ESMValTool User's and Developer's Guide

Release 2.15.0.dev17+g9d3e452cd

ESMValTool Development Team

May 08, 2026

ESMVALTOOL

I Basic documentation schema 3

II Learning resources: 7

III How to contribute 11

IV Get in touch! 15

1 Introduction 19

2 What ESMValTool can do for you 21

3 Gallery 25

4 Getting started 69

5 Recipes 83

6 ESMValTool Code API Documentation 543

7 Obtaining input data 619

8 Making a recipe or diagnostic 627

9 Contributing to the community 633

10 Utilities 681

11 Frequently Asked Questions 695

12 Changelog 699

V Indices and tables 743

Python Module Index 747

Index 749

ESMValTool is a community diagnostics and performance metrics tool for the evaluation of Earth System Models (ESMs) that allows for routine comparison of models and observations. It includes a large collection of community recipes and observation data formatters to CMOR standards.

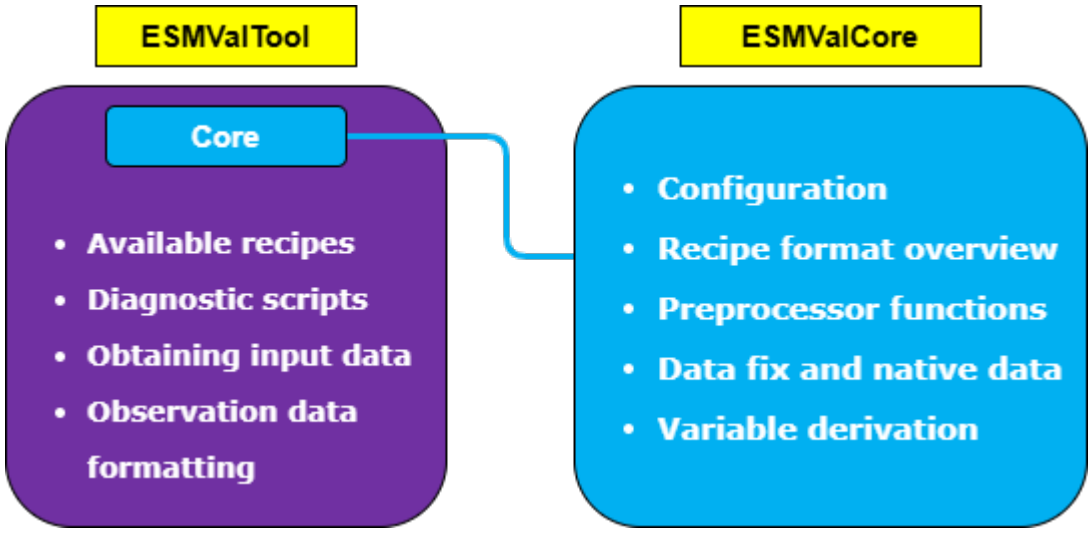
ESMValCore is a software package which provides the core functionality for ESMValTool. It is a workflow to find CMIP data, and apply commonly used pre-processing functions.

To get a first impression of what ESMValTool and ESMValCore can do for you, have a look at our blog posts Analysis-ready climate data with ESMValCore and ESMValTool: Recipes for solid climate science.

Part I

Basic documentation schema

This gives a brief idea of topics in each of the packages to help find information. As ESMValTool encompasses ESMValCore there will be some overlap between them. The documentation navigation to the left provides more detailed information about ESMValTool. ESMValCore documentation is also available. ESMValCore can be used without ESMValTool such as in Jupyter notebooks.



Part II

Learning resources:

A tutorial is available on https://tutorial.esmvaltool.org.

A series of video lectures has been created by ACCESS-NRI. While these are tailored for ACCESS users, they are still very informative.

Part III

How to contribute

Refer to ESMValTool *contributing to the community* for a guide on contributing recipes and diagnostics.

Refer to ESMValCore contributing for information on contributing code.

Part IV

Get in touch!

Contact information is available *here*.

CHAPTER

ONE

INTRODUCTION

1.1 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.

1.2 Support

Support for ESMValTool can be found in ESMValTool Discussions page where users can open an issue and a member of the User Engagement Team of ESMValTool will reply as soon as possible. This is open for all general and technical questions on the ESMValTool: installation, application, development, or any other question or comment you may have.

1.2.1 User mailing list

Subscribe to the ESMValTool announcements mailing list esmvaltool@listserv.dfn.de to stay up to date about new releases, monthly online meetings, upcoming workshops, and trainings.

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 a public archive online.

1.2.2 Monthly meetings

We have monthly online meetings using zoom, anyone with an interest in the ESMValTool is welcome to join these meetings to connect with the community. These meetings are always announced in a discussion on the ESMValTool Community repository and on the *mailing-list*.

1.2.3 Development team

ESMValTool is developed by a community of scientists and research software engineers from multiple institutes. An overview of the team is available on the development team page.

1.2.4 Recipes and diagnostics

Contacts for specific diagnostic sets are the respective authors, as listed in the corresponding *recipe and diagnostic documentation* and in the source code.

1.3 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.

CHAPTER

TWO

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

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

2.1 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);

2.2 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;

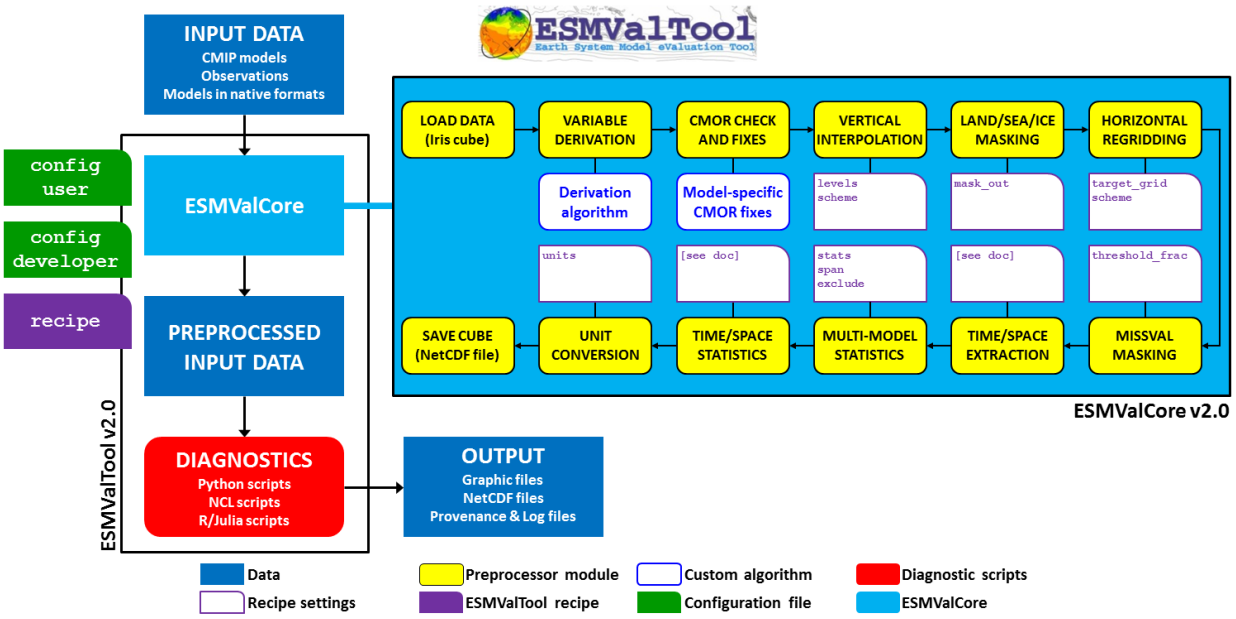


Fig. 1: Schematic of the system architecture.

2.3 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).

2.4 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 variable to be derived in the diagnostics section (as any other standard variable, just setting `derive: true`).

2.5 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 parameter value); the preprocessing order is very important since a number of steps depend on prior execution 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.

2.6 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.

CHAPTER

THREE

GALLERY

This section shows example plots produced by ESMValTool. For more information, follow the links in the figure captions.A website displaying results produced with the latest release of ESMValTool for all available recipes can be accessed here.

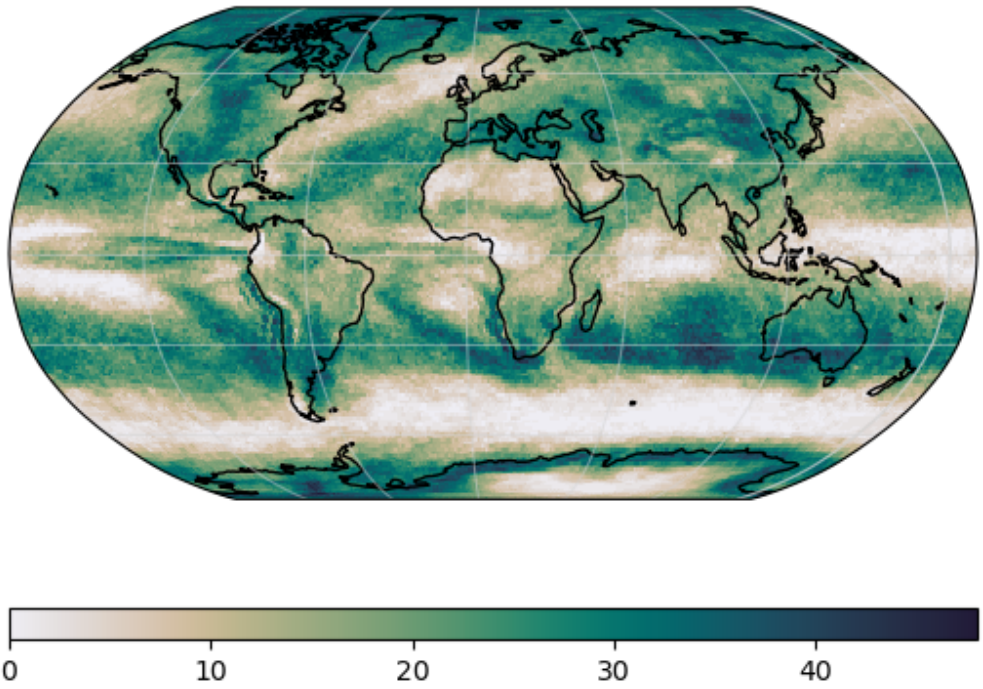
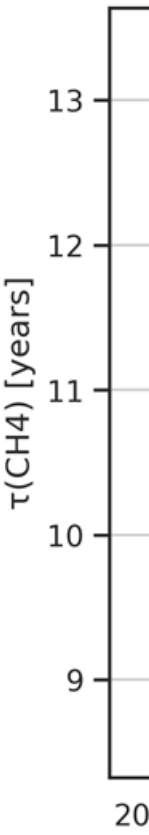


Fig. 1: *Example of the number of occurrences with consecutive dry days of more than five days in the period …*



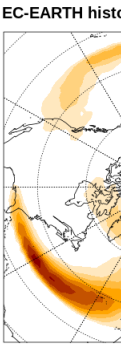
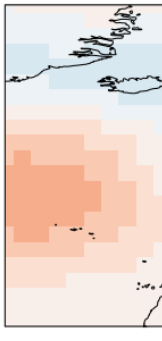


Fig. 56: *Blockin*



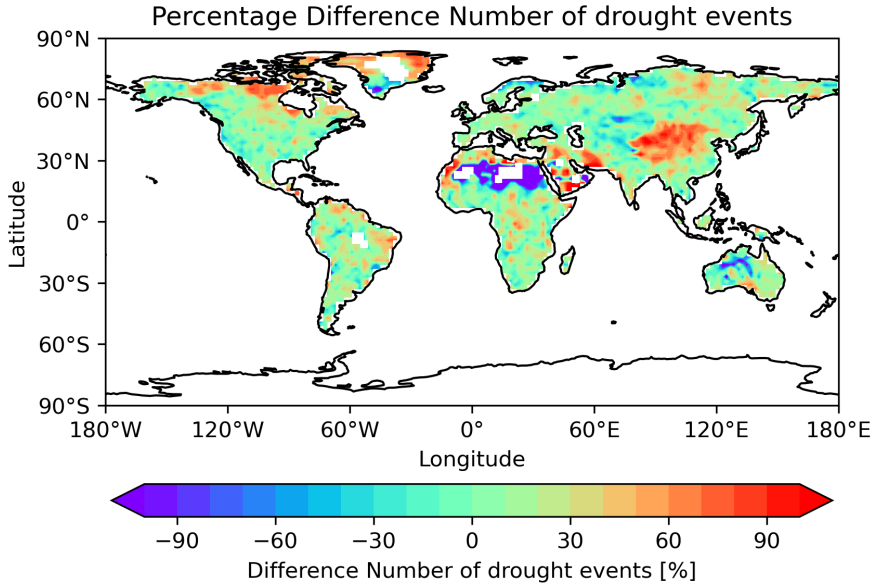


Fig. 2: *Global map of the percentage difference between multi-model mean of 15 CMIP models and the CRU data …*

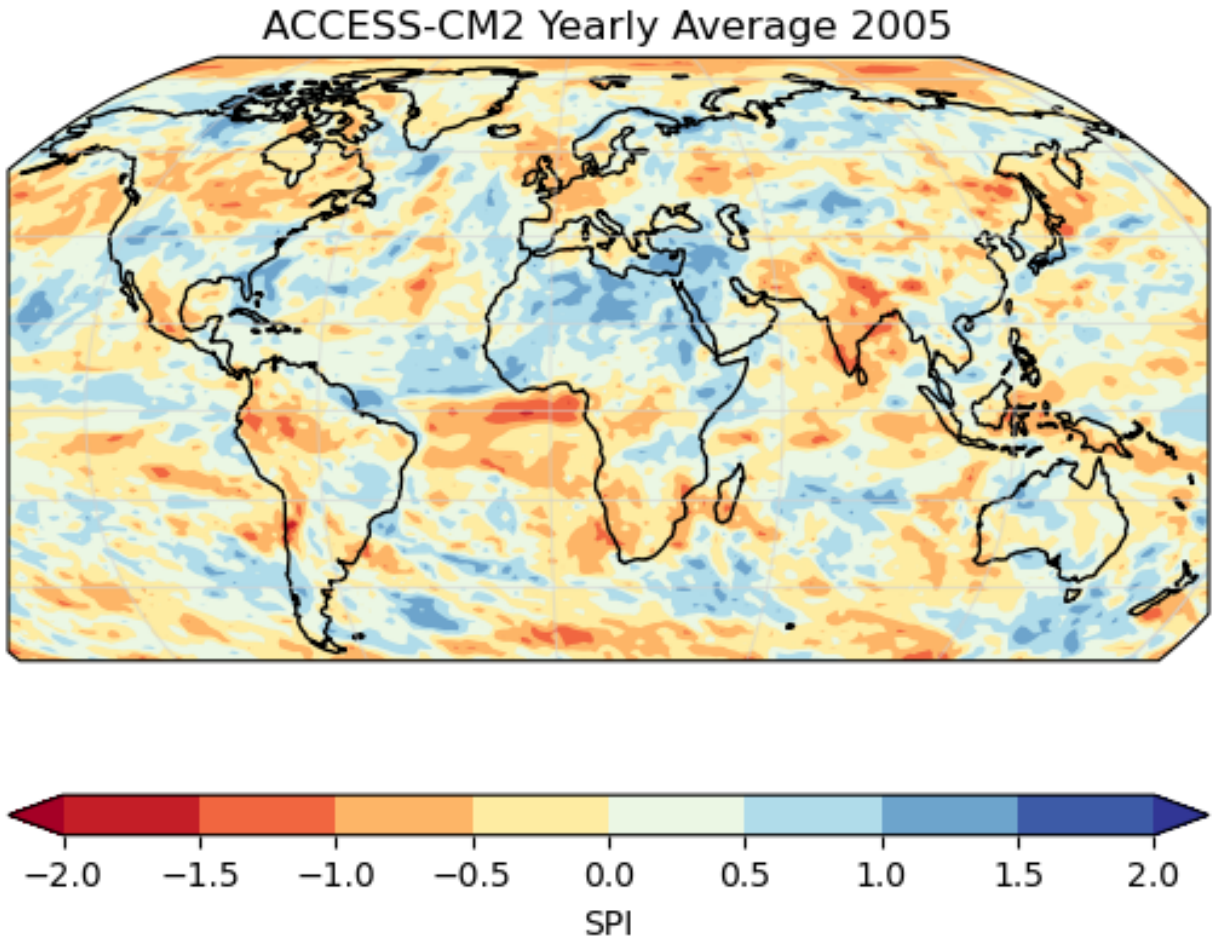


Fig. 3: *Example plot of SPI averaged over the year 2005. The reference period for index calibration is 2000-…*

