalars/mfo/
↪CMIP5_HadGEM2-ES_Omon_historical_r1i1p1_TO0M_mfo_2002-2004.nc
  frequency: mon
  institute: [INPE, MOHC]
  long_name: Sea Water Transport
  mip: Omon
  modeling_realm: [ocean]
  preprocessor: prep_timeseries_scalar
  project: CMIP5
  recipe_dataset_index: 0
  short_name: mfo
  standard_name: sea_water_transport_across_line
  start_year: 2002
  units: kg s-1
  variable_group: mfo
```

As you can see, this is effectively a dictionary with several items including data paths, metadata and other information.

There are several tools available in python which are built to read and parse these files. The tools are avaialbe in the shared directory in the diagnostics directory.
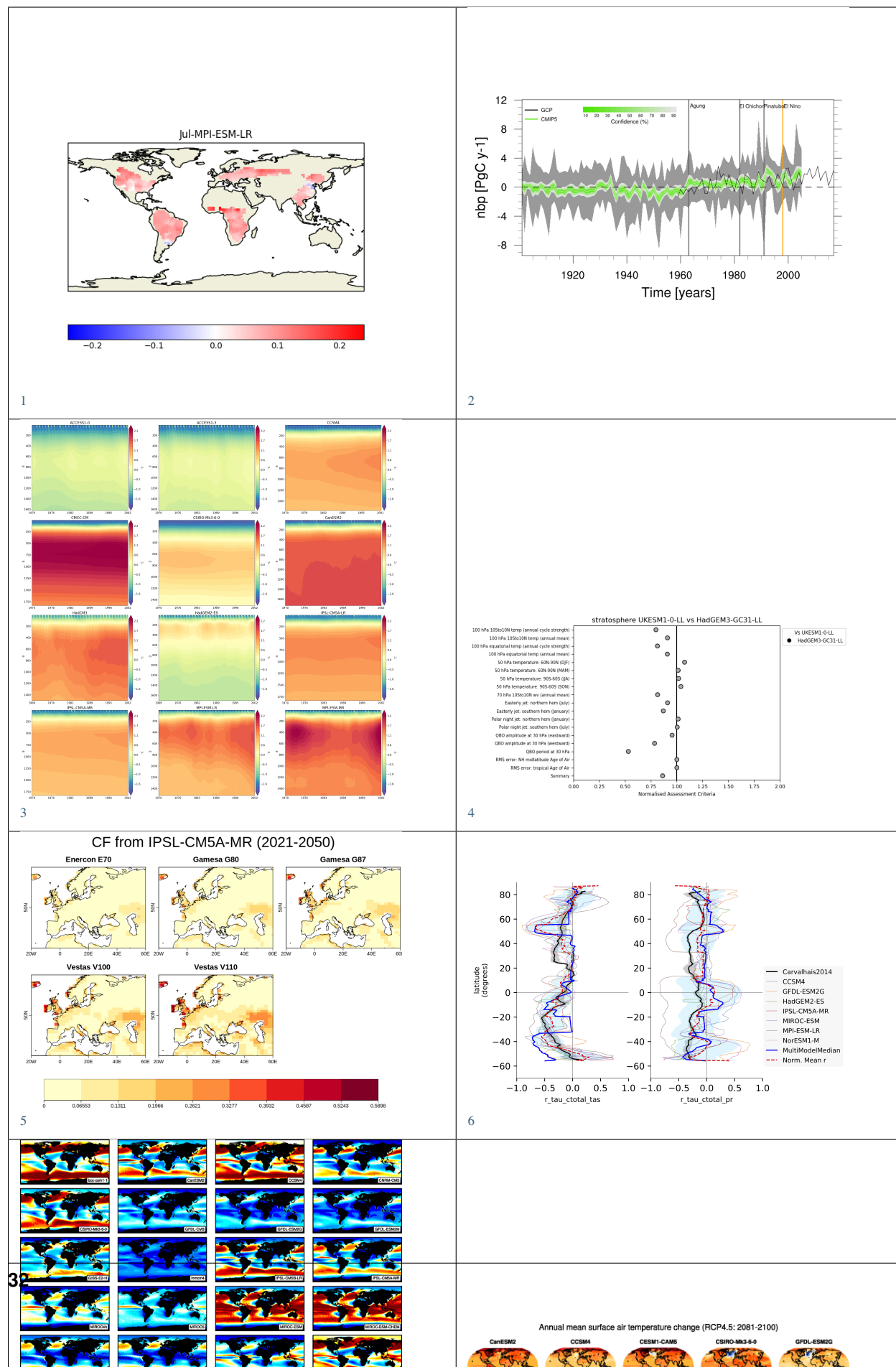
Part III

Gallery

This section shows example plots produced by ESMValTool. For more information, click on the footnote below the image.

32

[1] *Landcover - Albedo*

[2] *Land and ocean components of the global carbon cycle*

[3] *Recipe for evaluating Arctic Ocean*

[4] *Stratosphere - Autoassess diagnostics*

[5] *Capacity factor of wind power: Ratio of average estimated power to theoretical maximum power*

[6] *Turnover time of carbon over land ecosystems*

[7] *Clouds*

[8] *IPCC AR5 Chapter 12 (selected figures)*

[9] *Combined Climate Extreme Index*

[10] *Nino indices, North Atlantic Oscillation (NAO), Souther Oscillation Index (SOI)*

[11] *Consecutive dry days*

[12] *Emergent constraint on equilibrium climate sensitivity from global temperature variability*

[13] *Cloud Regime Error Metric (CREM)*

[14] *Climate Variability Diagnostics Package (CVDP)*

[15] *Evaluate water vapor short wave radiance absorption schemes of ESMs with the observations.*

[16] *Diurnal temperature range*

[17] *Equilibrium climate sensitivity*

[18] *Emergent constraints for equilibrium climate sensitivity*

[19] *Ensemble Clustering - a cluster analysis tool for climate model simulations (EnsClus)*

[20] *Extreme Events Indices (ETCCDI)*

[21] *Diagnostics of stratospheric dynamics and chemistry*

[22] *IPCC AR5 Chapter 9 (selected figures)*

[23] *Heat wave and cold wave duration*

[24] *Hydroclimatic intensity and extremes (HyInt)*

[25] *Landcover diagnostics*

[26] *Constraining future Indian Summer Monsoon projections with the present-day precipitation over the tropical western Pacific*

[27] *Drought characteristics following Martin (2018)*

[28] *Blocking metrics and indices, teleconnections and weather regimes (MiLES)*

[29] *Modes of variability*

[30] *Multi-model products*

[31] *Ocean diagnostics*

[32] *Performance metrics for essential climate parameters*

[33] *Precipitation quantile bias*

[34] *RainFARM stochastic downscaling*

[35] *Runoff, Precipitation, Evapotranspiration*

[36] *Ocean metrics*

[37] *Sea Ice*

[38] *Seaice drift*

[39] *Seaice feedback*

[40] *Shapeselect*

[41] *Single Model Performance Index (SMPI)*

[42] *Emergent constraint on snow-albedo effect*

[43] *Standardized Precipitation-Evapotranspiration Index (SPEI)*

[44] *Transient Climate Response*

[45] *Thermodynamics of the Climate System - The Diagnostic Tool TheDiaTo v1.0*

[46] *Toymodel*

[47] *Zonal and Meridional Means*

[48] *Multiple ensemble diagnostic regression (MDER) for constraining future austral jet position*

[49] *Projected land photosynthesis constrained by changes in the seasonal cycle of atmospheric CO2*

[50] *Stratosphere-troposphere coupling and annular modes indices (ZMNAM)*

Part IV

Recipes

ATMOSPHERE

8.1 Blocking metrics and indices, teleconnections and weather regimes (MiLES)

8.1.1 Overview

Atmospheric blocking is a recurrent mid-latitude weather pattern identified by a large-amplitude, quasi-stationary, long-lasting, high-pressure anomaly that "blocks" the westerly flow forcing the jet stream to split or meander (Rex, 1950).

It is typically initiated by the breaking of a Rossby wave in a diffluence region at the exit of the storm track, where it amplifies the underlying stationary ridge (Tibaldi and Molteni, 1990). Blocking occurs more frequently in the Northern Hemisphere cold season, with larger frequencies observed over the Euro-Atlantic and North Pacific sectors. Its lifetime oscillates from a few days up to several weeks (Davini et al., 2012) sometimes leading to winter cold spells or summer heat waves.

To this end, the MId-Latitude Evaluation System (MiLES) was developed as stand-alone package (https://github.com/oloapinivad/MiLES) to support analysis of mid-latitude weather patterns in terms of atmospheric blocking, teleconnections and weather regimes. The package was then implemented as recipe for ESMValTool.

The tool works on daily 500hPa geopotential height data (with data interpolated on a common 2.5x2.5 grid) and calculates the following diagnostics:

1D Atmospheric Blocking

Tibaldi and Molteni (1990) index for Northern Hemisphere. Computed at fixed latitude of 60N, with delta of -5,-2.5,0,2.5,5 deg, fiN=80N and fiS=40N. Full timeseries and climatologies are provided in NetCDF4 Zip format.

2D Atmospheric blocking

Following the index by Davini et al. (2012). It is a 2D version of Tibaldi and Molteni (1990) for Northern Hemisphere atmospheric blocking evaluating meridional gradient reversal at 500hPa. It computes both Instantaneous Blocking and Blocking Events frequency, where the latter allows the estimation of the each blocking duration. It includes also two blocking intensity indices, i.e. the Meridional Gradient Index and the Blocking Intensity index. In addition the orientation (i.e. cyclonic or anticyclonic) of the Rossby wave breaking is computed. A supplementary Instantaneous Blocking index with the GHGS2 condition (see Davini et al., 2012) is also evaluated. Full timeseries and climatologies are provided in NetCDF4 Zip format.

Z500 Empirical Orthogonal Functions

Based on SVD. The first 4 EOFs for North Atlantic (over the 90W-40E 20N-85N box) and Northern Hemisphere (20N-85N) or a custom region are computed. North Atlantic Oscillation, East Atlantic Pattern, and Arctic Oscillation can be evaluated. Figures showing linear regression of PCs on monthly Z500 are provided. PCs and eigenvectors, as well as the variances explained are provided in NetCDF4 Zip format.

North Atlantic Weather Regimes

Following k-means clustering of 500hPa geopotential height. 4 weather regimes over North Atlantic (80W-40E 30N-87.5N) are evaluated using anomalies from daily seasonal cycle. This is done retaining the first North Atlantic EOFs which explains the 80% of the variance to reduce the phase-space dimensions and then applying k-means clustering using Hartigan-Wong algorithm with k=4. Figures report patterns and frequencies of occurrence. NetCDF4 Zip data are saved. Only 4 regimes and DJF supported so far.

8.1.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_miles_block.yml
- recipe_miles_eof.yml
- recipe_miles_regimes.yml

Diagnostics are stored in diag_scripts/miles/

- miles_block.R
- miles_eof.R
- miles_regimes.R

and subroutines

- basis_functions.R
- block_figures.R
- eof_figures.R
- regimes_figures.R
- block_fast.R
- eof_fast.R
- miles_parameters.R
- regimes_fast.R

miles_parameters.R contains additional internal parameters which affect plot sizes, colortables etc.

8.1.3 User settings

1. miles_block.R

   Required settings for variables

   - reference_dataset: reference dataset for comparison

   - reference_exp: optional reference experiment for comparison (to use when comparing two experiments of the same dataset)

   Required settings for script

   - seasons: Selected season('DJF','MAM','JJA','SON','ALL') or your period as e.g. 'Jan_Feb_Mar'

2. miles_eof.R

   Required settings for variables

   - reference_dataset: reference dataset for comparison

   - reference_exp: optional reference experiment for comparison (to use when comparing two experiments of the same dataset)

   Required settings for script

   - seasons: Selected season('DJF','MAM','JJA','SON','ALL') or your period as e.g. 'Jan_Feb_Mar'

   - teles: Select EOFs ('NAO','AO','PNA') or specify custom area as "lon1_lon2_lat1_lat2"

3. miles_regimes.R

   Required settings for variables

   - reference_dataset: reference dataset

   - reference_exp: optional reference experiment for comparison (to use when comparing two experiments of the same dataset)

   Required or optional settings for script

   - None (the two parameters seasons and nclusters in the recipe should not be changed)

8.1.4 Variables

- zg (atmos, daily mean, longitude latitude time)

8.1.5 Observations and reformat scripts

- ERA-INTERIM

8.1.6 References

- REX, D. F. (1950), Blocking Action in the Middle Troposphere and its Effect upon Regional Climate. Tellus, 2: 196-211. doi: http://doi.org/10.1111/j.2153-3490.1950.tb00331.x

- Davini, P., C. Cagnazzo, S. Gualdi, and A. Navarra (2012): Bidimensional Diagnostics, Variability, and Trends of Northern Hemisphere Blocking. J. Climate, 25, 6496–6509, doi: http://doi.org/10.1175/JCLI-D-12-00032.1.

- Tibaldi S, Molteni F.: On the operational predictability of blocking. Tellus A 42(3): 343–365, doi: 10.1034/j.1600- 0870.1990.t01- 2- 00003.x, 1990. https://doi.org/10.1034/j.1600-0870.1990.t01-2-00003.x

- Paolo Davini. (2018, April 30). MiLES - Mid Latitude Evaluation System (Version v0.51). Zenodo. http://doi.org/10.5281/zenodo.1237838
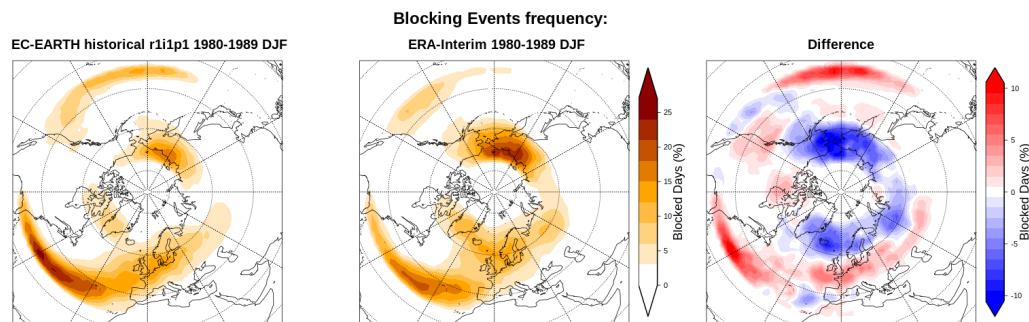
8.1.7 Example plots

Fig. 1: Blocking Events frequency for a CMIP5 EC-Earth historical run (DJF 1980-1989), compared to ERA-Interim. Units are percentage of blocked days per season.
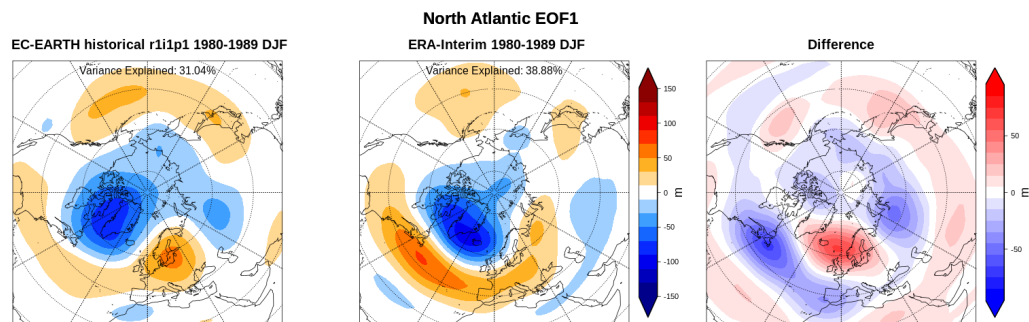

Fig. 2: North Atlantic Oscillation for a CMIP5 EC-Earth historical run (DJF 1980-1989) compared to ERA-Interim, shown as the linear regression of the monthly Z500 against the first Principal Component (PC1) of the North Atlantic region.

8.2 Clouds

8.2.1 Overview

The recipe recipe_lauer13jclim.yml computes the climatology and interannual variability of climate relevant cloud variables such as cloud radiative forcing (CRE), liquid water path (lwp), cloud amount (clt), and total precipitation (pr) reproducing some of the evaluation results of Lauer and Hamilton (2013). The recipe includes a comparison of the geographical distribution of multi-year average cloud parameters from individual models and the multi-model mean with satellite observations. Taylor diagrams are generated that show the multi-year annual or seasonal average performance of individual models and the multi-model mean in reproducing satellite observations. The diagnostic also facilitates the assessment of the bias of the multi-model mean and zonal averages of individual models compared with satellite observations. Interannual variability is estimated as the relative temporal standard deviation from multi-year timeseries of data with the temporal standard deviations calculated from monthly anomalies after subtracting the climatological mean seasonal cycle.

8.2.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_lauer13jclim.yml

Diagnostics are stored in diag_scripts/clouds/

- clouds.ncl: global maps of (multi-year) annual means including multi-model mean

- clouds_bias.ncl: global maps of the multi-model mean and the multi-model mean bias

- clouds_interannual: global maps of the interannual variability

- clouds_isccp: global maps of multi-model mean minus observations + zonal averages of individual models, multi-model mean and observations

- clouds_taylor.ncl: taylor diagrams

8.2.3 User settings in recipe

1. Script clouds.ncl

   Required settings (scripts)

   Optional settings (scripts)

   - embracesetup: true = 2 plots per line, false = 4 plots per line (default)

   - explicit_cn_levels: explicit contour levels (array)

   - extralegend: plot legend(s) to extra file(s)

   - filename_add: optionally add this string to plot filesnames

   - panel_labels: label individual panels (true, false)

   - PanelTop: manual override for "@gnsPanelTop" used by panel plot(s)

   - projection: map projection for plotting (default = "CylindricalEquidistant")

   - showdiff: calculate and plot differences model - reference (default = false)

   - rel_diff: if showdiff = true, then plot relative differences (%) (default = False)

- ref_diff_min: lower cutoff value in case of calculating relative differences (in units of input variable)

- region: show only selected geographic region given as latmin, latmax, lonmin, lonmax

- timemean: time averaging - "seasonal" = DJF, MAM, JJA, SON), "annual" = annual mean

- treat_var_as_error: treat variable as error when averaging (true, false); true: avg = sqrt(mean(var*var)), false: avg = mean(var)

Required settings (variables)

- Optional settings (variables)

- long_name: variable description

- reference_dataset: reference dataset; REQUIRED when calculating differences (showdiff = True)

- units: variable units (for labeling plot only)

Color tables

- variable "lwp": diag_scripts/shared/plot/rgb/qcm3.rgb

2. Script clouds_bias.ncl

Required settings (scripts)

Optional settings (scripts)

- plot_abs_diff: additionally also plot absolute differences (true, false)

- plot_rel_diff: additionally also plot relative differences (true, false)

- projection: map projection, e.g., Mollweide, Mercator

- timemean: time averaging, i.e. "seasonalclim" (DJF, MAM, JJA, SON), "annualclim" (annual mean)

- Required settings (variables)*

- reference_dataset: name of reference datatset

Optional settings (variables)

- long_name: description of variable

Color tables

- variable "tas": diag_scripts/shared/plot/rgb/ipcc-tas.rgb, diag_scripts/shared/plot/rgb/ipcc-tas-delta.rgb

- variable "pr-mmday": diag_scripts/shared/plots/rgb/ipcc-precip.rgb, diag_scripts/shared/plot/rgb/ipcc-precip-delta.rgb

3. Script clouds_interannual.ncl

Required settings (scripts)

Optional settings (scripts)

- colormap: e.g., WhiteBlueGreenYellowRed, rainbow

- explicit_cn_levels: use these contour levels for plotting

- extrafiles: write plots for individual models to separate files (true, false)

- projection: map projection, e.g., Mollweide, Mercator

Required settings (variables)

Optional settings (variables)

- long_name: description of variable

- reference_dataset: name of reference datatset

Color tables

- variable "lwp": diag_scripts/shared/plots/rgb/qcm3.rgb

4. Script clouds_ipcc.ncl

Required settings (scripts)

Optional settings (scripts)

- explicit_cn_levels: contour levels

- mask_ts_sea_ice: true = mask T < 272 K as sea ice (only for variable "ts"); false = no additional grid cells masked for variable "ts"

- projection: map projection, e.g., Mollweide, Mercator

- styleset: style set for zonal mean plot ("CMIP5", "DEFAULT")

- timemean: time averaging, i.e. "seasonalclim" (DJF, MAM, JJA, SON), "annualclim" (annual mean)

- valid_fraction: used for creating sea ice mask (mask_ts_sea_ice = true): fraction of valid time steps required to mask grid cell as valid data

Required settings (variables)

- reference_dataset: name of reference data set

Optional settings (variables)

- long_name: description of variable

- units: variable units

Color tables

- variables "pr", "pr-mmday": diag_scripts/shared/plot/rgb/ipcc-precip-delta.rgb

5. Script clouds_taylor.ncl

Required settings (scripts)

Optional settings (scripts)

- embracelegend: false (default) = include legend in plot, max. 2 columns with dataset names in legend; true = write extra file with legend, max. 7 dataset names per column in legend, alternative observational dataset(s) will be plotted as a red star and labeled "altern. ref. dataset" in legend (only if dataset is of class "OBS")

- estimate_obs_uncertainty: true = estimate observational uncertainties from mean values (assuming fractions of obs. RMSE from documentation of the obs data); only available for "CERES-EBAF", "MODIS", "MODIS-L3"; false = do not estimate obs. uncertainties from mean values

- filename_add: legacy feature: arbitrary string to be added to all filenames of plots and netcdf output produced (default = "")

- mask_ts_sea_ice: true = mask T < 272 K as sea ice (only for variable "ts"); false = no additional grid cells masked for variable "ts"

- styleset: "CMIP5", "DEFAULT" (if not set, clouds_taylor.ncl will create a color table and symbols for plotting)

- timemean: time averaging; annualclim (default) = 1 plot annual mean; seasonalclim = 4 plots (DJF, MAM, JJA, SON)

- valid_fraction: used for creating sea ice mask (mask_ts_sea_ice = true): fraction of valid time steps required to mask grid cell as valid data

Required settings (variables)

- reference_dataset: name of reference data set

Optional settings (variables)

8.2.4 Variables

- clwvi (atmos, monthly mean, longitude latitude time)

- clivi (atmos, monthly mean, longitude latitude time)

- clt (atmos, monthly mean, longitude latitude time)

- pr (atmos, monthly mean, longitude latitude time)

- rlut, rlutcs (atmos, monthly mean, longitude latitude time)

- rsut, rsutcs (atmos, monthly mean, longitude latitude time)

8.2.5 Observations and reformat scripts

Note: (1) obs4mips data can be used directly without any preprocessing; (2) see headers of reformat scripts for non-obs4mips data for download instructions.

- CERES-EBAF (obs4mips) - CERES TOA radiation fluxes (used for calculation of cloud forcing)

- GPCP-SG (obs4mips) - Global Precipitation Climatology Project total precipitation

- MODIS (obs4mips) - MODIS total cloud fraction

- UWisc - University of Wisconsin-Madison liquid water path climatology, based on satellite observbations from TMI, SSM/I, and AMSR-E, reference: O'Dell et al. (2008), J. Clim.

  Reformat script: reformat_scripts/obs/reformat_obs_UWisc.ncl

8.2.6 References

- Flato, G., J. Marotzke, B. Abiodun, P. Braconnot, S.C. Chou, W. Collins, P. Cox, F. Driouech, S. Emori, V. Eyring, C. Forest, P. Gleckler, E. Guilyardi, C. Jakob, V. Kattsov, C. Reason and M. Rummukainen, 2013: Evaluation of Climate Models. In: Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change [Stocker, T.F., D. Qin, G.-K. Plattner, M. Tignor, S.K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex and P.M. Midgley (eds.)]. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA.

- Lauer A., and K. Hamilton (2013), Simulating clouds with global climate models: A comparison of CMIP5 results with CMIP3 and satellite data, J. Clim., 26, 3823-3845, doi: 10.1175/JCLI-D-12-00451.1.

- O'Dell, C.W., F.J. Wentz, and R. Bennartz (2008), Cloud liquid water path from satellite-based passive microwave observations: A new climatology over the global oceans, J. Clim., 21, 1721-1739, doi:10.1175/2007JCLI1958.1.

- Pincus, R., S. Platnick, S.A. Ackerman, R.S. Hemler, Robert J. Patrick Hofmann (2012), Reconciling simulated and observed views of clouds: MODIS, ISCCP, and the limits of instrument simulators. J. Climate, 25, 4699-4720, doi: 10.1175/JCLI-D-11-00267.1.
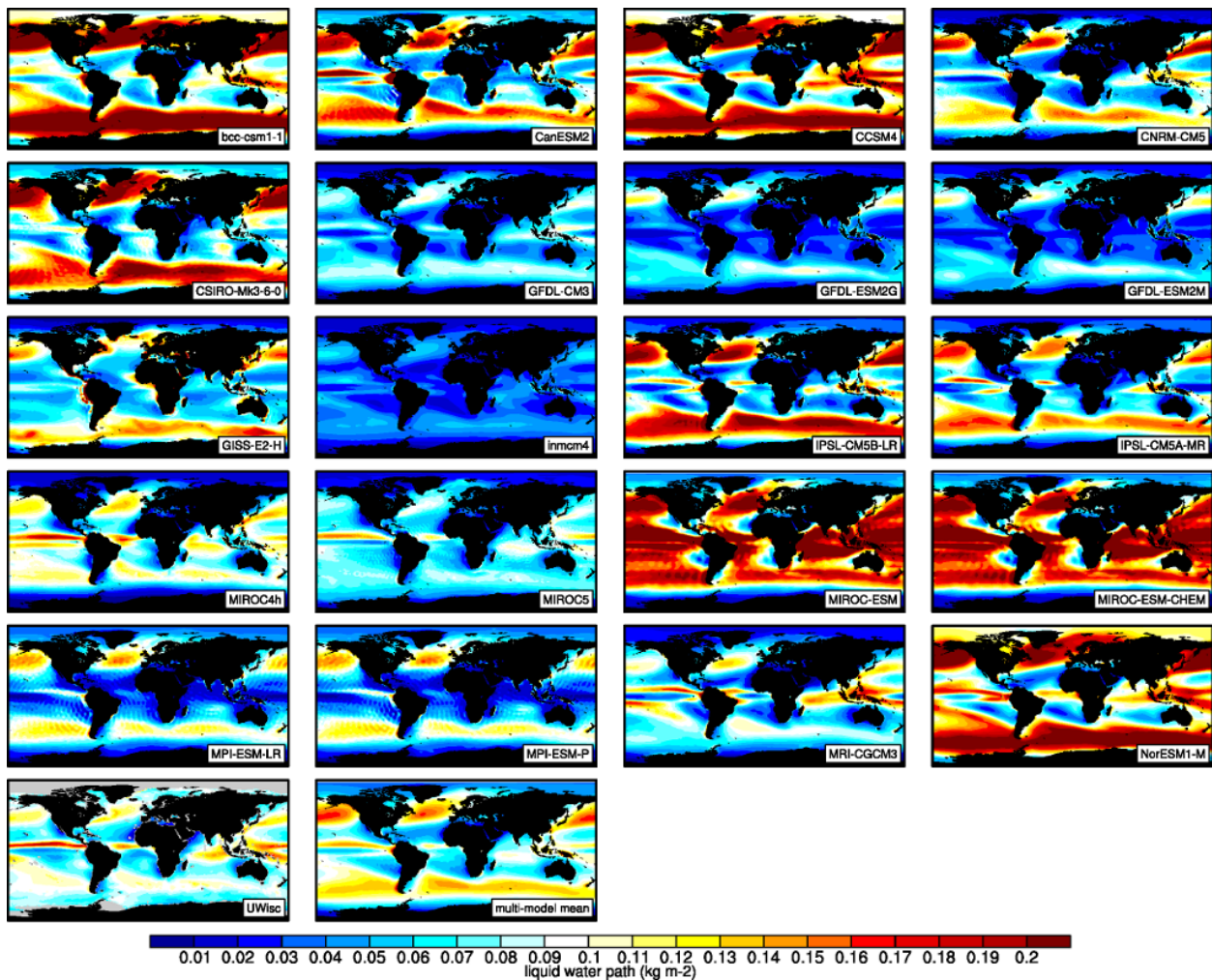
8.2.7 Example plots

Fig. 3: The 20-yr average LWP (1986-2005) from the CMIP5 historical model runs and the multi-model mean in comparison with the UWisc satellite climatology (1988-2007) based on SSM/I, TMI, and AMSR-E (O'Dell et al. 2008).
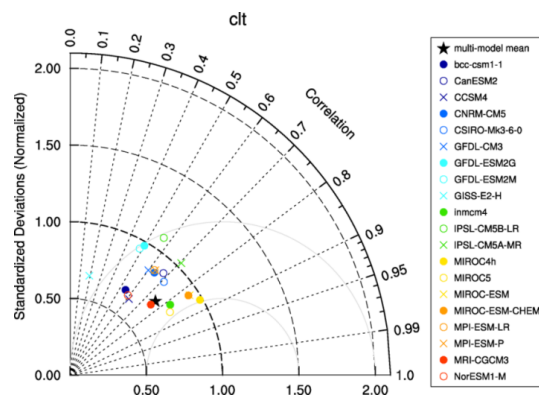
Fig. 4: Taylor diagram showing the 20-yr annual average performance of CMIP5 models for total cloud fraction as compared to MODIS satellite observations.
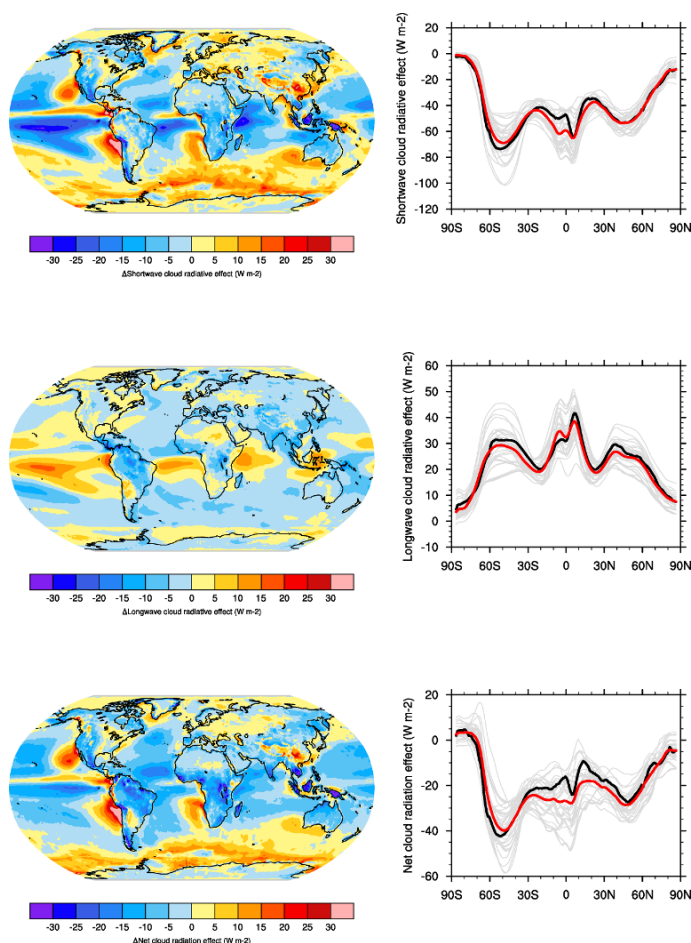

Fig. 5: 20-year average (1986-2005) annual mean cloud radiative effects of CMIP5 models against the CERES EBAF (2001–2012). Top row shows the shortwave effect; middle row the longwave effect, and bottom row the net effect. Multi-model mean biases against CERES EBAF are shown on the left, whereas the right panels show zonal averages from CERES EBAF (thick black), the individual CMIP5 models (thin gray lines) and the multi-model mean (thick red line). Similar to Figure 9.5 of Flato et al. (2013).
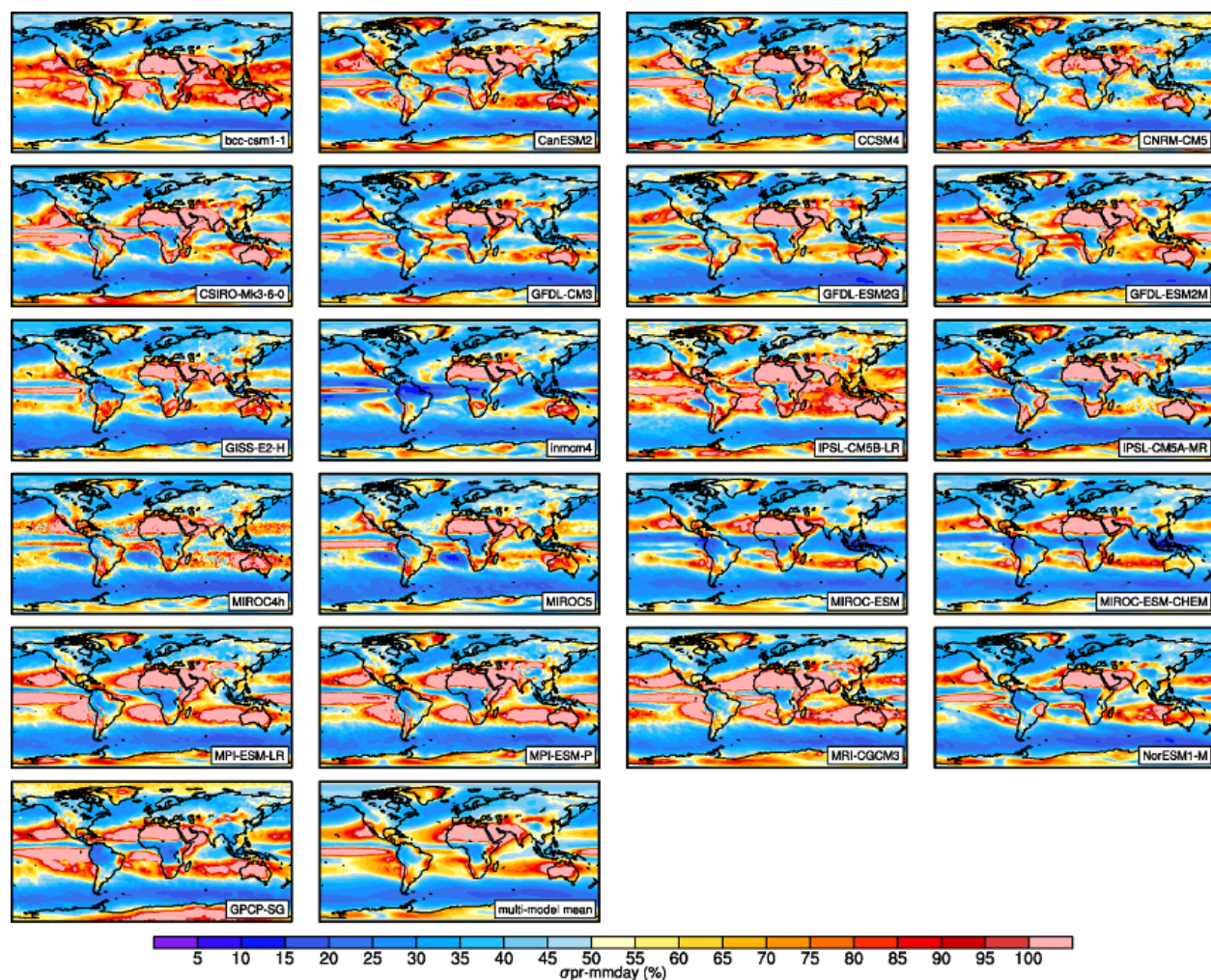
Fig. 6: Interannual variability of modeled and observed (GPCP) precipitation rates estimated as relative temporal standard deviation from 20 years (1986-2005) of data. The temporal standard devitions are calculated from monthly anomalies after subtracting the climatological mean seasonal cycle.

8.3 Cloud Regime Error Metric (CREM)

8.3.1 Overview

The radiative feedback from clouds remains the largest source of uncertainty in determining the climate sensitivity. Traditionally, cloud has been evaluated in terms of its impact on the mean top of atmosphere fluxes. However it is quite possible to achieve good performance on these criteria through compensating errors, with boundary layer clouds being too reflective but having insufficient horizontal coverage being a common example (e.g., Nam et al., 2012). Williams and Webb (2009) (WW09) propose a Cloud Regime Error Metric (CREM) which critically tests the ability of a model to simulate both the relative frequency of occurrence and the radiative properties correctly for a set of cloud regimes determined by the daily mean cloud top pressure, cloud albedo and fractional coverage at each grid-box. WW09 describe in detail how to calculate their metrics and we have included the CREMpd metric from their paper in ESMValTool, with clear references in the lodged code to tables in their paper. This has been applied to those CMIP5 models who have submitted the required diagnostics for their AMIP simulation (see Figure 8 below). As documented by WW09, a perfect score with respect to ISCCP would be zero. WW09 also compared MODIS/ERBE to ISCCP in order to provide an estimate of observational uncertainty. This was found to be 0.96 and this is marked on Figure 8, hence a model with a CREM similar to this value could be considered to have an error comparable with observational uncertainty, although it should be noted that this does not necessarily mean that the model lies within the observations for each regime. A limitation of the metric is that it requires a model to be good enough to simulate each regime. If a model is that poor that the simulated frequency of occurrence of a particular regime is zero, then a NaN will be returned from the code and a bar not plotted on the figure for that model.

8.3.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_williams09climdyn_CREM.yml

Diagnostics are stored in diag_scripts/crem/

- ww09_esmvaltool.py

8.3.3 User settings

None.

8.3.4 Variables

- albisccp (atmos, daily mean, longitude latitude time)
- cltisccp (atmos, daily mean, longitude latitude time)
- pctisccp (atmos, daily mean, longitude latitude time)
- rlut (atmos, daily mean, longitude latitude time)
- rlutcs (atmos, daily mean, longitude latitude time)
- rsut (atmos, daily mean, longitude latitude time)
- rsutcs (atmos, daily mean, longitude latitude time)
- sic/siconc (seaice, daily mean, longitude latitude time)
- snc (atmos, daily mean, longitude latitude time)

If snc is not available then snw can be used instead. For AMIP simulations, sic/siconc is often not submitted as it a boundary condition and effectively the same for every model. In this case the same daily sic data set can be used for each model.

Note: in case of using sic/siconc data from a different model (AMIP), it has to be checked by the user that the calendar definitions of all data sets are compatible, in particular whether leap days are included or not.

8.3.5 Observations and reformat scripts

All observational data have been pre-processed and included within the routine. These are ISCCP, ISCCP-FD, MODIS, ERBE. No additional observational data are required at runtime.

8.3.6 References

- Nam, C., Bony, S., Dufresne, J.-L., and Chepfer, H.: The 'too few, too bright' tropical low-cloud problem in CMIP5 models, Geophys. Res. Lett., 39, L21801, doi: 10.1029/2012GL053421, 2012.

- Williams, K.D. and Webb, M.J.: A quantitative performance assessment of cloud regimes in climate models. Clim. Dyn. 33, 141-157, doi: 10.1007/s00382-008-0443-1, 2009.
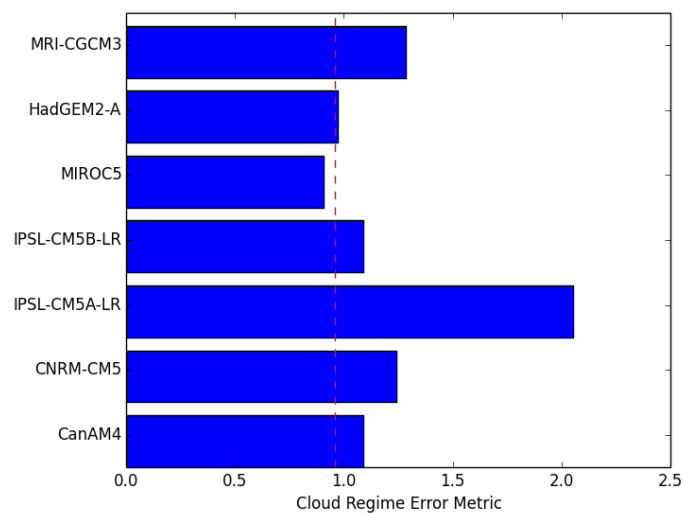
8.3.7 Example plots

Fig. 7: Cloud Regime Error Metrics (CREMpd) from William and Webb (2009) applied to those CMIP5 AMIP simulations with the required data in the archive. A perfect score with respect to ISCCP is zero; the dashed red line is an indication of observational uncertainty.

8.4 Combined Climate Extreme Index

8.4.1 Overview

The goal of this diagnostic is to compute time series of a number of extreme events: heatwave, coldwave, heavy precipitation, drought and high wind. Then, the user can combine these different components (with or without weights). The result is an index similar to the Climate Extremes Index (CEI; Karl et al., 1996), the modified CEI (mCEI; Gleason et al., 2008) or the Actuaries Climate Index (ACI; American Academy of Actuaries, 2018). The output consists of a netcdf file containing the area-weighted and multi-model multi-metric index. This recipe can be applied to data with any temporal resolution, and the running average is computed based on the user-defined window length (e.g. a window length of 5 would compute the 5-day running mean when applied to data, or 5-month running mean when applied to monthly data).

In recipe_extreme_index.yml, after defining the area and reference and projection period, the weigths for each metric selected. The options are

- weight_t90p the weight of the number of days when the maximum temperature exceeds the 90th percentile,

- weight_t10p the weight of the number of days when the minimum temperature falls below the 10th percentile,

- weight_Wx the weight of the number of days when wind power (third power of wind speed) exceeds the 90th percentile,

- weight_cdd the weight of the maximum length of a dry spell, defined as the maximum number of consecutive days when the daily precipitation is lower than 1 mm, and

- weight_rx5day the weight of the maximum precipitation accumulated during 5 consecutive days.

8.4.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_extreme_index.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- extreme_index.R

8.4.3 User settings

User setting files are stored in recipes/

1. recipe_extreme_index.yml

   Required settings for script

   - weight_t90p: 0.2 (from 0 to 1, the total sum of the weight should be 1)

   - weight_t10p: 0.2 (from 0 to 1, the total sum of the weight should be 1)

   - weight_Wx: 0.2 (from 0 to 1, the total sum of the weight should be 1)

   - weight_rx5day: 0.2 (from 0 to 1, the total sum of the weight should be 1)

   - weight_cdd: 0.2 (from 0 to 1, the total sum of the weight should be 1)

   - running_mean: 5 (depends on the length of the future projection period selected, but recommended not greater than 11)

8.4.4 Variables

- tasmax (atmos, daily, longitude, latitude, time)

- tasmin (atmos, daily, longitude, latitude, time)

- sfcWind (atmos, daily, longitude, latitude, time)

- pr (atmos, daily, longitude, latitude, time)

8.4.5 Observations and reformat scripts

None

8.4.6 References

- Alexander L.V. and Coauthors (2006). Global observed changes in daily climate extremes of temperature and precipitation. J. Geophys. Res., 111, D05109. https://doi.org/10.1029/2005JD006290

- American Academy of Actuaries, Canadian Institute of Actuaries, Casualty Actuarial Society and Society of Actuaries. Actuaries Climate Index. http://actuariesclimateindex.org (2018-10-06).

- Donat, M., and Coauthors (2013). Updated analyses of temperature and precipitation extreme indices since the beginning of the twentieth century: The HadEX2 dataset. J. Geophys. Res., 118, 2098–2118, https://doi.org/10.1002/jgrd.50150.

- Fouillet, A., Rey, G., Laurent, F., Pavillon, G. Bellec, S., Guihenneuc-Jouyaux, C., Clavel J., Jougla, E. and Hémon, D. (2006) Excess mortality related to the August 2003 heat wave in France. Int. Arch. Occup. Environ. Health, 80, 16–24. https://doi.org/10.1007/s00420-006-0089-4

- Gleason, K.L., J.H. Lawrimore, D.H. Levinson, T.R. Karl, and D.J. Karoly (2008). A Revised U.S. Climate Extremes Index. J. Climate, 21, 2124-2137 https://doi.org/10.1175/2007JCLI1883.1

- Meehl, G. A., and Coauthors (2000). An introduction to trends inextreme weather and climate events: Observations, socio-economic impacts, terrestrial ecological impacts, and model projections. Bull. Amer. Meteor. Soc., 81, 413–416. doi: 10.1175/1520-0477(2000)081<0413:AITTIE>2.3.CO;2

- Whitman, S., G. Good, E. R. Donoghue, N. Benbow, W. Y. Shou and S. X. Mou (1997). Mortality in Chicago attributed to the July 1995 heat wave. Amer. J. Public Health, 87, 1515–1518. https://doi.org/10.2105/AJPH.87.9.1515

- Zhang, Y., M. Nitschke, and P. Bi (2013). Risk factors for direct heat-related hospitalization during the 2009 Adelaide heat-wave: A case crossover study. Sci. Total Environ., 442, 1–5. https://doi.org/10.1016/j.scitotenv.2012.10.042

- Zhang, X. , Alexander, L. , Hegerl, G. C., Jones, P. , Tank, A. K., Peterson, T. C., Trewin, B. and Zwiers, F. W. (2011). Indices for monitoring changes in extremes based on daily temperature and precipitation data. WIREs Clim Change, 2: 851-870. doi:10.1002/wcc.147. https://doi.org/10.1002/wcc.147

8.4.7 Example plots

Index for t90p 2020-2040 (rcp85 MPI-ESM-MR)
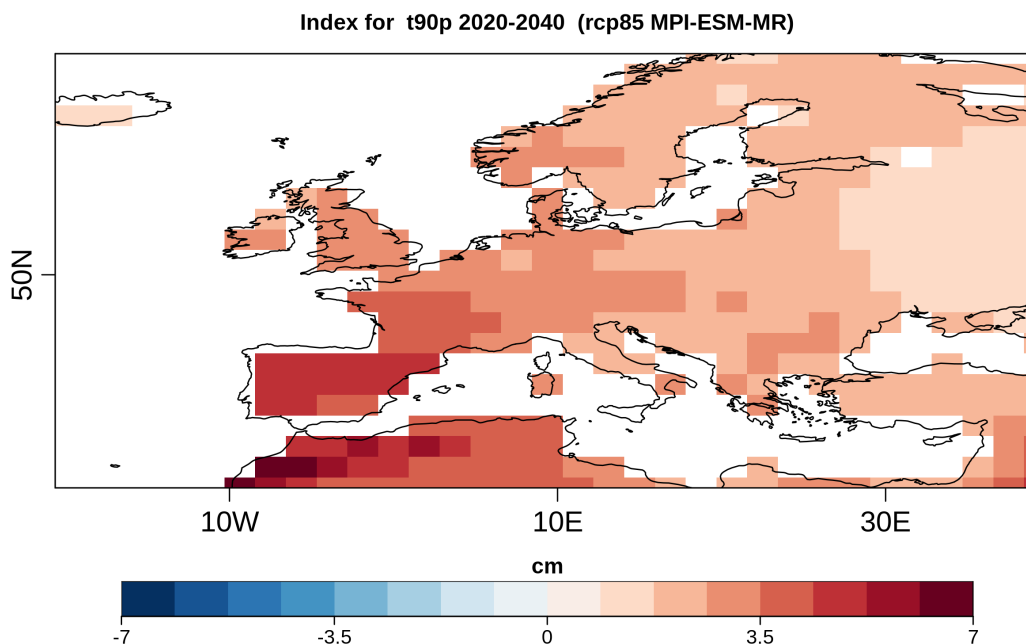
Average change in the heat component (t90p metric) of the Combined Climate Extreme Index for the 2020-2040 compared to the 1971-2000 reference period for the RCP 8.5 scenario simulated by MPI-ESM-MR.

8.5 Consecutive dry days

8.5.1 Overview

Meteorological drought can in its simplest form be described by a lack of precipitation. First, a wet day threshold is set, which can be either a limit related to measurement accuracy, or more directly process related to an amount that would break the drought. The diagnostic calculates the longest period of consecutive dry days, which is an indicator of the worst drought in the time series. Further, the diagnostic calculates the frequency of dry periods longer than a user defined number of days.

8.5.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_consecdrydays.yml

Diagnostics are stored in diag_scripts/droughtindex/

- diag_cdd.py: calculates the longest period of consecutive dry days, and the frequency of dry day periods longer than a user defined length

8.5.3 User settings in recipe

1. Script diag_cdd.py

   Required settings (script)

   - plim: limit for a day to be considered dry [mm/day]

   - frlim: the shortest number of consecutive dry days for entering statistic on frequency of dry periods.

8.5.4 Variables

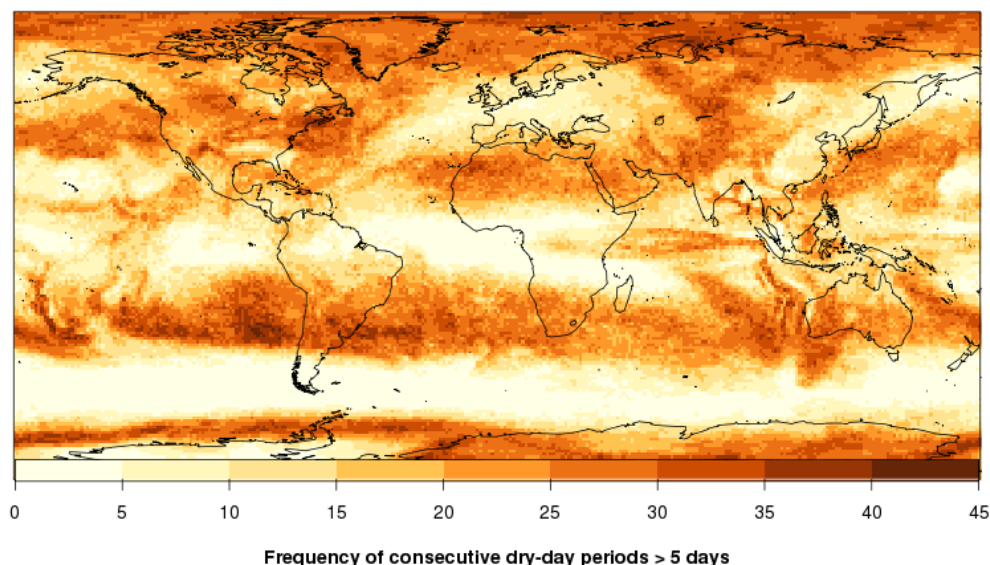- pr (atmos, daily mean, time latitude longitude)

8.5.5 Example plots

Fig. 8: Example of the number of occurrences with consecutive dry days of more than five days in the period 2001 to 2002 for the CMIP5 model bcc-csm1-1-m.

8.6 Evaluate water vapor short wave radiance absorption schemes of ESMs with the observations.

8.6.1 Overview

The recipe reproduces figures from DeAngelis et al. (2015): Figure 1b to 4 from the main part as well as extended data figure 1 and 2. This paper compares models with different schemes for water vapor short wave radiance absorption with the observations. Schemes using pseudo-k-distributions with more than 20 exponential terms show the best results.

8.6.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_deangelis15nat.yml

Diagnostics are stored in diag_scripts/

- deangelis15nat/deangelisf1b.py

- deangelis15nat/deangelisf2ext.py

- deangelis15nat/deangelisf3f4.py

8.6.3 User settings in recipe

The recipe can be run with different CMIP5 and CMIP6 models. deangelisf1b.py: Several flux variables (W m^{-2}) and up to 6 different model exeriements can be handeled. Each variable needs to be given for each model experiment. The same experiments must be given for all models. In DeAngelis et al. (2015) 150 year means are used but the recipe can handle any duration.

deangelisf2ext.py:

deangelisf3f4.py: For each model, two experiments must be given: a pre industrial control run, and a scenario with 4 times CO_2. Possibly, 150 years should be given, but shorter time series work as well.

8.6.4 Variables

deangelisf1b.py: Tested for:

- *rsnst* (atmos, monthly, longitude, latitude, time)

- *rlnst* (atmos, monthly, longitude, latitude, time)

- *lvp* (atmos, monthly, longitude, latitude, time)

- *hfss* (atmos, monthly, longitude, latitude, time)

any flux variable (W m^{-2}) should be possible.

deangelisf2ext.py:

- *rsnst* (atmos, monthly, longitude, latitude, time)

- *rlnst* (atmos, monthly, longitude, latitude, time)

- *rsnstcs* (atmos, monthly, longitude, latitude, time)

- *rlnstcs* (atmos, monthly, longitude, latitude, time)

- *lvp* (atmos, monthly, longitude, latitude, time)

- *hfss* (atmos, monthly, longitude, latitude, time)

- *tas* (atmos, monthly, longitude, latitude, time)

deangelisf3f4.py: * *rsnstcs* (atmos, monthly, longitude, latitude, time) * *rsnstcsnorm* (atmos, monthly, longitude, latitude, time) * *prw* (atmos, monthly, longitude, latitude, time) * *tas* (atmos, monthly, longitude, latitude, time)

8.6.5 Observations and reformat scripts

deangelisf1b.py: * None

deangelisf2ext.py: * None

deangelisf3f4.py:

- *rsnstcs*: CERES-EBAF
- *prw* ERA-Interim, SSMI

8.6.6 References

- DeAngelis, A. M., Qu, X., Zelinka, M. D., and Hall, A.: An observational radiative constraint on hydrologic cycle intensification, Nature, 528, 249, 2015.
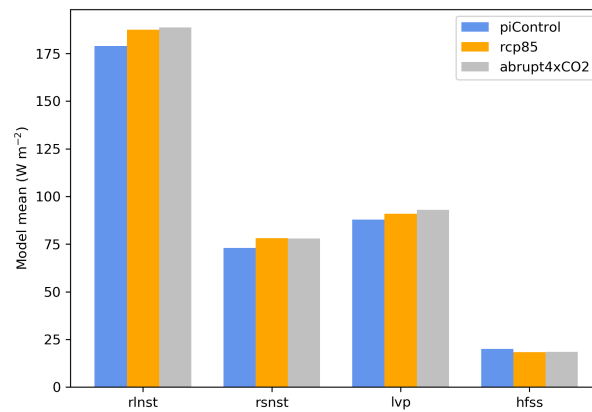
8.6.7 Example plots

Fig. 9: Global average multi-model mean comparing different model experiments for the sum of upward long wave flux at TOA and net downward long wave flux at the surface (rlnst), heating from short wave absorption (rsnst), latent heat release from precipitation (lvp), and sensible heat flux (hfss). The panel shows three model experiments, namely the pre-industrial control simulation averaged over 150 years (blue), the RCP8.5 scenario averaged over 2091-2100 (orange) and the abrupt quadrupled CO_2 scenario averaged over the years 141-150 after CO_2 quadrupling in all models except CNRM-CM5-2 and IPSL-CM5A-MR, where the average is calculated over the years 131-140 (gray). The figure shows that energy sources and sinks readjust in reply to an increase in greenhouse gases, leading to a decrease in the sensible heat flux and an increase in the other fluxes.
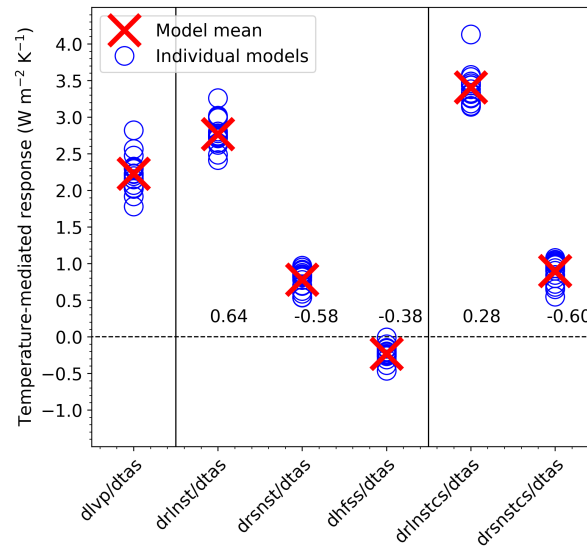
Fig. 10: The temperature-mediated response of each atmospheric energy budget term for each model as blue circles and the model mean as a red cross. The numbers above the abscissa are the cross-model correlations between dlvp/dtas and each other temperature-mediated response.'
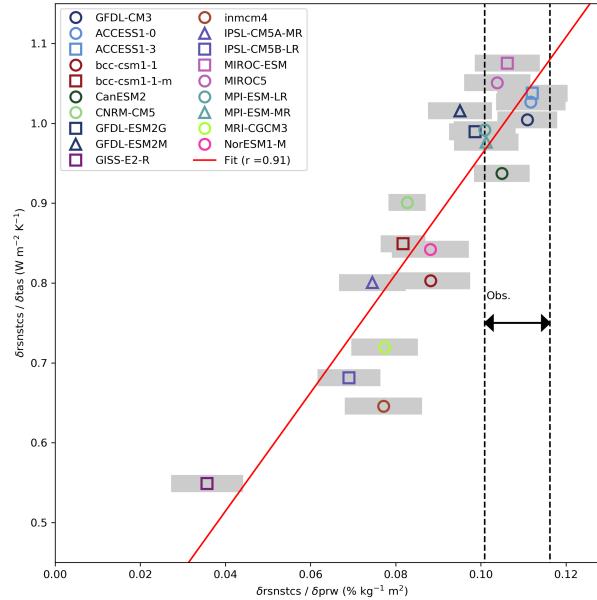

Fig. 11: Scatter plot and regression line the between the ratio of the change of net short wave radiation (rsnst) and the change of the Water Vapor Path (prw) against the ratio of the change of netshort wave radiation for clear skye (rsnstcs) and the the change of surface temperature (tas). The width of horizontal shading for models and the vertical dashed lines for observations (Obs.) represent statistical uncertainties of the ratio, as the 95% confidence interval (CI) of the regression slope to the rsnst versus prw curve. For the observations the minimum of the lower bounds of all CIs to the maximum of the upper bounds of all CIs is shown.

8.7 Diurnal temperature range

8.7.1 Overview

The goal of this diagnostic is to compute a vulnerability indicator for the diurnal temperature range (DTR); the maximum variation in temperature within a period of 24 hours at a given location. This indicator was first proposed by the energy sector, to identify locations which may experience increased diurnal temperature variation in the future, which would put additional stress on the operational management of district heating systems. This indicator was defined as the DTR exceeding 5 degrees celsius at a given location and day of the year (Deandreis et al., N.D.). Projections of this indicator currently present high uncertainties, uncertainties associated to both Tmax and Tmin in future climate projections.

As well as being of use to the energy sector, the global-average DTR has been evaluated using both observations and climate model simulations (Braganza et. al., 2004) and changes in the mean and variability of the DTR have been shown to have a wide range of impacts on society, such as on the transmission of diseases (Lambrechts et al., 2011; Paaijmans et al., 2010).

The recipe recipe_diurnal_temperature_index.yml computes first a mean DTR for a reference period using historical simulations and then, the number of days when the DTR from the future climate projections exceeds that of the reference period by 5 degrees or more. The user can define both the reference and projection periods, and the region to be considered. The output produced by this recipe consists of a four panel plot showing the maps of the projected mean DTR indicator for each season and a netcdf file containing the corresponding data.

8.7.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_diurnal_temperature_index.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- diurnal_temp_index.R : calculates the diaurnal temperature vulnerability index.

8.7.3 User settings

User setting files are stored in recipes/

1. recipe_diurnal_temperature_index.yml

   Required settings for script

   - None

8.7.4 Variables

- tasmin and tasmax (atmos, daily, longitude, latitude, time)

8.7.5 Observations and reformat scripts

None

8.7.6 References

- Amiri, S. (2013). Economic and Environmental Benefits of CHP-based District Heating Systems in Sweden. Retrieved from http://www.sgc.se/ckfinder/userfiles/files/sokmotor/LiU67.pdf

- Braganza, K., Karoly, D. J., & Arblaster, J. M. (2004). Diurnal temperature range as an index of global climate change during the twentieth century. Geophysical Research Letters, 31(13), n/a – n/a. https://doi.org/10.1029/2004GL019998

- Déandreis C. (IPSL), Braconnot P. (IPSL), Planton S. (CNRMGAME). Study performed for the DALKIA company. http://secif.ipsl.fr/images/SECIF/documents/Communication/fiche_invulnerable/RC_indicateur_EN.pdf

- Lambrechts, L., Paaijmans, K. P., Fansiri, T., Carrington, L. B., Kramer, L. D., Thomas, M. B., & Scott, T. W. (2011). Impact of daily temperature fluctuations on dengue virus transmission by Aedes aegypti. Proceedings of the National Academy of Sciences of the United States of America, 108(18), 7460–7465. https://doi.org/10.1073/pnas.1101377108

- Paaijmans, K. P., Blanford, S., Bell, A. S., Blanford, J. I., Read, A. F., & Thomas, M. B. (2010). Influence of climate on malaria transmission depends on daily temperature variation. Proceedings of the National Academy of Sciences of the United States of America, 107(34), 15135–15139. https://doi.org/10.1073/pnas.1006422107

- Kalnay, E., & Cai, M. (2003). Impact of urbanization and land-use change on climate. Nature, 423(6939), 528–531. https://doi.org/10.1038/nature01675

- Thyholt, M., & Hestnes, A. G. (2008). Heat supply to low-energy buildings in district heating areas: Analyses of CO2 emissions and electricity supply security. Energy and Buildings, 40(2), 131–139. https://doi.org/10.1016/J.ENBUILD.2007.01.016

8.7.7 Example plots

Mean number of days exceeding the Diurnal Temperature Range (DTR) simulated during the historical period (1961-1990) by 5 degrees during the period 2030-2080. The result is derived from one RCP 8.5 scenario simulated by MPI-ESM-MR.

8.8 Extreme Events Indices (ETCCDI)

8.8.1 Overview

This diagnostic uses the standard climdex.pcic.ncdf R library to compute the 27 climate change indices specified by the joint CCl/CLIVAR/JCOMM Expert Team (ET) on Climate Change Detection and Indices http://etccdi.pacificclimate.org/. The needed input fields are daily average precipitation flux and minimum, maximum and average daily surface temperatures. The recipe reproduces panels of figure 9.37 of the IPCC AR5 report, producing both a Gleckler plot, with relative error metrics for the CMIP5 temperature and precipitation extreme indices, and timeseries plots comparing the ensemble spread with observations. For plotting 1 to 4 observational reference datasets are supported. If no observational reference datasets are given, the plotting routines do not work, however, index generation without plotting is still possible. All datasets are regridded to a common grid and considered only over land.

Number of days exceeding the DTR by 5 degrees during the period 2030 - 2080
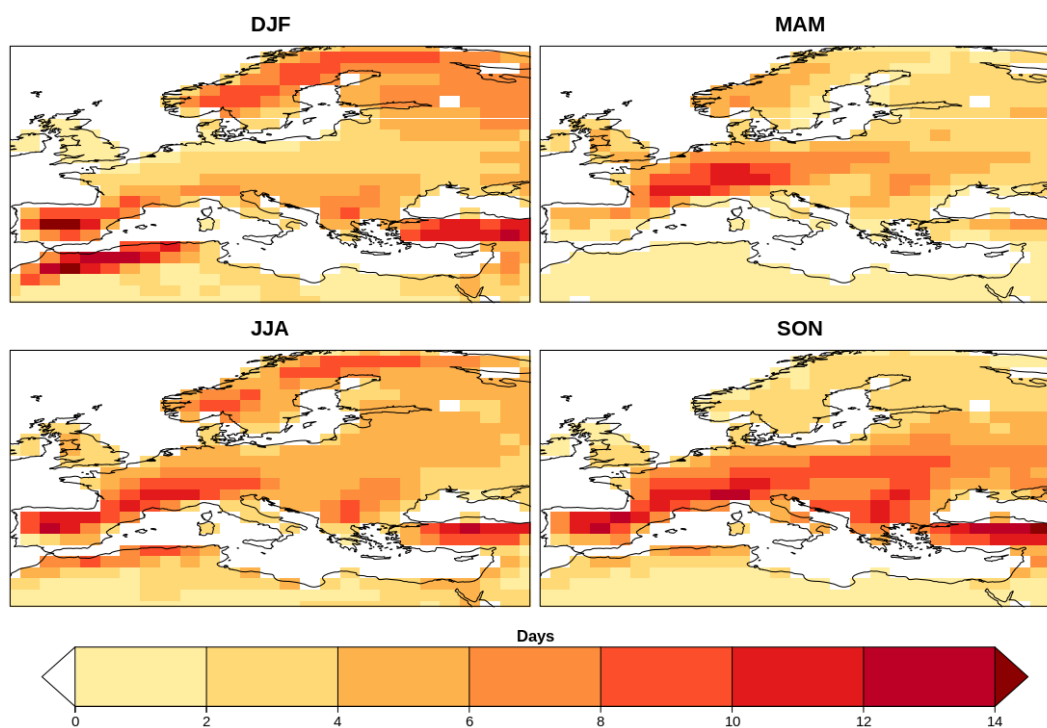
8.8.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_extreme_events.yml

Diagnostics are stored in diag_scripts/extreme_events/

- ExtremeEvents.r

and subroutines

- common_climdex_preprocessing_for_plots.r
- make_Glecker_plot2.r
- make_timeseries_plot.r
- cfg_climdex.r
- cfg_extreme.r

8.8.3 User settings

Required settings for script

- reference_datasets: list containing the reference datasets to compare with

- timeseries_idx: list of indices to compute for timeseries plot. The syntax is "XXXETCCDI_TT", where "TT" can be either "yr" or "mon" (yearly or monthly indices are computed) and "XXX" can be one of the following: "altcdd", "altcsdi", "altcwd", "altwsdi", "cdd", "csdi", "cwd", "dtr", "fd", "gsl", "id", "prcptot", "r10mm", "r1mm", "r20mm", "r95p", "r99p", "rx1day", "rx5day", "sdii", "su", "tn10p", "tn90p", "tnn", "tnx", "tr", "tx10p", "tx90p", "txn", "txx", "wsdi". The option "mon" for "TT" can be only used in combination with one of: "txx", "tnx", "txn", "tnn", tn10p", "tx10p", "tn90p", "tx90p", "dtr", "rx1day", "rx5day".

- gleckler_idx: list of indices to compute for Gleckler plot. Same syntax as above. The diagnostic computes all unique indices specified in either `gleckler_idx` or `timeseries_idx`. If at least one "mon" index is selected, the indices are computed but no plots are produced.

- base_range: a list of two years to specify the range to be used as "base range" for climdex (the period in which for example reference percentiles are computed)

Optional settings for script

- regrid_dataset: name of dataset to be used as common target for regridding. If missing the first reference dataset is used

- mip_name: string containing the name of the model ensemble, used for titles and labels in the plots (default: "CMIP")

- analysis_range: a list of two years to specify the range to be used for the analysis in the plots. The input data will need to cover both `analysis_range` and `base_range`. If missing the full period covered by the input datasets will be used.

- ts_plt: (logical) if to produce the timeseries or not (default: true)

- glc_plt: (logical) if to produce the Gleckler or not (default: true)

- climdex_parallel: number of parallel threads to be used for climdex calculation (default: 4). Also the logical `false` can be passed to switch off parallel computation.

- normalize: (logical) if to detrend and normalize with the standard deviation for the datasets for use in the timeseries plot. When this option is used the data for the following indices are detrended and normalized in the timeseries plots: "altcdd", "altcsdi", "altcwd", "altwsdi", "cdd", "cwd","dtr", "fd", "gsl", "id", "prcptot", "r10mm", "r1mm", "r20mm", "r95p", "r99p", "rx1day", "rx5day", "sdii", "su", "tnn", "tnx", "tr", "txn","txn","txx" (default: false)

Additional optional setting controlling the plots:

- Timeseries plots:

  - ts_png_width: width for png figures (dafult: 640)

  - ts_png_height: height for png figures (default: 480)

  - ts_png_units: units for figure size (default: "px")

  - ts_png_pointsize: fontsize (default: 12)

  - ts_png_bg: background color (default: "white")

  - ts_col_list: list of colors for lines (default: ["dodgerblue2", "darkgreen", "firebrick2", "darkorchid", "aquamarine3"]``)

  - ts_lty_list: list of linetypes (default: [1, 4, 2, 3, 5])

  - ts_lwd_list: list of linewidths (default: [2, 2, 2, 2, 2])

- Gleckler plot:

    - gl_png_res: height for png figures (default: 480). The width of the figure is computed automatically.

    - gl_png_units: units for figure size (default: "px")

    - gl_png_pointsize: fontsize (default: 12)

    - gl_png_bg: background color (default: "white")

    - gl_mar_par: page margins vector (default: [10, 4, 3, 14])

    - gl_rmsespacer: spacing of RMSE column (default: 0.01)

    - gl_scaling_factor: scaling factor for colorscale height (default: 0.9)

    - gl_text_scaling_factor: scaling factor for text size (default: 1.0)

    - gl_xscale_spacer_rmse: horizontal posizion of coloured colorbar (default: 0.05)

    - gl_xscale_spacer_rmsestd: horizontal posizion of gray colorbar (default: 0.05)

    - gl_symb_scaling_factor: scaling factor for white "symbol" square explaining the partition (default: 1.0)

    - gl_symb_xshift: horizontal position of the symbol box (default: 0.2)

    - gl_symb_yshift: vertical position of the symbol box (default: 0.275)

    - gl_text_symb_scaling_factor: scaling factor for text to be used for symbol box (default: 0.5)

8.8.4 Variables

- tas (atmos, daily mean, longitude latitude time)

- tasmin (atmos, daily minimum, longitude latitude time)

- tasmax (atmos, daily maximum, longitude latitude time)

- pr (atmos, daily mean, longitude latitude time)

8.8.5 Observations and reformat scripts

None.

8.8.6 References

- Zhang, X., Alexander, L., Hegerl, G. C., Jones, P., Klein Tank, A., Peterson, T. C., Trewin, B., Zwiers, F. W., Indices for monitoring changes in extremes based on daily temperature and precipitation data, WIREs Clim. Change, doi:10.1002/wcc.147, 2011

- Sillmann, J., V. V. Kharin, X. Zhang, and F. W. Zwiers, Climate extreme indices in the CMIP5 multi-model ensemble. Part 1: Model evaluation in the present climate. J. Geophys. Res., doi:10.1029/2012JD018390, 2013
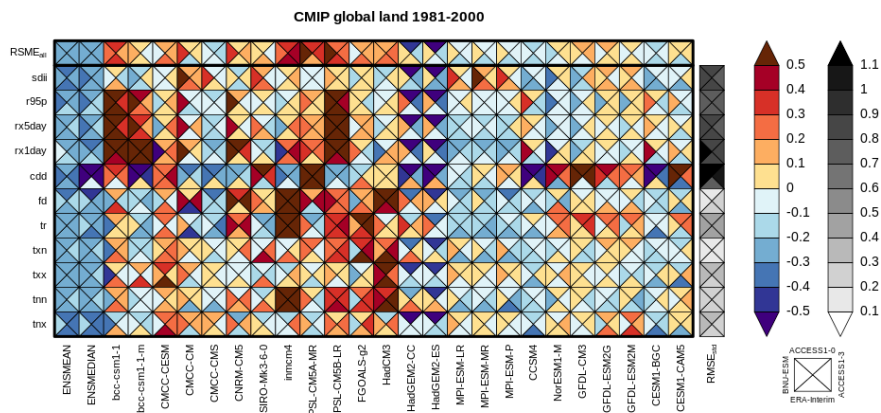
8.8.7 Example plots

Fig. 12: Portrait plot of relative error metrics for the CMIP5 temperature and precipitation extreme indices evaluated over 1981-2000. Reproduces Fig. 9.37 of the IPCC AR5 report, Chapter 9.
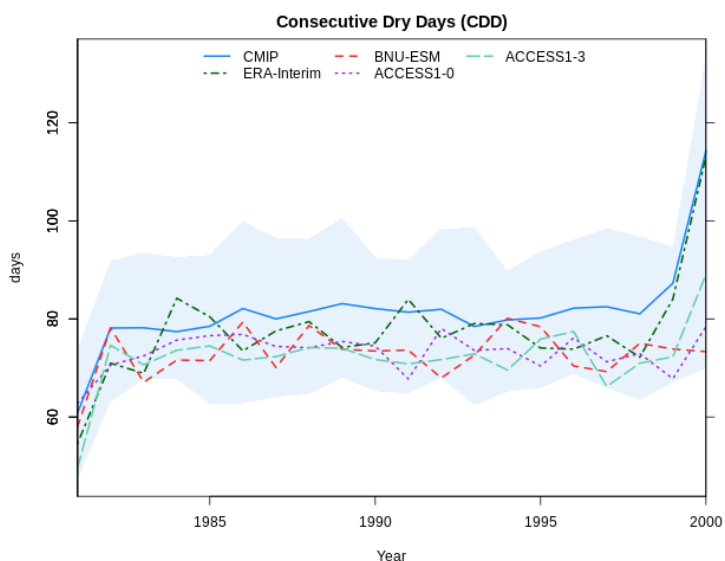

Fig. 13: Timeseries of the Consecutive Dry Days index over 1981-2000 for a selection of CMIP5 models, the CMIP5 multi-model mean (CMIP) and ERA-Interim. Shading is used to reproduce the multi-model spread.

8.9 Diagnostics of stratospheric dynamics and chemistry

8.9.1 Overview

This recipe reproduces the figures of Eyring et al. (2006) The following plots are reproduced:

- Vertical profile climatological mean bias of climatological mean for selected seasons and latitudinal region.

- Vertical and latitudinal profile of climatological mean for selected seasons this figure and setting is valid for figure 5 (CH4) figure 6 (H2O) figure 11 (HCL) figure 13 (tro3).

- Total ozone anomalies at different latitudinal band and seasons.

8.9.2 Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_eyring06jgr.yml

Diagnostics are stored in esmvaltool/diag_scripts/eyring06jgr/

- eyring06jgr_fig01.ncl

- eyring06jgr_fig05a.ncl

- eyring06jgr_fig05b.ncl

- eyring06jgr_fig15.ncl

8.9.3 User settings in recipe

1. Preprocessor

   - `regrid_interp_lev_zonal`: Regridding and interpolation reference_dataset levels used by eyring06jgr_fig01 and eyring06jgr_fig05

   - `zonal` : Regridding and zonal mean used by eyring06jgr_fig15

2. Script <eyring06jgr_fig01.ncl>

   Required settings for script

   - `latmin`: array of float, min lat where variable is averaged, i.e. [60., 60., -90., -90.]

   - `latmax`: array of float,and max lat where variable is averaged, i.e. [90., 90., -60., -60.]

   - `season`: array of string., season when variable is averaged, i.e. ["DJF", "MAM", "JJA", "SON"]

   - `XMin`: array of float, min limit X axis [-30., -30., -30., -30.]

   - `XMax`: array of float, max limit X axis [20., 20., 20., 20.]

   - `levmin`: array of float, min limit Y axis [1., 1., 1., 1.]

   - `levmax`: array of float, max limit Y axis [350., 350., 350., 350.]

   Optional settings for script

   - `start_year`: int, year when start the climatology calculation [1980] (default max among the models start year).

   - `end_year`:int, year when end the climatology calculation [1999] (default min among the models end year).

   - `multimean`: bool, calculate multi-model mean, (i.e. False/True) (default False).

   Required settings for variables

   - `preprocessor`: regrid_interp_lev.

   - `reference_dataset`: name of the reference model or observation for regridding and bias calculation (e.g. ERA-Interim").

   - `mip`: Amon.

8.9.4 Variables

- ta (atmos, monthly mean, longitude latitude level time)

8.9.5 Example plots

8.10 Heat wave and cold wave duration

8.10.1 Overview

The goal of this diagnostic is to estimate the relative change in heat/cold wave characteristics in future climates compared to a reference period using daily maximum or minimum temperatures.

The user can select whether to compute the frequency of exceedances or non-exceedances, which corresponds to extreme high or extreme low temperature events, respectively. The user can also select the minimum duration for an event to be classified as a heat/cold wave and the season of interest.

The diagnostic calculates the number of days in which the temperature exceeds or does not exceeds the necessary threshold for a consecutive number of days in future climate projections. The result is an annual time series of the total number of heat/cold wave days for the selected season at each grid point. The final output is the average number of heat/cold wave days for the selected season in the future climate projections.

8.10.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_heatwaves_coldwaves.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- extreme_spells.R: calculates the heatwave or coldwave duration.

8.10.3 User settings

User setting files are stored in recipes/

1. recipe_heatwaves_coldwaves.yml

   Required settings for script

   - quantile: quantile defining the exceedance/non-exceedance threshold
   - min_duration: Min duration in days of a heatwave/coldwave event
   - Operator: either '>' for exceedances or '<' for non-exceedances
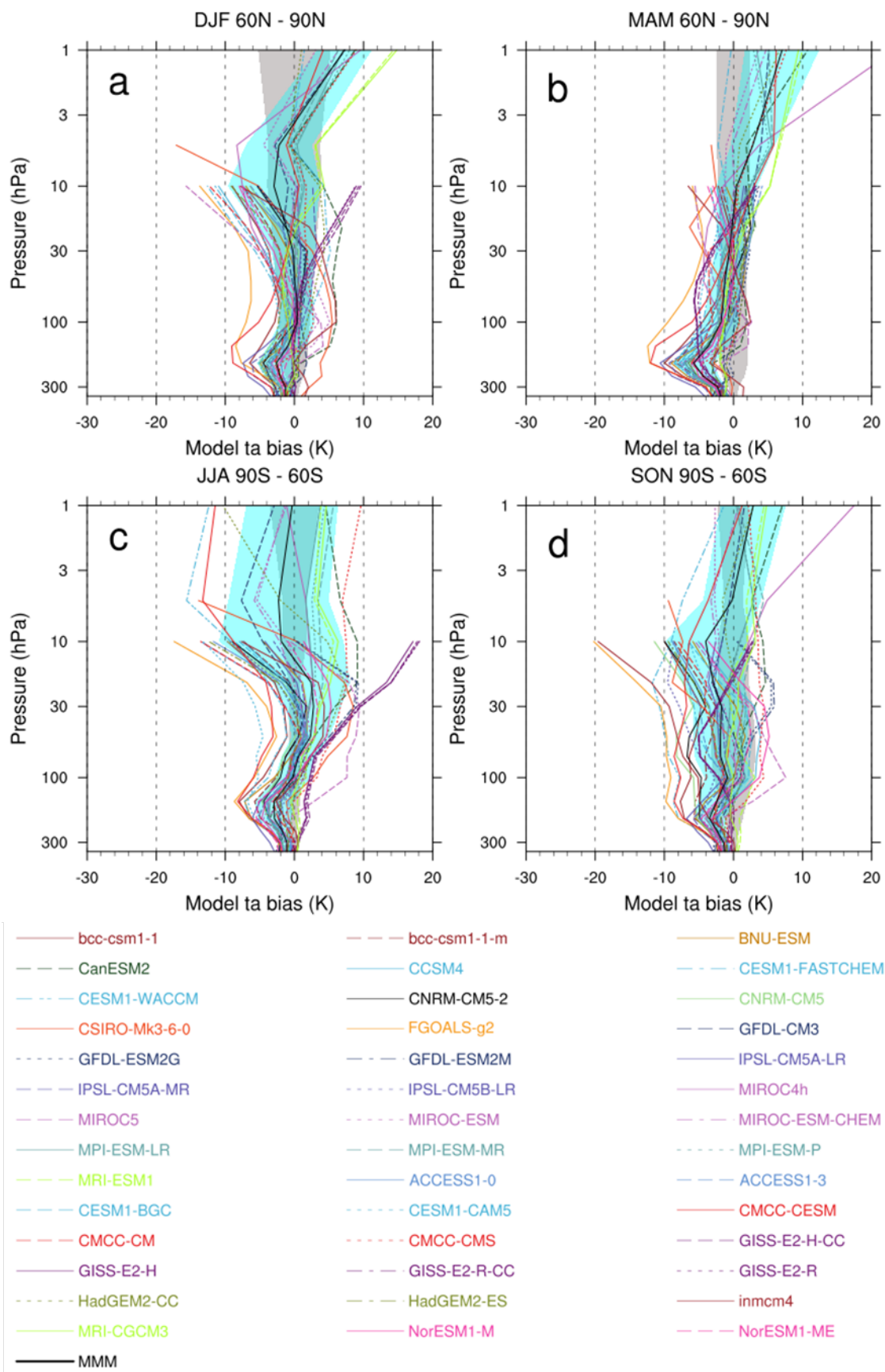   - season: 'summer' or 'winter

Fig. 14: Climatological mean temperature biases for (top) 60–90N and (bottom) 60–90S for the (left) winter and (right) spring seasons. The climatological means for the CCMs and ERA-Interim data from 1980 to 1999 are included. Biases are calculated relative to ERA-Interim reanalyses. The grey area shows ERA-Interim plus and minus 1 standard deviation (s) about the climatological mean. The turquoise area shows plus and minus 1 standard deviation about the multi-model mean.

8.10.4 Variables

- tasmax or tasmin (atmos, daily, longitude, latitude, time)

8.10.5 Observations and reformat scripts

None

8.10.6 References

- Cardoso, S., Marta-Almeida, M., Carvalho, A.C., & Rocha, A. (2017). Heat wave and cold spell changes in Iberia for a future climate scenario. International Journal of Climatology, 37(15), 5192-5205. https://doi.org/10.1002/joc.5158

- Ouzeau, G., Soubeyroux, J.-M., Schneider, M., Vautard, R., & Planton, S. (2016). Heat waves analysis over France in present and future climate: Application of a new method on the EURO-CORDEX ensemble. Climate Services, 4, 1-12. https://doi.org/10.1016/J.CLISER.2016.09.002

- Wang, Y., Shi, L., Zanobetti, A., & Schwartz, J. D. (2016). Estimating and projecting the effect of cold waves on mortality in 209 US cities. Environment International, 94, 141-149. https://doi.org/10.1016/j.envint.2016.05.008

- Zhang, X., Hegerl, G., Zwiers, F. W., & Kenyon, J. (2005). Avoiding inhomogeneity in percentile-based indices of temperature extremes. Journal of Climate, 18(11), 1641-1651. https://doi.org/10.1175/JCLI3366.1

8.10.7 Example plots
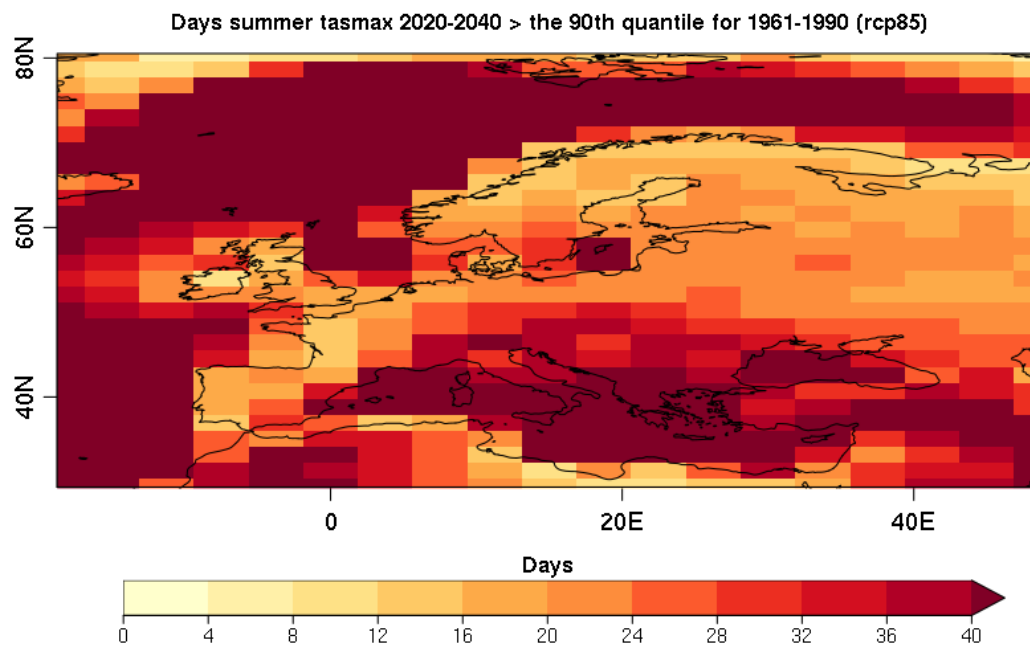
Mean number of summer days during the period 2060-2080 when the daily maximum near-surface air temperature exceeds the 80th quantile of the 1971-2000 reference period. The results are based on one RCP 8.5 scenario simulated by BCC-CSM1-1.

8.11 Hydroclimatic intensity and extremes (HyInt)

8.11.1 Overview

The HyInt tool calculates a suite of hydroclimatic and climate extremes indices to perform a multi-index evaluation of climate models. The tool firstly computes a set of 6 indices that allow to evaluate the response of the hydrological cycle to global warming with a joint view of both wet and dry extremes. The indices were selected following Giorgi et al. (2014) and include the simple precipitation intensity index (SDII) and extreme precipitation index (R95), the maximum dry spell length (DSL) and wet spell length (WSL), the hydroclimatic intensity index (HY-INT), which is a measure of the overall behaviour of the hydroclimatic cycle (Giorgi et al., 2011), and the precipitation area (PA), i.e. the area over which at any given day precipitation occurs, (Giorgi et al., 2014). Secondly, a selection of the 27 temperature and precipitation -based indices of extremes from the Expert Team on Climate Change Detection and Indices (ETCCDI) produced by the climdex (https://www.climdex.org) library can be ingested to produce a multi-index analysis. The tool allows then to perform a subsequent analysis of the selected indices calculating timeseries and trends over predefined continental areas, normalized to a reference period. Trends are calculated using the R *lm* function and significance testing performed with a Student T test on non-null coefficients hypothesis. Trend coefficients are stored together with their statistics which include standard error, t value and Pr(>|t|). The tool can then produce a variety of types of plots including global and regional maps, maps of comparison between models and a reference dataset, timeseries with their spread, trend lines and summary plots of trend coefficients.

The hydroclimatic indices calculated by the recipe_hyint.yml and included in the output are defined as follows:

- PRY = mean annual precipitation

- INT = mean annual precipitation intensity (intensity during wet days, or simple precipitation intensity index SDII)

- WSL = mean annual wet spell length (number of consecutive days during each wet spell)

- DSL = mean annual dry spell lenght (number of consecutive days during each dry spell)

- PA = precipitation area (area over which of any given day precipitation occurs)

- R95 = heavy precipitation index (percent of total precipitation above the 95% percentile of the reference distribution)

- HY-INT = hydroclimatic intensity. HY-INT = normalized(INT) x normalized(DSL).

The recipe_hyint_extreme_events.yml includes an additional call to the *Extreme Events Indices (ETCCDI)* diagnostics, which allows to calculate the ETCCDI indices and include them in the subsequent analysis together with the hydroclimatic indices. All of the selected indices are then stored in output files and figures.

8.11.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_hyint.yml (evaluating the 6 hydroclimatic indices, performing trend analysis and plotting)

- recipe_hyint_extreme_events.yml (similar to the recipe_hyint.yml but with an additional call to the *Extreme Events Indices (ETCCDI)* diagnostic for calculation of ETCCDI indices and inclusion of them in the trend analysis and plotting)

Diagnostics are stored in diag_scripts/hyint/

- hyint.R

and subroutines

- hyint_diagnostic.R

- hyint_functions.R

- hyint_parameters.R

- hyint_plot_trends.R

- hyint_etccdi_preproc.R

- hyint_metadata.R

- hyint_plot_maps.R

- hyint_preproc.R

- hyint_trends.R

See details of the extreme_events diagnostics under recipe_extreme_events.yml.

8.11.3 Known issues

recipe_hyint_extreme_events.yml

Call to the *Extreme Events Indices (ETCCDI)* diagnostic requires the ncdf4.helpers library, which is currently unavailable on CRAN. Users need therefore to install the library manually, e.g. through the following commands to download the package tarball from CRAN archive, install it and remove the package tarball:

- url <- "https://cran.r-project.org/src/contrib/Archive/ncdf4.helpers/ncdf4.helpers_0.3-3.tar.gz"

- pkgFile <- "ncdf4.helpers_0.3-3.tar.gz"

- download.file(url = url, destfile = pkgFile)

- install.packages(pkgs=pkgFile, type="source", repos=NULL)

- unlink(pkgFile)

8.11.4 User settings

Required settings for script

- norm_years: first and last year of reference normalization period to be used for normalized indices

- select_indices: indices to be analysed and plotted. Select one or more fields from the following list (order-sensitive): "pa_norm", "hyint", "int_norm", "r95_norm", "wsl_norm", "dsl_norm", "int", "dsl", "wsl"

- select_regions: Select regions for timeseries and maps from the following list: GL=Globe, GL60=Global 60S/60N, TR=Tropics (30S/30N), SA=South America, AF=Africa, NA=North America, IN=India, EU=Europe, EA=East-Asia, AU=Australia

- plot_type: type of figures to be plotted. Select one or more from: 1=lon/lat maps per individual field/exp/multi-year mean, 2=lon/lat maps per individual field exp-ref-diff/multi-year mean, 3=lon/lat maps multi-field/exp-ref-diff/multi-year mean, 11=timeseries over required individual region/exp, 12=timeseries over multiple regions/exp, 13=timeseries with multiple models, 14=summary trend coefficients multiple regions, 15=summary trend coefficients multiple models

Additional settings for recipe_hyint_extreme_events.yml

- call to the extreme_events diagnostics: see details in recipe_extreme_events.yml. Make sure that the base_range for extreme_events coincides with the norm_range of hyint and that all ETCCDI indices that are required to be imported in hyint are calculated by the extreme_events diagnostics.

- etccdi_preproc: set to true to pre-process and include ETCCDI indices in hyint

- etccdi_list_import: specify the list of ETCCDI indices to be imported, e.g.: "tn10pETCCDI", "tn90pETCCDI", "tx10pETCCDI", "tx90pETCCDI"

- select_indices: this required settings should here be revised to include the imported indices, e.g.: "pa_norm", "hyint", "tn10pETCCDI", "tn90pETCCDI", "tx10pETCCDI", "tx90pETCCDI"

Optional settings for script (with default setting)

1. Data

   - rgrid (false): Define whether model data should be regridded. (a) false to keep original resolution; (b) set desired regridding resolution in cdo format e.g., "r320x160"; (c) "REF" to use resolution of reference model

2. Plotting

   - npancol (2): number of columns in timeseries/trends multipanel figures

   - npanrow (3): number of rows in timeseries/trends multipanel figures

   - autolevels (true): select automated (true) or pre-set (false) range of values in plots

   - autolevels_scale (1): factor multiplying automated range for maps and timeseries

   - autolevels_scale_t (1.5): factor multiplying automated range for trend coefficients

3. Maps

   - oplot_grid (false): plot grid points over maps

   - boxregion (false): !=0 plot region boxes over global maps with thickness = abs(boxregion); white (>0) or grey (<0).

   - removedesert (false) remove (flag as NA) grid points with mean annual pr < 0.5 mm/day (deserts, Giorgi2014). This affects timeseries and trends calculations too.

4. Timeseries and trends

   - weight_tseries (true): adopt area weights in timeseries

   - trend_years (false): (a) false = apply trend to all years in dataset; (b) [year1, year2] to apply trend calculation and plotting only to a limited time interval

   - add_trend (true): add linear trend to plot

   - add_trend_sd (false): add dashed lines of stdev range to timeseries

   - add_trend_sd_shade (false): add shade of stdev range to timeseries

   - add_tseries_lines (true): plot lines connecting timeseries points

   - add_zeroline (true): plot a dashed line at y=0

   - trend_years_only (false): limit timeseries plotting to the time interval adopted for trend calculation (excluding the normalization period)

   - scale100years (true): plot trends scaled as 1/100 years

   - scalepercent (false): plot trends as percent change

8.11.5 Variables

- pr (atmos, daily mean, longitude latitude time)

Additional variables for recipe_hyint_extreme_events.yml

- tas (atmos, daily mean, longitude latitude time)
- tasmin (atmos, daily mean, longitude latitude time)
- tasmax (atmos, daily mean, longitude latitude time)

8.11.6 Observations and reformat scripts

None.

8.11.7 References

- Giorgi et al., 2014, J. Geophys. Res. Atmos., 119, 11,695–11,708, doi:10.1002/ 2014JD022238
- Giorgi et al., 2011, J. Climate 24, 5309-5324, doi:10.1175/2011JCLI3979.1

8.11.8 Example plots
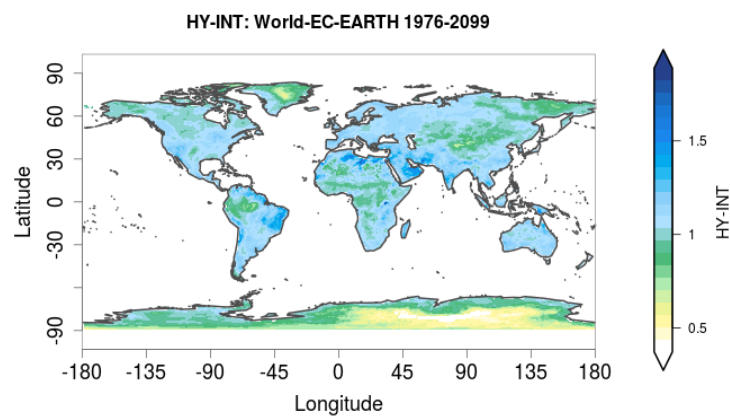
Fig. 15: Mean hydroclimatic intensity for the EC-EARTH model, for the historical + RCP8.5 projection in the period 1976-2099
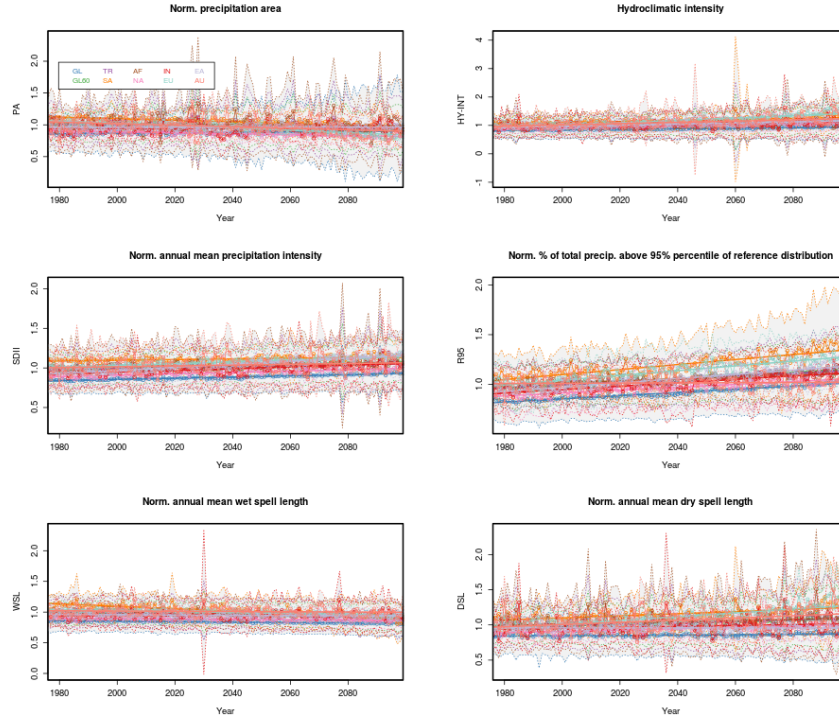
Fig. 16: Timeseries for multiple indices and regions for the ACCESS1-0 model, for the historical + RCP8.5 projection in the period 1976-2099, normalized to the 1976-2005 historical period.
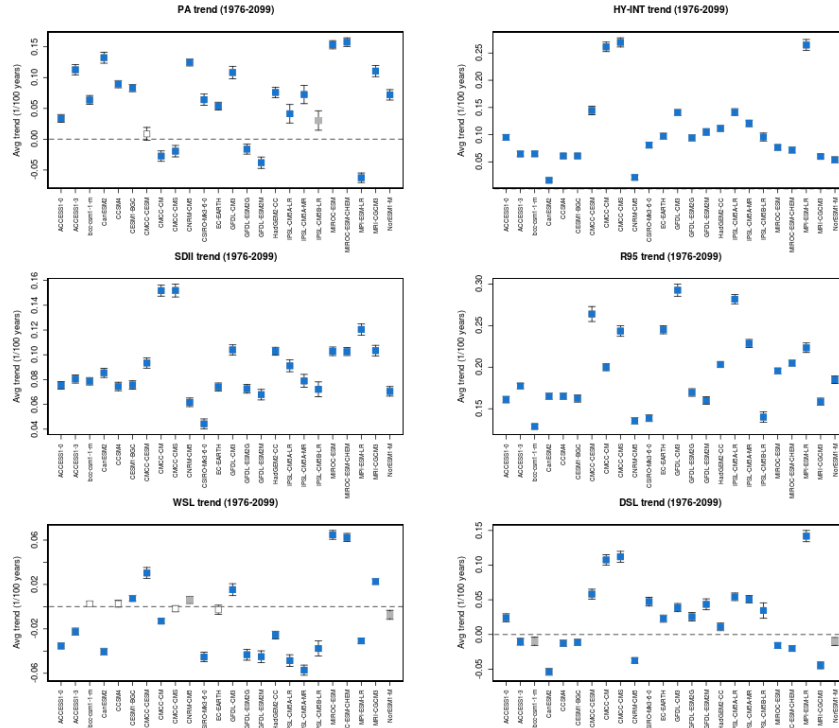
Fig. 17: Multi-model trend coefficients over selected indices for CMIP5 models in the RCP8.5 2006-2099 projection, normalized to the 1976-2005 historical period.

8.11. Hydroclimatic intensity and extremes (HyInt)   71

8.12 Modes of variability

8.12.1 Overview

The goal of this recipe is to compute modes of variability from a reference or observational dataset and from a set of climate projections and calculate the root-mean-square error between the mean anomalies obtained for the clusters from the reference and projection data sets. This is done through K-means or hierarchical clustering applied either directly to the spatial data or after computing the EOFs.

The user can specify the number of clusters to be computed.

The recipe's output consist of three netcdf files for both the observed and projected weather regimes and the RMSE between them.

8.12.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_modes_of_variability.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- WeatherRegime.R - function for computing the EOFs and k-means and hierarchical clusters.
- weather_regime.R - applies the above weather regimes function to the datasets

8.12.3 User settings

User setting files are stored in recipes/

1. recipe_modes_of_variability.yml

   Required settings for script

   - plot type: rectangular or polar
   - ncenters: number of centers to be computed by the clustering algorithm (maximum 4)
   - cluster_method: kmeans (only psl variable) or hierarchical clustering (for psl or sic variables)
   - detrend_order: the order of the polynomial detrending to be applied (0, 1 or 2)
   - EOFs: logical indicating wether the k-means clustering algorithm is applied directly to the spatial data ('false') or to the EOFs ('true')
   - frequency: select the month (format: JAN, FEB, . . .) or season (format: JJA, SON, MAM, DJF) for the diagnostic to be computed for (does not work yet for MAM with daily data).

8.12.4 Variables

- psl (atmos, monthly/daily, longitude, latitude, time)

8.12.5 Observations and reformat scripts

None

8.12.6 References

- Dawson, A., T. N. Palmer, and S. Corti, 2012: Simulating regime structures in weather and climate prediction models. Geophysical Research Letters, 39 (21), https://doi.org/10.1029/2012GL053284.
- Ferranti, L., S. Corti, and M. Janousek, 2015: Flow-dependent verification of the ECMWF ensemble over the Euro-Atlantic sector. Quarterly Journal of the Royal Meteorological Society, 141 (688), 916-924, https://doi.org/10.1002/qj.2411.
- Grams, C. M., Beerli, R., Pfenninger, S., Staffell, I., & Wernli, H. (2017). Balancing Europe's wind-power output through spatial deployment informed by weather regimes. Nature climate change, 7(8), 557, https://doi.org/10.1038/nclimate3338.
- Hannachi, A., D. M. Straus, C. L. E. Franzke, S. Corti, and T. Woollings, 2017: Low Frequency Nonlinearity and Regime Behavior in the Northern Hemisphere Extra-Tropical Atmosphere. Reviews of Geophysics, https://doi.org/10.1002/2015RG000509.
- Michelangeli, P.-A., R. Vautard, and B. Legras, 1995: Weather regimes: Recurrence and quasi stationarity. Journal of the atmospheric sciences, 52 (8), 1237-1256, doi: 10.1175/1520-0469(1995)052<1237:WRRAQS>2.0.CO.
- Vautard, R., 1990: Multiple weather regimes over the North Atlantic: Analysis of precursors and successors. Monthly weather review, 118 (10), 2056-2081, doi: 10.1175/1520-0493(1990)118<2056:MWROTN>2.0.CO;2.
- Yiou, P., K. Goubanova, Z. X. Li, and M. Nogaj, 2008: Weather regime dependence of extreme value statistics for summer temperature and precipitation. Nonlinear Processes in Geophysics, 15 (3), 365-378, https://doi.org/10.5194/npg-15-365-2008.

8.12.7 Example plots

Four modes of variability for autumn (September-October-November) in the North Atlantic European Sector for the RCP 8.5 scenario using BCC-CSM1-1 future projection during the period 2020-2075. The frequency of occurrence of each variability mode is indicated in the title of each map.

8.13 Precipitation quantile bias

8.13.1 Overview

Precipitation is a dominant component of the hydrological cycle, and as such a main driver of the climate system and human development. The reliability of climate projections and water resources strategies therefore depends on how well precipitation can be reproduced by the models used for simulations. While global circulation models from the CMIP5 project observations can reproduce the main patterns of mean precipitation, they often show shortages and biases in the ability to reproduce the strong precipitation tails of the distribution. Most models underestimate precipitation over arid regions and overestimate it over regions of complex topography, and these shortages are amplified at
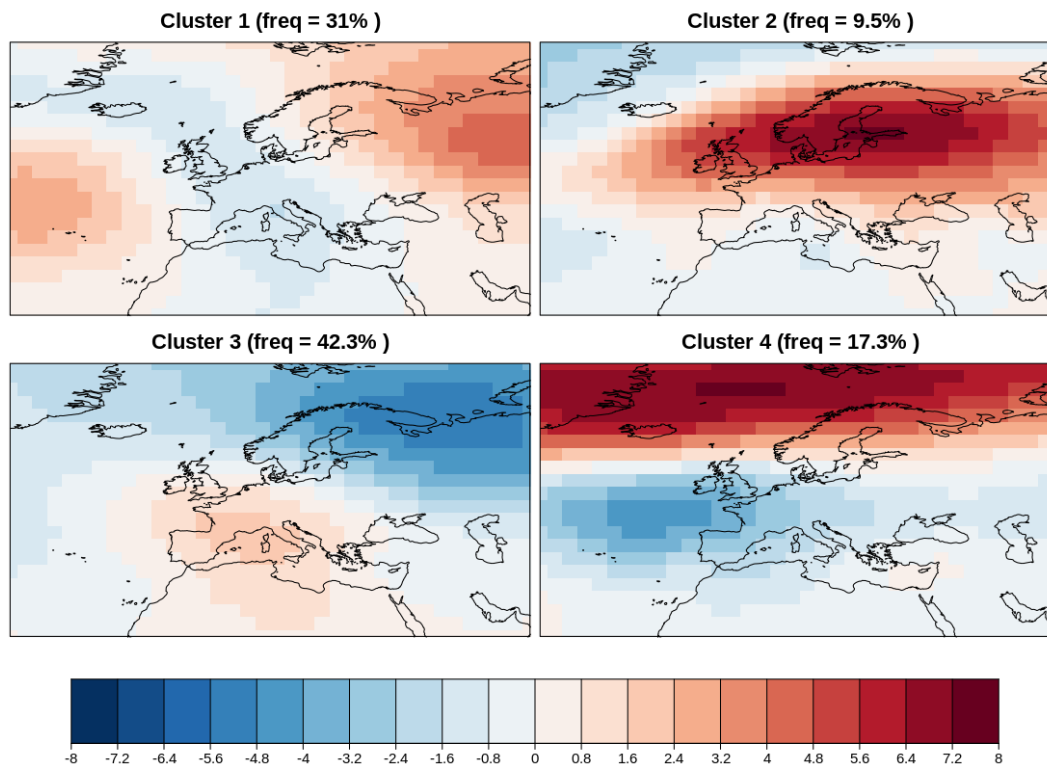
high quantile precipitation. The quantilebias recipe implements calculation of the quantile bias to allow evaluation of the precipitation bias based on a user defined quantile in models as compared to a reference dataset following Mehran et al. (2014). The quantile bias (QB) is defined as the ratio of monthly precipitation amounts in each simulation to that of the reference dataset (GPCP observations in the example) above a specied threshold t (e.g., the 75th percentile of all the local monthly values). A quantile bias equal to 1 indicates no bias in the simulations, whereas a value above (below) 1 corresponds to a climate model's overestimation (underestimation) of the precipitation amount above the specified threshold t, with respect to that of the reference dataset.

8.13.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_quantilebias.yml

Diagnostics are stored in diag_scripts/quantilebias/

- quantilebias.R

8.13.3 User settings

Required settings for script

- perc_lev: quantile (in %), e.g. 50

8.13.4 Variables

- pr (atmos, monthly, longitude latitude time)

8.13.5 Observations and reformat scripts

- GPCP-SG observations (accessible via the obs4mips project)

8.13.6 References

- Mehran, A. et al.: Journal of Geophysical Research: Atmospheres, Volume 119, Issue 4, pp. 1695-1707, 2014.
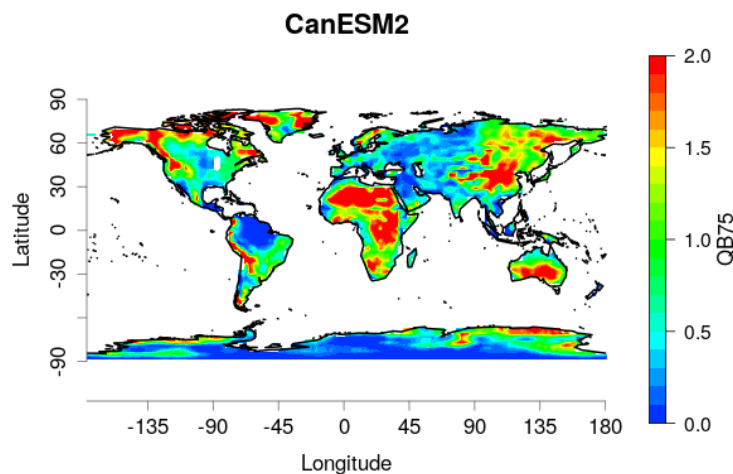
8.13.7 Example plots

Fig. 18: Quantile bias, as defined in Mehran et al. 2014, with threshold t=75th percentile, evaluated for the CanESM2 model over the 1979-2005 period, adopting GPCP-SG v 2.3 gridded precipitation as a reference dataset. The optimal reference value is 1. Both datasets have been regridded onto a 2° regular grid.

8.14 Standardized Precipitation-Evapotranspiration Index (SPEI)

8.14.1 Overview

Droughts can be separated into three main types: meteorological, hydrological, and agricultural drought.

Common for all types is that a drought needs to be put in context of local and seasonal characteristics, i.e. a drought should not be defined with an absolute threshold, but as an anomalous condition.

Meteorological droughts are often described using the standardized precipitation index (SPI; McKee et al, 1993), which in a standardized way describes local precipitation anomalies. It is calculated on monthly mean precipitation, and is therefore not accounting for the intensity of precipitation and the runoff process. Because SPI does not account for evaporation from the ground, it lacks one component of the water fluxes at the surface and is therefore not compatible with the concept of hydrological drought.

A hydrological drought occurs when low water supply becomes evident, especially in streams, reservoirs, and ground-water levels, usually after extended periods of meteorological drought. GCMs normally do not simulate hydrological processes in sufficient detail to give deeper insights into hydrological drought processes. Neither do they properly describe agricultural droughts, when crops become affected by the hydrological drought. However, hydrological drought can be estimated by accounting for evapotranspiration, and thereby estimate the surface retention of water. The standardized precipitation-evapotranspiration index (SPEI; Vicente-Serrano et al., 2010) has been developed to also account for temperature effects on the surface water fluxes. Evapotranspiration is not normally calculated in GCMs, so SPEI often takes other inputs to estimate the evapotranspiration. Here, the Thornthwaite (Thornthwaite, 1948) method based on temperature is applied.

8.14.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_spei.yml

Diagnostics are stored in diag_scripts/droughtindex/

- diag_spi.R: calculate the SPI index
- diag_spei.R: calculate the SPEI index

8.14.3 User settings

1. Script diag_spi.py

   Required settings (script)

   - reference_dataset: dataset_name The reference data set acts as a baseline for calculating model bias.

2. Script diag_spei.py

   Required settings (script)

   - reference_dataset: dataset_name The reference data set acts as a baseline for calculating model bias.

8.14.4 Variables

- pr (atmos, monthly mean, time latitude longitude)
- tas (atmos, monthly mean, time latitude longitude)

8.14.5 References

- McKee, T. B., Doesken, N. J., & Kleist, J. (1993). The relationship of drought frequency and duration to time scales. In Proceedings of the 8th Conference on Applied Climatology (Vol. 17, No. 22, pp. 179-183). Boston, MA: American Meteorological Society.
- Vicente-Serrano, S. M., Beguería, S., & López-Moreno, J. I. (2010). A multiscalar drought index sensitive to global warming: the standardized precipitation evapotranspiration index. Journal of climate, 23(7), 1696-1718.
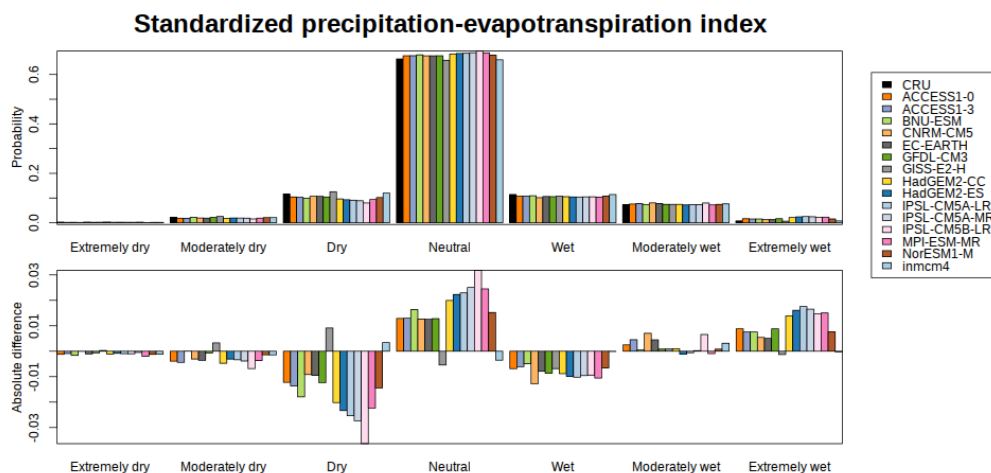
8.14.6 Example plots

Fig. 19: (top) Probability distribution of the standardized precipitation-evapotranspiration index of a sub-set of the CMIP5 models, and (bottom) bias relative to the CRU reference data set.
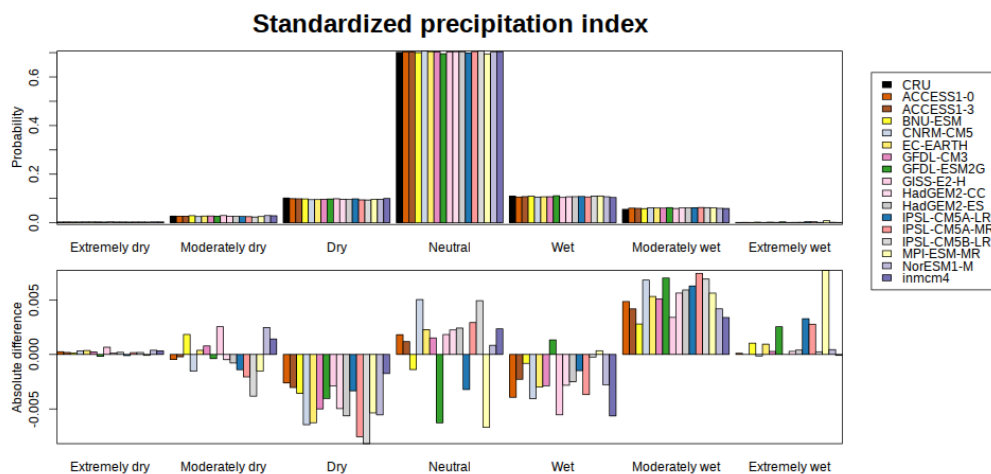

Fig. 20: (top) Probability distribution of the standardized precipitation index of a sub-set of the CMIP5 models, and (bottom) bias relative to the CRU reference data set.

8.15 Drought characteristics following Martin (2018)

8.15.1 Overview

Following Martin (2018) drought characteristics are calculated based on the standard precipitation index (SPI), see Mckee et al. (1993). These characteristics are frequency, average duration, SPI index and severity index of drought events.

8.15.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_martin18grl.yml

Diagnostics are stored in diag_scripts/

- droughtindex/diag_save_spi.R

- droughtindex/collect_drought_obs_multi.py

- droughtindex/collect_drought_model.py

- droughtindex/collect_drought_func.py

8.15.3 User settings in recipe

The recipe can be run with different CMIP5 and CMIP6 models and one observational or reanalysis data set.

The droughtindex/diag_save_spi.R script calculates the SPI index for any given time series. It is based on droughtindex/diag_spi.R but saves the SPI index and does not plot the histogram. The distribution and the representative time scale (smooth_month) can be set by the user, the values used in Martin (2018) are smooth_month: 6 and distribution: 'Gamma' for SPI.

There are two python diagnostics, which can use the SPI data to calculate the drought characteristics (frequency, average duration, SPI index and severity index of drought events) based on Martin (2018):

- To compare these characteristics between model data and observations or renanalysis data use droughtindex/collect_drought_obs_multi.py Here, the user can set: * indexname: Necessary to identify data produced by droughtindex/diag_save_spi.R as well as write captions and filenames. At the moment only indexname: 'SPI' is supported. * threshold: Threshold for this index below which an event is considered to be a drought, the setting for SPI should be usually threshold: -2.0 but any other value will be accepted. Values should not be < - 3.0 or > 3.0 for SPI (else it will identify none/always drought conditions).

- To compare these ccharacteristics between different time periods in model data use droughtindex/collect_drought_model.py Here, the user can set: * indexname: Necessary to identify data produced by droughtindex/diag_save_spi.R as well as write captions and filenames. At the moment only indexname: 'SPI' is supported. * threshold: Threshold for this index below which an event is considered to be a drought, the setting for SPI should be usually threshold: -2.0 but any other value will be accepted. Values should not be < - 3.0 or > 3.0 for SPI (else it will identify none/always drought conditions). * start_year: Needs to be equal or larger than the start_year for droughtindex/diag_save_spi.R. * end_year: Needs to be equal or smaller than the end_year for droughtindex/diag_save_spi.R. * comparison_period: should be < (end_year - start_year)/2 to have non overlapping time series in the comparison.

The third diagnostic droughtindex/collect_drought_func.py contains functions both ones above use.

8.15.4 Variables

- *pr* (atmos, monthly, longitude, latitude, time)

8.15.5 Observations and reformat scripts

None

8.15.6 References

- Martin, E.R. (2018). Future Projections of Global Pluvial and Drought Event Characteristics. Geophysical Research Letters, 45, 11913-11920.

- McKee, T. B., Doesken, N. J., & Kleist, J. (1993). The relationship of drought frequency and duration to time scales. In Proceedings of the 8th Conference on Applied Climatology (Vol. 17, No. 22, pp. 179-183). Boston, MA: American Meteorological Society.

8.15.7 Example plots
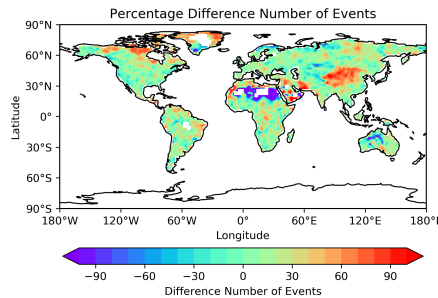
Fig. 21: Global map of the percentage difference between multi-model mean of 15 CMIP models and the CRU data for the number of drought events [%] based on SPI.
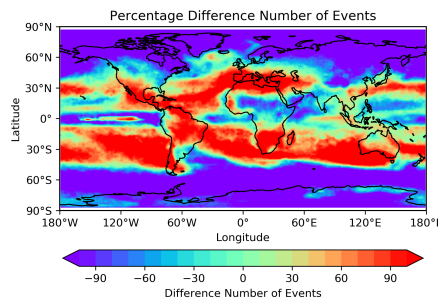
Fig. 22: Global map of the percentage difference between multi-model mean for RCP8.5 scenarios (2050-2100) runs and historical data (1950-2000) for 15 CMIP models for the number of drought events [%] based on SPI.

8.16 Stratosphere - Autoassess diagnostics

8.16.1 Overview

Polar night jet / easterly jet strengths are defined as the maximum / minimum wind speed of the climatological zonal mean jet, and measure how realistic the zonal wind climatology is in the stratosphere.

Extratropical temperature at 50hPa (area averaged poleward of 60 degrees) is important for polar stratospheric cloud formation (in winter/spring), determining the amount of heterogeneous ozone depletion simulated by models with interactive chemistry schemes.

The Quasi-Biennial Oscillation (QBO) is a good measure of tropical variability in the stratosphere. Zonal mean zonal wind at 30hPa is used to define the period and amplitude of the QBO.

The tropical tropopause cold point (100hPa, 10S-10N) temperature is an important factor in determining the stratospheric water vapour concentrations at entry point (70hPa, 10S-10N), and this in turn is important for the accurate simulation of stratospheric chemistry and radiative balance.

8.16.2 Prior and current contributors

Met Office:

- Prior to May 2008: Neal Butchart
- May 2008 - May 2016: Steven C Hardiman
- Since May 2016: Alistair Sellar and Paul Earnshaw

ESMValTool:

- Since April 2018: Porting into ESMValTool by Valeriu Predoi

8.16.3 Developers

Met Office:

- Prior to May 2008: Neal Butchart
- May 2008 - May 2016: Steven C Hardiman

ESMValTool:

- Since April 2018: Valeriu Predoi

8.16.4 Review of current port in ESMValTool

The code and results review of the port from native Autoassess to ESMValTool was conducted by Alistair Sellar (mailto:alistair.sellar@matoffice.gov.uk) and Valeriu Predoi (mailto:valeriu.predoi@ncas.ac.uk) in July 2019. Review consisted in comparing results from runs using ESMValTool's port and native Autoassess using the same models and data stretches.

8.16.5 Metrics and Diagnostics

Performance metrics:

- Polar night jet: northern hem (January) vs. ERA Interim
- Polar night jet: southern hem (July) vs. ERA Interim
- Easterly jet: southern hem (January) vs. ERA Interim
- Easterly jet: northern hem (July) vs. ERA Interim
- 50 hPa temperature: 60N-90N (DJF) vs. ERA Interim
- 50 hPa temperature: 60N-90N (MAM) vs. ERA Interim
- 50 hPa temperature: 90S-60S (JJA) vs. ERA Interim
- 50 hPa temperature: 90S-60S (SON) vs. ERA Interim
- QBO period at 30 hPa vs. ERA Interim
- QBO amplitude at 30 hPa (westward) vs. ERA Interim
- QBO amplitude at 30 hPa (eastward) vs. ERA Interim
- 100 hPa equatorial temp (annual mean) vs. ERA Interim
- 100 hPa equatorial temp (annual cycle strength) vs. ERA Interim
- 70 hPa 10S-10N water vapour (annual mean) vs. ERA-Interim

Diagnostics:

- Age of stratospheric air vs. observations from Andrews et al. (2001) and Engel et al. (2009)

8.16.6 Model Data

Variable/Field name	realm	frequency	Comment
Eastward wind (ua)	Atmosphere	monthly mean	original stash: x-wind, no stash
Air temperature (ta)	Atmosphere	monthly mean	original stash: m01s30i204
Specific humidity (hus)	Atmosphere	monthly mean	original stash: m01s30i205

The recipe takes as input a control model and experimental model, comparisons being made with these two CMIP models; additionally it can take observational data s input, in the current implementation ERA-Interim.

8.16.7 Inputs and usage

The `stratosphere` area metric is part of the `esmvaltool/diag_scripts/autoassess` diagnostics, and, as any other `autoassess` metric, it uses the `autoassess_area_base.py` as general purpose wrapper. This wrapper accepts a number of input arguments that are read through from the recipe.

This recipe is part of the larger group of Autoassess metrics ported to ESMValTool from the native Autoassess package from the UK's Met Office. The `diagnostics` settings are almost the same as for the other Atoassess metrics.

Note: Time gating for autoassess metrics.

To preserve the native Autoassess functionalities, data loading and selection on time is done somewhat differently for ESMValTool's autoassess metrics: the time selection is done in the preprocessor as per usual but a further time selection is performed as part of the diagnostic. For this purpose the user will specify a `start:` and `end:` pair of

---

arguments of `scripts:` `autoassess_script` (see below for example). These are formatted as `YYYY/MM/DD`; this is necessary since the Autoassess metrics are computed from 1-Dec through 1-Dec rather than 1-Jan through 1-Jan. This is a temporary implementation to fully replicate the native Autoassess functionality and a minor user inconvenience since they need to set an extra set of `start` and `end` arguments in the diagnostic; this will be phased when all the native Autoassess metrics hanve been ported to ESMVaITool review has completed.

Note: Polar Night/Easterly Jets Metrics

Polar Night Jets (PNJ) metrics require data available at very low air pressures ie very high altitudes; both Olar Night Jet and Easterly Jets computations should be preformed using `ta` and `ua` data at `<< 100 Pa`; the lowest air pressure found in atmospheric CMOR mip tables corresponds to `plev39` air pressure table, and is used in the `AERmonZ` mip. If the user requires correct calculations of these jets, it is highly advisable to use data from `AERmonZ`. Note that standard QBO calculation is exact for `plev17` or `plev19` tables.

An example of standard inputs as read by `autoassess_area_base.py` and passed over to the diagnostic/metric is listed below.