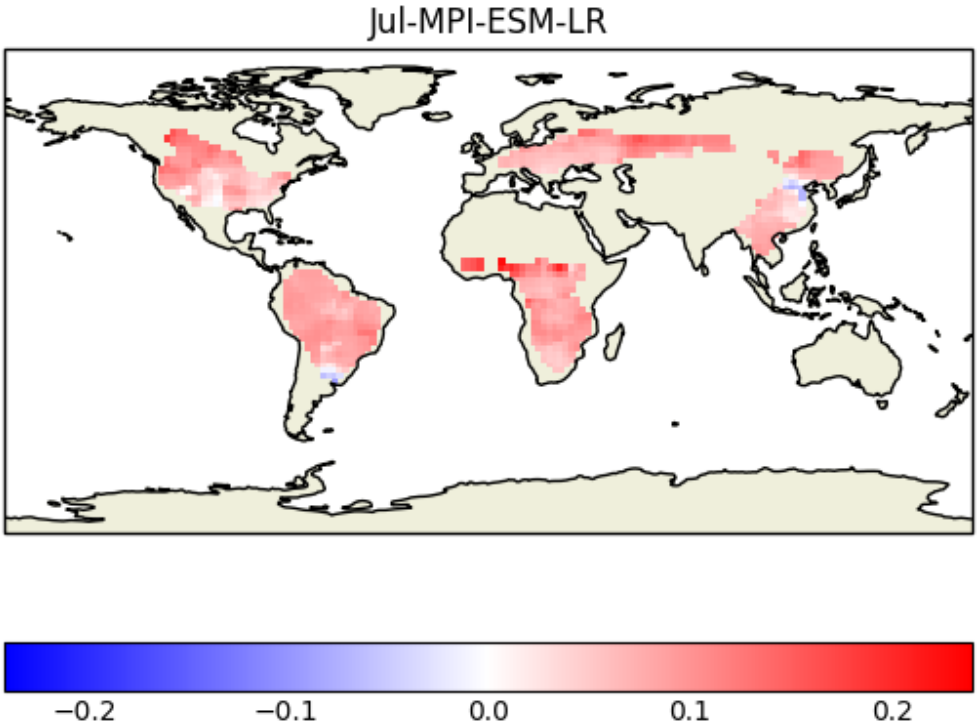


Fig. 4: *Example of albedo change from tree to crop and grass for the CMIP5 model MPI-ESM-LR derived for the …*

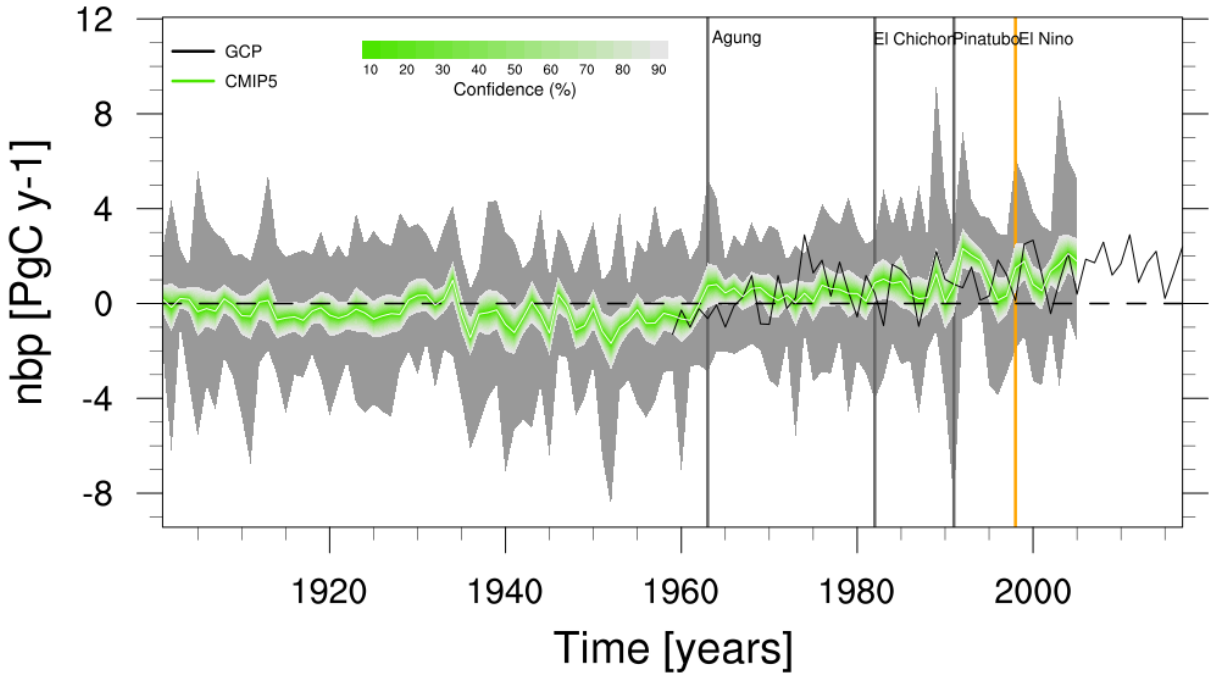


Fig. 5: *Time series of global net biome productivity (NBP) over the period 1901-2005. Similar to Anav et al.…*

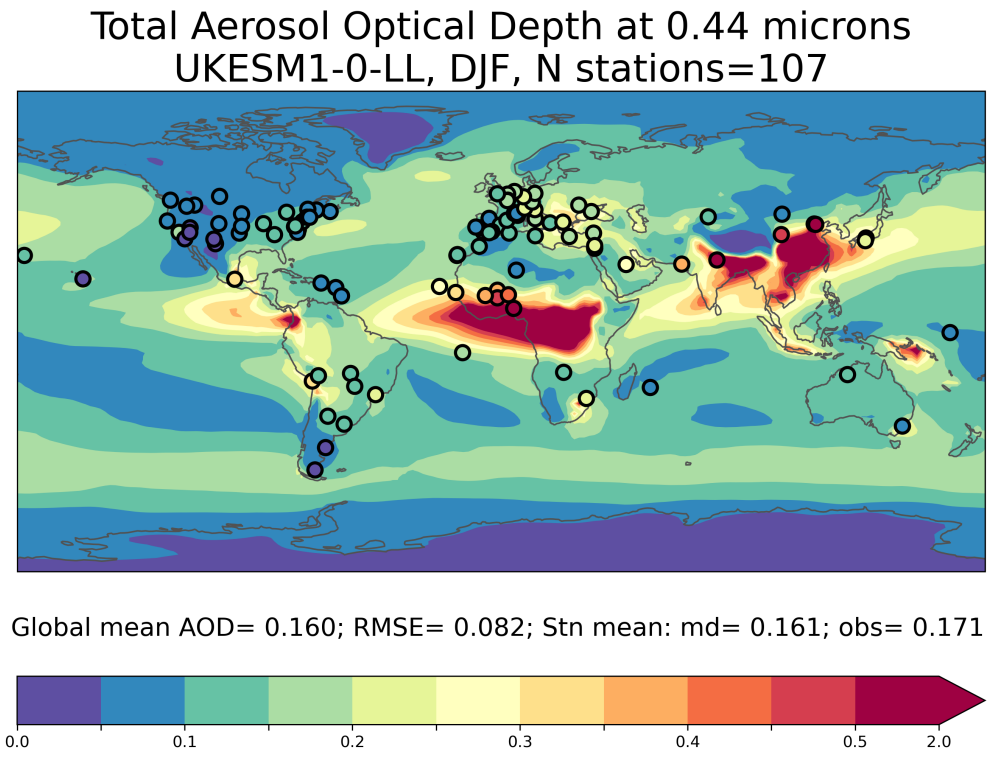


Fig. 6: *Evaluation of AOD at 440 nm from UKESM1 historical ensemble member r1i1p1f2 against the AeroNET clim...*

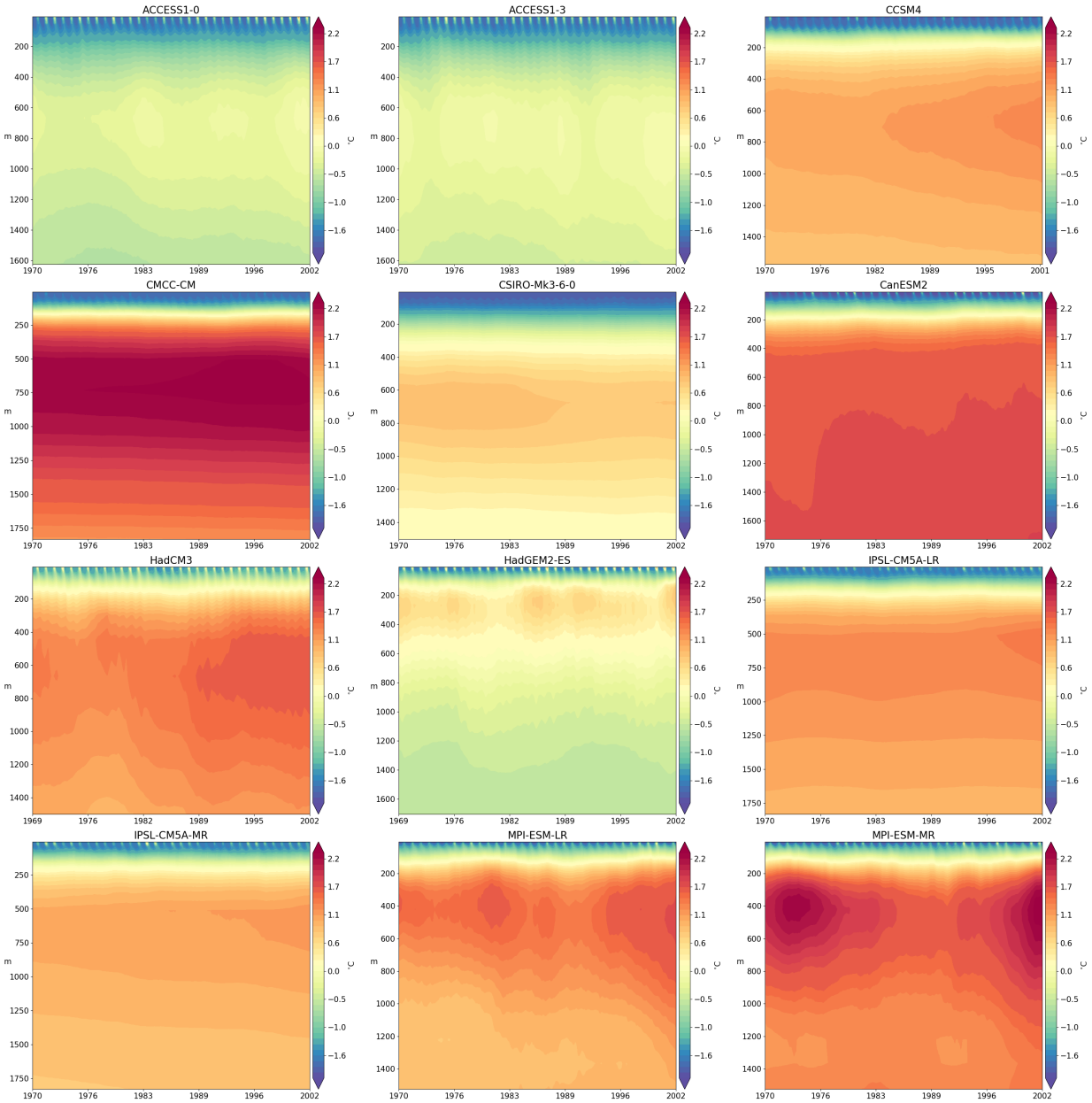


Fig. 7: *Hovmoller diagram of monthly spatially averaged potential temperature in the Eurasian Basin of the A…*

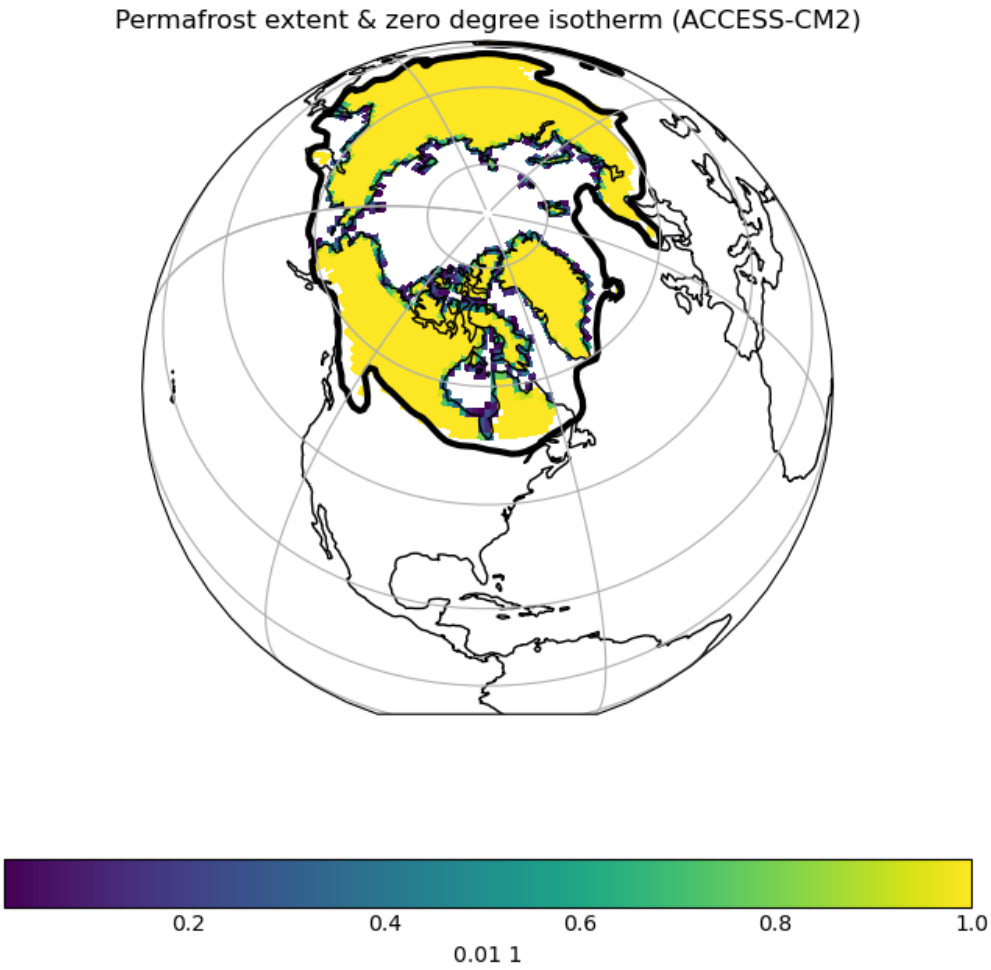


Fig. 8: *Permafrost extent and zero degC isotherm, showing North America*

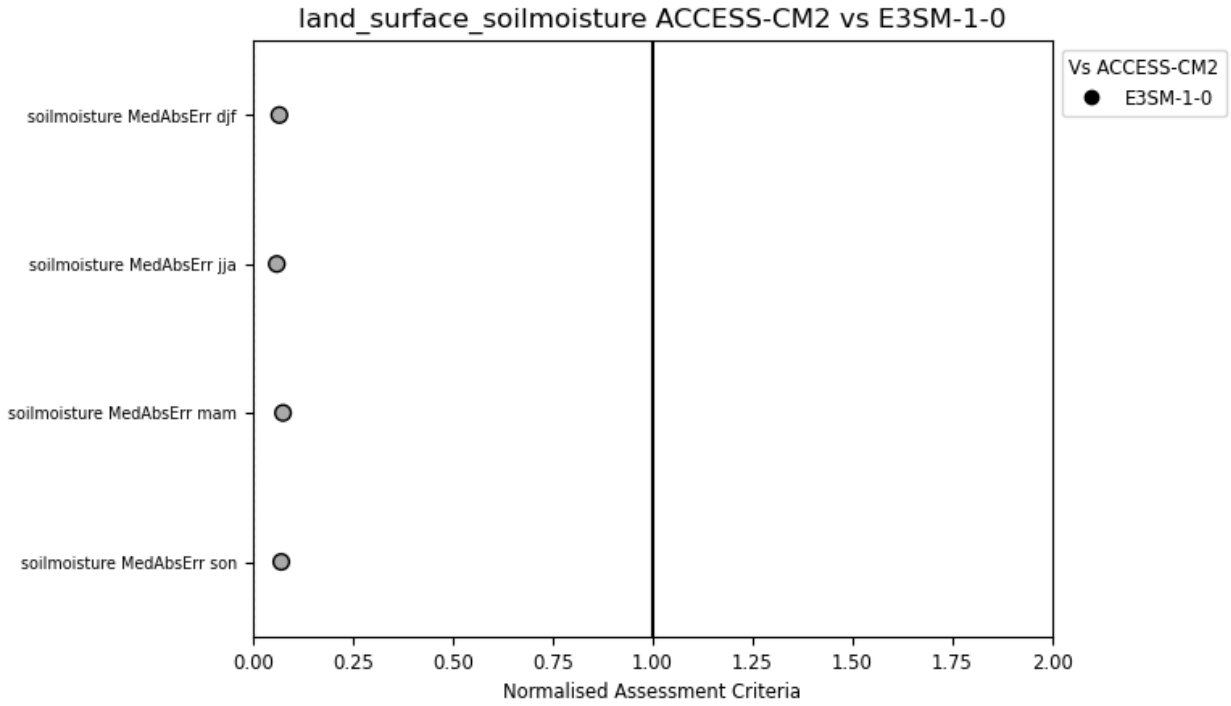


Fig. 9: *Normalised metrics plot comparing a control and experiment simulation*

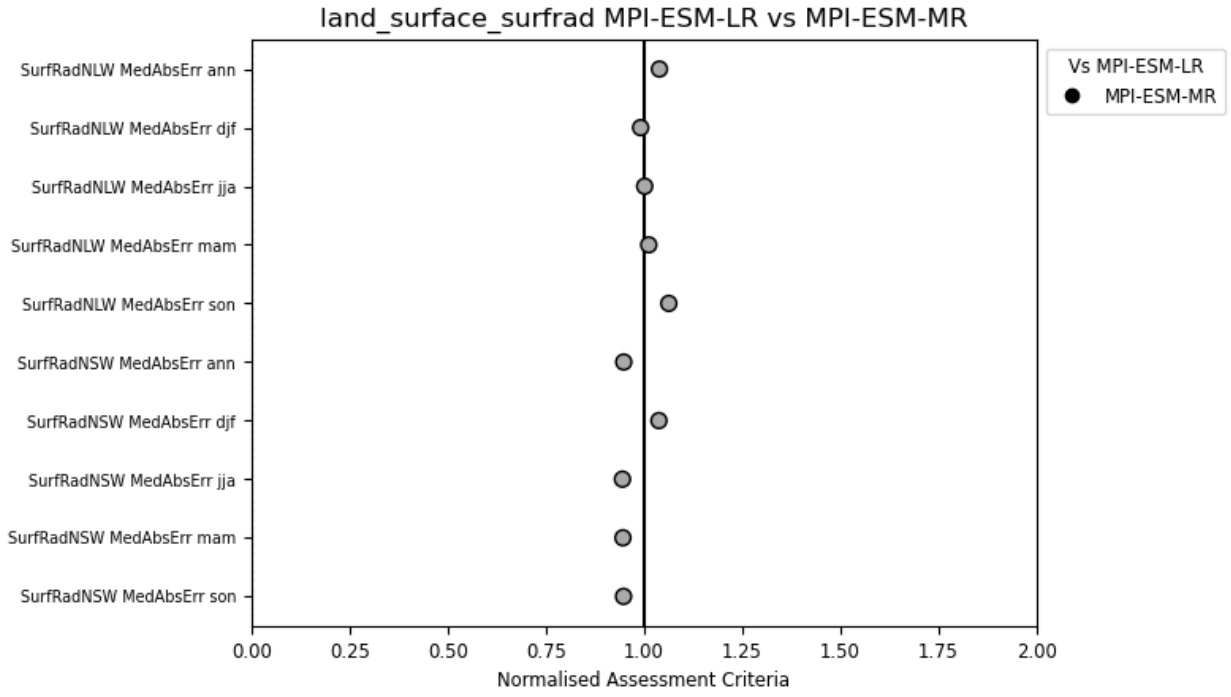


Fig. 10: *Normalised metrics plot comparing a control and experiment simulation*

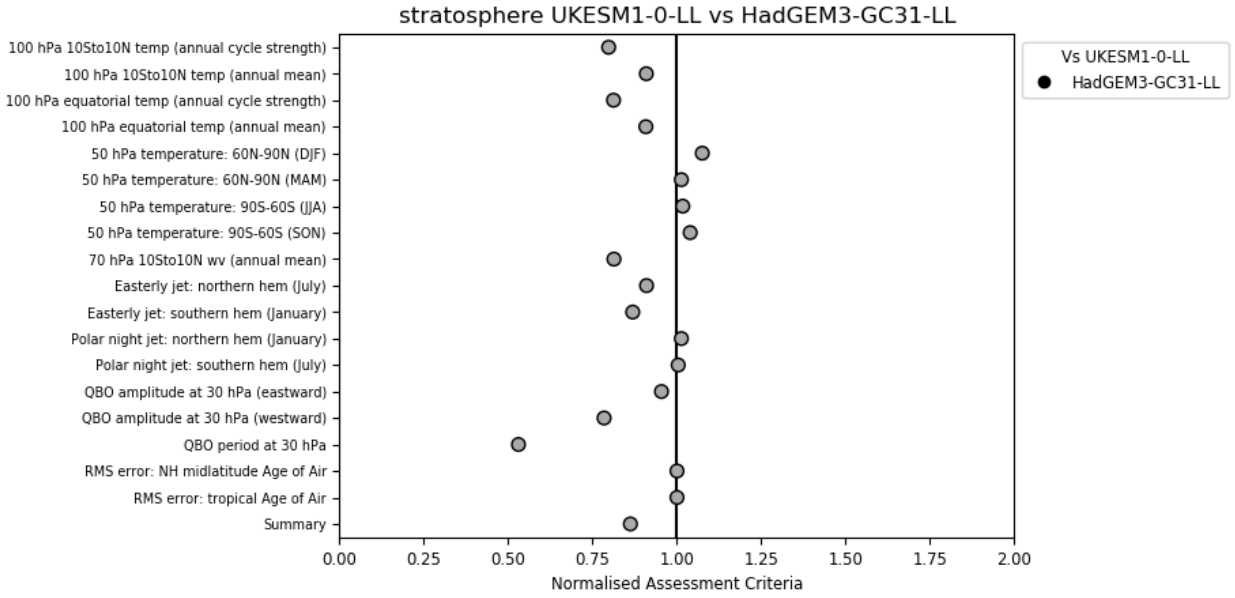


Fig. 11: *Standard metrics plot comparing standard metrics from UKESM1-0-LL and HadGEM3-GC31.*

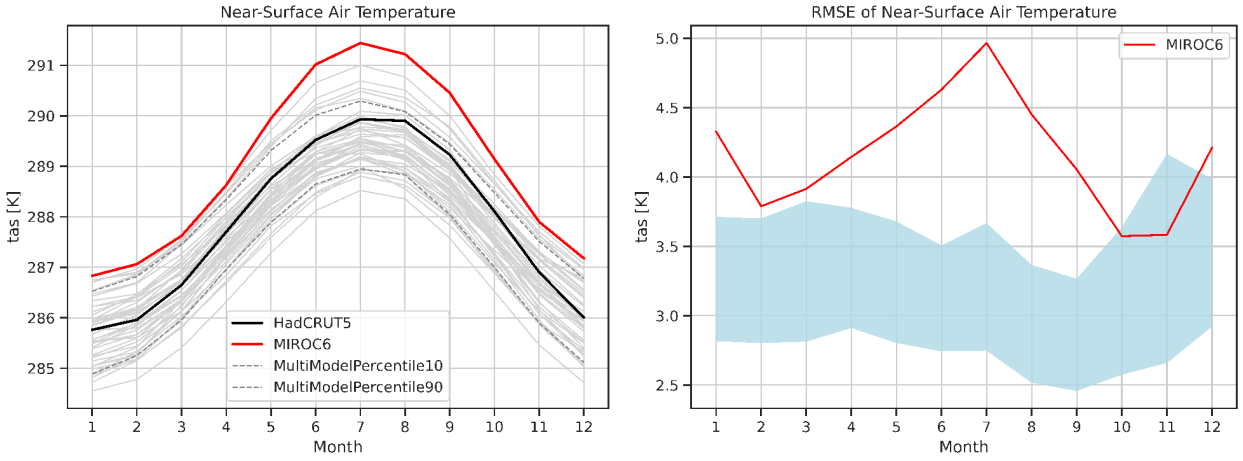


Fig. 12: *(Left) Multi-year global mean (2000-2004) of the seasonal cycle of near-surface temperature in K fro…*

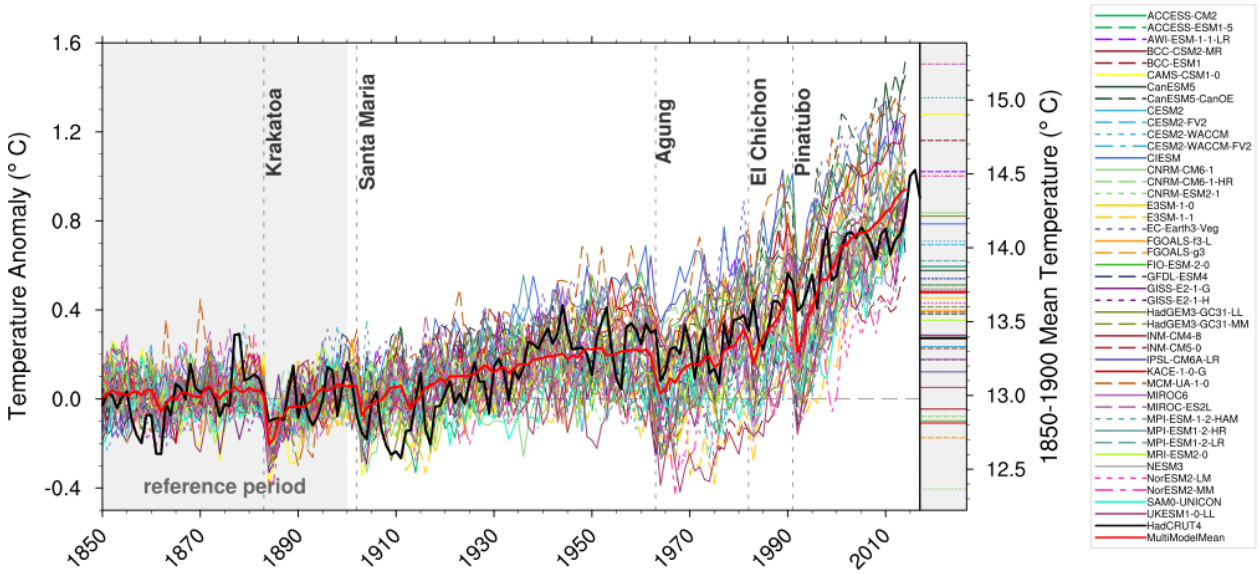


Fig. 13: *Observed and simulated time series of the anomalies in annual and global mean surface temperature. A…*

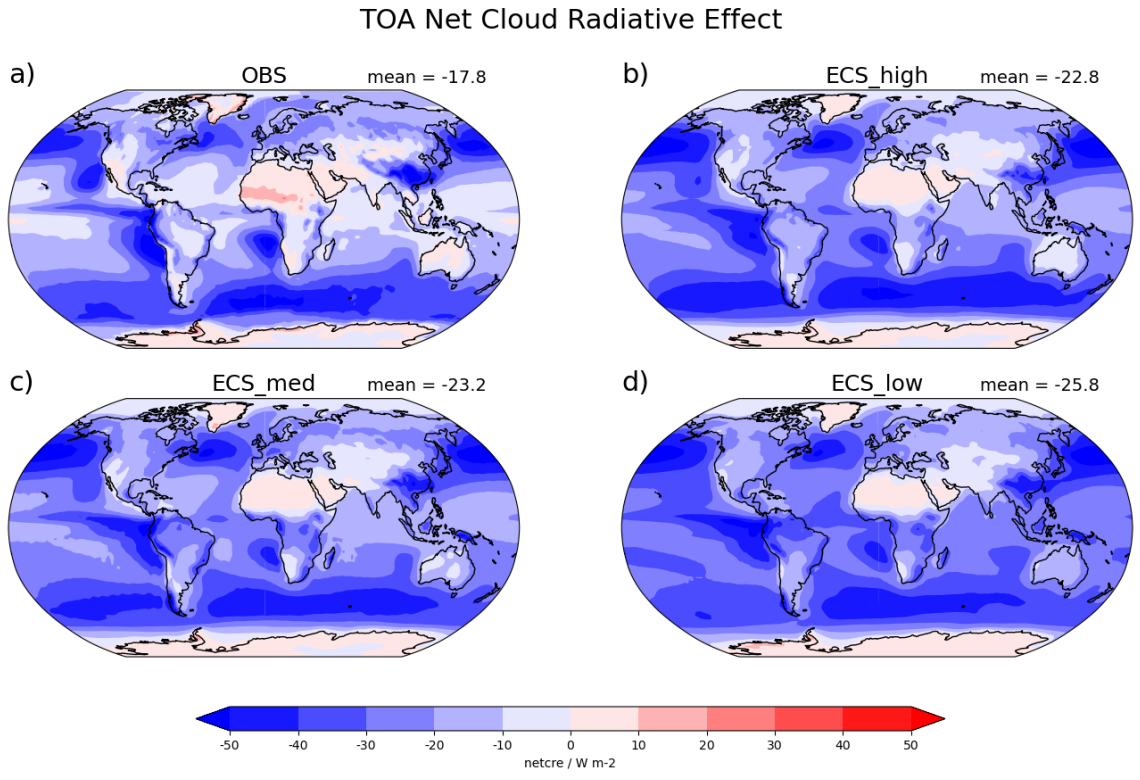


Fig. 14: *Geographical map of the multi-year annual mean net cloud radiative effect from (a) CERES–EBAF Ed4.2 …*

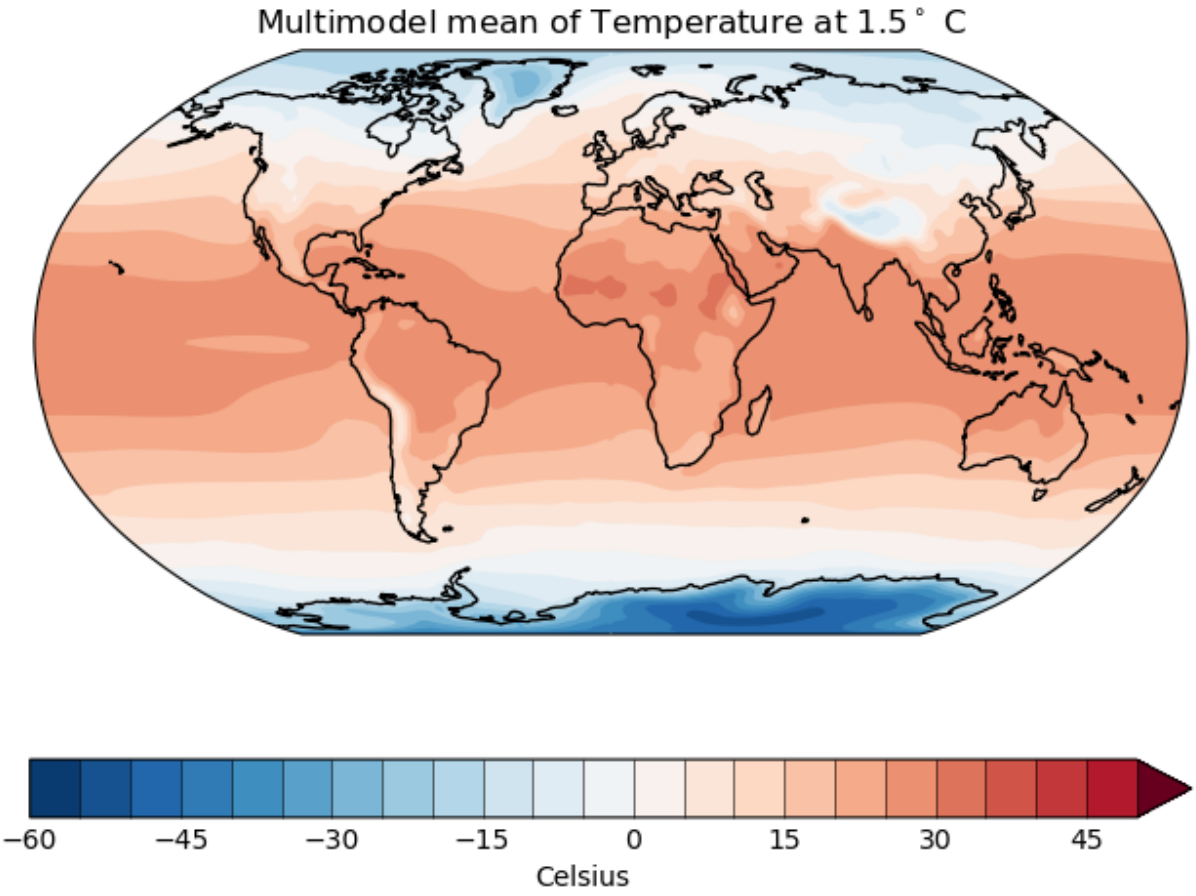


Fig. 15: *Multimodel mean of temperature under SSP1-2.6 at 1.5 degC warming.*

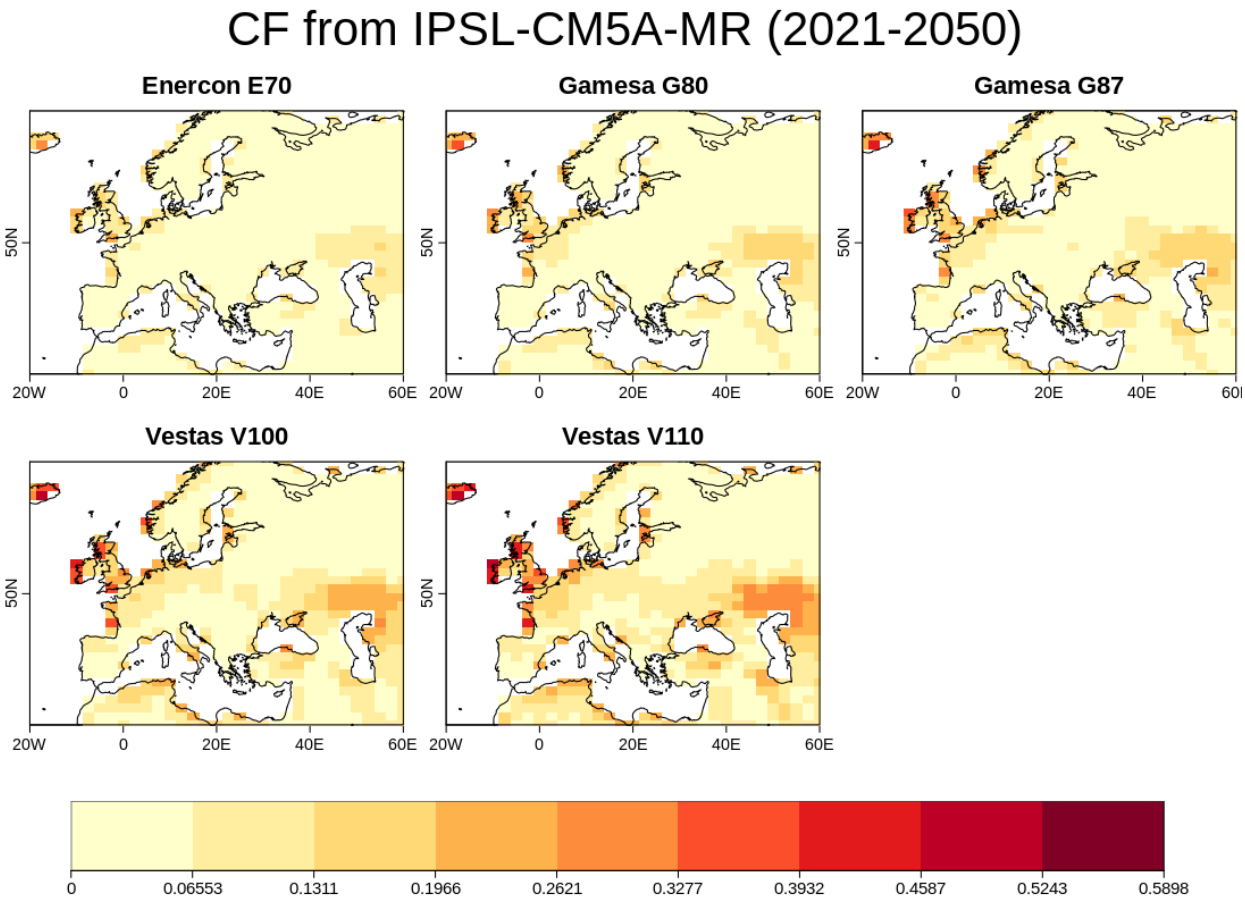


Fig. 16: *Wind capacity factor for five turbines: Enercon E70 (top-left), Gamesa G80 (middle-top), Gamesa G87 …*