```
scripts:
  autoassess_strato_test_1: &autoassess_strato_test_1_settings
    script: autoassess/autoassess_area_base.py  # the base wrapper
    title: "Autoassess Stratosphere Diagnostic Metric"  # title
    area: stratosphere  # assesment area
    control_model: UKESM1-0-LL-hist  # control dataset name
    exp_model: UKESM1-0-LL-piCont  # experiment dataset name
    obs_models: [ERA-Interim]  # list to hold models that are NOT for metrics but for
    →obs operations
    additional_metrics: [ERA-Interim]  # list to hold additional datasets for metrics
    start: 2004/12/01  # start date in native Autoassess format
    end: 2014/12/01  # end date in native Autoassess format
```

8.16.8 References

- Andrews, A. E., and Coauthors, 2001: Mean ages of stratospheric air derived from in situ observations of CO2, CH4, and N2O. J. Geophys. Res., 106 (D23), 32295-32314.

- Dee, D. P., and Coauthors, 2011: The ERA-Interim reanalysis: configuration and performance of the data assimilation system. Q. J. R. Meteorol. Soc, 137, 553-597, doi:10.1002/qj.828.

- Engel, A., and Coauthors, 2009: Age of stratospheric air unchanged within uncertainties over the past 30 years. Nat. Geosci., 2, 28-31, doi:10 .1038/NGEO388.

8.16.9 Observations Data sets

ERA-Interim data (Dee et al., 2011) data can be obtained online from ECMWF and NASA respectively. Monthly mean zonal mean U and T data are required. CMORized that exists on CEDA-Jasmin or DKRZ (contact Valeriu Predoi (mailto:valeriu.predoi@ncas.ac.uk) for Jasmin or Mattia Righi (mailto:mattia.righi@dlr.de)for DKRZ).

8.16.10 Sample Plots and metrics

Below is a set of metrics for UKESM1-0-LL (historical data); the table shows a comparison made between running ESMValTool on CMIP6 CMORized netCDF data freely available on ESGF nodes and the run made using native Autoassess performed at the Met Office using the pp output of the model.

Metric name	UKESM1-0-LL; CMIP6: AERmonZ; historical, ESGF	UKESM1-0-LL; pp files; historical, u-bc179
Polar night jet: northern hem (January)	44.86	44.91
Polar night jet: southern hem (July)	112.09	112.05
Easterly jet: southern hem (January)	76.12	75.85
Easterly jet: northern hem (July)	55.68	55.74
QBO period at 30 hPa	41.50	41.00
QBO amplitude at 30 hPa (westward)	27.39	27.39
QBO amplitude at 30 hPa (eastward)	17.36	17.36
50 hPa temperature: 60N-90N (DJF)	27.11	26.85
50 hPa temperature: 60N-90N (MAM)	40.94	40.92
50 hPa temperature: 90S-60S (JJA)	11.75	11.30
50 hPa temperature: 90S-60S (SON)	23.88	23.63
100 hPa equatorial temp (annual mean)	15.29	15.30
100 hPa equatorial temp (annual cycle strength)	1.67	1.67
100 hPa 10Sto10N temp (annual mean)	15.48	15.46
100 hPa 10Sto10N temp (annual cycle strength)	1.62	1.62
70 hPa 10Sto10N wv (annual mean)	5.75	5.75

Results from `u-bc179` have been obtained by running the native Autoassess/stratosphere on `.pp` data from UKESM1 `u-bc179` suite and are listed here to confirm the compliance between the ported Autoassess metric in ESMValTool and the original native metric.

Another reference run comparing UKESM1-0-LL to the physical model HadGEM3-GC31-LL can be found here .
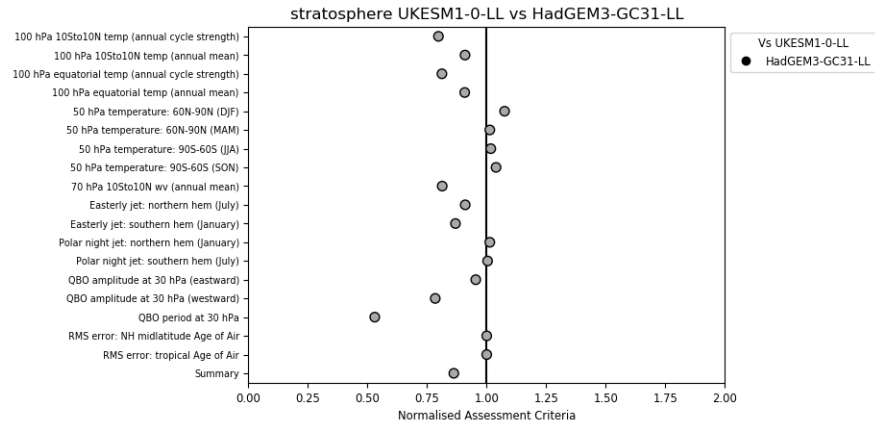
Fig. 23: Standard metrics plot comparing standard metrics from UKESM1-0-LL and HadGEM3-GC31.
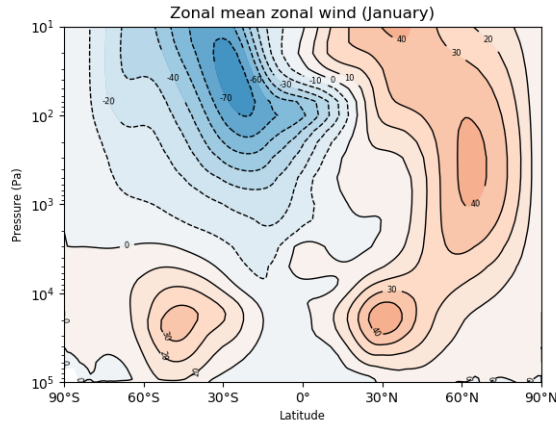
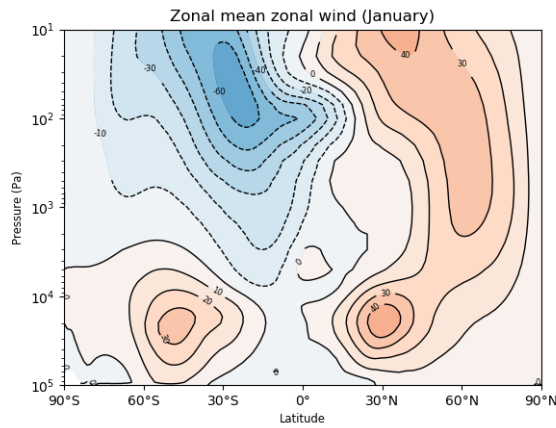Fig. 24: Zonal mean zonal wind in January for UKESM1-0-LL.

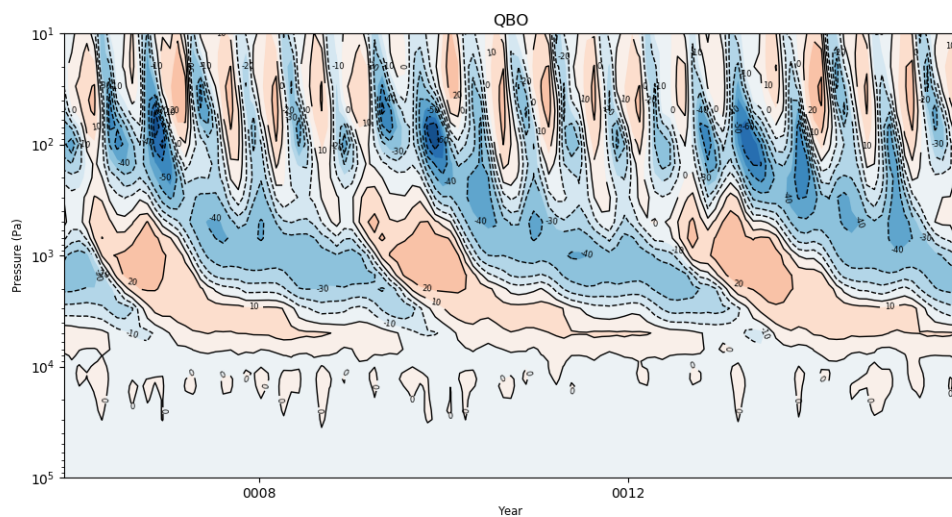Fig. 25: Zonal mean zonal wind in January for HadGEM3-GC31-LL.
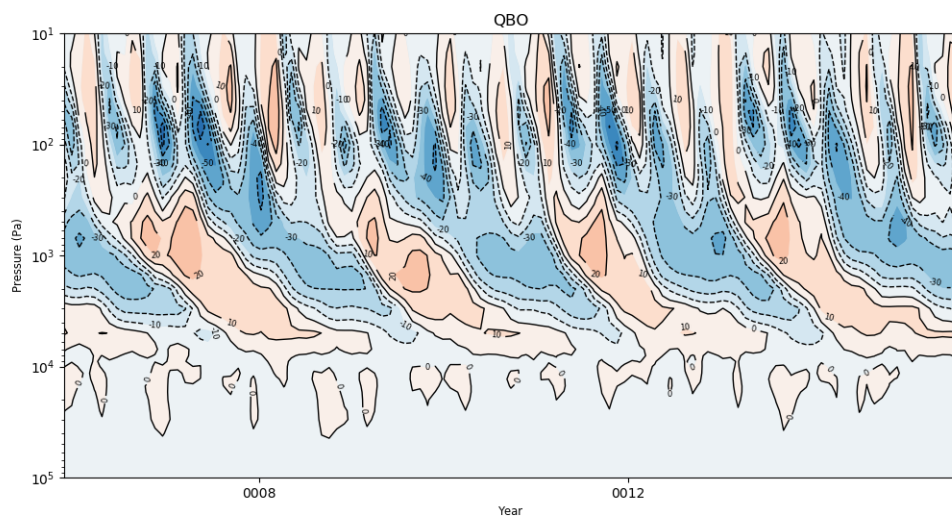
Fig. 26: QBO for UKESM1-0-LL.
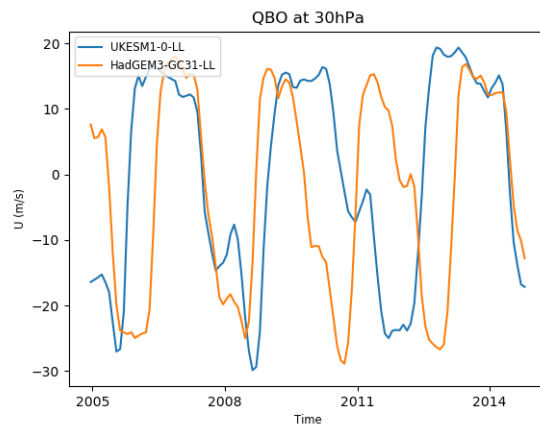

Fig. 27: QBO for HadGEM3-GC31-LL.

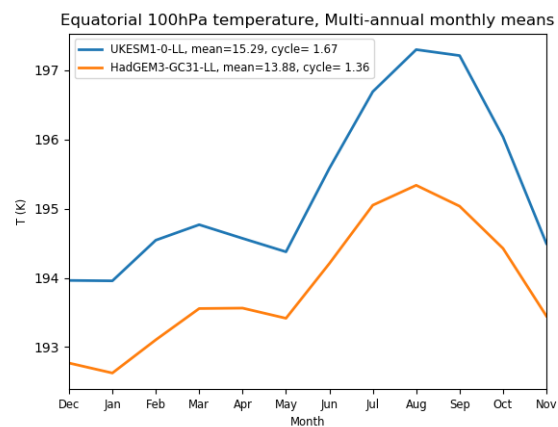Fig. 28: QBO at 30hPa comparison between UKESM1-0-LL and HadGEM3-GC31-LL.

Fig. 29: Equatorial temperature at 100hPa, multi annual means.

8.17 Stratosphere-troposphere coupling and annular modes indices (ZMNAM)

8.17.1 Overview

The current generation of climate models include the representation of stratospheric processes, as the vertical coupling with the troposphere is important for the weather and climate at the surface (e.g., Baldwin and Dunkerton, 2001).

The recipe recipe_zmnam.yml can be used to evaluate the representation of the Northern Annular Mode (NAM, e.g., Wallace, 2000) in climate simulations, using reanalysis datasets as reference.

The calculation is based on the "zonal mean algorithm" of Baldwin and Thompson (2009), and is alternative to pressure based or height-dependent methods.

This approach provides a robust description of the stratosphere-troposphere coupling on daily timescales, requiring less subjective choices and a reduced amount of input data. Starting from daily mean geopotential height on pressure levels, the leading empirical orthogonal function/principal component are computed from zonal mean daily anomalies, with the leading principal component representing the zonal mean NAM index. The regression of the monthly mean geopotential height onto this monthly averaged index represents the NAM pattern for each selected pressure level.

The outputs of the procedure are the monthly time series and the histogram of the daily zonal-mean NAM index, and the monthly regression maps for selected pressure levels. The users can select the specific datasets (climate model simulation and/or reanalysis) to be evaluated, and a subset of pressure levels of interest.

8.17.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_zmnam.yml

Diagnostics are stored in diag_scripts/zmnam/

- zmnam.py

and subroutines

- zmnam_calc.py
- zmnam_plot.py
- zmnam_preproc.py

8.17.3 User settings

None.

8.17.4 Variables

- zg (atmos, daily mean, longitude latitude time)

8.17.5 Observations and reformat scripts

None.

8.17.6 References

- Baldwin, M. P. and Thompson, D. W. (2009), A critical comparison of stratosphere–troposphere coupling indices. Q.J.R. Meteorol. Soc., 135: 1661-1672. doi:10.1002/qj.479.

- Baldwin, M. P and Dunkerton, T. J. (2001), Stratospheric Harbingers of Anomalous Weather Regimes. Science 294 (5542): 581-584. doi:10.1126/science.1063315.

- Wallace, J. M. (2000), North Atlantic Oscillation/annular mode: Two paradigms-one phenomenon. Q.J.R. Meteorol. Soc., 126 (564): 791-805. doi:10.1002/qj.49712656402.

8.17.7 Example plots
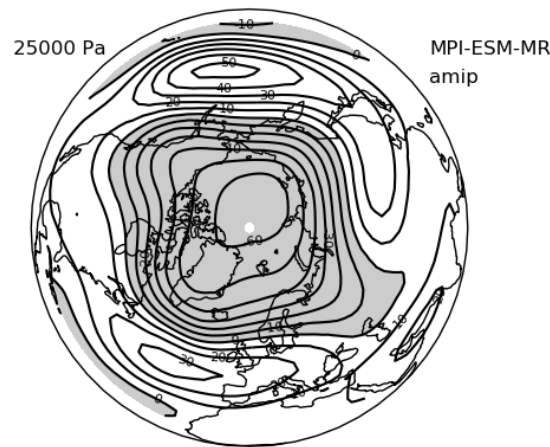
Fig. 30: Regression map of the zonal-mean NAM index onto geopotential height, for a selected pressure level (250 hPa) for the MPI-ESM-MR model (CMIP5 AMIP experiment, period 1979-2008). Negative values are shaded in grey.
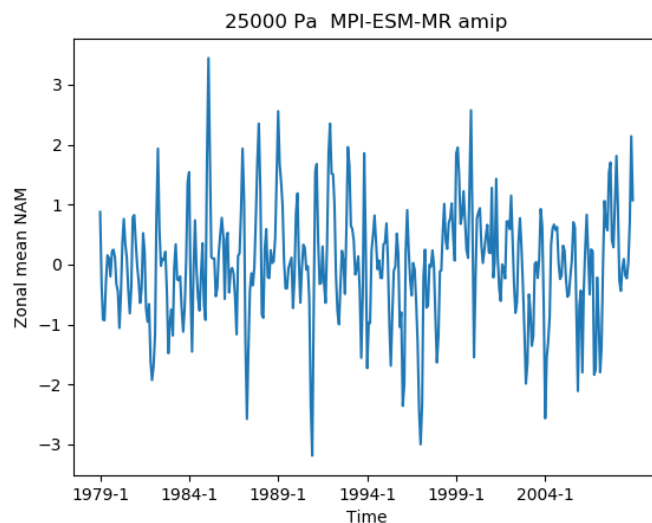
Fig. 31: Time series of the zonal-mean NAM index for a selected pressure level (250 hPa) for the MPI-ESM-MR model (CMIP5 AMIP experiment, period 1979-2008).

8.18 Thermodynamics of the Climate System - The Diagnostic Tool TheDiaTo v1.0

8.18.1 Overview

The tool allows to compute TOA, atmospheric and surface energy budgets, latent energy and water mass budgets, meridional heat transports, the Lorenz Energy Cycle (LEC), the material entropy production with the direct and indirect method.

The energy budgets are computed from monthly mean radiative and heat fluxes at the TOA and at the surface (cfr. Wild et al., 2013). The meridional heat transports are obtained from the latitudinal integration of the zonal mean energy budgets. When a land-sea mask is provided, results are also available for land and oceans, separately.

The water mass budget is obtained from monthly mean latent heat fluxes (for evaporation), total and snowfall precipitation (cfr. Liepert et al., 2012). The latent energy budget is obtained multiplying each component of the water mass budget by the respective latent heat constant. When a land-sea mask is provided, results are also available for land and oceans, separately.

The LEC is computed from 3D fields of daily mean velocity and temperature fields in the troposphere over pressure levels. The analysis is carried on in spectral fields, converting lonlat grids in Fourier coefficients. The components of the LEC are computed as in Ulbrich and Speth, 1991. In order to account for possible gaps in pressure levels, the daily fields of 2D near-surface temperature and horizontal velocities.

The material entropy production is computed by using the indirect or the direct method (or both). The former method relies on the convergence of radiative heat in the atmosphere (cfr. Lucarini et al., 2011; Pascale et al., 2011), the latter on all viscous and non-viscous dissipative processes occurring in the atmosphere (namely the sensible heat fluxes, the hydrological cycle with its components and the kinetic energy dissipation).

For a comprehensive report on the methods used and some descriptive results, please refer to Lembo et al., 2019.

8.18.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_thermodyn_diagtool.yml

Diagnostics are stored in diag_scripts/thermodyn_diagtool/

- thermodyn_diagnostics.py: the main script, handling input files, calling computation and plotting scricpts;

- computations.py: a module containing all the main computations that are carried out by the program;

- fluxogram.py: a module for the retrieval of the block diagrams displaying the reservoirs and conversion terms of the LEC

- fourier_coefficients.py: a module for the computation of the Fourier coefficients from the lonlat input grid

- lorenz_cycle.py: a module for the computation of the LEC components in Fourier coefficients

- mkthe.py: a module for the computation of indirect variables obtained from the input fields, such as LCL height, boundary layer top height and temperature, potential temperature

- plot_script.py: a module for the computation of maps, scatter plots, time series and meridional sections of some derived quantities for each model in the ensemble. The meridional heat and water mass transports are also computed here, as well as the peak magnitudes and locations;

- provenance_meta.py: a module for collecting metadata and writing them to produced outputs;

8.18.3 User settings

Besides the datasets, to be set according to usual ESMValTool convention, the user can set the following optional variables in the recipe_Thermodyn_diagtool.yml:

- wat: if set to 'true', computations are performed of the water mass and latent energy budgets and transports

- lsm: if set to true, the computations of the energy budgets, meridional energy transports, water mass and latent energy budgets and transports are performed separately over land and oceans

- lec: if set to 'true', computation of the LEC are performed

- entr: if set to 'true', computations of the material entropy production are performed

- met (1, 2 or 3): the computation of the material entropy production must be performed with the indirect method (1), the direct method (2), or both methods. If 2 or 3 options are chosen, the intensity of the LEC is needed for the entropy production related to the kinetic energy dissipation. If lec is set to 'false', a default value is provided.

These options apply to all models provided for the multi-model ensemble computations

8.18.4 Variables

- hfls (atmos, monthly mean, time latitude longitude)

- hfss (atmos, monthly mean, time latitude longitude)

- hus (atmos, monthly mean, time plev latitude longitude)

- pr (atmos, monthly mean, time latitude longitude)

- prsn (atmos, monthly mean, time latitude longitude)

- ps (atmos, monthly mean, time latitude longitude)

- rlds (atmos, monthly mean, time latitude longitude)

- rlus (atmos, monthly mean, time latitude longitude)

- rlut (atmos, monthly mean, time latitude longitude)

- rsds (atmos, monthly mean, time latitude longitude)

- rsdt (atmos, monthly mean, time latitude longitude)

- rsus (atmos, monthly mean, time latitude longitude)

- rsut (atmos, monthly mean, time latitude longitude)

- ta (atmos, daily mean, time plev latitude longitude)

- tas (atmos, daily mean, time latitude longitude)

- ts (atmos, monthly mean, time latitude longitude)

- ua (atmos, daily mean, time plev latitude longitude)

- uas (atmos, daily mean, time latitude longitude)

- va (atmos, daily mean, time plev latitude longitude)

- vas (atmos, daily mean, time latitude longitude)

- wap (atmos, daily mean, time plev latitude longitude)

8.18.5 References

- Lembo V, Lunkeit F, Lucarini V (2019) A new diagnostic tool for diagnosing water, energy and entropy budgets in climate models. Geophys Mod Dev Disc. doi:10.5194/gmd-2019-37. in review.

- Liepert BG, Previdi M (2012) Inter-model variability and biases of the global water cycle in CMIP3 coupled climate models. Environ Res Lett 7:014006. doi: 10.1088/1748-9326/7/1/014006

- Lorenz EN (1955) Available Potential Energy and the Maintenance of the General Circulation. Tellus 7:157–167. doi: 10.1111/j.2153-3490.1955.tb01148.x

- Lucarini V, Fraedrich K, Ragone F (2010) New Results on the Thermodynamical Properties of the Climate System. J Atmo 68:. doi: 10.1175/2011JAS3713.1

- Lucarini V, Blender R, Herbert C, et al (2014) Reviews of Geophysics Mathematical and physical ideas for climate science. doi: 10.1002/2013RG000446

- Pascale S, Gregory JM, Ambaum M, Tailleux R (2011) Climate entropy budget of the HadCM3 atmosphere–ocean general circulation model and of FAMOUS, its low-resolution version. Clim Dyn 36:1189–1206. doi: 10.1007/s00382-009-0718-1

- Ulbrich U, Speth P (1991) The global energy cycle of stationary and transient atmospheric waves: Results from ECMWF analyses. Meteorol Atmos Phys 45:125–138. doi: 10.1007/BF01029650

- Wild M, Folini D, Schär C, et al (2013) The global energy balance from a surface perspective. Clim Dyn 40:3107–3134. doi: 10.1007/s00382-012-1569-8

8.18.6 Example plots

8.19 Zonal and
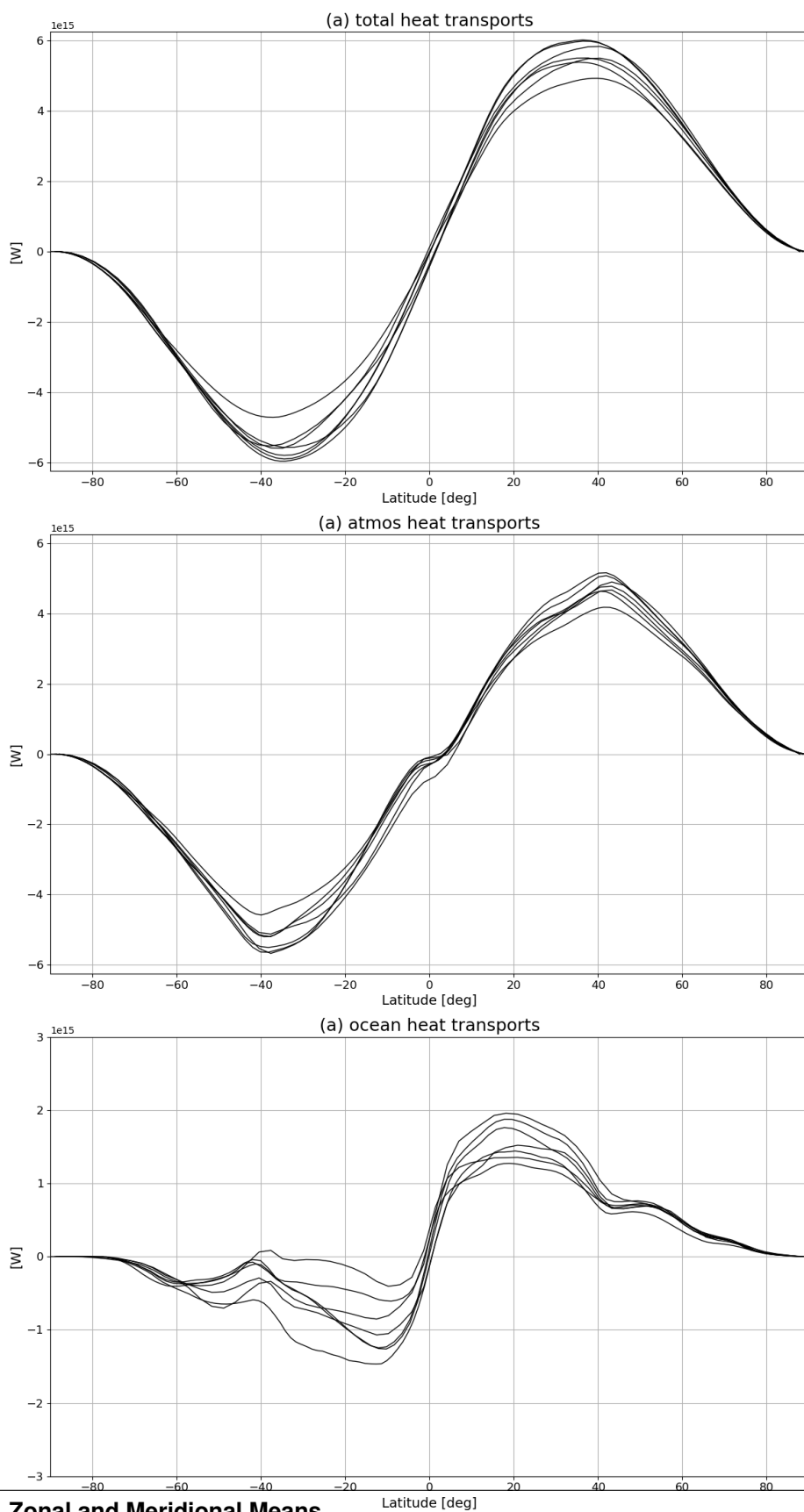
takes two models designated by CONTROL and EXPERIMENT and compares them via a number of analyses. Optionally a number of observational datasets can be added for processing. There are three types of standard analysis: lat_lon, meridional_mean and zonal_mean. Each of these diagnostics can be run on a separate basis (each an entry to diagnostics/scripts). The lat_lon analysis produces the following plots: a simple global plot for each variable for each dataset, a global plot for the difference between CONTROL and EXPERIMENT, a global plot for the difference between CONTROL and each of the observational datasets. The meridional_mean and zonal_mean produce variable vs coordinate (`latitude` or `longitude`) with both `CONTROL` and `EXPERIMENT` curves in each plot, for the entire duration of time specified and also, if the user wishes, for each season (seasonal means): winter DJF, spring MAM, summer JJA, autumn SON (by setting `seasonal_analysis:  true` in the recipe).

At least re-gridding on a common grid for all model and observational datasets should be performed in preprocessing (if datasets are on different grids). Also note that currently it is not allowed to use the same dataset (with varying parameters like experiment or ensemble) for both CONTROL and EXPERIMENT (the use case for comparison between different experiments or ensembles for the same model will be implemented in a future release).

8.19.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_validation.yml (CMIP5)

- recipe_validation_CMIP6.yml (CMIP6)

Diagnostics are stored in diag_scripts/

- validation.py

- shared/_validation.py

8.19.3 User settings

1. validation.py

   Required settings for script

- title: title of the analysis, user defined;

- control_model: control dataset name e.g. UKESM1-0-LL;

- exper_model: experiment dataset name e.g. IPSL-CM6A-LR;

- observational_datasets: list of at least one element; if no OBS

wanted comment out; e.g. ['ERA-Interim'];

- analysis_type: use any of: lat_lon, meridional_mean, zonal_mean;

- seasonal_analysis: boolean, if seasonal means are needed e.g. `true`;

- save_cubes: boolean, save each of the plotted cubes in `/work`;

8.19.4 Variables

- any variable

8.19.5 Observations and reformat scripts

Note: (1) obs4mips or OBS or ana4mips can be used.

- any observations

- it is important to note that all observational data should go through the same pre-processing as model data

8.19.6 References

- none, basic technical analysis

8.19.7 Example plots
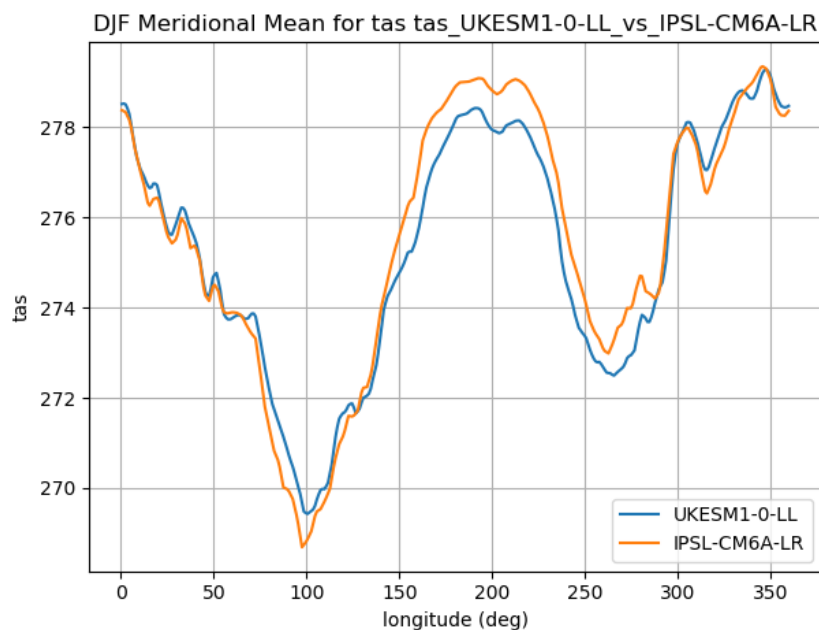
Fig. 32: Meridional seasonal mean for winter (DJF) comparison beween CMIP6 UKESM1 and IPSL models.
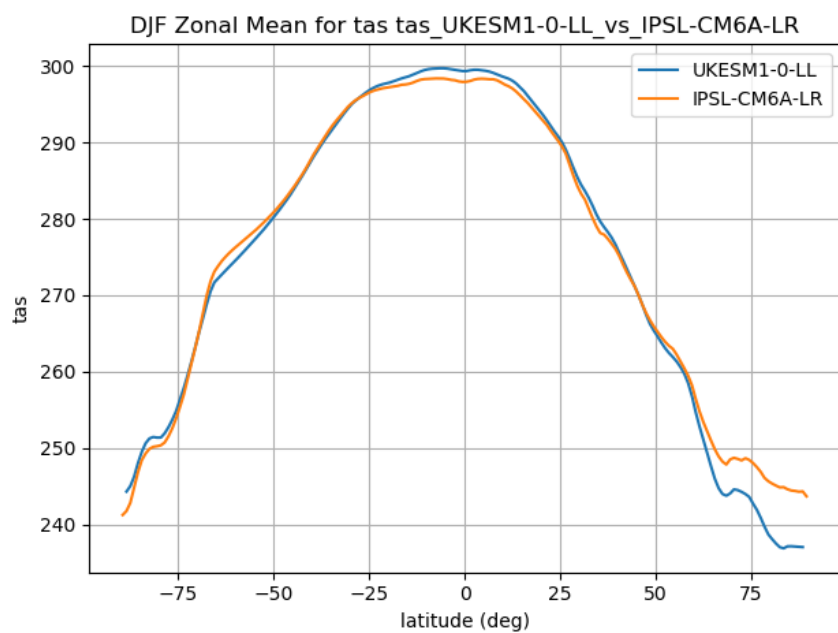
Fig. 33: Zonal seasonal mean for winter (DJF) comparison beween CMIP6 UKESM1 and IPSL models.

CLIMATE METRICS

9.1 Performance metrics for essential climate parameters

9.1.1 Overview

The goal is to create a standard recipe for the calculation of performance metrics to quantify the ability of the models to reproduce the climatological mean annual cycle for selected "Essential Climate Variables" (ECVs) plus some additional corresponding diagnostics and plots to better understand and interpret the results.

The recipe can be used to calculate performance metrics at different vertical levels (e.g., 5, 30, 200, 850 hPa as in Gleckler et al. (2008) and in different regions. As an additional reference, we consider Righi et al. (2015).

9.1.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_perfmetrics_CMIP5.yml

- recipe_perfmetrics_CMIP5_cds.yml

- recipe_perfmetrics_land_CMIP5.yml

Diagnostics are stored in diag_scripts/perfmetrics/

- main.ncl: calculates and (optionally) plots annual/seasonal cycles, zonal means, lat-lon fields and time-lat-lon fields. The calculated fields can also be plotted as difference w.r.t. a given reference dataset. main.ncl also calculates RMSD, bias and taylor metrics. Input data have to be regridded to a common grid in the preprocessor. Each plot type is created by a separated routine, as detailed below.

- cycle.ncl: creates an annual/seasonal cycle plot.

- zonal.ncl: creates a zonal (lat-pressure) plot.

- latlon.ncl: creates a lat-lon plot.

- cycle_latlon.ncl: precalculates the metrics for a time-lat-lon field, with different options for normalization.

- collect.ncl: collects and plots the metrics previously calculated by cycle_latlon.ncl.

9.1.3 User settings in recipe

1. Script main.ncl

   Required settings (scripts)

   - plot_type: cycle (time), zonal (plev, lat), latlon (lat, lon), cycle_latlon (time, lat, lon), cycle_zonal (time, plev, lat)
   - time_avg: type of time average (monthlyclim, seasonalclim, annualclim)
   - region: selected region (global, trop, nhext, shext, nhtrop, shtrop, nh, sh, nhmidlat, shmidlat, nhpolar, shpolar, eq)

   Optional settings (scripts)

   - styleset: for plot_type cycle only (cmip5, righi15gmd, cmip6, default)
   - plot_stddev: for plot_type cycle only, plots standard deviation as shading
   - legend_outside: for plot_type cycle only, plots the legend in a separate file
   - t_test: for plot_type zonal or latlon, calculates t-test in difference plots (default: False)
   - conf_level: for plot_type zonal or latlon, adds the confidence level for the t-test to the plot (default: False)
   - projection: map projection for plot_type latlon (default: CylindricalEquidistant)
   - plot_diff: draws difference plots (default: False)
   - calc_grading: calculates grading metrics (default: False)
   - stippling: uses stippling to mark statistically significant differences (default: False = mask out non-significant differences in gray)
   - show_global_avg: diplays the global avaerage of the input field as string at the top-right of lat-lon plots (default: False)
   - metric: chosen grading metric(s) (if calc_grading is True)
   - normalization: metric normalization (for RMSD and BIAS metrics only)
   - abs_levs: list of contour levels for absolute plot
   - diff_levs: list of contour levels for difference plot
   - zonal_cmap: for plot_type zonal only, chosen color table (default: "amwg_blueyellowred")
   - zonal_ymin: for plot_type zonal only, minimum pressure level on the y-axis (default: 5. hPa)
   - latlon_cmap: for plot_type latlon only, chosen color table (default: "amwg_blueyellowred")
   - plot_units: plotting units (if different from standard CMOR units)

   Required settings (variables)

   - reference_dataset: reference dataset to compare with (usually the observations).

   Optional settings (variables)

   - alternative_dataset: a second dataset to compare with.

   These settings are passed to the other scripts by main.ncl, depending on the selected plot_type.

2. Script collect.ncl

   Required settings (scripts)

   - metric: selected metric (RMSD, BIAS or taylor)

- label_bounds: for RMSD and BIAS metrics, min and max of the labelbar
- label_scale: for RMSD and BIAS metrics, bin width of the labelbar
- colormap: for RMSD and BIAS metrics, color table of the labelbar

Optional settings (scripts)

- label_lo: adds lower triange for values outside range
- label_hi: adds upper triange for values outside range
- cm_interval: min and max color of the color table
- cm_reverse: reverses the color table
- sort: sorts datasets in alphabetic order (excluding MMM)
- diag_order: sort diagnostics in a specific order (name = 'diagnostic'-'region')
- title: plots title
- scale_font: scaling factor applied to the default font size
- disp_values: switches on/off the grading values on the plot
- disp_rankings: switches on/off the rankings on the plot
- rank_order: displays rankings in increasing (1) or decreasing (-1) order

9.1.4 Variables

1. recipe_perfmetrics_CMIP5.yml
   - clt (atmos, monthly mean, longitude latitude time)
   - hus (atmos, monthly mean, longitude latitude lev time)
   - od550aer, od870aer, od550abs, od550lt1aer (aero, monthly mean, longitude latitude time)
   - pr (atmos, monthly mean, longitude latitude time)
   - rlut, rlutcs, rsut, rsutcs (atmos, monthly mean, longitude latitude time)
   - sm (land, monthly mean, longitude latitude time)
   - ta (atmos, monthly mean, longitude latitude lev time)
   - tas (atmos, monthly mean, longitude latitude time)
   - toz (atmos, monthly mean, longitude latitude time)
   - ts (atmos, monthly mean, longitude latitude time)
   - ua (atmos, monthly mean, longitude latitude lev time)
   - va (atmos, monthly mean, longitude latitude lev time)
   - zg (atmos, monthly mean, longitude latitude lev time)
2. recipe_perfmetrics_land_CMIP5.yml
   - sm (land, monthly mean, longitude latitude time)
   - nbp (land, monthly mean, longitude latitude time)
   - gpp (land, monthly mean, longitude latitude time)
   - lai (land, monthly mean, longitude latitude time)

- fgco2 (ocean, monthly mean, longitude latitude time)

- et (land, monthly mean, longitude latitude time)

- rlus, rlds, rsus, rdsd (atmos, monthly mean, longitude latitude time)

9.1.5 Observations and reformat scripts

The following list shows the currently used observational data sets for this recipe with their variable names and the reference to their respective reformat scripts in parentheses. Please note that obs4mips data can be used directly without any reformating. For non-obs4mips data see headers of cmorization scripts (in /esmvaltool/cmorizers/obs/) for downloading and processing instructions. #. recipe_perfmetrics_CMIP5.yml

- AIRS (hus - obs4mips)

- CERES-EBAF (rlut, rlutcs, rsut, rsutcs - obs4mips)

- ERA-Interim (tas, ta, ua, va, zg, hus - esmvaltool/cmorizers/obs/cmorize_obs_ERA-Interim.ncl)

- ESACCI-AEROSOL (od550aer, od870aer, od550abs, od550lt1aer - esmvaltool/cmorizers/obs/cmorize_obs_ESACCI-AEROSOL.ncl)

- ESACCI-CLOUD (clt - esmvaltool/cmorizers/obs/cmorize_obs_ESACCI-CLOUD.ncl)

- ESACCI-OZONE (toz - esmvaltool/cmorizers/obs/cmorize_obs_ESACCI-OZONE.ncl)

- ESACCI-SOILMOISTURE (sm - esmvaltool/cmorizers/obs/cmorize_obs_ESACCI-SOILMOISTURE.ncl)

- ESACCI-SST (ts - esmvaltool/ucmorizers/obs/cmorize_obs_ESACCI-SST.ncl)

- GPCP-SG (pr - obs4mips)

- HadISST (ts - esmvaltool/cmorizers/obs/cmorize_obs_HadISST.ncl)

- MODIS (od550aer - esmvaltool/cmorizers/obs/cmorize_obs_MODIS.ncl)

- NCEP (tas, ta, ua, va, zg - esmvaltool/cmorizers/obs/cmorize_obs_NCEP.ncl)

- NIWA-BS (toz - esmvaltool/cmorizers/obs/cmorize_obs_NIWA-BS.ncl)

- PATMOS-x (clt - esmvaltool/cmorizers/obs/cmorize_obs_PATMOS-x.ncl)

1. recipe_perfmetrics_land_CMIP5.yml

    - CERES-EBAF (rlus, rlds, rsus, rsds - obs4mips)

    - ESACCI-SOILMOISTURE (sm - esmvaltool/cmorizers/obs/cmorize_obs_ESACCI-SOILMOISTURE.ncl)

    - FLUXCOM (gpp - esmvaltool/cmorizers/obs/cmorize_obs_fluxcom.py)

    - JMA-TRANSCOM (nbp, fgco2 - esmvaltool/cmorizers/obs/cmorize_obs_jma_transcom.py)

    - LAI3d (lai - esmvaltool/cmorizers/obs/cmorize_obs_lai3g.py)

    - LandFlux-EVAL (et - esmvaltool/cmorizers/obs/cmorize_obs_landflux_eval.py)

    - Landschuetzer2016 (fgco2 - esmvaltool/cmorizers/obs/cmorize_obs_landschuetzer2016.py)

    - MTE (gpp - esmvaltool/cmorizers/obs/cmorize_obs_mte.py)

9.1.6 References

- Gleckler, P. J., K. E. Taylor, and C. Doutriaux, Performance metrics for climate models, J. Geophys. Res., 113, D06104, doi: 10.1029/2007JD008972 (2008).

- Righi, M., Eyring, V., Klinger, C., Frank, F., Gottschaldt, K.-D., Jöckel, P., and Cionni, I.: Quantitative evaluation of oone and selected climate parameters in a set of EMAC simulations, Geosci. Model Dev., 8, 733, doi: 10.5194/gmd-8-733-2015 (2015).
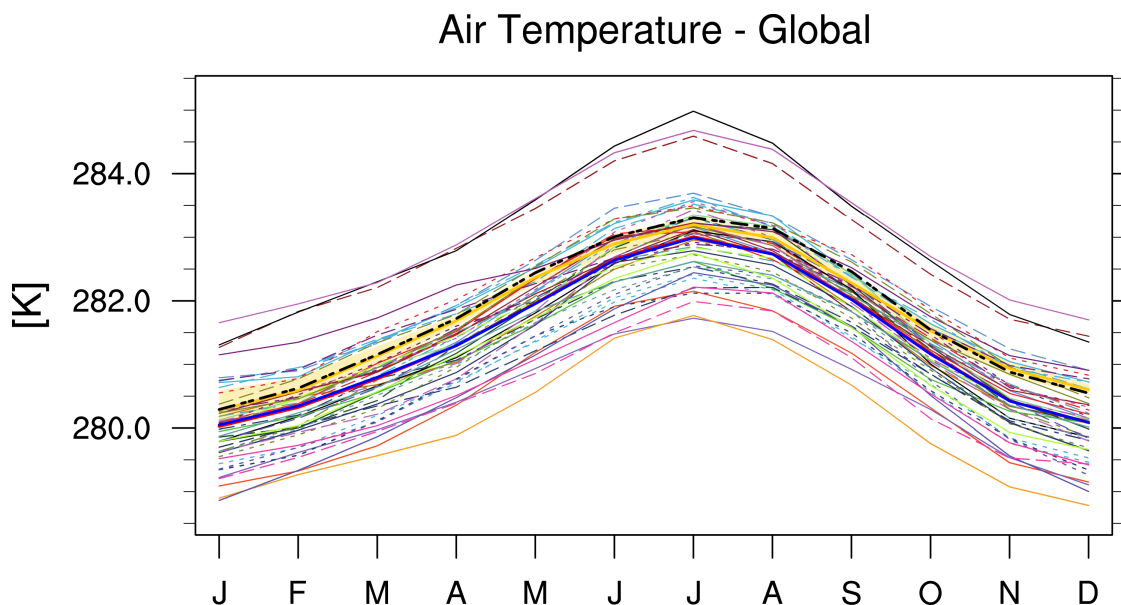
9.1.7 Example plots

Fig. 1: Annual cycle of globally averaged temperature at 850 hPa (time period 1980-2005) for different CMIP5 models (historical simulation) (thin colored lines) in comparison to ERA-Interim (thick yellow line) and NCEP (thick black dashed line) reanalysis data.

9.2 Single Model Performance Index (SMPI)

9.2.1 Overview

This diagnostic calculates the Single Model Performance Index (SMPI) following Reichler and Kim (2008). The SMPI (called "I^2") is based on the comparison of several different climate variables (atmospheric, surface and oceanic) between climate model simulations and observations or reanalyses, and it focuses on the validation of the time-mean state of climate. For I^2 to be determined, the differences between the climatological mean of each model variable and observations at each of the available data grid points are calculated, and scaled to the interannual variance from the validating observations. This interannual variability is determined by performing a bootstrapping method (random selection with replacement) for the creation of a large synthetic ensemble of observational climatologies. The results are then scaled to the average error from a reference ensemble of models, and in a final step the mean over all climate variables and one model is calculated. The plot shows the I^2 values for each model (orange circles) and the multi-model mean (black circle), with the diameter of each circle representing the range of I^2 values encompassed by the 5th and 95th percentiles of the bootstrap ensemble. The I^2 values vary around one, with values greater than one for underperforming models, and values less than one for more accurate models.
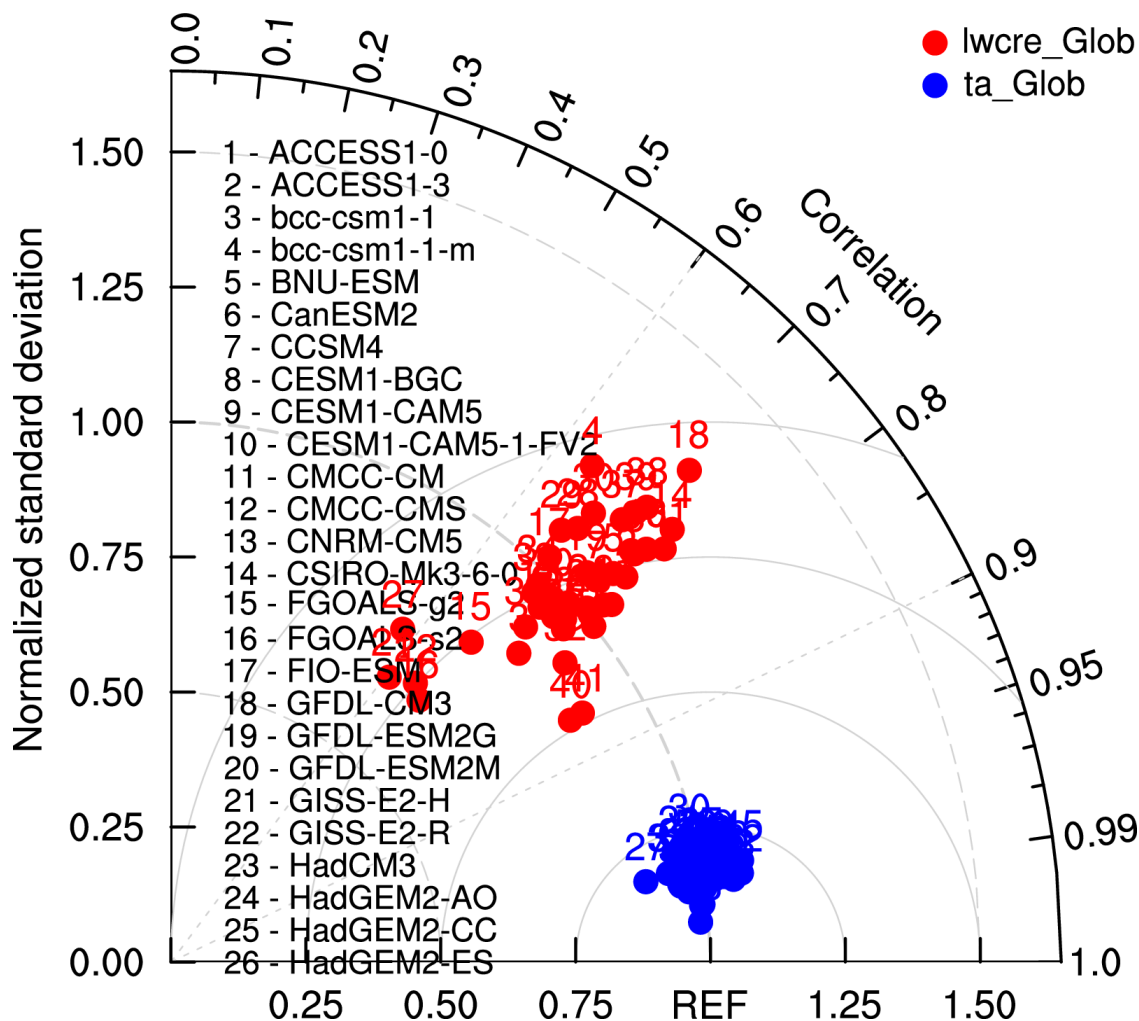
Fig. 2: Taylor diagram of globally averaged temperature at 850 hPa (ta) and longwave cloud radiative effect (lwcre) for different CMIP5 models (historical simulation, 1980-2005). Reference data (REF) are ERA-Interim for temperature (1980-2005) and CERES-EBAF (2001-2012) for longwave cloud radiative effect.
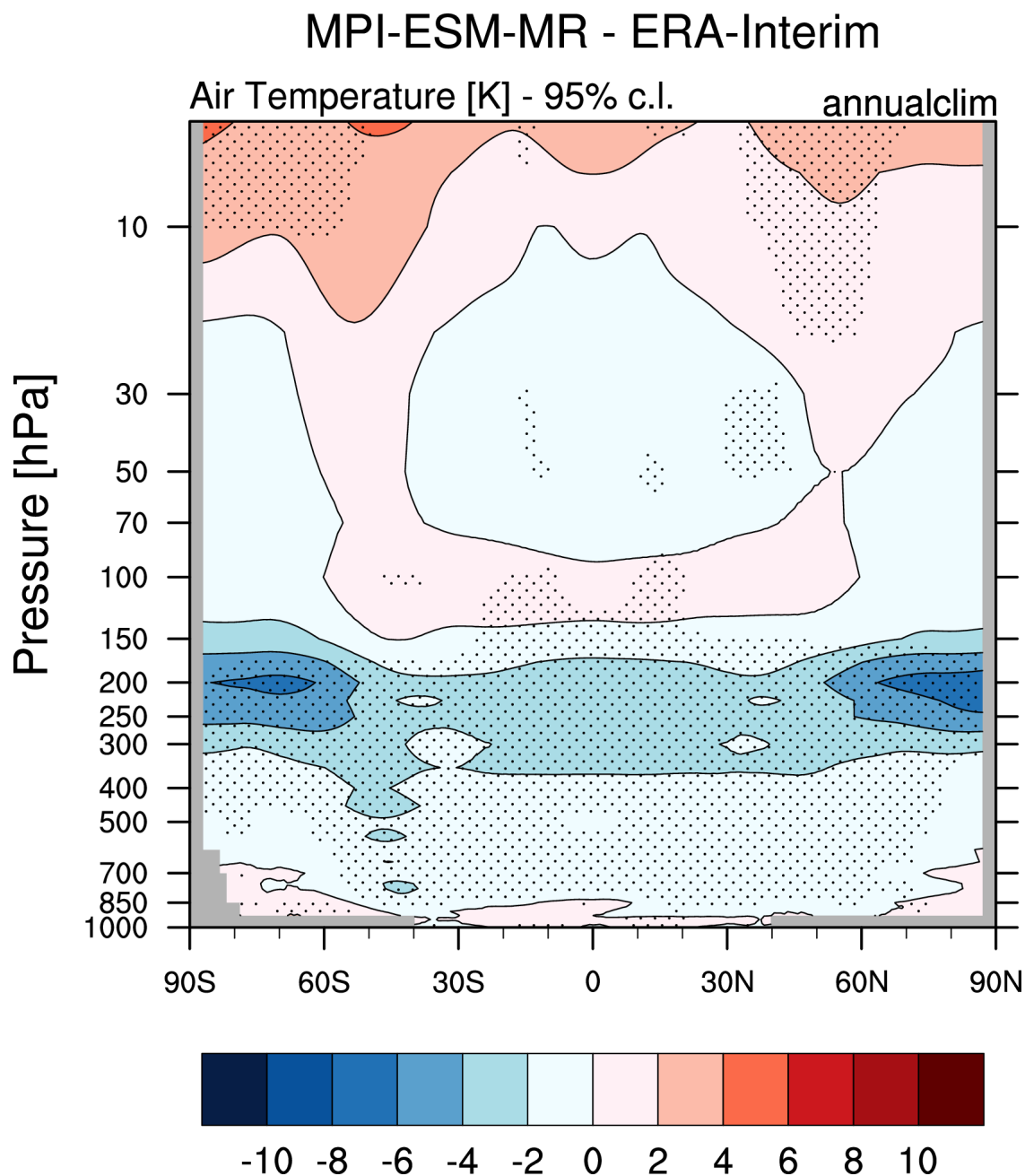
Fig. 3: Difference in annual mean of zonally averaged temperature (time period 1980-2005) between the CMIP5 model MPI-ESM-MR (historical simulation) and ERA-Interim. Stippled areas indicdate differences that are statistically significant at a 95% confidence level.
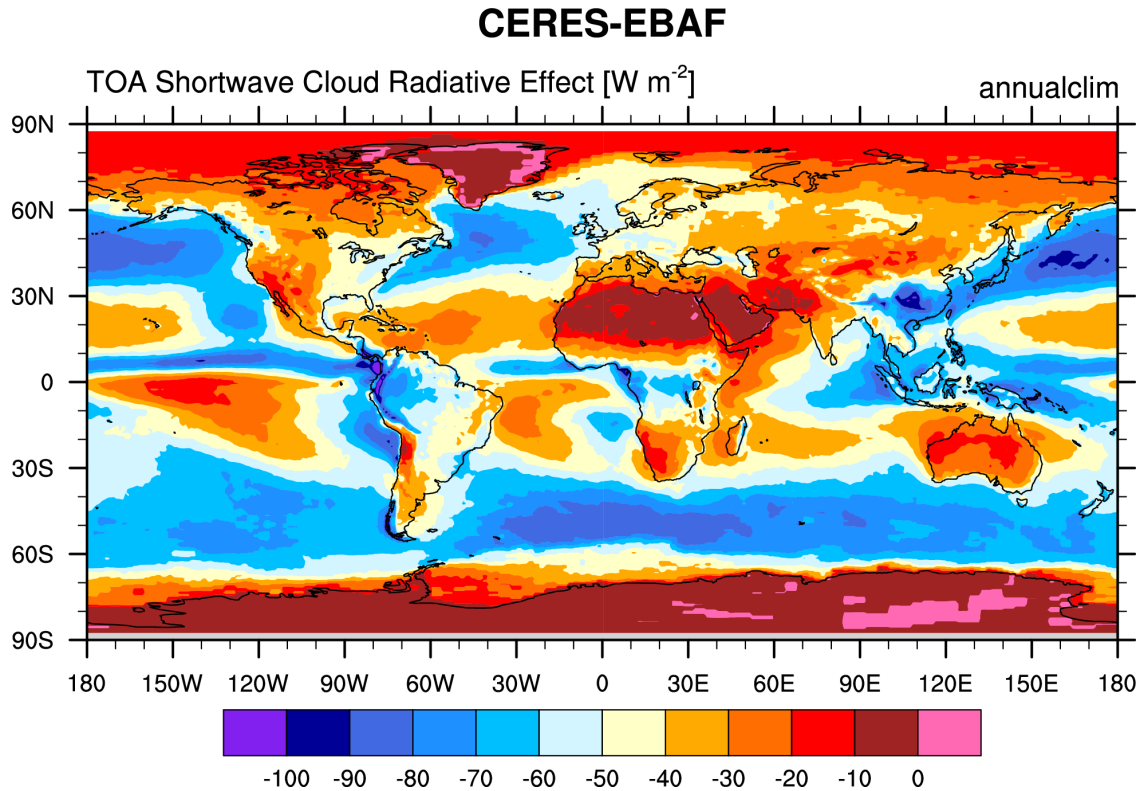
Fig. 4: Annual mean (2001-2012) of the shortwave cloud radiative effect from CERES-EBAF.

Note: The SMPI diagnostic needs all indicated variables from all added models for exactly the same time period to be calculated correctly. If one model does not provide a specific variable, either that model cannot be added to the SMPI calculations, or the missing variable has to be removed from the diagnostics all together.

9.2.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_smpi.yml
- recipe_smpi_4cds.yml

Diagnostics are stored in diag_scripts/perfmetrics/

- main.ncl: calculates and (optionally) plots annual/seasonal cycles, zonal means, lat-lon fields and time-lat-lon fields. The calculated fields can also be plotted as difference w.r.t. a given reference dataset. main.ncl also calculates RMSD, bias and taylor metrics. Input data have to be regridded to a common grid in the preprocessor. Each plot type is created by a separated routine, as detailed below.

- cycle_zonal.ncl: calculates single model perfomance index (Reichler and Kim, 2008). It requires fields precalculated by main.ncl.

- collect.ncl: collects the metrics previously calculated by cycle_latlon.ncl and passes them to the plotting functions.
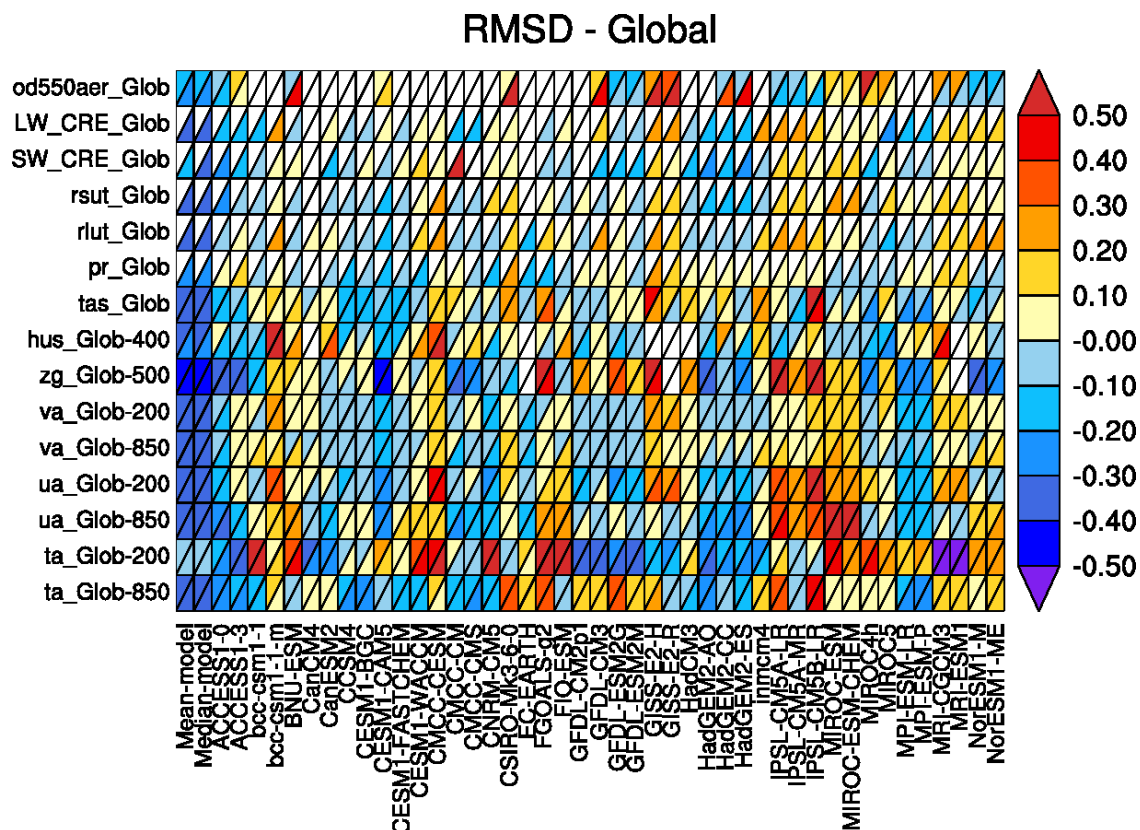
Fig. 5: Relative space-time root-mean-square deviation (RMSD) calculated from the climatological seasonal cycle of CMIP5 simulations. A relative performance is displayed, with blue shading indicating better and red shading indicating worse performance than the median of all model results. A diagonal split of a grid square shows the relative error with respect to the reference data set (lower right triangle) and the alternative data set (upper left triangle). White boxes are used when data are not available for a given model and variable.

9.2.3 User settings

1. perfmetrics/main.ncl

   Required settings for script

   - plot_type: only "cycle_latlon (time, lat, lon)" and "cycle_zonal (time, plev, lat)" available for SMPI; usage is defined in the recipe and is dependent on the used variable (2D variable: cycle_latlon, 3D variable: cycle_zonal)

   - time_avg: type of time average (only "yearly" allowed for SMPI, any other settings are not supported for this diagnostic)

   - region: selected region (only "global" allowed for SMPI, any other settings are not supported for this diagnostic)

   - normalization: metric normalization ("CMIP5" for analysis of CMIP5 simulations; to be adjusted accordingly for a different CMIP phase)

   - calc_grading: calculates grading metrics (has to be set to "true" in the recipe)

   - metric: chosen grading metric(s) (if calc_grading is True; has to be set to "SMPI")

   - smpi_n_bootstrap: number of bootstrapping members used to determine uncertainties on model-reference differences (typical number of bootstrapping members: 100)

   Required settings for variables

   - reference_dataset: reference dataset to compare with (usually the observations).

These settings are passed to the other scripts by main.ncl, depending on the selected plot_type.

1. collect.ncl

   Required settings for script

   - metric: selected metric (has to be "SMPI")

9.2.4 Variables

- hfds (ocean, monthly mean, longitude latitude time)
- hus (atmos, monthly mean, longitude latitude lev time)
- pr (atmos, monthly mean, longitude latitude time)
- psl (atmos, monthly mean, longitude latitude time)
- sic (ocean-ice, monthly mean, longitude latitude time)
- ta (atmos, monthly mean, longitude latitude lev time)
- tas (atmos, monthly mean, longitude latitude time)
- tauu (atmos, monthly mean, longitude latitude time)
- tauv (atmos, monthly mean, longitude latitude time)
- tos (ocean, monthly mean, longitude latitude time)
- ua (atmos, monthly mean, longitude latitude lev time)
- va (atmos, monthly mean, longitude latitude lev time)

9.2.5 Observations and reformat scripts

The following list shows the currently used observational data sets for this recipe with their variable names and the reference to their respective reformat scripts in parentheses. Please note that obs4mips data can be used directly without any reformating. For non-obs4mips data see headers of cmorization scripts (in /esmvaltool/cmorizers/obs/) for downloading and processing instructions.

- ERA-Interim (hfds, hus, psl, ta, tas, tauu, tauv, ua, va - esmvaltool/utils/cmorizers/obs/cmorize_obs_ERA-Interim.ncl)

- HadISST (sic, tos - reformat_scripts/obs/reformat_obs_HadISST.ncl)

- GPCP-SG (pr - obs4mips)

9.2.6 References

- Reichler, T. and J. Kim, How well do coupled models simulate today's climate? Bull. Amer. Meteor. Soc., 89, 303-311, doi: 10.1175/BAMS-89-3-303, 2008.

9.2.7 Example plots
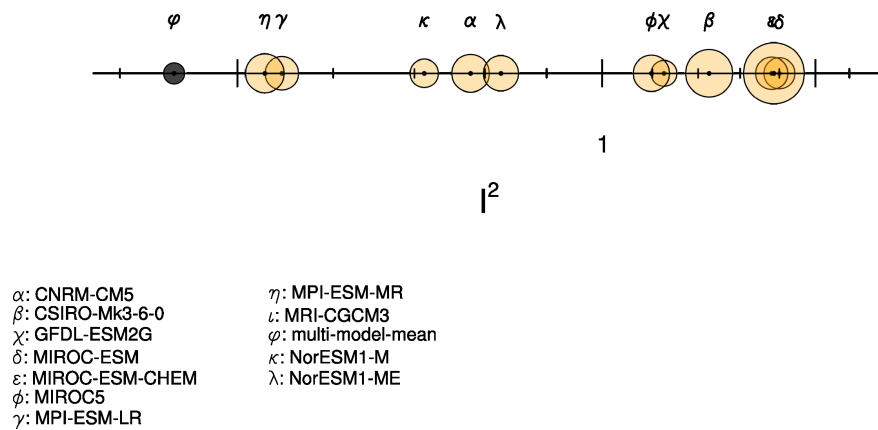
Fig. 6: Performance index I^2 for individual models (circles). Circle sizes indicate the length of the 95% confidence intervals. The black circle indicates the I^2 of the multi-model mean (similar to Reichler and Kim (2008), Figure 1).

FUTURE PROJECTIONS

10.1 Emergent constraints for equilibrium climate sensitivity

10.1.1 Overview

Calculates equilibrium climate sensitivity (ECS) versus

1) S index, D index and lower tropospheric mixing index (LTMI); similar to fig. 5 from Sherwood et al. (2014)

2) southern ITCZ index and tropical mid-tropospheric humidity asymmetry index; similar to fig. 2 and 4 from Tian (2015)

3) covariance of shortwave cloud reflection (Brient and Schneider, 2016)

4) climatological Hadley cell extent (Lipat et al., 2017)

5) temperature variability metric; similar to fig. 2 from Cox et al. (2018)

6) total cloud fraction difference between tropics and mid-latitudes; similar to fig. 3 from Volodin (2008)

7) response of marine boundary layer cloud (MBLC) fraction changes to sea surface temperature (SST); similar to fig. 3 of Zhai et al. (2015)

8) Cloud shallowness index (Brient et al., 2016)

9) Error in vertically-resolved tropospheric zonal average relative humidity (Su et al., 2014)

The results are displayed as scatterplots.

Note: The recipe `recipe_ecs_scatter.yml` requires pre-calulation of the equilibrium climate sensitivites (ECS) for all models. The ECS values are calculated with recipe_ecs.yml. The netcdf file containing the ECS values (path and filename) is specified by [diag_script_info@ecs_file](). Alternatively, the netcdf file containing the ECS values can be generated with the cdl-script $diag_scripts/emergent_constraints/ecs_cmip.cdl (recommended method):

1) save script given at the end of this recipe as ecs_cmip.cdl

2) run command: ncgen -o ecs_cmip.nc ecs_cmip.cdl

3) copy ecs_cmip.nc to directory given by [diag_script_info@ecs_file]() (e.g. $diag_scripts/emergent_constraints/ecs_cmip.nc)

10.1.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_ecs_scatter.yml
- recipe_ecs_constraints.yml

Diagnostics are stored in diag_scripts

- emergent_constraints/ecs_scatter.ncl: calculate emergent constraints for ECS
- emergent_constraints/ecs_scatter.py: calculate further emergent constraints for ECS
- emergent_constraints/univariate_constraint.py: create scatterplots for emergent constraints
- climate_metrics/psi.py: calculate temperature variabililty metric (Cox et al., 2018)

10.1.3 User settings in recipe

1. Script emergent_constraints/ecs_scatter.ncl

   Required settings (scripts)

   - diag: emergent constraint to calculate ("itczidx", "humidx", "ltmi", "covrefl", "shhc", "sherwood_d", "sherwood_s")
   - ecs_file: path and filename of netCDF containing precalculated ECS values (see note above)

   Optional settings (scripts)

   - calcmm: calculate multi-model mean (True, False)
   - legend_outside: plot legend outside of scatterplots (True, False)
   - output_diag_only: Only write netcdf files for X axis (True) or write all plots (False)
   - output_models_only: Only write models (no reference datasets) to netcdf files (True, False)
   - output_attributes: Additonal attributes for all output netcdf files
   - predef_minmax: use predefined internal min/max values for axes (True, False)
   - styleset: "CMIP5" (if not set, diagnostic will create a color table and symbols for plotting)
   - suffix: string to add to output filenames (e.g."cmip3")

   Required settings (variables)

   - reference_dataset: name of reference data set

   Optional settings (variables)

   Color tables

2. Script emergent_constraints/ecs_scatter.py

   Required settings (scripts)

   - diag: emergent constraint to calculate ("brient_shal", "su", "volodin", "zhai")

   Optional settings (scripts)

- metric: Metric to measure model error, only relevant for Su et al. (2014) constraint ("regression_slope", "correlation_coefficient")

- output_attributes: Additonal attributes for all output netcdf files

- pattern: Only accept ancestor files that match that pattern

- savefig_kwargs: Keyword arguments for matplotlib.pyplot's `savefig()` function, see https://matplotlib.org/3.2.0/api/_as_gen/matplotlib.pyplot.savefig.html

- seaborn_settings: Options for seaborn's `set()` methods (affects all plots), see https://seaborn.pydata.org/generated/seaborn.set.html

Required settings (variables)

- reference_dataset: name of reference data set

Optional settings (variables)

3. Script emergent_constraints/univariate_constraint.py

   All input data for this diagnostic must be marked with a `var_type` (either `feature`, `label`, `prediction_input` or `prediction_input_error`) and a `tag`, which describes the data. This diagnostic supports only a single `tag` for `label` and `feature`. For every `tag`, a `reference_dataset` can be specified, which will be automatically considered as `prediction_input`. If `reference_dataset` contains `'|'` (e.g. `'OBS1|OBS2'`), mutliple datasets are considered as `prediction_input` (in this case `'OBS1'` and `'OBS2'`).

   Required settings (scripts)

   Optional settings (scripts)

   - all_data_label: Label used in plots when all input data is considered. Only relevant if `group_by` is not used

   - confidence_level: Confidence level for estimation of constrained target variable.

   - group_by: Group input data by an attribute (e.g. produces separate plots for the individual groups, etc.)

   - ignore_patterns: Ignore ancestor files that match that patterns

   - merge_identical_pred_input: Use identical prediction_input values as single value

   - numbers_as_markers: Use numbers as markers in scatterplots

   - patterns: Only accept ancestor files that match that patterns

   - read_external_file: Read input datasets from external file given as absolute path or relative path. In the latter case, `'auxiliary_data_dir'` from the user configuration file is used as base directory

   - savefig_kwargs: Keyword arguments for matplotlib.pyplot's `savefig()` function, see https://matplotlib.org/3.2.0/api/_as_gen/matplotlib.pyplot.savefig.html

   - seaborn_settings: Options for seaborn's `set()` methods (affects all plots), see https://seaborn.pydata.org/generated/seaborn.set.html

   Required settings (variables)

   - reference_dataset: name of reference data set

   Optional settings (variables)

10.1. Emergent constraints for equilibrium climate sensitivity **113**

4. Script climate_metrics/psi.py

   See *Emergent constraint on equilibrium climate sensitivity from global temperature variability*.

10.1.4 Variables

- cl (atmos, monthly mean, longitude latitude level time)
- clt (atmos, monthly mean, longitude latitude time)
- pr (atmos, monthly mean, longitude latitude time)
- hur (atmos, monthly mean, longitude latitude level time)
- hus (atmos, monthly mean, longitude latitude level time)
- rsdt (atmos, monthly mean, longitude latitude time)
- rsut (atmos, monthly mean, longitude latitude time)
- rsutcs (atmos, monthly mean, longitude latitude time)
- rtnt or rtmt (atmos, monthly mean, longitude latitude time)
- ta (atmos, monthly mean, longitude latitude level time)
- tas (atmos, monthly mean, longitude latitude time)
- tasa (atmos, monthly mean, longitude latitude time)
- tos (atmos, monthly mean, longitude latitude time)
- ts (atmos, monthly mean, longitude latitude time)
- va (atmos, monthly mean, longitude latitude level time)
- wap (atmos, monthly mean, longitude latitude level time)
- zg (atmos, monthly mean, longitude latitude time)

10.1.5 Observations and reformat scripts

Note:

(1) Obs4mips data can be used directly without any preprocessing.

(2) See headers of reformat scripts for non-obs4mips data for download instructions.

- AIRS (obs4mips): hus, husStderr
- AIRS-2-0 (obs4mips): hur
- CERES-EBAF (obs4mips): rsdt, rsut, rsutcs
- ERA-Interim (OBS6): hur, ta, va, wap
- GPCP-SG (obs4mips): pr
- HadCRUT4 (OBS): tasa
- HadISST (OBS): ts
- MLS-AURA (OBS6): hur

- TRMM-L3 (obs4mips): pr, prStderr

10.1.6 References

- Brient, F., and T. Schneider, J. Climate, 29, 5821-5835, doi:10.1175/JCLIM-D-15-0897.1, 2016.

- Brient et al., Clim. Dyn., 47, doi:10.1007/s00382-015-2846-0, 2016.

- Cox et al., Nature, 553, doi:10.1038/nature25450, 2018.

- Gregory et al., Geophys. Res. Lett., 31, doi:10.1029/2003GL018747, 2004.

- Lipat et al., Geophys. Res. Lett., 44, 5739-5748, doi:10.1002/2017GL73151, 2017.

- Sherwood et al., nature, 505, 37-42, doi:10.1038/nature12829, 2014.

- Su, et al., J. Geophys. Res. Atmos., 119, doi:10.1002/2014JD021642, 2014.

- Tian, Geophys. Res. Lett., 42, 4133-4141, doi:10.1002/2015GL064119, 2015.

- Volodin, Izvestiya, Atmospheric and Oceanic Physics, 44, 288-299, doi:10.1134/S0001433808030043, 2008.

- Zhai, et al., Geophys. Res. Lett., 42, doi:10.1002/2015GL065911, 2015.

10.1.7 Example plots

10.2 Emergent constraints on carbon cycle feedbacks

10.2.1 Overview

Figures from Wenzel et al. (2014) are reproduced with recipe_wenzel14jgr.yml. Variables relevant for the carbon cycle - climate feedback such as near surface air temperature (tas), net biosphere productivity (nbp) and carbon flux into the ocean (fgco2) are analyzed for coupled (1pctCO2, here the carbon cycle is fully coupled to the climate response) and uncoupled (esmFixCLim1, here the carbon cycle is uncoupled to the climate response) simulations. The standard namelist includes a comparison of cumulated nbp from coupled and uncoupled simulations and includes a set of routines to diagnose the long-term carbon cycle - climate feedback parameter (GammaLT) from an ensemble of CMIP5 models. Also included in the recipe is a comparison of the interannual variability of nbp and fgco2 for historical simulations used to diagnose the observable sensitivity of CO2 to tropical temperature changes (GammaIAV). As a key figure of this recipe, the diagnosed values from the models GammaLT vs. GammaIAV are compared in a scatter plot constituting an emergent constraint.

10.2.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_wenzel14jgr.yml

Diagnostics are stored in diag_scripts/

- carbon_tsline.ncl: time line plots of annual means for spatial averages

- carbon_gammaHist.ncl: scatter plot of annual mean anomalies of two different variables; diagnosing and saving GammaIAV

- carbon_constraint.ncl: scatter plot of GammaLT vs. GammaIAV + line plot of probability density functions, diagnosing GammaLT
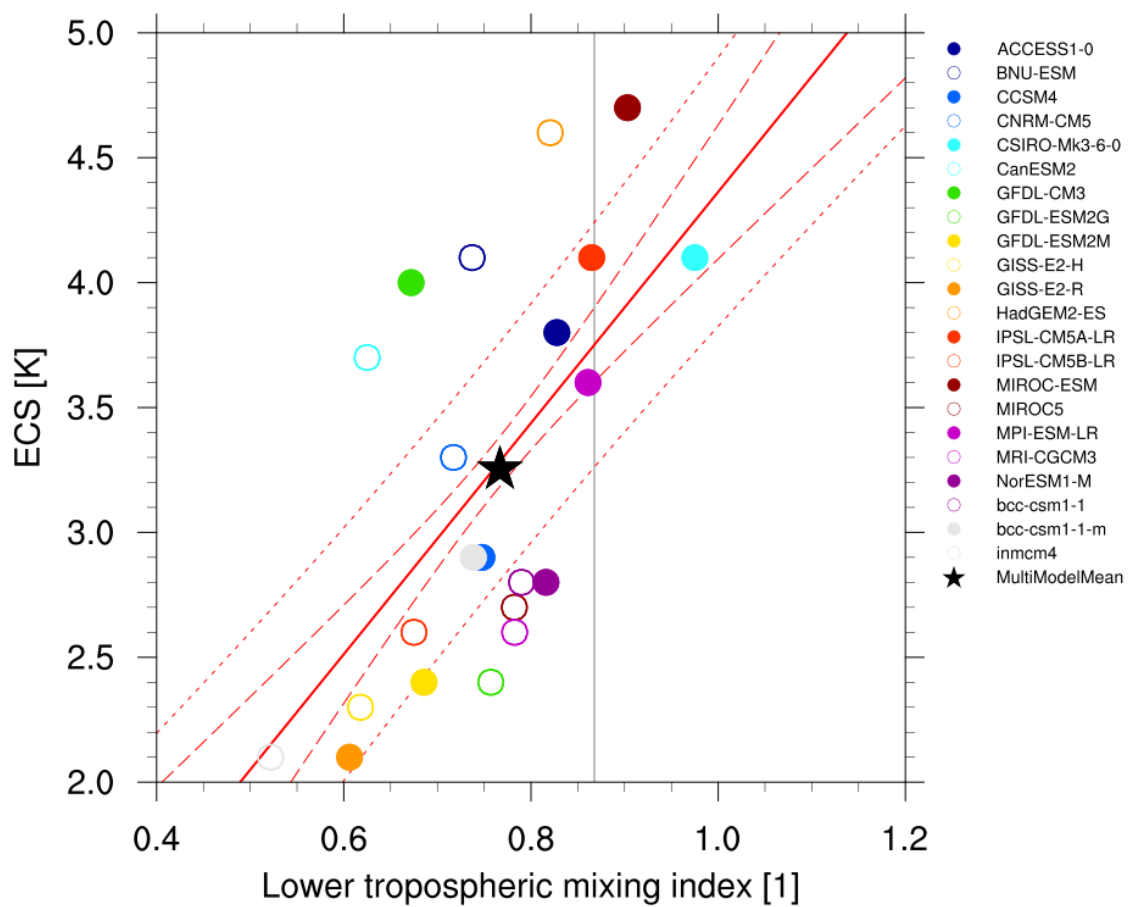
Fig. 1: Lower tropospheric mixing index (LTMI; Sherwood et al., 2014) vs. equilibrium climate sensitivity from CMIP5 models.
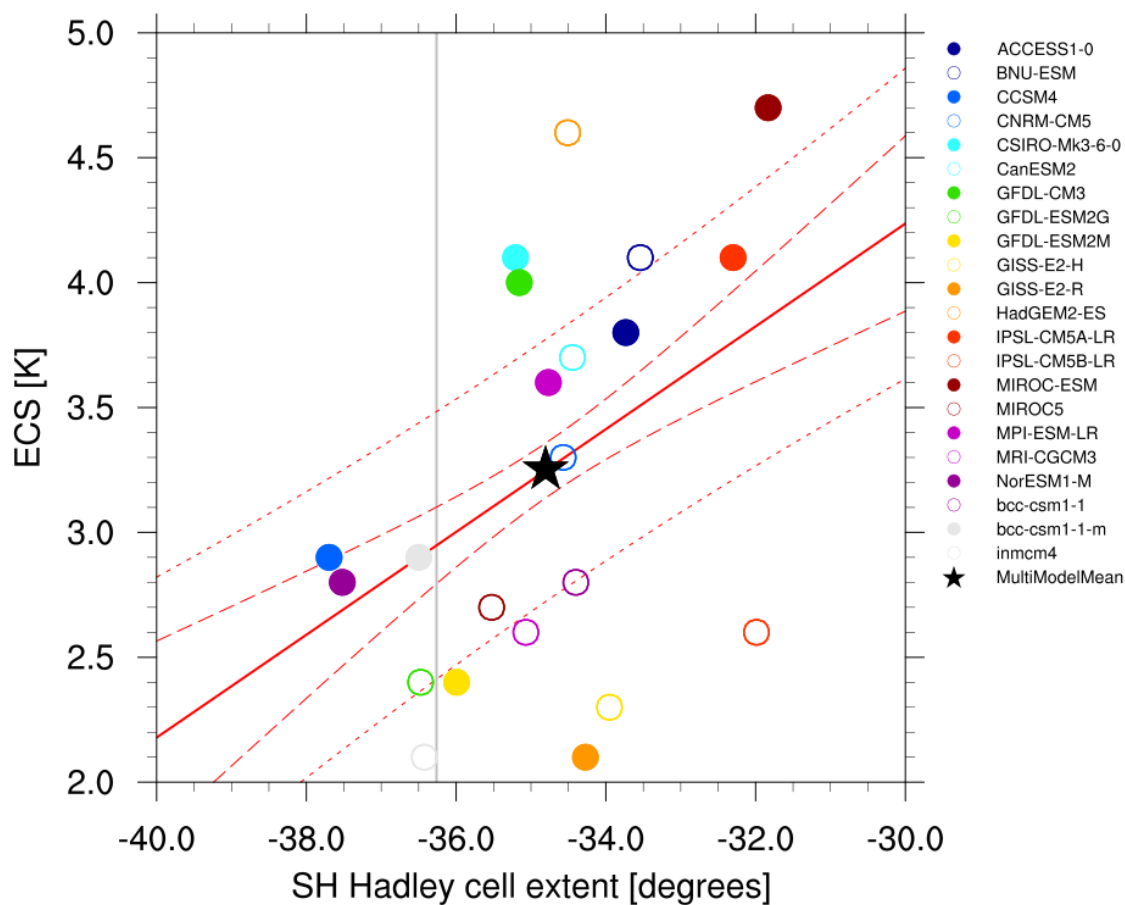
Fig. 2: Climatological Hadley cell extent (Lipat et al., 2017) vs. equilibrium climate sensitivity from CMIP5 models.
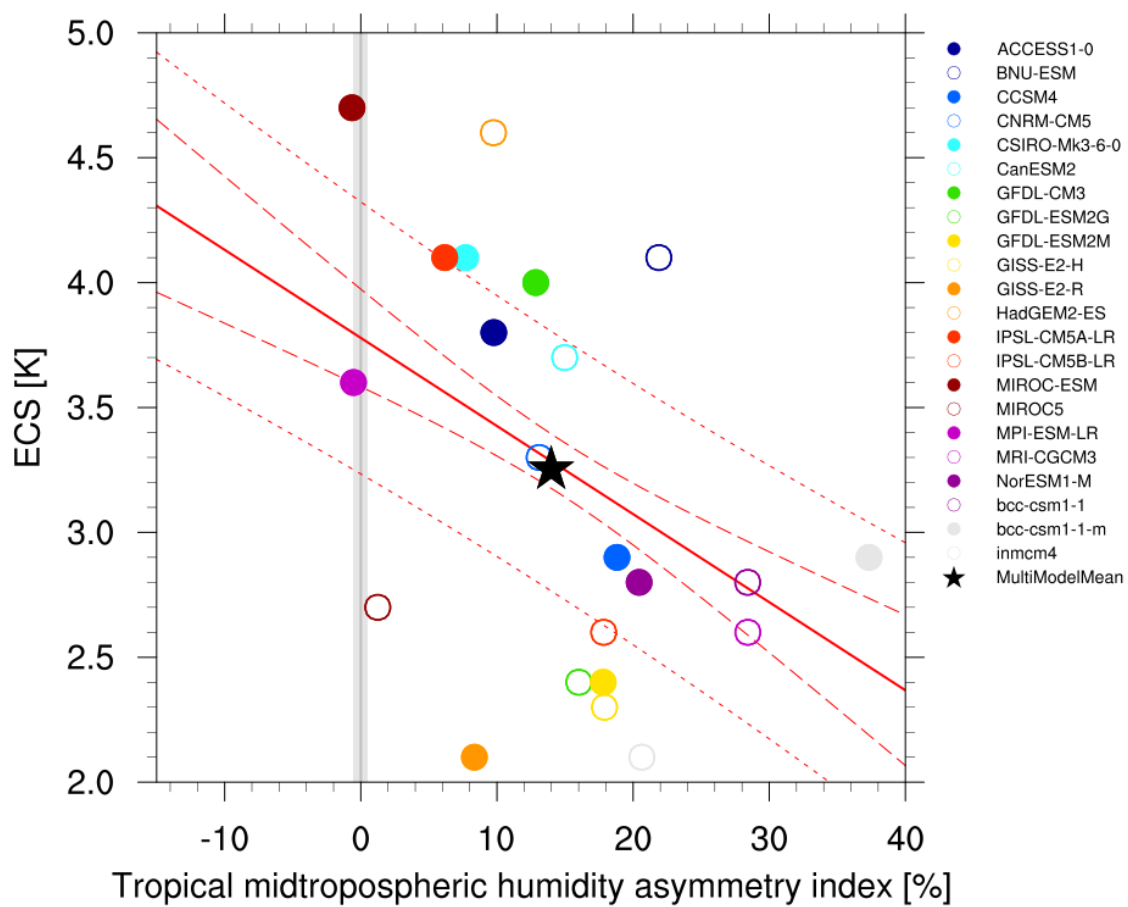
Fig. 3: Tropical mid-tropospheric humidity asymmetry index (Tian, 2015) vs. equilibrium climate sensitivity from CMIP5 models.
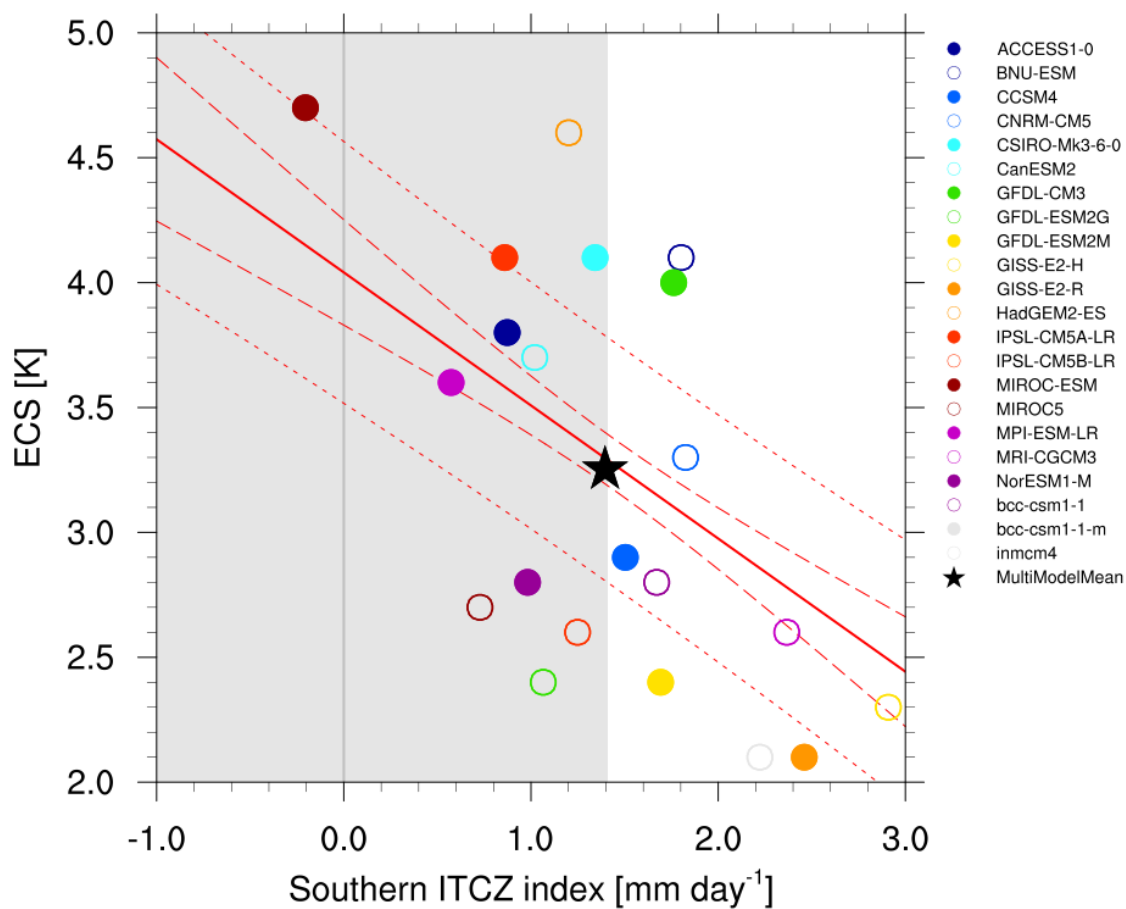
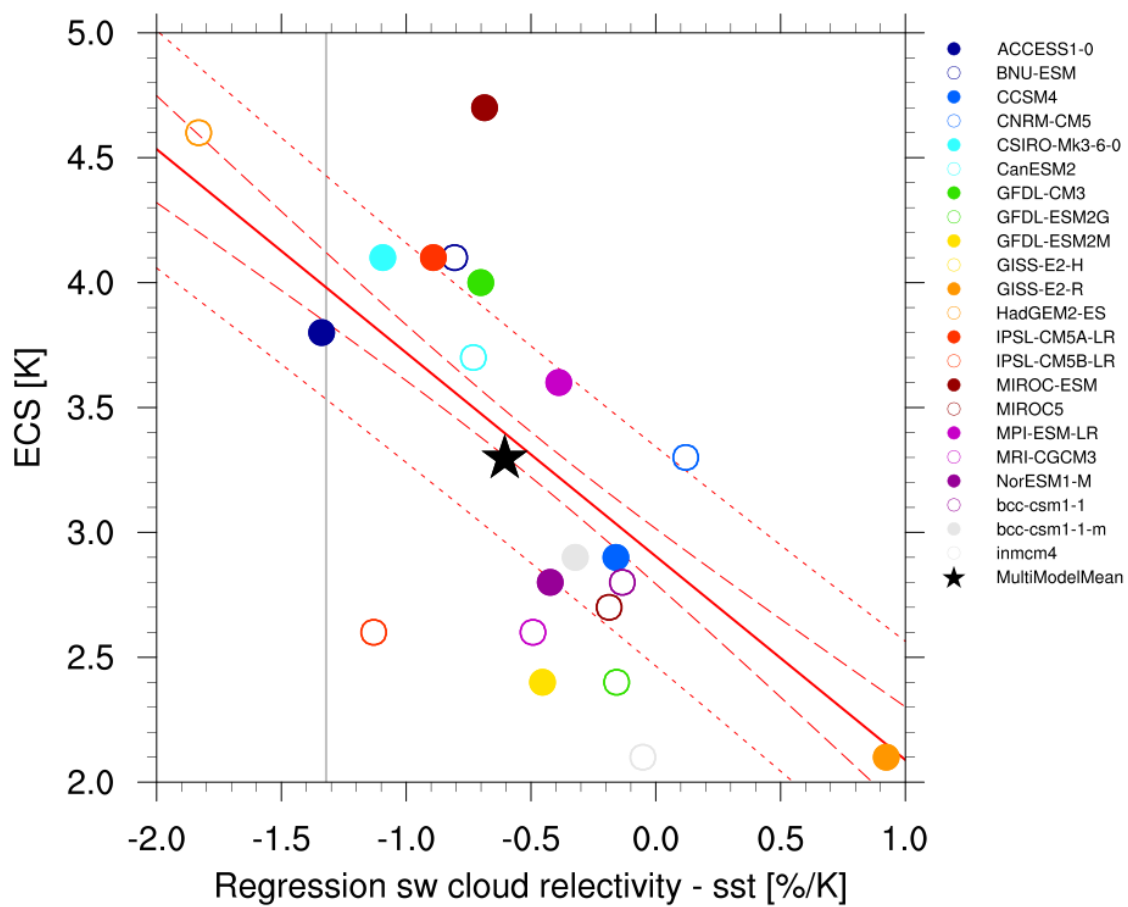Fig. 4: Southern ITCZ index (Tian, 2015) vs. equilibrium climate sensitivity from CMIP5 models.

Fig. 5: Covariance of shortwave cloud reflection (Brient and Schneider, 2016) vs. equilibrium climate sensitivity from CMIP5 models.
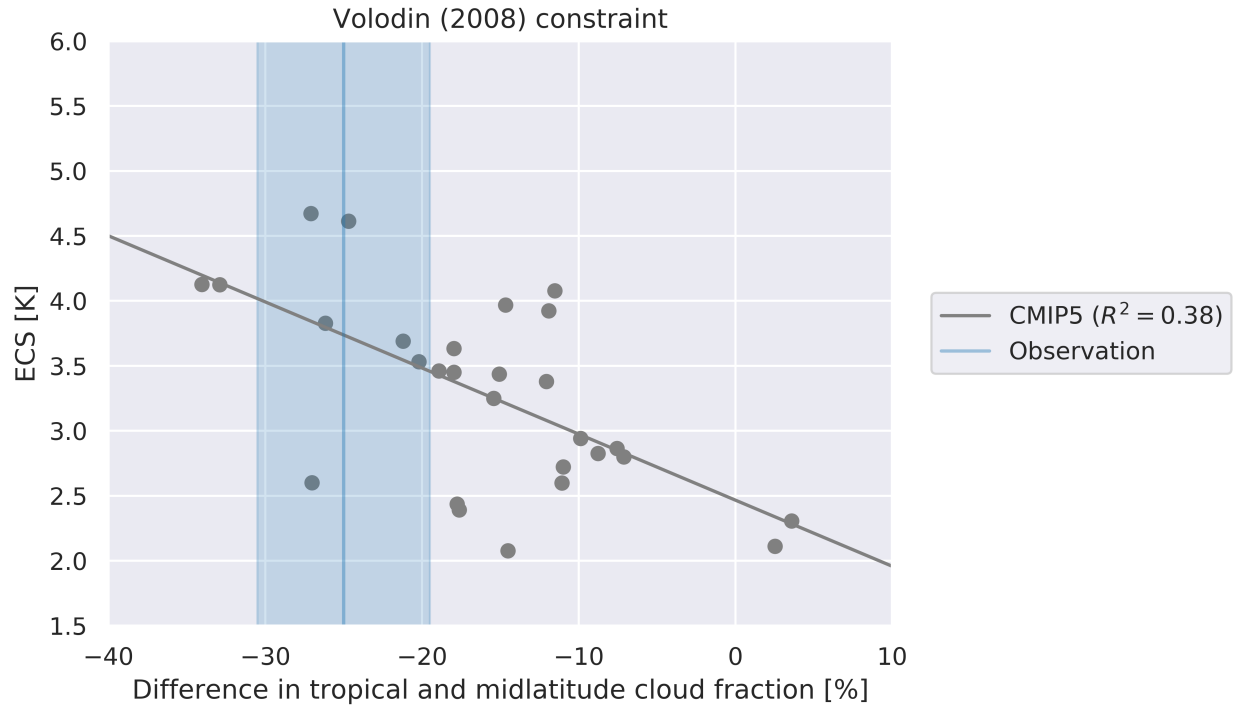
Fig. 6: Difference in total cloud fraction between tropics (28°S - 28°N) and Southern midlatitudes (56°S - 36°S) (Volodin, 2008) vs. equilibrium climate sensitivity from CMIP5 models.

10.2.3 User settings

User setting files (cfg files) are stored in nml/cfg_carbon/

1. carbon_tsline

   Required Settings (scripts)

   - ts_minlat: minimum latitude for area averaging
   - ts_maxlat: maximum latitude for area averaging
   - ts_minlon: minimum longitude for area averaging
   - ts_maxlon: maximum longitude for area averaging
   - ts_maxyear: last year (time range)
   - ts_minyear: first year (time range)
   - plot_units: units to appear on Figure
   - time_avg: currently, only yearly is available
   - area_opper: type of area operation (sum)
   - styleset: Plot style

   Optional settings (scripts)

   - multi_model_mean: True for multi-model mean calculation
   - volcanoes: True for marking years with lage volcanic eruptions

- align: True for aligning models to have the same start year (needed for idealized 2x CO2 simulations)

- ts_anomaly: calculates anomalies with respect to a defined time range average (anom)

- ridx_start: if ts_anomaly is True, define start time index for reference period

- ridx_end: if ts_anomaly is True, define end time index for reference period

- ref_start: if ts_anomaly is True, define start year for reference period

- ref_end: if ts_anomaly is True, define end year for reference period

Required settings (variables)

- reference_dataset: name of reference data set

2. carbon_gammaHist.ncl

Required Settings (scripts)

- start_year: first year (time range)

- end_year: last year (time range)

- plot_units: units to appear on Figure

- ec_anom: calculates anomalies with respect to the first 10-year average (anom)

- scatter_log: set logarithmic axes in scatterplot.ncl

- styleset: Plot style

Optional settings (scripts)

- ec_volc : exclude 2 years after volcanic erruptions (True/False)

3. carbon_constraint.ncl

Required Settings (scripts)

- gIAV_diagscript: "gammaHist_Fig3and4"

- gIAV_start: start year of GammIAV calculation period

- gIAV_end: end year of GammIAV calculation period

- ec_anom: True

- con_units: label string for units, e.g. (GtC/K)

- nc_infile: specify path to historical gamma values derived by carbon_gammaHist.ncl

- styleset: Plot style

Optional settings (scripts)

- reg_models: Explicit naming of individual models to be excluded from the regression

10.2.4 Variables

- tas (atmos, monthly mean, longitude latitude time)

- nbp (land, monthly mean, longitude latitude time)

- fgco2 (ocean, monthly mean, longitude latitude time)

10.2.5 Observations and reformat scripts

- GCP: Global Carbon Budget including land (nbp) and ocean (fgco2) carbon fluxes

- NCEP: National Centers for Environmental Prediction reanalysis data for near surface temperature

10.2.6 References

- Cox, P. M., D. B. Pearson, B. B. Booth, P. Friedlingstein, C. C. Huntingford, C. D. B. Jones, and C. M. Luke, 2013, Sensitivity of tropical carbon to climate change constrained by carbon dioxide variability, Nature, 494(7437), 341-344. doi: 10.1038/nature11882

- Wenzel, S., P. M. Cox, V. Eyring, and P. Friedlingstein, 2014, Emergent Constraints on Climate Carbon Cycle Feedbacks in the CMIP5 Earth System Models, JGR Biogeoscience, 119(5), doi: 2013JG002591.

10.2.7 Example plots
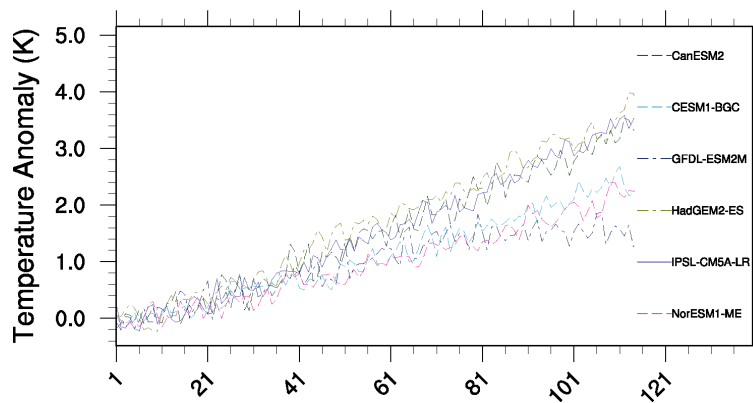
Fig. 7: Time series of tropical (30S to 30N) mean near surface temperature (tas) change between year 30 and year 110 for the CMIP5 models simulated with prescribed CO2 (1%/yr CO2 increase) coupled simulation (1pctCO2).

10.3 Emergent constraint on equilibrium climate sensitivity from global temperature variability

10.3.1 Overview

This recipe reproduces the emergent constraint proposed by Cox et al. (2018) for the equilibrium climate sensitivity (ECS) using global temperature variability. The latter is defined by a metric which can be calculated from the global
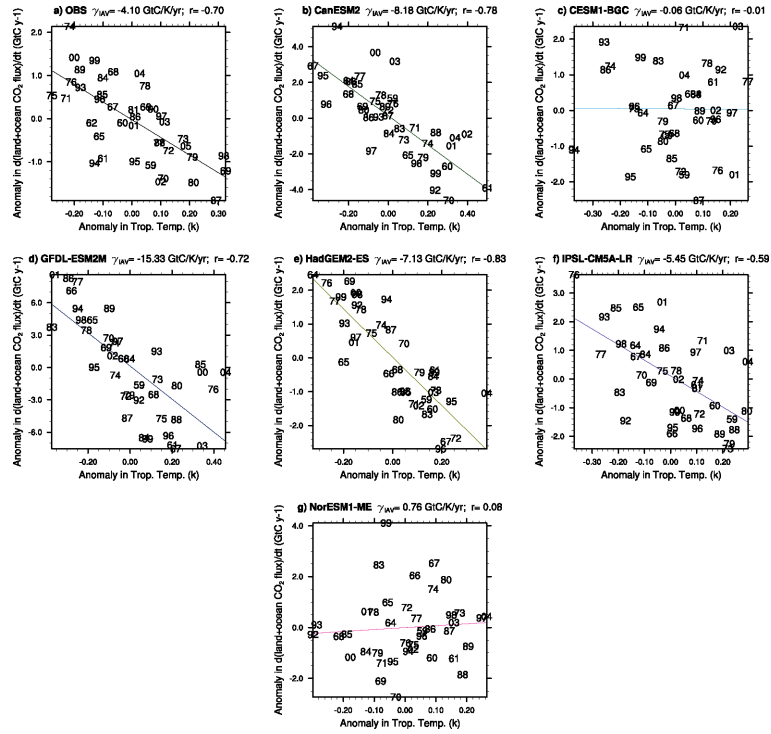
Fig. 8: Correlations between the interannual variability of global co2flux (nbp+fgco2) and tropical temperature for the individual CMIP5 models using esmHistorical simulations, and for observations.
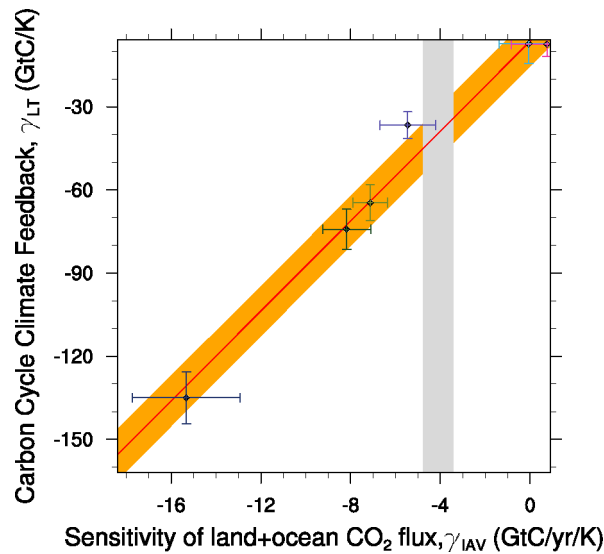

Fig. 9: Carbon cycle-climate feedback of tropical land carbon vs. the sensitivity of co2flux to interannual temperature variability in the tropics (30S to 30N). The red line shows the linear best fit of the regression together with the prediction error (orange shading) and the gray shading shows the observed range.
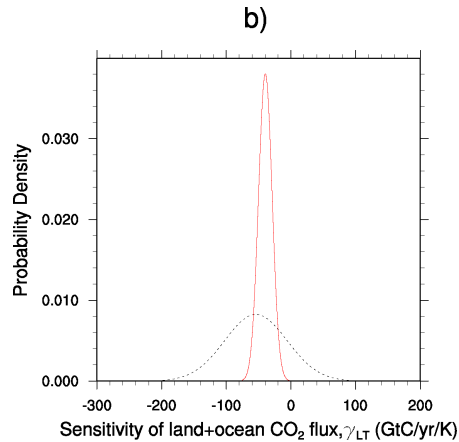
Fig. 10: Probability Density Functions for the pure CMIP5 ensemble (black dashed) and after applying the observed constraint to the models (red solid)

temperature variance (in time) σ_T and the one-year-lag autocorrelation of the global temperature α_{1T} by

$$\psi = \frac{\sigma_T}{\sqrt{-\ln(\alpha_{1T})}}$$

Using the simple Hasselmann model they show that this quantity is linearly correlated with the ECS. Since it only depends on the temporal evolution of the global surface temperature, there is lots of observational data available which allows the construction of an emergent relationship. This method predicts an ECS range of 2.2K to 3.4K (66% confidence limit).

10.3.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_cox18nature.yml

Diagnostics are stored in diag_scripts/

- emergent_constraints/cox18nature.py
- climate_metrics/ecs.py
- climate_metrics/psi.py

10.3.3 User settings in recipe

1. Preprocessor
    - `area_statistics` (*operation: mean*): Calculate global mean.
2. Script emergent_constraints/cox18nature.py
    - `confidence_level`, *float*, optional (default: 0.66): Confidence level for ECS error estimation.
3. Script climate_metrics/ecs.py
    - `read_external_file`, *str*, optional: Read ECS and net climate feedback parameter from external file. All other input data is ignored.
4. Script climate_metrics/psi.py

- `output_attributes`, *dict*, optional: Write additional attributes to all output netcdf files.
- `lag`, *int*, optional (default: 1): Lag (in years) for the autocorrelation function.
- `window_length`, *int*, optional (default: 55): Number of years used for the moving window average.

10.3.4 Variables

- *tas* (atmos, monthly, longitude, latitude, time)
- *tasa* (atmos, monthly, longitude, latitude, time)

10.3.5 Observations and reformat scripts

- HadCRUT4 (*tasa*)

10.3.6 References

- Cox, Peter M., Chris Huntingford, and Mark S. Williamson. "Emergent constraint on equilibrium climate sensitivity from global temperature variability." Nature 553.7688 (2018): 319.

10.3.7 Example plots
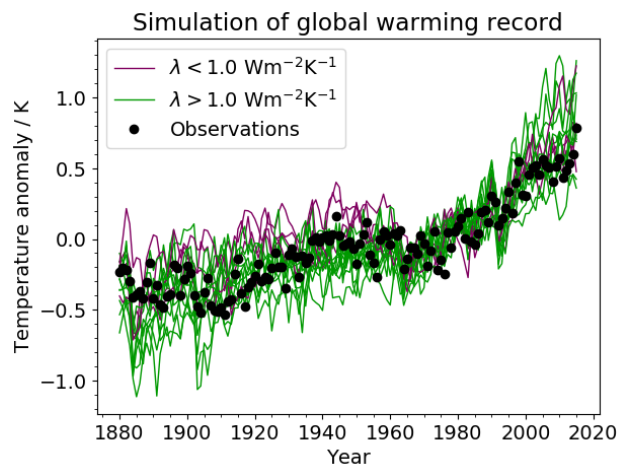
Fig. 11: Simulated change in global temperature from CMIP5 models (coloured lines), compared to the global temperature anomaly from the HadCRUT4 dataset (black dots). The anomalies are relative to a baseline period of 1961–1990. The model lines are colour-coded, with lower-sensitivity models (> 1 Wm^{-2}K^{-1}) shown by green lines and higher-sensitivity models (< 1 Wm^{-2}K^{-1}) shown by magenta lines.
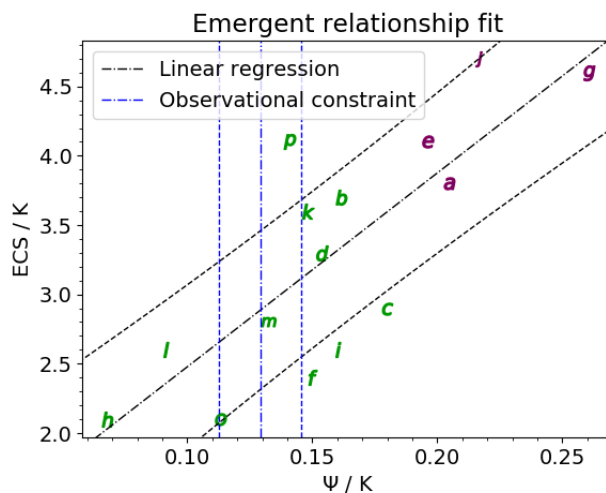
Fig. 12: Emergent relationship between ECS and the metric. The black dot-dashed line shows the best-fit linear regression across the model ensemble, with the prediction error for the fit given by the black dashed lines. The vertical blue lines show the observational constraint from the HadCRUT4 observations: the mean (dot-dashed line) and the mean plus and minus one standard deviation (dashed lines).
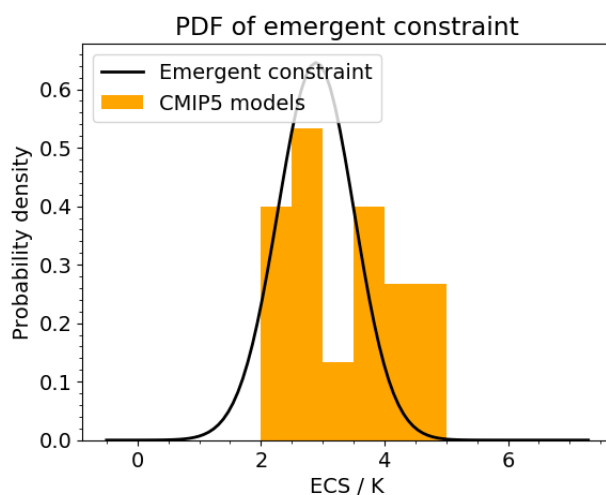

Fig. 13: The PDF for ECS. The orange histograms (both panels) show the prior distributions that arise from equal weighting of the CMIP5 models in 0.5 K bins.

10.4 Emergent constraint on snow-albedo effect

10.4.1 Overview

The recipe recipe_snowalbedo.yml computes the springtime snow-albedo feedback values in climate change versus springtime values in the seasonal cycle in transient climate change experiments following Hall and Qu (2006). The strength of the snow-albedo effect is quantified by the variation in net incoming shortwave radiation (Q) with surface air temperature (T_s) due to changes in surface albedo α_s:

$$\left(\frac{\partial Q}{\partial T_s} \right) = -I_t \cdot \frac{\partial \alpha_p}{\partial \alpha_s} \cdot \frac{\Delta \alpha_s}{\Delta T_s}$$

The diagnostic produces scatterplots of simulated springtime $\Delta\alpha_s/\Delta T_s$ values in climate change (ordinate) vs. simulated springtime $\Delta\alpha_s/\Delta T_s$ values in the seasonal cycle (abscissa).

Ordinate values: the change in April α_s (future projection - historical) averaged over NH land masses poleward of 30°N is divided by the change in April T_s (future projection - historical) averaged over the same region. The change in α_s (or T_s) is defined as the difference between 22nd-century-mean α_s: (T_s) and 20th-century-mean α_s. Values of α_s are weighted by April incoming insolation (I_t) prior to averaging.

Abscissa values: the seasonal cycle $\Delta\alpha_s/\Delta T_s$ values, based on 20th century climatological means, are calculated by dividing the difference between April and May α_s: averaged over NH continents poleward of 30°N by the difference between April and May T_s averaged over the same area. Values of α_s: are weighted by April incoming insolation prior to averaging.

10.4.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_snowalbedo.yml

Diagnostics are stored in diag_scripts/emergent_constraints/

- snowalbedo.ncl: springtime snow-albedo feedback values vs. seasonal cycle

10.4.3 User settings in recipe

1. Script snowalbedo.ncl

   Required settings for script

   - exp_presentday: name of present-day experiment (e.g. "historical")
   - exp_future: name of climate change experiment (e.g. "rcp45")

   Optional settings for script

   - diagminmax: observational uncertainty (min and max)
   - legend_outside: create extra file with legend (true, false)
   - styleset: e.g. "CMIP5" (if not set, this diagnostic will create its own color table and symbols for plotting)
   - suffix: string to be added to output filenames
   - xmax: upper limit of x-axis (default = automatic)
   - xmin: lower limit of x-axis (default = automatic)
   - ymax: upper limit of y-axis (default = automatic)

- ymin: lower limit of y-axis (default = automatic)

Required settings for variables

- ref_model: name of reference data set

Optional settings for variables

Required settings (scripts)

Optional settings (scripts)

10.4.4 Variables

- tas (atmos, monthly mean, longitude latitude time)
- rsdt (atmos, monthly mean, longitude latitude time)
- rsuscs, rsdscs (atmos, monthly mean, longitude latitude time)

10.4.5 Observations and reformat scripts

- ERA-Interim (tas - esmvaltool/utils/cmorizers/obs/cmorize_obs_ERA-Interim.ncl)
- ISCCP-FH (rsuscs, rsdscs, rsdt - esmvaltool/utils/cmorizers/obs/cmorize_obs_isccp_fh.ncl)

10.4.6 References

- Flato, G., J. Marotzke, B. Abiodun, P. Braconnot, S.C. Chou, W. Collins, P. Cox, F. Driouech, S. Emori, V. Eyring, C. Forest, P. Gleckler, E. Guilyardi, C. Jakob, V. Kattsov, C. Reason and M. Rummukainen, 2013: Evaluation of Climate Models. In: Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change [Stocker, T.F., D. Qin, G.-K. Plattner, M. Tignor, S.K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex and P.M. Midgley (eds.)]. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA.
- Hall, A., and X. Qu, 2006: Using the current seasonal cycle to constrain snow albedo feedback in future climate change, Geophys. Res. Lett., 33, L03502, doi:10.1029/2005GL025127.

10.4.7 Example plots

10.5 Equilibrium climate sensitivity

10.5.1 Overview

Equilibrium climate sensitivity is defined as the change in global mean temperature as a result of a doubling of the atmospheric CO_2 concentration compared to pre-industrial times after the climate system has reached a new equilibrium. This recipe uses a regression method based on Gregory et al. (2004) to calculate it for several CMIP models.
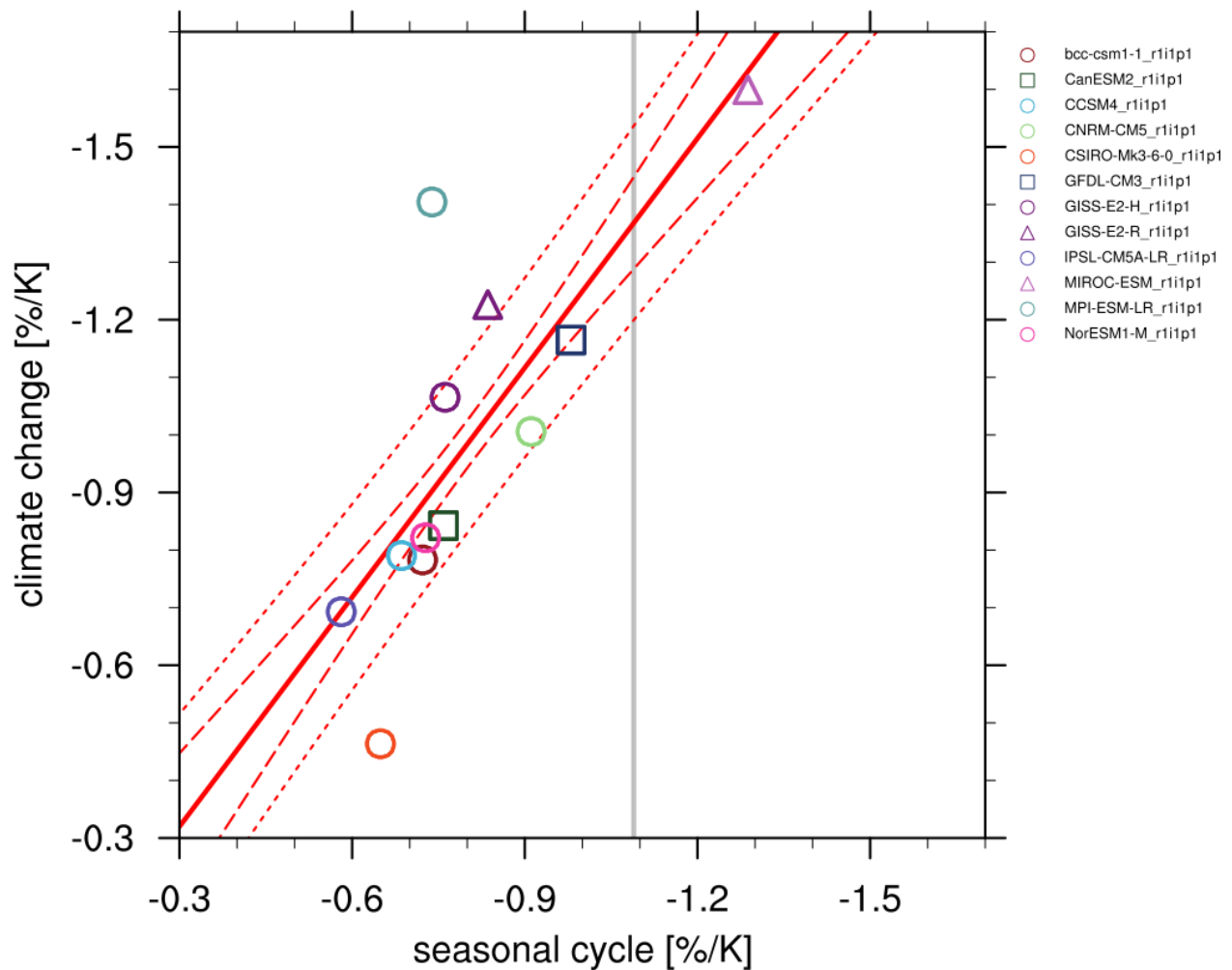
Fig. 14: Scatterplot of springtime snow-albedo effect values in climate change vs. springtime $\Delta\alpha_s/\Delta T_s$ values in the seasonal cycle in transient climate change experiments (CMIP5 historical experiments: 1901-2000, RCP4.5 experiments: 2101-2200). Similar to IPCC AR5 Chapter 9 (Flato et al., 2013), Figure 9.45a.

10.5.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_ecs.yml

Diagnostics are stored in diag_scripts/

- climate_metrics/ecs.py

- climate_metrics/create_barplot.py

- climate_metrics/create_scatterplot.py

10.5.3 User settings in recipe

1. Preprocessor

    - `area_statistics` (*operation: mean*): Calculate global mean.

2. Script climate_metrics/ecs.py

    - `calculate_mmm`, *bool*, optional (default: `True`): Calculate multi-model mean ECS.

    - `output_attributes`, *dict*, optional: Write additional attributes to all output netcdf files.

    - `read_external_file`, *str*, optional: Read ECS and net climate feedback parameter from external file. Can be given relative to the diagnostic script or as absolute path.

    - `seaborn_settings`, *dict*, optional: Options for seaborn's `set()` method (affects all plots), see https://seaborn.pydata.org/generated/seaborn.set.html.

3. Script climate_metrics/create_barplot.py

    - `label_attribute`, *str*, optional: Attribute of the cube which is used as label for the different input files in the barplot.

    - `patterns`, *list of str*, optional: Patterns to filter list of input files.

    - `seaborn_settings`, *dict*, optional: Options for seaborn's `set()` method (affects all plots), see https://seaborn.pydata.org/generated/seaborn.set.html.

    - `sort_ascending`, *bool*, optional (default: `False`): Sort bars in ascending order.

    - `sort_descending`, *bool*, optional (default: `False`): Sort bars in descending order.

    - `value_labels`, *bool*, optional (default: `False`): Label bars with value of that bar.

    - `y_range`, *list of float*, optional: Range for the Y axis of the plot.

4. Script climate_metrics/create_scatterplot.py

    - `dataset_style`, *str*, optional: Name of the style file (located in `esmvaltool.diag_scripts.shared.plot.styles_python`).

    - `pattern`, *str*, optional: Pattern to filter list of input files.

    - `seaborn_settings`, *dict*, optional: Options for seaborn's `set()` method (affects all plots), see https://seaborn.pydata.org/generated/seaborn.set.html.

    - `y_range`, *list of float*, optional: Range for the Y axis of the plot.

10.5.4 Variables

- *rlut* (atmos, monthly, longitude, latitude, time)

- *rsdt* (atmos, monthly, longitude, latitude, time)

- *rsut* (atmos, monthly, longitude, latitude, time)

- *tas* (atmos, monthly, longitude, latitude, time)

10.5.5 Observations and reformat scripts

None

10.5.6 References

- Gregory, Jonathan M., et al. "A new method for diagnosing radiative forcing and climate sensitivity." Geophysical research letters 31.3 (2004).
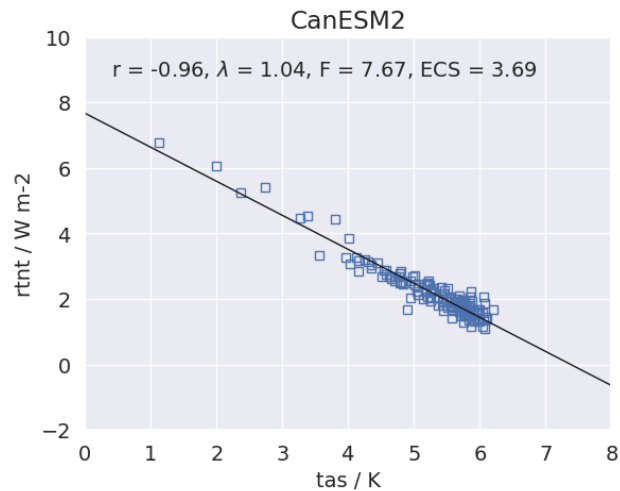
10.5.7 Example plots

Fig. 15: Scatterplot between TOA radiance and global mean surface temperature anomaly for 150 years of the abrupt 4x CO2 experiment including linear regression to calculate ECS for CanESM2 (CMIP5).

10.6 Multiple ensemble diagnostic regression (MDER) for constraining future austral jet position

10.6.1 Overview

Wenzel et al. (2016) use multiple ensemble diagnostic regression (MDER) to constrain the CMIP5 future projection of the summer austral jet position with several historical process-oriented diagnostics and respective observations.

The following plots are reproduced:

- Absolute correlation between the target variable and the diagnostics.

- Scatterplot between the target variable and the MDER-calculated linear combination of diagnostics.

- Boxplot of RMSE for the unweighted multi-model mean and the (MDER) weighted multi-model mean of the target variable in a pseudo-reality setup.

- Time series of the target variable for all models, observations and MDER predictions.

- Errorbar plots for all diagnostics.

- Scatterplots between the target variable and all diagnostics.

10.6.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_wenzel16jclim.yml

Diagnostics are stored in diag_scripts/

- austral_jet/asr.ncl

- austral_jet/main.ncl

- mder/absolute_correlation.ncl

- mder/regression_stepwise.ncl

- mder/select_for_mder.ncl

10.6.3 User settings in recipe

1. Preprocessor

   - `extract_region`: Region extraction.

   - `extract_levels`: Pressure level extraction.

   - `area_statistics`: Spatial average calculations.

2. Script austral_jet/asr.ncl

   - `season`, *str*: Season.

   - `average_ens`, *bool*, optional (default: `False`): Average over all given ensemble members of a climate model.

   - `wdiag`, *array of str*, optional: Names of the diagnostic for MDER output. Necessary when MDER output is desired.

   - `wdiag_title`, *array of str*, optional: Names of the diagnostic in plots.

3. Script austral_jet/main.ncl

   - `styleset`, *str*: Style set used for plotting the multi-model plots.

   - `season`, *str*: Season.

   - `average_ens`, *bool*, optional (default: `False`): Average over all given ensemble members of a climate model.

   - `rsondes`, *array of str*, optional: Additional observations used in the plot but not for MDER output.

- rsondes_file, *array of str*, optional: Paths to the additional observations Necessary when rsondes is given.

- rsondes_yr_min, *int*, optional: Minimum year for additional observations. Necessary when rsondes is given.

- rsondes_yr_max, *int*, optional: Maximum year for additional observations. Necessary when rsondes is given.

- wdiag, *array of str*, optional: Names of the diagnostic for MDER output. Necessary when MDER output is desired.

- wdiag_title, *array of str*, optional: Names of the diagnostic in plots.

- derive_var, *str*, optional: Derive variables using NCL functions. Must be one of "tpp", "mmstf".

- derive_latrange, *array of float*, optional: Latitude range for variable derivation. Necessary if derive_var is given.

- derive_lev, *float*, optional: Pressure level (given in *Pa*) for variable derivation. Necessary if derive_var is given.

4. Script mder/absolute_correlation.ncl

- p_time, *array of int*: Start years for future projections.

- p_step, *int*: Time range for future projections (in years).

- scal_time, *array of int*: Time range for base period (in years) for anomaly calculations used when calc_type = "trend".

- time_oper, *str*: Operation used in NCL time_operation function.

- time_opt, *str*: Option used in NCL time_operation function.

- calc_type, *str*: Calculation type for the target variable. Must be one of "trend", "pos", "int".

- domain, *str*: Domain tag for provenance tracking.

- average_ens, *bool*, optional (default: False): Average over all given ensemble members of a climate model.

- region, *str*, optional: Region used for area aggregation. Necessary if input of target variable is multidimensional.

- area_oper, *str*, optional: Operation used in NCL area_operation function. Necessary if multidimensional is given.

- plot_units, *str*, optional (attribute for variable_info): Units for the target variable used in the plots.

5. Script mder/regression_stepwise.ncl

- p_time, *array of int*: Start years for future projections.

- p_step, *int*: Time range for future projections (in years).

- scal_time, *array of int*: Time range for base period (in years) for anomaly calculations used when calc_type = "trend".

- time_oper, *str*: Operation used in NCL time_operation function.

- time_opt, *str*: Option used in NCL time_operation function.

- calc_type, *str*: Calculation type for the target variable. Must be one of "trend", "pos", "int".

- domain, *str*: Domain tag for provenance tracking.

- `average_ens`, *bool*, optional (default: `False`): Average over all given ensemble members of a climate model.

- `smooth`, *bool*, optional (default: `False`): Smooth time period with 1-2-1 filter.

- `iter`, *int*, optional: Number of iterations for smoothing. Necessary when `smooth` is given.

- `cross_validation_mode`, *bool*, optional (default: `False`): Perform cross-validation.

- `region`, *str*, optional: Region used for area aggregation. Necessary if input of target variable is multidimensional.

- `area_oper`, *str*, optional: Operation used in NCL `area_operation` function. Necessary if multidimensional is given.

- `plot_units`, *str*, optional (attribute for `variable_info`): Units for the target variable used in the plots.

6. Script mder/select_for_mder.ncl

- `wdiag`, *array of str*: Names of the diagnostic for MDER output. Necessary when MDER output is desired.

- `domain`, *str*: Domain tag for provenance tracking.

- `ref_dataset`, *str*: Style set used for plotting the multi-model plots.

- `average_ens`, *bool*, optional (default: `False`): Average over all given ensemble members of a climate model.

- `derive_var`, *str*, optional: Derive variables using NCL functions. Must be one of `"tpp"`, `"mmstf"`.

10.6.4 Variables

- *ta* (atmos, monthly, longitude, latitude, pressure level, time)

- *uajet* (atmos, monthly, time)

- *va* (atmos, monthly, longitude, latitude, pressure level, time)

- *ps* (atmos, monthly, longitude, latitude, time)

- *asr* (atmos, monthly, longitude, latitude, time)

10.6.5 Observations and reformat scripts

- ERA-Intermin (*ta*, *uajet*, *va*, *ps*)
- CERES-EBAF (*asr*)

10.6.6 References

- Wenzel, S., V. Eyring, E.P. Gerber, and A.Y. Karpechko: Constraining Future Summer Austral Jet Stream Positions in the CMIP5 Ensemble by Process-Oriented Multiple Diagnostic Regression. J. Climate, 29, 673–687, doi:10.1175/JCLI-D-15-0412.1, 2016.
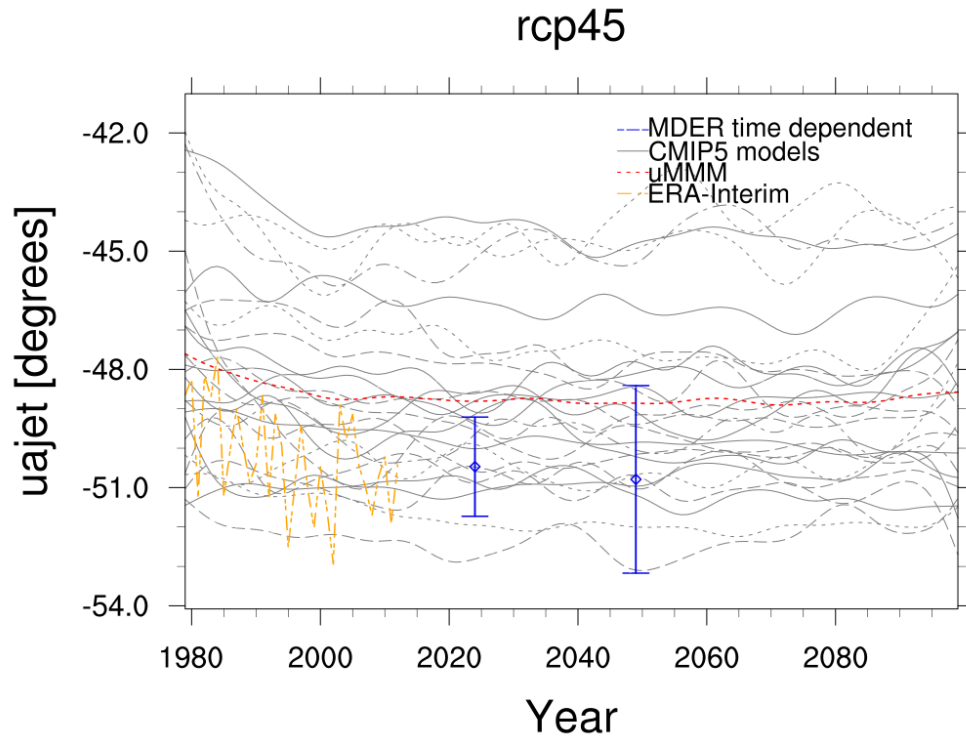
10.6.7 Example plots

Fig. 16: Time series of the the target variable (future austral jet position in the RCP 4.5 scenario) for the CMIP5 ensemble, observations, unweighted multi-model mean projections and (MDER) weighted multi-model mean projections.

10.7 Projected land photosynthesis constrained by changes in the seasonal cycle of atmospheric CO_2

10.7.1 Overview

Selected figures from Wenzel et al. (2016) are reproduced with recipe_wenzel16nat.yml. Gross primary productivity (gpp) and atmospheric CO_2 concentrations at the surface (co2s) are analyzed for the carbon cycle - concentration feedback in the historical (esmHistorical) and uncoupled (esmFixCLim1, here the carbon cycle is uncoupled to the climate response) simulations. The recipe includes a set of routines to diagnose the long-term carbon cycle - concentration feedback parameter (beta) from an ensemble of CMIP5 models and the observable change in the CO_2 seasonal cycle amplitude due to rising atmospheric CO_2 levels. As a key figure of this recipe, the diagnosed values from the models beta vs. the change in CO_2 amplitude are compared in a scatter plot constituting an emergent constraint.
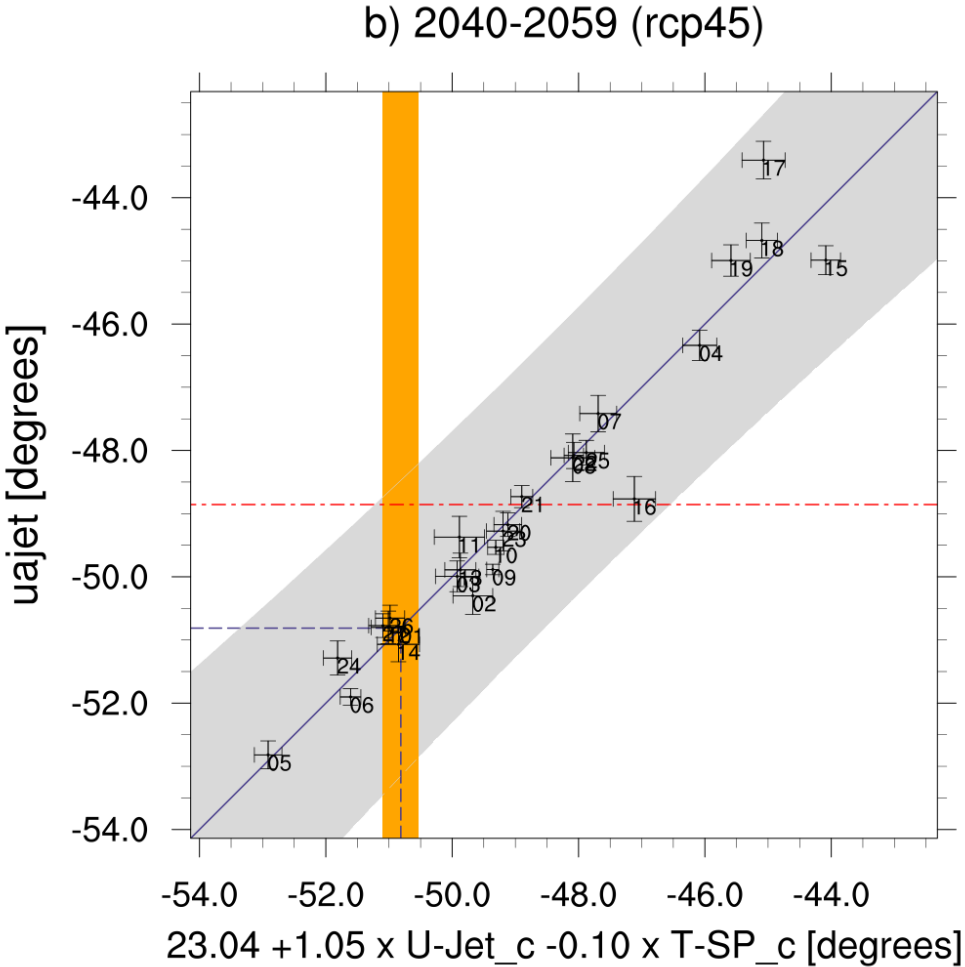
Fig. 17: Scatterplot of the target variable (future austral jet position in the RCP 4.5 scenario) vs. the MDER-determined linear combination of diagnostics for the CMIP5 ensemble.
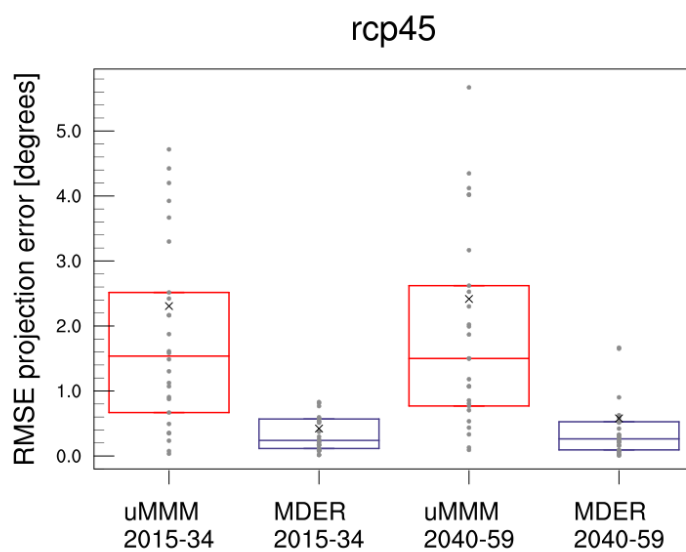
Fig. 18: Boxplot for the RMSE of the target variable for the unweighted and (MDER) weighted multi-model mean projections in a pseudo-reality setup.
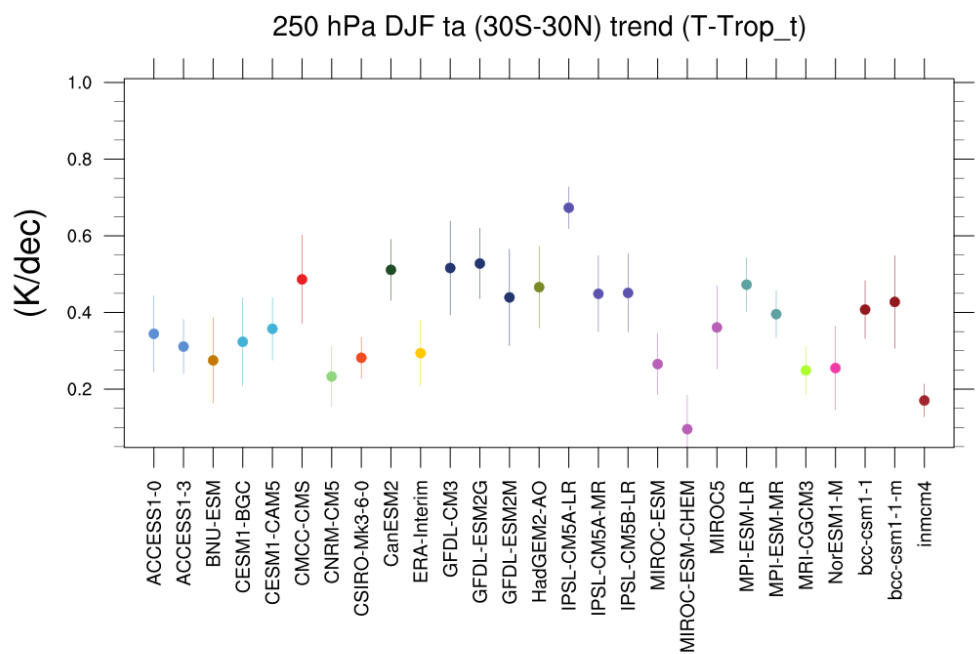
Fig. 19: Trends in tropical DJF temperature at 250hPa for different CMIP5 models and observations.
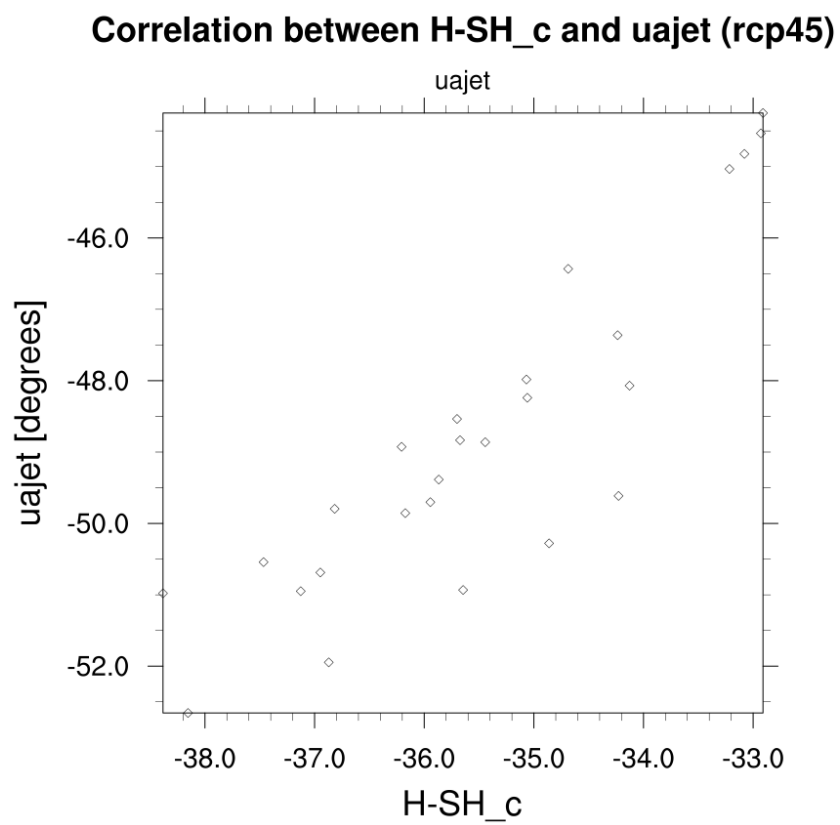
Fig. 20: Scatterplot of the target variable (future austral jet position in the RCP 4.5 scenario) vs. a single diagnostic, the historical location of the Southern hemisphere Hadley cell boundary for the CMIP5 ensemble.

10.7.2 Available recipe and diagnostics

Recipes are stored in recipes/

- recipe_wenzel16nat.yml

Diagnostics are stored in diag_scripts/carbon_ec/

- carbon_beta: (1) scatter plot of annual gpp vs. annual CO_2 and (2) barchart of gpp($2xCO_2$)/gpp($1xCO_2$); calculates beta for emergent constraint (carbon_co2_cycle.ncl)

- carbon_co2_cycle.ncl: (1) scatter plot of CO_2 amplitude vs. annual CO_2, (2) barchart of sensitivity of CO_2 amplitude to CO_2, (3) emergent constraint: gpp($2xCO_2$)/gpp($1xCO_2$) vs. sensitivity of CO_2 amplitude to CO_2, (4) probability density function of constrained and unconstrained sensitivity of CO_2 amplitude to CO_2

10.7.3 User settings

1. Script carbon_beta.ncl

   Required Settings (scripts)

   - styleset: project style for lines, colors and symbols

   Optional Settings (scripts)

   - bc_xmax_year: end year to calculate beta (default: use last available year of all models)
   - bc_xmin_year: start year to calculate beta (default: use first available year of all models)

   Required settings (variables)

   Optional settings (variables)

2. Script carbon_co2_cycle.ncl

   Required Settings (scripts)

   - nc_infile: path of netCDF file containing beta (output from carbon_beta.ncl)
   - styleset: project style for lines, colors and symbols

   Optional Settings (scripts)

   - bc_xmax_year: end year (default = last year of all model datasets available)
   - bc_xmin_year: start year (default = first year of all model datasets available)

   Required settings (variables)

   - reference_dataset: name of reference datatset (observations)

   Optional settings (variables)

10.7.4 Variables

- co2s (atmos, monthly mean, plev longitude latitude time)
- gpp (land, monthly mean, longitude latitude time)

10.7.5 Observations and reformat scripts

- ESRL: Earth System Research Laboratory, ground-based CO_2 measurements

10.7.6 References

- Wenzel, S., Cox, P., Eyring, V. et al., 2016, Projected land photosynthesis constrained by changes in the seasonal cycle of atmospheric CO_2. Nature 538, 499501, doi: doi.org/10.1038/nature19772
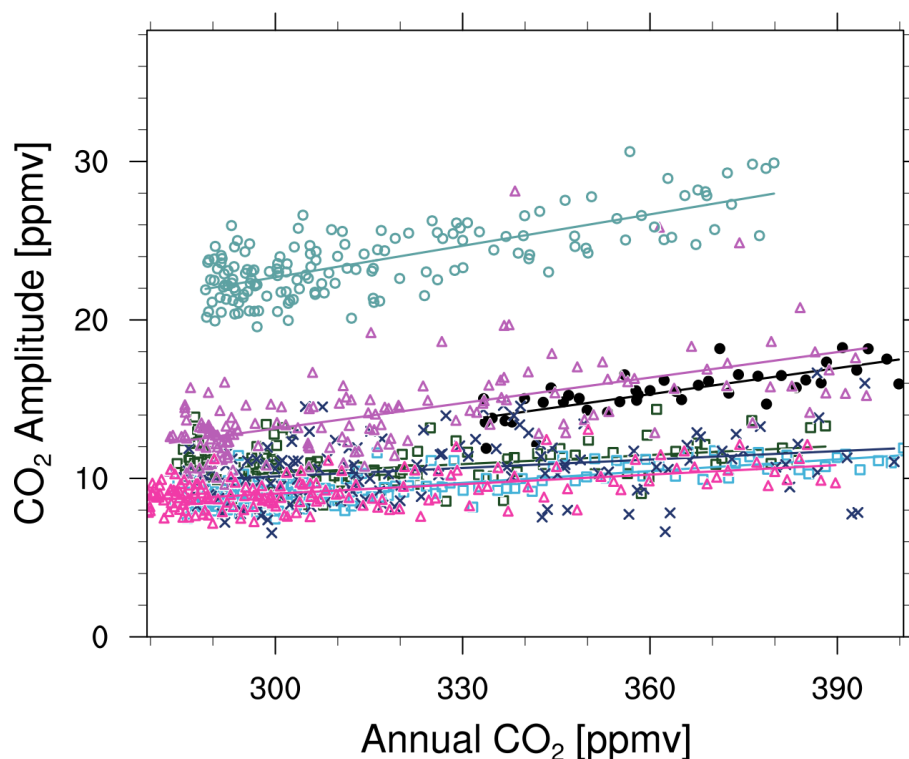
10.7.7 Example plots

Fig. 21: Comparison of CO_2 seasonal amplitudes for CMIP5 historical simulations and observations showing annual mean atmospheric CO_2 versus the amplitudes of the CO_2 seasonal cycle at Pt. Barrow, Alaska (produced with carbon_co2_cycle.ncl, similar to Fig. 1a from Wenzel et al. (2016)).
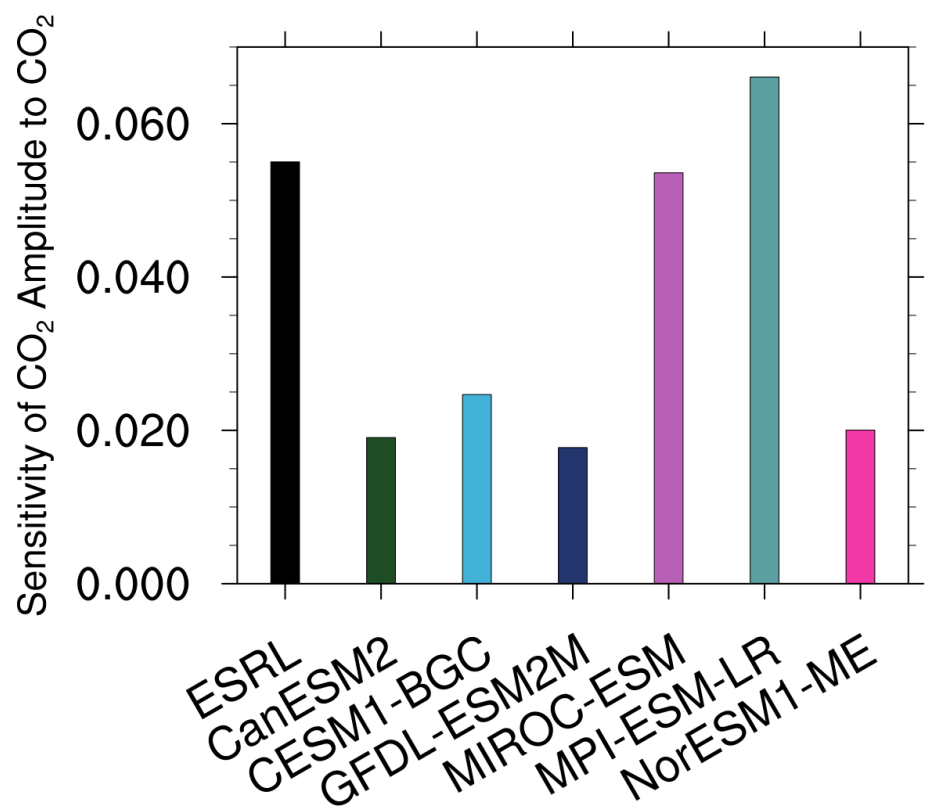
Fig. 22: Barchart showing the gradient of the linear correlations for the comparison of CO_2 seasonal amplitudes for CMIP5 historical for at Pt. Barrow, Alaska (produced with carbon_co2_cycle.ncl, similar to Fig. 1b from Wenzel et al. (2016)).
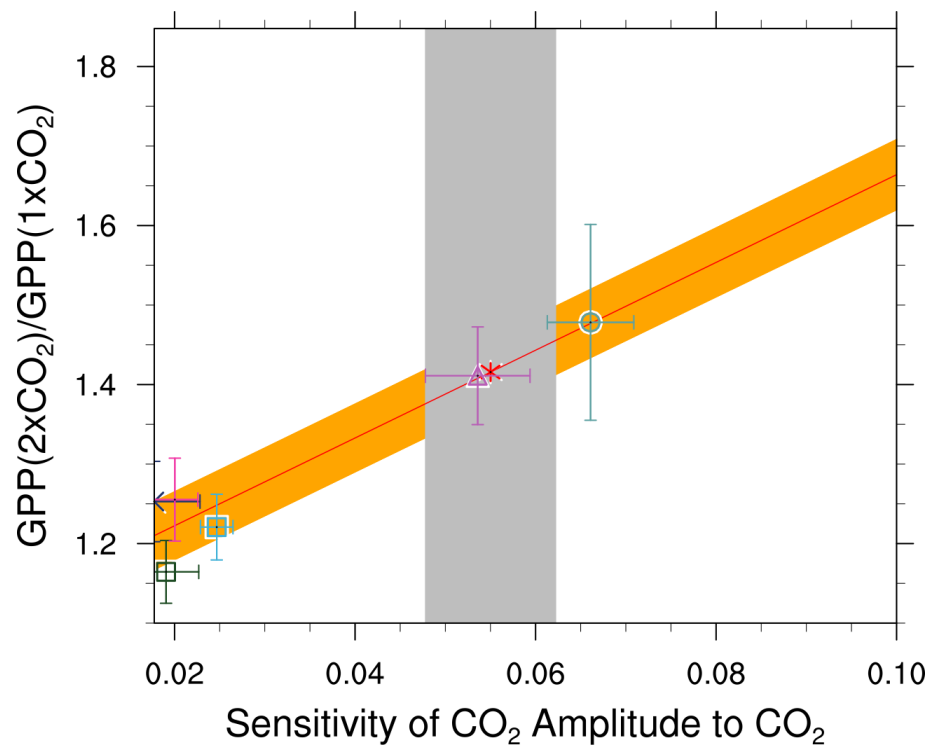
Fig. 23: Emergent constraint on the relative increase of large-scale GPP for a doubling of CO_2, showing the correlations between the sensitivity of the CO_2 amplitude to annual mean CO_2 increases at Pt. Barrow (x-axis) and the high-latitude (60N - 90N) CO_2 fertilization on GPP at 2xCO_2. The red line shows the linear best fit of the regression together with the prediction error (orange shading), the gray shading shows the observed range (produced with carbon_co2_cycle.ncl, similar to Fig. 3a from Wenzel et al. (2016)).

10.8 Transient Climate Response

10.8.1 Overview

The transient climate response (TCR) is defined as the global and annual mean surface air temperature anomaly in the *1pctCO2* scenario (1% CO_2 increase per year) for a 20 year period centered at the time of CO_2 doubling, i.e. using the years 61 to 80 after the start of the simulation. We calculate the temperature anomaly by subtracting a linear fit of the *piControl* run for all 140 years of the *1pctCO2* experiment prior to the TCR calculation (see Gregory and Forster, 2008).

10.8.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_tcr.yml

Diagnostics are stored in diag_scripts/

- climate_metrics/tcr.py
- climate_metrics/create_barplot.py
- climate_metrics/create_scatterplot.py

10.8.3 User settings in recipe

1. Preprocessor

   - `area_statistics` (*operation: mean*): Calculate global mean.

2. Script climate_metrics/tcr.py

   - `plot`, *bool*, optional (default: `True`): Plot temperature anomaly vs. time.
   - `read_external_file`, *str*, optional: Read TCR from external file. Can be given relative to the diagnostic script or as absolute path.
   - `seaborn_settings`, *dict*, optional: Options for seaborn's `set()` method (affects all plots), see https://seaborn.pydata.org/generated/seaborn.set.html.

3. Script climate_metrics/create_barplot.py

   See *Equilibrium climate sensitivity*.

4. Script climate_metrics/create_scatterplot.py

   See *Equilibrium climate sensitivity*.

10.8.4 Variables

- *tas* (atmos, monthly, longitude, latitude, time)

10.8.5 Observations and reformat scripts

None

10.8.6 References

- Gregory, J. M., and P. M. Forster. "Transient climate response estimated from radiative forcing and observed temperature change." Journal of Geophysical Research: Atmospheres 113.D23 (2008).
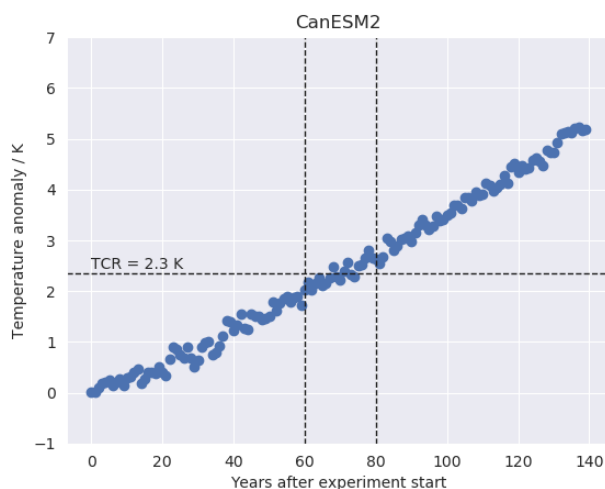
10.8.7 Example plots

Fig. 24: Time series of the global mean surface air temperature anomaly (relative to the linear fit of the pre-industrial control run) of CanESM2 (CMIP5) for the 1% CO_2 increase per year experiment. The horizontal dashed line indicates the transient climate response (TCR) defined as the 20 year average temperature anomaly centered at the time of CO_2 doubling (vertical dashed lines).

10.9 Constraining future Indian Summer Monsoon projections with the present-day precipitation over the tropical western Pacific

10.9.1 Overview

Following Li et al. (2017) the change between present-day and future Indian Summer Monsoon (ISM) precipitation is constrained using the precipitation over the tropical western Pacific compared to a fixed, observed amount of 6 mm d⁻¹ from Global Precipitation Climatology Project (GPCP) (Adler et al., 2003) for 1980-2009. For CMIP6, historical data for 1980-2009 should be used. For CMIP5 historical data from 1980-2005 should be used, due to the length of the data sets. At the moment it is not possible to use a combined `['historical', 'rcp']` data set, because the diagnostic requires that a historical data set is given.

10.9.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_li17natcc.yml

Diagnostics are stored in diag_scripts/

- emergent_constraints/lif1f2.py

10.9.3 User settings in recipe

The recipe can be run with different CMIP5 and CMIP6 models. For each model, two experiments must be given: one historical run, possibly between 1980-2009 and one other model experiment. The user can choose the other model experiment, but it needs to be the same for all given models. The start and end year for the second data set can be choosen by the user, but should be consistent for all models (the same for future scenarios, the same length for other experiments). Different ensemble members are not possible, yet.

10.9.4 Variables

- *pr* (atmos, monthly, longitude, latitude, time)
- *ua* (atmos, monthly, longitude, latitude, plev, time)
- *va* (atmos, monthly, longitude, latitude, plev, time)
- *ts* (atmos, monthly, longitude, latitude, time)

10.9.5 Observations and reformat scripts

None

10.9.6 References

- Li, G., Xie, S. P., He, C., and Chen, Z. S.: Western Pacific emergent constraint lowers projected increase in Indian summer monsoon rainfall, Nat Clim Change, 7, 708-+, 2017
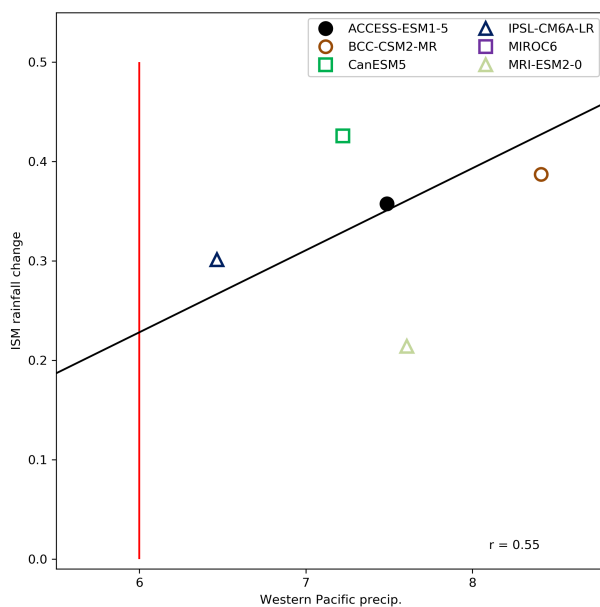
10.9.7 Example plots

Fig. 25: Scatter plot of the simulated tropical western Pacific precipitation (mm d^{-1}) versus projected average ISM (Indian Summer Monsoon) rainfall changes under the ssp585 scenario. The red line denotes the observed present-day western Pacific precipitation and the inter-model correlation (r) is shown. (CMIP6).
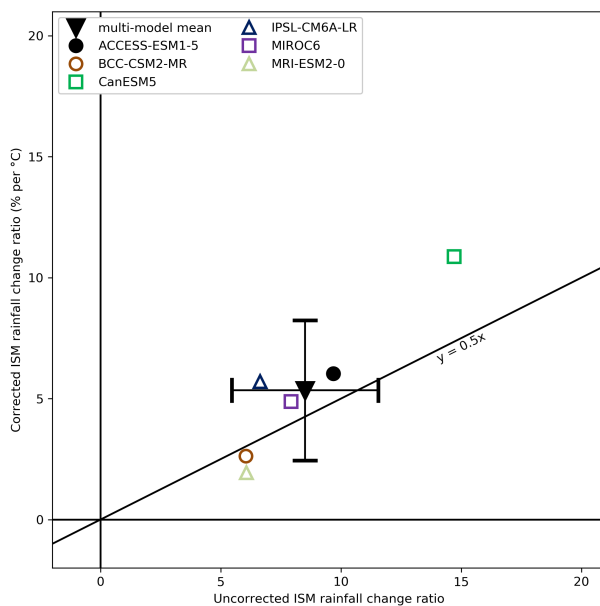

Fig. 26: Scatter plot of the uncorrected versus corrected average ISM (Indian Summer Monsoon) rainfall change ratios (% per degree Celsius of global SST warming). The error bars for the Multi-model mean indicate the standard deviation spread among models and the 2:1 line (y = 0.5x) is used to illustrate the Multi-model mean reduction in projected rainfall increase. (CMIP6).
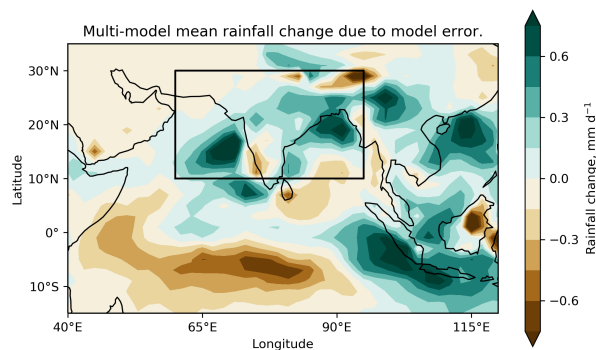
Fig. 27: Multi-model mean rainfall change due to model error. Box displays the area used to define the average ISM (Indian Summer Monsoon) rainfall. Precipitation changes are normalized by the corresponding global mean SST increase for each model. (CMIP6).
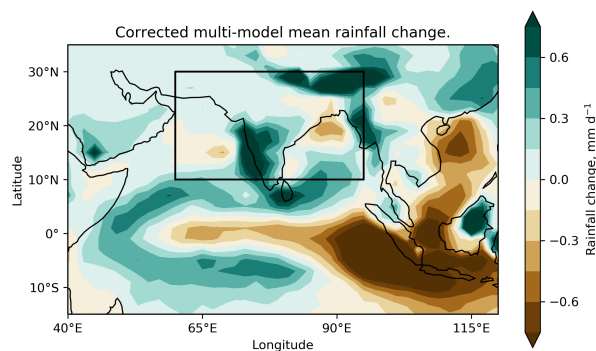

Fig. 28: Corrected multi-model mean rainfall change. Box displays the area used to define the average ISM (Indian Summer Monsoon) rainfall. Precipitation changes are normalized by the corresponding global mean SST increase for each model. (CMIP6).

IPCC

11.1 IPCC AR5 Chapter 9 (selected figures)

11.1.1 Overview

The goal of this recipe is to collect diagnostics to reproduce Chapter 9 of AR5, so that the plots can be readily reproduced and compared to previous CMIP versions. In this way we can next time start with what was available in the previous round and can focus on developing more innovative methods of analysis rather than constantly having to "re-invent the wheel".

The plots are produced collecting the diagnostics from individual recipes. The following figures from Flato et al. (2013) can currently be reproduced:

- Figure 9.2 a,b,c: Annual-mean surface air temperature for the period 1980-2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from ERA-Interim (Dee et al., 2011), c) mean absolute model error with respect to the climatology from ERA-Interim.

- Figure 9.3: Seasonality (December-January-February minus June-July-August) of surface (2 m) air temperature (°C) for the period 1980-2005. (a) Multi-model mean for the historical experiment. (b) Multi-model mean of absolute seasonality. (c) Difference between the multi-model mean and the ERA-Interim reanalysis seasonality. (d) Difference between the multi-model mean and the ERA-Interim absolute seasonality.

- Figure 9.4: Annual-mean precipitation rate (mm day-1) for the period 1980-2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from the Global Precipitation Climatology Project (Adler et al., 2003), c) multi-model mean absolute error with respect to observations, and d) multi-model mean error relative to the multi-model mean precipitation ifself.

- Figure 9.5: Climatological (1985-2005) annual-mean cloud radiative effects in Wm-2 for the CMIP5 models against CERES EBAF (2001-2011) in Wm-2. Top row shows the shortwave effect; middle row the longwave effect, and bottom row the net effect. Multi-model-mean biases against CERES EBAF 2.6 are shown on the left, whereas the right panels show zonal averages from CERES EBAF 2.6 (black), the individual CMIP5 models (thin gray lines), and the multi-model mean (thick red line).

- Figure 9.6: Centred pattern correlations between models and observations for the annual mean climatology over the period 1980–1999. Results are shown for individual CMIP3 (black) and CMIP5 (blue) models as thin dashes, along with the corresponding ensemble average (thick dash) and median (open circle). The four variables shown are surface air temperature (TAS), top of the atmosphere (TOA) outgoing longwave radiation (RLUT), precipitation (PR) and TOA shortwave cloud radiative effect (SW CRE). The correlations between the reference and alternate observations are also shown (solid green circles).

- Figure 9.8: Observed and simulated time series of the anomalies in annual and global mean surface temperature. All anomalies are differences from the 1961-1990 time-mean of each individual time series. The reference period 1961-1990 is indicated by yellow shading; vertical dashed grey lines represent times of major volcanic

eruptions. Single simulations for CMIP5 models (thin lines); multi-model mean (thick red line); different observations (thick black lines). Dataset pre-processing like described in Jones et al., 2013.

- Figure 9.14: Sea surface temperature plots for zonal mean error, equatorial (5 deg north to 5 deg south) mean error, and multi model mean for zonal error and equatorial mean.

- Figure 9.24: Time series of (a) Arctic and (b) Antarctic sea ice extent; trend distributions of (c) September Arctic and (d) February Antarctic sea ice extent.

- Figure 9.26: Ensemble-mean global ocean carbon uptake (a) and global land carbon uptake (b) in the CMIP5 ESMs for the historical period 1900–2005. For comparison, the observation-based estimates provided by the Global Carbon Project (GCP) are also shown (thick black line). The confidence limits on the ensemble mean are derived by assuming that the CMIP5 models are drawn from a t-distribution. The grey areas show the range of annual mean fluxes simulated across the model ensemble. This figure includes results from all CMIP5 models that reported land CO2 fluxes, ocean CO2 fluxes, or both (Anav et al., 2013).

- Figure 9.27: Simulation of global mean (a) atmosphere–ocean CO2 fluxes ("fgCO2") and (b) net atmosphere–land CO2 fluxes ("NBP"), by ESMs for the period 1986–2005. For comparison, the observation-based estimates provided by Global Carbon Project (GCP) and the Japanese Meteorological Agency (JMA) atmospheric inversion are also shown. The error bars for the ESMs and observations represent interannual variability in the fluxes, calculated as the standard deviation of the annual means over the period 1986–2005.

- Figure 9.42a: Equilibrium climate sensitivity (ECS) against the global mean surface air temperature, both for the period 1961-1990 and for the pre-industrial control runs.

- Figure 9.42b: Transient climate response (TCR) against equilibrium climate sensitivity (ECS).

- Figure 9.45a: Scatterplot of springtime snow-albedo effect values in climate change vs. springtime $d(alpha_s)/d(T_s)$ values in the seasonal cycle in transient climate change experiments (Hall and Qu, 2006).

11.1.2 Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_flato13ipcc.yml

Diagnostics are stored in esmvaltool/diag_scripts/

- carbon_cycle/main.ncl: See *Land and ocean components of the global carbon cycle*.

- climate_metrics/ecs.py: See *Equilibrium climate sensitivity*.

- clouds/clouds_bias.ncl: global maps of the multi-model mean and the multi-model mean bias (Fig. 9.2, 9.4)

- clouds/clouds_isccp: global maps of multi-model mean minus observations + zonal averages of individual models, multi-model mean and observations (Fig. 9.5)

- ipcc_ar5/ch09_fig09_3.ncl: multi-model mean seasonality of near-surface temperature (Fig. 9.3)

- ipcc_ar5/ch09_fig09_6.ncl: calculating pattern correlations of annual mean climatologies for one variable (Fig 9.6 preprocessing)

- ipcc_ar5/ch09_fig09_6_collect.ncl: collecting pattern correlation for each variable and plotting correlation plot (Fig 9.6)

- ipcc_ar5/tsline.ncl: time series of the global mean (anomaly) (Fig. 9.8)

- ipcc_ar5/ch09_fig09_14.py: Zonally averaged and equatorial SST (Fig. 9.14)

- seaice/seaice_tsline.ncl: Time series of sea ice extent (Fig. 9.24a/b)

- seaice/seaice_trends.ncl: Trend distributions of sea ice extent (Fig 9.24c/d)

- ipcc_ar5/ch09_fig09_42a.py: ECS vs. surface air temperature (Fig. 9.42a)

- ipcc_ar5/ch09_fig09_42b.py: TCR vs. ECS (Fig. 9.42b)

- emergent_constraints/snowalbedo.ncl: snow-albedo effect (Fig. 9.45a)

11.1.3 User settings in recipe

1. Script carbon_cycle/main.ncl

   See *Land and ocean components of the global carbon cycle*.

2. Script climate_metrics/ecs.py

   See *Equilibrium climate sensitivity*.

3. Script clouds/clouds_bias.ncl

4. Script clouds_bias.ncl

   Required settings (scripts)

   Optional settings (scripts)

   - plot_abs_diff: additionally also plot absolute differences (true, false)

   - plot_rel_diff: additionally also plot relative differences (true, false)

   - projection: map projection, e.g., Mollweide, Mercator

   - timemean: time averaging, i.e. "seasonalclim" (DJF, MAM, JJA, SON), "annualclim" (annual mean)

   - Required settings (variables)*

   - reference_dataset: name of reference datatset

   Optional settings (variables)

   - long_name: description of variable

   Color tables

   - variable "tas": diag_scripts/shared/plot/rgb/ipcc-tas.rgb, diag_scripts/shared/plot/rgb/ipcc-tas-delta.rgb

   - variable "pr-mmday": diag_scripts/shared/plots/rgb/ipcc-precip.rgb, diag_scripts/shared/plot/rgb/ipcc-precip-delta.rgb

5. Script clouds/clouds_ipcc.ncl

   Required settings (scripts)

   Optional settings (scripts)

   - explicit_cn_levels: contour levels

   - mask_ts_sea_ice: true = mask T < 272 K as sea ice (only for variable "ts"); false = no additional grid cells masked for variable "ts"

   - projection: map projection, e.g., Mollweide, Mercator

   - styleset: style set for zonal mean plot ("CMIP5", "DEFAULT")

   - timemean: time averaging, i.e. "seasonalclim" (DJF, MAM, JJA, SON), "annualclim" (annual mean)

   - valid_fraction: used for creating sea ice mask (mask_ts_sea_ice = true): fraction of valid time steps required to mask grid cell as valid data

---

Required settings (variables)

- reference_dataset: name of reference data set

Optional settings (variables)

- long_name: description of variable
- units: variable units

Color tables

- variables "pr", "pr-mmday": diag_scripts/shared/plot/rgb/ipcc-precip-delta.rgb

6. Script ipcc_ar5/tsline.ncl

Required settings for script

- styleset: as in diag_scripts/shared/plot/style.ncl functions

Optional settings for script

- time_avg: type of time average (currently only "yearly" and "monthly" are available).
- ts_anomaly: calculates anomalies with respect to the defined period; for each gird point by removing the mean for the given calendar month (requiring at least 50% of the data to be non-missing)
- ref_start: start year of reference period for anomalies
- ref_end: end year of reference period for anomalies
- ref_value: if true, right panel with mean values is attached
- ref_mask: if true, model fields will be masked by reference fields
- region: name of domain
- plot_units: variable unit for plotting
- y-min: set min of y-axis
- y-max: set max of y-axis
- mean_nh_sh: if true, calculate first NH and SH mean
- volcanoes: if true, lines of main volcanic eruptions will be added
- run_ave: if not equal 0 than calculate running mean over this number of years
- header: if true, region name as header

Required settings for variables

Optional settings for variables

- reference_dataset: reference dataset; REQUIRED when calculating anomalies

Color tables

- e.g. diag_scripts/shared/plot/styles/cmip5.style

7. Script ipcc_ar5/ch09_fig09_3.ncl

Required settings for script

Optional settings for script

- projection: map projection, e.g., Mollweide, Mercator (default = Robinson)

Required settings for variables

- reference_dataset: name of reference observation

Optional settings for variables

- map_diff_levels: explicit contour levels for plotting

8. Script ipcc_ar5/ch09_fig09_6.ncl

Required settings for variables

- reference_dataset: name of reference observation

Optional settings for variables

- alternative_dataset: name of alternative observations

9. Script ipcc_ar5/ch09_fig09_6_collect.ncl

Required settings for script

Optional settings for script

- diag_order: List of diagnostic names in the order variables should appear on x-axis

10. Script seaice/seaice_trends.ncl

Required settings (scripts)

- month: selected month (1, 2, . . . , 12) or annual mean ("A")
- region: region to be analyzed ("Arctic" or "Antarctic")

Optional settings (scripts)

- fill_pole_hole: fill observational hole at North pole, Default: False

Optional settings (variables)

- ref_model: array of references plotted as vertical lines

11. Script seaice/seaice_tsline.ncl

Required settings (scripts)

- region: Arctic, Antarctic
- month: annual mean (A), or month number (3 = March, for Antarctic; 9 = September for Arctic)

Optional settings (scripts)

- styleset: for plot_type cycle only (cmip5, cmip6, default)
- multi_model_mean: plot multi-model mean and standard deviation (default: False)
- EMs_in_lg: create a legend label for individual ensemble members (default: False)
- fill_pole_hole: fill polar hole (typically in satellite data) with sic = 1 (default: False)

12. Script ipcc_ar5/ch09_fig09_42a.py

Required settings for script

Optional settings for script

- axes_functions: `dict` containing methods executed for the plot's `matplotlib.axes.Axes` object.
- dataset_style: name of the style file (located in `esmvaltool.diag_scripts.shared.plot.styles_python`).
- matplotlib_style: name of the matplotlib style file (located in `esmvaltool.diag_scripts.shared.plot.styles_python.matplotlib`).
- save: `dict` containing keyword arguments for the function `matplotlib.pyplot.savefig()`.
- seaborn_settings: Options for seaborn's `set()` method (affects all plots), see https://seaborn.pydata.org/generated/seaborn.set.html.

13. Script ipcc_ar5/ch09_fig09_42b.py

    Required settings for script

    none

    Optional settings for script

    - dataset_style: name of the style file (located in `esmvaltool.diag_scripts.shared.plot.styles_python`).
    - log_x: Apply logarithm to X axis (ECS).
    - log_y: Apply logarithm to Y axis (TCR).
    - seaborn_settings: Options for seaborn's `set()` method (affects all plots), see https://seaborn.pydata.org/generated/seaborn.set.html.

14. Script emergent_constraints/snowalbedo.ncl

    Required settings for script

    - exp_presentday: name of present-day experiment (e.g. "historical")
    - exp_future: name of climate change experiment (e.g. "rcp45")

    Optional settings for script

    - diagminmax: observational uncertainty (min and max)
    - legend_outside: create extra file with legend (true, false)
    - styleset: e.g. "CMIP5" (if not set, this diagnostic will create its own color table and symbols for plotting)
    - suffix: string to be added to output filenames
    - xmax: upper limit of x-axis (default = automatic)
    - xmin: lower limit of x-axis (default = automatic)
    - ymax: upper limit of y-axis (default = automatic)
    - ymin: lower limit of y-axis (default = automatic)

    Required settings for variables

    - ref_model: name of reference data set

    Optional settings for variables

    none

11.1.4 Variables

- areacello (fx, longitude latitude)

- fgco2 (ocean, monthly mean, longitude latitude time)

- nbp (ocean, monthly mean, longitude latitude time)

- pr (atmos, monthly mean, longitude latitude time)

- rlut, rlutcs (atmos, monthly mean, longitude latitude time)

- rsdt (atmos, monthly mean, longitude latitude time)

- rsuscs, rsdscs (atmos, monthly mean, longitude latitude time)

- rsut, rsutcs (atmos, monthly mean, longitude latitude time)

- sic (ocean-ice, monthly mean, longitude latitude time)

- tas (atmos, monthly mean, longitude latitude time)

- tos (ocean, monthly mean, longitude, latitude, time)

11.1.5 Observations and reformat scripts

Note: (1) obs4mips data can be used directly without any preprocessing; (2) see headers of reformat scripts for non-obs4mips data for download instructions.

- CERES-EBAF (rlut, rlutcs, rsut, rsutcs - obs4mips)

- ERA-Interim (tas, ta, ua, va, zg, hus - esmvaltool/cmorizers/obs/cmorize_obs_ERA-Interim.ncl)

- GCP (fgco2, nbp - esmvaltool/cmorizers/obs/cmorize_obs_gcp.py)

- GPCP-SG (pr - obs4mips)

- JMA-TRANSCOM (fgco2, nbp - esmvaltool/cmorizers/obs/cmorize_obs_jma_transcom.py)

- HadCRUT4 (tas - esmvaltool/cmorizers/obs/cmorize_obs_hadcrut4.ncl)

- HadISST (sic, tos - esmvaltool/cmorizers/obs/cmorize_obs_hadisst.ncl)

- ISCCP-FH (rsuscs, rsdscs, rsdt - esmvaltool/cmorizers/obs/cmorize_obs_isccp_fh.ncl)

11.1.6 References

- Flato, G., J. Marotzke, B. Abiodun, P. Braconnot, S.C. Chou, W. Collins, P. Cox, F. Driouech, S. Emori, V. Eyring, C. Forest, P. Gleckler, E. Guilyardi, C. Jakob, V. Kattsov, C. Reason and M. Rummukainen, 2013: Evaluation of Climate Models. In: Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change [Stocker, T.F., D. Qin, G.-K. Plattner, M. Tignor, S.K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex and P.M. Midgley (eds.)]. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA.

- Hall, A., and X. Qu, 2006: Using the current seasonal cycle to constrain snow albedo feedback in future climate change, Geophys. Res. Lett., 33, L03502, doi:10.1029/2005GL025127.

- Jones et al., 2013: Attribution of observed historical near-surface temperature variations to anthropogenic and natural causes using CMIP5 simulations. Journal of Geophysical Research: Atmosphere, 118, 4001-4024, doi:10.1002/jgrd.50239.
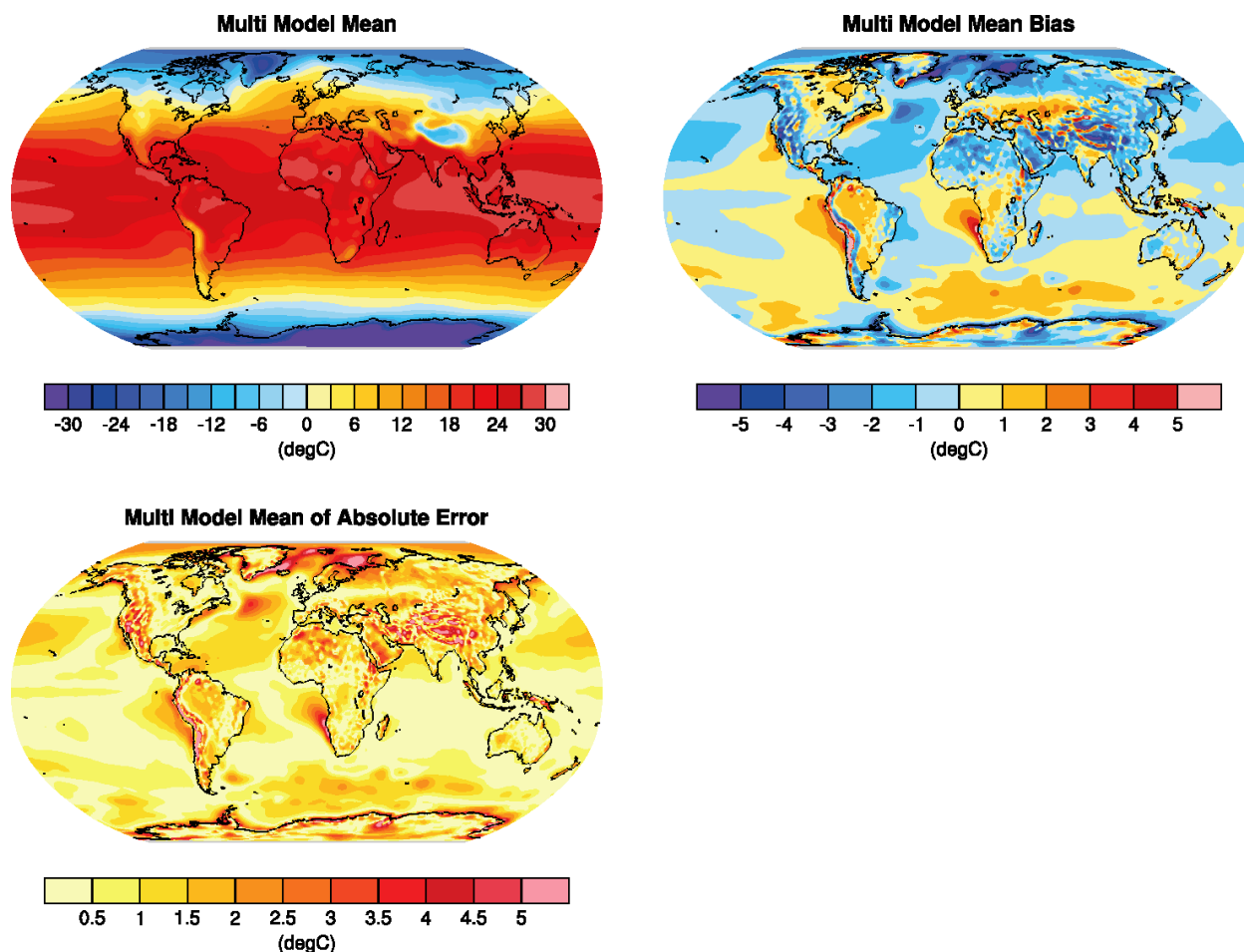
11.1.7 Example plots

Fig. 1: Figure 9.2 a,b,c: Annual-mean surface air temperature for the period 1980-2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from ERA-Interim (Dee et al., 2011), c) mean absolute model error with respect to the climatology from ERA-Interim.

11.2 IPCC AR5 Chapter 12 (selected figures)

11.2.1 Overview

The goal is to create a standard recipe for creating selected Figures from IPCC AR5 Chapter 12 on "Long-term Climate Change: Projections, Commitments and Irreversibility". These include figures showing the change in a variable between historical and future periods, e.g. maps (2D variables), zonal means (3D variables), timeseries showing the change in certain variables from historical to future periods for multiple scenarios, and maps visualizing change in variables normalized by global mean temperature change (pattern scaling) as in Collins et al., 2013.
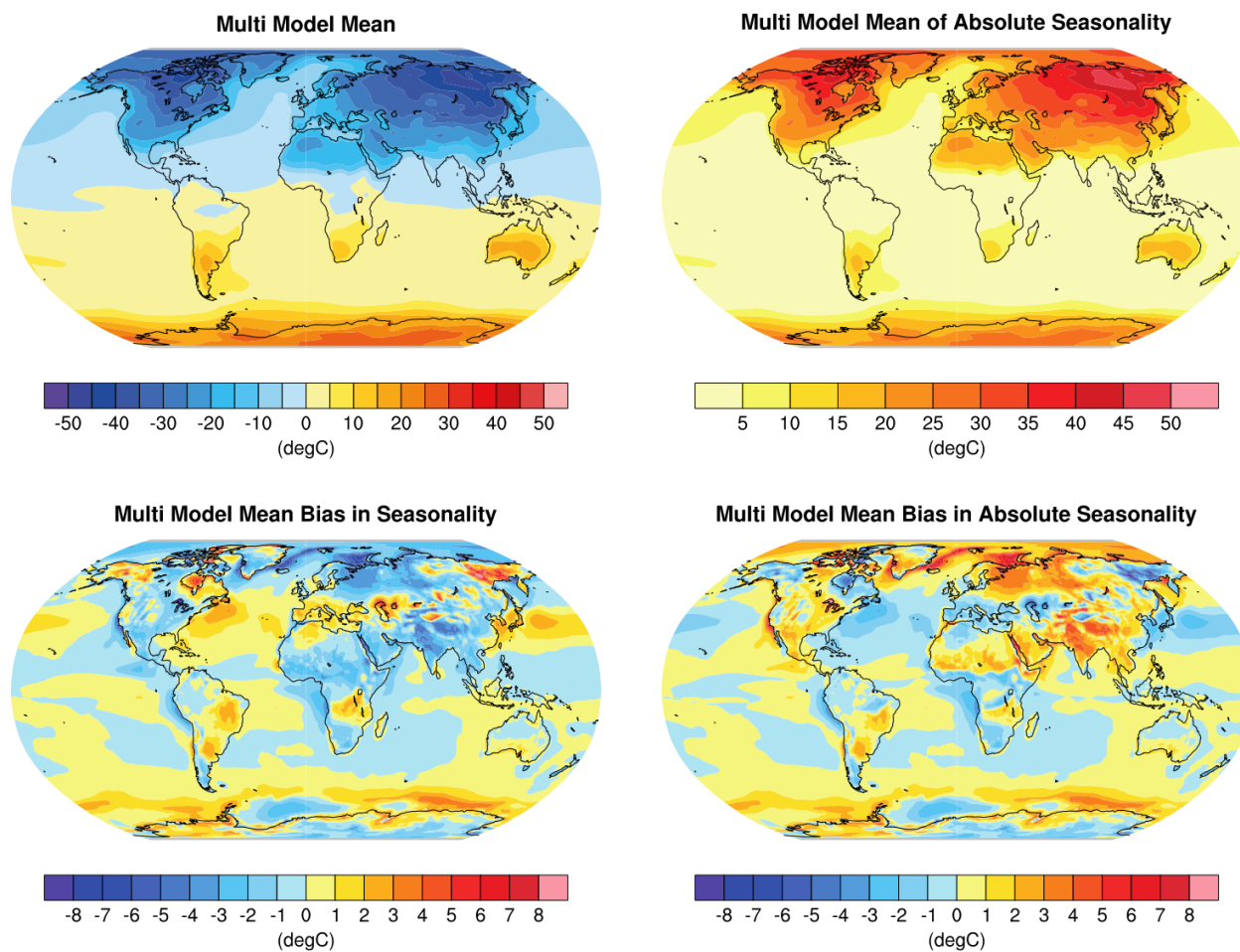
Fig. 2: Figure 9.3: Multi model values for seasonality of near-surface temperature, from top left to bottom right: mean, mean of absolute seasonality, mean bias in seasonality, mean bias in absolute seasonality. Reference dataset: ERA-Interim.
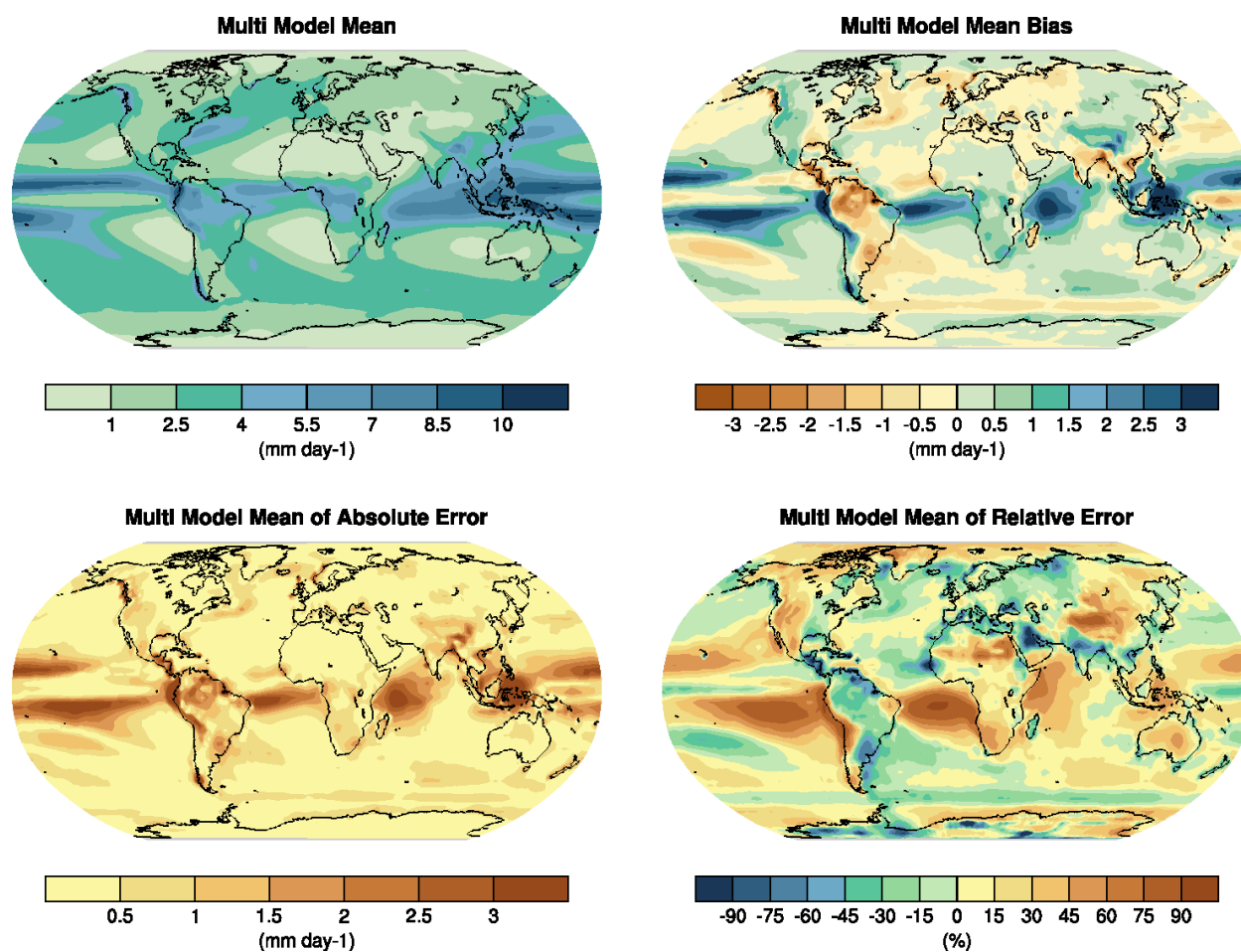
Multi Model Mean

Multi Model Mean Bias

Multi Model Mean of Absolute Error

Multi Model Mean of Relative Error

Fig. 3: Figure 9.4: Annual-mean precipitation rate (mm day-1) for the period 1980-2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from the Global Precipitation Climatology Project (Adler et al., 2003), c) multi-model mean absolute error with respect to observations, and d) multi-model mean error relative to the multi-model mean precipitation ifself.
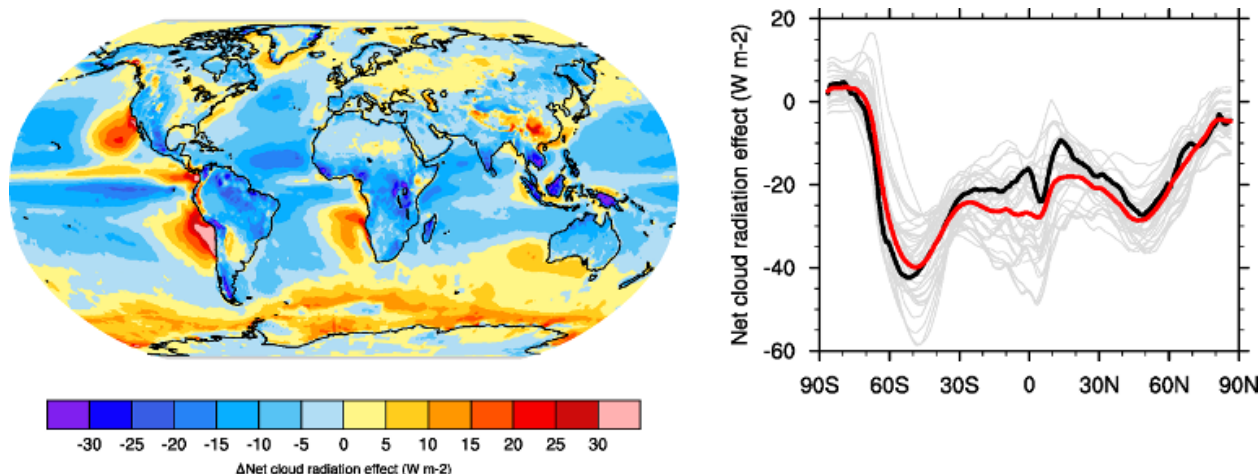
Fig. 4: Figure 9.5: Climatological (1985-2005) annual-mean cloud radiative effects in Wm-2 for the CMIP5 models against CERES EBAF (2001-2011) in Wm-2. Top row shows the shortwave effect; middle row the longwave effect, and bottom row the net effect. Multi-model-mean biases against CERES EBAF 2.6 are shown on the left, whereas the right panels show zonal averages from CERES EBAF 2.6 (black), the individual CMIP5 models (thin gray lines), and the multi-model mean (thick red line).

11.2.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_collins13ipcc.yml

Diagnostics are stored in diag_scripts/

- ipcc_ar5/ch12_map_diff_each_model_fig12-9.ncl: calculates the difference between future and historical runs for one scenario for each given model individually on their native grid and plots all of them in one Figure. As in Figure 12.9 in AR5.

- ipcc_ar5/ch12_ts_line_mean_spread.ncl: calculates time series for one variable, change in future relative to base period in historical, multi-model mean as well as spread around it (as standard deviation).

- ipcc_ar5/ch12_plot_ts_line_mean_spread.ncl: plots the timeseries multi-model mean and spread calculated above. As in Figure 12.5 in AR5.

- ipcc_ar5/ch12_calc_IAV_for_stippandhatch.ncl: calculates the interannual variability over piControl runs, either over the whole time period or in chunks over some years.

- ipcc_ar5/ch12_calc_map_diff_mmm_stippandhatch.ncl: calculates the difference between future and historical periods for each given model and then calculates multi-model mean as well as significance. Significant is where the multi-model mean change is greater than two standard deviations of the internal variability and where at least 90% of the models agree on the sign of change. Not significant is where the multi-model mean change is less than one standard deviation of internal variability.

- ipcc_ar5/ch12_plot_map_diff_mmm_stipp.ncl: plots multi-model mean maps calculated above including stippling where significant and hatching where not significant. As in Figure 12.11 in AR5.

- ipcc_ar5/ch12_calc_zonal_cont_diff_mmm_stippandhatch.ncl: calculates zonal means and the difference between future and historical periods for each given model and then calculates multi-model mean as well as significance as above.

- ipcc_ar5/ch12_plot_zonal_diff_mmm_stipp.ncl: plots the multi-model mean zonal plots calculated above including stippling where significant and hatching where not significant. As in Figure 12.12 in AR5.
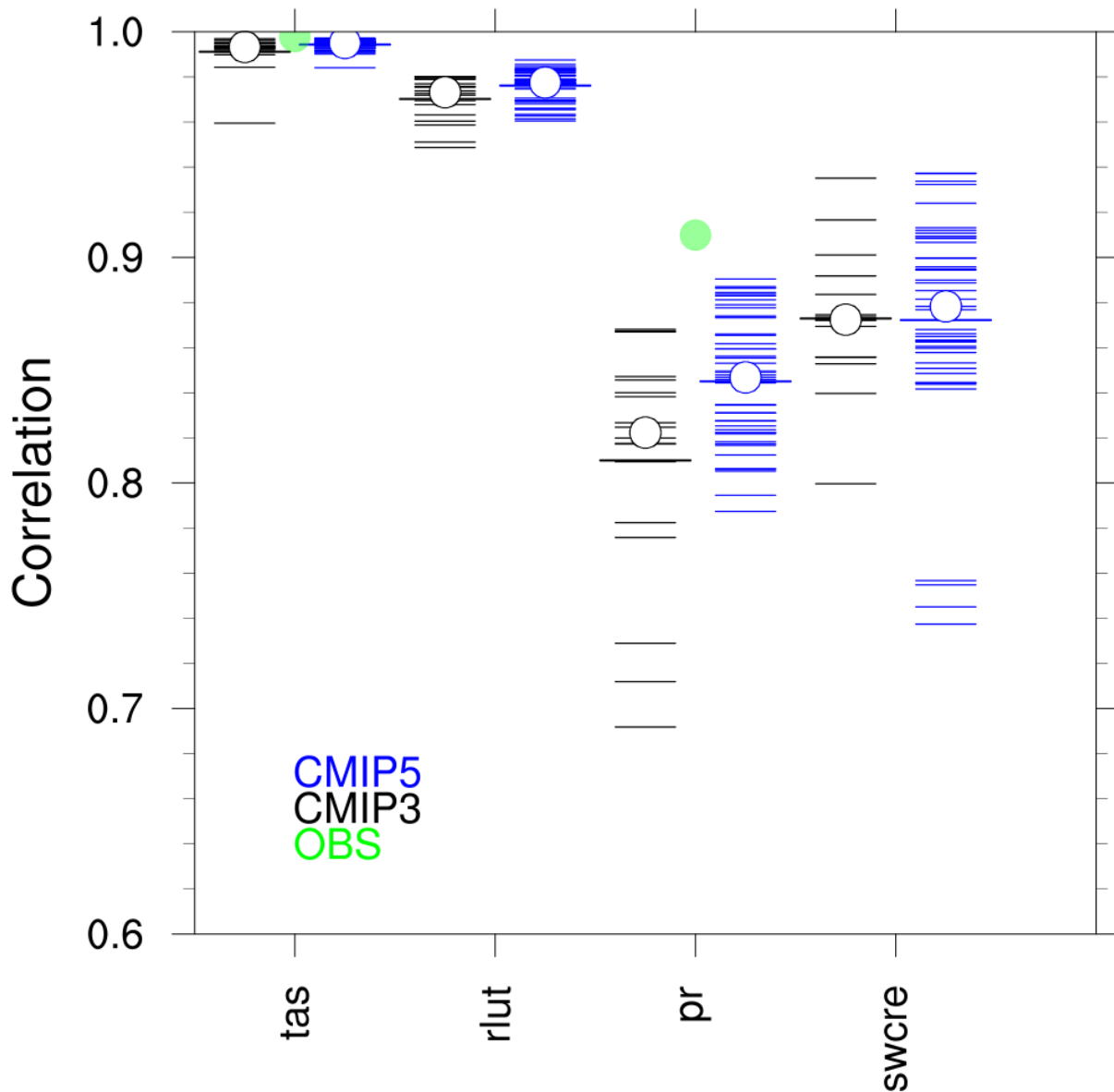
Fig. 5: Figure 9.6: Centred pattern correlations between models and observations for the annual mean climatology over the period 1980–1999. Results are shown for individual CMIP3 (black) and CMIP5 (blue) models as thin dashes, along with the corresponding ensemble average (thick dash) and median (open circle). The four variables shown are surface air temperature (TAS), top of the atmosphere (TOA) outgoing longwave radiation (RLUT), precipitation (PR) and TOA shortwave cloud radiative effect (SW CRE). The correlations between the reference and alternate observations are also shown (solid green circles).
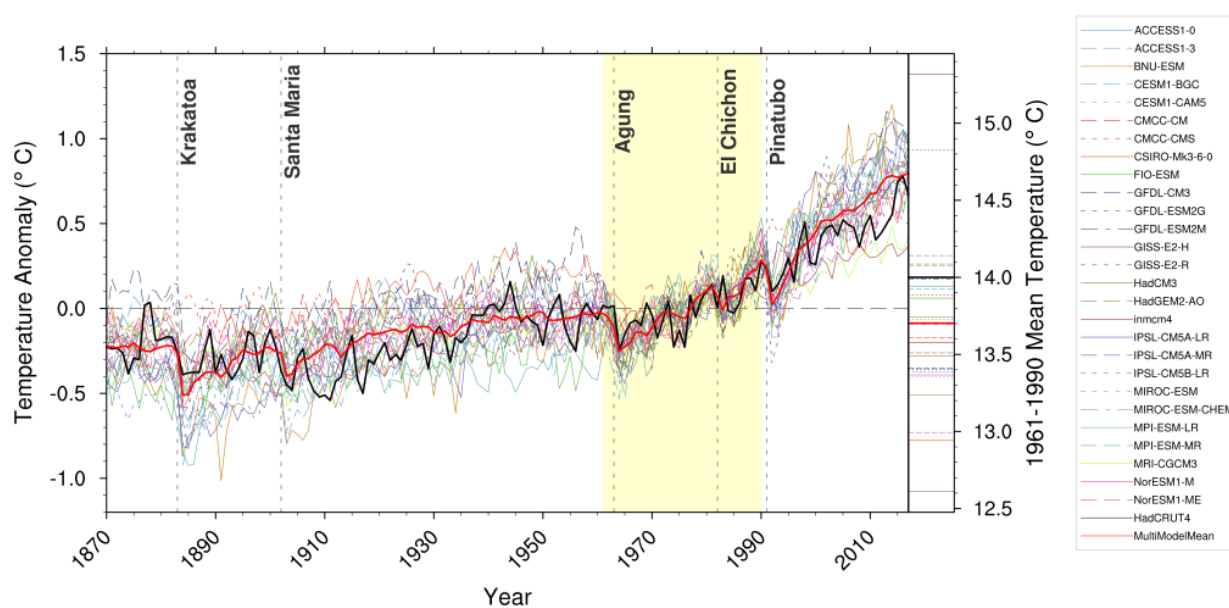
Fig. 6: Figure 9.8: Observed and simulated time series of the anomalies in annual and global mean surface temperature. All anomalies are differences from the 1961-1990 time-mean of each individual time series. The reference period 1961-1990 is indicated by yellow shading; vertical dashed grey lines represent times of major volcanic eruptions. Single simulations for CMIP5 models (thin lines); multi-model mean (thick red line); different observations (thick black lines). Dataset pre-processing like described in Jones et al., 2013.
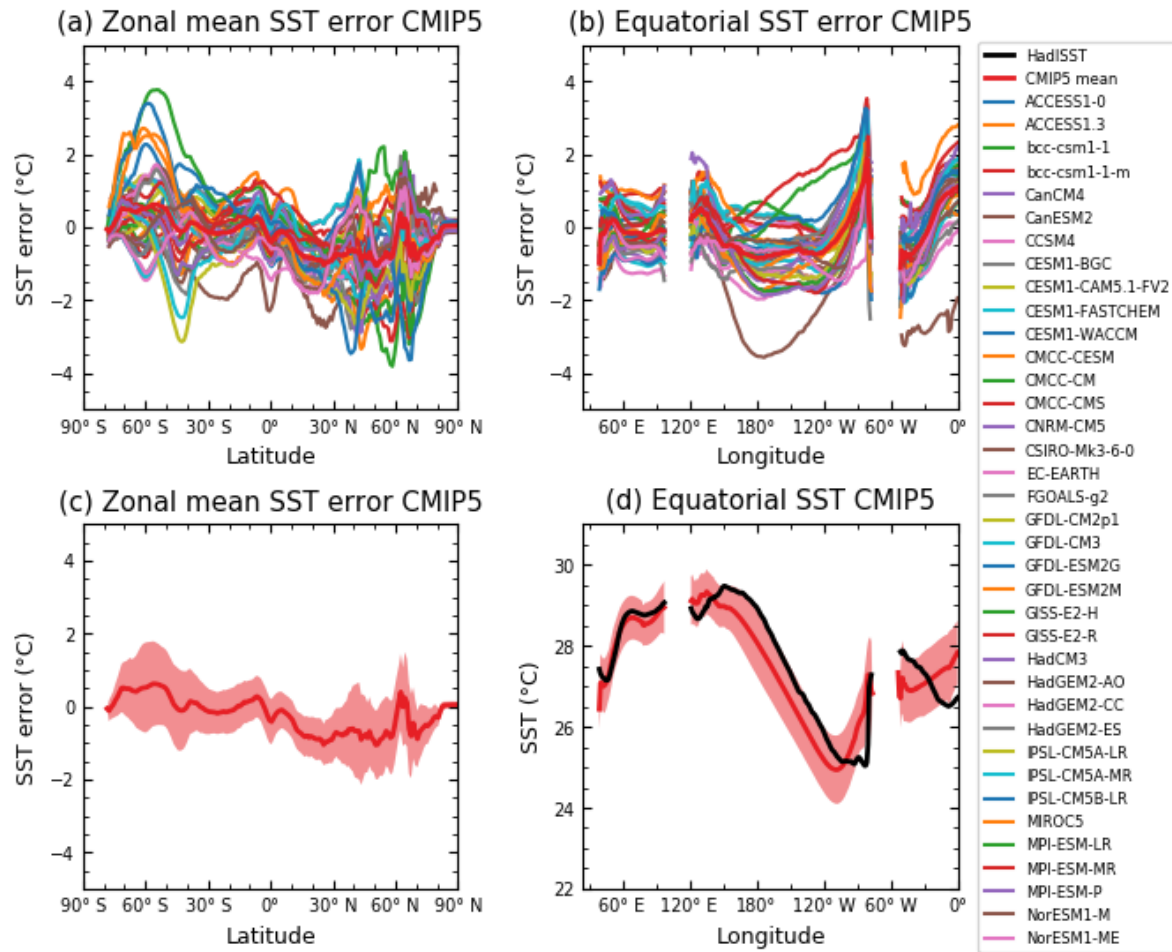
Fig. 7: Figure 9.14: (a) Zonally averaged sea surface temperature (SST) error in CMIP5 models. (b) Equatorial SST error in CMIP5 models. (c) Zonally averaged multi-model mean SST error for CMIP5 together with inter-model standard deviation (shading). (d) Equatorial multi-model mean SST in CMIP5 together with inter-model standard deviation (shading) and observations (black). Model climatologies are derived from the 1979-1999 mean of the historical simulations. The Hadley Centre Sea Ice and Sea Surface Temperature (HadISST) (Rayner et al., 2003) observational climatology for 1979-1999 is used as a reference for the error calculation (a), (b), and (c); and for observations in (d).
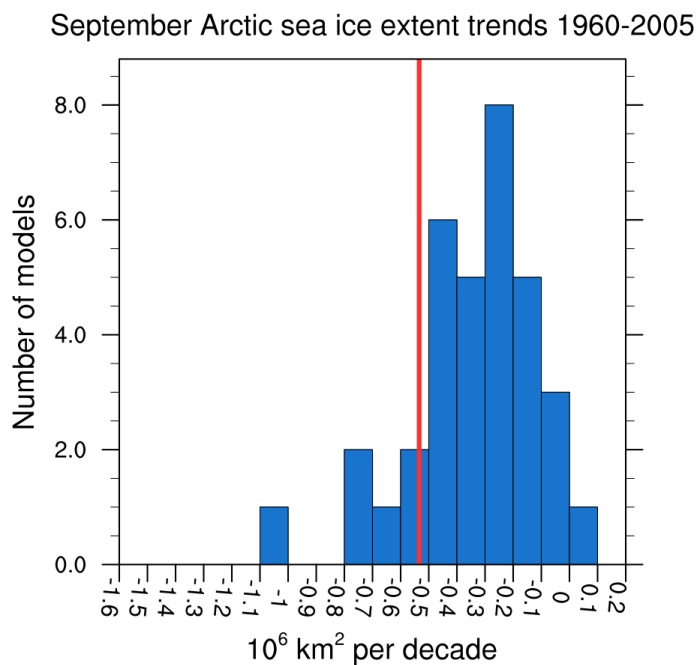
Fig. 8: Figure 9.24c: Sea ice extent trend distribution for the Arctic in September.
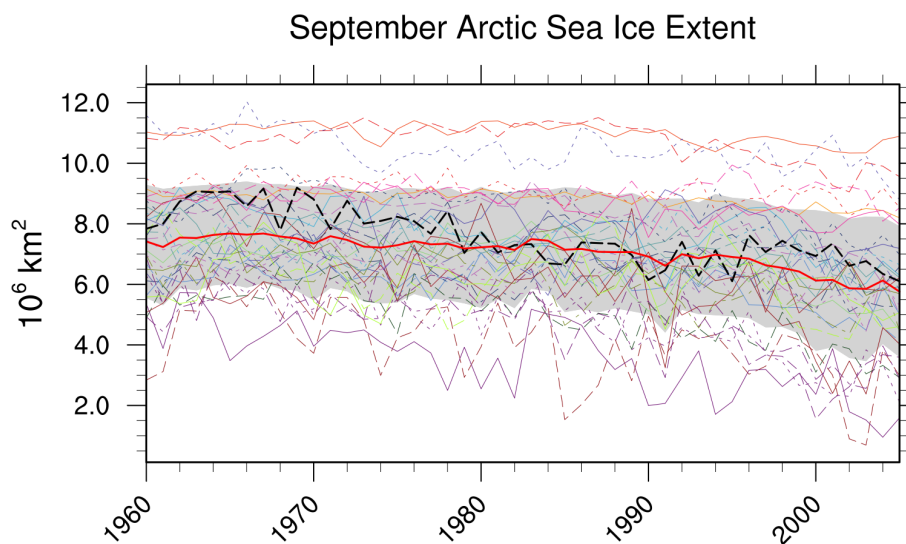

Fig. 9: Figure 9.24a: Time series of total sea ice area and extent (accumulated) for the Arctic in September including multi-model mean and standard deviation.
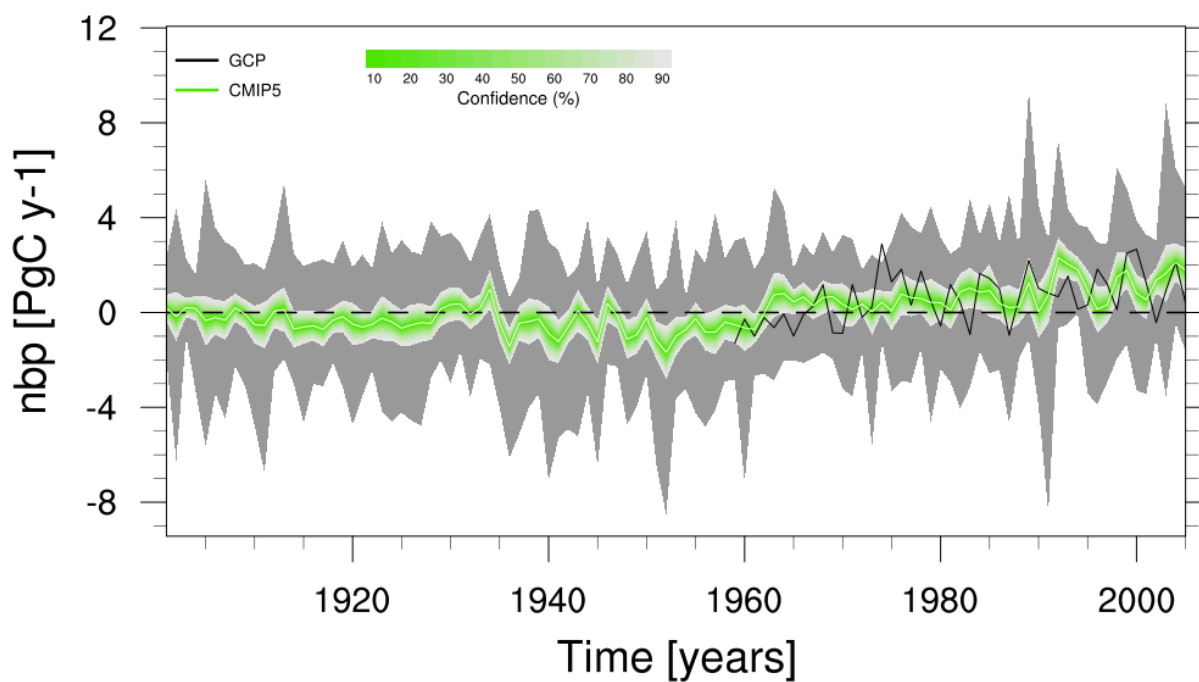
Fig. 10: Figure 9.26 (bottom): Ensemble-mean global land carbon uptake in the CMIP5 ESMs for the historical period 1900–2005. For comparison, the observation-based estimates provided by the Global Carbon Project (GCP) are also shown (black line). The confidence limits on the ensemble mean are derived by assuming that the CMIP5 models come from a t-distribution. The grey areas show the range of annual mean fluxes simulated across the model ensemble.
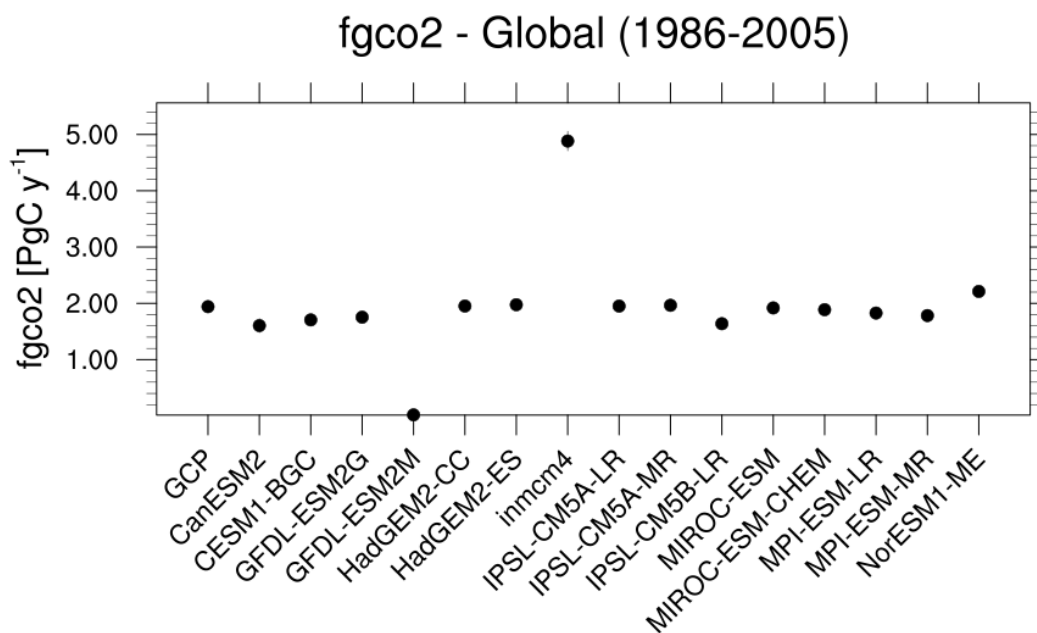
Fig. 11: Figure 9.27 (top): Simulation of global mean atmosphere–ocean CO2 fluxes ("fgCO2") by ESMs for the period 1986–2005. For comparison, the observation-based estimates provided by Global Carbon Project (GCP) are also shown. The error bars for the ESMs and observations represent interannual variability in the fluxes, calculated as the standard deviation of the annual means over the period 1986–2005.
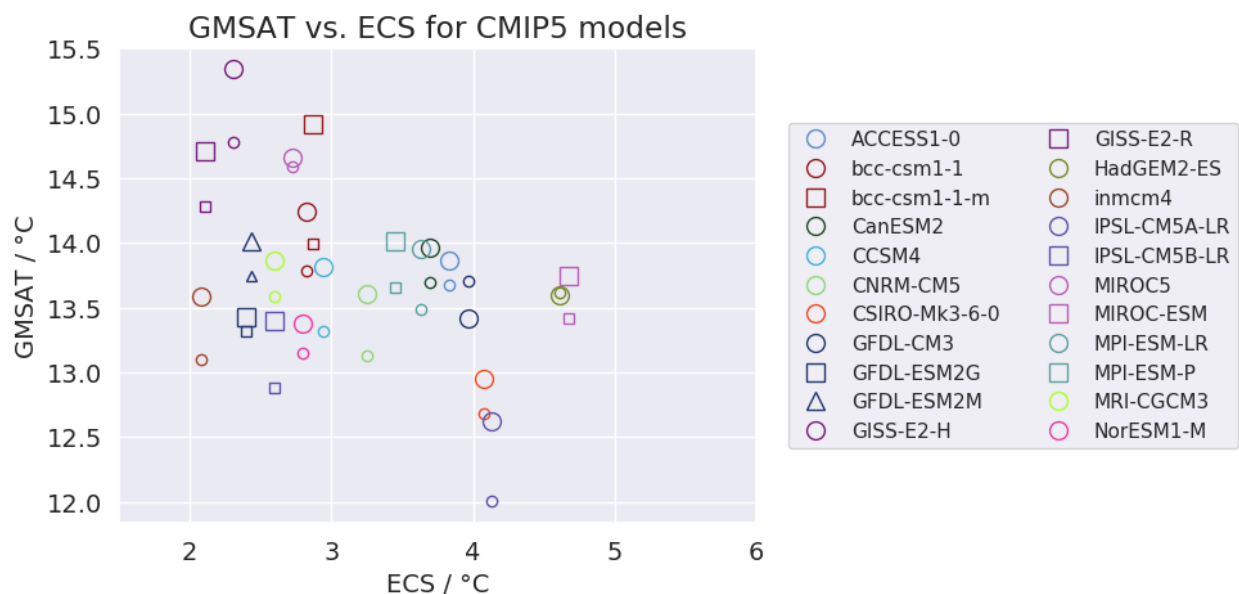
Fig. 12: Figure 9.42a: Equilibrium climate sensitivity (ECS) against the global mean surface air temperature of CMIP5 models, both for the period 1961-1990 (larger symbols) and for the pre-industrial control runs (smaller symbols).
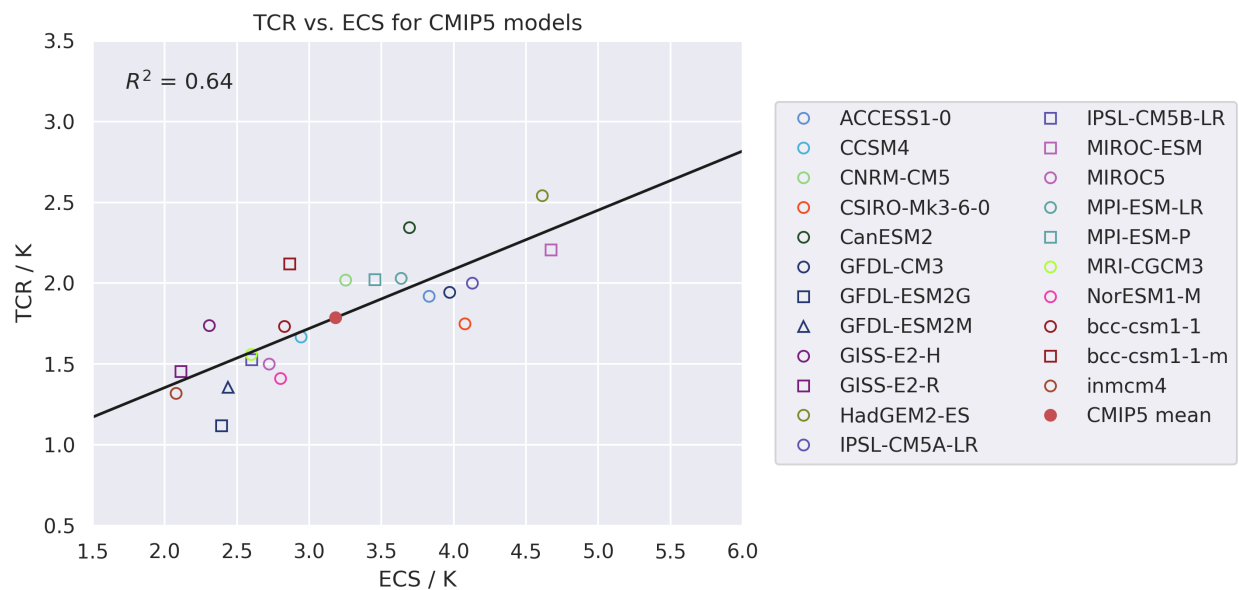

Fig. 13: Figure 9.42b: Transient climate response (TCR) against equilibrium climate sensitivity (ECS) for CMIP5 models.
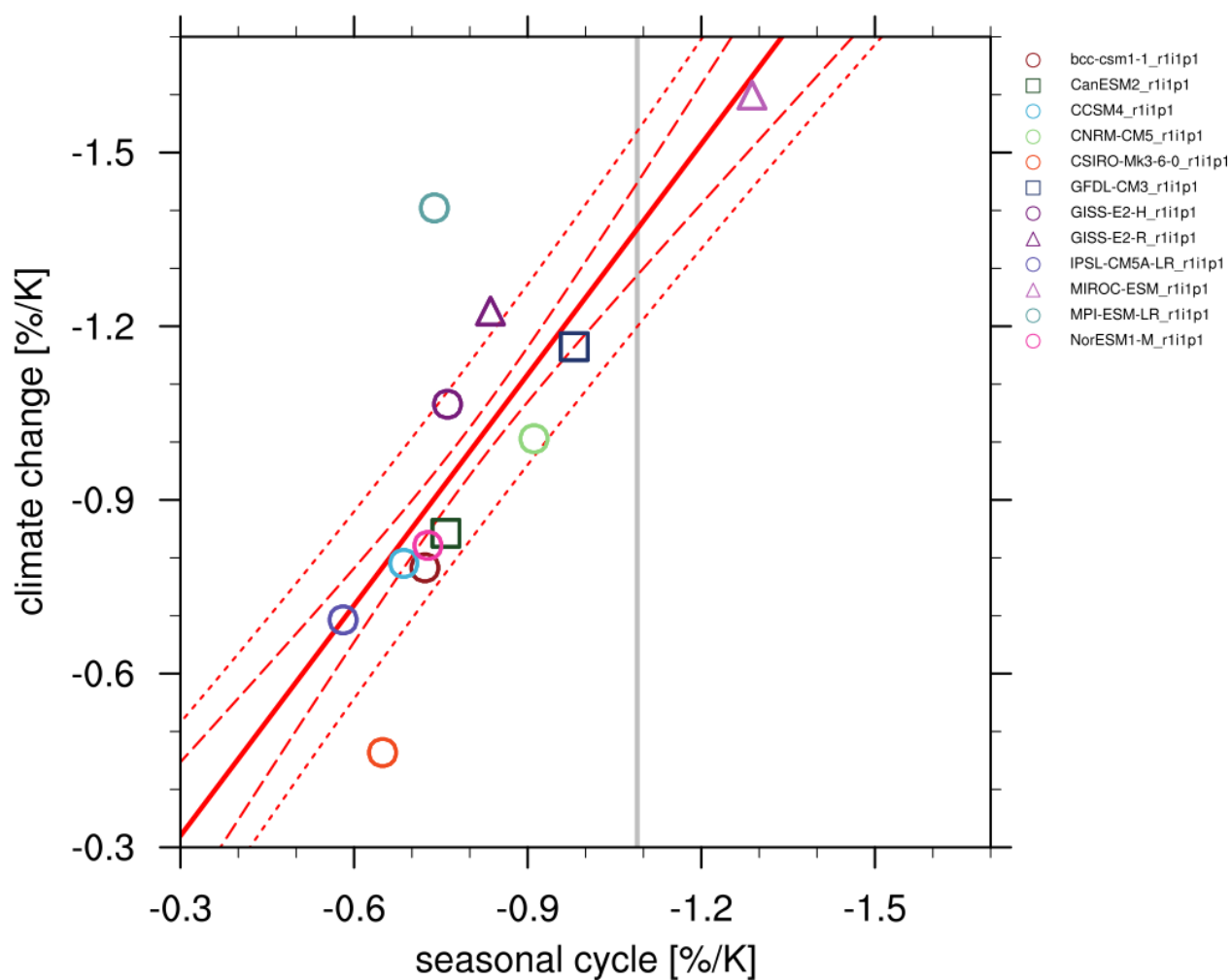
Fig. 14: Figure 9.45a: Scatterplot of springtime snow-albedo effect values in climate change vs. springtime $\Delta\alpha_s/\Delta T_s$ values in the seasonal cycle in transient climate change experiments (CMIP5 historical experiments: 1901-2000, RCP4.5 experiments: 2101-2200).

- ipcc_ar5/ch12_calc_map_diff_scaleT_mmm_stipp.ncl: calculates the change in variable between future and historical period normalized by gloabl mean temperature change of each given model and scenario. Then averages over all realizations and calculates significance. Significant is where the mean change averaged over all realizations is larger than the 95% percentile of the distribution of models (assumed to be gaussian). Can be plotted using ipcc_ar5/ch12_plot_map_diff_mmm_stipp.ncl.

- seaice/seaice_ecs.ncl: scatter plot of historical trend in September Arctic sea ice extent (SSIE) vs historical long-term mean SSIE (similar to Fig. 12.31a in AR5) and historical SSIE trend vs YOD RCP8.5 (similar to Fig. 12.31d in AR5).

- seaice/seaice_yod.ncl: calculation of year of near disappearance of Arctic sea ice (similar to Fig 12.31e in AR5)

- ipcc_ar5/ch12_snw_area_change_fig12-32.ncl: calculate snow area extent in a region (e.g Northern Hemisphere) and season (e.g. Northern Hemisphere spring March & April) relative to a reference period (e.g 1986-2005) and spread over models as in Fig. 12.32 of IPCC AR5. Can be plotted using ipcc_ar5/ch12_plot_ts_line_mean_spread.ncl.

11.2.3 User settings

1. Script ipcc_ar5/ch12_map_diff_each_model_fig12-9.ncl

   Required settings (script)

   - time_avg: time averaging ("annualclim", "seasonalclim")
   - experiment: IPCC Scenario, used to pair historical and rcp runs from same model

   Optional settings (script)

   - projection: map projection, any valid ncl projection, default = Robinson
   - max_vert: maximum number of plots in vertical
   - max_hori: maximum number of plots in horizontal
   - title: plot title
   - colormap: alternative colormap, path to rgb file or ncl name
   - diff_levs: list with contour levels for plots
   - span: span whole colormap? (True, False, default = False)

   Required settings (variables)

   - project: CMIP5 (or CMIP6)
   - mip: variable mip, generally Amon or Omon

2. Script ipcc_ar5/ch12_ts_line_mean_spread.ncl

   Required settings (script)

   - scenarios: list with scenarios included in figure
   - syears: list with start years in time periods (e.g. start of historical period and rcps)
   - eyears: list with end years in time periods (end year of historical runs and rcps)
   - begin_ref_year: start year of reference period (e.g. 1986)
   - end_ref_year: end year of reference period (e.g 2005)
   - label: list with labels to use in legend depending on scenarios

   Optional settings (script)

- spread: how many standard deviations to calculate the spread with? default is 1., ipcc tas used 1.64
- model_nr: save number of model runs per period and scenario in netcdf to print in plot? (True, False, default = False)
- ts_minlat: minimum latitude if not global
- ts_maxlat: maximum latitude if not global
- ts_minlon: minimum longitude if not global
- ts_maxlon: maximum longitude if not global

Required settings (variables)

- project: CMIP5 (or CMIP6)
- mip: variable mip, generally Amon or Omon

3. Script ipcc_ar5/ch12_plot_ts_line_mean_spread.ncl:

Required settings (script)

- ancestors: variable and diagnostics that calculated data to be plotted

Optional settings (script)

- title: specify plot title
- yaxis: specify y-axis title
- ymin: minimim value on y-axis, default calculated from data
- ymax: maximum value on y-axis
- colormap: alternative colormap, path to rgb file or ncl name

4. Script ipcc_ar5/ch12_calc_IAV_for_stippandhatch.ncl:

Required settings (script)

- time_avg: time averaging ("annualclim", "seasonalclim"), needs to be consistent with calculation in ch12_calc_map_diff_mmm_stippandhatch.ncl

Optional settings (script)

- periodlength: length of period in years to calculate variability over, default is total time period
- iavmode: calculate IAV from multi-model mean or save individual models ("each": save individual models, "mmm": multi-model mean, default), needs to be consistent with ch12_calc_map_diff_mmm_stippandhatch.ncl

Required settings (variables)

- project: CMIP5 (or CMIP6)
- mip: variable mip, generally Amon or Omon
- exp: piControl
- preprocessor: which preprocessor to use, depends on dimension of variable, for 2D preprocessor only needs to regrid, for 3D we need to extract levels either based on reference_dataset or specify levels.

Optional settings (variables)

- reference_dataset: the reference dataset for level extraction in case of 3D variables.

5. Script ipcc_ar5/ch12_calc_map_diff_mmm_stippandhatch.ncl:

Required settings (script)

- ancestors: variable and diagnostics that calculated interannual variability for stippling and hatching

- time_avg: time averaging ("annualclim", "seasonalclim")

- scenarios: list with scenarios to be included

- periods: list with start years of periods to be included

- label: list with labels to use in legend depending on scenarios

Optional settings (script)

- seasons: list with seasons index if time_avg "seasonalclim" (then required), DJF:0, MAM:1, JJA:2, SON:3

- iavmode: calculate IAV from multi-model mean or save individual models ("each": save individual models, "mmm": multi-model mean, default), needs to be consistent with ch12_calc_IAV_for_stippandhatch.ncl

- percent: determines if difference expressed in percent (0, 1, default = 0)

Required settings (variables)

- project: CMIP5 (or CMIP6)

- mip: variable mip, generally Amon or Omon

- preprocessor: which preprocessor to use, preprocessor only needs to regrid

6. Script ipcc_ar5/ch12_plot_map_diff_mmm_stipp.ncl:

Required settings (script)

- ancestors: variable and diagnostics that calculated field to be plotted

Optional settings (script)

- projection: map projection, any valid ncl projection, default = Robinson

- diff_levs: list with explicit levels for all contour plots

- max_vert: maximum number of plots in vertical

- max_hori: maximum number of plots in horizontal

- model_nr: save number of model runs per period and scenario in netcdf to print in plot? (True, False, default = False)

- colormap: alternative colormap, path to rgb file or ncl name

- span: span whole colormap? (True, False, default = True)

- sig: plot stippling for significance? (True, False)

- not_sig: plot hatching for uncertainty? (True, False)

- pltname: alternative name for output plot, default is diagnostic + varname + time_avg

- units: units written next to colorbar, e.g (~F35~J~F~C)

7. Script ipcc_ar5/ch12_calc_zonal_cont_diff_mmm_stippandhatch.ncl:

Required settings (script)

- ancestors: variable and diagnostics that calculated interannual variability for stippling and hatching

- time_avg: time averaging ("annualclim", "seasonalclim")

- scenarios: list with scenarios to be included

- periods: list with start years of periods to be included

- label: list with labels to use in legend depending on scenarios

Optional settings (script)

- base_cn: if want contours of base period as contour lines, need to save base period field (True, False)

- seasons: list with seasons index if time_avg "seasonalclim" (then required), DJF:0, MAM:1, JJA:2, SON:3

- iavmode: calculate IAV from multi-model mean or save individual models ("each": save individual models, "mmm": multi-model mean, default), needs to be consistent with ch12_calc_IAV_for_stippandhatch.ncl

- percent: determines if difference expressed in percent (0, 1, default = 0)

Required settings (variables)

- project: CMIP5 (or CMIP6)

- mip: variable mip, generally Amon or Omon

- preprocessor: which preprocessor to use, preprocessor needs to regrid, extract leves and calculate the zonal mean.

Optional settings (variables)

- reference_dataset: the reference dataset for level extraction

8. Script ipcc_ar5/ch12_plot_zonal_diff_mmm_stipp.ncl:

Required settings (script)

- ancestors: variable and diagnostics that calculated field to be plotted

Optional settings (script)

- diff_levs: list with explicit levels for all contour plots

- max_vert: maximum number of plots in vertical

- max_hori: maximum number of plots in horizontal

- model_nr: save number of model runs per period and scenario in netcdf to print in plot? (True, False, default = False)

- colormap: alternative colormap, path to rgb file or ncl name

- span: span whole colormap? (True, False, default = True)

- sig: plot stippling for significance? (True, False)

- not_sig: plot hatching for uncertainty? (True, False)

- pltname: alternative name for output plot, default is diagnostic + varname + time_avg

- units: units written next to colorbar in ncl strings, e.g (m s~S~-1~N~)

- if base_cn: True in ch12_calc_zonal_cont_diff_mmm_stippandhatch.ncl further settings to control contour lines:

  - base_cnLevelSpacing: spacing between contour levels

  - base_cnMinLevel: minimum contour line

  - base_cnMaxLevel: maximum contour line

9. Script ipcc_ar5/ch12_calc_map_diff_scaleT_mmm_stipp.ncl:

Required settings (script)

- time_avg: time averaging ("annualclim", "seasonalclim")

- scenarios: list with scenarios to be included

- periods: list with start years of periods to be included

- label: list with labels to use in legend depending on scenarios

Optional settings (script)

- seasons: list with seasons index if time_avg "seasonalclim" (then required), DJF:0, MAM:1, JJA:2, SON:3

- percent: determines if difference expressed in percent (0, 1, default = 0)

Required settings (variables)

- project: CMIP5 (or CMIP6)

- mip: variable mip, generally Amon or Omon

- preprocessor: which preprocessor to use, preprocessor only needs to regrid

10. Script ipcc_ar5/ch12_snw_area_change_fig12-32.ncl:

Required settings (script)

- scenarios: list with scenarios included in figure

- syears: list with start years in time periods (e.g. start of historical period and rcps)

- eyears: list with end years in time periods (end year of historical runs and rcps)

- begin_ref_year: start year of reference period (e.g. 1986)

- end_ref_year: end year of reference period (e.g 2005)

- months: first letters of months included in analysis? e.g. for MA (March + April) for Northern Hemisphere

- label: list with labels to use in legend depending on scenarios

Optional settings (script)

- spread: how many standard deviations to calculate the spread with? default is 1., ipcc tas used 1.64

- model_nr: save number of model runs per period and scenario in netcdf to print in plot? (True, False, default = False)

- colormap: alternative colormap, path to rgb file or ncl name

- ts_minlat: minimum latitude if not global

- ts_maxlat: maximum latitude if not global

- ts_minlon: minimum longitude if not global

- ts_maxlon: maximum longitude if not global

Required settings (variables)

- project: CMIP5 (or CMIP6)

- mip: variable mip, LImon

- fx_files: [sftlf, sftgif]

11. Script seaice/seaice_ecs.ncl

Required settings (scripts)

- hist_exp: name of historical experiment (string)

- month: selected month (1, 2, . . . , 12) or annual mean ("A")

- rcp_exp: name of RCP experiment (string)

- region: region to be analyzed ("Arctic" or "Antarctic")

Optional settings (scripts)

- fill_pole_hole: fill observational hole at North pole (default: False)

- styleset: color style (e.g. "CMIP5")

Optional settings (variables)

- reference_dataset: reference dataset

12. Script seaice/seaice_yod.ncl

Required settings (scripts)

- month: selected month (1, 2, . . . , 12) or annual mean ("A")

- region: region to be analyzed ("Arctic" or "Antarctic")

Optional settings (scripts)

- fill_pole_hole: fill observational hole at North pole, Default: False

- wgt_file: netCDF containing pre-determined model weights

Optional settings (variables)

- ref_model: array of references plotted as vertical lines

11.2.4 Variables

Note: These are the variables tested and used in IPCC AR5. However, the code is flexible and in theory other variables of the same kind can be used.

- areacello (fx, longitude latitude)

- clt (atmos, monthly mean, longitude latitude time)

- evspsbl (atmos, monthly mean, longitude latitude time)

- hurs (atmos, monthly mean, longitude latitude time)

- mrro (land, monthly mean, longitude latitude time)

- mrsos (land, monthly mean, longitude latitude time)

- pr (atmos, monthly mean, longitude latitude time)

- psl (atmos, monthly mean, longitude latitude time)

- rlut, rsut, rtmt (atmos, monthly mean, longitude latitude time)

- sic (ocean-ice, monthly mean, longitude latitude time)

- snw (land, monthly mean, longitude latitude time)

- sos (ocean, monthly mean, longitude latitude time)

- ta (atmos, monthly mean, longitude latitude lev time)

- tas (atmos, monthly mean, longitude latitude time)

- thetao (ocean, monthly mean, longitude latitude lev time)

- ua (atmos, monthly mean, longitude latitude lev time)

11.2.5 Observations and reformat scripts

- HadISST (sic - esmvaltool/utils/cmorizers/obs/cmorize_obs_HadISST.ncl)

11.2.6 Reference

- Collins, M., R. Knutti, J. Arblaster, J.-L. Dufresne, T. Fichefet, P. Friedlingstein, X. Gao, W.J. Gutowski, T. Johns, G. Krinner, M. Shongwe, C. Tebaldi, A.J. Weaver and M. Wehner, 2013: Long-term Climate Change: Projections, Commitments and Irreversibility. In: Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change [Stocker, T.F., D. Qin, G.-K. Plattner, M. Tignor, S.K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex and P.M. Midgley (eds.)]. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA.
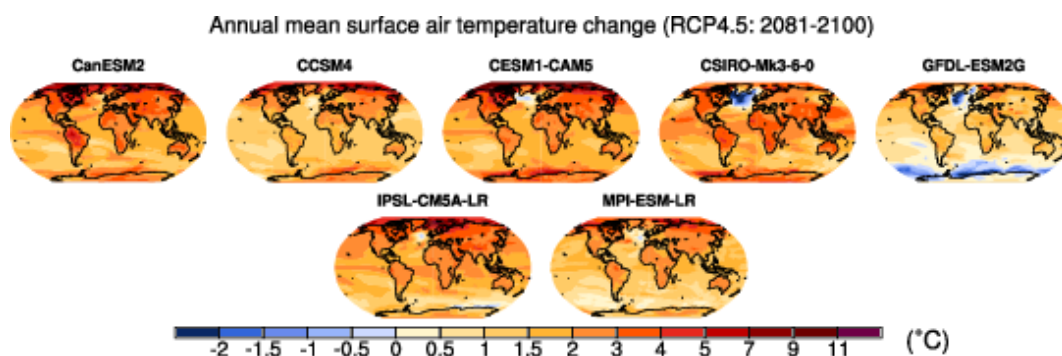
11.2.7 Example plots

Fig. 15: Surface air temperature change in 2081–2100 displayed as anomalies with respect to 1986–2005 for RCP4.5 from individual CMIP5 models.
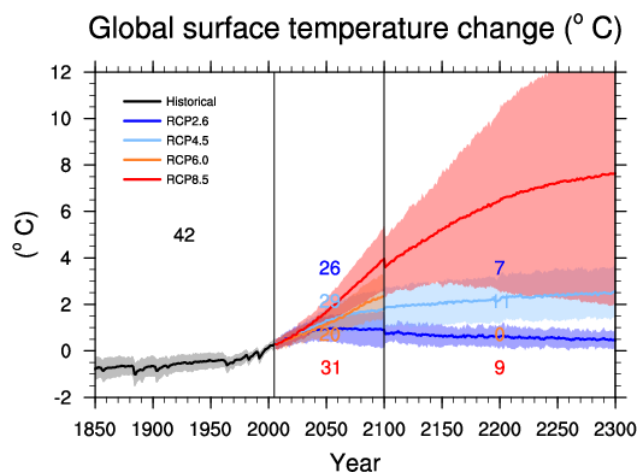

Fig. 16: Time series of global annual mean surface air temperature anomalie (relative to 1986–2005) from CMIP5 concentration-driven experiments.
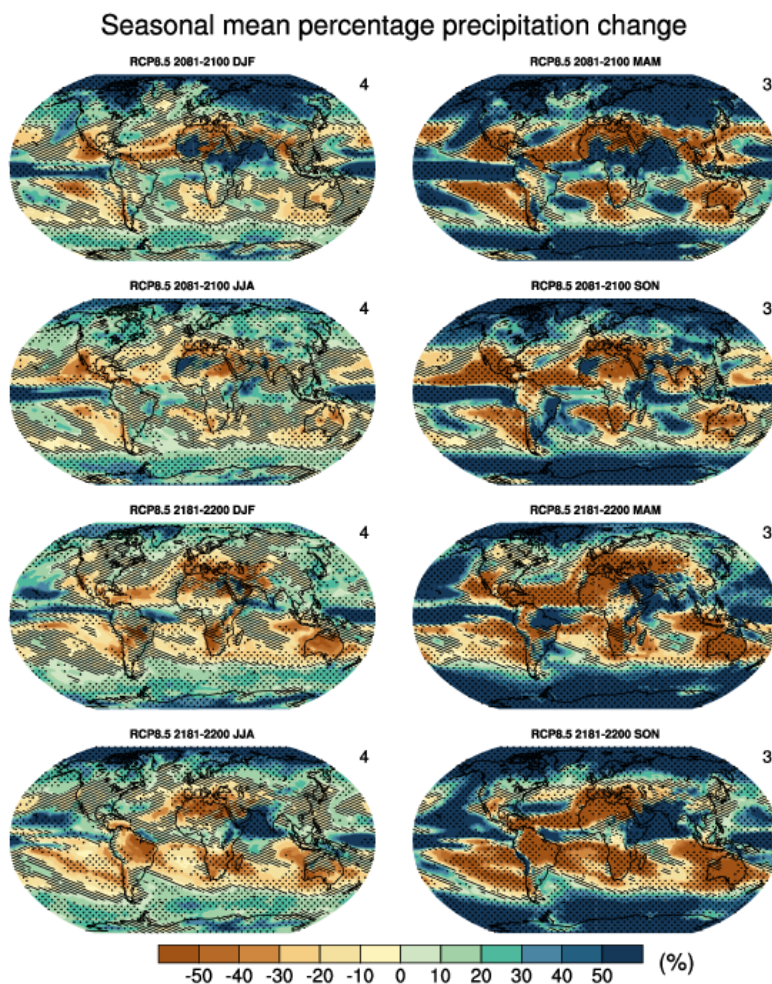
Fig. 17: Multi-model CMIP5 average percentage change in seasonal mean precipitation relative to the reference period 1986–2005 averaged over the periods 2081–2100 and 2181–2200 under the RCP8.5 forcing scenario. Hatching indicates regions where the multi-model mean change is less than one standard deviation of internal variability. Stippling indicates regions where the multi-model mean change is greater than two standard deviations of internal variability and where at least 90% of models agree on the sign of change
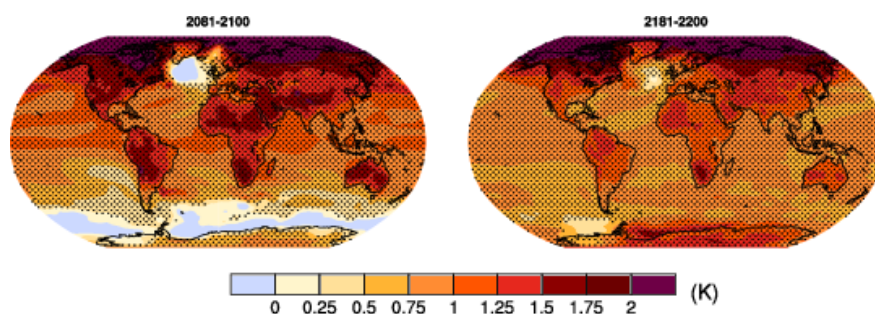

Fig. 18: Temperature change patterns scaled to 1°C of global mean surface temperature change.
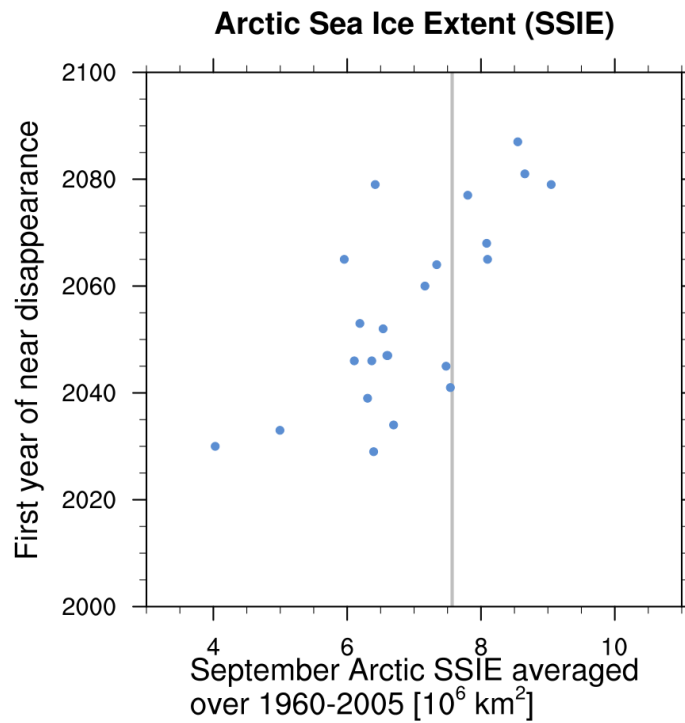
Fig. 19: Scatter plot of mean historical September Arctic sea ice extent vs 1st year of disappearance (RCP8.5) (similar to IPCC AR5 Chapter 12, Fig. 12.31a).
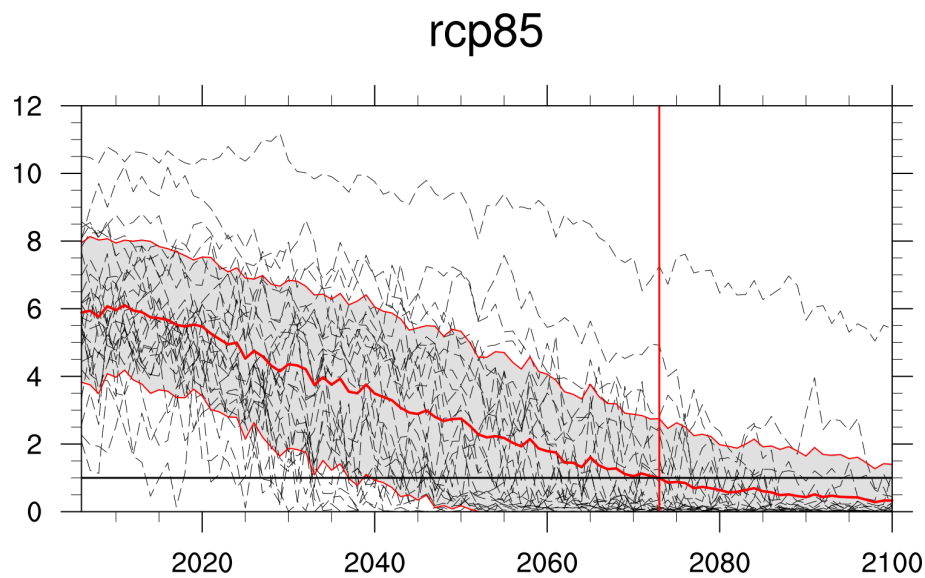

Fig. 20: Time series of September Arctic sea ice extent for individual CMIP5 models, multi-model mean and multi-model standard deviation, year of disappearance (similar to IPCC AR5 Chapter 12, Fig. 12.31e).

LAND

12.1 Landcover - Albedo

12.1.1 Overview

The diagnostic determines the coefficients of multiple linear regressions fitted between the albedo values and the tree, shrub, short vegetation (crops and grasses) fractions of each grid cell within spatially moving windows encompassing 5x5 model grid cells. Solving these regressions provides the albedo values for trees, shrubs and short vegetation (crops and grasses) from which the albedo changes associated with transitions between these three landcover types are derived. The diagnostic distinguishes between snow-free and snow-covered grid cells.

12.1.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_albedolandcover.yml

Diagnostics are stored in diag_scripts/landcover/

- albedolandcover.py

12.1.3 User settings

Several parameters can be set in the recipe

12.1.4 Variables

- rsus (atmos, monthly mean, time latitude longitude)
- rsds (atmos, monthly mean, time latitude longitude)
- snc (landice, monthly mean, time latitude longitude)
- grassFrac (land, monthly mean, time latitude longitude)
- treeFrac (land, monthly mean, time latitude longitude)
- shrubFrac (land, monthly mean, time latitude longitude)
- cropFrac (land, monthly mean, time latitude longitude)
- pastureFrac (land, monthly mean, time latitude longitude)

12.1.5 Observations and reformat scripts

A reformatting script for observational data is available here:

- cmorize_obs_duveiller2018.py

12.1.6 References

- Duveiller, G., Hooker, J. and Cescatti, A., 2018a. A dataset mapping the potential biophysical effects of vegetation cover change. Scientific Data, 5: 180014.

- Duveiller, G., Hooker, J. and Cescatti, A., 2018b. The mark of vegetation change on Earth's surface energy balance. Nature communications, 9(1): 679.
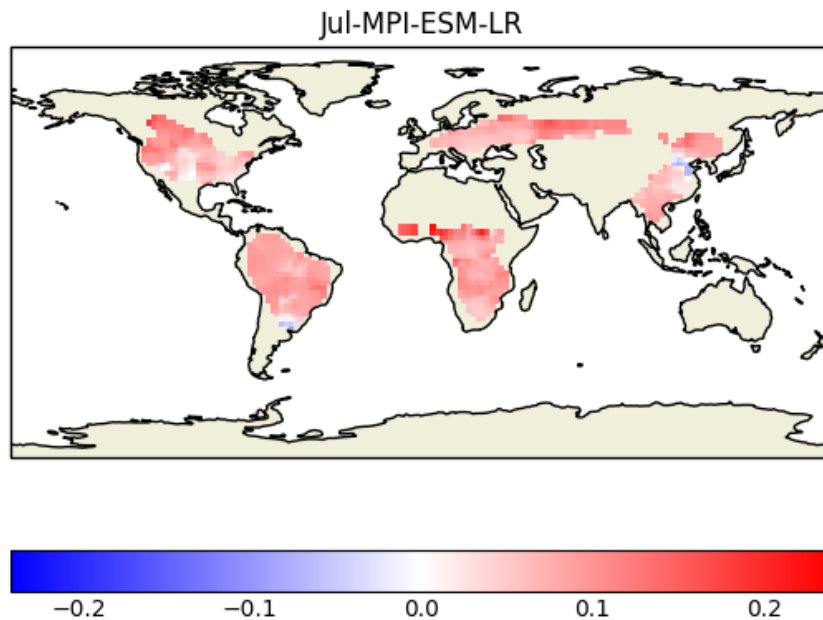
12.1.7 Example plots

Fig. 1: Example of albedo change from tree to crop and grass for the CMIP5 model MPI-ESM-LR derived for the month of July and averaged over the years 2000 to 2004.

12.2 Turnover time of carbon over land ecosystems

12.2.1 Overview

This recipe evaluates the turnover time of carbon over land ecosystems (tau_ctotal) based on the analysis of Carvalhais et al. (2014). In summary, it provides an overview on:

- Comparisons of global distributions of tau_ctotal from all models against observation and other models
- Variation of tau_ctotal across latitude (zonal distributions)
- Variation of association of tau_ctotal and climate across latitude (zonal correlations)
- metrics of global tau_ctotal and correlations

12.2.2 Calculation of turnover time

First, the total carbon content of land ecosystems is calculated as,

$$ctotal = cSoil + cVeg$$

where $cSoil$ and $cVeg$ are the carbon contents in soil and vegetation. **Note that this is not fully consistent with `Carvalhais et al. (2014)`_, in which `ctotal` includes all carbon storages that respire to the atmosphere. Due to inconsistency across models, it resulted in having different carbon storage components in calculation of ctotal for different models**.

The turnover time of carbon is then calculated as,

$$\tau_{ctotal} = \frac{ctotal}{gpp}$$

where *ctotal* and *gpp* are temporal means of total carbon content and gross primary productivity, respectively. **The equation is valid for steady state, and is only applicable when both ctotal and gpp are long-term averages.** Therefore, the recipe should always include the mean operator of climate_statistics in preprocessor.

12.2.3 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_carvalhais14nat.yml

Diagnostics are stored in diag_scripts/

- land_carbon_cycle/diag_global_turnover.py
- land_carbon_cycle/diag_zonal_turnover.py
- land_carbon_cycle/diag_zonal_correlation.py

12.2.4 User settings in recipe

Observation-related details

The settings needed for loading the observational dataset in all diagnostics are provided in the recipe through *obs_info* within *obs_details* section.

- `obs_data_subdir`: subdirectory of auxiliary_data_dir (set in config-user file) where observation data are stored {e.g., data_ESMValTool_Carvalhais2014}.

- `source_label`: source data label {'Carvalhais2014'}.

- `variant_label`: variant of the observation {'BE'} for best estimate.

- `grid_label`: label denoting the spatial grid specification {'gn'}.

- `frequency`: temporal frequency of the observation data {'fx'}

The observation data file used in the recipe should be changed through the fields above, as these are used to generate observation file name and locations. For details, see *Observations* section.

Preprocessor

- `climate_statistics`: {mean} - calculate the mean over full time period.

- `regrid`: {nearest} - nearest neighbor regridding to the selected observation resolution.

- `mask_landsea`: {sea} - mask out all the data points from sea.

- `multi_model_statistics`: {median} - calculate and include the multimodel median.

Script land_carbon_cycle/diag_global_turnover.py

- Required settings:
  - `obs_variable`: {str} list of the variable(s) to be read from the observation files
- Optional settings:
  - `ax_fs`: {float, 7.1} - fontsize in the figure.
  - `fill_value`: {float, nan} - fill value to be used in analysis and plotting.
  - `x0`: {float, 0.02} - X - coordinate of the left edge of the figure.
  - `y0`: {float, 1.0} Y - coordinate of the upper edge of the figure.
  - `wp`: {float, 1 / number of models} - width of each map.
  - `hp`: {float, = wp} - height of each map.
  - `xsp`: {float, 0} - spacing betweeen maps in X - direction.
  - `ysp`: {float, -0.03} - spacing between maps in Y -direction. Negative to reduce the spacing below default.
  - `aspect_map`: {float, 0.5} - aspect of the maps.
  - `xsp_sca`: {float, wp / 1.5} - spacing between the scatter plots in X - direction.
  - `ysp_sca`: {float, hp / 1.5} - spacing between the scatter plots in Y - direction.
  - `hcolo`: {float, 0.0123} - height (thickness for horizontal orientation) of the colorbar .

- `wcolo`: {`float`, 0.25} - width (length) of the colorbar.

- `cb_off_y`: {`float`, 0.06158} - distance of colorbar from top of the maps.

- `x_colo_d`: {`float`, 0.02} - X - coordinate of the colorbar for maps along the diagonal (left).

- `x_colo_r`: {`float`, 0.76} - Y - coordinate of the colorbar for ratio maps above the diagonal (right).

- `y_colo_single`: {`float`, 0.1086} - Y-coordinate of the colorbar in the maps per model (separate figures).

- `correlation_method`: {`str`, spearman | pearson} - correlation method to be used while calculating the correlation displayed in the scatter plots.

- `tx_y_corr`: {`float`, 1.075} - Y - coordinate of the inset text of correlation.

- `valrange_sc`: {`tuple`, (2, 256)} - range of turnover times in X - and Y - axes of scatter plots.

- `obs_global`: {`float`, 23} - global turnover time, provided as additional info for map of the observation. For models, they are calculated within the diagnostic.

- `gpp_threshold`: {`float`, 0.01} - The threshold of gpp in *kg m^{-2} yr^{-1}* below which the grid cells are masked.

Script land_carbon_cycle/diag_zonal_turnover.py

- Required settings:
  - `obs_variable`: {`str`} list of the variable(s) to be read from the observation files
- Optional settings:
  - `ax_fs`: {`float`, 7.1} - fontsize in the figure.
  - `fill_value`: {`float`, nan} - fill value to be used in analysis and plotting.
  - `valrange_x`: {`tuple`, (2, 1000)} - range of turnover values in the X - axis.
  - `valrange_y`: {`tuple`, (-70, 90)} - range of latitudes in the Y - axis.
  - `bandsize`: {`float`, 9.5} - size of the latitudinal rolling window in degrees. One latitude row if set to `None`.
  - `gpp_threshold`: {`float`, 0.01} - The threshold of gpp in *kg m^{-2} yr^{-1}* below which the grid cells are masked.

Script land_carbon_cycle/diag_zonal_correlation.py

- Required settings:
  - `obs_variable`: {`str`} list of the variable(s) to be read from the observation files
- Optional settings:
  - `ax_fs`: {`float`, 7.1} - fontsize in the figure.
  - `fill_value`: {`float`, nan} - fill value to be used in analysis and plotting.
  - `correlation_method`: {`str`, pearson | spearman} - correlation method to be used while calculating the zonal correlation.
  - `min_points_frac`: {``float`, 0.125} - minimum fraction of valid points within the latitudinal band for calculation of correlation.
  - `valrange_x`: {`tuple`, (-1, 1)} - range of correlation values in the X - axis.

- – `valrange_y`: {tuple, (-70, 90)} - range of latitudes in the Y - axis.

- – `bandsize`: {float, 9.5} - size of the latitudinal rolling window in degrees. One latitude row if set to `None`.

- – `gpp_threshold`: {float, 0.01} - The threshold of gpp in *kg m^{-2} yr^{-1}* below which the grid cells are masked.

12.2.5 Required Variables

- *tas* (atmos, monthly, longitude, latitude, time)

- *pr* (atmos, monthly, longitude, latitude, time)

- *gpp* (land, monthly, longitude, latitude, time)

- *cVeg* (land, monthly, longitude, latitude, time)

- *cSoil* (land, monthly, longitude, latitude, time)

12.2.6 Observations

The observations needed in the diagnostics are publicly available for download from the Data Portal of the Max Planck Institute for Biogeochemistry after registration.

Due to inherent dependence of the diagnostic on uncertainty estimates in observation, the data needed for each diagnostic script are processed at different spatial resolutions (as in Carvalhais et al., 2014), and provided in 11 different resolutions (see Table 1). Note that the uncertainties were estimated at the resolution of the selected models, and, thus, only the pre-processed observed data can be used with the recipe. It is not possible to use regridding functionalities of ESMValTool to regrid the observational data to other spatial resolutions, as the uncertainty estimates cannot be regridded.

Table 1. A summary of the observation datasets at different resolutions.

Reference	target_grid	grid_label*
Observation	0.5x0.5	gn
NorESM1-M	2.5x1.875	gr
bcc-csm1-1	2.812x2.813	gr1
CCSM4	1.25x0.937	gr2
CanESM2	2.812x2.813	gr3
GFDL-ESM2G	2.5x2.0	gr4
HadGEM2-ES	1.875x1.241	gr5
inmcm4	2.0x1.5	gr6
IPSL-CM5A-MR	2.5x1.259	gr7
MIROC-ESM	2.812x2.813	gr8
MPI-ESM-LR	1.875x1.875	gr9

* The grid_label is suffixed with z for data in zonal/latitude coordinates: the zonal turnover and zonal correlation.

To change the spatial resolution of the evaluation, change {grid_label} in obs_details and the corresponding {target_grid} in regrid preprocessor of the recipe.

At each spatial resolution, four data files are provided:

- `tau_ctotal_fx_Carvalhais2014_BE_gn.nc` - global data of tau_ctotal

- `tau_ctotal_fx_Carvalhais2014_BE_gnz.nc` - zonal data of tau_ctotal

- `r_tau_ctotal_tas_fx_Carvalhais2014_BE_gnz.nc` - zonal correlation of tau_ctotal and tas, controlled for pr

- `r_tau_ctotal_pr_fx_Carvalhais2014_BE_gnz.nc` - zonal correlation of tau_ctotal and pr, controlled for tas.

The data is produced in obs4MIPs standards, and provided in netCDF4 format. The filenames use the convention:

`{variable}_{frequency}_{source_label}_{variant_label}_{grid_label}.nc`

- {variable}: variable name, set in every diagnostic script as obs_variable

- {frequency}: temporal frequency of data, set from obs_details

- {source_label}: observational source, set from obs_details

- {variant_label}: observation variant, set from obs_details

- {grid_label}: temporal frequency of data, set from obs_details

Refer to the Obs4MIPs Data Specifications for details of the definitions above.

All data variables have additional variables ({variable}_5 and {variable}_95) in the same file. These variables are necessary for a successful execution of the diagnostics.

12.2.7 References

- Carvalhais, N., et al. (2014), Global covariation of carbon turnover times with climate in terrestrial ecosystems, Nature, 514(7521), 213-217, doi: 10.1038/nature13731.
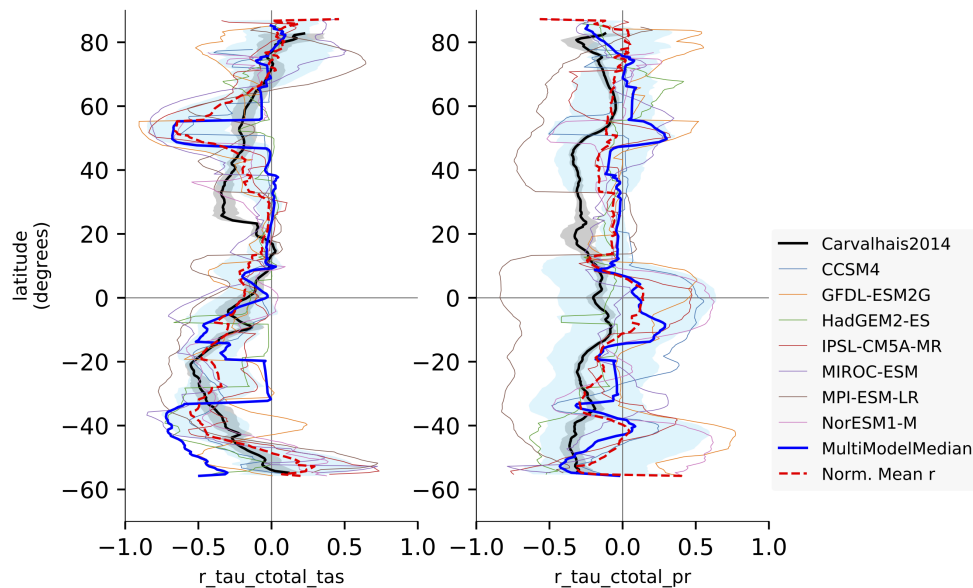
12.2.8 Example plots

Fig. 2: Comparison of latitudinal (zonal) variations of pearson correlation between turnover time and climate: turnover time and precipitation, controlled for temperature (left) and vice-versa (right). Reproduces figures 2c and 2d in Carvalhais et al. (2014).
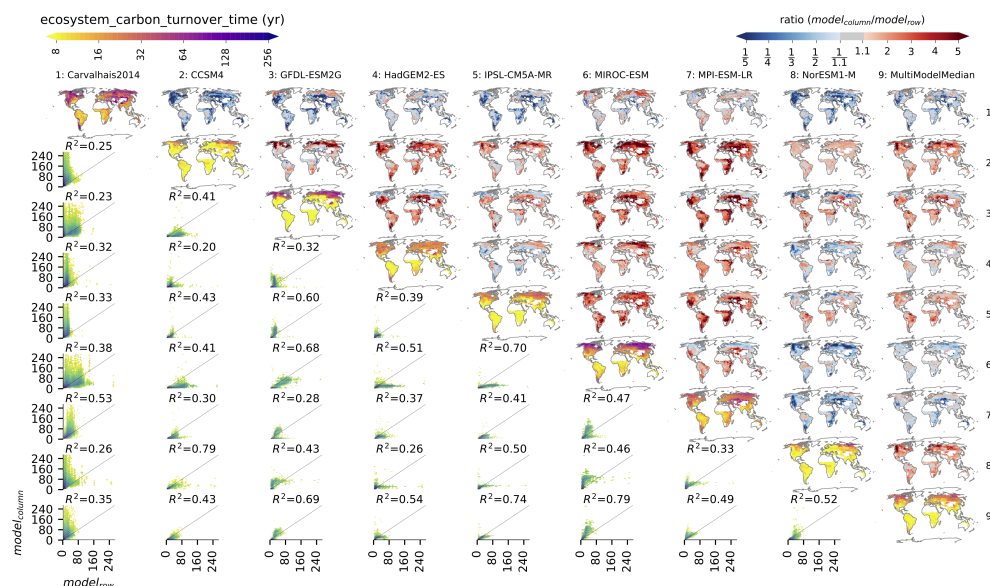
Fig. 3: Comparison of observation-based and modelled ecosystem carbon turnover time. Along the diagnonal, tau_ctotal are plotted, above the bias, and below density plots. The inset text in density plots indicate the correlation.
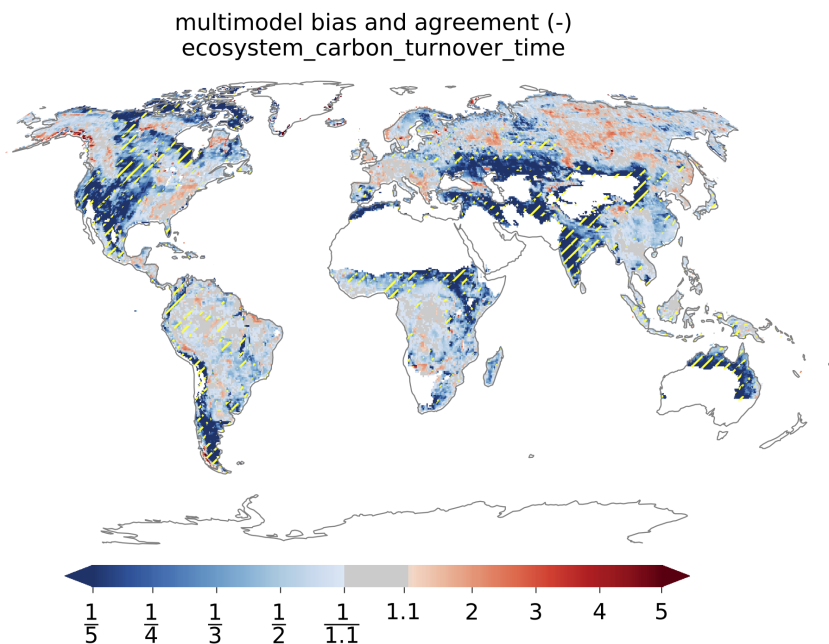

Fig. 4: Global distributions of multimodel bias and model agreement. Multimodel bias is calculated as the ratio of multimodel median turnover time and that from observation. Stippling indicates the regions where only less than one quarter of the models fall within the range of observational uncertainties (*5th* and *95th* percentiles). Reproduces figure 3 in Carvalhais et al. (2014).
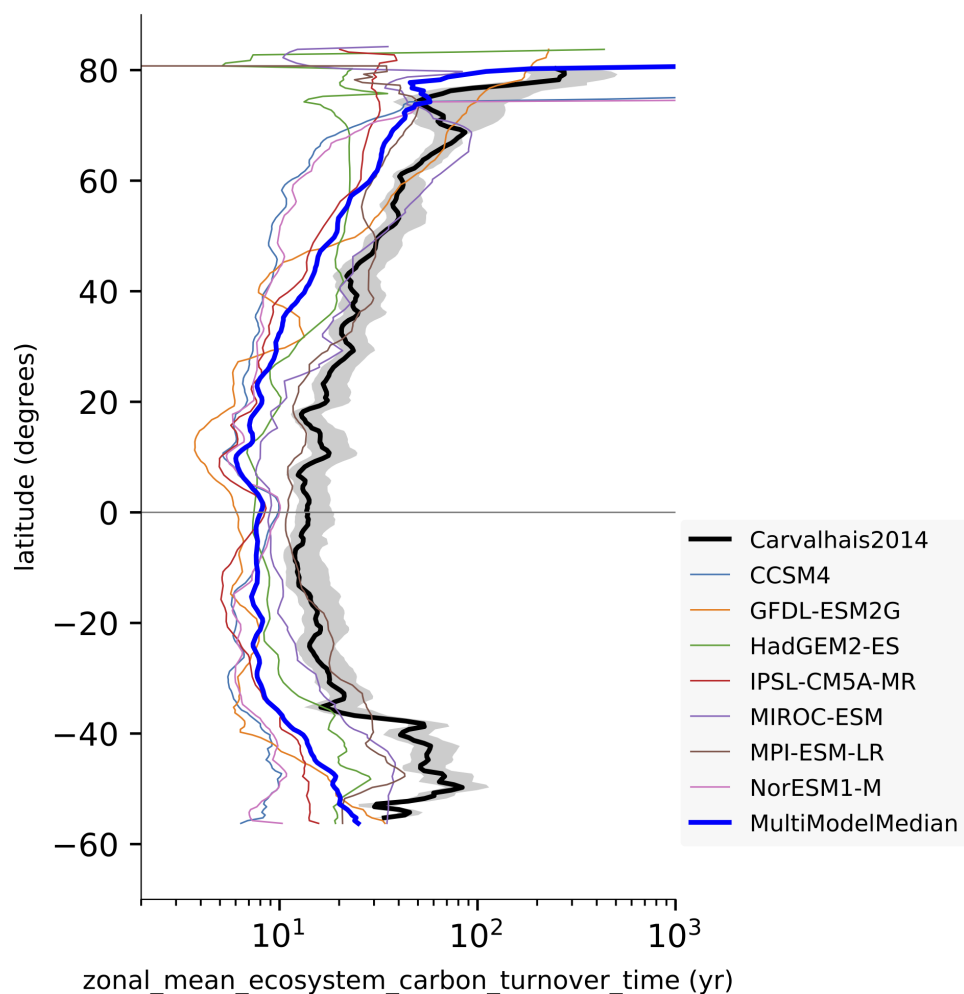
Fig. 5: Comparison of latitudinal (zonal) variations of observation-based and modelled ecosystem carbon turnover time. The zonal turnover time is calculated as the ratio of zonal *ctotal* and *gpp*. Reproduces figures 2a and 2b in Carvalhais et al. (2014).

12.3 Hydrological models - data pre-processing

12.3.1 Overview

We provide a collection of scripts that pre-processes environmental data for use in several hydrological models:

PCR-GLOBWB

PCR-GLOBWB (PCRaster Global Water Balance) is a large-scale hydrological model intended for global to regional studies and developed at the Department of Physical Geography, Utrecht University (Netherlands). The recipe pre-processes ERA-Interim reanalyses data for use in the PCR-GLOBWB.

MARRMoT

MARRMoT (Modular Assessment of Rainfall-Runoff Models Toolbox) is a rainfall-runoff model comparison framework that allows objective comparison between different conceptual hydrological model structures https://github.com/wknoben/MARRMoT. The recipe pre-processes ERA-Interim and ERA5 reanalyses data for use in the MARRMoT.

MARRMoT requires potential evapotranspiration (evspsblpot). The variable evspsblpot is not available in ERA-Interim. Thus, we use the debruin function (De Bruin et al. 2016) to obtain evspsblpot using both ERA-Interim and ERA5. This function needs the variables tas, psl, rsds, and rsdt as input.

wflow_sbm and wflow_topoflex

Forcing data for the wflow_sbm and wflow_topoflex hydrological models can be prepared using recipe_wflow.yml. If PET is not available from the source data (e.g. ERA-Interim), then it can be derived from psl, rsds and rsdt using De Bruin's 2016 formula (De Bruin et al. 2016). For daily ERA5 data, the time points of these variables are shifted 30 minutes with respect to one another. This is because in ERA5, accumulated variables are recorded over the past hour, and in the process of cmorization, we shift the time coordinates to the middle of the interval over which is accumulated. However, computing daily statistics then averages the times, which results in 12:00 UTC for accumulated variables and 11:30 UTC for instantaneous variables. Therefore, in this diagnostic, the time coordinates of the daily instantaneous variables are shifted 30 minutes forward in time.

LISFLOOD

LISFLOOD is a spatially distributed water resources model, developed by the Joint Research Centre (JRC) of the European Commission since 1997. We provide a recipe to produce meteorological forcing data for the Python 3 version of LISFLOOD.

LISFLOOD has a separate preprocessor LISVAP that derives some additional variables. We don't replace LISVAP. Rather, we provide input files that can readily be passed to LISVAP and then to LISFLOOD.

HYPE

The hydrological catchment model HYPE simulates water flow and substances on their way from precipitation through soil, river and lakes to the river outlet. HYPE is developed at the Swedish Meteorological and Hydrological Institute. The recipe pre-processes ERA-Interim and ERA5 data for use in HYPE.

12.3.2 Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/hydrology

- recipe_pcrglobwb.yml

- recipe_marrmot.yml

- recipe_wflow.yml

- recipe_lisflood.yml

- recipe_hype.yml

Diagnostics are stored in esmvaltool/diag_scripts/hydrology

- pcrglobwb.py

- marrmot.py

- wflow.py

- lisflood.py

- hype.py

12.3.3 User settings in recipe

All hydrological recipes require a shapefile as an input to produce forcing data. This shapefile determines the shape of the basin for which the data will be cut out and processed. All recipes are tested with the shapefiles that are used for the eWaterCycle project. In principle any shapefile can be used, for example, the freely available basin shapefiles from the HydroSHEDS project.

1. recipe_pcrglobwb.yml

   Required preprocessor settings:

   - start_year: 1979

   - end_year: 1979

2. recipe_marrmot.yml

   There is one diagnostic `diagnostic_daily` for using daily data.

   Required preprocessor settings:

   The settings below should not be changed.

   extract_shape:

   - shapefile: Meuse.shp (MARRMoT is a hydrological Lumped model that needs catchment-aggregated forcing data. The catchment is provided as a shapefile, the path can be relative to `auxiliary_data_dir` as defined in config-user.yml.).

   - method: contains

   - crop: true

Required diagnostic script settings:

- basin: Name of the catchment

3. recipe_wflow.yml

Optional preprocessor settings:

- extract_region: the region specified here should match the catchment

Required diagnostic script settings:

- basin: name of the catchment

- dem_file: netcdf file containing a digital elevation model with elevation in meters and coordinates latitude and longitude.

- regrid: the regridding scheme for regridding to the digital elevation model. Choose `area_weighted` (slow) or `linear`.

4. recipe_lisflood.yml

Required preprocessor settings:

- extract_region: A region bounding box slightly larger than the shapefile. This is run prior to regridding, to save memory.

- extract_shape:*

    - shapefile: A shapefile that specifies the extents of the catchment.

    These settings should not be changed

    - method: contains

    - crop: true

- regrid:*

    - target_grid: Grid of LISFLOOD input files

    These settings should not be changed

    - lon_offset: true

    - lat_offset: true

    - scheme: linear

There is one diagnostic `diagnostic_daily` for using daily data.

Required diagnostic script settings:

- catchment: Name of the catchment, used in output filenames

5. recipe_hype.yml

Required preprocessor settings:

- start_year: 1979

- end_year: 1979

- shapefile: Meuse_HYPE.shp (expects shapefile with subcatchments)

These settings should not be changed

- method: contains

- decomposed: true

12.3.4 Variables

1. recipe_pcrglobwb.yml

   - tas (atmos, daily, longitude, latitude, time)

   - pr (atmos, daily, longitude, latitude, time)

2. recipe_marrmot.yml

   - pr (atmos, daily or hourly mean, longitude, latitude, time)

   - psl (atmos, daily or hourly mean, longitude, latitude, time)

   - rsds (atmos, daily or hourly mean, longitude, latitude, time)

   - rsdt (atmos, daily or hourly mean, longitude, latitude, time)

   - tas (atmos, daily or hourly mean, longitude, latitude, time)

3. recipe_wflow.yml

   - orog (fx, longitude, latitude)

   - pr (atmos, daily or hourly mean, longitude, latitude, time)

   - tas (atmos, daily or hourly mean, longitude, latitude, time)

   Either potential evapotranspiration can be provided:

   - evspsblpot(atmos, daily or hourly mean, longitude, latitude, time)

   or it can be derived from tas, psl, rsds, and rsdt using the De Bruin formula, in that case the following variables need to be provided:

   - psl (atmos, daily or hourly mean, longitude, latitude, time)

   - rsds (atmos, daily or hourly mean, longitude, latitude, time)

   - rsdt (atmos, daily or hourly mean, longitude, latitude, time)

4. recipe_lisflood.yml

   - pr (atmos, daily, longitude, latitude, time)

   - tas (atmos, daily, longitude, latitude, time)

   - tasmax (atmos, daily, longitude, latitude, time)

   - tasmin (atmos, daily, longitude, latitude, time)

   - tdps (atmos, daily, longitude, latitude, time)

   - uas (atmos, daily, longitude, latitude, time)

   - vas (atmos, daily, longitude, latitude, time)

   - rsds (atmos, daily, longitude, latitude, time)

5. recipe_hype.yml

   - tas (atmos, daily or hourly, longitude, latitude, time)

   - tasmin (atmos, daily or hourly, longitude, latitude, time)

   - tasmax (atmos, daily or hourly, longitude, latitude, time)

   - pr (atmos, daily or hourly, longitude, latitude, time)

12.3.5 Observations and reformat scripts

Note: see headers of cmorization scripts (in esmvaltool/cmorizers/obs) for download instructions.

- ERA-Interim (esmvaltool/cmorizers/obs/cmorize_obs_era_interim.py)
- ERA5 (esmvaltool/cmorizers/obs/cmorize_obs_era5.py)

12.3.6 Output

1. recipe_pcrglobwb.yml

2. recipe_marrmot.yml

   The forcing data, the start and end times of the forcing data, the latitude and longitude of the catchment are saved in a .mat file as a data structure readable by MATLAB or Octave.

3. recipe_wflow.yml

   The forcing data, stored in a single NetCDF file.

4. recipe_lisflood.yml

   The forcing data, stored in separate files per variable.

12.3.7 References

- Sutanudjaja, E. H., van Beek, R., Wanders, N., Wada, Y., Bosmans, J. H. C., Drost, N., van der Ent, R. J., de Graaf, I. E. M., Hoch, J. M., de Jong, K., Karssenberg, D., López López, P., Peßenteiner, S., Schmitz, O., Straatsma, M. W., Vannametee, E., Wisser, D., and Bierkens, M. F. P.: PCR-GLOBWB 2: a 5arcmin global hydrological and water resources model, Geosci. Model Dev., 11, 2429-2453, https://doi.org/10.5194/gmd-11-2429-2018, 2018.

- De Bruin, H. A. R., Trigo, I. F., Bosveld, F. C., Meirink, J. F.: A Thermodynamically Based Model for Actual Evapotranspiration of an Extensive Grass Field Close to FAO Reference, Suitable for Remote Sensing Application, American Meteorological Society, 17, 1373-1382, DOI: 10.1175/JHM-D-15-0006.1, 2016.

- Arheimer, B., Lindström, G., Pers, C., Rosberg, J. och J. Strömqvist, 2008. Development and test of a new Swedish water quality model for small-scale and large-scale applications. XXV Nordic Hydrological Conference, Reykjavik, August 11-13, 2008. NHP Report No. 50, pp. 483-492.

- Lindström, G., Pers, C.P., Rosberg, R., Strömqvist, J., Arheimer, B. 2010. Development and test of the HYPE (Hydrological Predictions for the Environment) model – A water quality model for different spatial scales. Hydrology Research 41.3-4:295-319.

- van der Knijff, J. M., Younis, J. and de Roo, A. P. J.: LISFLOOD: A GIS-based distributed model for river basin scale water balance and flood simulation, Int. J. Geogr. Inf. Sci., 24(2), 189–212, 2010.

12.4 Landcover diagnostics

12.4.1 Overview

The diagnostic computes the accumulated and fractional extent of major land cover classes, namely bare soil, crops, grasses, shrubs and trees. The numbers are compiled for the whole land surface as well as separated into Tropics, northern Extratropics and southern Extratropics. The cover fractions are compared to ESA-CCI land cover data.

12.4.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_landcover.yml

Diagnostics are stored in diag_scripts/landcover/

- landcover.py: bar plots showing the accumulated area and mean fractional coverage for five land cover classes for all experiments as well as their bias compared to observations.

12.4.3 User settings

script landcover.py

Required settings for script

- reference_dataset: land cover extent dataset for comparison. The script was developed using ESACCI-LANDCOVER observations.

Optional settings for script

- comparison: [variable, model] Choose whether one plot per land cover class is generated comparing the different experiments (default) or one plot per model comparing the different land cover classes.

- colorscheme: Plotstyle used for the bar plots. A list of available style is found at https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html. Seaborn is used as default.

12.4.4 Variables

- baresoilFrac (land, monthly mean, time latitude longitude)

- grassFrac (land, monthly mean, time latitude longitude)

- treeFrac (land, monthly mean, time latitude longitude)

- shrubFrac (land, monthly mean, time latitude longitude)

- cropFrac (land, monthly mean, time latitude longitude)

12.4.5 Observations and reformat scripts

ESA-CCI land cover data (Defourny et al., 2015) needs to be downloaded manually by the user and converted to netCDF files containing the grid cell fractions for the five major land cover types. The data and a conversion tool are available at https://maps.elie.ucl.ac.be/CCI/viewer/ upon registration. After obtaining the data and the user tool, the remapping to 0.5 degree can be done with:

```
./bin/aggregate-map.sh
-PgridName=GEOGRAPHIC_LAT_LON
-PnumRows=360
-PoutputLCCSClasses=true
-PnumMajorityClasses=0
ESACCI-LC-L4-LCCS-Map-300m-P1Y-2015-v2.0.7b.nc
```

Next, the data needs to be aggregated into the five major classes (PFT) similar to the study of Georgievski & Hagemann (2018) and converted from grid cell fraction into percentage.

PFT	ESA-CCI Landcover Classes
baresoil-Frac	Bare_Soil
cropFrac	Managed_Grass
grassFrac	Natural_Grass
shrubFrac	Shrub_Broadleaf_Deciduous + Shrub_Broadleaf_Evergreen + Shrub_Needleleaf_Evergreen
treeFrac	Tree_Broadleaf_Deciduous + Tree_Broadleaf_Evergreen + Tree_Needleleaf_Deciduous + Tree_Needleleaf_Evergreen

Finally, it might be necessary to adapt the grid structure to the experiments files, e.g converting the -180 –> 180 degree grid to 0 –> 360 degree and inverting the order of latitudes. Note, that all experiments will be regridded onto the grid of the land cover observations, thus it is recommended to convert to the coarses resolution which is sufficient for the planned study. For the script development, ESA-CCI data on 0.5 degree resolution was used with land cover data averaged over the 2008-2012 period.

12.4.6 References

- Defourny et al. (2015): ESA Land Cover Climate Change Initiative (ESA LC_cci) data: ESACCI-LC-L4-LCCS-Map-300m-P5Y-[2000,2005,2010]-v1.6.1 via Centre for Environmental Data Analysis
- Georgievski, G. & Hagemann, S. Characterizing uncertainties in the ESA-CCI land cover map of the epoch 2010 and their impacts on MPI-ESM climate simulations, Theor Appl Climatol (2018). https://doi.org/10.1007/s00704-018-2675-2

12.4.7 Example plots

12.5 Land and ocean components of the global carbon cycle

12.5.1 Overview

This recipe reproduces most of the figures of Anav et al. (2013):

- Timeseries plot for different regions
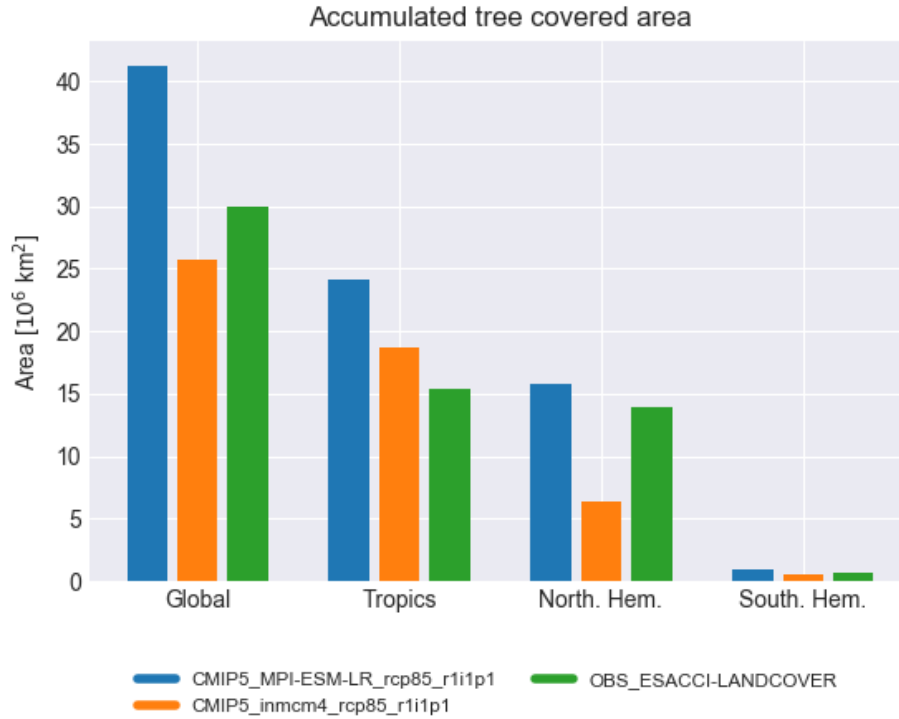- Seasonal cycle plot for different regions

Accumulated tree covered area

Fig. 6: Accumulated tree covered area for different regions and experiments.

- Errorbar plot for different regions showing mean and standard deviation

- Scatterplot for different regions showing mean vs. interannual variability

- 3D-scatterplot for different regions showing mean vs. linear trend and the model variability index (MVI) as a third dimension (color coded)

- Scatterplot for different regions comparing two variable against each other (*cSoil* vs. *cVeg*)

In addition, performance metrics are calculated for all variables using the performance metric diagnostics (see details in *Performance metrics for essential climate parameters*).

12.5.2 MVI calculation

The Model variability index (MVI) on a single grid point (calculated in `carbon_cycle/mvi.ncl` is defined as

$$MVI = \left(\frac{s^M}{s^O} - \frac{s^O}{s^M} \right)^2$$

where s^M and s^O are the standard deviations of the annual time series on a single grid point of a climate model M and the reference observation O. In order to get a global or regional result, this index is simple averaged over the respective domain.

In its given form, this equation is prone to small standard deviations close to zero. For example, values of $s^M = 10^{-5}\mu$ and $s^O = 10^{-7}\mu$ (where μ is the mean of s^O over all grid cells) results in a MVI of the order of 10^4 for this single grid cell even though the two standard deviations are close to zero and negligible compared to other grid cells. Due to the use of the arithmetic mean, a single high value is able to distort the overall MVI.
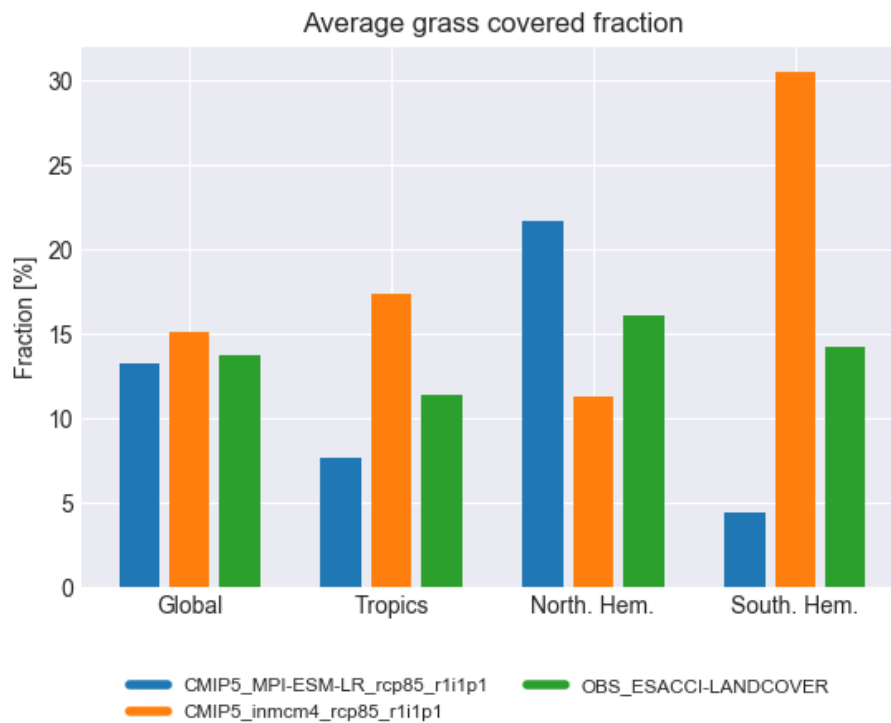
Fig. 7: Average grass cover fraction for different regions and experiments

In the original publication, the maximum MVI is in the order of 10 (for the variable *gpp*). However, a naive application of the MVI definition yields values over 10^9 for some models. Unfortunately, Anav et al. (2013) do not provide an explanation on how to deal with this problem. Nevertheless, this script provides two configuration options to avoid high MVI values, but they are not related to the original paper or any other peer-revied study and should be used with great caution (see *User settings in recipe*).

12.5.3 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_anav13jclim.yml

Diagnostics are stored in diag_scripts/

- carbon_cycle/main.ncl
- carbon_cycle/mvi.ncl
- carbon_cycle/two_variables.ncl
- perfmetrics/main.ncl
- perfmetrics/collect.ncl
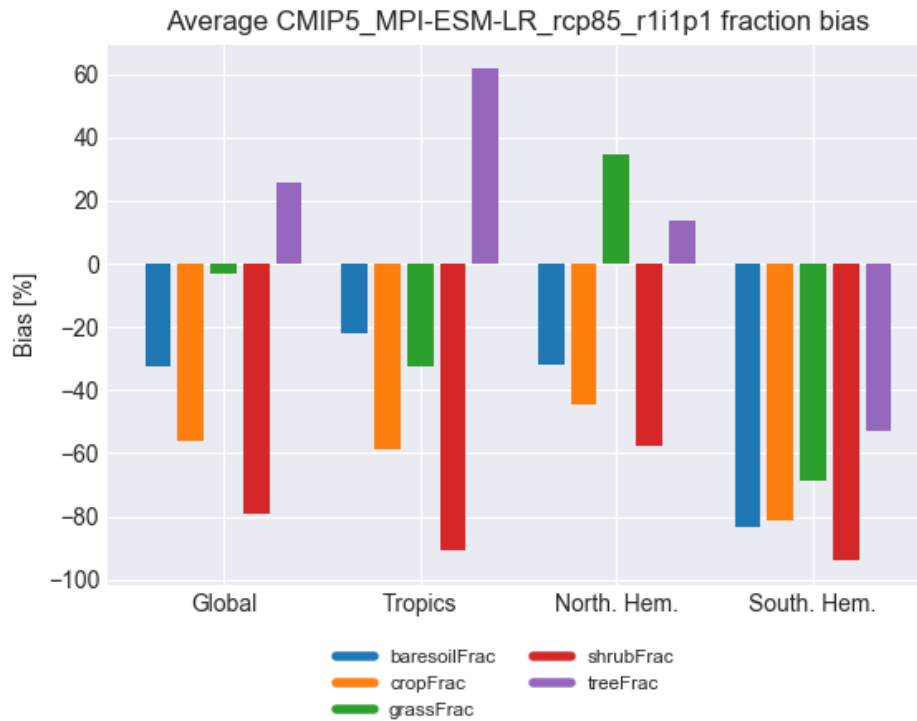
Fig. 8: Biases in five major land cover fractions for different regions and one experiment.

12.5.4 User settings in recipe

1. Preprocessor

    - `mask_fillvalues`: Mask common missing values on different datasets.

    - `mask_landsea`: Mask land/ocean.

    - `regrid`: Regridding.

    - `weighting_landsea_fraction`: Land/ocean fraction weighting.

2. Script carbon_cycle/main.ncl

    - `region`, *str*: Region to be averaged.

    - `legend_outside`, *bool*: Plot legend in a separate file (does not affect errorbar plot and evolution plot)

    - `seasonal_cycle_plot`, *bool*: Draw seasonal cycle plot.

    - `errorbar_plot`, *bool*: Draw errorbar plot.

    - `mean_IAV_plot`, *bool*: Draw Mean (x-axis), IAV (y-axis) plot.

    - `evolution_plot`, *bool*: Draw time evolution of a variable comparing a reference dataset to multi-dataset mean; requires ref_dataset in recipe.

    - `sort`, *bool*, optional (default: `False`): Sort dataset in alphabetical order.

    - `anav_month`, *bool*, optional (default: `False`): Conversion of y-axis to PgC/month instead of /year.

- evolution_plot_ref_dataset, *str*, optional: Reference dataset for evolution_plot. Required when evolution_plot is True.

- evolution_plot_anomaly, *str*, optional (default: False): Plot anomalies in evolution plot.

- evolution_plot_ignore, *list*, optional: Datasets to ignore in evolution plot.

- evolution_plot_volcanoes, *bool*, optional (default: False): Turns on/off lines of volcano eruptions in evolution plot.

- evolution_plot_color, *int*, optional (default: 0): Hue of the contours in the evolution plot.

- ensemble_name, *string*, optional: Name of ensemble for use in evolution plot legend

3. Script carbon_cycle/mvi.ncl

   - region, *str*: Region to be averaged.

   - reference_dataset, *str*: Reference dataset for the MVI calculation specified for each variable seperately.

   - mean_time_range, *list*, optional: Time period over which the mean is calculated (if not given, use whole time span).

   - trend_time_range, *list*, optional: Time period over which the trend is calculated (if not given, use whole time span).

   - mvi_time_range, *list*, optional: Time period over which the MVI is calculated (if not given, use whole time span).

   - stddev_threshold, *float*, optional (default: 1e-2): Threshold to ignore low standard deviations (relative to the mean) in the MVI calculations. See also *MVI calculation*.

   - mask_below, *float*, optional: Threshold to mask low absolute values (relative to the mean) in the input data (not used by default). See also *MVI calculation*.

4. Script carbon_cycle/two_variables.ncl

   - region, *str*: Region to be averaged.

5. Script perfmetrics/main.ncl

   See *Performance metrics for essential climate parameters*.

6. Script perfmetrics/collect.ncl

   See *Performance metrics for essential climate parameters*.

12.5.5 Variables

- *tas* (atmos, monthly, longitude, latitude, time)
- *pr* (atmos, monthly, longitude, latitude, time)
- *nbp* (land, monthly, longitude, latitude, time)
- *gpp* (land, monthly, longitude, latitude, time)
- *lai* (land, monthly, longitude, latitude, time)
- *cveg* (land, monthly, longitude, latitude, time)
- *csoil* (land, monthly, longitude, latitude, time)
- *tos* (ocean, monthly, longitude, latitude, time)
- *fgco2* (ocean, monthly, longitude, latitude, time)

12.5.6 Observations and reformat scripts

- CRU (*tas*, *pr*)
- JMA-TRANSCOM (*nbp*, *fgco2*)
- MTE (*gpp*)
- LAI3g (*lai*)
- NDP (*cveg*)
- HWSD (*csoil*)
- HadISST (*tos*)

12.5.7 References

- Anav, A. et al.: Evaluating the land and ocean components of the global carbon cycle in the CMIP5 Earth System Models, J. Climate, 26, 6901-6843, doi: 10.1175/JCLI-D-12-00417.1, 2013.

12.5.8 Example plots

12.6 Runoff, Precipitation, Evapotranspiration

12.6.1 Overview

This diagnostic calculates biases of long-term climatological annual means of total runoff R, precipitation P and evapotranspiration E for 12 large-scale catchments on different continents and climates. For total runoff, catchment averaged model values are compared to climatological GRDC station observations of river runoff (Duemenil Gates et al., 2000). Due to the incompleteness of these station data, a year-to-year correspondence of data cannot be achieved in a generalized way, so that only climatological data are considered, such it has been done in Hagemann, et al. (2013). For precipitation, catchment-averaged WFDEI precipitation data (Weedon et al., 2014) from 1979-2010 is used as reference. For evapotranspiration, observations are estimated using the difference of the above mentioned precipitation reference minus the climatological GRDC river runoff.

The catchments are Amazon, Congo, Danube, Ganges-Brahmaputra, Lena, Mackenzie, Mississippi, Murray, Niger, Nile, Parana and Yangtze-Kiang. Variable names are expected to follow CMOR standard, e.g. precipitation as pr, total runoff as mrro and evapotranspiration as evspsbl with all fluxes given in kg m-2 s-1 . Evapotranspiration furthermore has to be defined positive upwards.

The diagnostic produces text files with absolute and relative bias to the observations, as well as the respective absolute values. Furthermore it creates a bar plot for relative and absolute bias, calculates and plots biases in runoff coefficient (R/P) and evapotranspiration coefficient (E/P) and saves everything as one pdf file per model or one png file per model and analysis.

The bias of the runoff coefficient is calculated via: $C_R = \frac{R_{model}}{P_{model}} - \frac{R_{GRDC}}{P_{WFDEI}}$ and similar for the evapotranspiration coefficient. In a very first approximation, evapotranspiration and runoff are determined only by precipitation. In other words $R = P - E$. Hence, the runoff coefficient (and similar the evapotranspiration coefficient) tells you how important runoff (or evapotranspiration) is in this region. By plotting the bias of the runoff coefficient against the evapotranspiration coefficient we can immediately see whether there is a shift from runoff to evapotranspiration. On the other hand, by plotting the bias of the runoff coefficient against the relative bias of precipitation we can see whether an error in runoff is due to an error in precipitation.
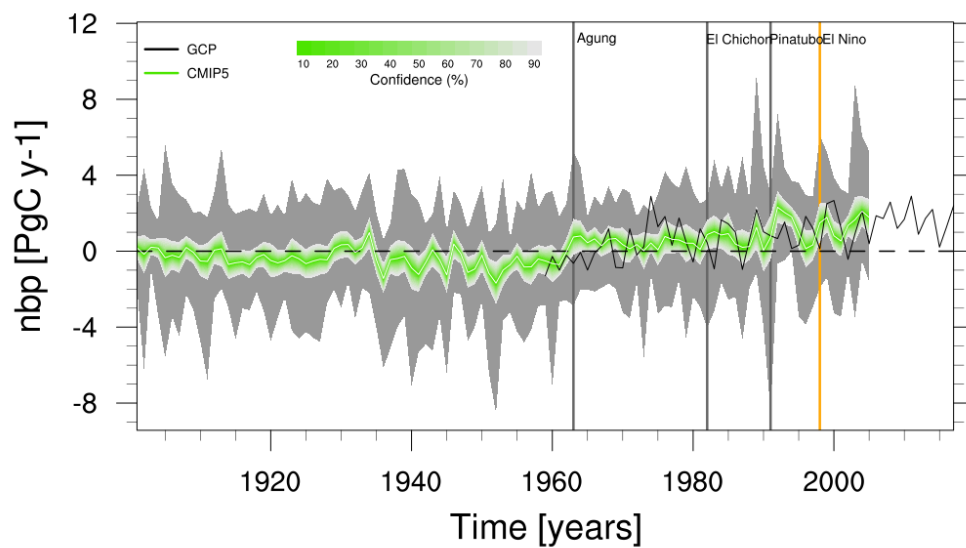
Fig. 9: Time series of global net biome productivity (NBP) over the period 1901-2005. Similar to Anav et al. (2013), Figure 5.
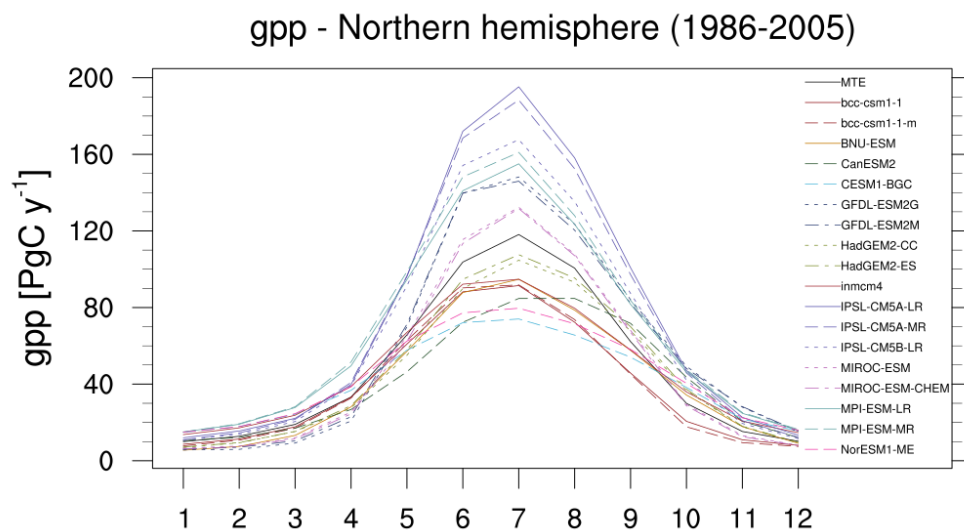
Fig. 10: Seasonal cycle plot for nothern hemisphere gross primary production (GPP) over the period 1986-2005. Similar to Anav et al. (2013), Figure 9.
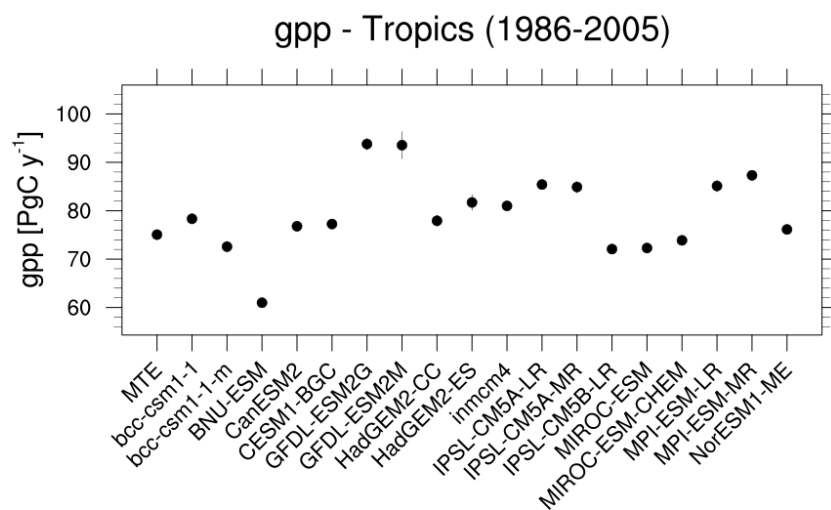
Fig. 11: Errorbar plot for tropical gross primary production (GPP) over the period 1986-2005.
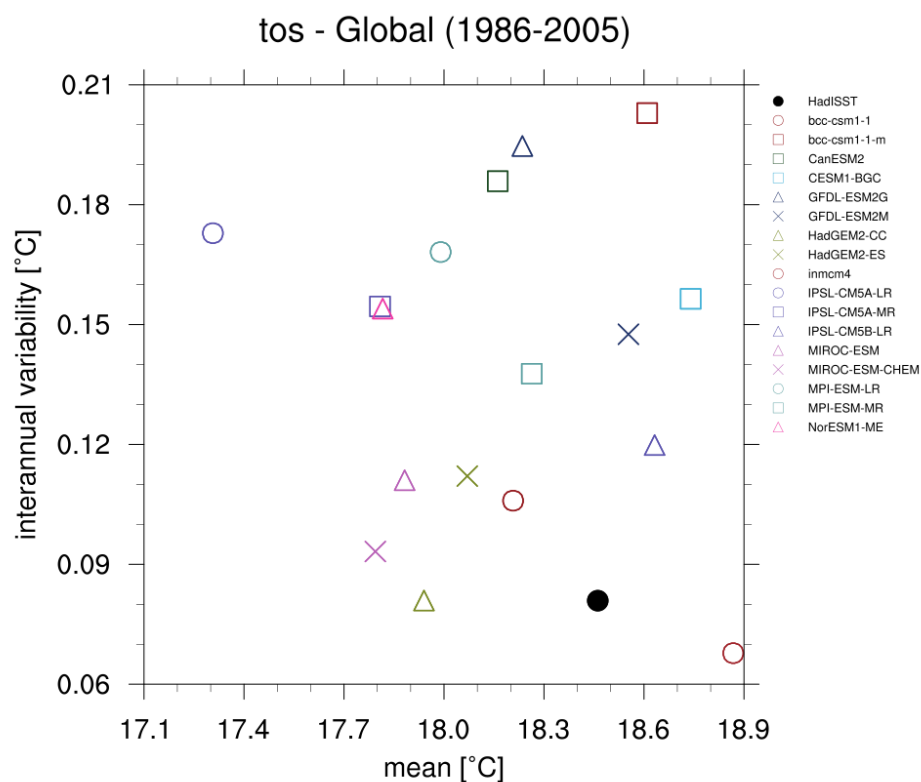
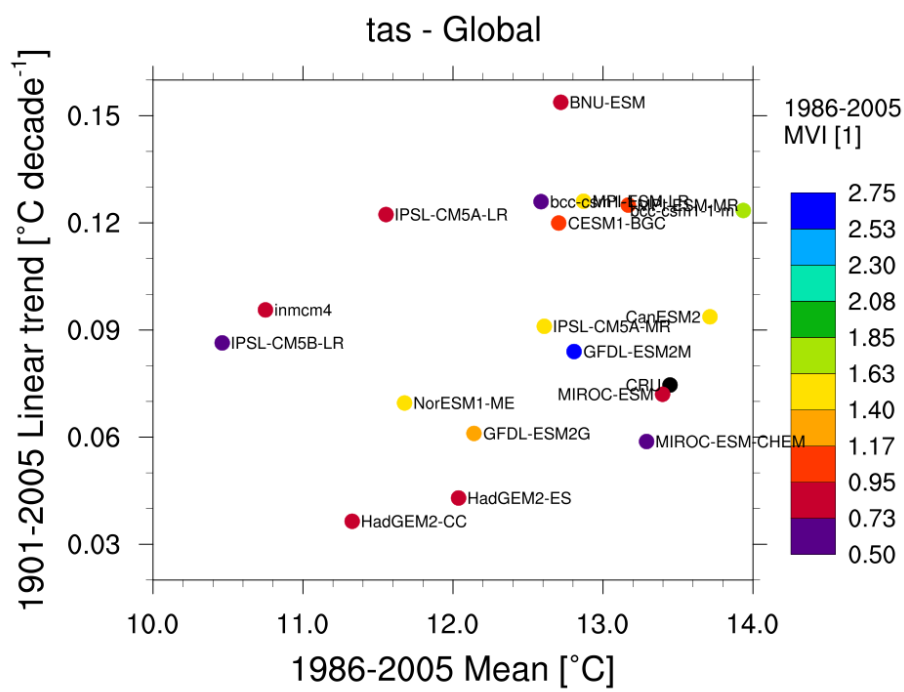Fig. 12: Scatterplot for interannual variability and mean of global sea surface temperature (TOS) over the period 1986-2005.

Fig. 13: Scatterplot for multiyear average of 2m surface temperature (TAS) in x axis, its linear trend in y axis, and MVI. Similar to Anav et al. (2013) Figure 1 (bottom).
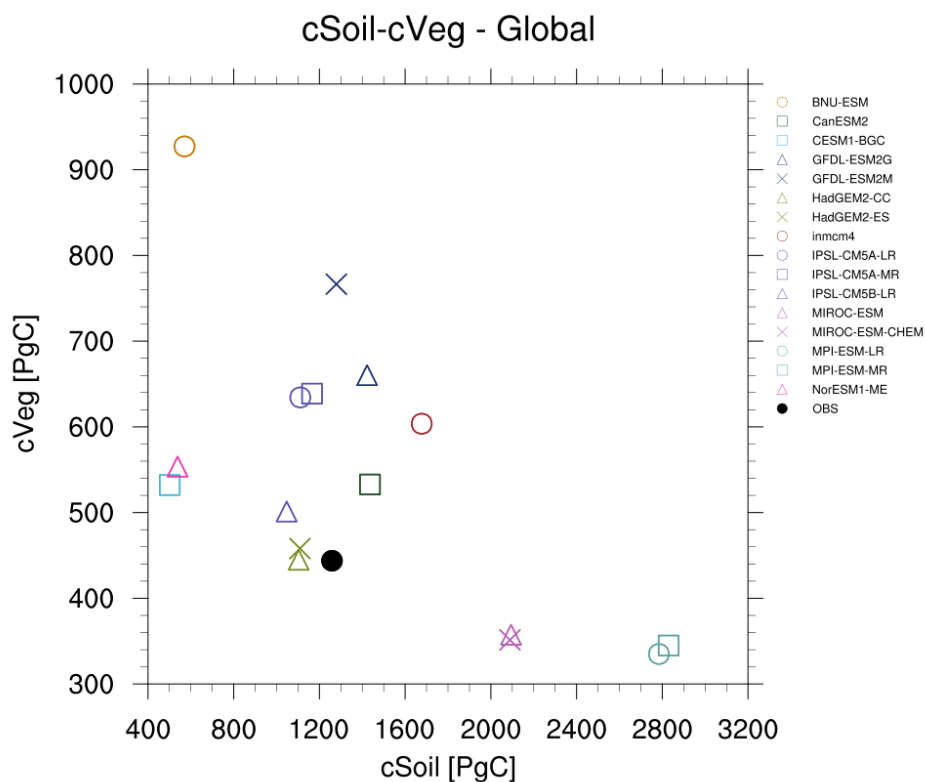
Fig. 14: Scatterplot for vegetation carbon content (cVeg) and soil carbon content (cSoil) over the period 1986-2005. Similar to Anav et al. (2013), Figure 12.
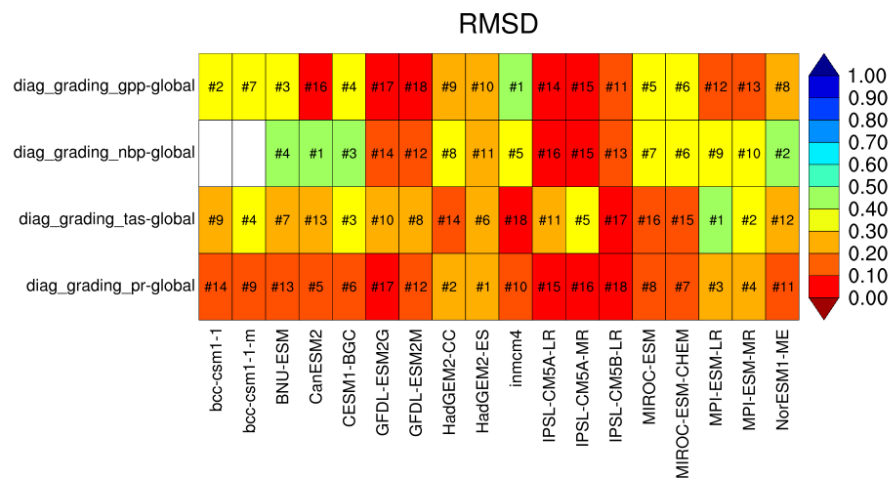
Fig. 15: Performance metrics plot for carbon-cycle-relevant diagnostics.

12.6.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_runoff_et.yml

Diagnostics are stored in diag_scripts/runoff_et/

- catchment_analysis.py: bar and scatter plots for catchment averages of runoff, evapotranspiration and precipitation

12.6.3 User settings in recipe

1. Script catchment_analysis.py

   Required settings (scripts)

   - catchmentmask: netCDF file indicating the grid cell for a specific catchment. Modus of distribution not yet clearified. ESGF?

   Optional settings (variables)

   - reference_dataset: dataset_name Datasets can be used as reference instead of defaults provided with the diagnostics. Must be identical for all variables.

12.6.4 Variables

- evspsbl (atmos, monthly mean, time latitude longitude)

- pr (atmos, monthly mean, time latitude longitude)

- mrro (land, monthly mean, time latitude longitude)

12.6.5 Observations and reformat scripts

Default reference data based on GRDC and WFDEI are included in the diagnostic script as catchment averages. They can be replaced with any gridded dataset by defining a reference_dataset. The necessary catchment mask is available at

All other datasets are remapped onto the catchment mask grid as part of the diagnostics.

12.6.6 References

- Duemenil Gates, L., S. Hagemann and C. Golz, Observed historical discharge data from major rivers for climate model validation. Max Planck Institute for Meteorology Report 307, Hamburg, Germany, 2000.

- Hagemann, S., A. Loew, A. Andersson, Combined evaluation of MPI-ESM land surface water and energy fluxes J. Adv. Model. Earth Syst., 5, doi:10.1029/2012MS000173, 2013.

- Weedon, G. P., G. Balsamo, N. Bellouin, S. Gomes, M. J. Best, and P. Viterbo, The WFDEI meteorological forcing data set: WATCH Forcing Data methodology applied to ERA-Interim reanalysis data, Water Resour. Res., 50, 7505–7514, doi: 10.1002/2014WR015638, 2014
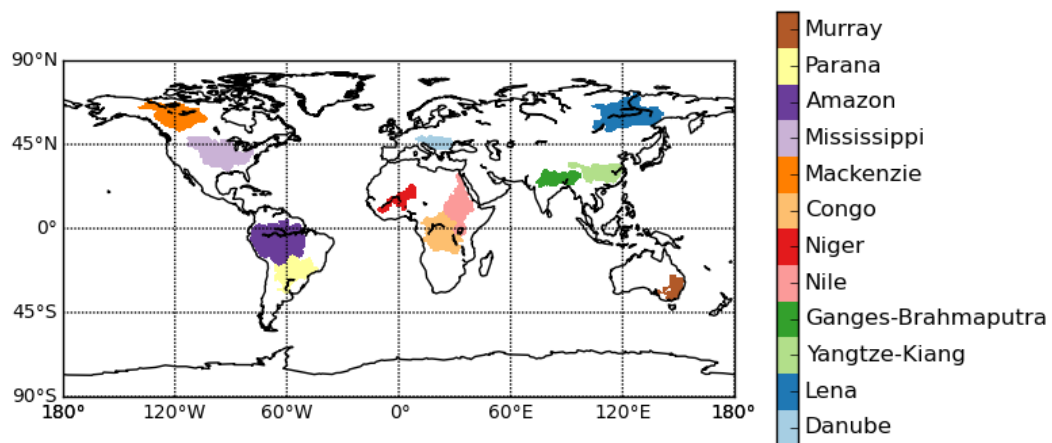
12.6.7 Example plots

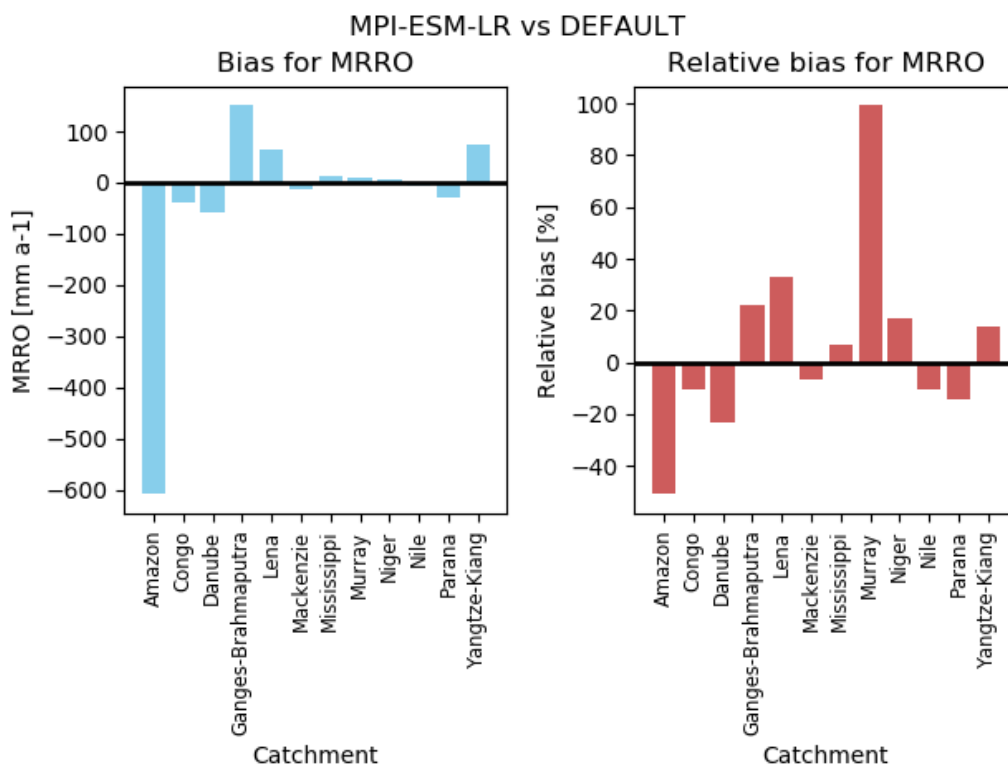Fig. 16: Catchment definitions used in the diagnostics.

Fig. 17: Barplot indicating the absolute and relative bias in annual runoff between MPI-ESM-LR (1970-2000) and long term GRDC data for specific catchments.
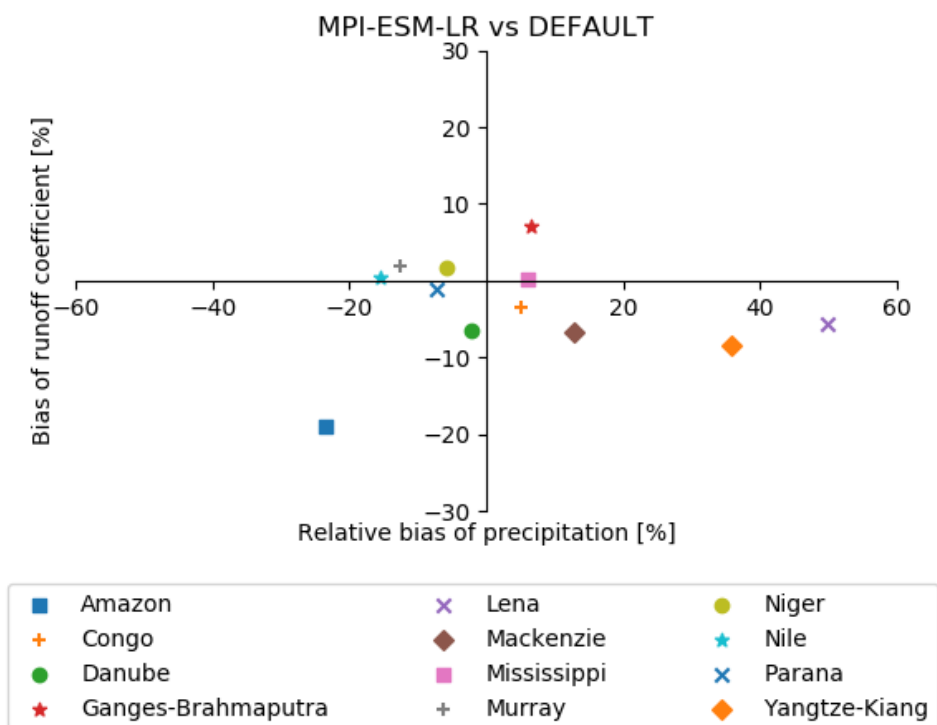
Fig. 18: Biases in runoff coefficient (runoff/precipitation) and precipitation for major catchments of the globe. The MPI-ESM-LR historical simulation (1970-2000) is used as an example.

OCEAN

13.1 Recipe for evaluating Arctic Ocean

13.1.1 Overview

The Arctic ocean is one of the areas of the Earth where the effects of climate change are especially visible today. Two most prominent processes are Arctic amplification [e.g. Serreze and Barry, 2011] and decrease of the sea ice area and thickness. Both receive good coverage in the literature and are already well-studied. Much less attention is paid to the interior of the Arctic Ocean itself. In order to increase our confidence in projections of the Arctic climate future proper representation of the Arctic Ocean hydrography is necessary.

The main focus of this diagnostics is evaluation of ocean components of climate models in the Arctic Ocean, however most of the diagnostics are implemented in a way that can be easily expanded to other parts of the World Ocean. Most of the diagnostics aim at model comparison to climatological data (PHC3), so we target historical CMIP simulations. However scenario runs also can be analysed to have an impression of how Arcti Ocean hydrography will chnage in the future.

At present only the subset of CMIP models can be used in particular because our analysis is limited to z coordinate models.

13.1.2 Available recipes

Recipe is stored in recipes/

- recipe_arctic_ocean.yml : contains all setting nessesary to run diagnostics and metrics.

Currenly the workflow do not allow to easily separate diagnostics from each other, since some of the diagnostics rely on the results of other diagnostics. The recipe currently do not use preprocessor options, so input files are CMORised monthly mean 3D ocean varibales on original grid.

The following plots will be produced by the recipe:

Hovmoeller diagrams

The characteristics of vertical TS distribution can change with time, and consequently the vertical TS distribution is an important indicator of the behaviour of the coupled ocean-sea ice-atmosphere system in the North Atlantic and Arctic Oceans. One way to evaluate these changes is by using Hovmoller diagrams. Hovmoller diagrams for two main Arctic Ocean basins – Eurasian and Amerasian with T and S spatially averaged on a monthly basis for every vertical level are available. This diagnostic allows the temporal evolution of vertical ocean potential temperature distribution to be assessed.

Related settings in the recipe:

```
# Define regions, as a list.
# 'EB' – Eurasian Basin of the Arctic Ocean
# 'AB' – Amerasian Basin of the Arctic Ocean
# 'Barents_sea' – Barrents Sea
# 'North_sea'   – North Sea
hofm_regions: ["AB" ,  'EB']
# Define variables to use, should also be in "variables"
# entry of your diagnostic
hofm_vars: ['thetao', 'so']
# Maximum depth of Hovmoeller and vertical profiles
hofm_depth: 1500
# Define if Hovmoeller diagrams will be ploted.
hofm_plot: True
# Define colormap (as a list, same size as list with variables)
# Only cmaps from matplotlib and cmocean are supported.
# Additional cmap – 'custom_salinity1'.
hofm_cmap: ['Spectral_r', 'custom_salinity1']
# Data limits for plots,
# List of the same size as the list of the variables
# each entry is [vmin, vmax, number of levels, rounding limit]
hofm_limits: [[-2, 2.3, 41, 1], [30.5, 35.1, 47, 2]]
# Number of columns in the plot
hofm_ncol: 3
```

Vertical profiles

The vertical structure of temperature and salinity (T and S) in the ocean model is a key diagnostic that is used for ocean model evaluation. Realistic T and S distributions means that model properly represent dynamic and thermodynamic processes in the ocean. Different ocean basins have different hydrological regimes so it is important to perform analysis of vertical TS distribution for different basins separately. The basic diagnostic in this sense is mean vertical profiles of temperature and salinity over some basin averaged for relatively long period of time. In addition to individual vertical profiles for every model, we also show the mean over all participating models and similar profile from climatological data (PHC3).

Several settings for vertical profiles (region, variables, maximum depths) will be determined by the Hovmoeller diagram settings. The reason is that vertical profiles are calculated from Hovmoeller diagram data. Mean vertical profile is calculated by linearly interpolating data on standard WOA/PHC depths.

Related settings in the recipe:

```
# Define regions, as a list.
# 'EB' – Eurasian Basin of the Arctic Ocean
# 'AB' – Amerasian Basin of the Arctic Ocean
# 'Barents_sea' – Barrents Sea
# 'North_sea'   – North Sea
```
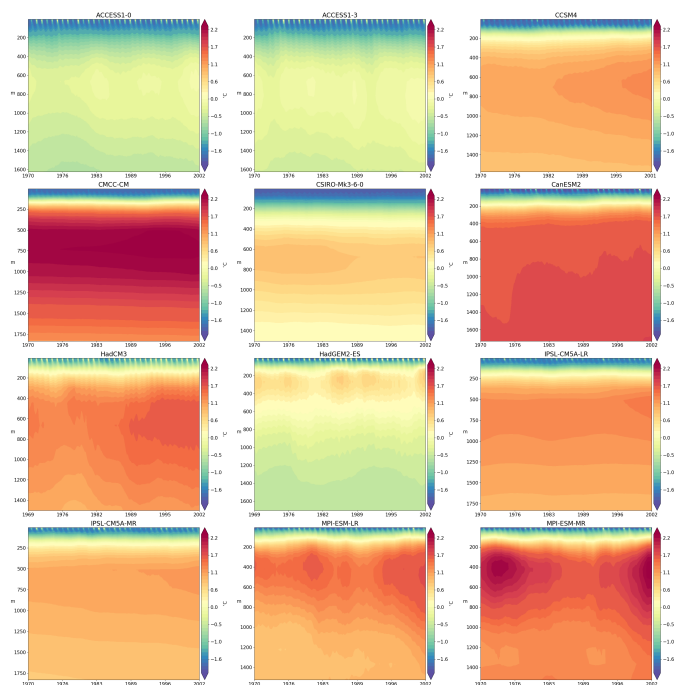
Fig. 1: Hovmoller diagram of monthly spatially averaged potential temperature in the Eurasian Basin of the Arctic Ocean for selected CMIP5 climate models (1970-2005).

```
hofm_regions: ["AB" ,  'EB']
# Define variables to use, should also be in "variables" entry of your
↪diagnostic
hofm_vars: ['thetao', 'so']
# Maximum depth of Hovmoeller and vertical profiles
hofm_depth: 1500
```

Spatial distribution maps of variables

The spatial distribution of basic oceanographic variables characterises the properties and spreading of ocean water masses. For the coupled models, capturing the spatial distribution of oceanographic variables is especially important in order to correctly represent the ocean-ice-atmosphere interface. We have implemented plots with spatial maps of temperature and salinity at original model levels.

Plots spatial distribution of variables at selected depths in North Polar projection on original model grid. For plotting the model depths that are closest to provided *plot2d_depths* will be selected. Settings allow to define color maps and limits for each variable individually. Color maps should be ehter part of standard matplotlib set or one of the cmocean color maps. Additional colormap *custom_salinity1* is provided.

Related settings in the recipe:

```
# Depths for spatial distribution maps
plot2d_depths: [10, 100]
# Variables to plot spatial distribution maps
plot2d_vars: ['thetao', 'so']
# Define colormap (as a list, same size as list with variables)
# Only cmaps from matplotlib and cmocean are supported.
```

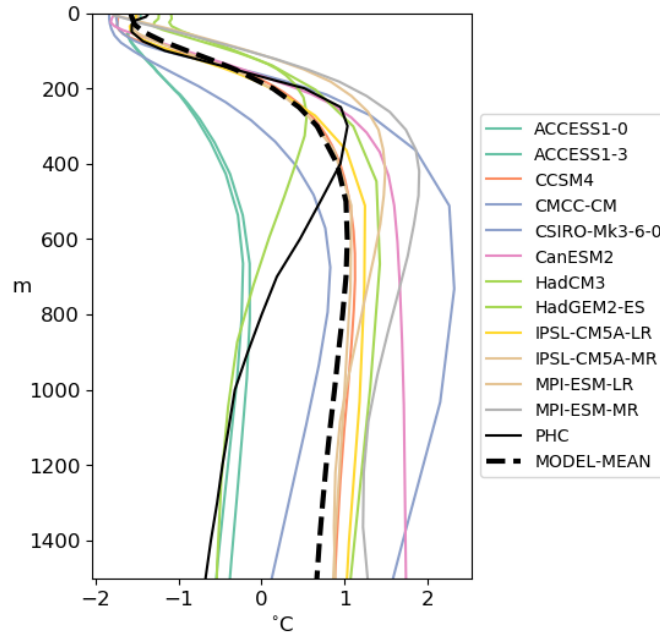**13.1. Recipe for evaluating Arctic Ocean** 213

Fig. 2: Mean (1970-2005) vertical potential temperature distribution in the Eurasian basin for participating CMIP5 coupled ocean models, PHC3 climatology (dotted red line) and multi-model mean (dotted black line).

```
# Additional cmap - 'custom_salinity1'.
plot2d_cmap: ['Spectral_r', 'custom_salinity1']
# Data limits for plots,
# List of the same size as the list of the variables
# each entry is [vmin, vmax, number of levels, rounding limit]
plot2d_limits: [[-2, 4, 20, 1], [30.5, 35.1, 47, 2]]
# number of columns for plots
plot2d_ncol: 3
```

Spatial distribution maps of biases

For temperature and salinity, we have implemented spatial maps of model biases from the observed climatology. For the model biases, values from the original model levels are linearly interpolated to the climatology and then spatially interpolated from the model grid to the regular PHC (climatology) grid. Resulting fields show model performance in simulating spatial distribution of temperature and salinity.

Related settings in the recipe:

```
plot2d_bias_depths: [10, 100]
# Variables to plot spatial distribution of the bias for.
plot2d_bias_vars: ['thetao', 'so']
# Color map names for every variable
plot2d_bias_cmap: ['balance', 'balance']
# Data limits for plots,
# List of the same size as the list of the variables
# each entry is [vmin, vmax, number of levels, rounding limit]
plot2d_bias_limits: [[-3, 3, 20, 1], [-2, 2, 47, 2]]
```

Fig. 3: Mean (1970-2005) salinity distribution at 100 meters.