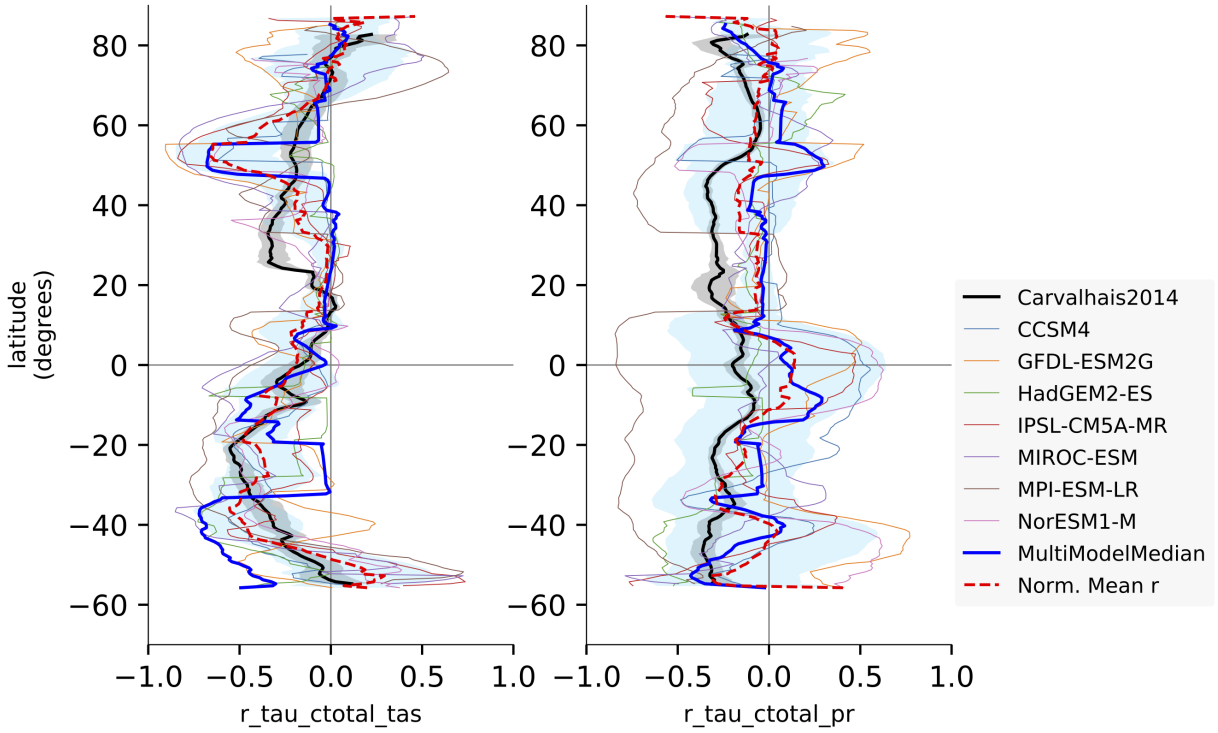


Fig. 17: *Comparison of latitudinal (zonal) variations of pearson correlation between turnover time and climat…*

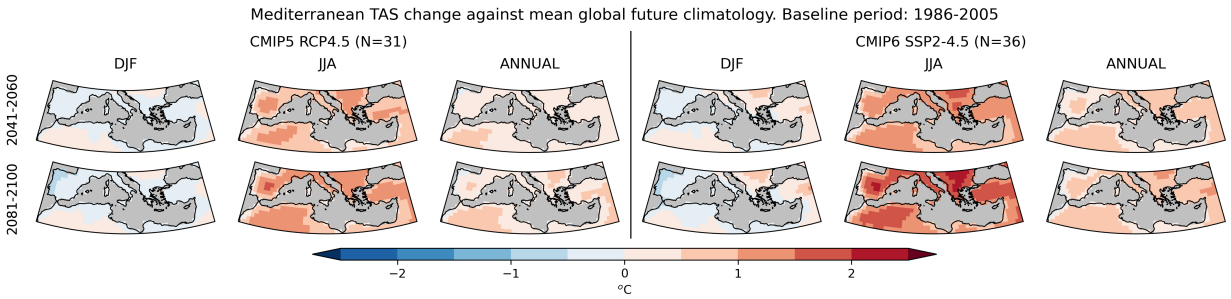


Fig. 18: *Mediterranean region temperature change differences against the mean global temperature change. The …*

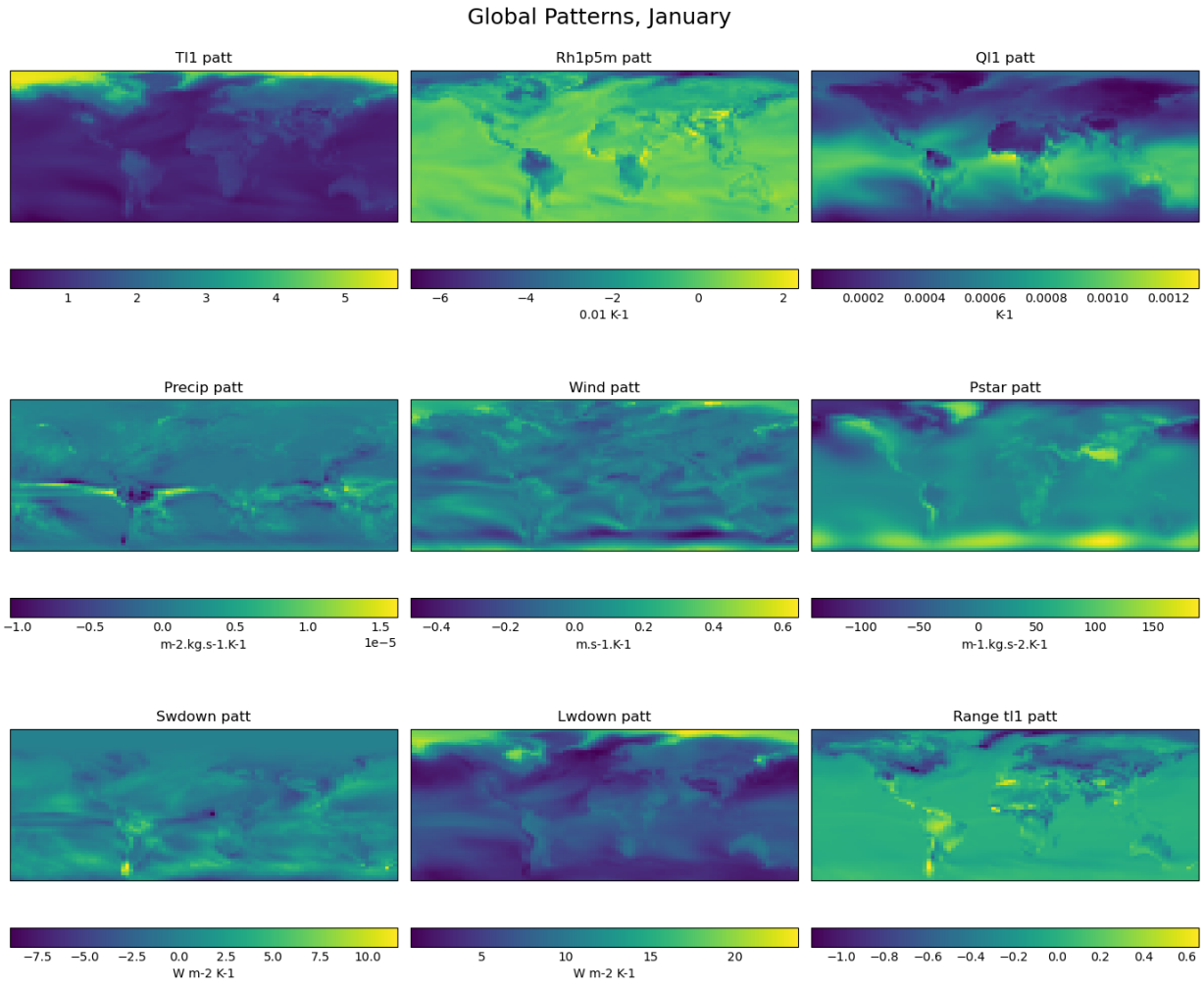


Fig. 19: *Patterns generated for CMIP6 models, gridded view. Patterns are shown per variable, for the month of…*

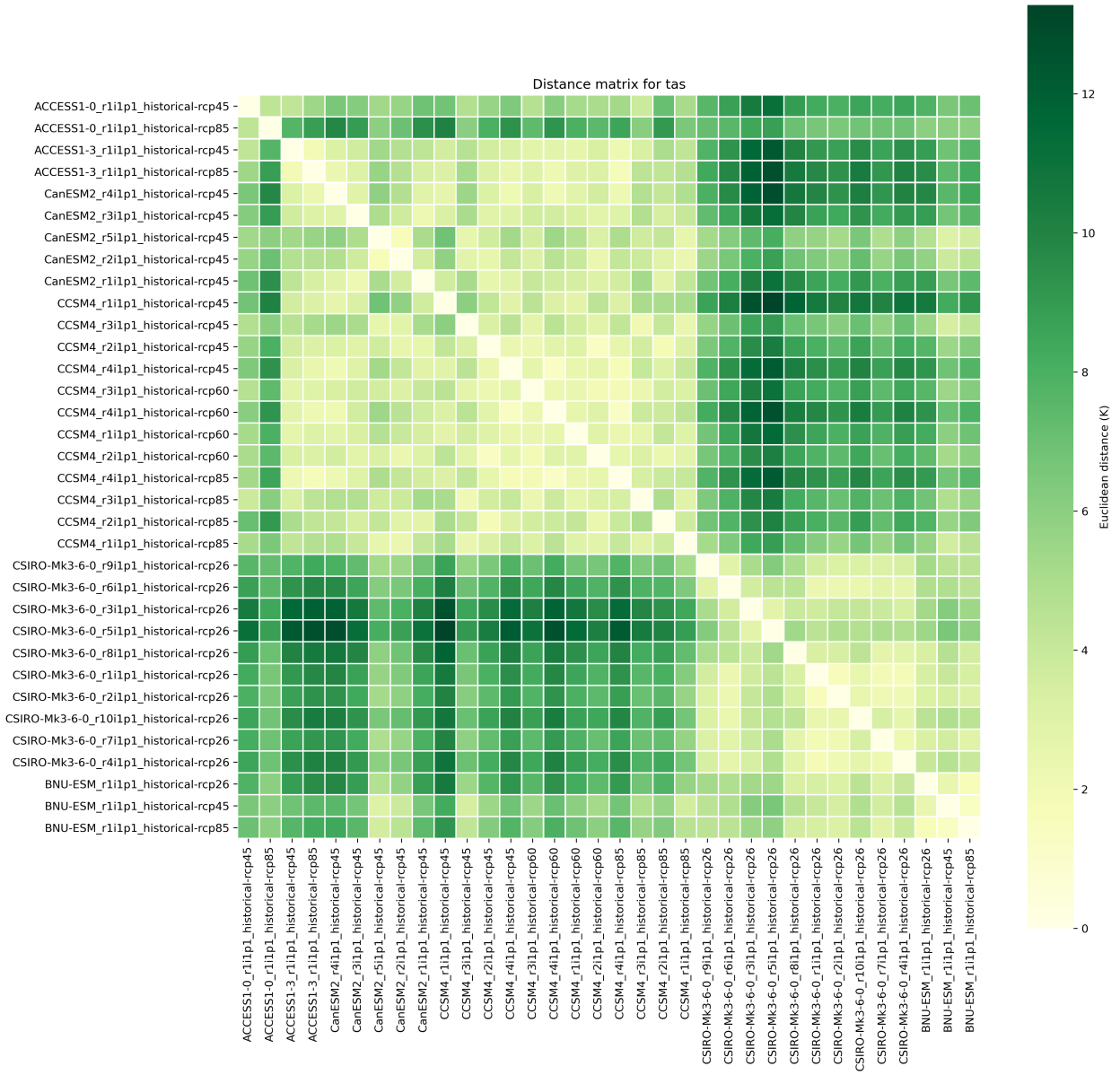


Fig. 20: *Distance matrix for temperature, providing the independence metric.*

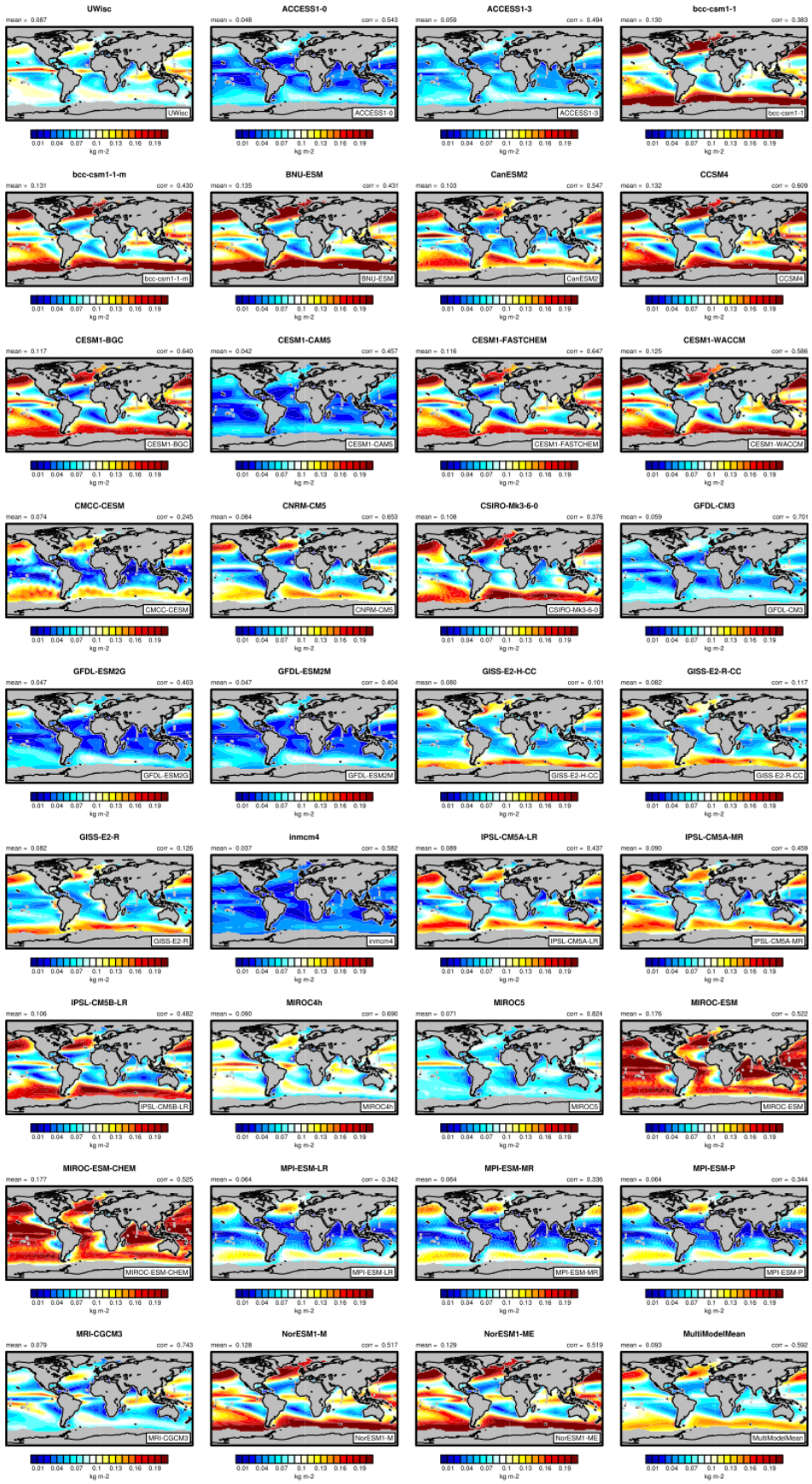


Fig. 21: *The 20-yr average LWP (1986-2005) from the CMIP5 historical model runs and the multi-model mean in c...*

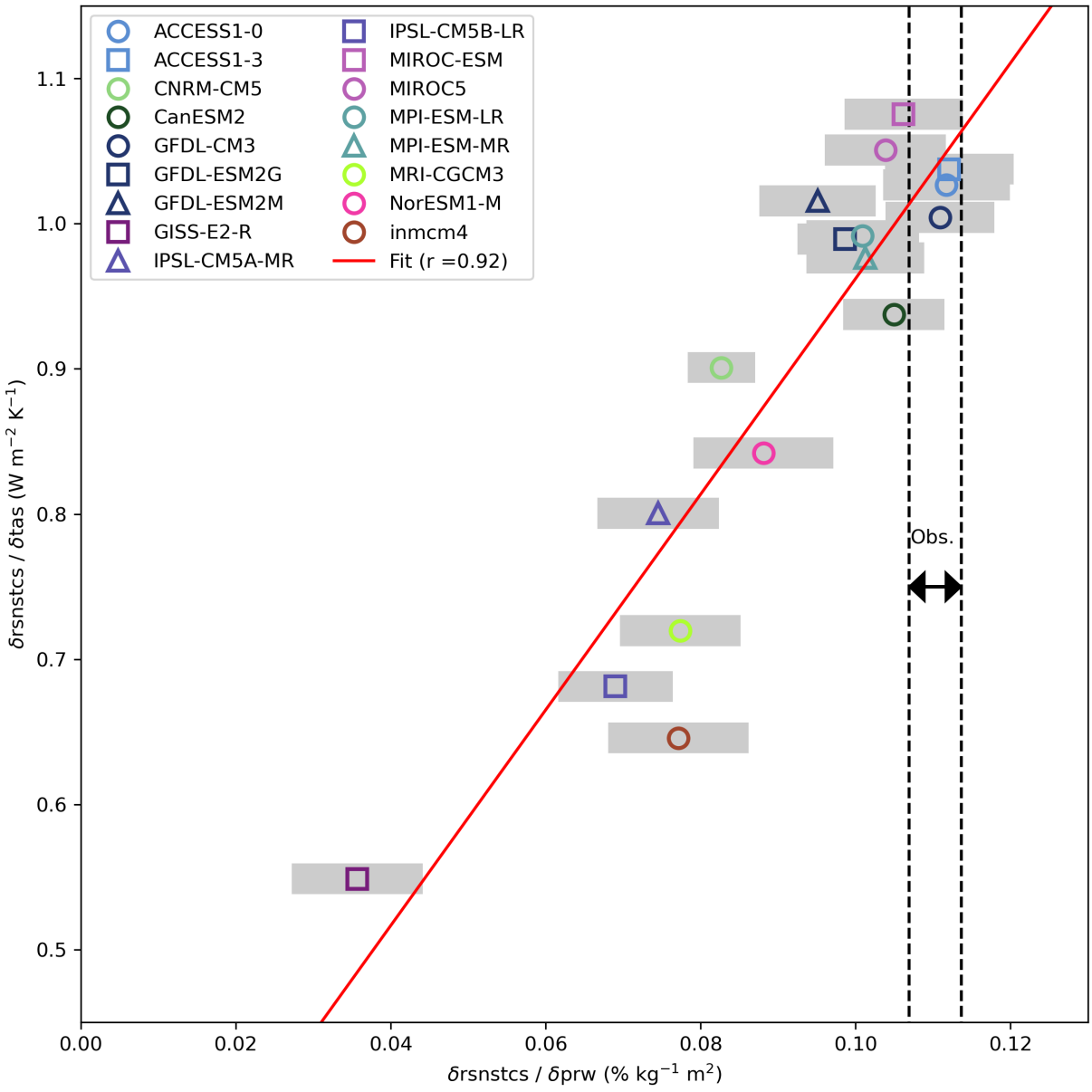


Fig. 22: *Scatter plot and regression line computed between the ratio of the change of net short wave radiatio…*

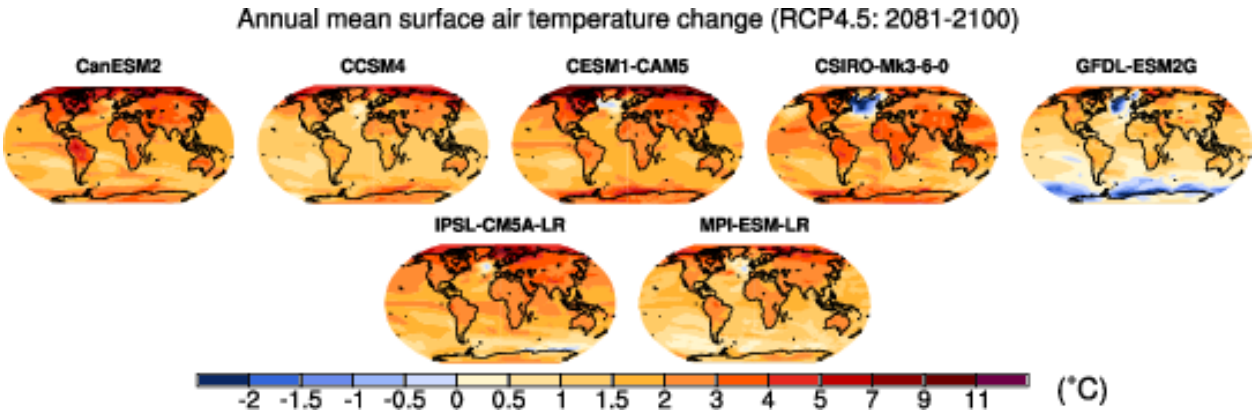


Fig. 23: *Surface air temperature change in 2081–2100 displayed as anomalies with respect to 1986–2005 for RCP…*

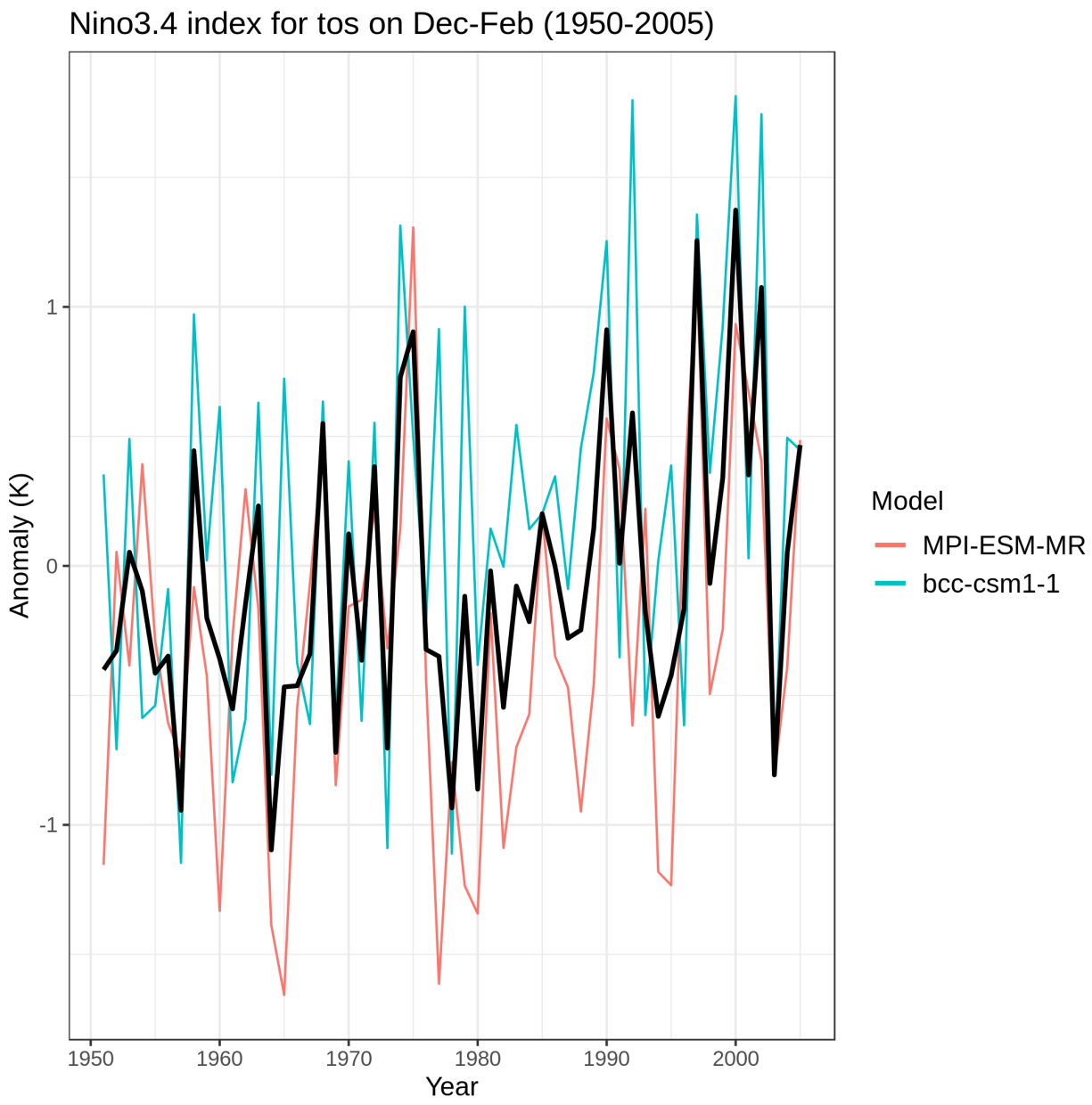


Fig. 24: *Time series of the standardized sea surface temperature (tos) area averaged over the Nino 3.4 region…*

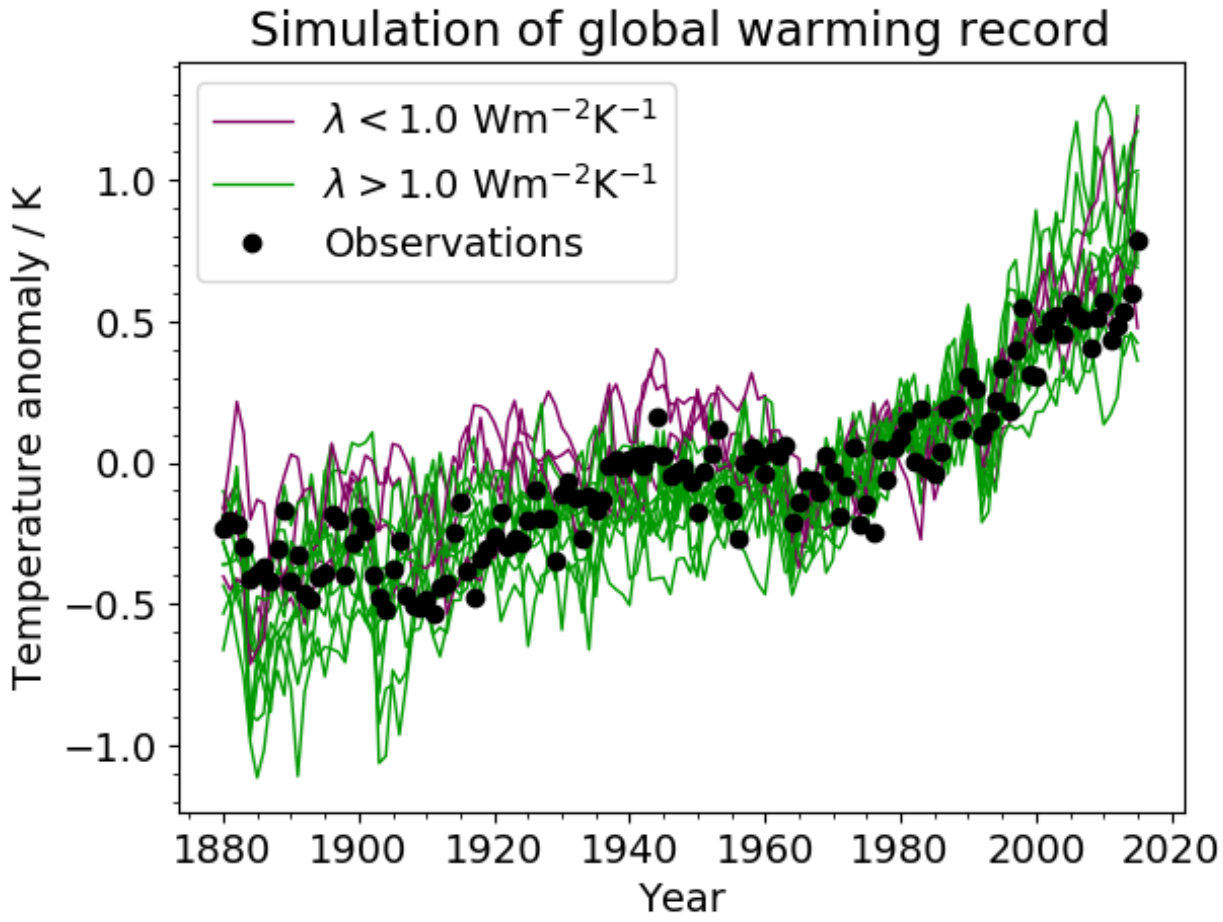


Fig. 25: *Simulated change in global temperature from CMIP5 models (coloured lines), compared to the global te…*

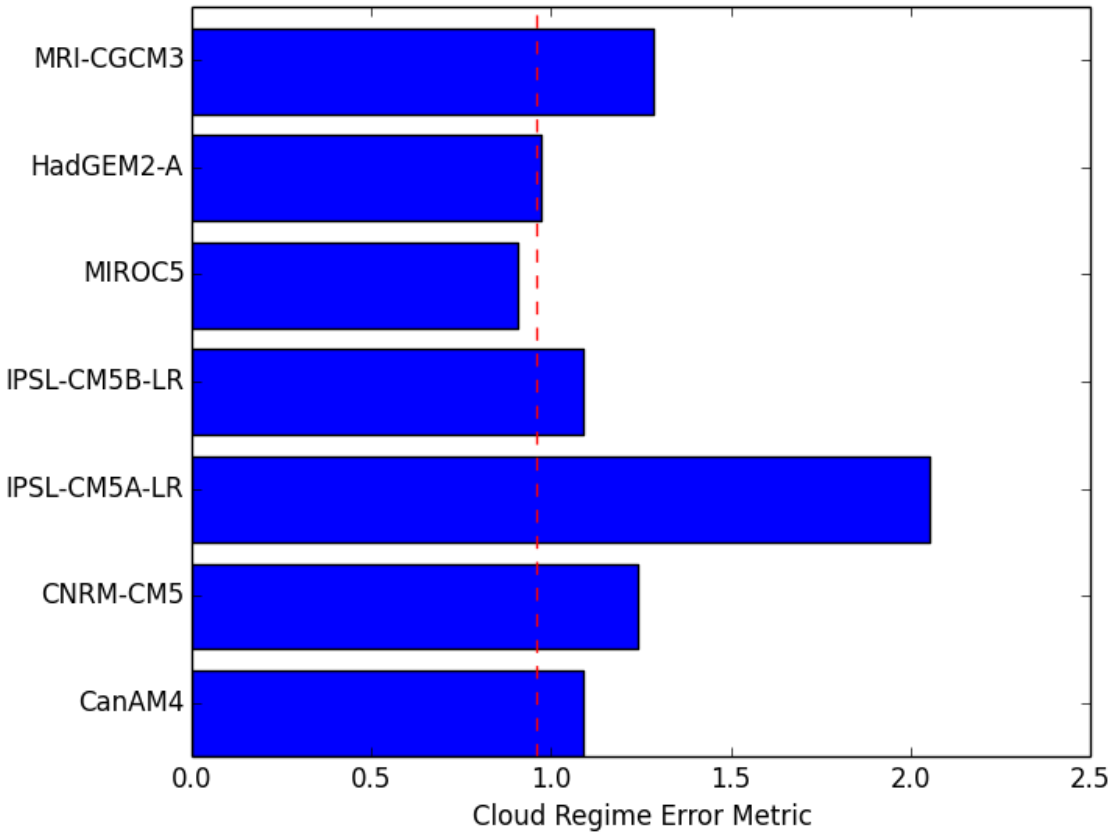


Fig. 26: *Cloud Regime Error Metrics (CREMpd) from William and Webb (2009) applied to selected CMIP5 AMIP simu…*