```
# number of columns in the bias plots
plot2d_bias_ncol: 3
```
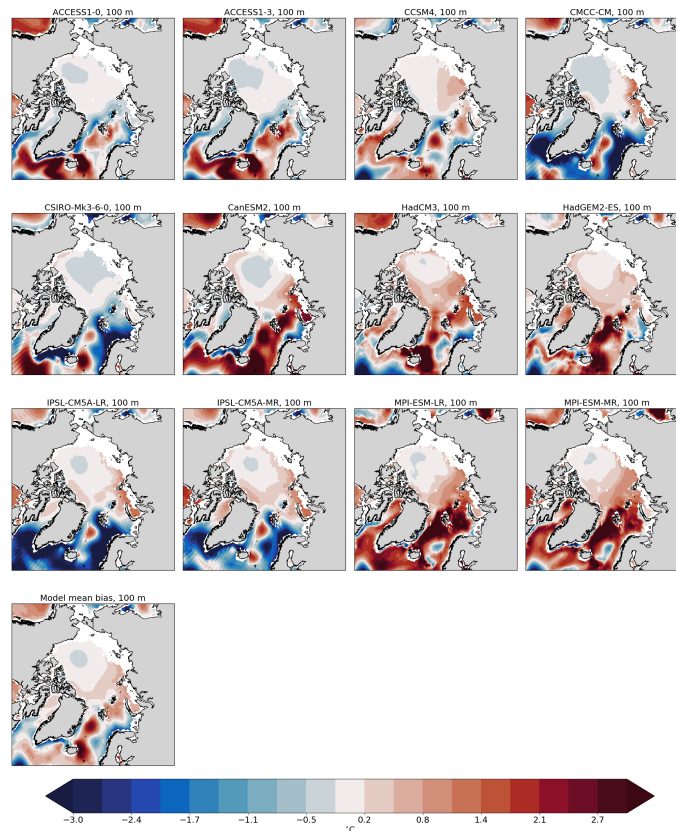

Fig. 4: Mean (1970-2005) salinity bias at 100m relative to PHC3 climatology

Transects

Vertical transects through arbitrary sections are important for analysis of vertical distribution of ocean water properties and especially useful when exchange between different ocean basins is evaluated. We have implemented diagnostics that allow for the definition of an arbitrary ocean section by providing set of points on the ocean surface. For each point, a vertical profile on the original model levels is interpolated. All profiles are then connected to form a transect. The great-circle distance between the points is calculated and used as along-track distance.

One of the main use cases is to create vertical sections across ocean passages, for example Fram Strait.

Plots transect maps for pre-defined set of transects (defined in *regions.py*, see below). The *transect_depth* defines maximum depth of the transect. Transects are calculated from data averaged over the whole time period.

Related settings in the recipe:

```
# Select regions (transects) to plot
# Available options are:
# AWpath - transect along the path of the Atlantic Water
# Fram - Fram strait
transects_regions: ["AWpath", "Fram"]
```

```
# Variables to plot on transects
transects_vars: ['thetao', 'so']
# Color maps for every variable
transects_cmap: ['Spectral_r', 'custom_salinity1']
# Data limits for plots,
# List of the same size as the list of the variables
# each entry is [vmin, vmax, number of levels, rounding limit]
transects_limits: [[-2, 4, 20, 1], [30.5, 35.1, 47, 2]]
# Maximum depth to plot the data
transects_depth: 1500
# number of columns
transects_ncol: 3
```
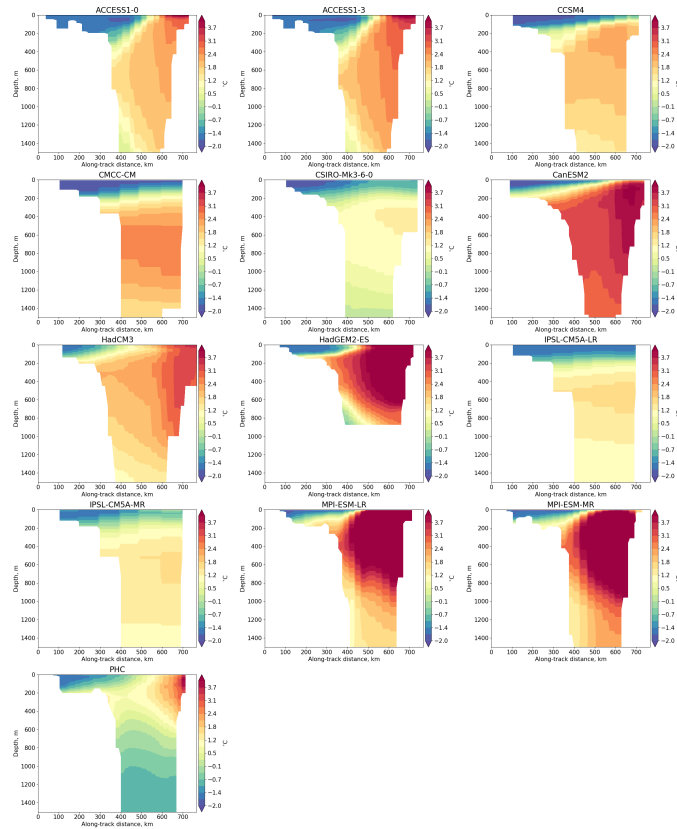

Fig. 5: Mean (1970-2005) potential temperature across the Fram strait.

Atlantic Water core depth and temperature

Atlantic water is a key water mass of the Arctic Ocean and its proper representation is one of the main challenges in Arctic Ocean modelling. We have created two metrics by which models can be easily compared in terms of Atlantic water simulation. The temperature of the Atlantic Water core is calculated for every model as the maximum potential temperature between 200 and 1000 meters depth in the Eurasian Basin. The depth of the Atlantic Water core is calculated as the model level depth where the maximum temperature is found in Eurasian Basin (Atlantic water core temperature).

The AW core depth and temperature will be calculated from data generated for Hovmoeller diagrams for *EB* region, so it should be selected in the Hovmoeller diagrams settings as one of the *hofm_regions*.

In order to evaluate the spatial distribution of Atlantic water in different climate models we also provide diagnostics with maps of the spatial temperature distribution at model's Atlantic Water depth.
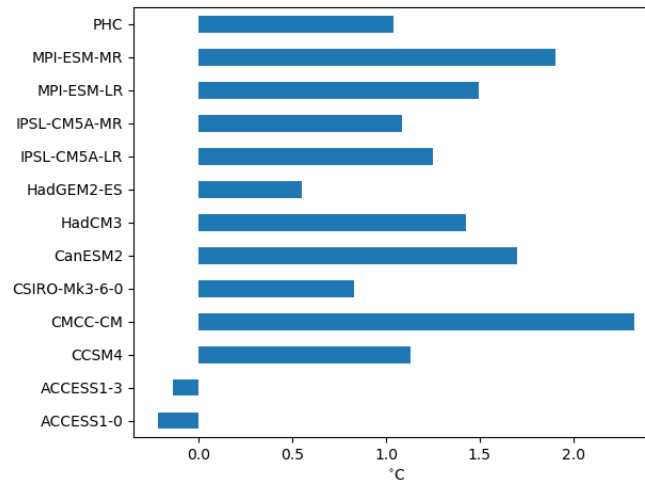

Fig. 6: Mean (1970-2005) Atlantic Water core temperature. PHC33 is an observed climatology.

TS-diagrams

T-S diagrams combine temperature and salinity, which allows the analysis of water masses and their potential for mixing. The lines of constant density for specific ranges of temperature and salinity are shown on the background of the T-S diagram. The dots on the diagram are individual grid points from specified region at all model levels within user specified depth range.

Related settings in the recipe:

```
tsdiag_regions: ["AB" , 'EB']
# Maximum depth to consider data for TS diagrams
tsdiag_depth: 1500
# Number of columns
tsdiag_ncol: 3
```
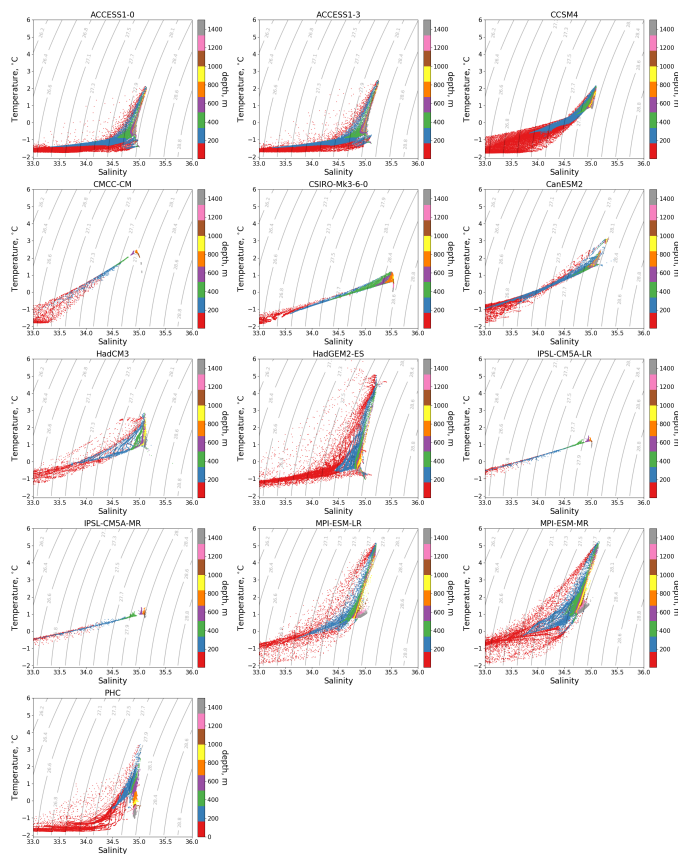
Fig. 7: Mean (1970-2005) T-S diagrams for Eurasian Basin of the Arctic Ocean.

13.1.3 Available diagnostics

The following python modules are included in the diagnostics package:

- arctic_ocean.py : Reads settings from the recipe and call functions to do analysis and plots.
- getdata.py : Deals with data preparation.
- interpolation.py : Include horizontal and vertical interpolation functions specific for ocean models.
- plotting.py : Ocean specific plotting functions
- regions.py : Contains code to select specific regions, and definition of the regions themselves.
- utils.py : Helpful utilites.

Diagnostics are stored in diag_scripts/arctic_ocean/

Variables

- thetao (ocean, monthly, longitude, latitude, time)
- so (ocean, monthly, longitude, latitude, time)

Observations and reformat scripts

- PHC3 climatology

References

- Ilıcak, M. et al., An assessment of the Arctic Ocean in a suite of interannual CORE-II simulations. Part III: Hydrography and fluxes, Ocean Modelling, Volume 100, April 2016, Pages 141-161, ISSN 1463-5003, doi.org/10.1016/j.ocemod.2016.02.004
- Steele, M., Morley, R., & Ermold, W. (2001). PHC: A global ocean hydrography with a high-quality Arctic Ocean. Journal of Climate, 14(9), 2079-2087.
- Wang, Q., et al., An assessment of the Arctic Ocean in a suite of interannual CORE-II simulations. Part I: Sea ice and solid freshwater, Ocean Modelling, Volume 99, March 2016, Pages 110-132, ISSN 1463-5003, doi.org/10.1016/j.ocemod.2015.12.008
- Wang, Q., Ilicak, M., Gerdes, R., Drange, H., Aksenov, Y., Bailey, D. A., . . . & Cassou, C. (2016). An assessment of the Arctic Ocean in a suite of interannual CORE-II simulations. Part II: Liquid freshwater. Ocean Modelling, 99, 86-109, doi.org/10.1016/j.ocemod.2015.12.009

13.2 Climate Variability Diagnostics Package (CVDP)

13.2.1 Overview

The Climate Variability Diagnostics Package (CVDP) developed by NCAR's Climate Analysis Section is an analysis tool that documents the major modes of climate variability in models and observations, including ENSO, Pacific Decadal Oscillation, Atlantic Multi-decadal Oscillation, Northern and Southern Annular Modes, North Atlantic Oscillation, Pacific North and South American teleconnection patterns. For details please refer to the [1] and [2].

13.2.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_cvdp.yml

Diagnostics are stored in diag_scripts/cvdp/

- cvdp_wrapper.py

13.2.3 User settings in recipe

The recipe can be run with several data sets including different model ensembles, multi-model mean statistics are currently not supported.

13.2.4 Variables

- ts (atmos, monthly mean, longitude latitude time)
- tas (atmos, monthly mean, longitude latitude time)
- pr (atmos, monthly mean, longitude latitude time)
- psl (atmos, monthly mean, longitude latitude time)

13.2.5 Observations and reformat scripts

None.

13.2.6 References

[1] http://www.cesm.ucar.edu/working_groups/CVC/cvdp/

[2] https://github.com/NCAR/CVDP-ncl

13.2.7 Example plots

13.3 Nino indices, North Atlantic Oscillation (NAO), Souther Oscillation Index (SOI)

13.3.1 Overview

The goal of this diagnostic is to compute indices based on area averages.

In recipe_combined_indices.yml, after defining the period (historical or future projection), the variable is selected. The predefined areas are:

- Nino 3
- Nino 3.4
- Nino 4
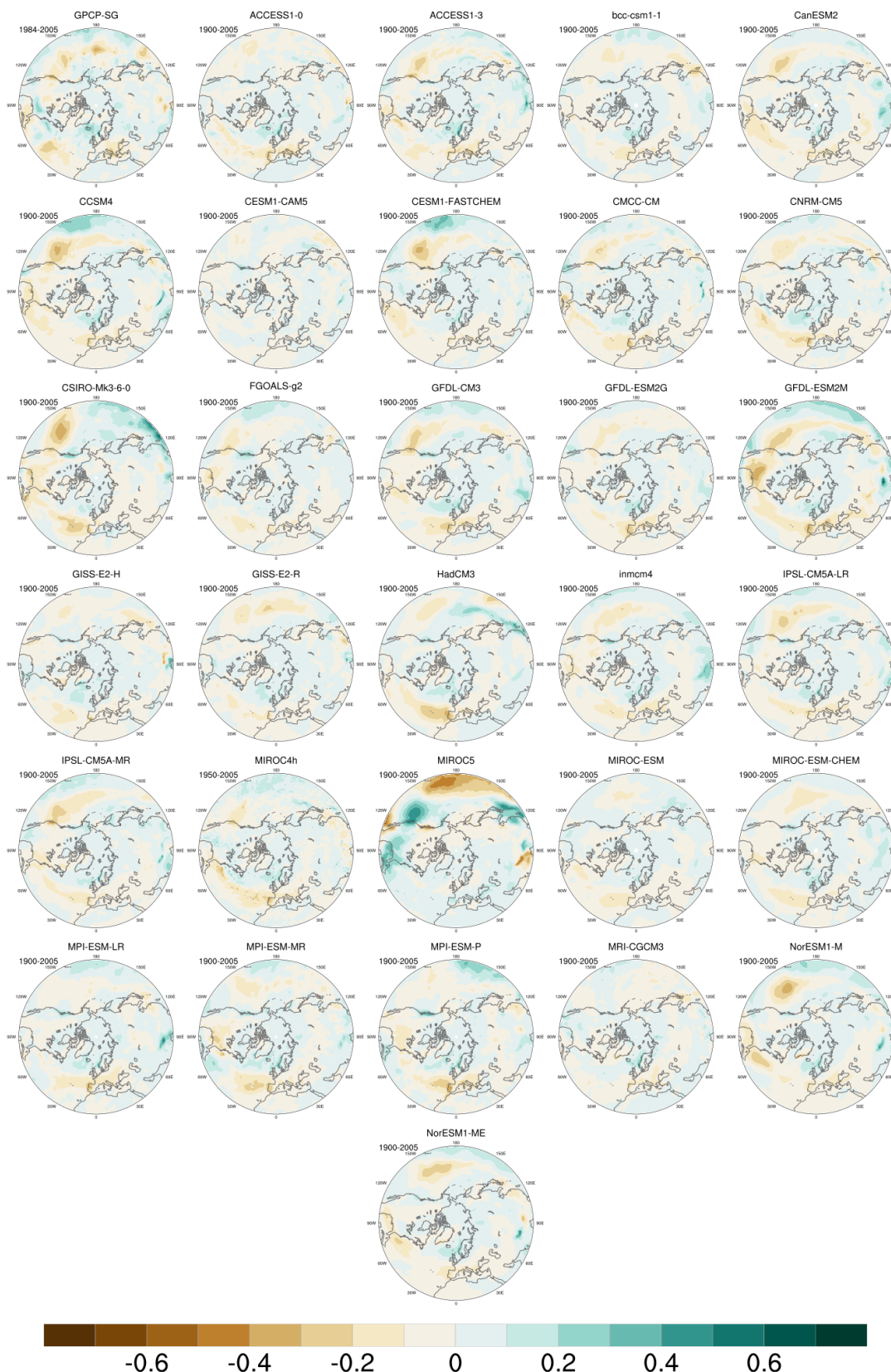- North Atlantic Oscillation (NAO)

NAM PR Regressions (Annual)

Fig. 8: Regression of the precipitation anomalies (PR) onto the Northern Annular Mode (NAM) index for the time period 1900-2005 for 30 CMIP5 models and observations (GPCP (pr) / IFS-Cy31r2 (psl); time period 1984-2005).

- Southern Oscillation Index (SOI)

13.3.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_combined_indices.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- combined_indices.R : calculates the area-weighted means and multi-model means, with or without weights

13.3.3 User settings

User setting files are stored in recipes/

1. recipe_combined_indices.yml

   Required settings for script

   - region: one of the following strings Nino3, Nino3.4, Nino4, NAO, SOI

   - running_mean: an integer specifying the length of the window (in months) to be used for computing the running mean.

   - moninf: an integer can be given to determine the first month of the seasonal mean to be computed (from 1 to 12, corresponding to January to December respectively).

   - monsup: an integer specifying the last month to be computed (from 1 to 12, corresponding to January to December respectively).

   - standardized: 'true' or 'false' to specify whether to compute the standarization of the variable.

     Required settings for preprocessor (only for 3D variables)

     extract_levels:

   - levels: [50000] # e.g. for 500 hPa level

   - scheme: nearest

13.3.4 Variables

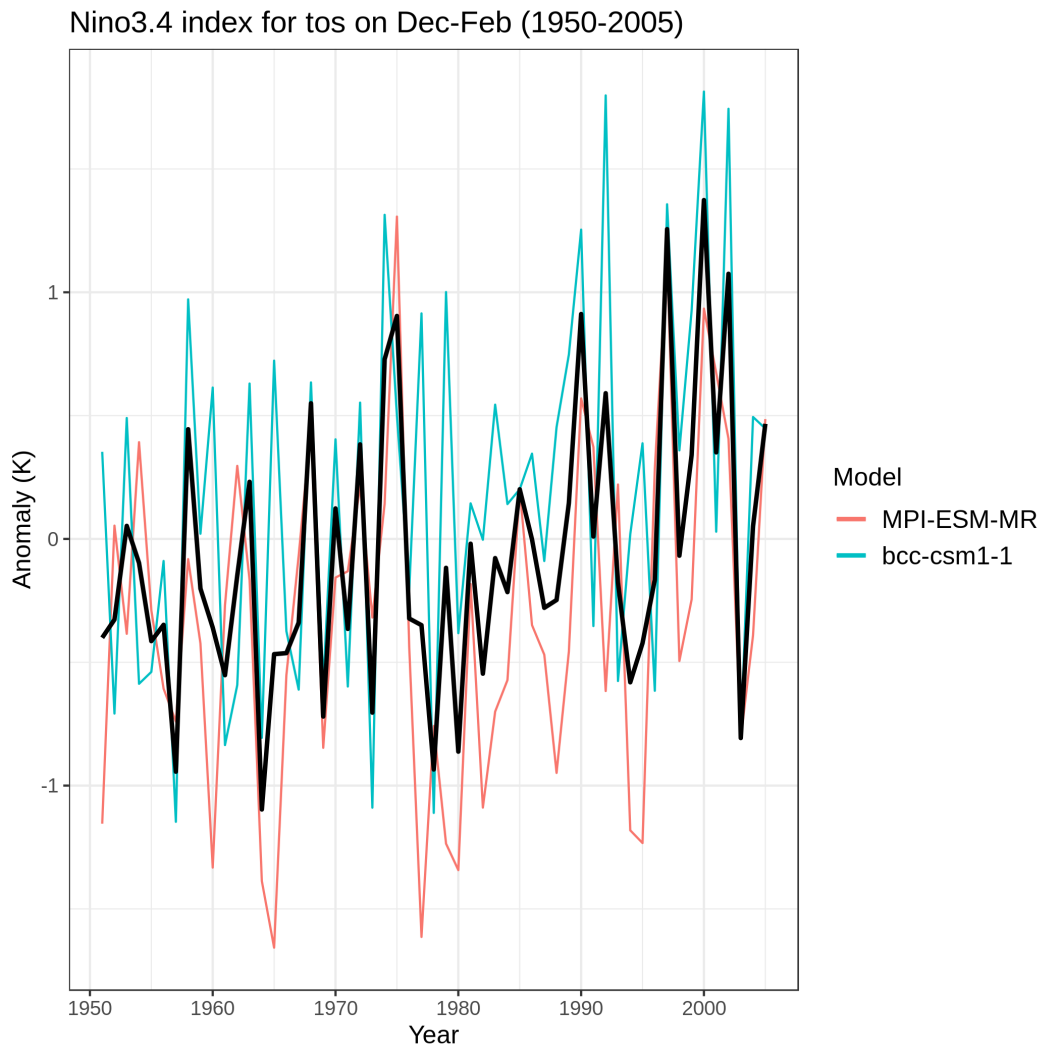- all variables (atmos/ocean, monthly, longitude, latitude, time)

13.3.5 Observations and reformat scripts

None

13.3.6 References

- Trenberth, Kevin & National Center for Atmospheric Research Staff (Eds). Last modified 11 Jan 2019. "The Climate Data Guide: Nino SST Indices (Nino 1+2, 3, 3.4, 4; ONI and TNI)." Retrieved from https://climatedataguide.ucar.edu/climate-data/nino-sst-indices-nino-12-3-34-4-oni-and-tni.

13.3.7 Example plots

Nino3.4 index for tos on Dec-Feb (1950-2005)

Time series of the standardized sea surface temperature (tos) area averaged over the Nino 3.4 region during the boreal winter (December-January-February). The time series correspond to the MPI-ESM-MR (red) and BCC-CSM1-1 (blue) models and their mean (black) during the period 1950-2005 for the ensemble r1p1i1 of the historical simulations.

13.4 Ocean diagnostics

13.4.1 Overview

These recipes are used for evaluating the marine component of models of the earth system. Using these recipes, it should be possible to evaluate both the physical models and biogeochemistry models. All these recipes use the ocean diagnostics package.

The ocean diagnostics package contains several diagnostics which produce figures and statistical information of models of the ocean. The datasets have been pre-processed by ESMValTool, based on recipes in the recipes directory. Most of the diagnostics produce two or less types of figure, and several diagnostics are called by multiple recipes.

Each diagnostic script expects a metadata file, automatically generated by ESMValTool, and one or more pre-processed dataset. These are passed to the diagnostic by ESMValTool in the settings.yml and metadata.yml files.

The ocean diagnostics toolkit can not figure out how to plot data by itself. The current version requires the recipe to produce the correct pre-processed data for each diagnostic script. ie: to produce a time series plot, the preprocessor must produce a time-dimensional dataset.

While these tools were built to evaluate the ocean component models, they also can be used to produce figures for other domains. However, there are some ocean specific elements, such as the z-direction being positive and reversed, and some of the map plots have the continents coloured in by default.

As elsewhere, both the model and observational datasets need to be compliant with the CMOR data.

13.4.2 Available recipes

- recipe_ocean_amoc.yml
- recipe_ocean_example.yml
- recipe_ocean_scalar_fields.yml
- recipe_ocean_bgc.yml
- recipe_ocean_quadmap.yml
- *recipe_ocean_ice_extent.yml*
- recipe_ocean_multimap.yml

recipe_ocean_amoc.yml

The recipe_ocean_amoc.yml is an recipe that produces figures describing the Atlantic Meridional Overturning Circulation (AMOC) and the drake passage current.

The recipes produces time series of the AMOC at 26 north and the drake passage current.
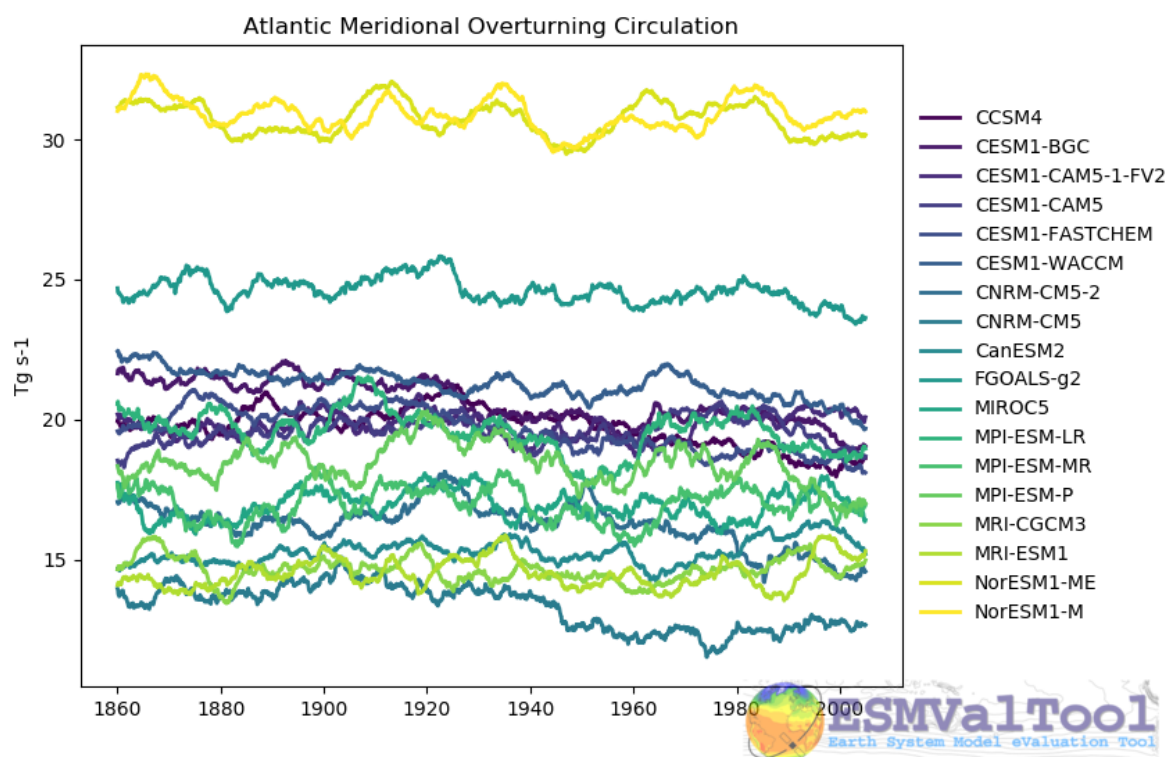
This figure shows the multi model comparison of the AMOC from several CMIP5 historical simulations, with a 6 year moving average (3 years either side of the central value). A similar figure is produced for each individual model, and for the Drake Passage current.

This recipe also produces a contour transect and a coloured transect plot showing the Atlantic stream function for each individual model, and a multi-model contour is also produced:
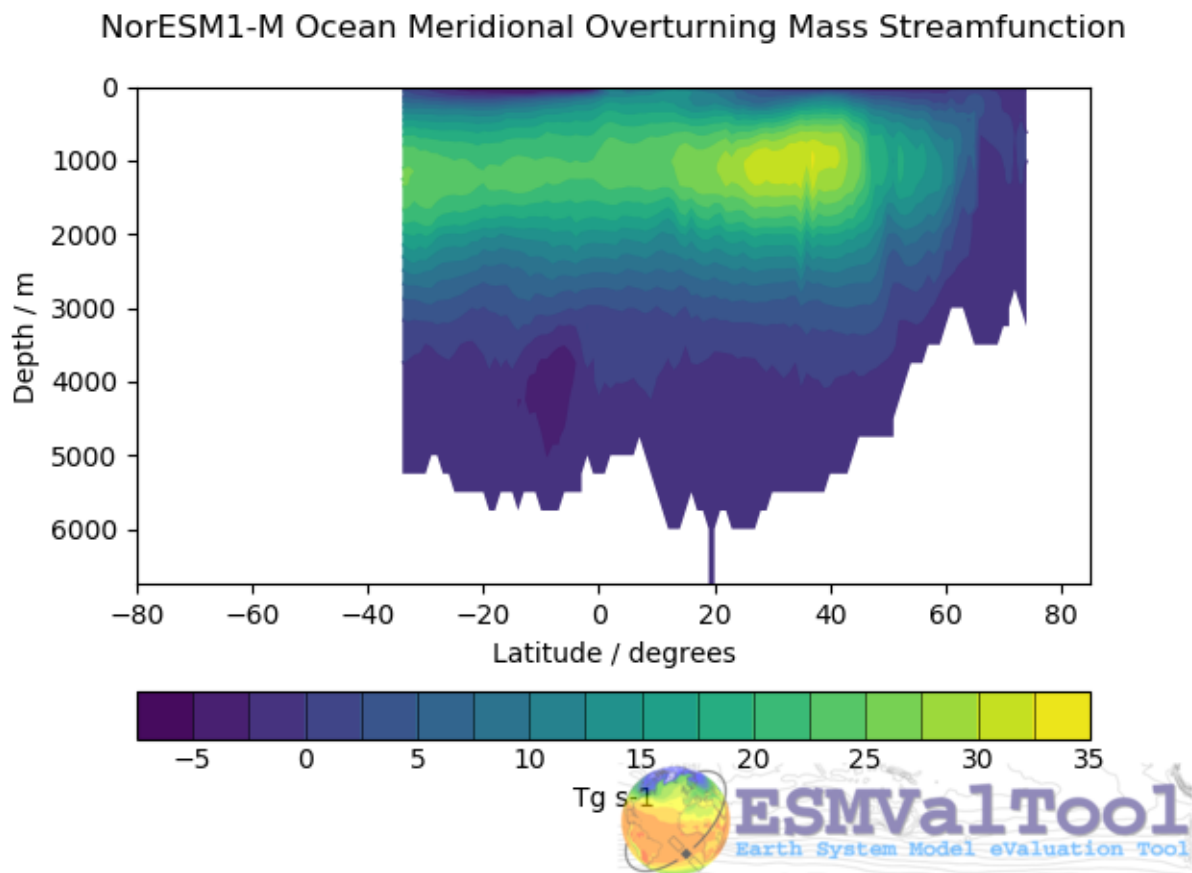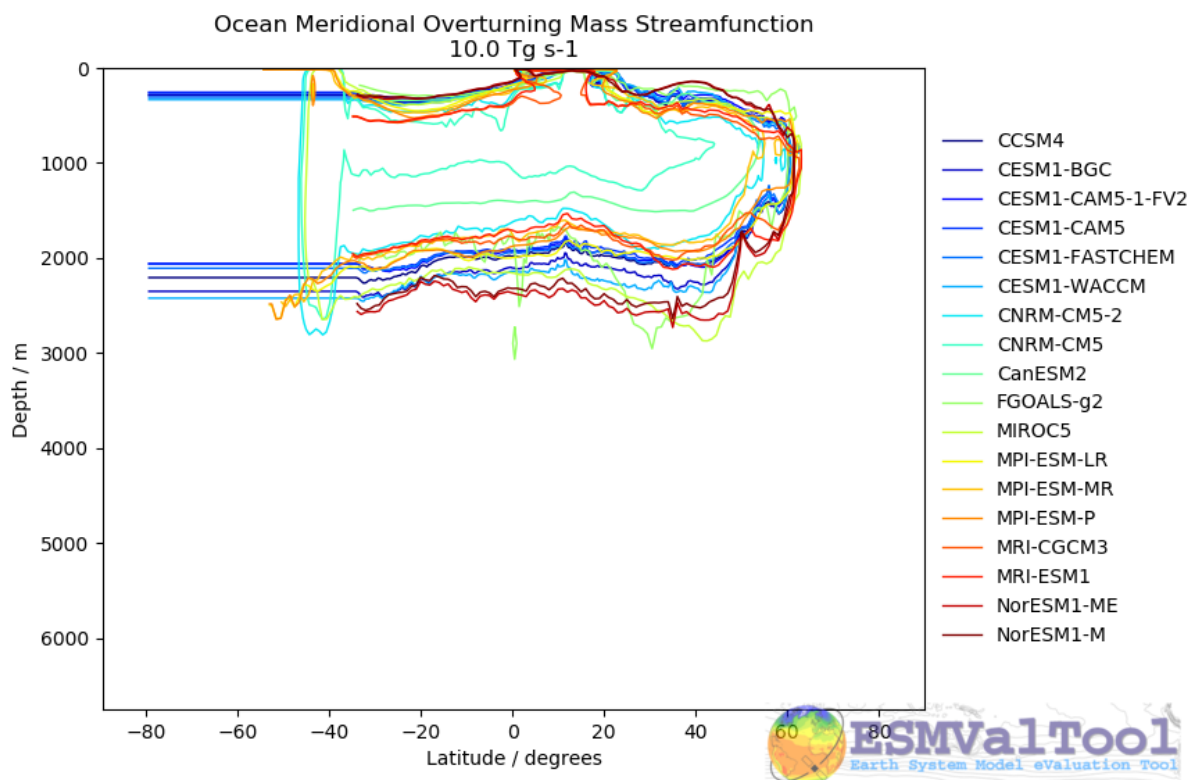
recipe_ocean_example.yml

The recipe_ocean_example.yml is an example recipe which shows several examples of how to manipulate marine model data using the ocean diagnostics tools.

While several of the diagnostics here have specific uses in evaluating models, it is meant to be a catch-all recipe demonstrating many different ways to evaluate models.

All example calculations are performed using the ocean temperature in a three dimensional field (thetao), or at the surface (tos). This recipe demonstrates the use of a range of preprocessors in a marine context, and also shows many of the standard model-only diagnostics (no observational component is included.)

This recipe includes examples of how to manipulate both 2D and 3D fields to produce:

- Time series:
    - Global surface area weighted mean time series
    - Volume weighted average time series within a specific depth range
    - Area weighted average time series at a specific depth
    - Area weighted average time series at a specific depth in a specific region.
    - Global volume weighted average time series
    - Regional volume weighted average time series
- Maps:
    - Global surface map (from 2D ad 3D initial fields)

- Global surface map using re-gridding to a regular grid

- Global map using re-gridding to a regular grid at a specific depth level

- Regional map using re-gridding to a regular grid at a specific depth level

- Transects:

  - Produce various transect figure showing a re-gridded transect plot, and multi model comparisons

- Profile:

  - Produce a Global area-weighted depth profile figure

  - Produce a regional area-weighted depth profile figure

All the these fields can be expanded using a

recipe_ocean_bgc.yml

The recipe_ocean_bgc.yml is an example recipe which shows a several simple examples of how to manipulate marine biogeochemical model data.

This recipe includes the following fields:

- Global total volume-weighted average time series:

  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *

  - chlorophyll, iron, total alkalinity (no observations)

- Surface area-weighted average time series:

  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *

  - fgco2 (global total), integrated primary production, chlorophyll, iron, total alkalinity (no observations)

- Scalar fields time series:

  - mfo (including stuff like drake passage)

- Profiles:

  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *

  - chlorophyll, iron, total alkalinity (no observations)

- Maps + contours:

  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *

  - chlorophyll, iron, total alkalinity (no observations)

- Transects + contours:

  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *

  - chlorophyll, iron, no observations)

* Note that Phosphate is also available as a WOA diagnostic, but I haven't included it as HadGEM2-ES doesn't include a phosphate field.

This recipe uses the World Ocean Atlas data, which can be downloaded from: https://www.nodc.noaa.gov/OC5/woa13/woa13data.html (last access 10/25/2018)

Instructions: Select the "All fields data links (1° grid)" netCDF file, which contain all fields.

---

recipe_ocean_quadmap.yml

The recipe_ocean_quadmap.yml is an example recipe showing the diagnostic_maps_quad.py diagnostic. This diagnostic produces an image showing four maps. Each of these four maps show latitude vs longitude and the cube value is used as the colour scale. The four plots are:

model1	model 1 minus model2
model2 minus obs	model1 minus obs

These figures are also known as Model vs Model vs Obs plots.

The figure produced by this recipe compares two versions of the HadGEM2 model against ATSR sea surface temperature:
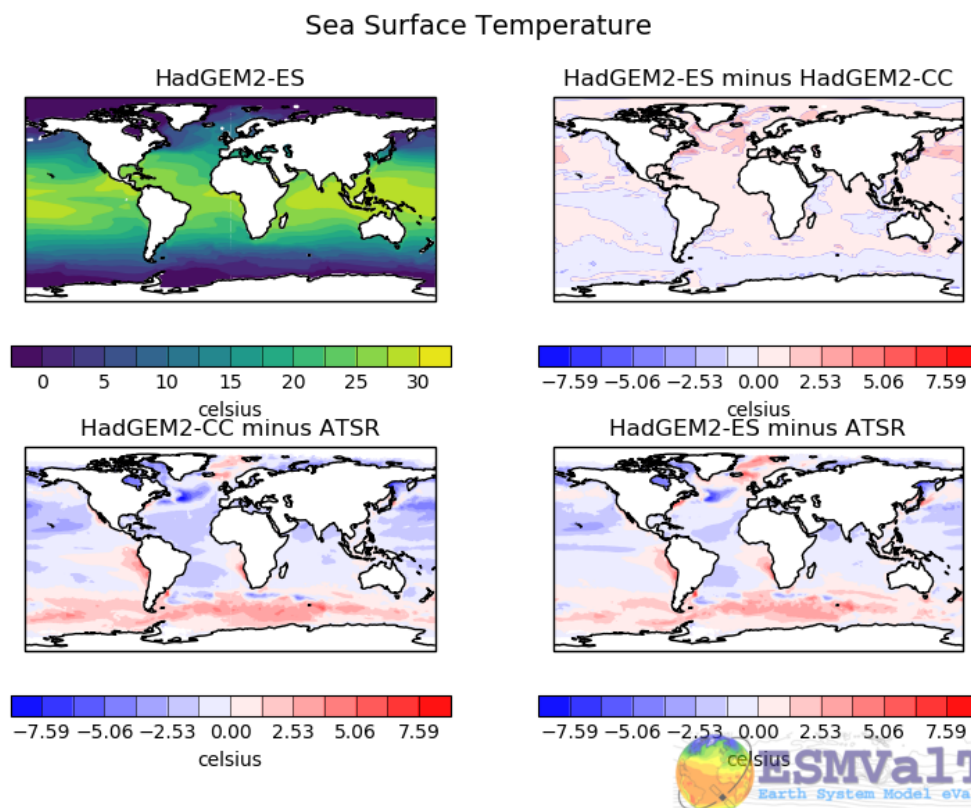
This kind of figure can be very useful when developing a model, as it allows model developers to quickly see the impact of recent changes to the model.

recipe_ocean_ice_extent.yml

The *recipe_ocean_ice_extent.yml* recipe produces several metrics describing the behaviour of sea ice in a model, or in multiple models.

This recipe has four preprocessors, a combinatorial combination of

- Regions: Northern or Southern Hemisphere
- Seasons: December-January-February or June-July-August

Once these seasonal hemispherical fractional ice cover is processed, the resulting cube is passed 'as is' to the diagnostic_seaice.py diagnostic.

This diagnostic produces the plots:

- Polar Stereographic projection Extent plots of individual models years.

- Polar Stereographic projection maps of the ice cover and ice extent for individual models.

- A time series of Polar Stereographic projection Extent plots - see below.

- Time series plots of the total ice area and the total ice extent.

The following image shows an example of the sea ice extent plot, showing the Summer Northern hemisphere ice extent for the HadGEM2-CC model, in the historical scenario.
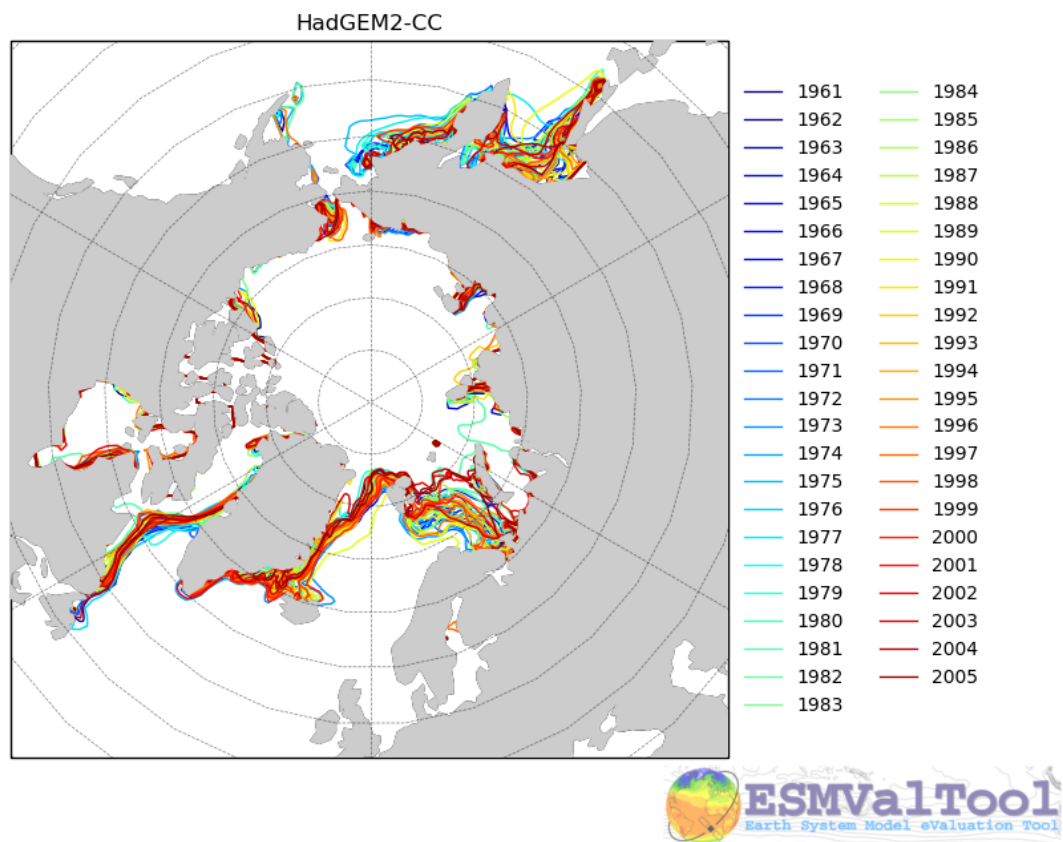
The sea ice diagnostic is unlike the other diagnostics in the ocean diagnostics toolkit. The other tools are build to be generic plotting tools which work with any field (ie `diagnostic_timeseries.py` works fine for Temperature, Chlorophyll, or any other field. On the other hand, the sea ice diagnostic is the only tool that performs a field specific evaluation.

The diagnostic_seaice.py diagnostic is more fully described below.

recipe_ocean_multimap.yml

The recipe_ocean_multimap.yml is an example recipe showing the diagnostic_maps_multimodel.py diagnostic. This diagnostic produces an image showing Model vs Observations maps or only Model fields when observational data are not provided. Each map shows latitude vs longitude fields and user defined values are used to set the colour scale. Plot layout can be modified by modifying the *layout_rowcol* argument.

The figure produced by this recipe compares the ocean surface CO2 fluxes for 16 different CMIP5 model against Landschuetzer2016 observations.

The diagnostic_maps_multimodel.py diagnostic is documented below.

13.4.3 Available diagnostics

Diagnostics are stored in the diag_scripts directory: ocean.

The following python modules are included in the ocean diagnostics package. Each module is described in more detail both below and inside the module.

- diagnostic_maps.py
- diagnostic_maps_quad.py
- diagnostic_model_vs_obs.py
- diagnostic_profiles.py
- diagnostic_seaice.py
- diagnostic_timeseries.py
- diagnostic_tools.py
- diagnostic_transects.py
- diagnostic_maps_multimodel.py

diagnostic_maps.py

The diagnostic_maps.py produces a spatial map from a NetCDF. It requires the input netCDF to have the following dimensions. Either:

- A two dimensional file: latitude, longitude.
- A three dimensional file: depth, latitude, longitude.

In the case of a 3D netCDF file, this diagnostic produces a map for EVERY layer. For this reason, we recommend extracting a small number of specific layers in the preprocessor, using the *extract_layer* preprocessor.

This script can not process NetCDFs with multiple time steps. Please use the *climate_statistics* preprocessor to collapse the time dimension.

This diagnostic also includes the optional arguments, *threshold* and *thresholds*.

- threshold: a single float.
- thresholds: a list of floats.

Only one of these arguments should be provided at a time. These two arguments produce a second kind of diagnostic map plot: a contour map showing the spatial distribution of the threshold value, for each dataset. Alternatively, if the thresholds argument is used instead of threshold, the single-dataset contour map shows the contours of all the values in the thresholds list.

If multiple datasets are provided, in addition to the single dataset contour, a multi-dataset contour map is also produced for each value in the thresholds list.

Some appropriate preprocessors for this diagnostic would be:

For a Global 2D field:

```
prep_map_1:
  climate_statistics:
```

For a regional 2D field:

```
prep_map_2:
    extract_region:
      start_longitude: -80.
      end_longitude: 30.
      start_latitude: -80.
      end_latitude: 80.
climate_statistics:
  operator: mean
```

For a Global 3D field at the surface and 10m depth:

```
  prep_map_3:
    custom_order: true
    extract_levels:
      levels: [0., 10.]
      scheme: linear_horizontal_extrapolate_vertical
climate_statistics:
  operator: mean
```

For a multi-model comparison mean of 2D global fields including contour thresholds.

```
  prep_map_4:
    custom_order: true
climate_statistics:
  operator: mean
    regrid:
      target_grid: 1x1
      scheme: linear
```

And this also requires the threshold key in the diagnostic:

```
diagnostic_map:
  variables:
    tos: # Temperature ocean surface
      preprocessor: prep_map_4
      field: TO2M
  scripts:
    Ocean_regrid_map:
      script: ocean/diagnostic_maps.py
      thresholds: [5, 10, 15, 20]
```

diagnostic_maps_quad.py

The diagnostic_maps_quad.py diagnostic produces an image showing four maps. Each of these four maps show latitude vs longitude and the cube value is used as the colour scale. The four plots are:

model1	model 1 minus model2
model2 minus obs	model1 minus obs

These figures are also known as Model vs Model vs Obs plots.

This diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cubes received by this diagnostic (via the settings.yml and metadata.yml files) have no time component, a small number of depth layers, and a latitude and longitude coordinates.

An appropriate preprocessor for a 2D field would be:

```
prep_quad_map:
climate_statistics:
    operator: mean
```

and an example of an appropriate diagnostic section of the recipe would be:

```
diag_map_1:
  variables:
    tos: # Temperature ocean surface
      preprocessor: prep_quad_map
      field: TO2Ms
      mip: Omon
  additional_datasets:
#       filename: tos_ATSR_L3_ARC-v1.1.1_199701-201112.nc
#       download from: https://datashare.is.ed.ac.uk/handle/10283/536
    - {dataset: ATSR, project: obs4mips, level: L3, version: ARC-v1.1.1, ␣
↪start_year: 2001, end_year: 2003, tier: 3}
  scripts:
    Global_Ocean_map:
      script: ocean/diagnostic_maps_quad.py
      control_model: {dataset: HadGEM2-CC, project: CMIP5, mip: Omon, exp:␣
↪historical, ensemble: r1i1p1}
      exper_model: {dataset: HadGEM2-ES, project: CMIP5, mip: Omon, exp:␣
↪historical, ensemble: r1i1p1}
      observational_dataset: {dataset: ATSR, project: obs4mips,}
```

Note that the details about the control model, the experiment models and the observational dataset are all provided in the script section of the recipe.

diagnostic_model_vs_obs.py

The diagnostic_model_vs_obs.py diagnostic makes model vs observations maps and scatter plots. The map plots shows four latitude vs longitude maps:

Model	Observations
Model minus Observations	Model over Observations

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

This diagnostic also includes the optional arguments, *maps_range* and *diff_range* to manually define plot ranges. Both arguments are a list of two floats to set plot range minimun and maximum values respectively for Model and Observations maps (Top panels) and for the Model minus Observations panel (bottom left). Note that if input data have negative values the Model over Observations map (bottom right) is not produced.

The scatter plots plot the matched model coordinate on the x axis, and the observational dataset on the y coordinate, then performs a linear regression of those data and plots the line of best fit on the plot. The parameters of the fit are also shown on the figure.

An appropriate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_map:
    extract_levels:
      levels:  [100., ]
      scheme: linear_extrap
climate_statistics:
  operator: mean
    regrid:
      target_grid: 1x1
      scheme: linear
```

diagnostic_maps_multimodel.py

The diagnostic_maps_multimodel.py diagnostic makes model(s) vs observations maps and if data are not provided it draws only model field.

It is always nessary to define the overall layout trough the argument *layout_rowcol*, which is a list of two integers indicating respectively the number of rows and columns to organize the plot. Observations has not be accounted in here as they are automatically added at the top of the figure.

This diagnostic also includes the optional arguments, *maps_range* and *diff_range* to manually define plot ranges. Both arguments are a list of two floats to set plot range minimun and maximum values respectively for variable data and the Model minus Observations range.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

An appropriate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_map:
    extract_levels:
      levels:  [100., ]
      scheme: linear_extrap
climate_statistics:
  operator: mean
    regrid:
      target_grid: 1x1
      scheme: linear
```

diagnostic_profiles.py

The diagnostic_profiles.py diagnostic produces images of the profile over time from a cube. These plots show cube value (ie temperature) on the x-axis, and depth/height on the y axis. The colour scale is the annual mean of the cube data. Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has a time component, and depth component, but no latitude or longitude coordinates.

An appropriate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_profile:
    extract_volume:
      long1: 0.
      long2:  20.
      lat1:  -30.
      lat2:  30.
      z_min: 0.
      z_max: 3000.
    area_statistics:
      operator: mean
```

diagnostic_timeseries.py

The diagnostic_timeseries.py diagnostic produces images of the time development of a metric from a cube. These plots show time on the x-axis and cube value (ie temperature) on the y-axis.

Two types of plots are produced: individual model timeseries plots and multi model time series plots. The individual plots show the results from a single cube, even if this cube is a multi-model mean made by the *multimodel* preprocessor.

The multi model time series plots show several models on the same axes, where each model is represented by a different line colour. The line colours are determined by the number of models, their alphabetical order and the *jet* colour scale. Observational datasets and multimodel means are shown as black lines.

This diagnostic assumes that the preprocessors do the bulk of the work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) is time-dimensional cube. This means that the pre-processed netcdf has a time component, no depth component, and no latitude or longitude coordinates.

Some appropriate preprocessors would be :

For a global area-weighted average 2D field:
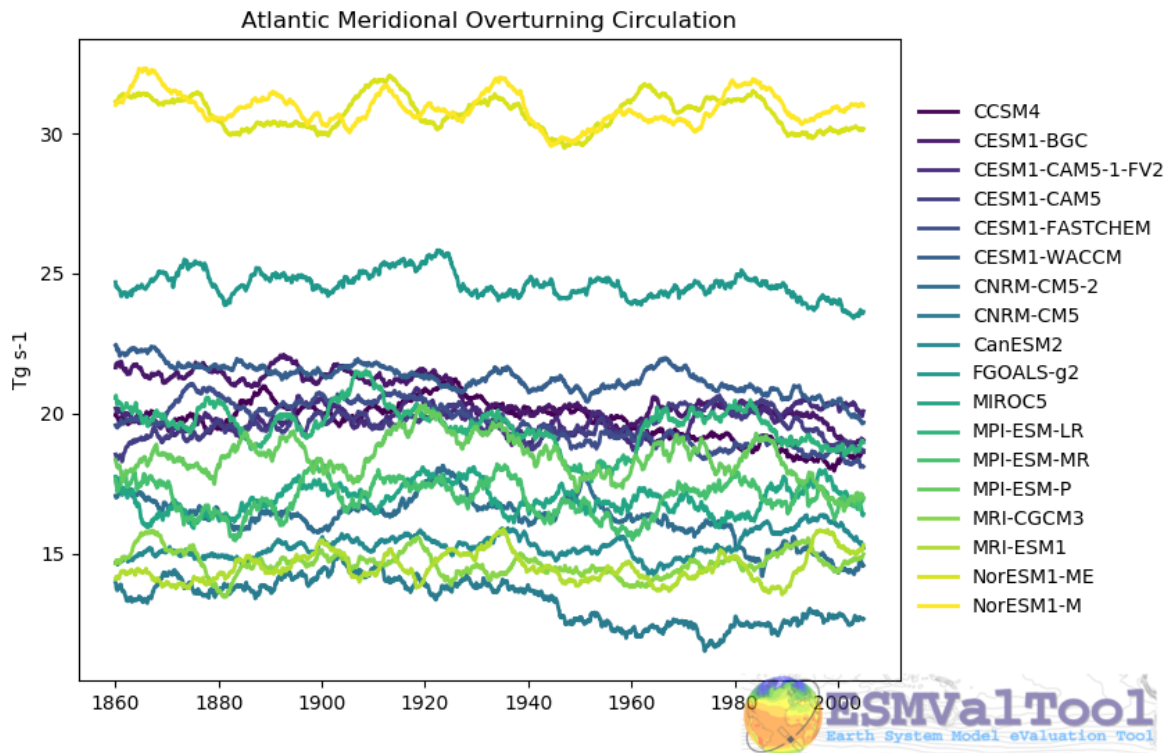
```
area_statistics:
  operator: mean
```

For a global volume-weighted average 3D field:

```
volume_statistics:
  operator: mean
```

For a global area-weighted surface of a 3D field:

```
extract_levels:
  levels: [0., ]
  scheme: linear_horizontal_extrapolate_vertical
area_statistics:
  operator: mean
```

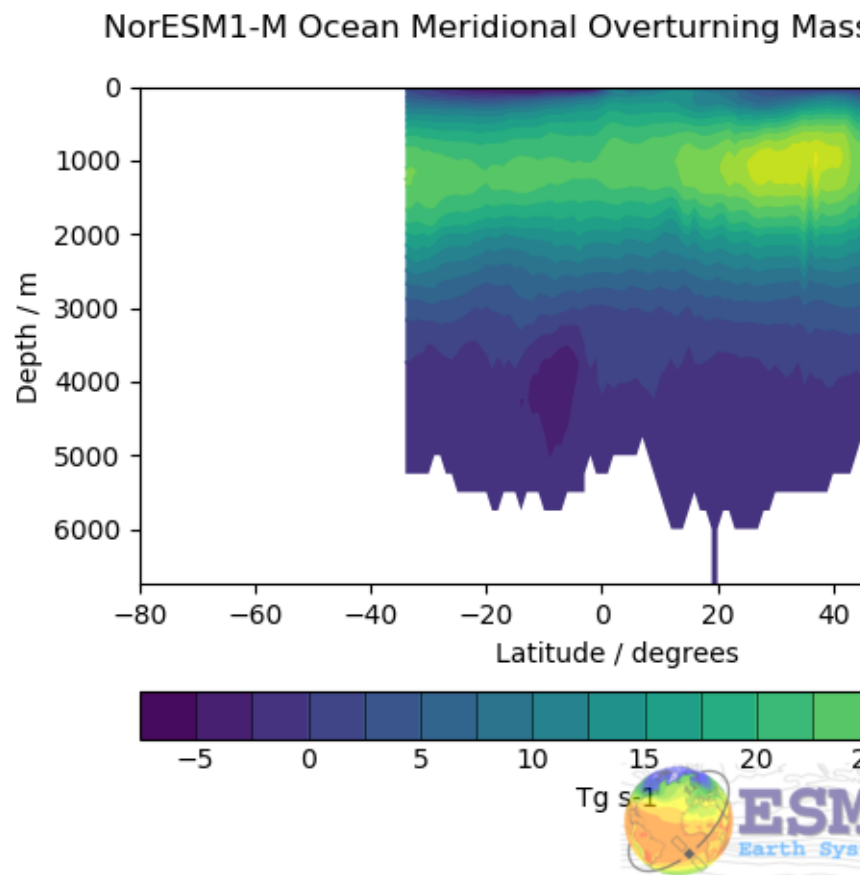An example of the multi-model time series plots can seen here:

diagnostic_transects.py

The diagnostic_transects.py diagnostic produces images of a transect, typically along a constant latitude or longitude.

These plots show 2D plots with either latitude or longitude along the x-axis, depth along the y-axis and and the cube value is used as the colour scale.

This diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, and one of the latitude or longitude coordinates has been reduced to a single value.

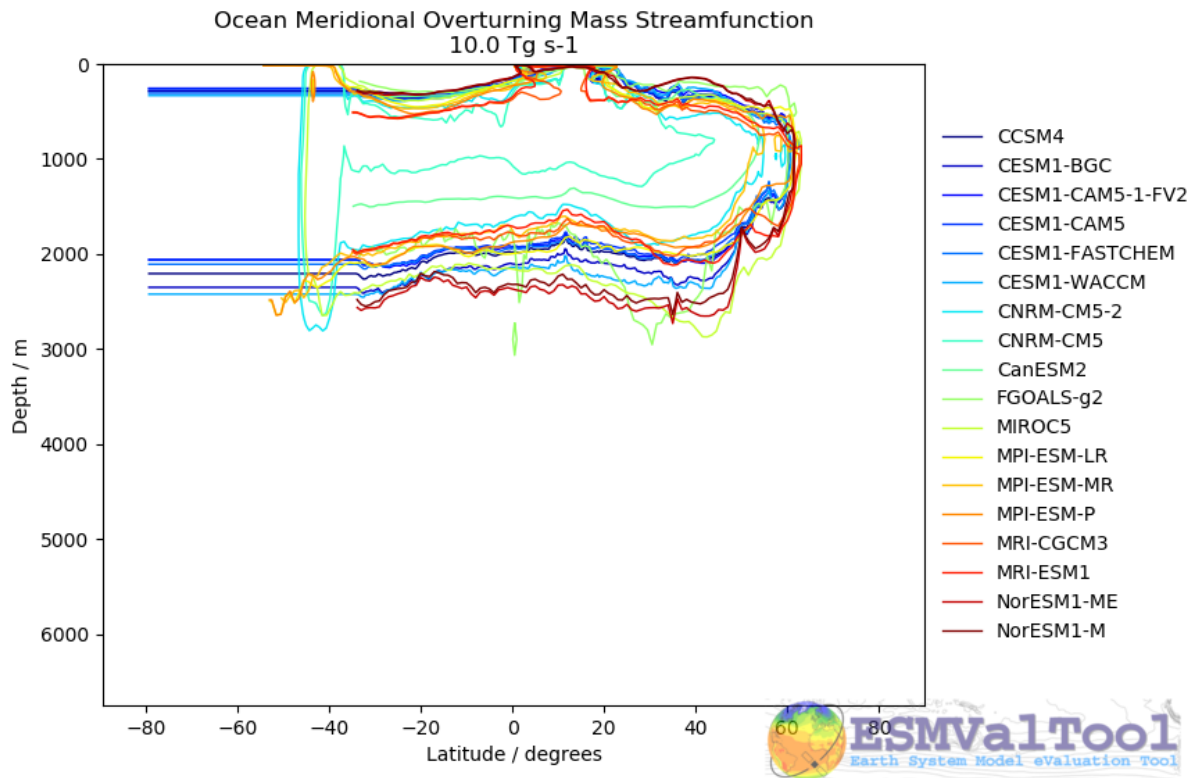An appropriate preprocessor for a 3D+time field would be:

```
climate_statistics:
  operator: mean
extract_slice:
  latitude: [-50.,50.]
  longitude: 332.
```

Here is an example of the transect figure: .. centered::

And here is an example of the multi-model transect contour figure:

diagnostic_seaice.py

The diagnostic_seaice.py diagnostic is unique in this module, as it produces several different kinds of images, including time series, maps, and contours. It is a good example of a diagnostic where the preprocessor does very little work, and the diagnostic does a lot of the hard work.

This was done purposely, firstly to demonstrate the flexibility of ESMValTool, and secondly because Sea Ice is a unique field where several Metrics can be calculated from the sea ice cover fraction.

The recipe Associated with with diagnostic is the recipe_SeaIceExtent.yml. This recipe contains 4 preprocessors which all perform approximately the same calculation. All four preprocessors extract a season: - December, January and February (DJF) - June, July and August (JJA) and they also extract either the North or South hemisphere. The four preprocessors are combinations of DJF or JJA and North or South hemisphere.

One of the four preprocessors is North Hemisphere Winter ice extent:

```
timeseries_NHW_ice_extent: # North Hemisphere Winter ice_extent
  custom_order: true
  extract_time: &time_anchor # declare time here.
      start_year: 1960
      start_month: 12
      start_day: 1
      end_year: 2005
      end_month: 9
      end_day: 31
  extract_season:
    season: DJF
  extract_region:
```

(continues on next page)

```
    start_longitude: -180.
    end_longitude: 180.
    start_latitude: 0.
    end_latitude: 90.
```

Note that the default settings for ESMValTool assume that the year starts on the first of January. This causes a problem for this preprocessor, as the first DJF season would not include the first Month, December, and the final would not include both January and February. For this reason, we also add the *extract_time* preprocessor.

This preprocessor group produces a 2D field with a time component, allowing the diagnostic to investigate the time development of the sea ice extend.

The diagnostic section of the recipe should look like this:

```
diag_ice_NHW:
  description: North Hemisphere Winter Sea Ice diagnostics
  variables:
    sic: # surface ice cover
      preprocessor: timeseries_NHW_ice_extent
      field: TO2M
      mip: OImon
  scripts:
    Global_seaice_timeseries:
      script: ocean/diagnostic_seaice.py
      threshold: 15.
```

Note the the threshold here is 15%, which is the standard cut of for the ice extent.

The sea ice diagnostic script produces three kinds of plots, using the methods:

- *make_map_extent_plots*: extent maps plots of individual models using a Polar Stereographic project.

- *make_map_plots*: maps plots of individual models using a Polar Stereographic project.

- *make_ts_plots*: time series plots of individual models

There are no multi model comparisons included here (yet).

diagnostic_tools.py

The diagnostic_tools.py is a module that contains several python tools used by the ocean diagnostics tools.

These tools are:

- folder: produces a directory at the path provided and returns a string.

- get_input_files: loads a dictionary from the input files in the metadata.yml.

- bgc_units: converts to sensible units where appropriate (ie Celsius, mmol/m3)

- timecoord_to_float: Converts time series to decimal time ie: Midnight on January 1st 1970 is 1970.0

- add_legend_outside_right: a plotting tool, which adds a legend outside the axes.

- get_image_format: loads the image format, as defined in the global user config.yml.

- get_image_path: creates a path for an image output.

- make_cube_layer_dict: makes a dictionary for several layers of a cube.

We just show a simple description here, each individual function is more fully documented in the diagnostic_tools.py module.

13.4.4 A note on the auxiliary data directory

Some of these diagnostic scripts may not function on machines with no access to the internet, as cartopy may try to download the shape files. The solution to this issue is the put the relevant cartopy shapefiles in a directory which is visible to esmvaltool, then link that path to ESMValTool via the *auxiliary_data_dir* variable in your config-user.yml file.

The cartopy masking files can be downloaded from: https://www.naturalearthdata.com/downloads/

In these recipes, cartopy uses the 1:10, physical coastlines and land files:

```
110m_coastline.dbf
110m_coastline.shp
110m_coastline.shx
110m_land.dbf
110m_land.shp
110m_land.shx
```

13.4.5 Associated Observational datasets

The following observations datasets are used by these recipes:

World Ocean ATLAS

These data can be downloaded from: https://www.nodc.noaa.gov/OC5/woa13/woa13data.html (last access 10/25/2018) Select the "All fields data links (1° grid)" netCDF file, which contain all fields.

The following WOA datasets are used by the ocean diagnostics:

- Temperature
- Salinity
- Nitrate
- Phosphate
- Silicate
- Dissolved Oxygen

These files need to be reformatted using the *cmorize_obs_py* script with output name *WOA*.

Landschuetzer 2016

These data can be downloaded from: ftp://ftp.nodc.noaa.gov/nodc/archive/arc0105/0160558/1.1/data/0-data/spco2_1998-2011_ETH_SOM-FFN_CDIAC_G05.nc (last access 02/28/2019)

The following variables are used by the ocean diagnostics:

- fgco2, Surface Downward Flux of Total CO2
- spco2, Surface Aqueous Partial Pressure of CO2
- dpco2, Delta CO2 Partial Pressure

The file needs to be reformatted using the *cmorize_obs_py* script with output name *Landschuetzer2016*.

13.5 Ocean metrics

13.5.1 Overview

The Southern Ocean is central to the global climate and the global carbon cycle, and to the climate's response to increasing levels of atmospheric greenhouse gases. Global coupled climate models and earth system models, however, vary widely in their simulations of the Southern Ocean and its role in, and response to, the ongoing anthropogenic trend. Observationally-based metrics are critical for discerning processes and mechanisms, and for validating and comparing climate and earth system models. New observations and understanding have allowed for progress in the creation of observationally-based data/model metrics for the Southern Ocean.

The metrics presented in this recipe provide a means to assess multiple simulations relative to the best available observations and observational products. Climate models that perform better according to these metrics also better simulate the uptake of heat and carbon by the Southern Ocean. Russell et al. 2018 assessed only a few of the available CMIP5 simulations, but most of the available CMIP5 and CMIP6 climate models can be analyzed with these recipes.

The goal is to create a recipe for recreation of metrics in Russell, J.L., et al., 2018, J. Geophys. Res. – Oceans, 123, 3120-3143, doi: 10.1002/2017JC013461.

13.5.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_russell18jgr.yml

Diagnostics are stored in diag_scripts/russell18jgr/

- russell18jgr-polar.ncl (figures 1, 7, 8): calculates and plots annual-mean variables (tauu, sic, fgco2, pH) as polar contour map.

- russell18jgr-fig2.ncl: calculates and plots The zonal and annual means of the zonal wind stress (N/m^2).

- russell18jgr-fig3b.ncl: calculates and plots the latitudinal position of Subantarctic Front. Using definitions from Orsi et al (1995).

- russell18jgr-fig3b-2.ncl: calculates and plots the latitudinal position of Polar Front. Using definitions from Orsi et al (1995).

- russell18jgr-fig4.ncl: calculates and plots the zonal velocity through Drake Passage (at 69W) and total transport through the passage if the volcello file is available.

- russell18jgr-fig5.ncl: calculates and plots the mean extent of sea ice for September(max) in blue and mean extent of sea ice for February(min) in red.

- russell18jgr-fig5g.ncl: calculates and plots the annual cycle of sea ice area in southern ocean.

- russell18jgr-fig6a.ncl: calculates and plots the density layer based volume transport(in Sv) across 30S based on the layer definitions in Talley (2008).

- russell18jgr-fig6b.ncl: calculates and plots the Density layer based heat transport(in PW) across 30S based on the layer definitions in Talley (2008).

- russell18jgr-fig7h.ncl: calculates and plots the zonal mean flux of fgco2 in gC/(yr * m^2).

- russell18jgr-fig7i.ncl: calculates and plots the cumulative integral of the net CO2 flux from 90S to 30S (in PgC/yr).

- russell18jgr-fig9a.ncl: calculates and plots the scatter plot of the width of the Southern Hemisphere westerly wind band against the annual-mean integrated heat uptake south of 30S (in PW), along with the line of best fit.

- russell18jgr-fig9b.ncl: calculates and plots the scatter plot of the width of the Southern Hemisphere westerly wind band against the annual-mean integrated carbon uptake south of 30S (in Pg C/yr), along with the line of best fit.

- russell18jgr-fig9c.ncl: calculates and plots the scatter plot of the net heat uptake south of 30S (in PW) against the annual-mean integrated carbon uptake south of 30S (in Pg C/yr), along with the line of best fit.

13.5.3 User settings in recipe

1. Script russell18jgr-polar.ncl

   Required settings (scripts)

   - styleset : CMIP5(recommended), default, etc.

   - ncdf : default(recommended), CMIP5, etc.

   - max_lat : -30.0

   Optional settings (scripts)

   - grid_max : 0.4 (figure 1), 30 (figure 7), 8.2 (figure 8)

   - grid_min : -0.4 (figure 1), -30 (figure 7), 8.0 (figure 8)

   - grid_step : 0.1 (figure 1), 2.5 (figure 7), 0.1 (figure 8)

   - colormap : BlWhRe (figure 7)

   - colors : [[237.6, 237.6, 0.], [255, 255, 66.4], [255, 255, 119.6], [255, 255, 191.8], [223.8, 191.8, 223.8], [192.8, 127.5, 190.8], [161.6, 65.3, 158.6], [129.5, 1.0, 126.5]] (figure 1) [[132,12,127], [147,5,153], [172,12,173], [195,33,196], [203,63,209], [215,89,225], [229,117,230], [243,129,238], [253,155,247], [255,178,254], [255,255,255], [255,255,255], [126,240,138], [134,234,138], [95,219,89], [57,201,54], [39,182,57], [33,161,36], [16,139,22], [0,123,10], [6,96,6], [12,77,9.0]] (figure 8)

   - max_vert : 1 - 4 (user preference)

   - max_hori : 1 - 4 (user preference)

   - grid_color: blue4 (figure 8)

   - labelBar_end_type: ExcludeOuterBoxes (figure 1), both_triangle (figure 7, 8)

   - unitCorrectionalFactor: -3.154e+10 (figure 7)

   - new_units : "gC/ (m~S~2~N~ * yr)" (figure 7)

   Required settings (variables)

   - additional_dataset: datasets to plot.

   Optional settings (variables)

   - none

2. Script russell18jgr-fig2.ncl

   Required settings (scripts)

   - styleset : CMIP5(recommended), default, etc.

   - ncdf : default(recommended), CMIP5, etc.

   Optional settings (scripts)

   - none

3. Script russell18jgr-fig3b.ncl

   Required settings (scripts)

   - styleset : CMIP5(recommended), default, etc.

   - ncdf : default(recommended), CMIP5, etc.

   Optional settings (scripts)

   - none

4. Script russell18jgr-fig3b-2.ncl

   Required settings (scripts)

   - styleset : CMIP5(recommended), default, etc.

   - ncdf : default(recommended), CMIP5, etc.

   Optional settings (scripts)

   - none

5. Script russell18jgr-fig4.ncl

   Required settings (scripts)

   - styleset : CMIP5(recommended), default, etc.

   - ncdf : default(recommended), CMIP5, etc.

   Optional settings (scripts)

   - max_vert : 1 - 4 (user preference)

   - max_hori : 1 - 4 (user preference)

   - unitCorrectionalFactor: 100 (m/s to cm/s)

   - new_units : "cm/s"

6. Script russell18jgr-fig5.ncl

   Required settings (scripts)

   - styleset : CMIP5(recommended), default, etc.

   - ncdf : default(recommended), CMIP5, etc.

   - max_lat : -45.0

   Optional settings (scripts)

   - max_vert : 1 - 4 (user preference)

   - max_hori : 1 - 4 (user preference)

7. Script russell18jgr-fig5g.ncl

   Required settings (scripts)

   - styleset : CMIP5(recommended), default, etc.

   Optional settings (scripts)

   - none

8. Script russell18jgr-fig6a.ncl

   Required settings (scripts)

- styleset : CMIP5(recommended), default, etc.

- ncdf : default(recommended), CMIP5, etc.

Optional settings (scripts)

- none

9. Script russell18jgr-fig6b.ncl

Required settings (scripts)

- styleset : CMIP5(recommended), default, etc.

- ncdf : default(recommended), CMIP5, etc.

Optional settings (scripts)

- none

10. Script russell18jgr-fig7h.ncl

Required settings (scripts)

- styleset : CMIP5(recommended), default, etc.

- ncdf : default(recommended), CMIP5, etc.

Optional settings (scripts)

- none

11. Script russell18jgr-fig7i.ncl

Required settings (scripts)

- styleset : CMIP5(recommended), default, etc.

- ncdf : default(recommended), CMIP5, etc.

Optional settings (scripts)

- none

12. Script russell18jgr-fig9a.ncl

Required settings (scripts)

- styleset : CMIP5(recommended), default, etc.

- ncdf : default(recommended), CMIP5, etc.

Optional settings (scripts)

- none

13. Script russell18jgr-fig9b.ncl

Required settings (scripts)

- styleset : CMIP5(recommended), default, etc.

- ncdf : default(recommended), CMIP5, etc.

Optional settings (scripts)

- none

14. Script russell18jgr-fig9c.ncl

Required settings (scripts)

- styleset : CMIP5(recommended), default, etc.
- ncdf : default(recommended), CMIP5, etc.

Optional settings (scripts)

- none

13.5.4 Variables

- tauu (atmos, monthly mean, longitude latitude time)
- tauuo, hfds, fgco2 (ocean, monthly mean, longitude latitude time)
- thetao, so, vo (ocean, monthly mean, longitude latitude lev time)
- pH (ocnBgchem, monthly mean, longitude latitude time)
- uo (ocean, monthly mean, longitude latitude lev time)
- sic (seaIce, monthly mean, longitude latitude time)

13.5.5 Observations and reformat scripts

Note: WOA data has not been tested with reciepe_russell18jgr.yml and corresponding diagnostic scripts.

- WOA (thetao, so - **esmvaltool/utils/cmorizers/obs/cmorize_obs_woa.py**)

13.5.6 References

- Russell, J.L., et al., 2018, J. Geophys. Res. – Oceans, 123, 3120-3143. https://doi.org/10.1002/2017JC013461
- Talley, L.D., 2003. Shallow,intermediate and deep overturning components of the global heat budget. Journal of Physical Oceanography 33, 530–560

13.5.7 Example plots

CanESM2
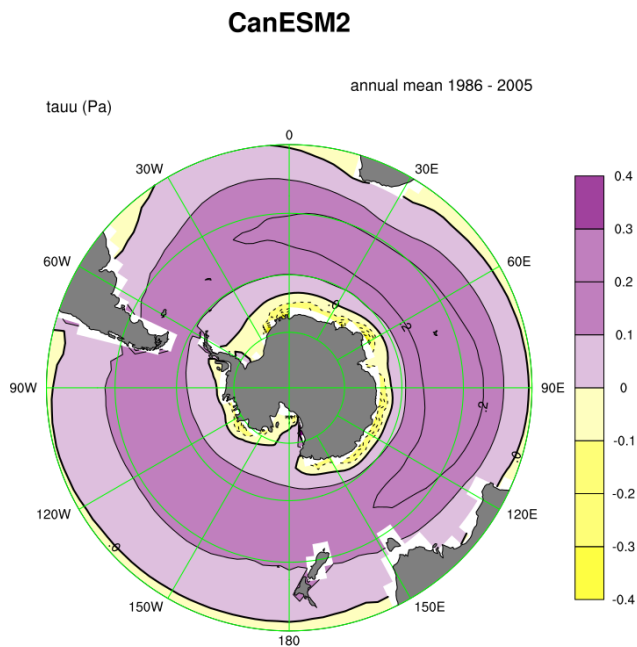
Fig. 9: Figure 1: Annual-mean zonal wind stress (tauu - N/m^2) with eastward wind stress as positive plotted as a polar contour map.
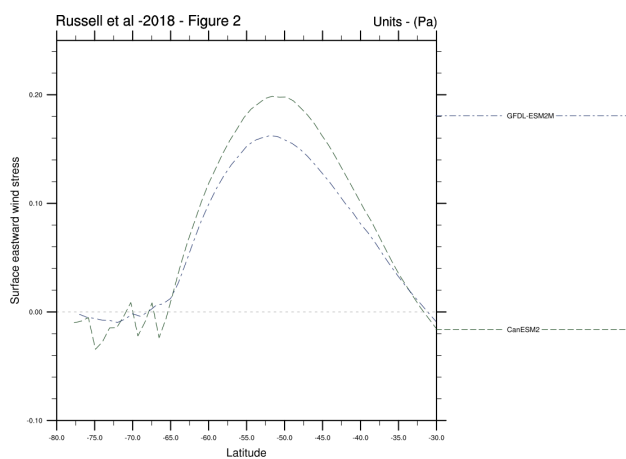
Fig. 10: Figure 2: The zonal and annual means of the zonal wind stress (N/m^2) plotted in a line plot.
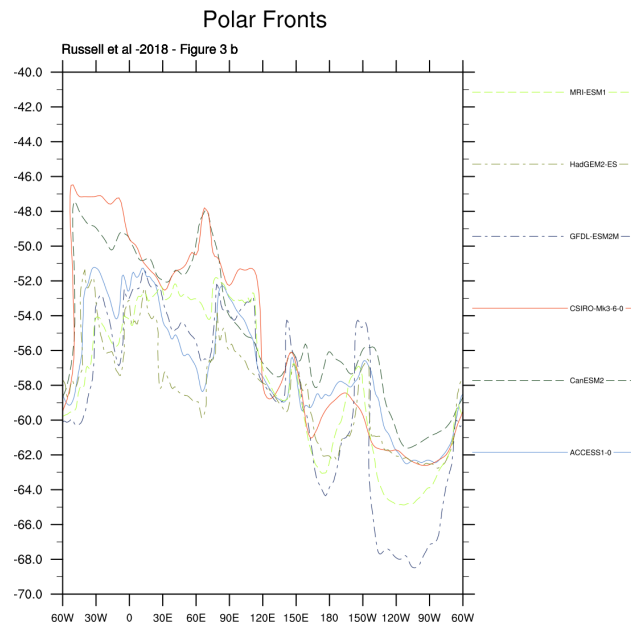
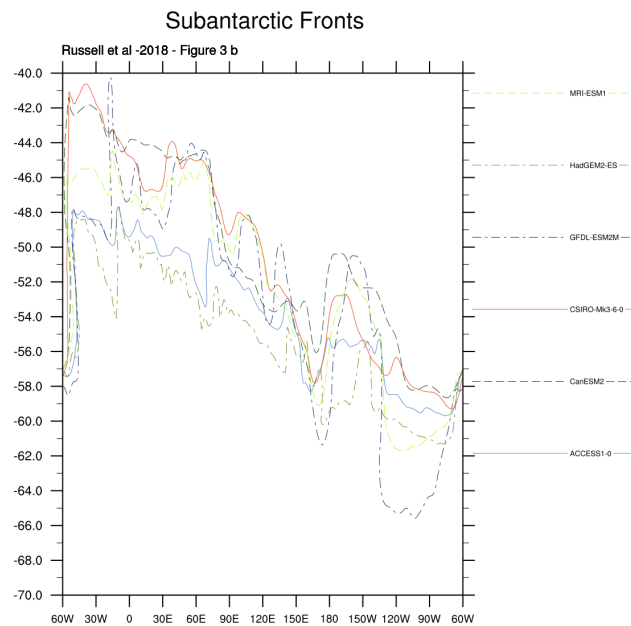Fig. 11: Figure 3a: The latitudinal position of Subantarctic Front using definitions from Orsi et al (1995).

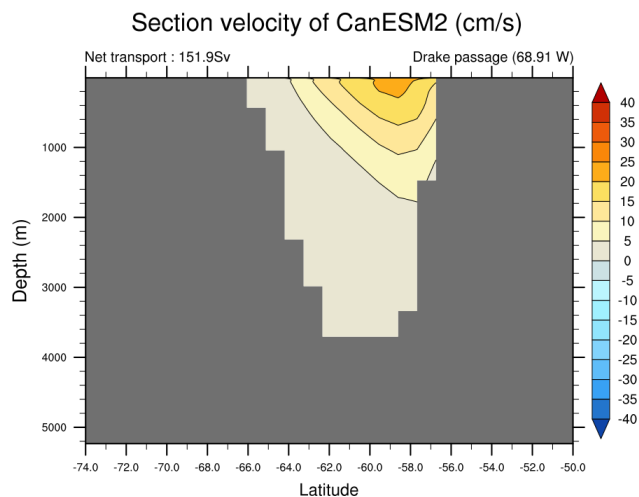Fig. 12: Figure 3b: The latitudinal position of Polar Front using definitions from Orsi et al (1995).

Fig. 13: Figure 4: Time averaged zonal velocity through Drake Passage (at 69W, in cm/s, eastward is positive). The total transport by the ACC is calculated if volcello file is available.
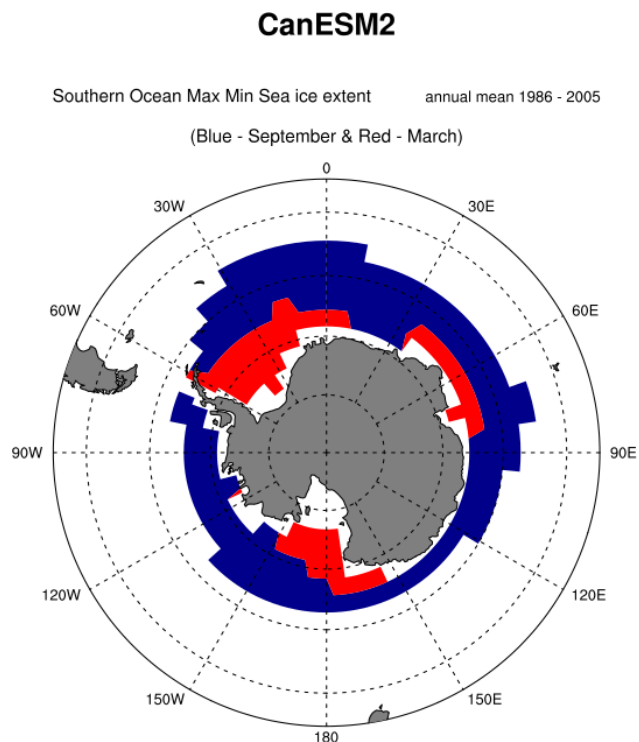
Fig. 14: Figure 5: Mean extent of sea ice for September(max) in blue and February(min) in red plotted as polar contour map.
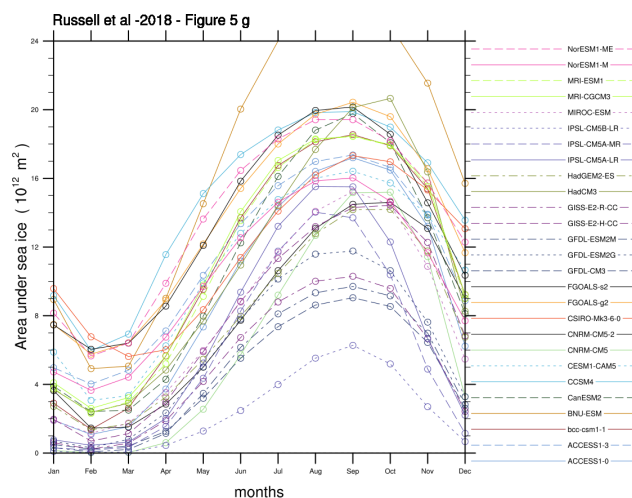
Fig. 15: Figure 5g: Annual cycle of sea ice area in southern ocean as a line plot (monthly climatology).
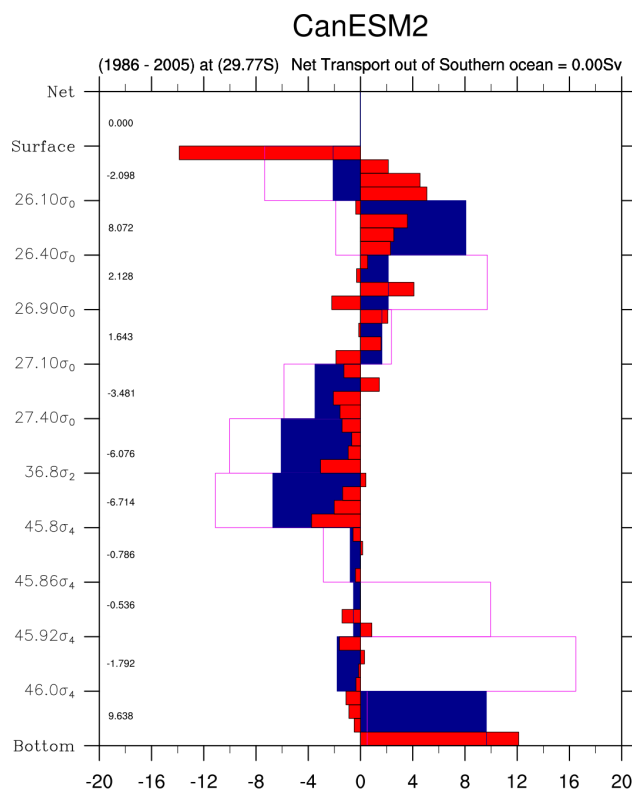

Fig. 16: Figure 6a: Density layer based volume transport (in Sv) across 30S based on the layer definitions in Talley (2008).
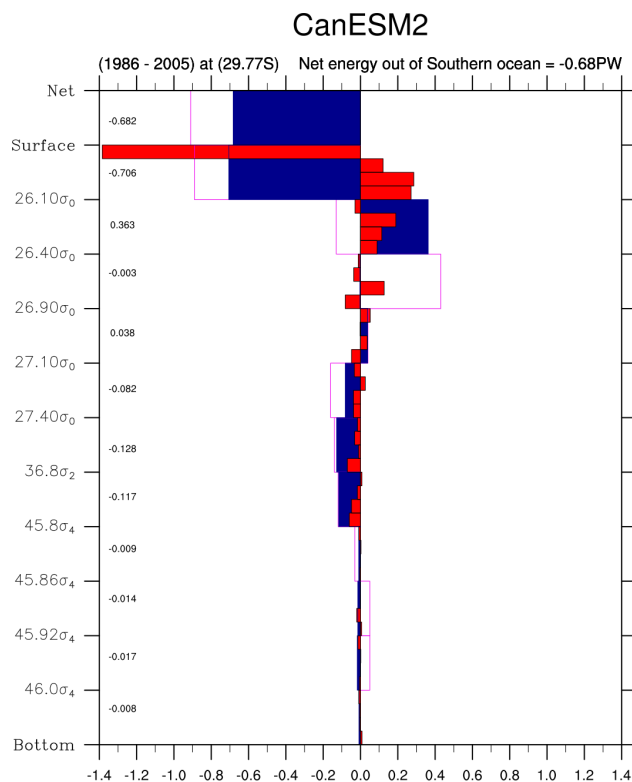
Fig. 17: Figure 6b: Density layer based heat transport(in PW) across 30S based on the layer definitions in Talley (2008).
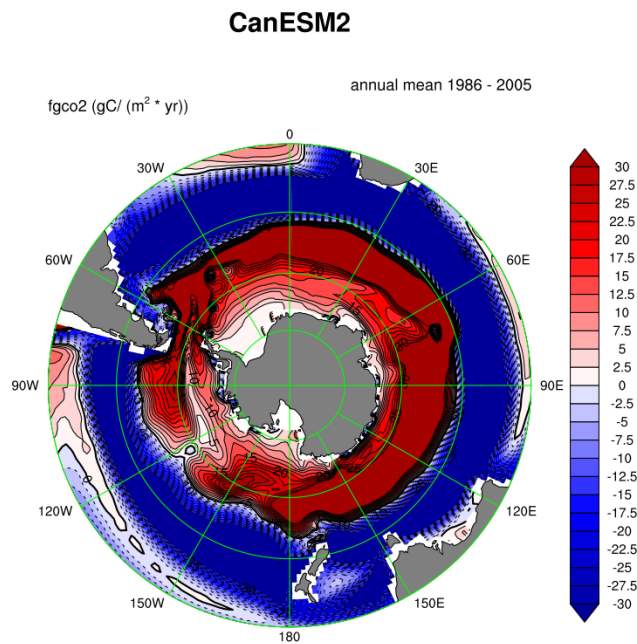

Fig. 18: Figure 7: Annual mean CO_2 flux (sea to air, gC/(yr * m^2), positive (red) is out of the ocean) as a polar contour map.
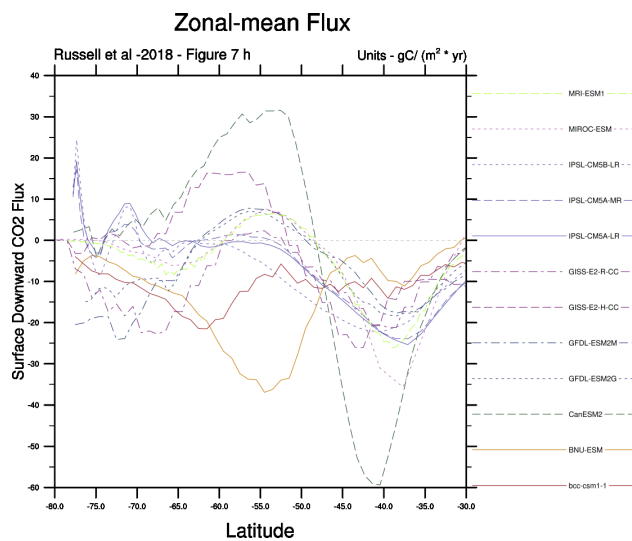
Fig. 19: Figure 7h: the time and zonal mean flux of CO_2 in gC/(yr * m^2) plotted as a line plot.
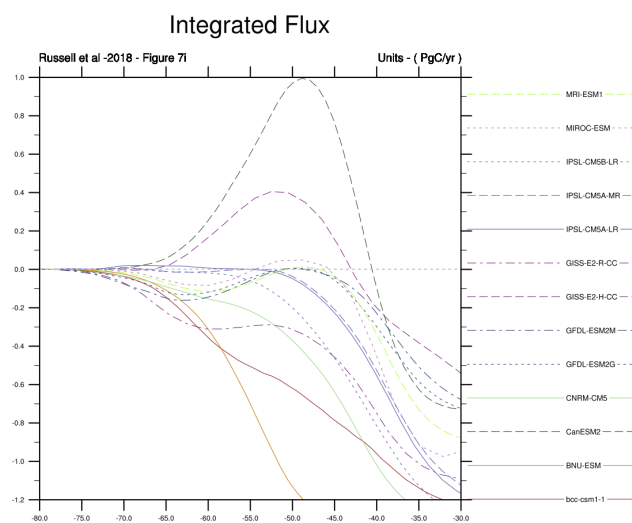

Fig. 20: Figure 7i is the cumulative integral of the net CO_2 flux from 90S to 30S (in PgC/yr) plotted as a line plot.
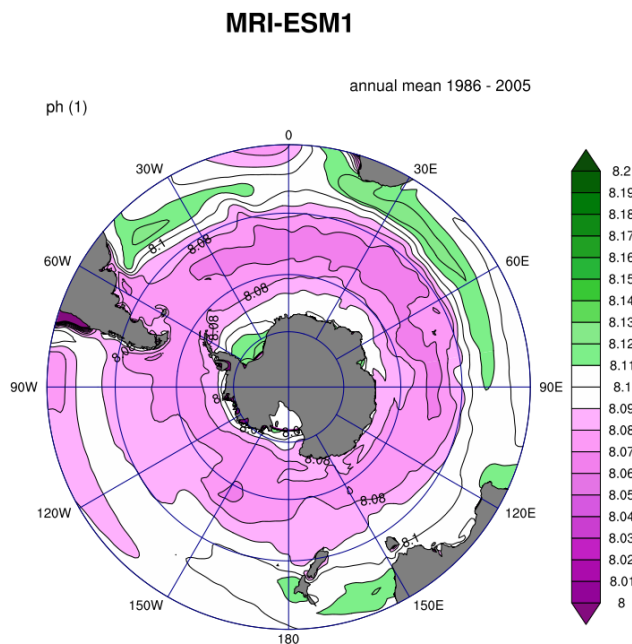
MRI-ESM1

Fig. 21: Figure 8: Annual-mean surface pH plotted as a polar contour map.

Fig. 22: Figure 9a: Scatter plot of the width of the Southern Hemisphere westerly wind band (in degrees of latitude) against the annual-mean integrated heat uptake south of 30S (in PW—negative uptake is heat lost from the ocean) along with the best fit line.
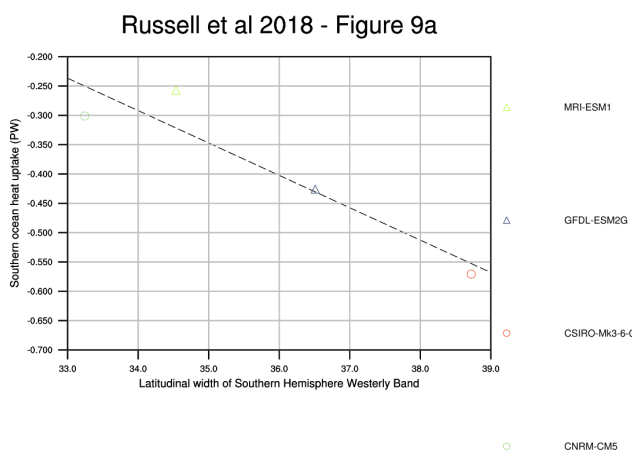
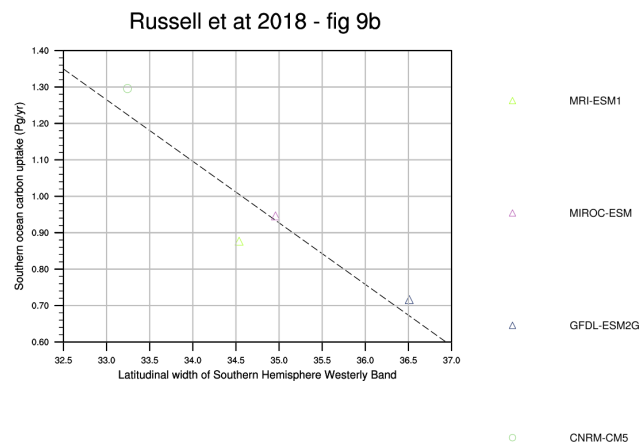Fig. 23: Figure 9b: Scatter plot of the width of the Southern Hemisphere westerly wind band (in degrees of latitude) against the annual-mean integrated carbon uptake south of 30S (in Pg C/yr), along with the best fit line.
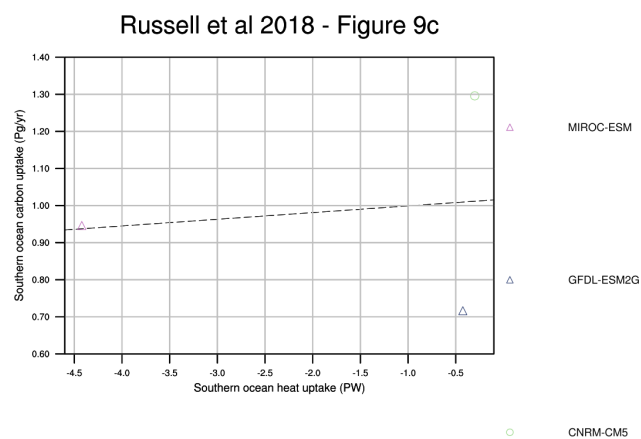

Fig. 24: Figure 9c: Scatter plot of the net heat uptake south of 30S (in PW) against the annual-mean integrated carbon uptake south of 30S (in Pg C/yr), along with the best fit line.

OTHER

14.1 Capacity factor of wind power: Ratio of average estimated power to theoretical maximum power

14.1.1 Overview

The goal of this diagnostic is to compute the wind capacity factor, taking as input the daily instantaneous surface wind speed, which is then extrapolated to obtain the wind speed at a height of 100 m as described in Lledó (2017).

The capacity factor is a normalized indicator of the suitability of wind speed conditions to produce electricity, irrespective of the size and number of installed turbines. This indicator is provided for three different classes of wind turbines (IEC, 2005) that are designed specifically for low, medium and high wind speed conditions.

The user can select the region, temporal range and season of interest.

The output of the recipe is a netcdf file containing the capacity factor for each of the three turbine classes.

14.1.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_capacity_factor.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- capacity_factor.R: calculates the capacity factor for the three turbine classes.
- PC.R: calculates the power curves for the three turbine classes.

14.1.3 User settings

User setting files are stored in recipes/

1. recipe_capacity_factor.yml

   Required settings for script

   - power_curves: (should not be changed)

14.1.4 Variables

- sfcWind (atmos, daily, longitude, latitude, time)

14.1.5 Observations and reformat scripts

Main features of the selected turbines:

Turbine name	Rotor diameter (m)	Rated power (MW)	Cut-in speed (m/s)	Rated speed (m/s)	Cut-out speed (m/s)
Enercon E70 2.3MW	70	2.3	2.0	16.0	25.0
Gamesa G80 2.0MW	80	2.0	4.0	17.0	25.0
Gamesa G87 2.0MW	87	2.0	4.0	16.0	25.0
Vestas V100 2.0MW	100	2.0	3.0	15.0	20.0
Vestas V110 2.0MW	110	2.0	3.0	11.5	20.0

14.1.6 References

- IEC. (2005). International Standard IEC 61400-1, third edition, International Electrotechnical Commission. https://webstore.iec.ch/preview/info_iec61400-1%7Bed3.0%7Den.pdf

- Lledó, L. (2017). Computing capacity factor. Technical note BSC-ESS-2017-001, Barcelona Supercomputing Center. Available online at https://earth.bsc.es/wiki/lib/exe/fetch.php?media=library:external:bsc-ess-2017-001-c4e_capacity_factor.pdf [last accessed 11 October 2018]

14.1.7 Example plots

Wind capacity factor for five turbines: Enercon E70 (top-left), Gamesa G80 (middle-top), Gamesa G87 (top-right), Vestas V100 (bottom-left) and Vestas V110 (middle-bottom) using the IPSL-CM5A-MR simulations for the r1p1i1 ensemble for the rcp8.5 scenario during the period 2021-2050.

14.2 Ensemble Clustering - a cluster analysis tool for climate model simulations (EnsClus)

14.2.1 Overview

EnsClus is a cluster analysis tool in Python, based on the k-means algorithm, for ensembles of climate model simulations.

Multi-model studies allow to investigate climate processes beyond the limitations of individual models by means of inter-comparison or averages of several members of an ensemble. With large ensembles, it is often an advantage to be able to group members according to similar characteristics and to select the most representative member for each cluster.