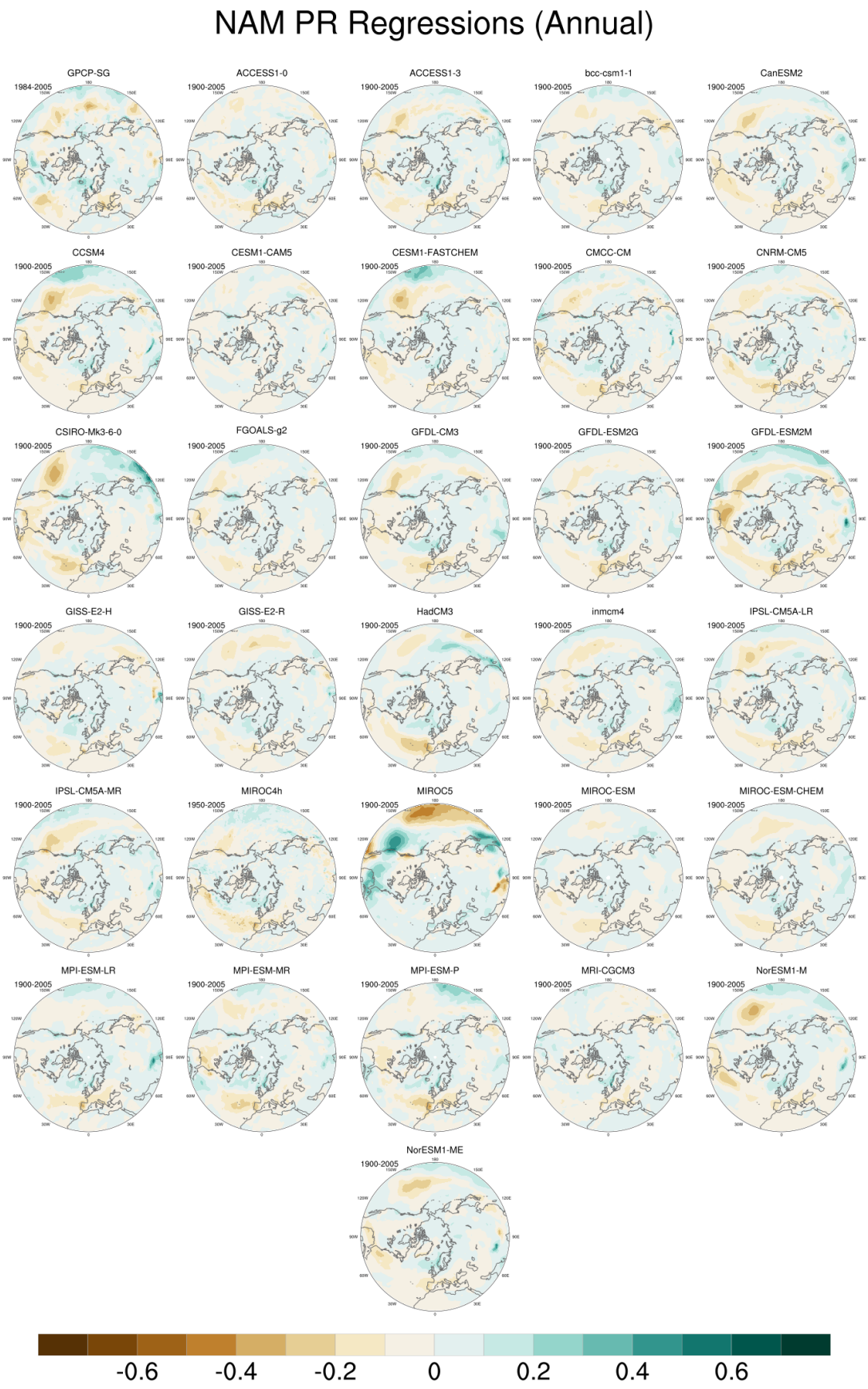


Fig. 27: *Regression of the precipitation anomalies (PR) onto the Northern Annular Mode (NAM) index for the ti…*

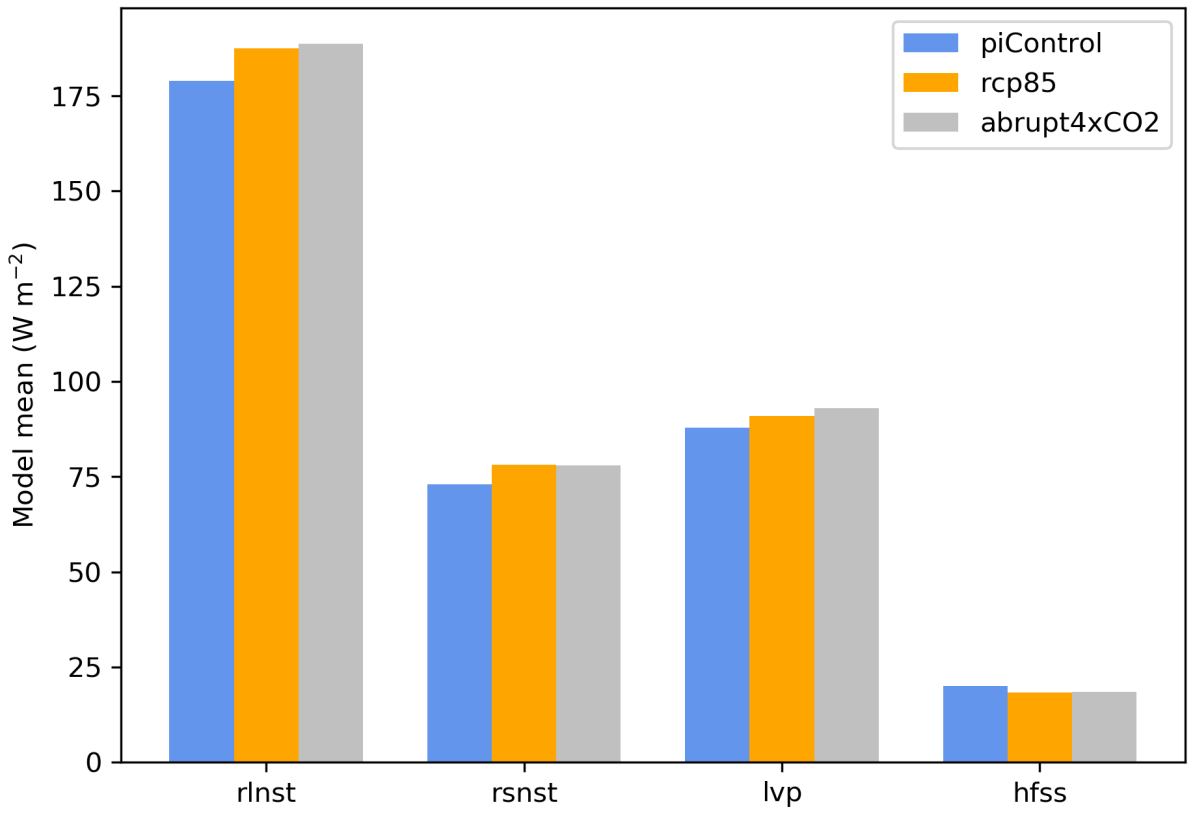


Fig. 28: *Global average multi-model mean comparing different model experiments for the sum of upward long wav…*

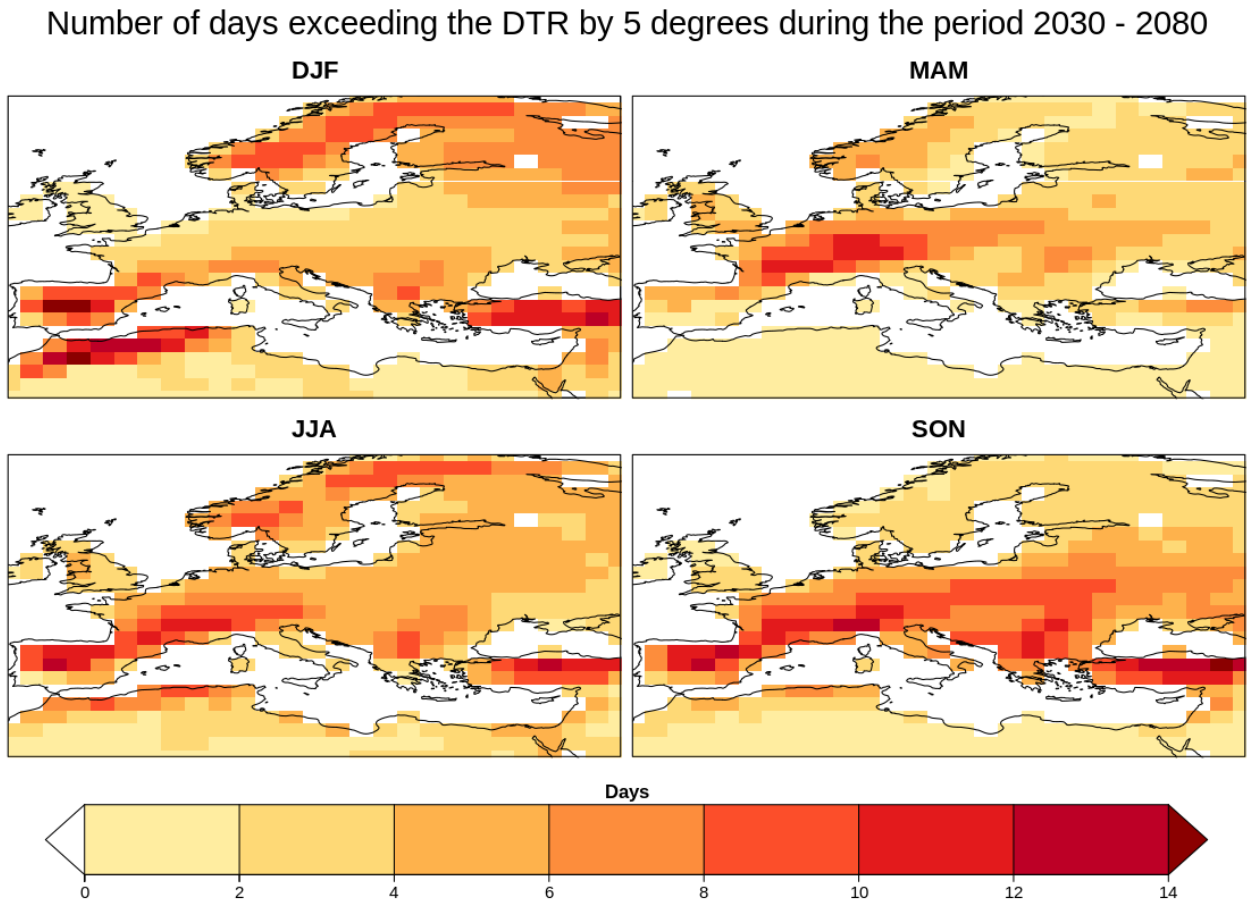


Fig. 29: *Mean number of days exceeding the Diurnal Temperature Range (DTR) simulated during the historical pe…*

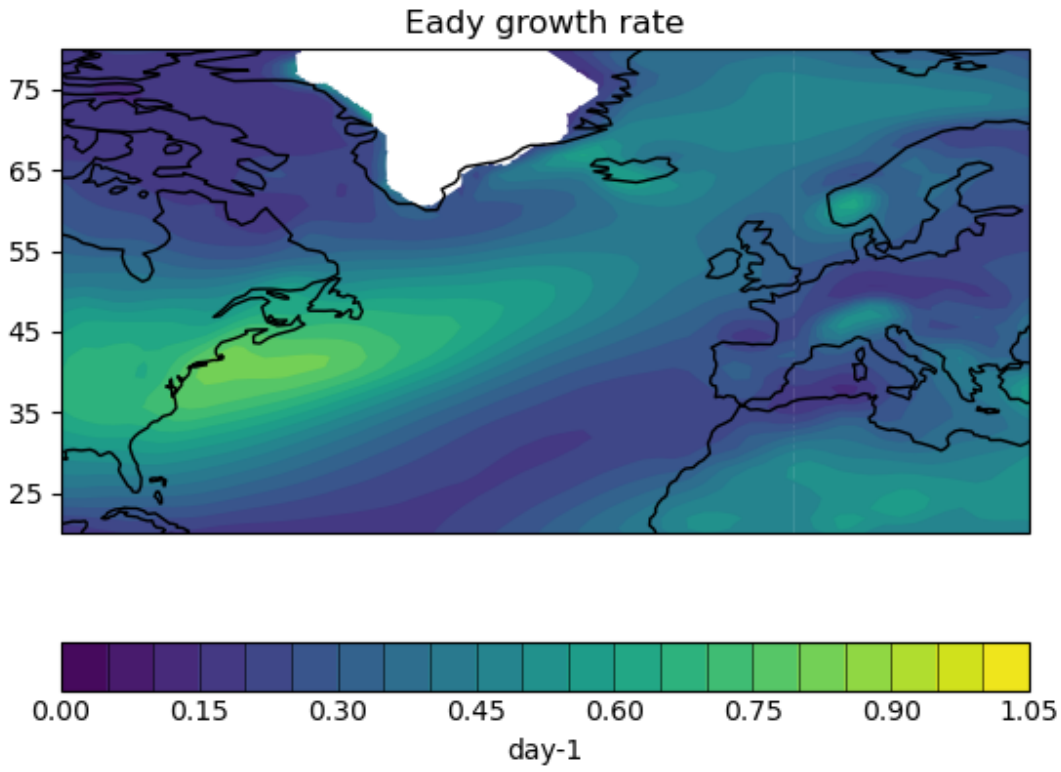


Fig. 30: *Eady Growth Rate values over the North-Atlantic region at 70000 Pa.*

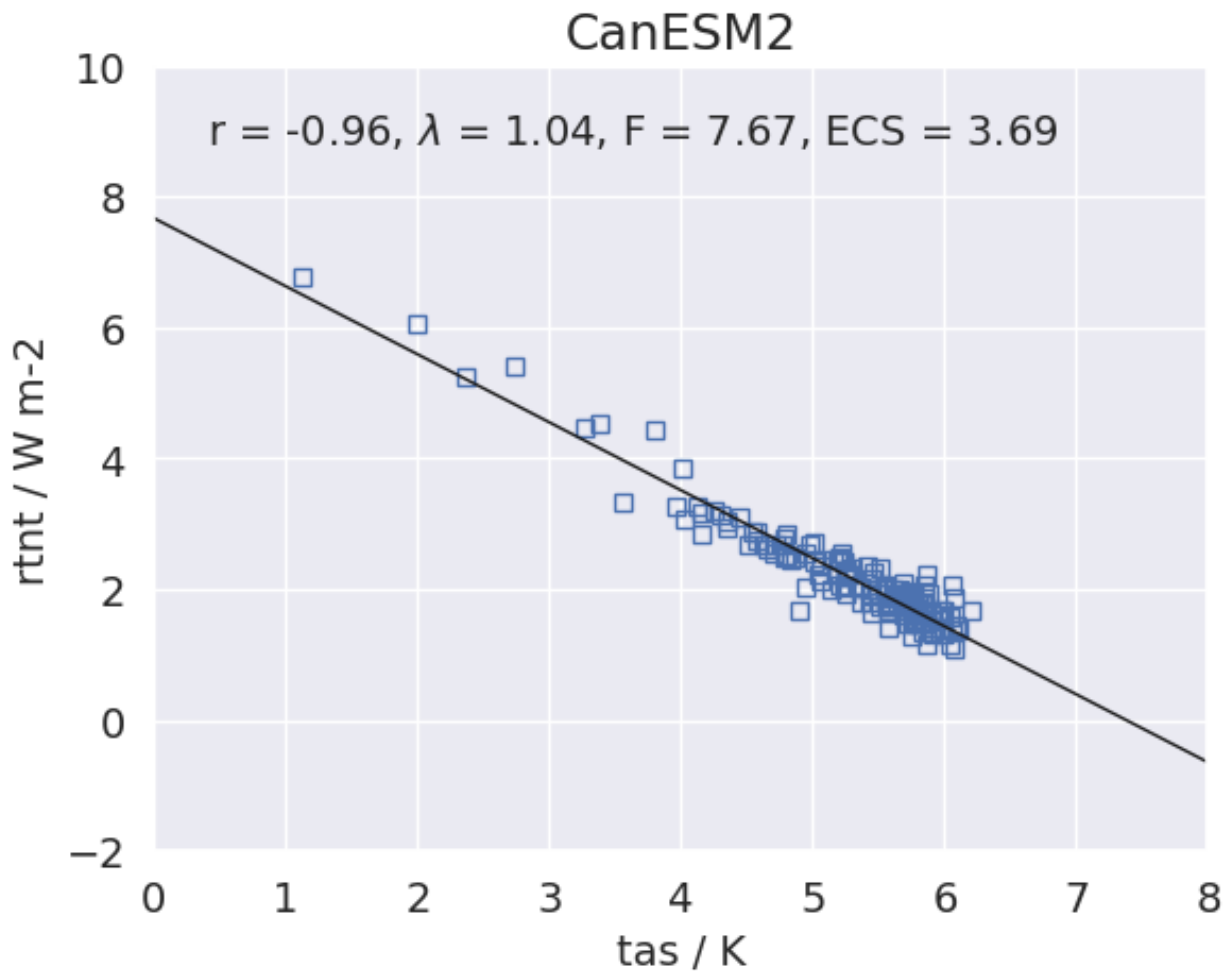


Fig. 31: *Scatterplot between TOA radiance and global mean surface temperature anomaly for 150 years of the ab…*

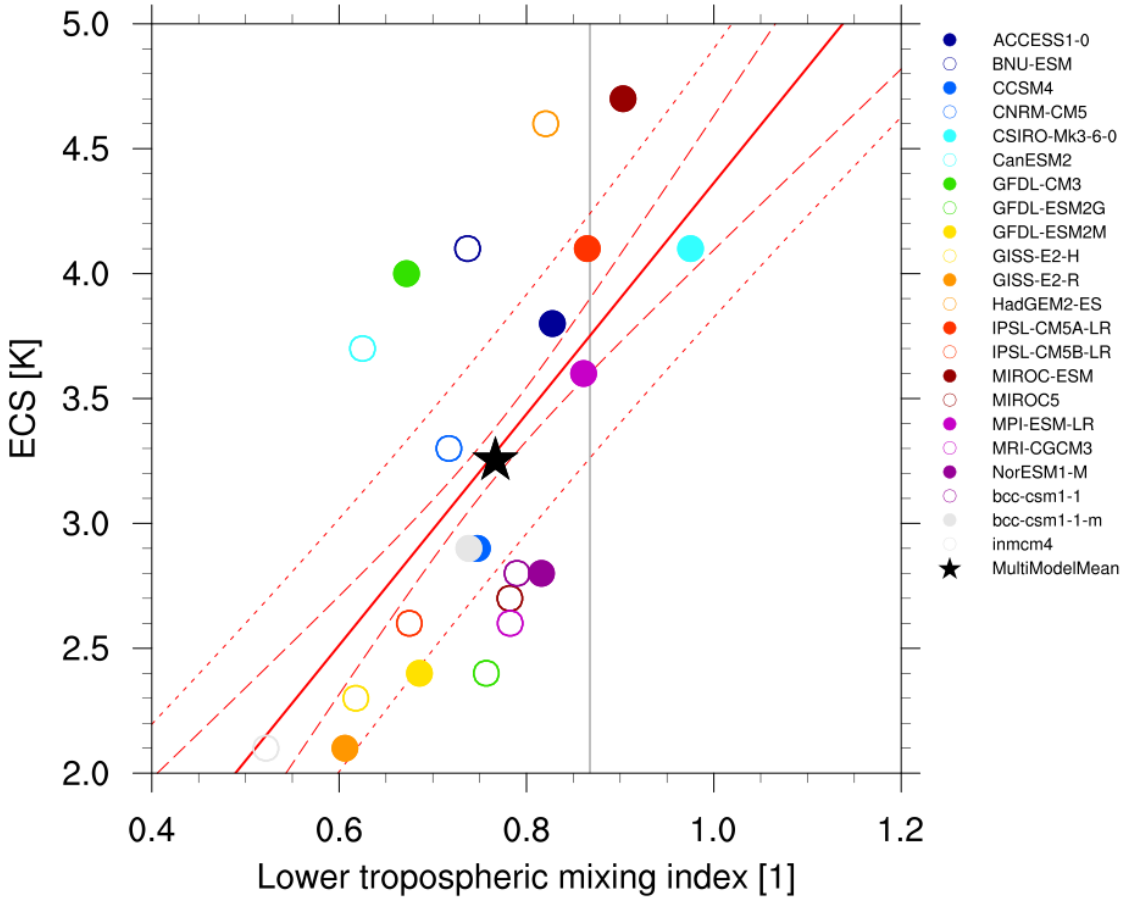


Fig. 32: *Lower tropospheric mixing index (LTMI; Sherwood et al., 2014) vs. equilibrium climate sensitivity fr…*

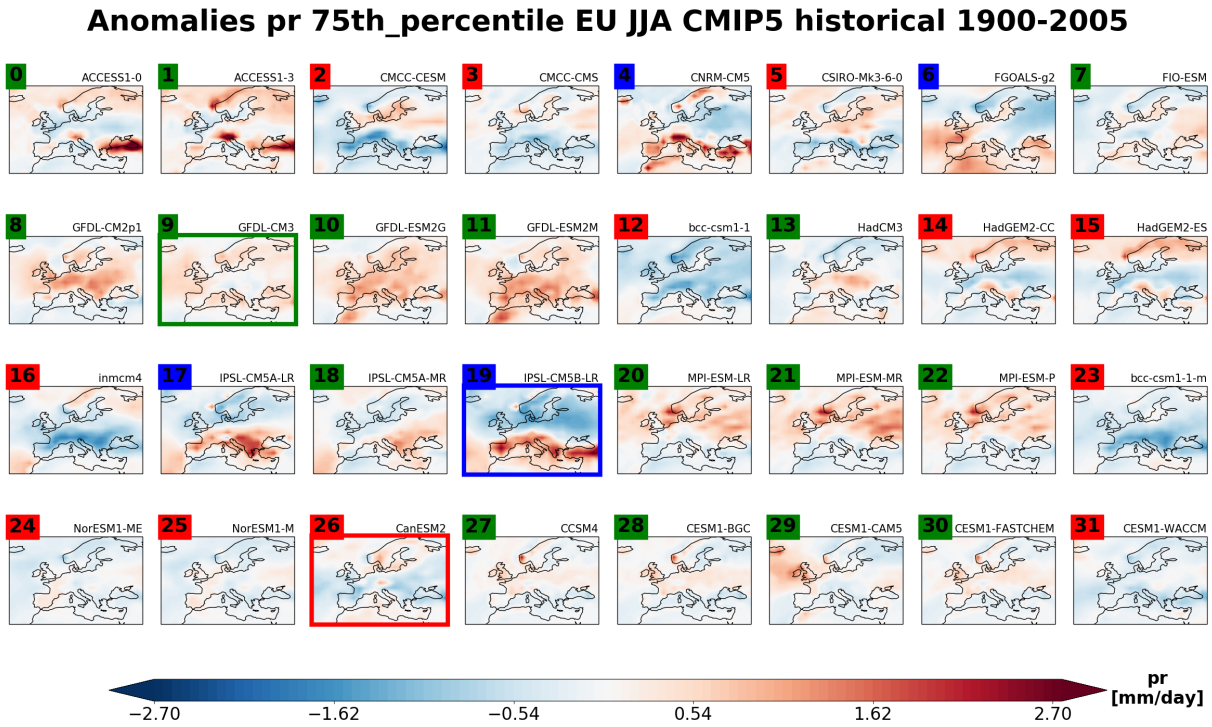


Fig. 33: *Clustering based on the 75th percentile of historical summer (JJA) precipitation rate for CMIP5 mode…*

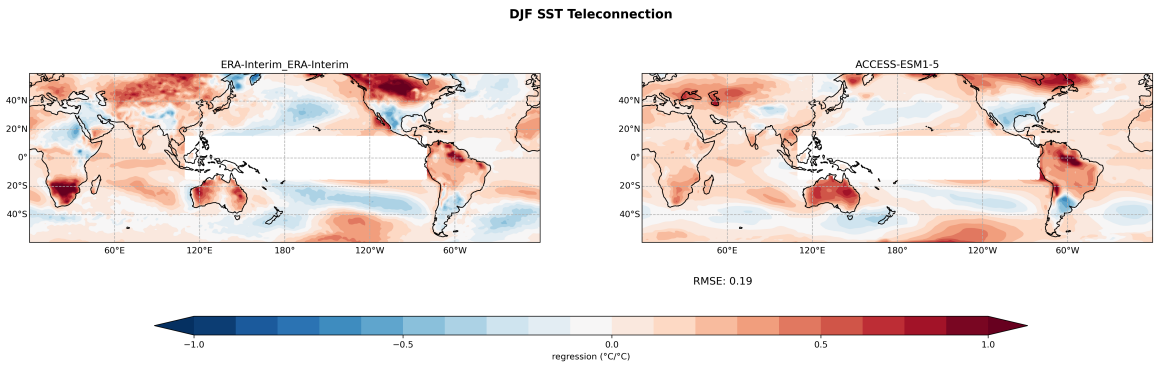


Fig. 34: *PR or SST anomalies on Earth (between 60°S-60°N), showing the location associated with ENSO.*

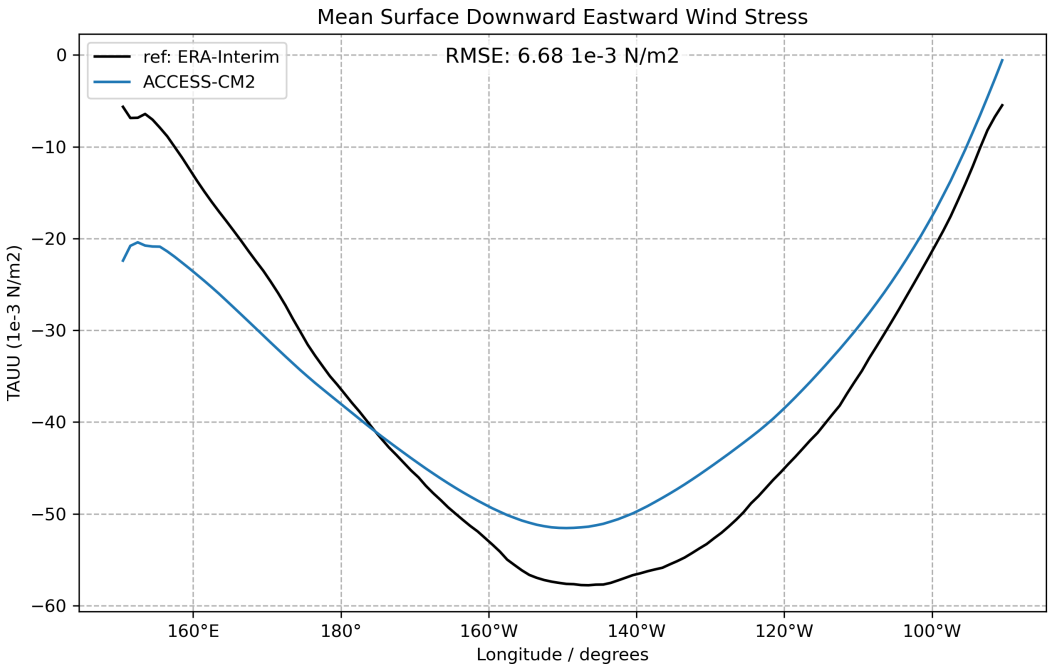


Fig. 35: *Bias in the zonal structure of zonal wind stress (Taux) in the equatorial Pacific (5°S-5°N averaged)…*

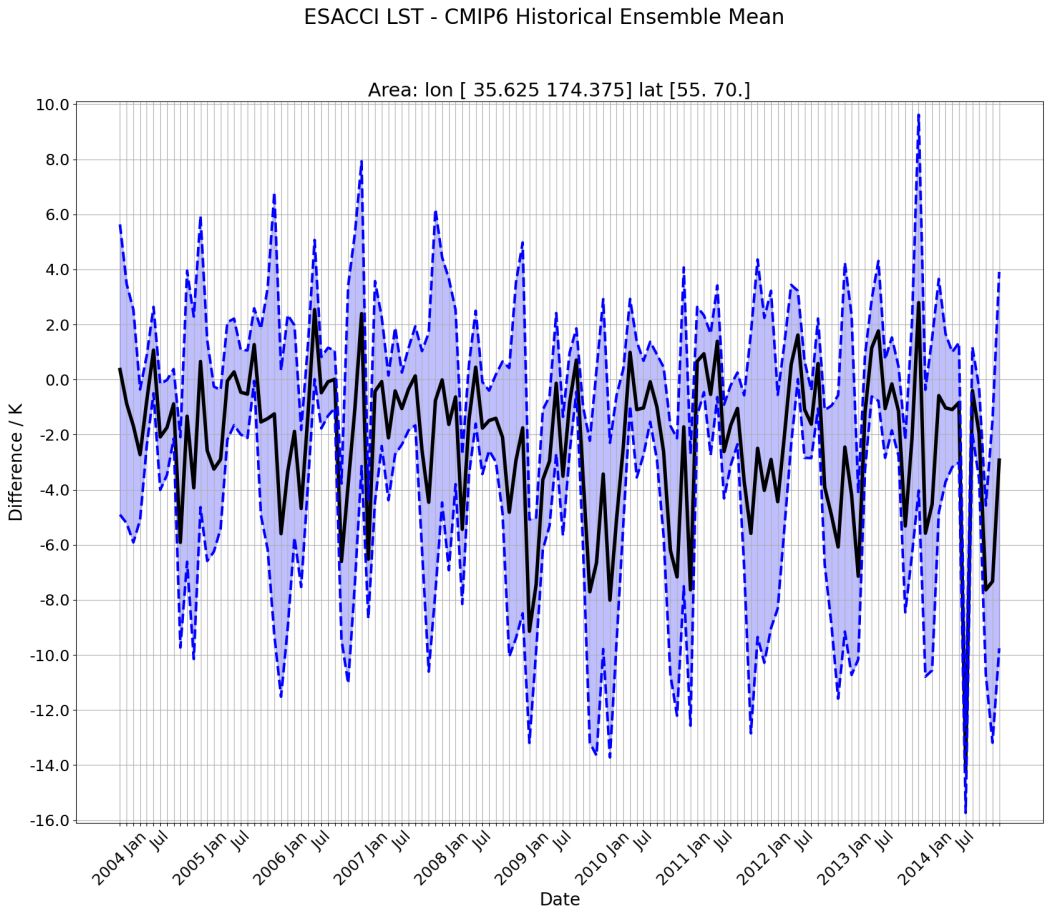


Fig. 36: *Timeseries of the ESA CCI LST minus mean of CMIP6 ensembles. The selected region is 35E-175E, 55N-70…*

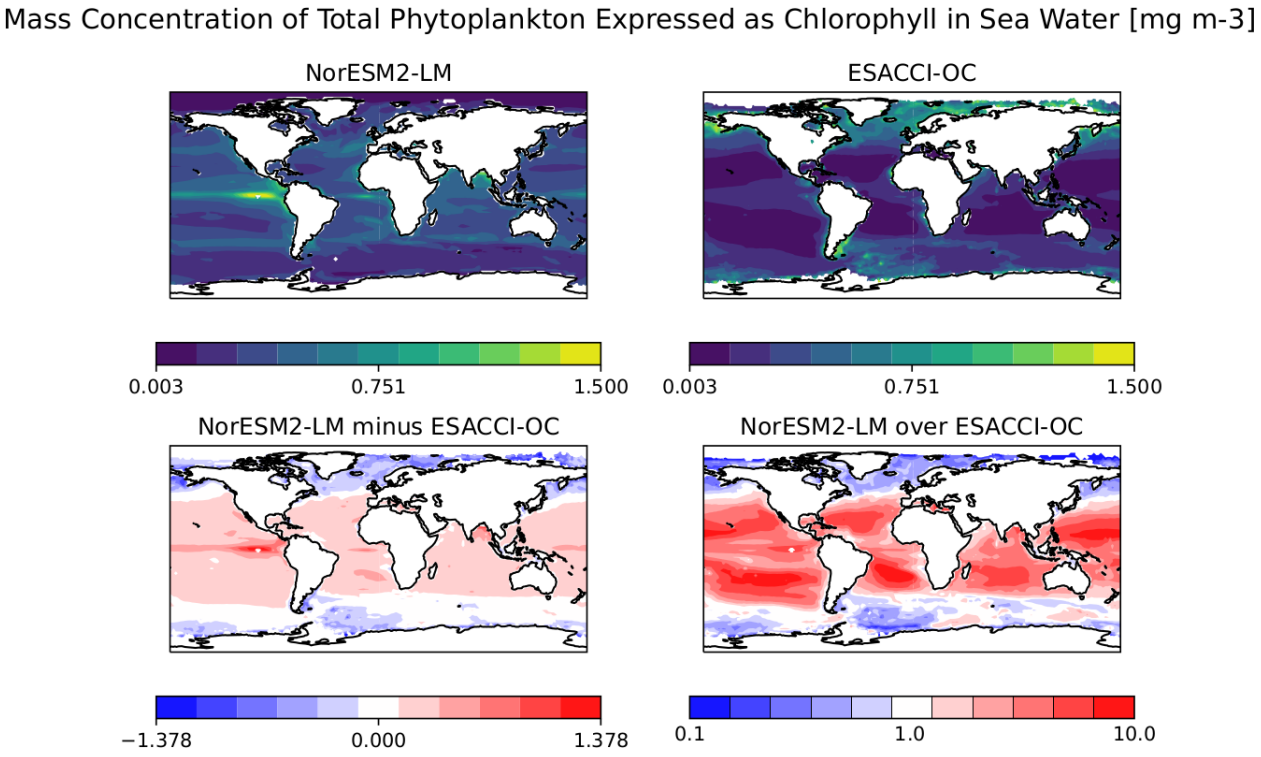


Fig. 37: *Surface chlorophyll from ESACCI-OC ocean colour data version 5.0 and the CMIP6 model NorESM2-LM. Thi…*