CF from IPSL-CM5A-MR (2021-2050)
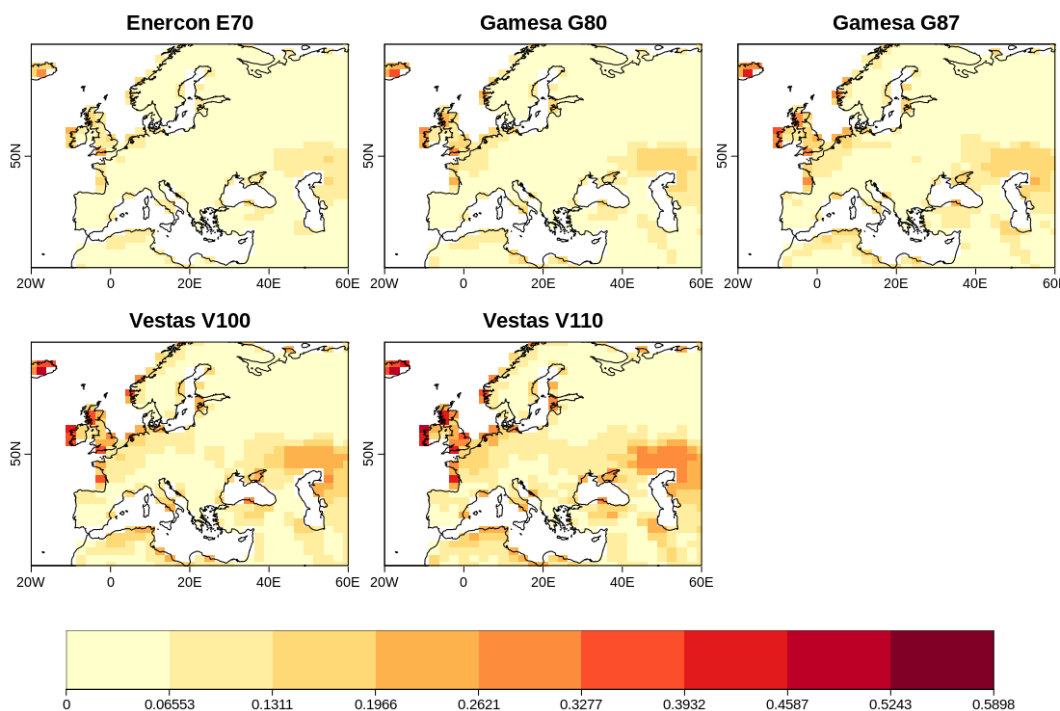
The user chooses which feature of the data is used to group the ensemble members by clustering: time mean, maximum, a certain percentile (e.g., 75% as in the examples below), standard deviation and trend over the time period. For each ensemble member this value is computed at each grid point, obtaining N lat-lon maps, where N is the number of ensemble members. The anomaly is computed subtracting the ensemble mean of these maps to each of the single maps. The anomaly is therefore computed with respect to the ensemble members (and not with respect to the time) and the Empirical Orthogonal Function (EOF) analysis is applied to these anomaly maps.

Regarding the EOF analysis, the user can choose either how many Principal Components (PCs) to retain or the percentage of explained variance to keep. After reducing dimensionality via EOF analysis, k-means analysis is applied using the desired subset of PCs.

The major final outputs are the classification in clusters, i.e. which member belongs to which cluster (in k-means analysis the number k of clusters needs to be defined prior to the analysis) and the most representative member for each cluster, which is the closest member to the cluster centroid.

Other outputs refer to the statistics of clustering: in the PC space, the minimum and the maximum distance between a member in a cluster and the cluster centroid (i.e. the closest and the furthest member), the intra-cluster standard deviation for each cluster (i.e. how much the cluster is compact).

14.2.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_ensclus.yml

Diagnostics are stored in diag_scripts/ensclus/

- ensclus.py

and subroutines

- ens_anom.py

- ens_eof_kmeans.py

- ens_plots.py

- eof_tool.py

- read_netcdf.py

- sel_season_area.py

14.2.3 User settings

Required settings for script

- season: season over which to perform seasonal averaging (DJF, DJFM, NDJFM, JJA)

- area: region of interest (EAT=Euro-Atlantic, PNA=Pacific North American, NH=Northern Hemisphere, EU=Europe)

- extreme: extreme to consider: XXth_percentile (XX can be set arbitrarily, e.g. 75th_percentile), mean (mean value over the period), maximum (maximum value over the period), std (standard deviation), trend (linear trend over the period)

- numclus: number of clusters to be computed

- perc: percentage of variance to be explained by PCs (select either this or numpcs, default=80)

- numpcs: number of PCs to retain (has priority over perc unless it is set to 0 (default))

Optional settings for script

- max_plot_panels: maximum number of panels (datasets) in a plot. When exceeded multiple plots are created. Default: 72

14.2.4 Variables

- chosen by user (e.g., precipitation as in the example)

14.2.5 Observations and reformat scripts

None.

14.2.6 References

- Straus, D. M., S. Corti, and F. Molteni: Circulation regimes: Chaotic variability vs. SST forced predictability. J. Climate, 20, 2251–2272, 2007. https://doi.org/10.1175/JCLI4070.1
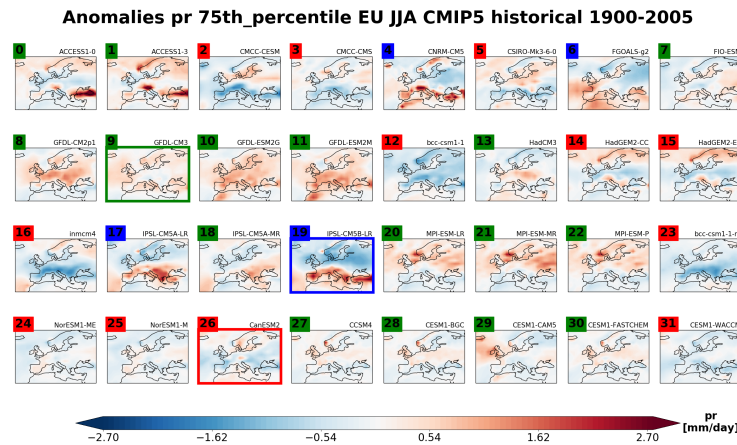
14.2.7 Example plots

Fig. 1: Clustering based on the 75th percentile of historical summer (JJA) precipitation rate for CMIP5 models over 1900-2005. 3 clusters are computed, based on the principal components explaining 80% of the variance. The 32 models are grouped in three different clusters. The green cluster is the most populated with 16 ensemble members mostly characterized by a positive anomaly over central-north Europe. The red cluster counts 12 elements that exhibit a negative anomaly centered over southern Europe. The third cluster – labelled in blue- includes only 4 models showing a north-south dipolar precipitation anomaly, with a wetter than average Mediterranean counteracting dryer North-Europe. Ensemble members No.9, No.26 and No.19 are the "specimen" of each cluster, i.e. the model simulations that better represent the main features of that cluster. These ensemble members can eventually be used as representative of the whole possible outcomes of the multi-model ensemble distribution associated to the 32 CMIP5 historical integrations for the summer precipitation rate 75 th percentile over Europe when these outcomes are reduced from 32 to 3. The number of ensemble members of each cluster might provide a measure of the probability of occurrence of each cluster.

14.3 Multi-model products

14.3.1 Overview

The goal of this diagnostic is to compute the multi-model ensemble mean for a set of models selected by the user for individual variables and different temporal resolutions (annual, seasonal, monthly).

After selecting the region (defined by the lowermost and uppermost longitudes and latitudes), the mean for the selected reference period is subtracted from the projections in order to obtain the anomalies for the desired period. In addition, the recipe computes the percentage of models agreeing on the sign of this anomaly, thus providing some indication on the robustness of the climate signal.

The output of the recipe consists of a colored map showing the time average of the multi-model mean anomaly and stippling to indicate locations where the percentage of models agreeing on the sign of the anomaly exceeds a threshold selected by the user. Furthermore, a time series of the area-weighted mean anomaly for the projections is plotted. For the plots, the user can select the length of the running window for temporal smoothing and choose to display the ensemble mean with a light shading to represent the spread of the ensemble or choose to display each individual models.

14.3.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_multimodel_products.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- multimodel_products.R - script for computing multimodel anomalies and their agreement.

14.3.3 User settings

User setting files are stored in recipes/

1. recipe_multimodel_products.yml

   Required settings for script

   - colorbar_lim: positive number specifying the range (-colorbar_lim … +colorbar_lim) of the colorbar (0 = automatic colorbar scaling)

   - moninf: integer specifying the first month of the seasonal mean period to be computed

   - monsup: integer specifying the last month of the seasonal mean period to be computed, if it's null the anomaly of month indicated in moninf will be computed

   - agreement_threshold: integer between 0 and 100 indicating the threshold in percent for the minimum agreement between models on the sign of the multi-model mean anomaly for the stipling to be plotted

   - running_mean: integer indictating the length of the window for the running mean to be computed

   - time_series_plot: Either single or maxmin (plot the individual or the mean with shading between the max and min).

14.3.4 Variables

- any Amon variable (atmos, monthly mean, longitude latitude time)

14.3.5 Observations and reformat scripts

None

14.3.6 References

- Hagedorn, R., Doblas-Reyes, F. J., Palmer, T. N., Nat E H Ag E D O R N, R. E., & Pa, T. N. (2005). The rationale behind the success of multi-model ensembles in seasonal forecasting-I. Basic concept, 57, 219–233. https://doi.org/10.3402/tellusa.v57i3.14657

- Weigel, A. P., Liniger, M. A., & Appenzeller, C. (2008). Can multi-model combination really enhance the prediction skill of probabilistic ensemble forecasts? Quarterly Journal of the Royal Meteorological Society, 134(630), 241–260. https://doi.org/10.1002/qj.210

14.3.7 Example plots
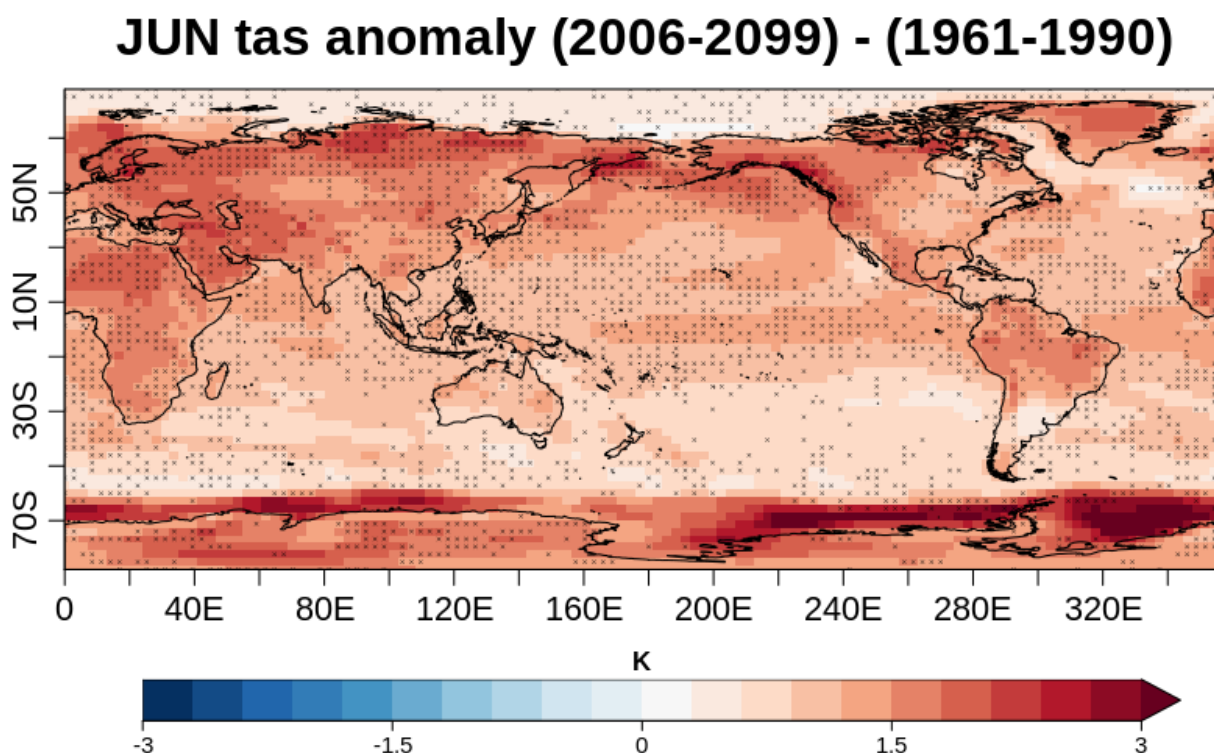
Multi-model mean anomaly of 2-m air temperature during the future projection 2006-2099 in June considering the reference period 1961-1990 (colours). Crosses indicate that the 80% of models agree in the sign of the multi-model mean anomaly. The models selected are BCC-CSM1-1, MPI-ESM-MR and MIROC5 in the r1i1p1 ensembles for the RCP 2.6 scenario.

14.4 RainFARM stochastic downscaling

14.4.1 Overview

Precipitation extremes and small-scale variability are essential drivers in many climate change impact studies. However, the spatial resolution currently achieved by global and regional climate models is still insufficient to correctly identify the fine structure of precipitation intensity fields. In the absence of a proper physically based representation, this scale gap can be at least temporarily bridged by adopting a stochastic rainfall downscaling technique (Rebora et al, 2006). With this aim, the Rainfall Filtered Autoregressive Model (RainFARM) was developed to apply the stochastic

precipitation downscaling method to climate models. The RainFARM Julia library and command-line tool version (https://github.com/jhardenberg/RainFARM.jl) was implemented as recipe. The stochastic method allows to predict climate variables at local scale from information simulated by climate models at regional scale: It first evaluates the statistical distribution of precipitation fields at regional scale and then applies the relationship to the boundary conditions of the climate model to produce synthetic fields at the requested higher resolution. RainFARM exploits the nonlinear transformation of a Gaussian random precipitation field, conserving the information present in the fields at larger scale (Rebora et al., 2006; D'Onofrio et al., 2014).

14.4.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_rainfarm.yml

Diagnostics are stored in diag_scripts/rainfarm/

- rainfarm.jl

14.4.3 User settings

Required settings for script

- slope: spatial spectral slope (set to 0 to compute automatically from large scales)

- nens: number of ensemble members to be calculated

- nf: number of subdivisions for downscaling (e.g. 8 will produce output fields with linear resolution increased by a factor 8)

- conserv_glob: logical, if to conserve precipitation over full domain

- conserv_smooth: logical, if to conserve precipitation using convolution (if neither conserv_glob or conserv_smooth is chosen, box conservation is used)

- weights_climo: set to false or omit if no orographic weights are to be used, else set it to the path to a fine-scale precipitation climatology file. If a relative file path is used, *auxiliary_data_dir* will be searched for this file. The file is expected to be in NetCDF format and should contain at least one precipitation field. If several fields at different times are provided, a climatology is derived by time averaging. Suitable climatology files could be for example a fine-scale precipitation climatology from a high-resolution regional climate model (see e.g. Terzago et al. 2018), a local high-resolution gridded climatology from observations, or a reconstruction such as those which can be downloaded from the WORLDCLIM (http://www.worldclim.org) or CHELSA (http://chelsa-climate.org) websites. The latter data will need to be converted to NetCDF format before being used (see for example the GDAL tools (https://www.gdal.org).

14.4.4 Variables

- pr (atmos, daily mean, longitude latitude time)

14.4.5 Observations and reformat scripts

None.

14.4.6 References

- Terzago et al. 2018, Nat. Hazards Earth Syst. Sci., 18, 2825-2840

- D'Onofrio et al. 2014, J of Hydrometeorology 15, 830-843

- Rebora et. al 2006, JHM 7, 724
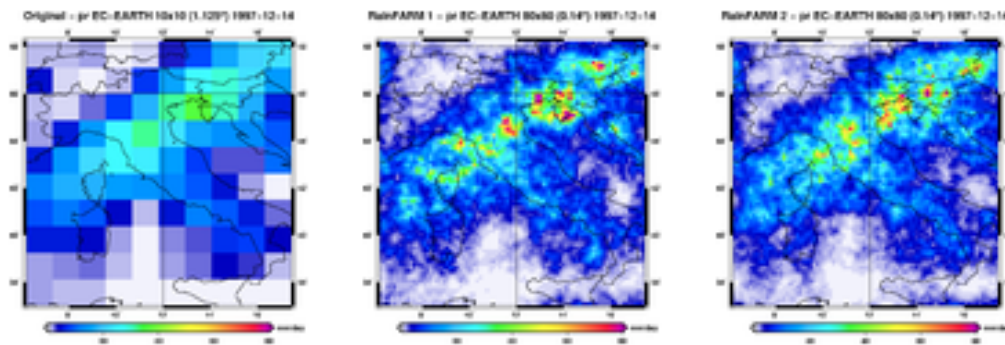
14.4.7 Example plots

Fig. 2: Example of daily cumulated precipitation from the CMIP5 EC-EARTH model on a specific day, downscaled using RainFARM from its original resolution (1.125°) (left panel), increasing spatial resolution by a factor of 8 to 0.14°; Two stochastic realizations are shown (central and right panel). A fixed spectral slope of s=1.7 was used. Notice how the downscaled fields introduce fine scale precipitation structures, while still maintaining on average the original coarse-resolution precipitation. Different stochastic realizations are shown to demonstrate how an ensemble of realizations can be used to reproduce unresolved subgrid variability. (N.B.: this plot was not produced by ESMValTool - the recipe output is netcdf only).

14.5 Seaice feedback

14.5.1 Overview

In this recipe, one process-based diagnostic named the Ice Formation Efficiency (IFE) is computed based on monthly mean sea-ice volume estimated north of 80°N. The choice of this domain is motivated by the desire to minimize the influence of dynamic processes but also by the availability of sea-ice thickness measurements. The diagnostic intends to evaluate the strength of the negative sea-ice thickness/growth feedback, which causes late-summer negative anomalies in sea-ice area and volume to be partially recovered during the next growing season. A chief cause behind the existence of this feedback is the non-linear inverse dependence between heat conduction fluxes and sea-ice thickness, which implies that thin sea ice grows faster than thick sea ice. To estimate the strength of that feedback, anomalies of the annual minimum of sea-ice volume north of 80°N are first estimated. Then, the increase in sea-ice volume until the next annual maximum is computed for each year. The IFE is defined as the regression of this ice volume production onto the baseline summer volume anomaly.

14.5.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_seaice_feedback.yml

Diagnostics are stored in diag_scripts/seaice_feedback/

- negative_seaice_feedback.py: scatterplot showing the feedback between seaice volume and seaice growth

14.5.3 User settings

script negative_seaice_feedback.py

> *Optional settings for script*
>
> - plot: dictionary containing plot options:
>   - point_color: color of the plot points. (Default: black)
>   - point_size: size of the plot points. (Default: 10)
>   - show_values: show numerical values of feedback in plot. (Default: True)

14.5.4 Variables

- sit (seaice, monthly mean, time latitude longitude)

14.5.5 References

- Massonnet, F., Vancoppenolle, M., Goosse, H., Docquier, D., Fichefet, T. and Blanchard-Wrigglesworth, E., 2018. Arctic sea-ice change tied to its mean state through thermodynamic processes. Nature Climate Change, 8: 599-603.

14.5.6 Example plots

14.6 Sea Ice

14.6.1 Overview

The sea ice diagnostics include:

(1) time series of Arctic and Antarctic sea ice area and extent (calculated as the total area (km^2) of grid cells with sea ice concentrations (sic) of at least 15%).

(2) ice extent trend distributions for the Arctic in September and the Antarctic in February.

(3) calculation of year of near disappearance of Arctic sea ice

(4) scatter plots of (a) historical trend in September Arctic sea ice extent (SSIE) vs historical long-term mean SSIE; (b) historical SSIE mean vs 1st year of disappearance (YOD) RCP8.5; (c) historical SSIE trend vs YOD RCP8.5.
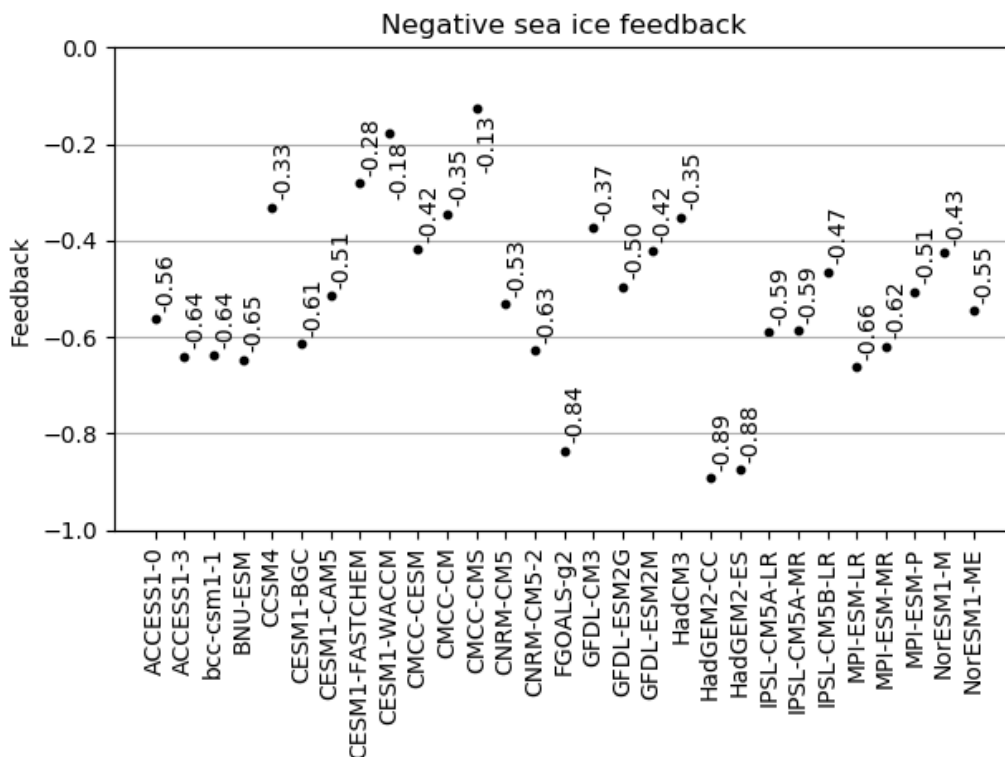
Fig. 3: Seaice negative feedback values (CMIP5 historical experiment 1979-2004).

14.6.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_seaice.yml

Diagnostics are stored in diag_scripts/seaice/

- seaice_aux.ncl: contains functions for calculating sea ice area or extent from sea ice concentration and first year of disappearance

- seaice_ecs.ncl: scatter plots of mean/trend of historical September Arctic sea ice extent vs 1st year of disappearance (RCP8.5) (similar to IPCC AR5 Chapter 12, Fig. 12.31a)

- seaice_trends.ncl: calculates ice extent trend distributions (similar to IPCC AR5 Chapter 9, Fig. 9.24c/d)

- seaice_tsline.ncl: creates a time series line plots of total sea ice area and extent (accumulated) for northern and southern hemispheres with optional multi-model mean and standard deviation. One value is used per model per year, either annual mean or the mean value of a selected month (similar to IPCC AR5 Chapter 9, Fig. 9.24a/b)

- seaice_yod.ncl: calculation of year of near disappearance of Arctic sea ice

14.6.3 User settings in recipe

1. Script seaice_ecs.ncl

   Required settings (scripts)

   - hist_exp: name of historical experiment (string)

   - month: selected month (1, 2, . . . , 12) or annual mean ("A")

   - rcp_exp: name of RCP experiment (string)

   - region: region to be analyzed ("Arctic" or "Antarctic")

   Optional settings (scripts)

   - fill_pole_hole: fill observational hole at North pole (default: False)

   - styleset: color style (e.g. "CMIP5")

   Optional settings (variables)

   - reference_dataset: reference dataset

2. Script seaice_trends.ncl

   Required settings (scripts)

   - month: selected month (1, 2, . . . , 12) or annual mean ("A")

   - region: region to be analyzed ("Arctic" or "Antarctic")

   Optional settings (scripts)

   - fill_pole_hole: fill observational hole at North pole, Default: False

   Optional settings (variables)

   - ref_model: array of references plotted as vertical lines

3. Script seaice_tsline.ncl

   Required settings (scripts)

   - region: Arctic, Antarctic

   - month: annual mean (A), or month number (3 = March, for Antarctic; 9 = September for Arctic)

   Optional settings (scripts)

   - styleset: for plot_type cycle only (cmip5, cmip6, default)

   - multi_model_mean: plot multi-model mean and standard deviation (default: False)

   - EMs_in_lg: create a legend label for individual ensemble members (default: False)

   - fill_pole_hole: fill polar hole (typically in satellite data) with sic = 1 (default: False)

4. Script seaice_yod.ncl

   Required settings (scripts)

   - month: selected month (1, 2, . . . , 12) or annual mean ("A")

   - region: region to be analyzed ("Arctic" or "Antarctic")

   Optional settings (scripts)

   - fill_pole_hole: fill observational hole at North pole, Default: False

   - wgt_file: netCDF containing pre-determined model weights

Optional settings (variables)

- ref_model: array of references plotted as vertical lines

14.6.4 Variables

- sic (ocean-ice, monthly mean, longitude latitude time)
- areacello (fx, longitude latitude)

14.6.5 Observations and reformat scripts

Note: (1) obs4mips data can be used directly without any preprocessing; (2) see headers of cmorization scripts (in esmvaltool/utils/cmorizers/obs) for non-obs4mips data for download instructions.

- HadISST (sic - esmvaltool/utils/cmorizers/obs/cmorize_obs_HadISST.ncl)

14.6.6 References

- Massonnet, F. et al., The Cryosphere, 6, 1383-1394, doi: 10.5194/tc-6-1383-2012, 2012.
- Stroeve, J. et al., Geophys. Res. Lett., 34, L09501, doi:10.1029/2007GL029703, 2007.
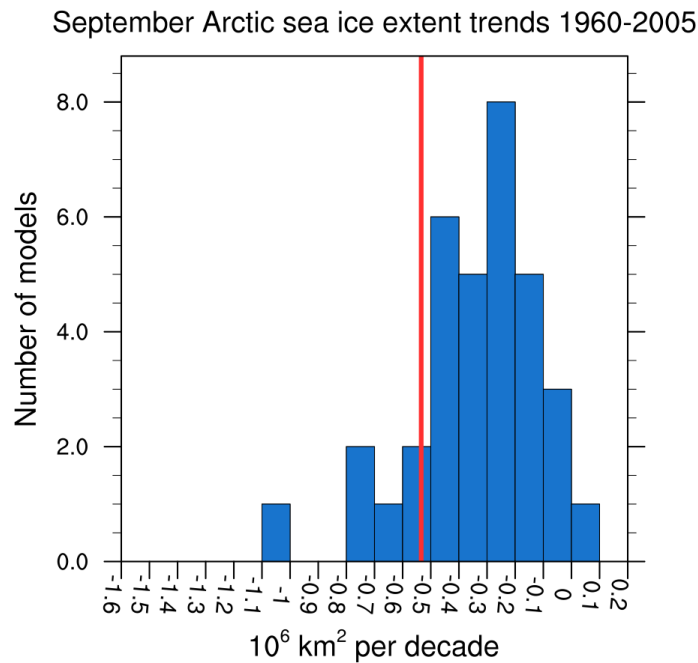
14.6.7 Example plots

Fig. 4: Sea ice extent trend distribution for the Arctic in September (similar to IPCC AR5 Chapter 9, Fig. 9.24c). [seaice_trends.ncl]
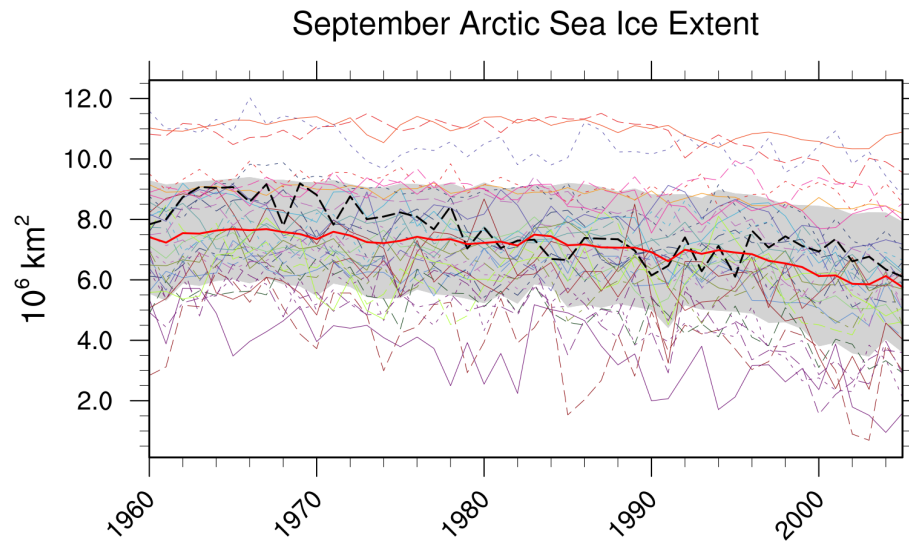
September Arctic Sea Ice Extent

Fig. 5: Time series of total sea ice area and extent (accumulated) for the Arctic in September including multi-model mean and standard deviation (similar to IPCC AR5 Chapter 9, Fig. 9.24a). [seaice_tsline.ncl]
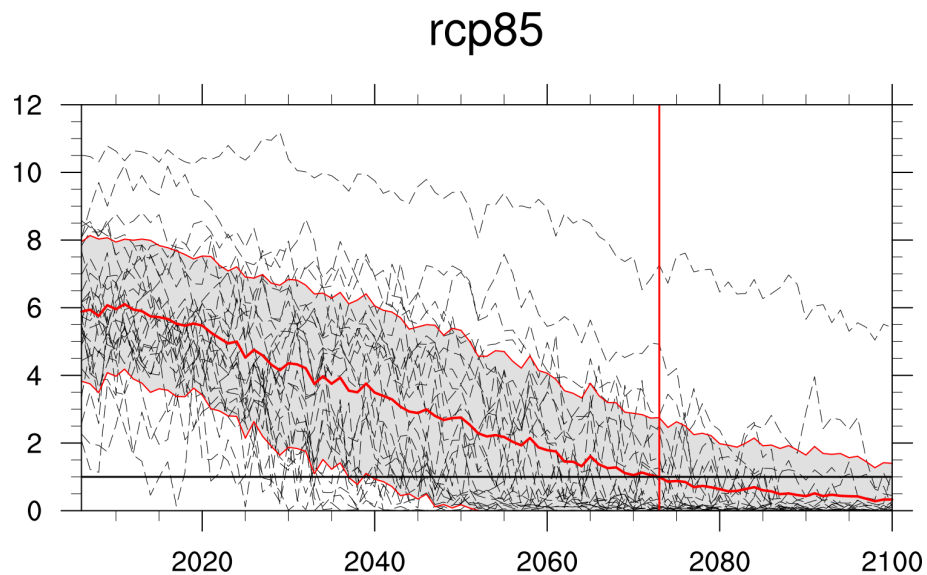
rcp85

Fig. 6: Time series of September Arctic sea ice extent for individual CMIP5 models, multi-model mean and multi-model standard deviation, year of disappearance (similar to IPCC AR5 Chapter 12, Fig. 12.31e). [seaice_yod.ncl]
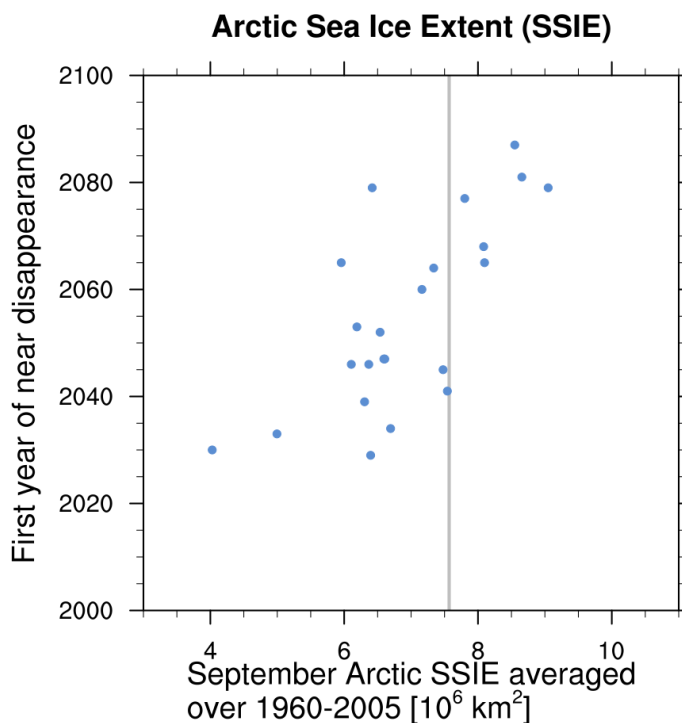
Fig. 7: Scatter plot of mean historical September Arctic sea ice extent vs 1st year of disappearance (RCP8.5) (similar to IPCC AR5 Chapter 12, Fig. 12.31a). [seaice_ecs.ncl]

14.7 Seaice drift

14.7.1 Overview

This recipe allows to quantify the relationships between Arctic sea-ice drift speed, concentration and thickness (Docquier et al., 2017). A decrease in concentration or thickness, as observed in recent decades in the Arctic Ocean (Kwok, 2018; Stroeve and Notz, 2018), leads to reduced sea-ice strength and internal stress, and thus larger sea-ice drift speed (Rampal et al., 2011). This in turn could provide higher export of sea ice out of the Arctic Basin, resulting in lower sea-ice concentration and further thinning. Olason and Notz (2014) investigate the relationships between Arctic sea-ice drift speed, concentration and thickness using satellite and buoy observations. They show that both seasonal and recent long-term changes in sea ice drift are primarily correlated to changes in sea ice concentration and thickness. This recipe allows to quantify these relationships in climate models.

In this recipe, four process-based metrics are computed based on the multi-year monthly mean sea-ice drift speed, concentration and thickness, averaged over the Central Arctic.

The first metric is the ratio between the modelled drift-concentration slope and the observed drift-concentration slope. The second metric is similar to the first one, except that sea-ice thickness is involved instead of sea-ice concentration. The third metric is the normalised distance between the model and observations in the drift-concentration space. The fourth metric is similar to the third one, except that sea-ice thickness is involved instead of sea-ice concentration.

14.7.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_seaice_drift.yml

Diagnostics are stored in diag_scripts/seaice_drift/

- seaice_drift.py: Compute metrics and plot results

14.7.3 User settings in recipe

1. Script diag_shapeselect.py

   Required settings (scripts)

   One of the following two combinations is required:

   1. Latitude threshold:

      - latitude_threshold: metric will be computed north of this latitude value

   2. Polygon:

      - polygon: metric will be computed inside the give polygon. Polygon is defined as a list of (lon, lat) tuple

      - polygon_name: name of the region defined by the polygon

14.7.4 Variables

- sispeed, sithick, siconc (daily)

14.7.5 Example plots
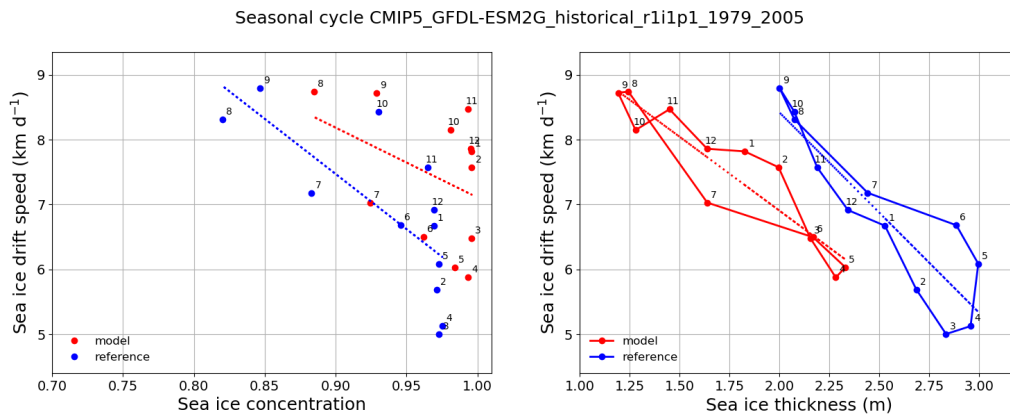
Scatter plots of modelled (red) and observed (blue) monthly mean sea-ice drift speed against sea-ice concentration (left panel) and sea-ice thickness (right panel) temporally averaged over the period 1979–2005 and spatially averaged over the SCICEX box.

14.8 Shapeselect

14.8.1 Overview

Impact modelers are often interested in data for irregular regions best defined by a shapefile. With the shapefile selector tool, the user can extract time series or CII data for a user defined region. The region is defined by a user provided shapefile that includes one or several polygons. For each polygon, a new timeseries, or CII, is produced with only one time series per polygon. The spatial information is reduced to a representative point for the polygon ('representative') or as an average of all grid points within the polygon boundaries ('mean_inside'). If there are no grid points strictly inside the polygon, the 'mean_inside' method defaults to 'representative' for that polygon. An option for displaying the grid points together with the shapefile polygon allows the user to assess which method is most optimal. In case interpolation to a high input grid is necessary, this can be provided in a pre-processing stage. Outputs are in the form of a NetCDF file, or as ascii code in csv format.

14.8.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_shapeselect.yml

Diagnostics are stored in diag_scripts/shapeselect/

- diag_shapeselect.py: calculate the average of grid points inside the user provided shapefile and returns the result as a NetCDF or Excel sheet.

14.8.3 User settings in recipe

1. Script diag_shapeselect.py

   Required settings (scripts)

   - shapefile: path to the user provided shapefile. A relative path is relative to the auxiliary_data_dir as configured in config-user.yml.

   - weighting_method: the preferred weighting method 'mean_inside' - mean of all grid points inside polygon; 'representative' - one point inside or close to the polygon is used to represent the complete area.

   - write_xlsx: true or false to write output as Excel sheet or not.

   - write_netcdf: true or false to write output as NetCDF or not.

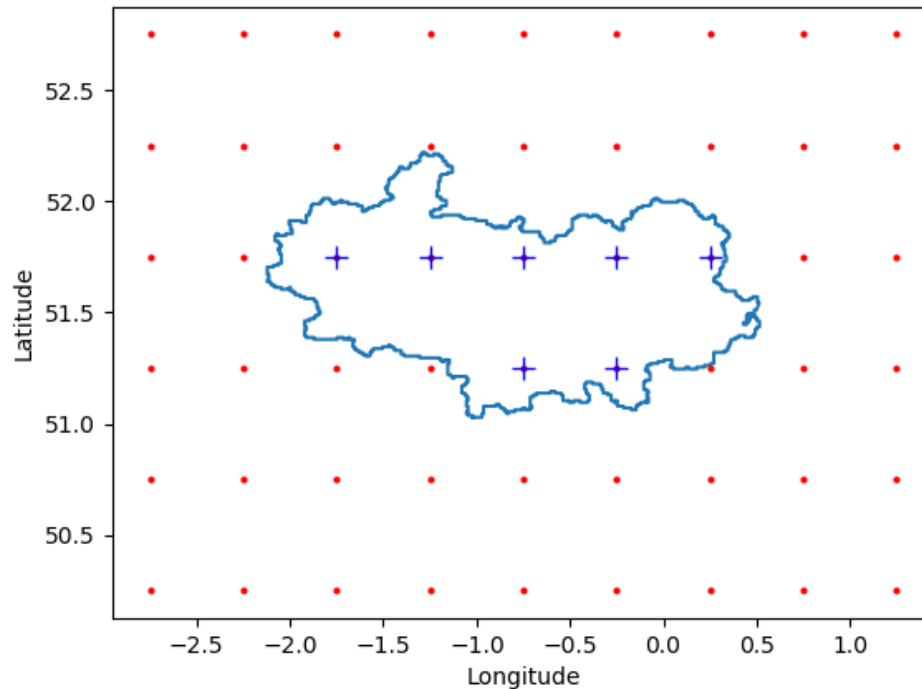### 14.8.4 Variables

- pr,tas (daily)

14.8.5 Example plots

Fig. 8: Example of the selection of model grid points falling within (blue pluses) and without (red dots) a provided shapefile (blue contour).

14.9 Toymodel

14.9.1 Overview

The goal of this diagnostic is to simulate single-model ensembles from an observational dataset to investigate the effect of observational uncertainty. For further discussion of this synthetic value generator, its general application to forecasts and its limitations, see Weigel et al. (2008). The output is a netcdf file containing the synthetic observations. Due to the sampling of the perturbations from a Gaussian distribution, running the recipe multiple times, with the same observation dataset and input parameters, will result in different outputs.

14.9.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_toymodel.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- toymodel.R: generates a single model ensemble of synthetic observations

14.9.3 User settings

User setting files are stored in recipes/

1. recipe_toymodel.yml

   Required settings for preprocessor

   > extract_region:

   - start_longitude: minimum longitude
   - end_longitude: maximum longitude
   - start_latitude: minimum longitude
   - end_latitude: maximum latitude

   > extract_levels: (for 3D variables)

   - levels: [50000] # e.g. for 500 hPa level

   Required settings for script

   - number_of_members: integer specifying the number of members to be generated
   - beta: the user defined underdispersion (beta >= 0)

14.9.4 Variables

- any variable (atmos/ocean, daily-monthly, longitude, latitude, time)

14.9.5 Observations and reformat scripts

None

14.9.6 References

- Bellprat, O., Massonnet, F., Siegert, S., Prodhomme, C., Macias-Gómez, D., Guemas, V., & Doblas-Reyes, F. (2017). Uncertainty propagation in observational references to climate model scales. Remote Sensing of Environment, 203, 101-108.

- Massonet, F., Bellprat, O. Guemas, V., & Doblas-Reyes, F. J. (2016). Using climate models to estimate the quality of global observational data sets. Science, aaf6369.

- Weigel, A. P., Liniger, M. A., & Appenzeller, C. (2008). Can multi-model combinations really enhance the prediction skill of probabilistic ensemble forecasts? Quarterly Journal of the Royal Meteorological Society, 134(630), 241-260.

14.9.7 Example plots

20 synthetic members generated
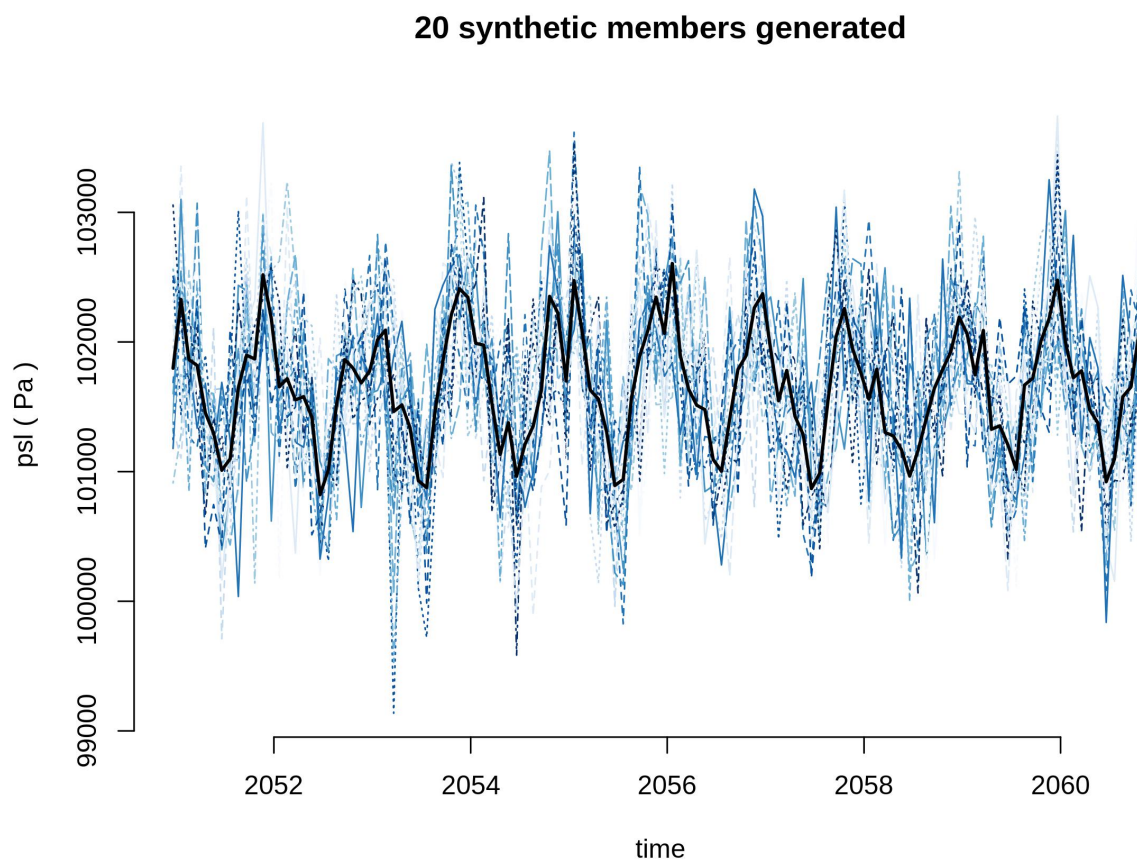
Twenty synthetic single-model ensemble generated by the recipe_toymodel.yml (see Section 3.7.2) for the 2051-2060 monthly data of r1i1p1 RCP 4.5 scenario of BCC_CSM1-1 simulation.

Part V

Obtaining input data

ESMValTool accepts input data from various models as well as observations and reanalysis data, provided that they adhere to the CF/CMOR format. This section provides some guidelines for unfamiliar users.

FIFTEEN

MODELS

ESMValTool will look for existing data in the directories specified in the user configuration file. Alternatively, it can use an external tool called Synda (http://prodiguer.github.io/synda/index.html). Here, we describe the basic steps to configure EMSValTool to work with Synda. This is the recommended approach for first-time users to quickly obtain some data for running ESMValTool.

To install Synda, follow the steps listed in the Synda documentation. This description assumes that you use the conda install. As the last step, Synda will ask to set your openID credentials. Therefore, you'll need to create an account at https://esgf-node.llnl.gov/projects/esgf-llnl/ and join a Data Access Control Group, e.g. CMIP5 Research. For more information, see https://esgf.github.io/esgf-user-support/user_guide.html.

Once you have set up Synda, you'll need to configure ESMValTool to recognize your Synda installation. Note that it is not possible to combine the two in a single conda environment, for Synda requires python 2 and ESMValTool requires Python 3. Typing `which synda` while your synda environment is active will print its location. To make the `synda` program usable from ESMValTool we suggest creating a directory `mkdir ~/bin` and and appending that folder to your PATH environment variable, e.g. by adding the following line to your `~/.bashrc` file: `PATH=$PATH:$HOME/bin`.

Finally, in the new bin folder, make a link to synda: `ln -s /path/to/conda/envs/synda/bin/synda ~/bin/synda`.

Now, ESMValTool should be able to recognize your Synda installation. First time users can now continue with *Running ESMValTool*.

OBSERVATIONS

Observational and reanalysis products in the standard CF/CMOR format used in CMIP and required by the ESM-ValTool are available via the obs4mips and ana4mips projects at the ESGF (e.g., https://esgf-data.dkrz.de/projects/esgf-dkrz/). Their use is strongly recommended, when possible.

Other datasets not available in these archives can be obtained by the user from the respective sources and reformatted to the CF/CMOR standard. ESMValTool currently support two ways to perform this reformatting (aka 'cmorization'). The first is to use a cmorizer script to generate a local pool of reformatted data that can readily be used by the ESMValTool. The second way is to implement specific 'fixes' for your dataset. In that case, the reformatting is performed 'on the fly' during the execution of an ESMValTool recipe (note that one of the first preprocessor tasks is 'cmor checks and fixes'). Below, both methods are explained in more detail.

16.1 Using a cmorizer script

ESMValTool comes with a set of cmorizers readily available. The cmorizers are dataset-specific scripts that can be run once to generate a local pool of CMOR-compliant data. The necessary information to download and process the data is provided in the header of each cmorizing script. These scripts also serve as template to create new cmorizers for datasets not yet included. Note that datasets cmorized for ESMValTool v1 may not be working with v2, due to the much stronger constraints on metadata set by the iris library.

To cmorize one or more datasets, run:

```
cmorize_obs -c [CONFIG_FILE] -o [DATASET_LIST]
```

The path to the raw data to be cmorized must be specified in the CONFIG_FILE as RAWOBS. Within this path, the data are expected to be organized in subdirectories corresponding to the data tier: Tier2 for freely-available datasets (other than obs4mips and ana4mips) and Tier3 for restricted datasets (i.e., dataset which requires a registration to be retrieved or provided upon request to the respective contact or PI). The cmorization follows the CMIP5 CMOR tables. The resulting output is saved in the output_dir, again following the Tier structure. The output file names follow the definition given in `config-developer.yml` for the `OBS` project: `OBS_[dataset]_[type]_[version]_[mip]_[short_name]_YYYYMM_YYYYMM.nc`, where `type` may be `sat` (satellite data), `reanaly` (reanalysis data), `ground` (ground observations), `clim` (derived climatologies), `campaign` (aircraft campaign).

At the moment, cmorize_obs supports Python and NCL scripts.

16.2 Cmorization as a fix

As of early 2020, ESMValTool also provides (limited) support for data in their native format. In this case, the steps needed to reformat the data are executed as datasets fixes during the execution of an ESMValTool recipe, as one of the first preprocessor steps. Compared to the workflow described above, this has the advantage that the user does not need to store a duplicate (cmorized) copy of the data. Instead, the cmorization is performed 'on the fly' when running a recipe. ERA5 is the first dataset for which this 'cmorization on the fly' is supported.

To use this functionality, users need to provide a path for the `native6` project data in the *user configuration file*. Then, in the recipe, they can refer to the native6 project, like so:

```
datasets:
- {dataset: ERA5, project: native6, type: reanaly, version: '1', tier: 3, start_year:
→1990, end_year: 1990}
```

Currently, the native6 project only supports ERA5 data in the format defined in the config-developer file. The filenames correspond to the default filenames from era5cli To support other datasets as well, we need to make it possible to have a dataset specific DRS. This is still on the horizon.

While it is not strictly necessary, it may still be useful in some cases to create a local pool of cmorized observations. This can be achieved by using a cmorizer *recipe*. For an example, see recipe_era5.yml. This recipe reads native, hourly ERA5 data, performs a daily aggregation preprocessor, and then calls a diagnostic that operates on the data. In this example, the diagnostic renames the data to the standard OBS6 format. The output are thus daily, cmorized ERA5 data, that can be used through the OBS6 project. As such, this example recipe does exactly the same as the cmorizer scripts described above: create a local pool of cmorized data. The advantage, in this case, is that the daily aggregation is performed only once, which can save a lot of time and compute if it is used often.

The example cmorizer recipe can be run like any other ESMValTool recipe:

```
esmvaltool -c [CONFIG_FILE] cmorizers/recipe_era5.yml
```

(Note that the `recipe_era5.yml` adds the next day of the new year to the input data. This is because one of the fixes needed for the ERA5 data is to shift (some of) the data half an hour back in time, resulting in a missing record on the last day of the year.)

To add support for new variables using this method, one needs to add dataset-specific fixes to the ESMValCore. For more information about fixes, see: fixing data.

16.3 Supported datasets

A list of the datasets for which a cmorizers is available is provided in the following table.

Dataset	Variables (MIP)	Tier	Script language
APHRO-MA	pr, tas (day), pr, tas (Amon)	3	Python
AURA-TES	tro3 (Amon)	3	NCL
BerkelyEarth	tas, tasa (Amon), sftlf (fx)	2	Python
CALIPSO-GOCCP	clcalipso (cfMon)	2	NCL
CDS-SATELLITE-ALBEDO	bdalb (Lmon), bhalb (Lmon)	3	Python
CDS-SATELLITE-LAI-FAPAR	fapar (Lmon), lai (Lmon)	3	Python
CDS-SATELLITE-SOIL-MOISTURE	sm (day), sm (Lmon)	3	NCL

continues on next page

Table 1 – continued from previous page

Dataset	Variables (MIP)	Tier	Script language
CDS-UERRA	sm (E6hr)	3	Python
CDS-XCH4	xch4 (Amon)	3	NCL
CDS-XCO2	xco2 (Amon)	3	NCL
CERES-EBAF	rlut, rlutcs, rsut, rsutcs (Amon)	2	Python
CERES-SYN1deg	rlds, rldscs, rlus, rluscs, rlut, rlutcs, rsds, rsdscs, rsus, rsuscs, rsut, rsutcs (3hr) rlds, rldscs, rlus, rlut, rlutcs, rsds, rsdt, rsus, rsut, rsutcs (Amon)	3	NCL
CowtanWay	tasa (Amon)	2	Python
CRU	tas, pr (Amon)	2	Python
CT2019	co2s (Amon)	2	Python
Duveiller2018	albDiffiTr13	2	Python
E-OBS	tas, tasmin, tasmax, pr, psl (day, Amon)	2	Python
Eppley-VGPM-MODIS	intpp (Omon)	2	Python
ERA5*[0]	clt, evspsbl, evspsblpot, mrro, pr, prsn, ps, psl, ptype, rls, rlds, rsds, rsdt, rss, uas, vas, tas, tasmax, tasmin, tdps, ts, tsn (E1hr/Amon), orog (fx)	3	n/a
ERA-Interim	clivi, clt, clwvi, evspsbl, hur, hus, pr, prsn, prw, ps, psl, rlds, rsds, rsdt, ta, tas, tauu, tauv, ts, ua, uas, va, vas, wap, zg (Amon), ps, rsdt (CFday), clt, pr, prsn, psl, rsds, rss, ta, tas, tasmax, tasmin, uas, va, vas, zg (day), evspsbl, tdps, ts, tsn, rss, tdps (Eday), tsn (LImon), hfds, tos (Omon), orog, sftlf (fx)	3	Python
ERA-Interim-Land	sm (Lmon)	3	Python
ESACCI-AEROSOL	abs550aer, od550aer, od550aerStderr, od550lt1aer, od870aer, od870aerStderr (aero)	2	NCL
ESACCI-CLOUD	clivi, clt, cltStderr, clwvi (Amon)	2	NCL
ESACCI-FIRE	burntArea (Lmon)	2	NCL
ESACCI-LANDCOVER	baresoilFrac, cropFrac, grassFrac, shrubFrac, treeFrac (Lmon)	2	NCL
ESACCI-OC	chl (Omon)	2	Python
ESACCI-OZONE	toz, tozStderr, tro3prof, tro3profStderr (Amon)	2	NCL
ESACCI-SOILMOISTURE	dos, dosStderr, sm, smStderr (Lmon)	2	NCL
ESACCI-SST	ts, tsStderr (Amon)	2	NCL
ESRL	co2s (Amon)	2	NCL
FLUXCOM	gpp (Lmon)	3	Python
GCP	fgco2 (Omon), nbp (Lmon)	2	Python
GHCN	pr (Amon)	2	NCL
GHCN-CAMS	tas (Amon)	2	Python
GISTEMP	tasa (Amon)	2	Python
GPCC	pr (Amon)	2	Python
HadCRUT3	tas, tasa (Amon)	2	NCL
HadCRUT4	tas, tasa (Amon)	2	NCL

Table 1 – continued from previous page

Dataset	Variables (MIP)	Tier	Script language
HadISST	sic (OImon), tos (Omon), ts (Amon)	2	NCL
HALOE	tro3, hus (Amon)	2	NCL
HWSD	cSoil (Lmon), areacella (fx), sftlf (fx)	3	Python
ISCCP-FH	alb, prw, ps, rlds, rlus, rlut, rlutcs, rsds, rsdt, rsus, rsut, rsutcs, tas, ts (Amon)	2	NCL
JMA-TRANSCOM	nbp (Lmon), fgco2 (Omon)	3	Python
LAI3g	lai (Lmon)	3	Python
LandFlux-EVAL	et, etStderr (Lmon)	3	Python
Landschuetzer2016	dpco2, fgco2, spco2 (Omon)	2	Python
MAC-LWP	lwp, lwpStderr (Amon)	3	NCL
MERRA2	sm (Lmon)	3	Python
MLS-AURA	hur, hurStderr (day)	3	Python
MODIS	cliwi, clt, clwvi, iwpStderr, lwpStderr (Amon), od550aer (aero)	3	NCL
MTE	gpp, gppStderr (Lmon)	3	Python
NCEP	hur, hus, pr, ta, tas, ua, va, wap, zg (Amon) pr, rlut, ua, va (day)	2	NCL
NDP	cVeg (Lmon)	3	Python
NIWA-BS	toz, tozStderr (Amon)	3	NCL
NSIDC-0116-[nh|sh]	usi, vsi (day)	3	Python
OSI-450-[nh|sh]	sic (OImon), sic (day)	2	Python
PATMOS-x	clt (Amon)	2	NCL
PERSIANN-CDR	pr (Amon), pr (day)	2	Python
PHC	thetao, so	2	Python
PIOMAS	sit (day)	2	Python
REGEN	pr (day, Amon)	2	Python
UWisc	clwvi, lwpStderr (Amon)	3	NCL
WOA	no3, o2, po4, si (Oyr), so, thetao (Omon)	2	Python

---

[0] ERA5 cmorization is built into ESMValTool through the native6 project, so there is no separate cmorizer script.

Part VI

Making a recipe or diagnostic

INTRODUCTION

This chapter contains instructions for developing your own recipes and/or diagnostics. It also contains a section describing how to use additional datasets with ESMValTool. While it is possible to use just the ESMValCore package and run any recipes/diagnostics you develop with just this package, it is highly recommended that you consider contributing the work you do back to the ESMValTool community. Among the advantages of contributing to the community are improved visibility of your work and support by the community with making and maintaining your diagnostic. See the *Community* chapter for a guide on how to contribute to the community.

RECIPE

18.1 Writing a basic recipe

The user will need to write a basic recipe to be able to run their own personal diagnostic. An example of such a recipe is found in *esmvaltool/recipes/recipe_my_personal_diagnostic.yml*. For general guidelines with regards to ESMValTool recipes please consult the User Guide; the specific parameters needed by a recipe that runs a personal diagnostic are:

```
scripts:
  my_diagnostic:
    script: /path/to/your/my_little_diagnostic.py
```

i.e. the full path to the personal diagnostic that the user needs to run.

DIAGNOSTIC

19.1 Instructions for personal diagnostic

Anyone can run a personal diagnostic, no matter where the location of it; there is no need to install esmvaltool in developer mode nor is it to git push or for that matter, do any git operations; the example recipe

```
esmvaltool/recipes/recipe_my_personal_diagnostic.yml
```

shows the use of running a personal diagnostic; the example

```
esmvaltool/diag_scripts/examples/my_little_diagnostic.py
```

and any of its alterations may be used as training wheels for the future ESMValTool diagnostic developer. The purpose of this example is to familiarize the user with the framework of ESMValTool without the constraints of installing and running the tool as developer.

19.2 Functionality

my_little_diagnostic (or whatever the user will call their diagnostic) makes full use of ESMValTool's preprocessor output (both phyisical files and run variables); this output comes in form of a nested dictionary, or config dictionary, see an example below; it also makes full use of the ability to call any of the preprocessor's functions, note that relative imports of modules from the esmvaltool package are allowed and work without altering the $PYTHONPATH.

The user may parse this dictionary so that they execute a number of operations on the preprocessed data; for example the *my_little_diagnostic.plot_time_series* grabs the preprocessed data output, computes global area averages for each model, then plots a time-series for each model. Different manipulation functionalities for grouping, sorting etc of the data in the config dictionary are available, please consult ESMValTool User Manual.

19.3 Example of config dictionary

To be added (use python-style code-block).

CONTRIBUTING A CMORIZING SCRIPT FOR AN ADDITIONAL DATASET

ESMValTool is designed to work with CF compliant data and follows the CMOR tables from the CMIP data request, therefore the observational datasets need to be CMORized for usage in ESMValTool. The following steps are necessary to prepare an observational data set for the use in ESMValTool.

Note: CMORization as a fix. As of early 2020, we've started implementing cmorization as *fixes*. As compared to the workflow described below, this has the advantage that the user does not need to store a duplicate (CMORized) copy of the data. Instead, the CMORization is performed 'on the fly' when running a recipe. **ERA5** is the first dataset for which this 'CMORization on the fly' is supported. For more information, see: *Cmorization as a fix*.

20.1 1. Check if your variable is CMOR standard

Most variables are defined in the CMIP data request and can be found in the CMOR tables in the folder /esmval-core/cmor/tables/cmip6/Tables/, differentiated according to the `MIP` they belong to. The tables are a copy of the PCMDI guidelines. If you find the variable in one of these tables, you can proceed to the next section.

If your variable is not available in the standard CMOR tables, you need to write a custom CMOR table entry for the variable as outlined below and add it to /esmvalcore/cmor/tables/custom/.

To create a new custom CMOR table you need to follow these guidelines:

- Provide the `variable_entry`;

- Provide the `modeling_realm`;

- Provide the variable attributes, but leave `standard_name` blank. Necessary variable attributes are: `units`, `cell_methods`, `cell_measures`, `long_name`, `comment`.

- Provide some additional variable attributes. Necessary additional variable attributes are: `dimensions`, `out_name`, `type`. There are also additional variable attributes that can be defined here (see the already available cmorizers).

It is recommended to use an existing custom table as a template, to edit the content and save it as `CMOR_<short_name>.dat`.

20.2 2. Edit your configuration file

Make sure that beside the paths to the model simulations and observations, also the path to raw observational data to be cmorized (`RAWOBS`) is present in your configuration file.

20.3 3. Store your dataset in the right place

The folder `RAWOBS` needs the subdirectories `Tier1`, `Tier2` and `Tier3`. The different tiers describe the different levels of restrictions for downloading (e.g. providing contact information, licence agreements) and using the observations. The unformatted (raw) observations should then be stored then in the appropriate of these three folders.

20.4 4. Create a cmorizer for the dataset

There are many cmorizing scripts available in /esmvaltool/cmorizers/obs/ where solutions to many kinds of format issues with observational data are addressed. Most of these scripts are written in NCL at the moment, but more and more examples for Python-based cmorizing scripts become available.

Note: NCL support will terminate soon, so new cmorizer scripts should preferably be written in Python.

How much cmorizing an observational data set needs is strongly dependent on the original NetCDF file and how close the original formatting already is to the strict CMOR standard.

In the following two subsections two cmorizing scripts, one written in Python and one written in NCL, are explained in more detail.

20.4.1 4.1 Cmorizer script written in python

Find here an example of a cmorizing script, written for the `MTE` dataset that is available at the MPI for Biogeochemistry in Jena: cmorize_obs_mte.py.

All the necessary information about the dataset to write the filename correctly, and which variable is of interest, is stored in a seperate configuration file: MTE.yml in the directory `ESMValTool/esmvaltool/cmorizers/obs/cmor_config/`. Note that the name of this configuration file has to be identical to the name of your data set. It is recommended that you set `project` to `OBS6` in the configuration file. That way, the variables defined in the CMIP6 CMOR table, augmented with the custom variables described above, are available to your script.

The first part of this configuration file defines the filename of the raw observations file, the second part defines the common global attributes for the cmorizer output, e.g. information that is needed to piece together the final observations file name in the correct structure (see Section *6. Naming convention of the observational data files*). The third part defines the variables that are supposed to be cmorized.

The actual cmorizing script `cmorize_obs_mte.py` consists of a header with information on where and how to download the data, and noting the last access of the data webpage.

The main body of the CMORizer script must contain a function called

```
def cmorization(in_dir, out_dir, cfg, config_user):
```

with this exact call signature. Here, `in_dir` corresponds to the input directory of the raw files, `out_dir` to the output directory of final reformatted data set and `cfg` to the configuration dictionary given by the `.yml` configuration file. The return value of this function is ignored. All the work, i.e. loading of the raw files, processing them and saving the final output, has to be performed inside its body. To simplify this process, ESMValTool provides a set of predefined utilities.py, which can be imported into your CMORizer by

```
from . import utilities as utils
```

Apart from a function to easily save data, this module contains different kinds of small fixes to the data attributes, coordinates, and metadata which are necessary for the data field to be CMOR-compliant.

Note that this specific CMORizer script contains several subroutines in order to make the code clearer and more readable (we strongly recommend to follow that code style). For example, the function `_get_filepath` converts the raw filepath to the correct one and the function `_extract_variable` extracts and saves a single variable from the raw data.

20.4.2 4.2 Cmorizer script written in NCL

Find here an example of a cmorizing script, written for the `ESACCI XCH4` dataset that is available on the Copernicus Climate Data Store: cmorize_obs_cds_xch4.ncl.

The first part of the script collects all the information about the dataset that are necessary to write the filename correctly and to understand which variable is of interest here. Please make sure to provide the correct information for following key words: DIAG_SCRIPT, VAR, NAME, MIP, FREQ, CMOR_TABLE.

- **Note:** the fields `VAR`, `NAME`, `MIP` and `FREQ` all ask for one or more entries. If more than one entry is provided, make sure that the order of the entries is the same for all four fields! (for example, that the first entry in all four fields describe the variable `xch4` that you would like to extract);

- **Note:** some functions in the script are NCL-specific and are available through the loading of the script interface.ncl. There are similar functions available for python scripts.

In the second part of the script each variable defined in `VAR` is separately extracted from the original data file and processed. Most parts of the code are commented, and therefore it should be easy to follow. ESMValTool provides a set of predefined utilities.ncl, which can be imported into your CMORizer by

```
loadscript(getenv("esmvaltool_root") + "/esmvaltool/cmorizers/obs/utilities.ncl")
```

This module contains different kinds of small fixes to the data attributes, coordinates, and metadata which are necessary for the data field to be CMOR-compliant.

20.5 5. Run the cmorizing script

The cmorizing script for the given dataset can be run with:

```
cmorize_obs -c <config-user.yml> -o <dataset-name>
```

Note: The output path given in the configuration file is the path where your cmorized dataset will be stored. The ESMValTool will create a folder with the correct tier information (see Section *2. Edit your configuration file*) if that tier folder is not already available, and then a folder named after the data set. In this folder the cmorized data set will be stored as a netCDF file.

If your run was successful, one or more NetCDF files are produced in your output directory.

20.6 6. Naming convention of the observational data files

For the ESMValTool to be able to read the observations from the NetCDF file, the file name needs a very specific structure and order of information parts (very similar to the naming convention for observations in ESMValTool v1.0). The file name will be automatically correctly created if a cmorizing script has been used to create the netCDF file.

The correct structure of an observational data set is defined in config-developer.yml, and looks like the following:

```
OBS_[dataset]_[type]_[version]_[mip]_[short_name]_YYYYMM-YYYYMM.nc
```

For the example of the `CDS-XCH4` data set, the correct structure of the file name looks then like this:

```
OBS_CDS-XCH4_sat_L3_Amon_xch4_200301-201612.nc
```

The different parts of the name are explained in more detail here:

- OBS: describes what kind of data can be expected in the file, in this case `observations`;
- CDS-XCH4: that is the name of the dataset. It has been named this way for illustration purposes (so that everybody understands it is the xch4 dataset downloaded from the CDS), but a better name would indeed be `ESACCI-XCH4` since it is a ESA-CCI dataset;
- sat: describes the source of the data, here we are looking at satellite data (therefore `sat`), could also be `reanaly` for reanalyses;
- L3: describes the version of the dataset:
- Amon: is the information in which `mip` the variable is to be expected, and what kind of temporal resolution it has; here we expect `xch4` to be part of the atmosphere (`A`) and we have the dataset in a monthly resolution (`mon`);
- xch4: Is the name of the variable. Each observational data file is supposed to only include one variable per file;
- 200301-201612: Is the period the dataset spans with `200301` being the start year and month, and `201612` being the end year and month;

Note: There is a different naming convention for `obs4mips` data (see the exact specifications for the obs4mips data file naming convention in the `config-developer.yml` file).

20.7 7. Test the cmorized dataset

To verify that the cmorized data file is indeed correctly formatted, you can run a dedicated test recipe, that does not include any diagnostic, but only reads in the data file and has it processed in the preprocessor. Such a recipe is called `recipes/examples/recipe_check_obs.yml`. You just need to add a diagnostic for your dataset following the existing entries. Only the diagnostic of interest needs to be run, the others should be commented out for testing.

Part VII

Contributing to the community

CONTRIBUTION GUIDELINES

21.1 Contributions are very welcome

We greatly value contributions of any kind. Contributions could include, but are not limited to documentation improvements, bug reports, new or improved diagnostic code, scientific and technical code reviews, infrastructure improvements, mailing list and chat participation, community help/building, education and outreach. We value the time you invest in contributing and strive to make the process as easy as possible. If you have suggestions for improving the process of contributing, please do not hesitate to propose them.

If you have a bug or other issue to report or just need help, please open an issue on the issues tab on the ESMValTool github repository.

If you would like to contribute a new diagnostic and recipe or a new feature, please discuss your idea with the development team before getting started, to avoid double work and/or disappointment later. A good way to do this is to open an issue on GitHub. This is also a good way to get help.

21.2 Getting started

To install in development mode, follow these instructions.

- Download and install conda (this should be done even if the system in use already has a preinstalled version of conda, as problems have been reported with NCL when using such a version)

- To make the `conda` command availble, add `source <prefix>/etc/profile.d/conda.sh` to your `.bashrc` file and restart your shell. If using (t)csh shell, add `source <prefix>/etc/profile.d/conda.csh` to your `.cshrc/.tcshrc` file instead.

- Update conda: `conda update -y conda`

- Clone the ESMValTool public github repository: `git clone git@github.com:ESMValGroup/ESMValTool`, or one of the private github repositories (e.g. `git clone git@github.com:ESMValGroup/ESMValTool-private`)

- Go to the esmvaltool directory: `cd ESMValTool`

- Create the esmvaltool conda environment `conda env create --name esmvaltool --file environment.yml`

- Activate the esmvaltool environment: `conda activate esmvaltool`

- Install in development mode: `pip install -e '.[develop]'`. If you are installing behind a proxy that does not trust the usual pip-urls you can declare them with the option `--trusted-host`, e.g. `pip install --trusted-host=pypi.python.org --trusted-host=pypi.org --trusted-host=files.pythonhosted.org -e .[develop]`

- If you want to use R diagnostics, run `esmvaltool install R` to install the R dependences. Note that if you only want to run the lint test for R scripts you will have to install the `lintr` package. You can do that by running `Rscript esmvaltool/install/R/setup_devutils.R`.

- If you want to use Julia diagnostics, first install Julia as described below in section "Installing Julia", then run `esmvaltool install Julia` to install the Julia dependences. Install Julia dependences after R dependences if you plan to use both.

- Test that your installation was succesful by running `esmvaltool -h`.

- If you log into a cluster or other device via `ssh` and your origin machine sends the `locale` environment via the `ssh` connection, make sure the environment is set correctly, specifically `LANG` and `LC_ALL` are set correctly (for GB English UTF-8 encoding these variables must be set to `en_GB.UTF-8`; you can set them by adding `export LANG=en_GB.UTF-8` and `export LC_ALL=en_GB.UTF-8` in your origin or login machines' `.profile`)

- Do not run `conda update --update-all` in the `esmvaltool` environment since that will update some packages that are pinned to specific versions for the correct functionality of the environment.

21.3 Using the development version of the ESMValTool Core package

If you need the latest developments of the ESMValTool Core package, you can install it from source into the same conda environment. First follow the steps above and then:

- Clone the ESMValTool Core github repository: `git clone git@github.com:ESMValGroup/ESMValCore`)

- Go to the esmvalcore directory: `cd ESMValCore`

- Update the esmvaltool conda environment `conda env update --name esmvaltool --file environment.yml`. This step is only needed if the dependencies changed since the latest release, which will rarely happen.

- Activate the esmvaltool environment: `conda activate esmvaltool`

- Install `esmvalcore` in development mode: `pip install -e '.[develop]'`.

21.4 Installing Julia

To run Julia diagnostics you will have to install Julia; the safest way is to use the official pre-built executable and link it in the conda environment:

- Get the tarball (for v1.0.3 in this case): `wget https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.3-linux-x86_64.tar.gz`

- Unpack the tarball: `tar xfz julia-*-linux-x86_64.tar.gz`

- Symlink the Julia executable into the conda environment: `ln -s $PWD/julia-*/bin/julia $HOME/$ANACONDA/envs/esmvaltool/bin` (here `$ANACONDA` represents the name of your `anaconda` or `miniconda` directory, most commonly `anaconda3` or `miniconda3`)

- Check executable location: `which julia`

- Check Julia startup: `julia --help`

- Optionally install the Julia diagnostics dependencies: `julia esmvaltool/install/Julia/setup.jl`

Note that sometimes, if you are under a firewall, the installation of Julia diagnostics dependencies may fail due to failure of cloning the references in `$HOME/.julia/registries/General`. To fix this issue you will have to touch the registry files: `touch $HOME/.julia/environments/v1.0/Manifest.toml && touch $HOME/.julia/environments/v1.0/Project.toml` and manually git clone the references: `git clone https://github.com/JuliaRegistries/General.git $HOME/.julia/registries/General`.

21.5 Running tests

Go to the directory where the repository is cloned and run `python setup.py test`. Tests will also be run automatically by CircleCI.

21.6 Code style

To increase the readability and maintainability or the ESMValTool source code, we aim to adhere to best practices and coding standards. All pull requests are reviewed and tested by one or more members of the core development team. For code in all languages, it is highly recommended that you split your code up in functions that are short enough to view without scrolling.

21.6.1 Python

The standard document on best practices for Python code is PEP8 and there is PEP257 for documentation. We make use of numpy style docstrings to document Python functions that are visible on readthedocs.

Most formatting issues in Python code can be fixed automatically by running the commands

```
isort some_file.py
```

to sort the imports in the standard way and

```
yapf -i some_file.py
```

to add/remove whitespace as required by the standard.

To check if your code adheres to the standard, go to the directory where the repository is cloned, e.g. `cd ESMValTool`. and run

```
prospector esmvaltool/diag_scripts/your_diagnostic/your_script.py
```

Run

```
python setup.py lint
```

to see the warnings about the code style of the entire project.

We use `pycodestyle` on CircleCI to automatically check that there are no formatting mistakes and Codacy for monitoring (Python) code quality. Running prospector locally will give you quicker and sometimes more accurate results.

21.6.2 NCL

Because there is no standard best practices document for NCL, we use PEP8 for NCL code as well, with some minor adjustments to accomodate for differences in the languages. The most important difference is that for NCL code the indentation should be 2 spaces instead of 4. Use the command `nclcodestyle /path/to/file.ncl` to check if your code follows the style guide.

21.6.3 R

Best practices for R code are described in The tidyverse style guide. We check adherence to this style guide by using lintr on CircleCI. Please use styler to automatically format your code according to this style guide. In the future we would also like to make use of goodpractice to assess the quality of R code.

21.6.4 YAML

Please use `yamllint` to check that your YAML files do not contain mistakes.

21.6.5 Any text file

A generic tool to check for common spelling mistakes is codespell.

21.7 Documentation

21.7.1 What should be documented

Any code documentation that is visible on readthedocs should be well written and adhere to the standards for documentation for the respective language. Recipes should have a page in the *Recipes* section on readthedocs. This is also the place to document recipe options for the diagnostic scripts used in those recipes. Note that there is no need to write extensive documentation for functions that are not visible on readthedocs. However, adding a one line docstring describing what a function does is always a good idea.

21.7.2 How to build the documentation locally

Go to the directory where the repository is cloned and run

```
python setup.py build_sphinx -Ea
```

Make sure that your newly added documentation builds without warnings or errors.

21.8 Branches, pull requests and code review

New development should preferably be done in the main ESMValTool github repository, however, for scientists requiring confidentiality, private repositories are available. The default git branch is `master`. Use this branch to create a new feature branch from and make a pull request against. This page offers a good introduction to git branches, but it was written for BitBucket while we use GitHub, so replace the word BitBucket by GitHub whenever you read it.

It is recommended that you open a draft pull request early, as this will cause CircleCI to run the unit tests and Codacy to analyse your code. It's also easier to get help from other developers if your code is visible in a pull request.

You can view the results of the automatic checks below your pull request. If one of the tests shows a red cross instead of a green approval sign, please click the link and try to solve the issue. Note that this kind of automated checks make it easier to review code, but they are not flawless, so occasionally Codacy will report false positives.

21.8.1 Diagnostic script contributions

A pull request with diagnostic code should preferably not introduce new Codacy issues. However, we understand that there is a limit to how much time can be spend on polishing code, so up to 10 new (non-trivial) issues is still an acceptable amount.

21.8.2 List of authors

If you make a (significant) contribution to ESMValTool, please add your name to the list of authors in CITATION.cff and regenerate the file .zenodo.json by running the command

```
pip install cffconvert
cffconvert --ignore-suspect-keys --outputformat zenodo --outfile .zenodo.json
```

21.9 How to make a release

To make a new release of the package, follow these steps:

21.9.1 1. Check that the nightly build on CircleCI was successful

Check the `nightly` build on CircleCI. All tests should pass before making a release.

21.9.2 2. Make a pull request to increase the version number

The version number is stored in `esmvaltool/__init__.py`, `package/meta.yaml`, `CITATION.cff`. Make sure to update all files. See https://semver.org for more information on choosing a version number.

21.9.3 3. Make the release on GitHub

Click the releases tab and draft the new release. Do not forget to tick the pre-release box for a beta release. Use the script `esmvalcore/utils/draft_release_notes.py <`https://github.com/ESMValGroup/ESMValCore/ blob/master/esmvalcore/utils/draft_release_notes.py>`__ from the ESMValCore project to create a draft version of the release notes and edit those.

21.9.4 4. Create and upload the Conda package

Follow these steps to create a new conda package:

- Check out the tag corresponding to the release, e.g. `git checkout v2.0.0b2`

- Edit package/meta.yaml and uncomment the lines starting with `git_rev` and `git_url`, remove the line starting with `path` in the `source` section.

- Activate the base environment `conda activate base`

- Run `conda build package -c conda-forge -c esmvalgroup` to build the conda package

- If the build was successful, upload all the packages to the esmvalgroup conda channel, e.g. `anaconda upload --user esmvalgroup /path/to/conda/conda-bld/noarch/esmvaltool-2.0. 0b2-py_0.tar.bz2.`

MAKING A NEW DIAGNOSTIC OR RECIPE

22.1 Getting started

Please discuss your idea for a new diagnostic or recipe with the development team before getting started, to avoid disappointment later. A good way to do this is to open an issue on GitHub. This is also a good way to get help.

22.2 Creating a recipe and diagnostic script(s)

First create a recipe in esmvaltool/recipes to define the input data your analysis script needs and optionally preprocessing and other settings. Also create a script in the esmvaltool/diag_scripts directory and make sure it is referenced from your recipe. The easiest way to do this is probably to copy the example recipe and diagnostic script and adjust those to your needs.

If you have no preferred programming language yet, Python 3 is highly recommended, because it is most well supported. However, NCL, R, and Julia scripts are also supported.

Good example recipes for the different languages are:

- python: esmvaltool/recipes/examples/recipe_python.yml

- R: esmvaltool/recipes/examples/recipe_r.yml

- julia: esmvaltool/recipes/examples/recipe_julia.yml

- ncl: esmvaltool/recipes/examples/recipe_ncl.yml

Good example diagnostics are:

- python: esmvaltool/diag_scripts/examples/diagnostic.py

- R: esmvaltool/diag_scripts/examples/diagnostic.R

- julia: esmvaltool/diag_scripts/examples/diagnostic.jl

- ncl: esmvaltool/diag_scripts/examples/diagnostic.ncl

Unfortunately not much documentation is available at this stage, so have a look at the other recipes and diagnostics for further inspiration.

22.3 Re-using existing code

Always make sure your code is or can be released under a license that is compatible with the Apache 2 license.

If you have existing code in a supported scripting language, you have two options for re-using it. If it is fairly mature and a large amount of code, the preferred way is to package and publish it on the official package repository for that language and add it as a dependency of esmvaltool. If it is just a few simple scripts or packaging is not possible (i.e. for NCL) you can simply copy and paste the source code into the esmvaltool/diag_scripts directory.

If you have existing code in a compiled language like C, C++, or Fortran that you want to re-use, the recommended way to proceed is to add Python bindings and publish the package on PyPI so it can be installed as a Python dependency. You can then call the functions it provides using a Python diagnostic.

22.4 Recording provenance

When ESMValCore (the `esmvaltool` command) runs a recipe, it will first find all data and run the default preprocessor steps plus any additional preprocessing steps defined in the recipe. Next it will run the diagnostic script defined in the recipe and finally it will store provenance information. Provenance information is stored in the W3C PROV XML format and also plotted in an SVG file for human inspection. In addition to provenance information, a caption is also added to the plots.

22.4.1 Provenance items provided by the recipe

For each diagnostic in the recipe, ESMValCore supports the following additional information:

- `realms` a list of high-level modeling components
- `themes` a list of themes

Please see the (installed version of the) file esmvaltool/config-references.yml for all available information on each item.

22.4.2 Provenance items provided by the diagnostic script

For each output file produced by the diagnostic script, ESMValCore supports the following additional information:

- `ancestors` a list of input files used to create the plot.
- `caption` a caption text for the plot

Note that the level of detail is limited, the only valid choices for `ancestors` are files produced by ancestor tasks.

It is also possible to add more information for the implemented diagnostics using the following items:

- `authors` a list of authors
- `references` a list of references, see *Adding references* below
- `projects` a list of projects
- `domains` a list of spatial coverage of the dataset
- `plot_types` a list of plot types if the diagnostic created a plot, e.g. error bar
- `statistics` a list of types of the statistic, e.g. anomaly

Arbitrarily named other items are also supported.

Please see the (installed version of the) file esmvaltool/config-references.yml for all available information on each item. In this file, the information is written in the form of `key:     value`. Note that we add the keys to the diagnostics. The keys will automatically be replaced by their values in the final provenance records. For example, in the `config-references.yml` there is a category for types of the plots:

```
plot_types:
  errorbar: error bar plot
```

In the diagnostics, we add the key as: `plot_types:    [errorbar]` It is also possible to add custom provenance information by adding items to each category in this file.

In order to communicate with the diagnostic script, two interfaces have been defined, which are described in the ESMValCore documentation. Note that for Python and NCL diagnostics much more convenient methods are available than directly reading and writing the interface files. For other languages these are not implemented (yet).

Depending on your preferred programming language for developing a diagnostic, see the instructions and examples below on how to add provenance information:

22.4.3 Recording provenance in a Python diagnostic script

Always use *esmvaltool.diag_scripts.shared.run_diagnostic()* at the end of your script:

```python
if __name__ == '__main__':
  with run_diagnostic() as config:
      main(config)
```

And make use of a *esmvaltool.diag_scripts.shared.ProvenanceLogger* to log provenance:

```python
with ProvenanceLogger(cfg) as provenance_logger:
      provenance_logger.log(diagnostic_file, provenance_record)
```

The diagnostic_file can be obtained using *esmvaltool.diag_scripts.shared.get_diagnostic_filename*.

The `provenance_record` is a dictionary of provenance items, for example:

```python
provenance_record = {
      'ancestors': ancestor_files,
      'authors': [
          'andela_bouwe',
          'righi_mattia',
      ],
      'caption': caption,
      'domains': ['global'],
      'plot_types': ['zonal'],
      'references': [
          'acknow_project',
      ],
      'statistics': ['mean'],
    }
```

Have a look at the example Python diagnostic in esmvaltool/diag_scripts/examples/diagnostic.py for a complete example.

22.4.4 Recording provenance in an NCL diagnostic script

Always call the `log_provenance` procedure after plotting from your NCL diag_script:

```
log_provenance(nc-file,plot_file,caption,statistics,domain,plottype,authors,
↪references,input-files)
```

For example:

```
log_provenance(ncdf_outfile, \
              map@outfile, \
              "Mean of variable: " + var0, \
              "mean", \
              "global", \
              "geo", \
              (/"righi_mattia", "gottschaldt_klaus-dirk"/), \
              (/"acknow_author"/), \
              metadata_att_as_array(info0, "filename"))
```

Have a look at the example NCL diagnostic in esmvaltool/diag_scripts/examples/diagnostic.ncl for a complete example.

22.4.5 Recording provenance in a Julia diagnostic script

The provenance information is written in a `diagnostic_provenance.yml` that will be located in `run_dir`. For example a `provenance_record` can be stored in a yaml file as:

```
provenance_file = string(run_dir, "/diagnostic_provenance.yml")

open(provenance_file, "w") do io
    JSON.print(io, provenance_records, 4)
end
```

The `provenance_records` can be defined as a dictionary of provenance items. For example:

```
provenance_records = Dict()

provenance_record = Dict(
    "ancestors" => [input_file],
    "authors" => ["vonhardenberg_jost", "arnone_enrico"],
    "caption" => "Example diagnostic in Julia",
    "domains" => ["global"],
    "projects" => ["crescendo", "c3s-magic"],
    "references" => ["zhang11wcc"],
    "statistics" => ["other"],
)

provenance_records[output_file] = provenance_record
```

Have a look at the example Julia diagnostic in esmvaltool/diag_scripts/examples/diagnostic.jl for a complete example.

22.4.6 Recording provenance in an R diagnostic script

The provenance information is written in a `diagnostic_provenance.yml` that will be located in `run_dir`. For example a `provenance_record` can be stored in a yaml file as:

```
provenance_file <- paste0(run_dir, "/", "diagnostic_provenance.yml")
write_yaml(provenance_records, provenance_file)
```

The `provenance_records` can be defined as a list of provenance items. For example:

```
provenance_records <- list()

provenance_record <- list(
  ancestors = input_filenames,
  authors = list("hunter_alasdair", "perez-zanon_nuria"),
  caption = title,
  projects = list("c3s-magic"),
  statistics = list("other"),
)

provenance_records[[output_file]] <- provenance_record
```

22.5 Adding references

Recipes and diagnostic scripts can include references. When a recipe is run, citation information is stored in BibTeX format. Follow the steps below to add a reference to a recipe (or a diagnostic):

- make a `tag` that is representative of the reference entry. For example, `righi15gmd` shows the last name of the first author, year and journal abbreviation.

- add the `tag` to the `references` section in the recipe (or the diagnostic).

- make a BibTeX file for the reference entry. There are some online tools to convert a doi to BibTeX format like https://doi2bib.org/

- rename the file to the `tag`, keep the `.bibtex` extension.

- add the file to the folder `esmvaltool/references`.

Note: the `references` section in `config-references.yaml` has been replaced by the folder `esmvaltool/references`.

PORTING A NAMELIST (RECIPE) OR DIAGNOSTIC TO ESMVALTOOL V2.0

This guide summarizes the main steps to be taken in order to port an ESMValTool namelist (now called **recipe**) and the corresponding diagnostic(s) from v1.0 to v2.0, hereafter also referred as the *"old"* and the *"new version"*, respectively. The new ESMValTool version is being developed in the public git branch `master`. An identical version of this branch is maintained in the private repository as well and kept synchronized on an hourly basis.

In the following, it is assumed that the user has successfully installed ESMValTool v2 and has a rough overview of its structure (see Technical Overview).

23.1 Create a github issue

Create an issue in the public repository to keep track of your work and inform other developers. See an example here. Use the following title for the issue: "PORTING <recipe> into v2.0". Do not forget to assign it to yourself.

23.2 Create your own branch

Create your own branch from `master` for each namelist (recipe) to be ported:

```
git checkout master
git pull
git checkout -b <recipe>
```

`master` contains only v2.0 under the `./esmvaltool/` directory.

23.3 Convert xml to yml

In ESMValTool v2.0, the namelist (now recipe) is written in yaml format (Yet Another Markup Language format). It may be useful to activate the yaml syntax highlighting for the editor in use. This improves the readability of the recipe file and facilitates the editing, especially concerning the indentations which are essential in this format (like in python). Instructions can be easily found online, for example for emacs and vim.

A xml2yml converter is available in `esmvaltool/utils/xml2yml/`, please refer to the corresponding README file for detailed instructions on how to use it.

Once the recipe is converted, a first attempt to run it can be done, possibly starting with a few datasets and one diagnostics and proceed gradually. The recipe file `./esmvaltool/recipes/recipe_perfmetrics_CMIP5.yml` can be used as an example, as it covers most of the common cases.

Do not forget to also rewrite the recipe header in a `documentation` section using the yaml syntax and, if possible, to add themes and realms item to each diagnostic section. All keys and tags used for this part must be defined in `./esmvaltool/config-references.yml`. See `./esmvaltool/recipes/recipe_perfmetrics_CMIP5.yml` for an example.

23.4 Create a copy of the diag script in v2.0

The diagnostic script to be ported goes into the directory ./esmvaltool/diag_script/. It is recommended to get a copy of the very last version of the script to be ported from the `version1` branch (either in the public or in the private repository). Just create a local (offline) copy of this file from the repository and add it to ../esmvaltool/diag_script/ as a new file.

Note that (in general) this is not necessary for plot scripts and for the libraries in `./esmvaltool/diag_script/ncl/lib/`, which have already been ported. Changes may however still be necessary, especially in the plot scripts which have not yet been fully tested with all diagnostics.

23.5 Check and apply renamings

The new ESMValTool version includes a completely revised interface, handling the communication between the python workflow and the (NCL) scripts. This required several variables and functions to be renamed or removed. These chagnes are listed in the following table and have to be applied to the diagnostic code before starting with testing.

Name in v1.0	Name in v2.0	Affected code
`getenv("ESMValTool_wrk_dir")`	`config_user_info@work_dir`	all .ncl scripts
`getenv(ESMValTool_att)`	`diag_script_info@att` or `config_user_info@att`	all .ncl scripts
`xml`	`yml`	all scripts
`var_attr_ref(0)`	`variable_info@reference_dataset`	all .ncl scripts
`var_attr_ref(1)`	`variable_info@alternative_dataset`	all .ncl scripts
`models`	`input_file_info`	all .ncl scripts
`models@name`	`input_file_info@dataset`	all .ncl scripts
`verbosity`	`config_user_info@log_level`	all .ncl scripts
`isfilepresent_esmval`	`fileexists`	all .ncl scripts
`messaging.ncl`	`logging.ncl`	all .ncl scripts
`info_output(arg1, arg2, arg3)`	`log_info(arg1)` if arg3=1	all .ncl scripts
`info_output(arg1, arg2, arg3)`	`log_debug(arg1)` if arg3>1	all .ncl scripts
`verbosity = config_user_info@verbosity`	remove this statement	all .ncl scripts
`enter_msg(arg1, arg2, arg3)`	`enter_msg(arg1, arg2)`	all .ncl scripts
`leave_msg(arg1, arg2, arg3)`	`leave_msg(arg1, arg2)`	all .ncl scripts
`noop()`	appropriate `if-else` statement	all .ncl scripts
`nooperation()`	appropriate `if-else` stsatement	all .ncl scripts
`fullpaths`	`input_file_info@filename`	all .ncl scripts
`get_output_dir(arg1, arg2)`	`config_user_info@plot_dir`	all .ncl scripts
`get_work_dir`	`config_user_info@work_dir`	all .ncl scripts
`inlist(arg1, arg2)`	`any(arg1.eq.arg2)`	all .ncl scripts

continues on next page

Table 1 – continued from previous page

Name in v1.0	Name in v2.0	Affected code
`load interface_scripts/*.ncl`	`load $diag_scripts/../ interface_scripts/interface. ncl`	all .ncl scripts
`<varname>_info.tmp`	`<varname>_info.ncl` in preproc dir	all .ncl scripts
`ncl.interface`	`settings.ncl` in run_dir and `interface_scripts/interface. ncl`	all .ncl scripts
`load diag_scripts/lib/ncl/`	`load $diag_scripts/shared/`	all .ncl scripts
`load plot_scripts/ncl/`	`load $diag_scripts/shared/ plot/`	all .ncl scripts
`load diag_scripts/lib/ncl/ rgb/`	`load $diag_scripts/shared/ plot/rgb/`	all .ncl scripts
`load diag_scripts/lib/ncl/ styles/`	`load $diag_scripts/shared/ plot/styles`	all .ncl scripts
`load diag_scripts/lib/ncl/ misc_function.ncl`	`load $diag_scripts/shared/ plot/misc_function.ncl`	all .ncl scripts
`LW_CRE, SW_CRE`	`lwcre, swcre`	some yml recipes
`check_min_max_models`	`check_min_max_datasets`	all .ncl scripts
`get_ref_model_idx`	`get_ref_dataset_idx`	all .ncl scripts
`get_model_minus_ref`	`get_dataset_minus_ref`	all .ncl scripts

The following changes may also have to be considered:

- namelists are now called recipes and collected in `esmvaltool/recipes`;

- models are now called datasets and all files have been updated accordingly, including NCL functions (see table above);

- `run_dir` (previous `interface_data`), `plot_dir`, `work_dir` are now unique to each diagnostic script, so it is no longer necessary to define specific paths in the diagnostic scripts to prevent file collision;

- `input_file_info` is now a list of a list of logicals, where each element describes one dataset and one variable. Convenience functions to extract the required elements (e.g., all datasets of a given variable) are provided in `esmvaltool/interface_scripts/interface.ncl`;

- the interface functions `interface_get_*` and `get_figure_filename` are no longer available: their functionalities can be easily reproduced using the `input_file_info` and the convenience functions in `esmvaltool/interface_scripts/interface.ncl` to access the required attributes;

- there are now only 4 log levels (`debug`, `info`, `warning`, and `error`) instead of (infinite) numerical values in `verbosity`

- diagnostic scripts are now organized in subdirectories in `esmvaltool/diag_scripts/`: all scripts belonging to the same diagnostics are to be collected in a single subdirectory (see `esmvaltool/diag_scripts/ perfmetrics/` for example). This applies also to the `aux_` scripts, unless they are shared among multiple diagnostics (in this case they go in `shared/`);

- the relevant input_file_info items required by a plot routine should be passed as argument to the routine itself;

- upper case characters have to be avoided in script names, if possible.

As for the recipe, the diagnostic script `./esmvaltool/diag_scripts/perfmetrics/main.ncl` can be followed as working example.

23.6 Move preprocessing from the diagnostic script to the backend

Many operations previously performed by the diagnostic scripts, are now included in the backend, including level extraction, regridding, masking, and multi-model statistics. If the diagnostics to be ported contains code performing any of such operations, the corresponding code has to be removed from the diagnostic script and the respective backend functionality can be used instead.

The backend operations are fully controlled by the `preprocessors` section in the recipe. Here, a number of preprocessor sets can be defined, with different options for each of the operations. The sets defined in this section are applied in the `diagnostics` section to preprocess a given variable.

It is recommended to proceed step by step, porting and testing each operation separately before proceeding with the next one. A useful setting in the user configuration file (`config-private.yml`) called `write_intermediary_cube` allows writing out the variable field after each preprocessing step, thus facilitating the comparison with the old version (e.g., after CMORization, level selection, after regridding, etc.). The CMORization step of the new backend exactly corresponds to the operation performed by the old backend (and stored in the `climo` directory, now called `preprec`): this is the very first step to be checked, by simply comparing the intermediary file produced by the new backend after CMORization with the output of the old backend in the `climo` directorsy (see "Testing" below for instructions).

The new backend also performs variable derivation, replacing the `calculate` function in the `variable_defs` scripts. If the recipe which is being ported makes use of derived variables, the corresponding calculation must be ported from the `./variable_defs/<variable>.ncl` file to `./esmvaltool/preprocessor/_derive.py`.

Note that the Python library `esmval_lib`, containing the `ESMValProject` class is no longer available in version 2. Most functionalities have been moved to the new preprocessor. If you miss a feature, please open an issue on github [https://github.com/ESMValGroup/ESMValTool/issues].

23.7 Move diagnostic- and variable-specific settings to the recipe

In the new version, all settings are centralized in the recipe, completely replacing the diagnostic-specific settings in `./nml/cfg_files/` (passed as `diag_script_info` to the diagnostic scripts) and the variable-specific settings in `variable_defs/<variable>.ncl` (passed as `variable_info`). There is also no distinction anymore between diagnostic- and variable-specific settings: they are collectively defined in the `scripts` dictionary of each diagnostic in the recipe and passed as `diag_script_info` attributes by the new ESMValTool interface. Note that the `variable_info` logical still exists, but it is used to pass variable information as given in the corresponding dictionary of the recipe.

23.8 Make sure the diagnostic script writes NetCDF output

Each diagnostic script is required to write the output of the anaylsis in one or more NetCDF files. This is to give the user the possibility to further look into the results, besides the plots, but (most importantly) for tagging purposes when publishing the data in a report and/or on a website.

For each of the plot produced by the diagnostic script a single NetCDF file has to be generated. The variable saved in this file should also contain all the necessary metadata that documents the plot (dataset names, units, statistical methods, etc.). The files have to be saved in the work directory (defined in *cfg['work_dir']* and *config_user_info@work_dir*, for the python and NCL diagnostics, respectively).

23.9 Test the recipe/diagnostic in the new version

Once complete, the porting of the diagnostic script can be tested. Most of the diagnostic script allows writing the output in a NetCDF file before calling the plotting routine. This output can be used to check whether the results of v1.0 are correctly reproduced. As a reference for v1.0, it is recommended to use the development branch.

There are two methods for comparing NetCDF files: `cdo` and `ncdiff`. The first method is applied with the command:

```
cdo diffv old_output.nc new_output.nc
```

which will print a log on the stdout, reporting how many records of the file differ and the absolute/relative differences.

The second method produces a NetCDF file (e.g., `diff.nc`) with the difference between two given files:

```
ncdiff old_output.nc new_output.nc diff.nc
```

This file can be opened with `ncview` to visually inspect the differences.

In general, binary identical results cannot be expected, due to the use of different languages and algorithms in the two versions, especially for complex operations such as regridding. However, difference within machine precision are desirable. At this stage, it is essential to test all datasets in the recipe and not just a subset of them.

It is also recommended to compare the graphical output (this may be necessary if the ported diagnostic does not produce a NetCDF output). For this comparison, the PostScript format is preferable, since it is easy to directly compare two PostScript files with the standard `diff` command in Linux:

```
diff old_graphic.ps new_graphic.ps
```

but it is very unlikely to produce no differences, therefore visual inspection of the output may also be required.

23.10 Clean the code

Before submitting a pull request, the code should be cleaned to adhere to the coding standard, which are somehow stricter in v2.0. This check is performed automatically on GitHub (CircleCI and Codacy) when opening a pull request on the public repository. A code-style checker (`nclcodestyle`) is available in the tool to check NCL scripts and installed alongside the tool itself. When checking NCL code style, the following should be considered in addition to the warning issued by the style checker:

- two-space instead of four-space indentation is now adopted for NCL as per NCL standard;

- `load` statements for NCL standard libraries should be removed: these are automatically loaded since NCL v6.4.0 (see NCL documentation);

- the description of diagnostic- and variable-specific settings can be moved from the header of the diagnostic script to the recipe, since the settings are now defined there (see above);

- NCL `print` and `printVarSummary` statements must be avoided and replaced by the `log_info` and `log_debug` functions;

- for error and warning statments, the `error_msg` function can be used, which automatically include an exit statement.

23.11 Update the documentation

If necessary, add or update the documentation for your recipes in the corrsponding rst file, which is now in `doc\sphinx\source\recipes`. Do not forget to also add the documentation file to the list in `doc\sphinx\source\annex_c` to make sure it actually appears in the documentation.

23.12 Open a pull request

Create a pull request on github to merge your branch back to `master`, provide a short description of what has been done and nominate one or more reviewers.

GITHUB WORKFLOW

24.1 Basics

The source code of the ESMValTool is hosted on GitHub. The following description gives an overview of the typical workflow and usage for implementing new diagnostics or technical changes into the ESMValTool. For general information on Git, see e.g. the online documentation at https://www.git-scm.com/doc.

There are *two* ESMValTool GitHub repositories available:

1. The **PUBLIC** GitHub repository is open to the public. The ESMValTool is released as open-source software under the Apache License 2.0. Use of the software constitutes acceptance of this license and terms. The PUBLIC ESMValTool repository is located at https://github.com/ESMValGroup/ESMValTool

2. The **PRIVATE** GitHub repository is restricted to the ESMValTool Development Team. This repository is only accessible to ESMValTool developers that have accepted the terms of use for the ESMValTool development environment. The use of the ESMValTool software and access to the private ESMValTool GitHub repository constitutes acceptance of these terms. *When you fork or copy this repository, you must ensure that you do not copy the PRIVATE repository into an open domain!* The PRIVATE ESMValTool repository for the ESMValTool development team is located at https://github.com/ESMValGroup/ESMValTool-private

All developments can be made in either of the two repositories. The creation of *FEATURE BRANCHES* (see below), however, is restricted to registered ESMValTool developers in both repositories. We encourage all developers to join the ESMValTool development team. Please contact the *ESMValTool Core Development Team* if you want to join the ESMValTool development team. The PRIVATE GitHub repository offers a central protected environment for ESMValTool developers who would like to keep their contributions undisclosed (e.g., unpublished scientific work, work in progress by PhD students) while at the same time benefiting from the possibilities of collaborating with other ESMValTool developers and having a backup of their work. *FEATURE BRANCHES* created in the PRIVATE repository are only visible to the ESMValTool development team but not to the public. The concept of a PRIVATE repository has proven to be very useful to efficiently share code during the development across institutions and projects in a common repository without having the contributions immediately accessible to the public.

Both, the PUBLIC and the PRIVATE repository, contain the following kinds of branches:

- *MASTER BRANCH* (official releases),

- *DEVELOPMENT BRANCH* (includes approved new contributions but version is not yet fully tested),

- *FEATURE BRANCH* (development branches for new features and diagnostics created by developers, the naming convention for *FEATURE BRANCHES* is <Project>_<myfeature>).

24.2 Access rights

- Write access to the *MASTER* and *DEVELOPMENT BRANCH* in both, the PUBLIC and the PRIVATE GitHub repositories, is restricted to the *ESMValTool Core Development Team*.

- *FEATURE BRANCHES* in both the PUBLIC and the PRIVATE repository can be created by all members of the ESMValTool development team (i.e. members in the GitHub organization "ESMValGroup"). If needed, branches can be individually write-protected within each repository so that other developers cannot accidently push changes to these branches.

The *MASTER BRANCH* of the PRIVATE repository will be regularly synchronized with the *MASTER BRANCH* of the PUBLIC repository (currently by hand). This ensures that they are identical at all times (see schematic in Figure Fig. 1). The recommended workflow for members of the ESMValTool development team is to create additional *FEATURE BRANCHES* in either the PUBLIC or the PRIVATE repository, see further instructions below.
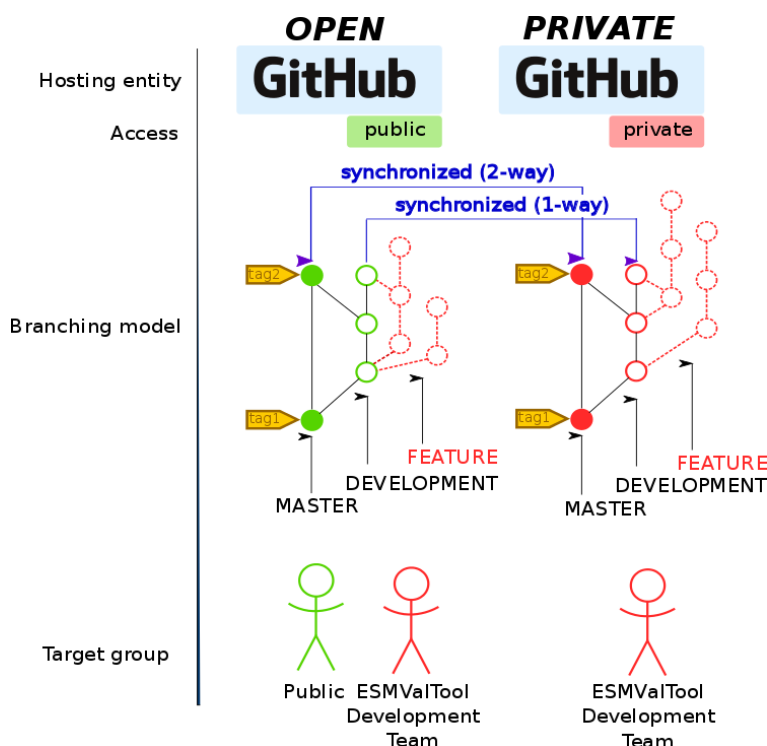
Fig. 1: Schematic diagram of the ESMValTool GitHub repositories.

24.3 Workflow

The following description gives an overview of the typical workflow and usage for implementing new diagnostics or technical changes into the ESMValTool. The description assumes that your local development machine is running a Unix-like operating system. For a general introduction to Git tutorials such as, for instance, https://www.git-scm.com/docs/gittutorial are recommended.

24.3.1 Getting started

First make sure that you have Git installed on your development machine. On shared machines, software is usually installed using the environment modules. Try e.g.

```
module avail git
```

if this is the case. You can ask your system administrator for assistance. You can test this with the command:

```
git --version
```

In order to properly identify your contributions to the ESMValTool you need to configure your local Git with some personal data. This can be done with the following commands:

```
git config --global user.name "YOUR NAME"
git config --global user.email "YOUR EMAIL"
```

Note: For working on GitHub you need to create an account and login to https://github.com/.

24.3.2 Working with the ESMValTool GitHub Repositories

As a member of the ESMValTool development team you can create *FEATURE BRANCHES* in the PUBLIC as well as in the PRIVATE repository. We encourage all ESMValTool developers to use the following workflow for long-lived developments (>2 weeks).

- Login to GitHub.com

- On GitHub, go to the website of the ESMValTool repository (https://github.com/ESMValGroup/ESMValTool-private or https://github.com/ESMValGroup/ESMValTool)

- Click on the button create *FEATURE BRANCH*

- Select the *"DEVELOPMENT" BRANCH* and create a new *FEATURE BRANCH* for the diagnostic/feature you want to implement. Please follow the following naming convention for your new *FEATURE BRANCH*: <Project>_<myfeature>.
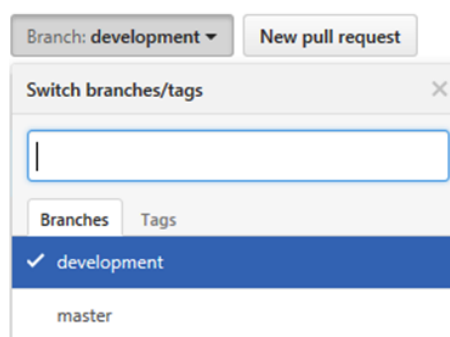
- Click the button "Clone or Download" and copy the URL shown there

- Open a terminal window and go to the folder where you would like to store your local copy of the ESMValTool source

- Type git clone, and paste the URL:

```
git clone <URL_FROM_CLIPBOARD>
```

This will clone the ESMValTool repository at GitHub to a local folder. You can now query the status of your local working copy with:

```
git status
```

You will see that you are on a branch called master and your local working copy is up to date with the remote repository. With

```
git branch --all
```

you can list all available remote and local branches. Now switch to your feature branch by:

```
git checkout <NAME_OF_YOUR_FEATURE_BRANCH>
```

You can now start coding. To check your current developments you can use the command

```
git status
```

You can add new files and folders that you want to have tracked by Git using:

```
git add <NEW_FILE|FOLDER>
```

Commit your tracked changes to your local working copy via:

```
git commit -m "YOUR COMMIT MESSAGE"
```

You can inspect your changes with (use man git-log for all options):

```
git log
```

To share your work and to have an online backup, push your local development to your *FEATURE BRANCH* on GitHub:

```
git push origin <YOUR_FEATURE_BRANCH>
```

Note: An overview on Git commands and best practices can be found e.g. here: https://zeroturnaround.com/rebellabs/git-commands-and-best-practices-cheat-sheet/

24.3.3 Pull requests

Once your development is completely finished, go to the GitHub website of the ESMValTool repository and switch to your *FEATURE BRANCH*. You can then initiate a pull request by clicking on the button "New pull request". Select the *DEVELOPMENT BRANCH* as "base branch" and click on "Create pull request". Your pull request will then be tested, discussed and implemented into the *DEVELPOMENT BRANCH* by the *ESMValTool Core Development Team*.

> **Attention:** When creating a pull request, please carefully review the requirements and recommendations in CON-TRIBUTING.md and try to implement those (see also checklist in the pull request template). It is recommended that you create a draft pull request early in the development process, when it is still possible to implement feedback. Do not wait until shortly before the deadline of the project you are working on. If you are unsure how to implement any of the requirements, please do not hesitate to ask for help in the pull request.

24.3.4 GitHub issues

In case you encounter a bug of if you have a feature request or something similar you can open an issue on the PUBLIC ESMValTool GitHub repository.

24.4 General do-s and don't-s

24.4.1 Do-s

- Create a *FEATURE BRANCH* and use exclusively this branch for developing the ESMValTool. The naming convention for *FEATURE BRANCHES* is <Project>_<myfeature>.

- Comment your code as much as possible and in English.

- Use short but self-explanatory variable names (e.g., model_input and reference_input instead of xm and xr).

- Consider a modular/functional programming style. This often makes code easier to read and deletes intermediate variables immediately. If possible, separate diagnostic calculations from plotting routines.

- Consider reusing or extending existing code. General-purpose code can be found in esmvaltool/diag_scripts/shared/.

- Comment all switches and parameters including a list of all possible settings/options in the header section of your code (see also . . .).

- Use templates for recipes (see . . .) and diagnostics (see . . .) to help with proper documentation.

- Keep your *FEATURE BRANCH* regularly synchronized with the *DEVELOPMENT BRANCH* (git merge).

- Keep developments / modifications of the ESMValTool framework / backend / basic structure separate from developments of diagnostics by creating different *FEATURE BRANCHES* for these two kinds of developments. Create *FEATURE BRANCHES* for changes / modifications of the ESMValTool framework only in the *PUBLIC* repository.

24.4.2 Don't-s

- Do not use other programming languages than the ones currently supported (Python, R, NCL, Julia). If you are unsure what language to use, Python is probably the best choice, because it has very good libraries available and is supported by a large community. Contact the *ESMValTool Core Development Team* if you wish to use another language, but remember that only open-source languages are supported by the ESMValTool.

- Do not develop without proper version control (see do-s above).

- Avoid large (memory, disk space) intermediate results. Delete intermediate files/variables or see modular/functional programming style.

- Do not use hard-coded pathnames or filenames.

- Do not mix developments / modifications of the ESMValTool framework and developments / modifications of diagnostics in the same *FEATURE BRANCH*.

CONTACT INFORMATION

See www.esmvaltool.org for general contact information.

25.1 Core development team

- Deutsches Zentrum für Luft- und Raumfahrt (DLR), Institut für Physik der Atmosphäre, Germany (PI)

  ESMValTool Core PI and Developer: contact for requests to use the ESMValTool and for collaboration with the development team, access to the PRIVATE GitHub repository.

- Alfred Wegener institute (AWI) Bremerhaven, Germany

- Barcelona Supercomputing Center (BSC), Spain

- Netherlands eScience Center (NLeSC), The Netherlands

- Ludwig Maximilian University of Munich, Germany

- Plymouth Marine Laboratory (PML), United Kingdom

- Swedish Meteorological and Hydrological Institute (SMHI), Sweden

- University of Reading, United Kingdom

25.2 Recipes and diagnostics

Contacts for specific diagnostic sets are the respective authors, as listed in the corresponding diagnostic documentation and in the source code.

25.3 User mailing list

The ESMValTool user mailing list is open for all general and technical questions on the ESMValTool, for example about installation, application, development, etc.

To subscribe send an email to sympa@listserv.dfn.de with the following subject line:

- *subscribe esmvaltool*

or

- *subscribe esmvaltool YOUR_FIRSTNAME YOUR_LASTNAME*

The mailing list also has an [public archive](#) online.

Part VIII

Utilities

This section provides information on small tools that are available in the esmvaltool/utils directory.

DRAFT_RELEASE_NOTES.PY

Script for drafting release notes based on the titles of the GitHub pull requests that have been merged since the previous release.

NCLCODESTYLE

A tool for checking the style of NCL code, based on pycodestyle. Install ESMValTool in development mode (`pip install -e '.[develop]'`) to make it available. To use it, run

```
nclcodestyle /path/to/file.ncl
```

XML2YML

A tool for converting version 1 recipes to version 2 recipes. See the README.md file in the directory esmvaltool/utils/xml2yml for detailed usage instructions.

TWENTYNINE

TESTING

Tools for testing recipes.

29.1 test recipe settings

A tool for generating recipes with various diagnostic settings, to test of those work. Install ESMVaLTool in development mode (`pip install -e '.[develop]'`) to make it available. To use it, run

```
test_recipe --help
```

Part IX

ESMValTool Code API Documentation

ESMValTool is mostly used as a commandline tool. However, it is also possibly to use (parts of) EsmValTool as a library. This section documents the public API of ESMValTool.

SHARED DIAGNOSTIC SCRIPT CODE

Code that is shared between multiple diagnostic scripts.

Functions

run_diagnostic()	Run a Python diagnostic.
get_diagnostic_filename(basename, cfg[, . . .])	Get a valid path for saving a diagnostic data file.
get_plot_filename(basename, cfg)	Get a valid path for saving a diagnostic plot.
select_metadata(metadata, **attributes)	Select specific metadata describing preprocessed data.
sorted_metadata(metadata, sort)	Sort a list of metadata describing preprocessed data.
group_metadata(metadata, attribute[, sort])	Group metadata describing preprocessed data by attribute.
sorted_group_metadata(metadata_groups, sort)	Sort grouped metadata.
extract_variables(cfg[, as_iris])	Extract basic variable information from configuration dictionary.
variables_available(cfg, short_names)	Check if data from certain variables is available.
get_cfg([filename])	Read diagnostic script configuration from settings.yml.
get_control_exper_obs(short_name, . . .)	Get control, exper and obs datasets
apply_supermeans(ctrl, exper, obs_list)	Apply supermeans on data components ie MEAN on time

Classes

ProvenanceLogger(cfg)	Open the provenance logger.
Variable(short_name, standard_name, . . .)	
Variables([cfg])	Class to easily access a recipe's variables in a diagnostic.
Datasets(cfg)	Class to easily access a recipe's datasets in a diagnostic script.

class esmvaltool.diag_scripts.shared.**Datasets**(*cfg*)
　　Bases: `object`

　　Class to easily access a recipe's datasets in a diagnostic script.

Examples

Methods

add_dataset(path[, data])	Add dataset to class.
add_to_data(data[, path])	Add element to a dataset's data.
get_data([path])	Access a dataset's data.
get_data_list(**dataset_info)	Access the datasets' data in a list.
get_dataset_info([path])	Access a dataset's information.
get_dataset_info_list(**dataset_info)	Access dataset's information in a list.
get_info(key[, path])	Access a 'dataset_info`'s *key*.
get_info_list(key, **dataset_info)	Access *dataset_info*'s *key* values.
get_path(**dataset_info)	Access a dataset's path.
get_path_list(**dataset_info)	Access dataset's paths in a list.
set_data(data[, path])	Set element as a dataset's data.

Get all variables of a recipe configuration *cfg*:

```
datasets = Datasets(cfg)
```

Access data of a dataset with path *dataset_path*:

```
datasets.get_data(path=dataset_path)
```

Access dataset information of the dataset:

```
datasets.get_dataset_info(path=dataset_path)
```

Access the data of all datasets with *exp=piControl*:

```
datasets.get_data_list(exp=piControl)
```

add_dataset(*path*, *data=None*, ***dataset_info*)
   Add dataset to class.

> **Parameters**
>
> - **path** (*str*) – (Unique) path to the dataset.
>
> - **data** (*optional*) – Arbitrary object to be saved as data for the dataset.
>
> - ***dataset_info** (*optional*) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.

add_to_data(*data*, *path=None*, ***dataset_info*)
   Add element to a dataset's data.

Notes

Either *path* or a unique *dataset_info* description have to be given. Fails when given information is ambiguous.

> **Parameters**
>
> - **data** – Element to be added to the dataset's data.
>
> - **path** (`str, optional`) – Path to the dataset
>
> - ****dataset_info** (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Raises** `RuntimeError` – If data given by *dataset_info* is ambiguous.

get_data (*path=None*, ***dataset_info*)

Access a dataset's data.

Notes

Either *path* or a unique *dataset_info* description have to be given. Fails when given information is ambiguous.

> **Parameters**
>
> - **path** (`str, optional`) – Path to the dataset
>
> - ****dataset_info** (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Returns** Data of the selected dataset.
>
> **Return type** *data_object*
>
> **Raises** `RuntimeError` – If data given by *dataset_info* is ambiguous.

get_data_list (***dataset_info*)

Access the datasets' data in a list.

Notes

The returned data is sorted alphabetically respective to the *paths*.

> **Parameters** ****dataset_info** (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Returns** Data of the selected datasets.
>
> **Return type** list

get_dataset_info (*path=None*, ***dataset_info*)

Access a dataset's information.

Notes

Either *path* or a unique *dataset_info* description have to be given. Fails when given information is ambiguous.

> **Parameters**
>
> - **path** (`str, optional`) – Path to the dataset.
>
> - ****dataset_info** (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Returns**   All dataset information.
>
> **Return type** [dict]
>
> **Raises** **RuntimeError** – If data given by *dataset_info* is ambiguous.

get_dataset_info_list(***dataset_info*)

Access dataset's information in a list.

Notes

The returned data is sorted alphabetically respective to the *paths*.

> **Parameters **dataset_info** (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Returns**   Information dictionaries of the selected datasets.
>
> **Return type** [list]

get_info(*key*, *path=None*, ***dataset_info*)

Access a 'dataset_info`'s *key*.

Notes

Either *path* or a unique *dataset_info* description have to be given. Fails when given information is ambiguous. If the *dataset_info* does not contain the *key*, returns None.

> **Parameters**
>
> - **key** (`str`) – Desired dictionary key.
>
> - **path** (`str`) – Path to the dataset.
>
> - ****dataset_info** (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Returns**   *key* information of the given dataset.
>
> **Return type** [str]
>
> **Raises** **RuntimeError** – If data given by *dataset_info* is ambiguous.

get_info_list(*key*, ***dataset_info*)

Access *dataset_info*'s *key* values.

Notes

The returned data is sorted alphabetically respective to the *paths*.

> **Parameters** `**dataset_info` (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Returns** *key* information of the selected datasets.
>
> **Return type** list

get_path(***dataset_info*)
> Access a dataset's path.

Notes

A unique *dataset_info* description has to be given. Fails when given information is ambiguous.

> **Parameters** `**dataset_info` (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Returns** Path of the selected dataset.
>
> **Return type** str
>
> **Raises** `RuntimeError` – If data given by *dataset_info* is ambiguous.

get_path_list(***dataset_info*)
> Access dataset's paths in a list.

Notes

The returned data is sorted alphabetically respective to the *paths*.

> **Parameters** `**dataset_info` (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Returns** Paths of the selected datasets.
>
> **Return type** list

set_data(*data*, *path=None*, ***dataset_info*)
> Set element as a dataset's data.

Notes

Either *path* or a unique *dataset_info* description have to be given. Fails when if given information is ambiguous.

> **Parameters**
>
> - **data** – Element to be set as the dataset's data.
> - **path** (`str, optional`) – Path to the dataset.
> - `**dataset_info` (`optional`) – Keyword arguments describing the dataset, e.g. *dataset=CanESM2*, *exp=piControl* or *short_name=tas*.
>
> **Raises** `RuntimeError` – If data given by *dataset_info* is ambiguous.

class esmvaltool.diag_scripts.shared.**ProvenanceLogger**(*cfg*)

    Bases: `object`

    Open the provenance logger.

        Parameters **cfg** (`dict`) – Dictionary with diagnostic configuration.

    Methods

log(filename, record)	Record provenance.

Example

Use as a context manager:

```
record = {
    'caption': "This is a nice plot.",
    'statistics': ['mean'],
    'domain': 'global',
    'plot_type': 'zonal',
    'plot_file': '/path/to/result.png',
    'authors': [
        'first_author',
        'second_author',
    ],
    'references': [
        'acknow_project',
    ],
    'ancestors': [
        '/path/to/input_file_1.nc',
        '/path/to/input_file_2.nc',
    ],
}
output_file = '/path/to/result.nc'

with ProvenanceLogger(cfg) as provenance_logger:
    provenance_logger.log(output_file, record)
```

log(*filename*, *record*)

    Record provenance.

        Parameters

            • **filename** (`str`) – Name of the file containing the diagnostic data.

            • **record** (`dict`) – Dictionary with the provenance information to be logged.

                Typical keys are:

                    – plot_type

                    – plot_file

                    – caption

                    – ancestors

                    – authors

                    – references

Note: See also esmvaltool/config-references.yml

---

class `esmvaltool.diag_scripts.shared.`**`Variable`**(*short_name*, *standard_name*, *long_name*, *units*)

Bases: `tuple` **Methods**

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

long_name	Alias for field number 2
short_name	Alias for field number 0
standard_name	Alias for field number 1
units	Alias for field number 3

count(*value*, */*)
> Return number of occurrences of value.

index(*value*, *start=0*, *stop=9223372036854775807*, */*)
> Return first index of value.
>
> Raises ValueError if the value is not present.

property long_name
> Alias for field number 2

property short_name
> Alias for field number 0

property standard_name
> Alias for field number 1

property units
> Alias for field number 3

class `esmvaltool.diag_scripts.shared.`**`Variables`**(*cfg=None*, ***names*)
Bases: `object`

Class to easily access a recipe's variables in a diagnostic.

Examples

Methods

add_vars(**names)	Add costum variables to the class.
iris_dict(var)	Access `iris` dictionary of the variable.
long_name(var)	Access long name.
modify_var(var, **names)	Modify an already existing variable of the class.
short_name(var)	Access short name.
short_names()	Get list of all *short_names*.
standard_name(var)	Access standard name.
standard_names()	Get list of all *standard_names*.

Table 7 – continued from previous page

units(var)	Access units.
var_name(var)	Access var name.
vars_available(*args)	Check if given variables are available.

Get all variables of a recipe configuration *cfg*:

```
variables = Variables(cfg)
```

Access information of a variable *tas*:

```
variables.short_name('tas')
variables.standard_name('tas')
variables.long_name('tas')
variables.units('tas')
```

Access `iris`-suitable dictionary of a variable *tas*:

```
variables.iris_dict('tas')
```

Check if variables *tas* and *pr* are available:

```
variables.vars_available('tas', 'pr')
```

add_vars(***names*)
    Add costum variables to the class.

> **Parameters** ****names** (`dict or Variable, optional`) – Keyword arguments of the form *short_name=Variable_object* where *Variable_object* can be given as `dict` or `Variable`.

iris_dict(*var*)
    Access `iris` dictionary of the variable.

> **Parameters** **var** (`str`) – (Short) name of the variable.

> **Returns** Dictionary containing all attributes of the variable which can be used directly in `iris` (*short_name* replaced by *var_name*).

> **Return type** dict

long_name(*var*)
    Access long name.

> **Parameters** **var** (`str`) – (Short) name of the variable.

> **Returns** Long name of the variable.

> **Return type** str

modify_var(*var*, ***names*)
    Modify an already existing variable of the class.

> **Parameters**
>
> - **var** (`str`) – (Short) name of the existing variable.
>
> - ****names** – Keyword arguments of the form *short_name=tas*.
>
> **Raises**
>
> - **ValueError** – If *var* is not an existing variable.

• **TypeError** – If a non-valid keyword argument is given.

short_name(*var*)

Access short name.

> **Parameters** **var** (`str`) – (Short) name of the variable.

> **Returns** Short name of the variable.

> **Return type** str

short_names()

Get list of all *short_names*.

> **Returns** List of all *short_names*.

> **Return type** list

standard_name(*var*)

Access standard name.

> **Parameters** **var** (`str`) – (Short) name of the variable.

> **Returns** Standard name of the variable.

> **Return type** str

standard_names()

Get list of all *standard_names*.

> **Returns** List of all *standard_names*.

> **Return type** list

units(*var*)

Access units.

> **Parameters** **var** (`str`) – (Short) name of the variable.

> **Returns** Units of the variable.

> **Return type** str

var_name(*var*)

Access var name.

> **Parameters** **var** (`str`) – (Short) name of the variable.

> **Returns** Var name (=short name) of the variable.

> **Return type** str

vars_available(**args*)

Check if given variables are available.

> **Parameters** ***args** – Short names of the variables to be tested.

> **Returns** *True* if variables are available, *False* if not.

> **Return type** bool

esmvaltool.diag_scripts.shared.**apply_supermeans**(*ctrl*, *exper*, *obs_list*)

Apply supermeans on data components ie MEAN on time

This function is an extension of climate_statistics() meant to ease the time-meaning procedure when dealing with CONTROL, EXPERIMENT and OBS (if any) datasets. ctrl: dictionary of CONTROL dataset exper: dictionary of EXPERIMENT dataset obs_lis: list of dicts for OBS datasets (0, 1 or many)

Returns: control and experiment cubes and list of obs cubes

esmvaltool.diag_scripts.shared.**extract_variables**(*cfg*, *as_iris=False*)
    Extract basic variable information from configuration dictionary.

    Returns *short_name*, *standard_name*, *long_name* and *units* keys for each variable.

        Parameters

- **cfg** (`dict`) – Diagnostic script configuration.

- **as_iris** (`bool, optional`) – Replace *short_name* by *var_name*, this can be used directly in `iris` classes.

        Returns Variable information in `dict`s (values) for each `short_name` (key).

        Return type dict

esmvaltool.diag_scripts.shared.**get_cfg**(*filename=None*)
    Read diagnostic script configuration from settings.yml.

esmvaltool.diag_scripts.shared.**get_control_exper_obs**(*short_name*, *input_data*, *cfg*, *cmip_type*)

    Get control, exper and obs datasets

    This function is used when running recipes that need a clear distinction between a control dataset, an experiment dataset and have optional obs (OBS, obs4mips etc) datasets; such recipes include recipe_validation, and all the autoassess ones; short_name: variable short name input_data: dict containing the input data info cfg: config file as used in this module

esmvaltool.diag_scripts.shared.**get_diagnostic_filename**(*basename*, *cfg*, *extension='nc'*)
    Get a valid path for saving a diagnostic data file.

        Parameters

- **basename** (`str`) – The basename of the file.

- **cfg** (`dict`) – Dictionary with diagnostic configuration.

- **extension** (`str`) – File name extension.

        Returns A valid path for saving a diagnostic data file.

        Return type str

esmvaltool.diag_scripts.shared.**get_plot_filename**(*basename*, *cfg*)
    Get a valid path for saving a diagnostic plot.

        Parameters

- **basename** (`str`) – The basename of the file.

- **cfg** (`dict`) – Dictionary with diagnostic configuration.

        Returns A valid path for saving a diagnostic plot.

        Return type str

esmvaltool.diag_scripts.shared.**group_metadata**(*metadata*, *attribute*, *sort=None*)
    Group metadata describing preprocessed data by attribute.

        Parameters

- **metadata** (`list` of `dict`) – A list of metadata describing preprocessed data.

- **attribute** (`str`) – The attribute name that the metadata should be grouped by.

- **sort** – See *sorted_group_metadata*.

> **Returns** A dictionary containing the requested groups. If sorting is requested, an *OrderedDict* will be returned.

> **Return type** `dict` of `list` of `dict`

esmvaltool.diag_scripts.shared.**run_diagnostic**()
Run a Python diagnostic.

This context manager is the main entry point for most Python diagnostics.

Example

See esmvaltool/diag_scripts/examples/diagnostic.py for an extensive example of how to start your diagnostic.

Basic usage is as follows, add these lines at the bottom of your script:

```python
def main(cfg):
    # Your diagnostic code goes here.
    print(cfg)

if __name__ == '__main__':
    with run_diagnostic() as cfg:
        main(cfg)
```

The *cfg* dict passed to *main* contains the script configuration that can be used with the other functions in this module.

esmvaltool.diag_scripts.shared.**select_metadata**(*metadata*, ***attributes*)
Select specific metadata describing preprocessed data.

> **Parameters**
>
> - **metadata** (`list` of `dict`) – A list of metadata describing preprocessed data.
>
> - ****attributes** – Keyword arguments specifying the required variable attributes and their values. Use the value '*' to select any variable that has the attribute.

> **Returns** A list of matching metadata.

> **Return type** `list` of `dict`

esmvaltool.diag_scripts.shared.**sorted_group_metadata**(*metadata_groups*, *sort*)
Sort grouped metadata.

Sorting is done on strings and is not case sensitive.

> **Parameters**
>
> - **metadata_groups** (`dict` of `list` of `dict`) – Dictionary containing the groups of metadata.
>
> - **sort** (`bool` or `str` or `list` of `str`) – One or more attributes to sort by or True to just sort the groups but not the lists.

> **Returns** A dictionary containing the requested groups.

> **Return type** `OrderedDict` of `list` of `dict`

esmvaltool.diag_scripts.shared.**sorted_metadata**(*metadata*, *sort*)
Sort a list of metadata describing preprocessed data.

Sorting is done on strings and is not case sensitive.

Parameters

- **metadata** (`list` of `dict`) – A list of metadata describing preprocessed data.

- **sort** (`str` or `list` of `str`) – One or more attributes to sort by.

Returns The sorted list of variable metadata.

Return type `list` of `dict`

esmvaltool.diag_scripts.shared.**variables_available**(*cfg*, *short_names*)

Check if data from certain variables is available.

Parameters

- **cfg** (`dict`) – Diagnostic script configuration.

- **short_names** (`list of str`) – Variable *short_names* which should be checked.

Returns *True* if all variables available, *False* if not.

Return type bool

30.1 Plotting

Module that provides common plot functions.

Functions

get_path_to_mpl_style([style_file])	Get path to matplotlib style file.
get_dataset_style(dataset[, style_file])	Retrieve the style information for the given dataset.
quickplot(cube, filename, plot_type, **kwargs)	Plot a cube using one of the iris.quickplot functions.
multi_dataset_scatterplot(x_data, y_data, …)	Plot a multi dataset scatterplot.
scatterplot(x_data, y_data, filepath, **kwargs)	Plot a scatterplot.

esmvaltool.diag_scripts.shared.plot.**get_dataset_style**(*dataset*, *style_file=None*)

Retrieve the style information for the given dataset.

esmvaltool.diag_scripts.shared.plot.**get_path_to_mpl_style**(*style_file=None*)

Get path to matplotlib style file.

esmvaltool.diag_scripts.shared.plot.**multi_dataset_scatterplot**(*x_data*, *y_data*, *datasets*, *filepath*, ***kwargs*)

Plot a multi dataset scatterplot.

Notes

Allowed keyword arguments:

- *mpl_style_file* (`str`): Path to the matplotlib style file.

- *dataset_style_file* (`str`): Path to the dataset style file.

- *plot_kwargs* (*array-like*): Keyword arguments for the plot (e.g. *label*, *makersize*, etc.).

- *save_kwargs* (`dict`): Keyword arguments for saving the plot.

- *axes_functions* (`dict`): Arbitrary functions for axes, i.e. *axes.set_title('title')*.

Parameters

- **x_data** (*array-like*) – x data of each dataset.

- **y_data** (*array-like*) – y data of each dataset.

- **datasets** (*array-like*) – Names of the datasets.

- **filepath** (*str*) – Path to which plot is written.

- ****kwargs** – Keyword arguments.

Raises

- **TypeError** – A non-valid keyword argument is given or *x_data*, *y_data*, *datasets* or (if given) *plot_kwargs* is not array-like.

- **ValueError** – *x_data*, *y_data*, *datasets* or *plot_kwargs* do not have the same size.

esmvaltool.diag_scripts.shared.plot.**quickplot**(*cube*, *filename*, *plot_type*, ***kwargs*)

Plot a cube using one of the iris.quickplot functions.

esmvaltool.diag_scripts.shared.plot.**scatterplot**(*x_data*, *y_data*, *filepath*, ***kwargs*)

Plot a scatterplot.

Notes

Allowed keyword arguments:

- *mpl_style_file* (*str*): Path to the matplotlib style file.

- *plot_kwargs* (*array-like*): Keyword arguments for the plot (e.g. *label*, *makersize*, etc.).

- *save_kwargs* (*dict*): Keyword arguments for saving the plot.

- *axes_functions* (*dict*): Arbitrary functions for axes, i.e. *axes.set_title('title')*.

Parameters

- **x_data** (*array-like*) – x data of each dataset.

- **y_data** (*array-like*) – y data of each dataset.

- **filepath** (*str*) – Path to which plot is written.

- ****kwargs** – Keyword arguments.

Raises

- **TypeError** – A non-valid keyword argument is given or *x_data*, *y_data* or (if given) *plot_kwargs* is not array-like.

- **ValueError** – *x_data*, *y_data* or *plot_kwargs* do not have the same size.

DIAGNOSTIC SCRIPTS

Various diagnostic packages exist as part of ESMValTool.

ESMValTool diagnostic scripts.

OCEAN DIAGNOSTICS TOOLKIT

Welcome to the API documentation for the ocean diagnostics tool kit. This toolkit is built to assist in the evaluation of models of the ocean.

This toolkit is part of ESMValTool v2.

Author: Lee de Mora (PML) ledm@pml.ac.uk

32.1 Maps diagnostics

Diagnostic to produce images of a map with coastlines from a cube. These plost show latitude vs longitude and the cube value is used as the colour scale.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

An approproate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_map:
    extract_levels:
      levels: [100., ]
        scheme: linear_extrap
    climate_statistics:
      operator: mean
```

Note that this recipe may not function on machines with no access to the internet, as cartopy may try to download the shapefiles. The solution to this issue is the put the relevant cartopy shapefiles on a disk visible to your machine, then link that path to ESMValTool via the *auxiliary_data_dir* variable. The cartopy masking files can be downloaded from:

```
https://www.naturalearthdata.com/downloads/
```

Here, cartopy uses the 1:10, physical coastlines and land files:

```
110m_coastline.dbf  110m_coastline.shp  110m_coastline.shx
110m_land.dbf  110m_land.shp  110m_land.shx
```

This tool is part of the ocean diagnostic tools package in the ESMValTool.

Author: Lee de Mora (PML) ledm@pml.ac.uk

Functions

[*main*](cfg)	Load the config file, and send it to the plot makers.
[*make_map_contour*](cfg, metadata, filename)	Make a simple contour map plot for an individual model.
[*make_map_plots*](cfg, metadata, filename)	Make a simple map plot for an individual model.
[*multi_model_contours*](cfg, metadata)	Make a contour map showing several models.

esmvaltool.diag_scripts.ocean.diagnostic_maps.**main**(*cfg*)

Load the config file, and send it to the plot makers.

> **Parameters cfg** (*dict*) – the opened global config dictionary, passed by ESMValTool.

esmvaltool.diag_scripts.ocean.diagnostic_maps.**make_map_contour**(*cfg*, *metadata*, *filename*)

Make a simple contour map plot for an individual model.

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionary, passed by ESMValTool.
>
> - **metadata** (*dict*) – the metadata dictionary
>
> - **filename** (*str*) – the preprocessed model file.

esmvaltool.diag_scripts.ocean.diagnostic_maps.**make_map_plots**(*cfg*, *metadata*, *filename*)

Make a simple map plot for an individual model.

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionary, passed by ESMValTool.
>
> - **metadata** (*dict*) – the metadata dictionary
>
> - **filename** (*str*) – the preprocessed model file.

esmvaltool.diag_scripts.ocean.diagnostic_maps.**multi_model_contours**(*cfg*, *metadata*)

Make a contour map showing several models.

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionary, passed by ESMValTool.
>
> - **metadata** (*dict*) – the metadata dictionary.

32.2 Model 1 vs Model 2 vs Observations diagnostics.

Diagnostic to produce an image showing four maps, based on a comparison of two differnt models results against an observational dataset. This process is often used to compare a new iteration of a model under development against a previous version of the same model. The four map plots are:

- Top left: model 1

- Top right: model 1 minus model 2

- Bottom left: model 2 minus obs

- Bottom right: model 1 minus obs

All four plots show latitude vs longitude and the cube value is used as the colour scale.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

An approproate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_map:
    extract_levels:
      levels:  [100., ]
      scheme: linear_extrap
    climate_statistics:
      operator: mean
```

This diagnostic also requires the `exper_model`, `exper_model` and `observational_dataset` keys in the recipe:

```
diagnostics:
  diag_name:
    ...
    scripts:
      Global_Ocean_map:
        script: ocean/diagnostic_maps_quad.py
        exper_model:  {Model 1 dataset details}
        control_model: {Model 2 dataset details}
        observational_dataset: {Observational dataset details}
```

This tool is part of the ocean diagnostic tools package in the ESMValTool, and was based on the plots produced by the Ocean Assess/Marine Assess toolkit.

Author: Lee de Mora (PML) ledm@pml.ac.uk

Functions

add_map_subplot(subplot, cube, nspace[, . . .])	Add a map subplot to the current pyplot figure.
main(cfg)	Load the config file, and send it to the plot maker.
multi_model_maps(cfg, input_files)	Make the four pane model vs model vs obs comparison plot.

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**add_map_subplot**(*subplot*, *cube*, *nspace*, *title=''*, *cmap=''*)

Add a map subplot to the current pyplot figure.

> **Parameters**
>
> - **subplot** (*int*) – The matplotlib.pyplot subplot number. (ie 221)
>
> - **cube** (*iris.cube.Cube*) – the iris cube to be plotted.
>
> - **nspace** (*numpy.array*) – An array of the ticks of the colour part.
>
> - **title** (*str*) – A string to set as the subplot title.
>
> - **cmap** (*str*) – A string to describe the matplotlib colour map.

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**main**(*cfg*)

    Load the config file, and send it to the plot maker.

        Parameters cfg (*dict*) – the opened global config dictionairy, passed by ESMValTool.

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**multi_model_maps**(*cfg*, *input_files*)

    Make the four pane model vs model vs obs comparison plot.

        Parameters

- **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.

- **input_files** (*dict*) – the metadata dictionairy

32.3 Model vs Observations maps Diagnostic.

Diagnostic to produce comparison of model and data. The first kind of image shows four maps and the other shows a scatter plot.

The four pane image is a latitude vs longitude figures showing:

- Top left: model

- Top right: observations

- Bottom left: model minus observations

- Bottom right: model over observations

The scatter plots plot the matched model coordinate on the x axis, and the observational dataset on the y coordinate, then performs a linear regression of those data and plots the line of best fit on the plot. The parameters of the fit are also shown on the figure.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

An approproate preprocessor for a 3D + time field would be:

```
preprocessors:
  prep_map:
    extract_levels:
      levels: [100., ]
      scheme: linear_extrap
    climate_statistics:
      operator: mean
    regrid:
      target_grid: 1x1
      scheme: linear
```

This tool is part of the ocean diagnostic tools package in the ESMValTool, and was based on the plots produced by the Ocean Assess/Marine Assess toolkit.

Author: Lee de Mora (PML) ledm@pml.ac.uk

Functions

add_linear_regression(plot_axes, arr_x, arr_y)	Add a straight line fit to an axis.
add_map_subplot(subplot, cube, nspace[, . . .])	Add a map subplot to the current pyplot figure.

continues on next page

Table 3 – continued from previous page

main(cfg)	Load the config file, and send it to the plot maker.
make_model_vs_obs_plots(cfg, metadata, . . .)	Make a figure showing four maps and the other shows a scatter plot.
make_scatter(cfg, metadata, model_filename, . . .)	Makes Scatter plots of model vs observational data.
rounds_sig(value[, sig])	Round a float to a specific number of sig.

esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs.**add_linear_regression**(*plot_axes*, *arr_x*, *arr_y*, *showtext=True*, *add_diagonal=False*, *extent=None*)

> Add a straight line fit to an axis.

>> **Parameters**

>>> - **plot_axes** (*matplotlib.pyplot.axes*) – The matplotlib axes on which to plot the linear regression.

>>> - **arr_x** (*numpy.array*) – The data for the x coordinate.

>>> - **arr_y** (*numpy array*) – The data for the y coordinate.

>>> - **showtext** (*bool*) – A flag to turn on or off the result of the fit on the plot.

>>> - **add_diagonal** (*bool*) – A flag to also add the 1:1 diagonal line to the figure

>>> - **extent** (*list of floats*) – The extent of the plot axes.

esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs.**add_map_subplot**(*subplot*, *cube*, *nspace*, *title=''*, *cmap=''*, *extend='neither'*, *log=False*)

> Add a map subplot to the current pyplot figure.

>> **Parameters**

>>> - **subplot** (*int*) – The matplotlib.pyplot subplot number. (ie 221)

>>> - **cube** (*iris.cube.Cube*) – the iris cube to be plotted.

>>> - **nspace** (*numpy.array*) – An array of the ticks of the colour part.

>>> - **title** (*str*) – A string to set as the subplot title.

>>> - **cmap** (*str*) – A string to describe the matplotlib colour map.

>>> - **extend** (*str*) – Contourf-coloring of values outside the levels range

>>> - **log** (*bool*) – Flag to plot the colour scale linearly (False) or logarithmically (True)

esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs.**main**(*cfg*)

> Load the config file, and send it to the plot maker.

>> **Parameters cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.

32.3. Model vs Observations maps Diagnostic. 363

`esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs.`**`make_model_vs_obs_plots`**(*cfg*, *metadata*, *model_filename*, *obs_filename*)

Make a figure showing four maps and the other shows a scatter plot.

The four pane image is a latitude vs longitude figures showing:

- Top left: model
- Top right: observations
- Bottom left: model minus observations
- Bottom right: model over observations

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.
> - **metadata** (*dict*) – the input files dictionairy
> - **model_filename** (*str*) – the preprocessed model file.
> - **obs_filename** (*str*) – the preprocessed observations file.

`esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs.`**`make_scatter`**(*cfg*, *metadata*, *model_filename*, *obs_filename*)

Makes Scatter plots of model vs observational data.

Make scatter plot showing the matched model and observational data with the model data as the x-axis coordinate and the observational data as the y-axis coordinate. A linear regression is also applied to the matched data and the result of the fit is shown on the figure.

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.
> - **metadata** (*dict*) – the input files dictionairy
> - **model_filename** (*str*) – the preprocessed model file.
> - **obs_filename** (*str*) – the preprocessed observations file.

`esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs.`**`rounds_sig`**(*value*, *sig=3*)

Round a float to a specific number of sig. figs. & return it as a string.

> **Parameters**
>
> - **value** (*float*) – The float that is to be rounded.
> - **sig** (*int*) – The number of significant figures.
>
> **Returns** The rounded output string.
>
> **Return type** str

32.4 Profile diagnostics.

Diagnostic to produce figure of the profile over time from a cube. These plost show cube value (ie temperature) on the x-axis, and depth/height on the y axis. The colour scale is the time series.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has a time component, and depth component, but no latitude or longitude coordinates.

An approproate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_profile:
    extract_volume:
      long1: 0.
      long2:  20.
      lat1:  -30.
      lat2:  30.
      z_min: 0.
      z_max: 3000.
    area_statistics:
      operator: mean
```

In order to add an observational dataset to the profile plot, the following arguments are needed in the diagnostic script:

```
diagnostics:
  diagnostic_name:
    variables:
      ...
    additional_datasets:
    - {observational dataset description}
    scripts:
      script_name:
        script: ocean/diagnostic_profiles.py
        observational_dataset: {observational dataset description}
```

This tool is part of the ocean diagnostic tools package in the ESMValTool.

Author: Lee de Mora (PML) ledm@pml.ac.uk

Functions

determine_profiles_str(cube)	Determine a string from the cube, to describe the profile.
main(cfg)	Run the diagnostics profile tool.
make_profiles_plots(cfg, metadata, filename)	Make a profile plot for an individual model.

esmvaltool.diag_scripts.ocean.diagnostic_profiles.**determine_profiles_str**(*cube*)
> Determine a string from the cube, to describe the profile.

>> **Parameters** **cube** (*iris.cube.Cube*) – the opened dataset as a cube.

>> **Returns** Returns a string which describes the profile.

>> **Return type** str

esmvaltool.diag_scripts.ocean.diagnostic_profiles.**main**(*cfg*)
> Run the diagnostics profile tool.

> Load the config file, find an observational dataset filename, pass loaded into the plot making tool.

> Parameters **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.

esmvaltool.diag_scripts.ocean.diagnostic_profiles.**make_profiles_plots**(*cfg*,
meta-
data,
file-
name,
obs_metadata={},
obs_filename='')

Make a profile plot for an individual model.

The optional observational dataset can also be added.

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.
> - **metadata** (*dict*) – The metadata dictionairy for a specific model.
> - **filename** (*str*) – The preprocessed model file.
> - **obs_metadata** (*dict*) – The metadata dictionairy for the observational dataset.
> - **obs_filename** (*str*) – The preprocessed observational dataset file.

32.5 Time series diagnostics

Diagnostic to produce figures of the time development of a field from cubes. These plost show time on the x-axis and cube value (ie temperature) on the y-axis.

Two types of plots are produced: individual model timeseries plots and multi model time series plots. The inidivual plots show the results from a single cube, even if this is a mutli-model mean made by the _multimodel.py preproccessor. The multi model time series plots show several models on the same axes, where each model is represented by a different line colour.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has a time component, no depth component, and no latitude or longitude coordinates.

An approproate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_timeseries_1:# For Global Volume Averaged
    volume_statistics:
      operator: mean
```

An approproate preprocessor for a 3D+time field at the surface would be:

```
prep_timeseries_2: # For Global surface Averaged
  extract_levels:
    levels:  [0., ]
    scheme: linear_extrap
  area_statistics:
    operator: mean
```

An approproate preprocessor for a 2D+time field would be:

```
prep_timeseries_2: # For Global surface Averaged
  area_statistics:
    operator: mean
```

This tool is part of the ocean diagnostic tools package in the ESMValTool.

Author: Lee de Mora (PML) ledm@pml.ac.uk

Functions

main(cfg)	Load the config file and some metadata, then pass them the plot making tools.
make_time_series_plots(cfg, metadata, filename)	Make a simple time series plot for an indivudual model 1D cube.
moving_average(cube, window)	Calculate a moving average.
multi_model_time_series(cfg, metadata)	Make a time series plot showing several preprocesssed datasets.
timeplot(cube, **kwargs)	Create a time series plot from the cube.

esmvaltool.diag_scripts.ocean.diagnostic_timeseries.**main**(*cfg*)
    Load the config file and some metadata, then pass them the plot making tools.

> **Parameters** **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.

esmvaltool.diag_scripts.ocean.diagnostic_timeseries.**make_time_series_plots**(*cfg*,
                                                                                metadata,
                                                                                filename)

    Make a simple time series plot for an indivudual model 1D cube.

    This tool loads the cube from the file, checks that the units are sensible BGC units, checks for layers, adjusts the titles accordingly, determines the ultimate file name and format, then saves the image.

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.
> - **metadata** (*dict*) – The metadata dictionairy for a specific model.
> - **filename** (*str*) – The preprocessed model file.

esmvaltool.diag_scripts.ocean.diagnostic_timeseries.**moving_average**(*cube*, *window*)

    Calculate a moving average.

    The window is a string which is a number and a measuremet of time. For instance, the following are acceptable window strings:

    - 5 days
    - 12 years
    - 1 month
    - 5 yr

    Also note the the value used is the total width of the window. For instance, if the window provided was '10 years', the the moving average returned would be the average of all values within 5 years of the central value.

In the case of edge conditions, at the start an end of the data, they only include the average of the data available. Ie the first value in the moving average of a `10 year` window will only include the average of the five subsequent years.

> **Parameters**
>
> > • **cube** (`iris.cube.Cube`) – Input cube
> >
> > • **window** (`str`) – A description of the window to use for the
>
> **Returns** A cube with the movinage average set as the data points.
>
> **Return type** iris.cube.Cube

esmvaltool.diag_scripts.ocean.diagnostic_timeseries.**multi_model_time_series**(*cfg*, *metadata*)

> Make a time series plot showing several preprocesssed datasets.
>
> This tool loads several cubes from the files, checks that the units are sensible BGC units, checks for layers, adjusts the titles accordingly, determines the ultimate file name and format, then saves the image.
>
> > **Parameters**
> >
> > > • **cfg** (`dict`) – the opened global config dictionairy, passed by ESMValTool.
> > >
> > > • **metadata** (`dict`) – The metadata dictionairy for a specific model.

esmvaltool.diag_scripts.ocean.diagnostic_timeseries.**timeplot**(*cube*, ***kwargs*)

> Create a time series plot from the cube.
>
> Note that this function simple does the plotting, it does not save the image or do any of the complex work. This function also takes and of the key word arguments accepted by the matplotlib.pyplot.plot function. These arguments are typically, color, linewidth, linestyle, etc. . .
>
> If there's only one datapoint in the cube, it is plotted as a horizontal line.
>
> > **Parameters** **cube** (`iris.cube.Cube`) – Input cube

32.6 Transects diagnostics

Diagnostic to produce images of a transect. These plost show either latitude or longitude against depth, and the cube value is used as the colour scale.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, and one of the latitude or longitude coordinates has been reduced to a single value.

An approproate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_transect:
    climate_statistics:
      operator: mean
    extract_transect: # Atlantic Meridional Transect
      latitude: [-50.,50.]
      longitude: 332.
```

This tool is part of the ocean diagnostic tools package in the ESMValTool.

Author: Lee de Mora (PML) ledm@pml.ac.uk

Functions

add_sea_floor(cube)	Add a simple sea floor line from the cube mask.
determine_set_y_logscale(cfg, metadata)	Determine whether to use a log scale y axis.
determine_transect_str(cube[, region])	Determine the Transect String.
main(cfg)	Load the config file and some metadata, then pass them the plot making tools.
make_cube_region_dict(cube)	Take a cube and return a dictionairy region: cube.
make_depth_safe(cube)	Make the depth coordinate safe.
make_transect_contours(cfg, metadata, file-name)	Make a contour plot of the transect for an indivudual model.
make_transects_plots(cfg, metadata, filename)	Make a simple plot of the transect for an indivudual model.
multi_model_contours(cfg, metadatas)	Make a multi model comparison plot showing several transect contour plots.
titlify(title)	Check whether a title is too long then add it to current figure.

esmvaltool.diag_scripts.ocean.diagnostic_transects.**add_sea_floor**(*cube*)
    Add a simple sea floor line from the cube mask.

>   **Parameters cube** (*iris.cube.Cube*) – Input cube to use to produce the sea floor.

esmvaltool.diag_scripts.ocean.diagnostic_transects.**determine_set_y_logscale**(*cfg*, *meta-data*)

    Determine whether to use a log scale y axis.

>   **Parameters**
>
>   • **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.
>
>   • **metadata** (*dict*) – The metadata dictionairy for a specific model.

>   **Returns** Boolean to flag whether to plot as a log scale.

>   **Return type** bool

esmvaltool.diag_scripts.ocean.diagnostic_transects.**determine_transect_str**(*cube*, *re-gion=''*)

    Determine the Transect String.

    Takes a guess at a string to describe the transect.

>   **Parameters cube** (*iris.cube.Cube*) – Input cube to use to determine the transect name.

esmvaltool.diag_scripts.ocean.diagnostic_transects.**main**(*cfg*)
    Load the config file and some metadata, then pass them the plot making tools.

>   **Parameters cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.

esmvaltool.diag_scripts.ocean.diagnostic_transects.**make_cube_region_dict**(*cube*)
    Take a cube and return a dictionairy region: cube.

    Each item in the dict is a layer with a separate cube for each layer. ie: cubes[region] = cube from specific region

    Cubes with no region component are returns as: cubes[''] = cube with no region component.

    This is based on the method diagnostics_tools.make_cube_layer_dict, however, it wouldn't make sense to look for depth layers here.

> > **Parameters cube** (`iris.cube.Cube`) – the opened dataset as a cube.
> >
> > **Returns** A dictionairy of layer name : layer cube.
> >
> > **Return type** dict

esmvaltool.diag_scripts.ocean.diagnostic_transects.**make_depth_safe**(*cube*)

> Make the depth coordinate safe.
>
> If the depth coordinate has a value of zero or above, we replace the zero with the average point of the first depth layer.
>
> > **Parameters cube** (`iris.cube.Cube`) – Input cube to make the depth coordinate safe
> >
> > **Returns** Output cube with a safe depth coordinate
> >
> > **Return type** iris.cube.Cube

esmvaltool.diag_scripts.ocean.diagnostic_transects.**make_transect_contours**(*cfg*, *metadata*, *filename*)

> Make a contour plot of the transect for an indivudual model.
>
> This tool loads the cube from the file, checks that the units are sensible BGC units, checks for layers, adjusts the titles accordingly, determines the ultimate file name and format, then saves the image.
>
> > **Parameters**
> >
> > - **cfg** (`dict`) – the opened global config dictionairy, passed by ESMValTool.
> > - **metadata** (`dict`) – The metadata dictionairy for a specific model.
> > - **filename** (`str`) – The preprocessed model file.

esmvaltool.diag_scripts.ocean.diagnostic_transects.**make_transects_plots**(*cfg*, *metadata*, *filename*)

> Make a simple plot of the transect for an indivudual model.
>
> This tool loads the cube from the file, checks that the units are sensible BGC units, checks for layers, adjusts the titles accordingly, determines the ultimate file name and format, then saves the image.
>
> > **Parameters**
> >
> > - **cfg** (`dict`) – the opened global config dictionairy, passed by ESMValTool.
> > - **metadata** (`dict`) – The metadata dictionairy for a specific model.
> > - **filename** (`str`) – The preprocessed model file.

esmvaltool.diag_scripts.ocean.diagnostic_transects.**multi_model_contours**(*cfg*, *metadatas*)

> Make a multi model comparison plot showing several transect contour plots.
>
> This tool loads several cubes from the files, checks that the units are sensible BGC units, checks for layers, adjusts the titles accordingly, determines the ultimate file name and format, then saves the image.
>
> > **Parameters**
> >
> > - **cfg** (`dict`) – the opened global config dictionairy, passed by ESMValTool.
> > - **metadatas** (`dict`) – The metadatas dictionairy for a specific model.

esmvaltool.diag_scripts.ocean.diagnostic_transects.**titlify**(*title*)
>   Check whether a title is too long then add it to current figure.

>>>   **Parameters** **title** (*str*) – The title for the figure.

32.7 Sea Ice Diagnostics.

Diagnostic to produce a series of images which are useful for evaluating the behaviour of the a sea ice model.

There are three kinds of plots shown here. 1. Sea ice Extent maps plots with a stereoscoic projection. 2. Maps plots of individual models ice fracrtion. 3. Time series plots for the total ice extent.

All three kinds of plots are made for both Summer and Winter in both the North and Southern hemisphere.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

This diagnostic takes data from either North or South hemisphere, and from either December-January-February or June-July-August. This diagnostic requires the data to be 2D+time, and typically expects the data field to be the sea ice cover. An approproate preprocessor would be:

```
preprocessors:
  timeseries_NHW_ice_extent: # North Hemisphere Winter ice_extent
    custom_order: true
    extract_time:
        start_year: 1960
        start_month: 12
        start_day: 1
        end_year: 2005
        end_month: 9
        end_day: 31
    extract_season:
      season: DJF
    extract_region:
      start_longitude: -180.
      end_longitude: 180.
      start_latitude: 0.
      end_latitude: 90.
```

Note that this recipe may not function on machines with no access to the internet, as cartopy may try to download the shapefiles. The solution to this issue is the put the relevant cartopy shapefiles on a disk visible to your machine, then link that path to ESMValTool via the *auxiliary_data_dir* variable. The cartopy masking files can be downloaded from:

```
https://www.naturalearthdata.com/downloads/
```

Here, cartopy uses the 1:10, physical coastlines and land files:

```
110m_coastline.dbf  110m_coastline.shp  110m_coastline.shx
110m_land.dbf  110m_land.shp  110m_land.shx
```

This tool is part of the ocean diagnostic tools package in the ESMValTool.

Author: Lee de Mora (PML) ledm@pml.ac.uk

Functions

agregate_by_season(cube)	Aggregate the cube into seasonal means.
calculate_area_time_series(cube, plot_type, . . .)	Calculate the area of unmasked cube cells.
create_ice_cmap([threshold])	Create colour map with ocean blue below a threshold and white above.
get_pole(cube)	Figure out the hemisphere and returns it as a string (North or South).
get_season(cube)	Return a climatological season time string.
get_time_string(cube)	Return a climatological season string in the format: "year season".
get_year(cube)	Return the cube year as a string.
main(cfg)	Load the config file and metadata, then pass them the plot making tools.
make_map_extent_plots(cfg, metadata, filename)	Make an extent map plot showing several times for an individual model.
make_map_plots(cfg, metadata, filename)	Make a simple map plot for an individual model.
make_polar_map(cube[, pole, cmap])	Make a polar stereoscopic map plot.
make_ts_plots(cfg, metadata, filename)	Make a ice extent and ice area time series plot for an individual model.

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**agregate_by_season**(*cube*)

Aggregate the cube into seasonal means.

Note that it is not currently possible to do this in the preprocessor, as the seasonal mean changes the cube units.

> **Parameters cube** (*iris.cube.Cube*) – Data Cube
>
> **Returns** Data Cube with the seasonal means
>
> **Return type** iris.cube.Cube

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**calculate_area_time_series**(*cube*, *plot_type*, *threshold*)

Calculate the area of unmasked cube cells.

Requires a cube with two spacial dimensions. (no depth coordinate).

> **Parameters**
>
> - **cube** (*iris.cube.Cube*) – Data Cube
>
> - **plot_type** (*str*) – The type of plot: ice extent or ice area
>
> - **threshold** (*float*) – The threshold for ice fraction (typically 15%)
>
> **Returns**
>
> - *numpy array* – An numpy array containing the time points.
>
> - *numpy.array* – An numpy array containing the total ice extent or total ice area.

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**create_ice_cmap**(*threshold=0.15*)

Create colour map with ocean blue below a threshold and white above.

> **Parameters threshold** (*float*) – The threshold for the line between blue and white.
>
> **Returns** The resulting colour map.
>
> **Return type** matplotlib.colors.LinearSegmentedColormap

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**get_pole**(*cube*)
    Figure out the hemisphere and returns it as a string (North or South).

> **Parameters cube** (*iris.cube.Cube*) – Data Cube
>
> **Returns** The hemisphere (North or South)
>
> **Return type** str

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**get_season**(*cube*)
    Return a climatological season time string.

> **Parameters cube** (*iris.cube.Cube*) – Data Cube
>
> **Returns** The climatological season as a string
>
> **Return type** str

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**get_time_string**(*cube*)
    Return a climatological season string in the format: "year season".

> **Parameters cube** (*iris.cube.Cube*) – Data Cube
>
> **Returns** The climatological season as a string
>
> **Return type** str

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**get_year**(*cube*)
    Return the cube year as a string.

> **Parameters cube** (*iris.cube.Cube*) – Data Cube
>
> **Returns** The year as a string
>
> **Return type** str

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**main**(*cfg*)
    Load the config file and metadata, then pass them the plot making tools.

> **Parameters cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**make_map_extent_plots**(*cfg, metadata, filename*)
    Make an extent map plot showing several times for an individual model.

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.
>
> - **metadata** (*dict*) – The metadata dictionairy for a specific model.
>
> - **filename** (*str*) – The preprocessed model file.

esmvaltool.diag_scripts.ocean.diagnostic_seaice.**make_map_plots**(*cfg, metadata, filename*)
    Make a simple map plot for an individual model.

> **Parameters**
>
> - **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.
>
> - **metadata** (*dict*) – The metadata dictionairy for a specific model.
>
> - **filename** (*str*) – The preprocessed model file.

`esmvaltool.diag_scripts.ocean.diagnostic_seaice.`**`make_polar_map`**(*cube*,
pole='North',
cmap='Blues_r')

Make a polar stereoscopic map plot.

The cube is the opened cube (two dimensional), pole is the polar region (North/South) cmap is the colourmap,

> **Parameters**
>> • **cube** (*iris.cube.Cube*) – Data Cube
>>
>> • **pole** (*str*) – The hemisphere
>>
>> • **cmap** (*str*) – The string describing the matplotlib colourmap.
>
> **Returns**
>> • *matplotlib.pyplot.figure* – The matplotlib figure where the map was drawn.
>>
>> • *matplotlib.pyplot.axes* – The matplotlib axes where the map was drawn.

`esmvaltool.diag_scripts.ocean.diagnostic_seaice.`**`make_ts_plots`**(*cfg*, *metadata*,
filename)

Make a ice extent and ice area time series plot for an individual model.

> **Parameters**
>> • **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.
>>
>> • **metadata** (*dict*) – The metadata dictionairy for a specific model.
>>
>> • **filename** (*str*) – The preprocessed model file.

32.8 Diagnostic tools

This module contains several python tools used elsewhere by the ocean diagnostics package.

This tool is part of the ocean diagnostic tools package in the ESMValTool.

Author: Lee de Mora (PML) ledm@pml.ac.uk

Functions

add_legend_outside_right(plot_details, ax1)	Add a legend outside the plot, to the right.
bgc_units(cube, name)	Convert the cubes into some friendlier units.
cube_time_to_float(cube)	Convert from time coordinate into decimal time.
decadal_average(cube)	Calculate the decadal_average.
folder(name)	Make a directory out of a string or list or strings.
get_array_range(arrays)	Determinue the minimum and maximum values of a list of arrays..
get_colour_from_cmap(number, total[, cmap])	Get a colour *number* of *total* from a cmap.
get_cube_range(cubes)	Determinue the minimum and maximum values of a list of cubes.
get_cube_range_diff(cubes)	Determinue the largest deviation from zero in an list of cubes.
get_decade(coord, value)	Determine the decade.
get_image_format(cfg[, default])	Load the image format from the global config file.
get_image_path(cfg, metadata[, prefix, . . .])	Produce a path to the final location of the image.
get_input_files(cfg[, index])	Load input configuration file as a Dictionairy.

continues on next page

Table 8 – continued from previous page

get_obs_projects()	Return a list of strings with the names of observations projects.
guess_calendar_datetime(cube)	Guess the cftime.datetime form to create datetimes.
load_thresholds(cfg, metadata)	Load the thresholds for contour plots from the config files.
make_cube_layer_dict(cube)	Take a cube and return a dictionairy layer:cube
match_model_to_key(model_type, cfg_dict, …)	Match up model or observations dataset dictionairies from config file.

esmvaltool.diag_scripts.ocean.diagnostic_tools.**add_legend_outside_right**(*plot_details,*
ax1,
col-
umn_width=0.1,
loc='right')

> Add a legend outside the plot, to the right.
>
> plot_details is a 2 level dict, where the first level is some key (which is hidden) and the 2nd level contains the keys: 'c': color 'lw': line width 'label': label for the legend. ax1 is the axis where the plot was drawn.
>
> > **Parameters**
> >
> > - **plot_details** (*dict*) – A dictionary of the plot details (color, linestyle, linewidth, label)
> >
> > - **ax1** (*matplotlib.pyplot.axes*) – The pyplot axes to add the
> >
> > - **column_width** (*float*) – The width of the legend column. This is used to adjust for longer words in the legends
> >
> > - **loc** (*string*) – Location of the legend. Options are "right" and "below".
> >
> > **Returns** A datetime creator function from cftime, based on the cube's calendar.
> >
> > **Return type** cftime.datetime

esmvaltool.diag_scripts.ocean.diagnostic_tools.**bgc_units**(*cube, name*)
> Convert the cubes into some friendlier units.
>
> This is because many CMIP standard units are not the standard units used by the BGC community (ie, Celsius is prefered over Kelvin, etc.)
>
> > **Parameters**
> >
> > - **cube** (*iris.cube.Cube*) – the opened dataset as a cube.
> >
> > - **name** (*str*) – The string describing the data field.
> >
> > **Returns** the cube with the new units.
> >
> > **Return type** iris.cube.Cube

esmvaltool.diag_scripts.ocean.diagnostic_tools.**cube_time_to_float**(*cube*)
> Convert from time coordinate into decimal time.
>
> Takes an iris time coordinate and returns a list of floats. :param cube: the opened dataset as a cube. :type cube: iris.cube.Cube
>
> > **Returns** List of floats showing the time coordinate in decimal time.
> >
> > **Return type** list

esmvaltool.diag_scripts.ocean.diagnostic_tools.**decadal_average**(*cube*)
> Calculate the decadal_average.

> Parameters **cube** (`iris.cube.Cube`) – The input cube
>
> Returns
>
> Return type iris.cube

esmvaltool.diag_scripts.ocean.diagnostic_tools.**folder**(*name*)

> Make a directory out of a string or list or strings.
>
> Take a string or a list of strings, convert it to a directory style, then make the folder and the string. Returns folder string and final character is always os.sep. ('/')
>
> > Parameters **name** (`list or string`) – A list of nested directories, or a path to a directory.
> >
> > Returns Returns a string of a full (potentially new) path of the directory.
> >
> > Return type str

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_array_range**(*arrays*)

> Determinue the minimum and maximum values of a list of arrays..
>
> > Parameters **arrays** (`list of numpy.array`) – A list of numpy.array.
> >
> > Returns A list of two values, the overall minumum and maximum values of the list of cubes.
> >
> > Return type list

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_colour_from_cmap**(*number,*
total,
cmap='jet')

> Get a colour *number* of *total* from a cmap.
>
> This function is used when several lines are created evenly along a colour map.
>
> > Parameters
> >
> > - **number** (`int, float`) – The
> > - **total** (`int`) –
> > - **cmap** (`string, plt.cm`) – A colour map, either by name (string) or from matplotlib

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_cube_range**(*cubes*)

> Determinue the minimum and maximum values of a list of cubes.
>
> > Parameters **cubes** (`list of iris.cube.Cube`) – A list of cubes.
> >
> > Returns A list of two values: the overall minumum and maximum values of the list of cubes.
> >
> > Return type list

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_cube_range_diff**(*cubes*)

> Determinue the largest deviation from zero in an list of cubes.
>
> > Parameters **cubes** (`list of iris.cube.Cube`) – A list of cubes.
> >
> > Returns A list of two values: the maximum deviation from zero and its opposite.
> >
> > Return type list

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_decade**(*coord, value*)

> Determine the decade.
>
> Called by iris.coord_categorisation.add_categorised_coord.

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_image_format**(*cfg,* *de-*
fault='png')

> Load the image format from the global config file.

Current tested options are svg, png.

The cfg is the opened global config. The default format is used if no specific format is requested. The default is set in the user config.yml Individual diagnostics can set their own format which will supercede the main config.yml.

> **Parameters** **cfg** (`dict`) – the opened global config dictionairy, passed by ESMValTool.
>
> **Returns** The image format extention.
>
> **Return type** str

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_image_path**(*cfg*, *metadata*, *prefix='diag'*, *suffix='image'*, *metadata_id_list='default'*)

Produce a path to the final location of the image.

The cfg is the opened global config, metadata is the metadata dictionairy (for the individual dataset file)

> **Parameters**
>
> - **cfg** (`dict`) – the opened global config dictionairy, passed by ESMValTool.
> - **metadata** (`dict`) – The metadata dictionairy for a specific model.
> - **prefix** (`str`) – A string to prepend to the image basename.
> - **suffix** (`str`) – A string to append to the image basename
> - **metadata_id_list** (`list`) – A list of strings to add to the file path. It loads these from the cfg.
>
> **Returns** The ultimate image path
>
> **Return type** str

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_input_files**(*cfg*, *index=''*)

Load input configuration file as a Dictionairy.

Get a dictionary with input files from the metadata.yml files. This is a wrappper for the _get_input_data_files function from diag_scripts.shared._base.

> **Parameters**
>
> - **cfg** (`dict`) – the opened global config dictionairy, passed by ESMValTool.
> - **index** (`int`) – the index of the file in the cfg file.
>
> **Returns** A dictionairy of the input files and their linked details.
>
> **Return type** dict

esmvaltool.diag_scripts.ocean.diagnostic_tools.**get_obs_projects**()

Return a list of strings with the names of observations projects.

Please keep this list up to date, or replace it with something more sensible.

> **Returns** Returns a list of strings of the various types of observational data.
>
> **Return type** list

esmvaltool.diag_scripts.ocean.diagnostic_tools.**guess_calendar_datetime**(*cube*)

Guess the cftime.datetime form to create datetimes.

> **Parameters** **cube** (`iris.cube.Cube`) – the opened dataset as a cube.
>
> **Returns** A datetime creator function from cftime, based on the cube's calendar.

> **Return type** cftime.datetime

esmvaltool.diag_scripts.ocean.diagnostic_tools.**load_thresholds**(*cfg*, *metadata*)
> Load the thresholds for contour plots from the config files.

> > **Parameters**

> > > • **cfg** (*dict*) – the opened global config dictionairy, passed by ESMValTool.

> > > • **metadata** (*dict*) – the metadata dictionairy

> > **Returns** List of thresholds

> > **Return type** list

esmvaltool.diag_scripts.ocean.diagnostic_tools.**make_cube_layer_dict**(*cube*)
> Take a cube and return a dictionairy layer:cube

> Each item in the dict is a layer with a separate cube for each layer. ie: cubes[depth] = cube from specific layer

> Cubes with no depth component are returned as dict, where the dict key is a blank empty string, and the value is the cube.

> > **Parameters** **cube** (*iris.cube.Cube*) – the opened dataset as a cube.

> > **Returns** A dictionairy of layer name : layer cube.

> > **Return type** dict

esmvaltool.diag_scripts.ocean.diagnostic_tools.**match_model_to_key**(*model_type*,
cfg_dict, *input_files_dict*)
> Match up model or observations dataset dictionairies from config file.

> This function checks that the control_model, exper_model and observational_dataset dictionairies from the recipe are matched with the input file dictionairy in the cfg metadata.

> > **Parameters**

> > > • **model_type** (*str*) – The string model_type to match (only used in debugging).

> > > • **cfg_dict** (*dict*) – the config dictionairy item for this model type, parsed directly from the diagnostics/ scripts, part of the recipe.

> > > • **input_files_dict** (*dict*) –

> > > **The input file dictionairy, loaded directly from the get_input_files**() function, in diagnostics_tools.py.

> > **Returns** A dictionairy of the input files and their linked details.

> > **Return type** dict

Part X

Frequently Asked Questions

IS THERE A MAILING LIST?

Yes, you can subscribe to the ESMValTool user mailing list and join the discussion on general topics (installation, configuration, etc). See *User mailing list*.

WHAT IS YAML?

While `.yaml` or `.yml` is a relatively common format, users may not have encountered this language before. The key information about this format is:

- yaml is a human friendly markup language;

- yaml is commonly used for configuration files (gradually replacing the venerable `.ini`);

- the syntax is relatively straightforward;

- indentation matters a lot (like `Python`)!

- yaml is case sensitive;

More information can be found in the yaml tutorial and yaml quick reference card. ESMValTool uses the yamllint linter tool to check recipe syntax.

RE-RUNNING DIAGNOSTICS

If a diagnostic fails, you will get the message

```
INFO    To re-run this diagnostic script, run:
```

If you run the command in the stdout you will be able to re-run the diagnostic without having to re-run the whole preprocessor. If you add the `-f` argument (available only for Python diagnostics, check your options with `--help`) that will force an overwrite, and it will delete not just the failed diagnostic, but the contents of its `work_dir` and `plot_dir` directories - this is useful when needing to redo the whole work. Adding `-i` or `--ignore-existing` will not delete any existing files, and it can be used to skip work that was already succesfully, provided that the diagnostic script supports this.

THIRTYSIX

ENTER INTERACTIVE MODE WITH IPYTHON

Sometimes it is useful to enter an interactive session to have a look what's going on. Insert a single line in the code where you want to enter IPython: `import IPython; IPython.embed()`

This is a useful functionality because it allows the user to *fix* things on-the-fly and after quitting the Ipython console, code execution continues as per normal.

USE MULTIPLE CONFIG-USER.YML FILES

The user selects the configuration yaml file at run time. It's possible to have several configurations files. For instance, it may be practical to have one config file for debugging runs and another for production runs.

CREATE A SYMBOLIC LINK TO THE LATEST OUTPUT DIRECTORY

When running multiple times the same recipe, the tool creates separate output directories sorted by the time tag that they were created at; sometimes, when running quite a few times, it is not straightforward to detect which one is the *latest* output directory, so a symbolic link attached to it would make things more clear e.g.:

```
recipe_example_20190905_163431
recipe_example_20190905_163519
recipe_example_latest -> recipe_example_20190905_163519
```

You can do that by running the tool using the latest output as basis and creating a symbolic link to it so it gets picked up at every re-run iteration:

```
esmvaltool -c ~/config-user.yml recipe_example.yml; \
ln -sfT $(ls -1d ~/esmvaltool_output/recipe_example_* | tail -1) ~/esmvaltool_output/
→recipe_example_latest
```

Part XI

Changelog

V2.0.0

This release includes

39.1 Bug fixes

- Fix pep8-naming errors and fix zmnam diagnostic (#1702) Bouwe Andela
- Fix keyword argument in cmorize_obs (#1721) Mattia Righi
- Fixed JMA-TRANSCOM CMORizer (#1735) Manuel Schlund
- Fix bug in extract_doi_value (#1734) bascrezee
- Fix small errors in the arctic_ocean diagnostic (#1722) Nikolay Koldunov
- Flatten ancestor lists for diag_spei.R and diag_spi.R. (#1745) katjaweigel
- Fix for recipe_ocean_ice_extent.yml (#1744) Mattia Righi
- Fix recipe_combined_indices.yml provenance (#1746) Javier Vegas-Regidor
- Fix provenance in recipe_multimodel_products (#1747) Javier Vegas-Regidor
- Exclude FGOALS-g2 due to ESMValCore issue #728 (#1749) Mattia Righi
- Fix recipe_modes_of_variability (#1753) Javier Vegas-Regidor
- Flatten lists for ancestors for hyint to prevent nested lists. (#1752) katjaweigel
- Fix bug in cmorize_obs_eppley_vgpm_modis.py (#1729) (#1759) Tomas Lovato
- Correct mip for clltkisccp in example derive preprocessor recipe (#1768) Bouwe Andela
- Update date conversion in recipe_hype.yml (#1769) Bouwe Andela
- Fix recipe_correlation.yml (#1767) Bouwe Andela
- Add attribute positive: down to plev coordinate in ERA-Interim CMORizer (#1771) Bouwe Andela
- Fix sispeed in recipe_preprocessor_derive_test (#1772) Javier Vegas-Regidor
- Fix extreme events and extreme index ancestors (#1774) katjaweigel
- Correct date in output filenames of ERA5 CMORizer recipe (#1773) Bouwe Andela
- Exclude WOA from multi-model stats in recipe_ocean_bgc (#1778) Mattia Righi

39.2 Diagnostics

- Enhancement of the hyint recipe to include etccdi indices (#1133) Enrico Arnone
- Add lazy regridding for wflow diagnostic (#1630) Bouwe Andela
- Miles default domains to include lat=0 (#1626) Jost von Hardenberg
- Miles: selection of reference dataset based on experiment (#1632) Jost von Hardenberg
- New recipe/diagnostic: recipe_li17natcc.yml for Axels GMD Paper (#1567) katjaweigel
- New recipe/diagnostics: recipe_deangelis_for_gmdpart4.yml for Axels GMD Paper (#1576) katjaweigel
- EWaterCycle: Add recipe to prepare input for LISFLOOD (#1298) Stefan Verhoeven
- Use area weighted regridding in wflow diagnostic (#1643) Bouwe Andela
- Workaround for permetrics recipe until Iris3 (#1674) Mattia Righi
- C3S_511_MPQB_bas-features (#1465) bascrezee
- Additional Land perfmetrics (#1641) Bettina Gier
- Necessary diagnostic from eyring06jgr for the release of version2 (#1686) Birgit Hassler
- Drought characteristics based on Martin2018 and SPI for gmd paper (#1689) katjaweigel
- Additional features and bugfixes for recipe anav13clim (#1723) Bettina Gier
- Gmd laueretal2020 revisions (#1725) Axel Lauer
- Wenzel16nature (#1692) zechlau
- Add mask albedolandcover (#1673) bascrezee
- IPCC AR5 fig. 9.3 (seasonality) (#1726) Axel Lauer
- Added additional emergent constraints on ECS (#1585) Manuel Schlund
- A diagnostic to evaluate the turnover times of land ecosystem carbon (#1395) koir-su
- Removed multi_model_statistics step in recipe_oceans_example.yml as a workaround (#1779) Valeriu Predoi

39.3 Documentation

- Extend getting started instructions to obtain config-user.yml (#1642) Peter Kalverla
- Extend information about native6 support on RTD (#1652) Peter Kalverla
- Update citation of ESMValTool paper in the doc (#1664) Mattia Righi
- Updated references to documentation (now docs.esmvaltool.org) (#1679) Axel Lauer
- Replace dead link with ESGF link. (#1681) Mattia Righi
- Add all European grants to Zenodo (#1682) Bouwe Andela
- Update Sphinx to v3 or later (#1685) Bouwe Andela
- Small fix to number of models in ensclus documentation (#1691) Jost von Hardenberg
- Move draft_release_notes.py from ESMValCore to here and update (#1701) Bouwe Andela
- Improve the installation instructions (#1634) Valeriu Predoi
- Improve description of how to implement provenance in diagnostic (#1750) SarahAlidoost

- Update command line interface documentation and add links to ESMValCore configuration documentation (#1776) Bouwe Andela

- Documentation on how to find shapefiles for hydrology recipes (#1777) Jaro Camphuijsen

39.4 Improvements

- Pin flake8<3.8.0 (#1635) Valeriu Predoi

- Update conda package path in more places (#1636) Bouwe Andela

- Remove curly brackets around issue number in pull request template (#1637) Bouwe Andela

- Fix style issue in test (#1639) Bouwe Andela

- Update Codacy badges (#1662) Bouwe Andela

- Support extra installation methods in R (#1360) Javier Vegas-Regidor

- Add ncdf4.helpers package as a dependency again (#1678) Bouwe Andela

- Speed up conda installation (#1677) Bouwe Andela

- Update CMORizers and recipes for ESMValCore v2.0.0 (#1699) SarahAlidoost

- Update setup.py for PyPI package (#1700) Bouwe Andela

- Cleanup recipe headers before the release (#1740) Mattia Righi

- Add colortables as esmvaltool subcommand (#1666) Javier Vegas-Regidor

- Increase version to v2.0.0 (#1756) Bouwe Andela

- Update job script (#1757) Mattia Righi

- Read authors and description from .zenodo.json (#1758) Bouwe Andela

- Update docker recipe to install from source (#1651) Javier Vegas-Regidor

39.5 Observational and re-analysis dataset support

- Cmorize aphro ma (#1555) mwjury

- Respectable testing for cmorizers/obs/utilities.py and cmorizers/obs/cmorize_obs.py (#1517) Valeriu Predoi

- Fix start year in recipe_check_obs (#1638) Mattia Righi

- Cmorizer for the PERSIANN-CDR precipitation data (#1633) Birgit Hassler

- Cmorize eobs (#1554) mwjury

- Update download cds satellite lai fapar (#1654) bascrezee

- Added monthly mean vars (ta, va, zg) to era5 cmorizer via recipe (#1644) Evgenia Galytska

- Make format time check more flexible (#1661) Mattia Righi

- Exclude od550lt1aer from recipe_check_obs.yml (#1720) Mattia Righi

- PERSIANN-CDR cmorizer update: adding the capability to save monthly mean files (#1728) Birgit Hassler

- Add standard_name attribute to lon and lat in cmorize_obs_esacci_oc.py (#1760) Tomas Lovato

- Allow for incomplete months on daily frequency in cmorizer ncl utilities (#1754) Mattia Righi

- Fix AURA-TES cmorizer (#1766) Mattia Righi

V2.0.0B4

This release includes

40.1 Bug fixes

- Fix HALOE plev coordinate (#1590) Mattia Righi
- Fix tro3 units in HALOE (#1591) Mattia Righi

40.2 Diagnostics

- Applicate sea ice negative feedback (#1299) Javier Vegas-Regidor
- Add Russell18jgr ocean diagnostics (#1592) Bouwe Andela
- Refactor marrmot recipe and diagnostic to use ERA5 daily data made by new cmorizer (#1600) SarahAlidoost
- In recipe_wflow, use daily ERA5 data from the new cmorizer. (#1599) Peter Kalverla
- In wflow diagnostic, calculate PET after(!) interpolation and lapse rate correction (#1618) Jerom Aerts
- Fixed wenz14jgr (#1562) zechlau
- Update portrait_plot.ncl (#1625) Bettina Gier

40.3 Documentation

- Restructure documentation (#1587) Bouwe Andela
- Add more links to documentation (#1595) Bouwe Andela
- Update links in readme (#1598) Bouwe Andela
- Minor improvements to installation documentation (#1608) Bouwe Andela
- Add info for new mailing list to documentation. (#1607) Björn Brötz
- Update making a release documentation (#1627) Bouwe Andela

40.4 Improvements

- Avoid broken pytest-html plugin (#1583) Bouwe Andela

- Remove reference section in config-references.yml (#1545) SarahAlidoost

- Various improvements to development infrastructure (#1570) Bouwe Andela

- Install scikit-learn from conda, remove libunwind as a direct dependency (#1611) Valeriu Predoi

- Create conda subpackages and enable tests (#1624) Bouwe Andela

40.5 Observational and re-analysis dataset support

- Cmorizer for HALOE (#1581) Mattia Righi

- Add CMORizer for CT2019 (#1604) Manuel Schlund

For older releases, see the release notes on https://github.com/ESMValGroup/ESMValTool/releases.

Part XII

Indices and tables

- genindex

- search

PYTHON MODULE INDEX

e