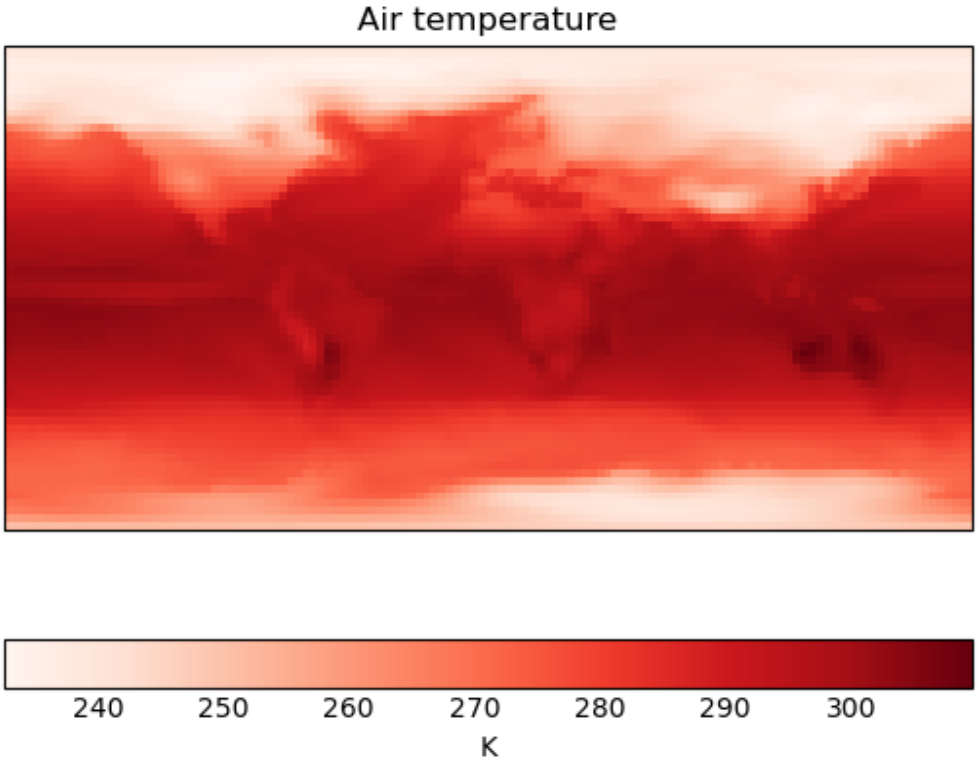


Fig. 38: *Air temperature in January 2000 (BCC-ESM1 CMIP6).*

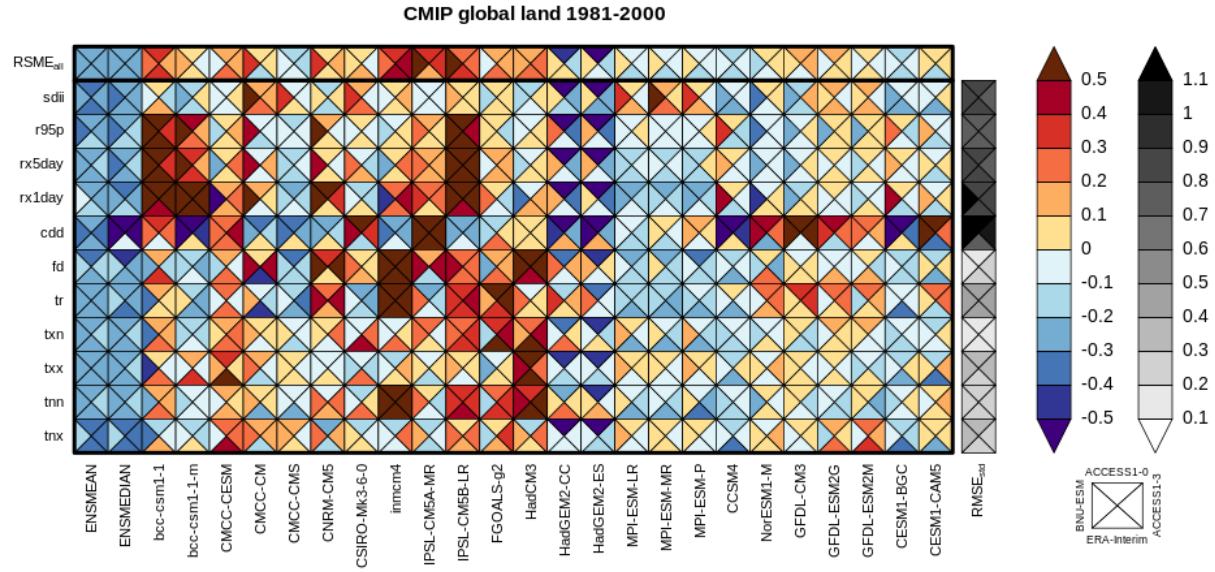


Fig. 39: *Portrait plot of relative error metrics for the CMIP5 temperature and precipitation extreme indices …*

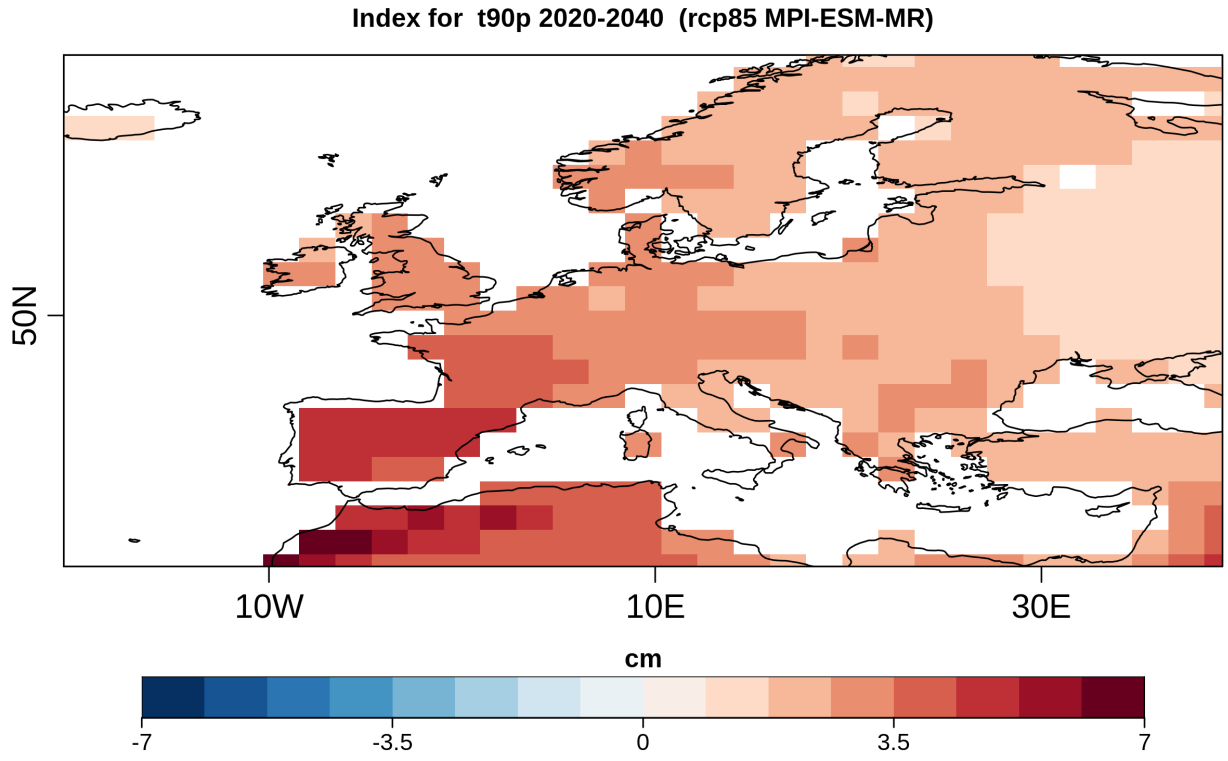


Fig. 40: *Average change in the heat component (t90p metric) of the Combined Climate Extreme Index for the 202…*

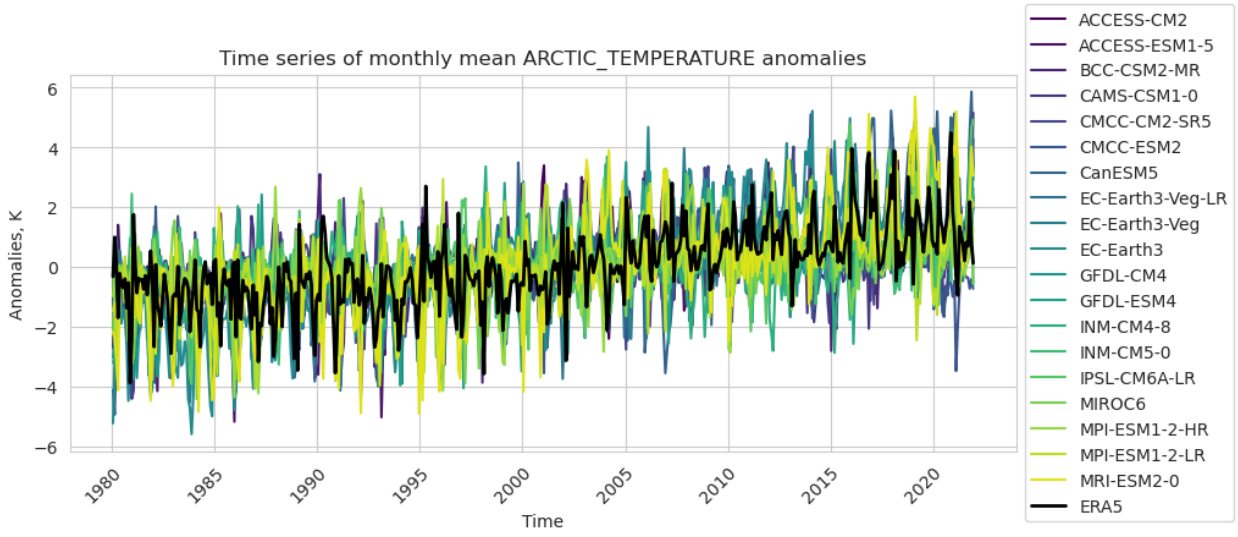


Fig. 41: *Monthly mean temperature anomalies in the Arctic (65°–90°N) from observations and selected CMIP6 mod…*

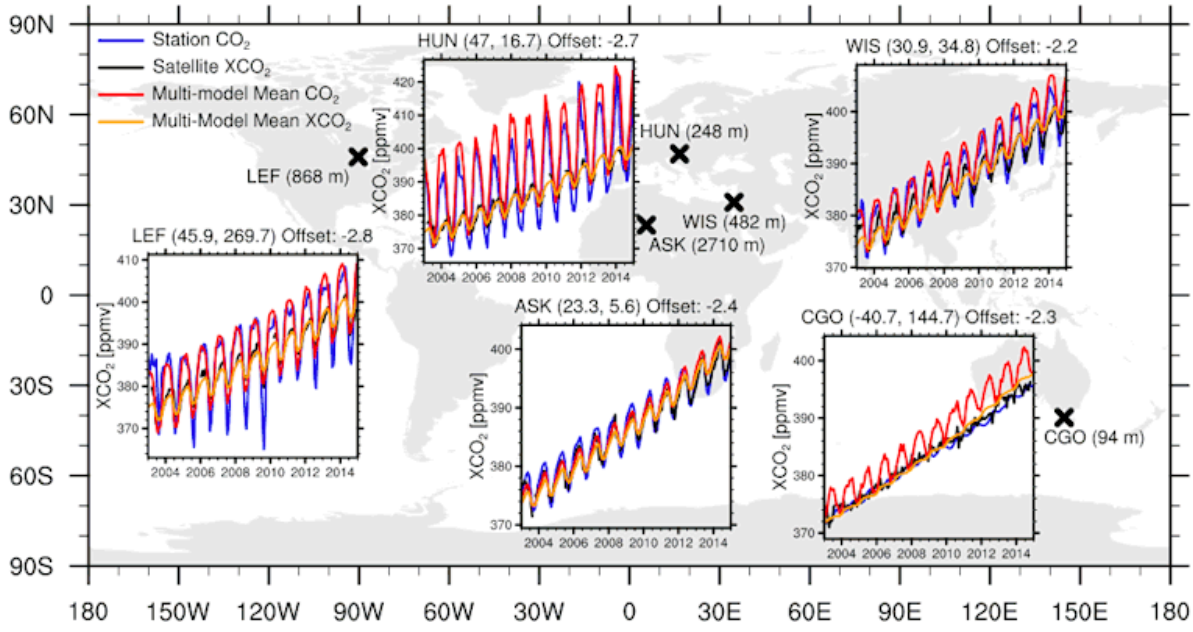


Fig. 42: *Comparison of time series from satellite, in situ, and models sampled accordingly. Caveat: inset plo…*

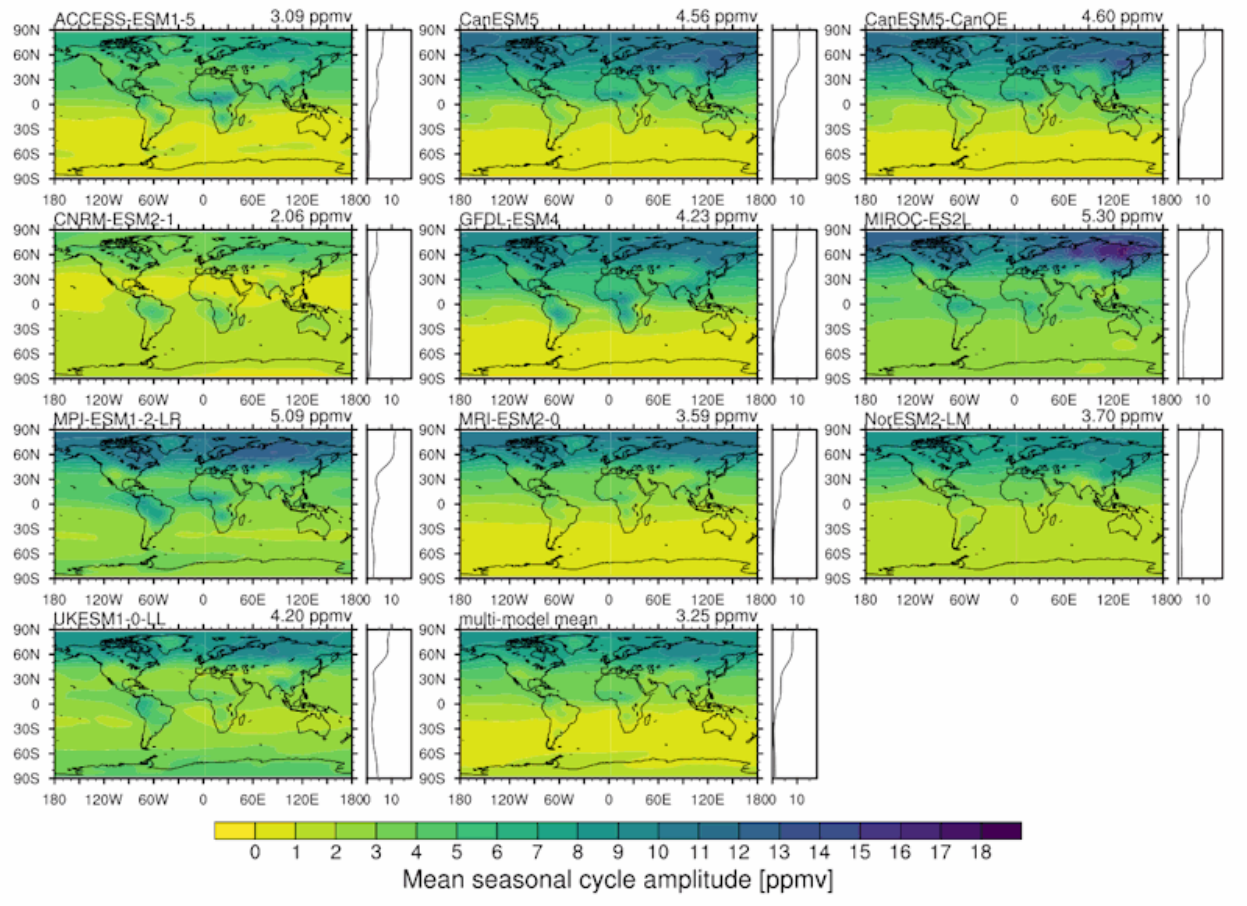


Fig. 43: *Panel plot of spatially resolved seasonal cycle amplitude for all models, including a zonal average …*

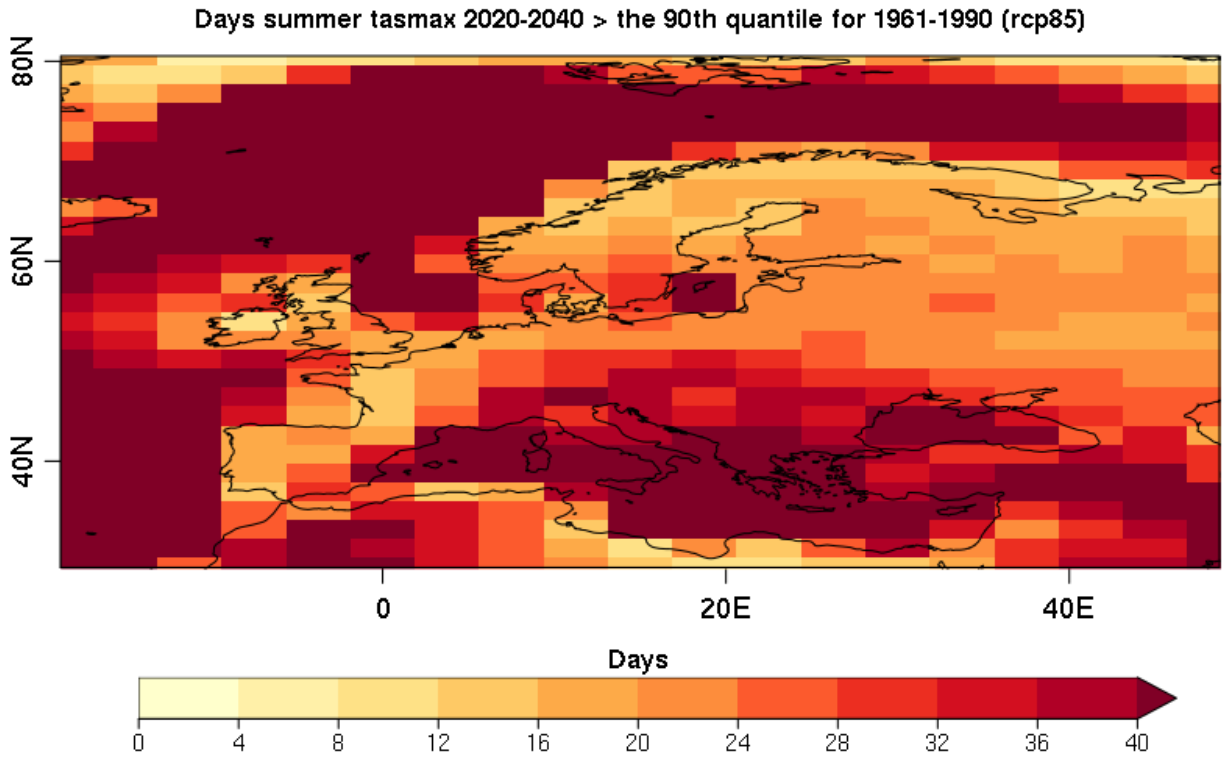


Fig. 44: *Mean number of summer days during the period 2060-2080 when the daily maximum near-surface air tempe…*

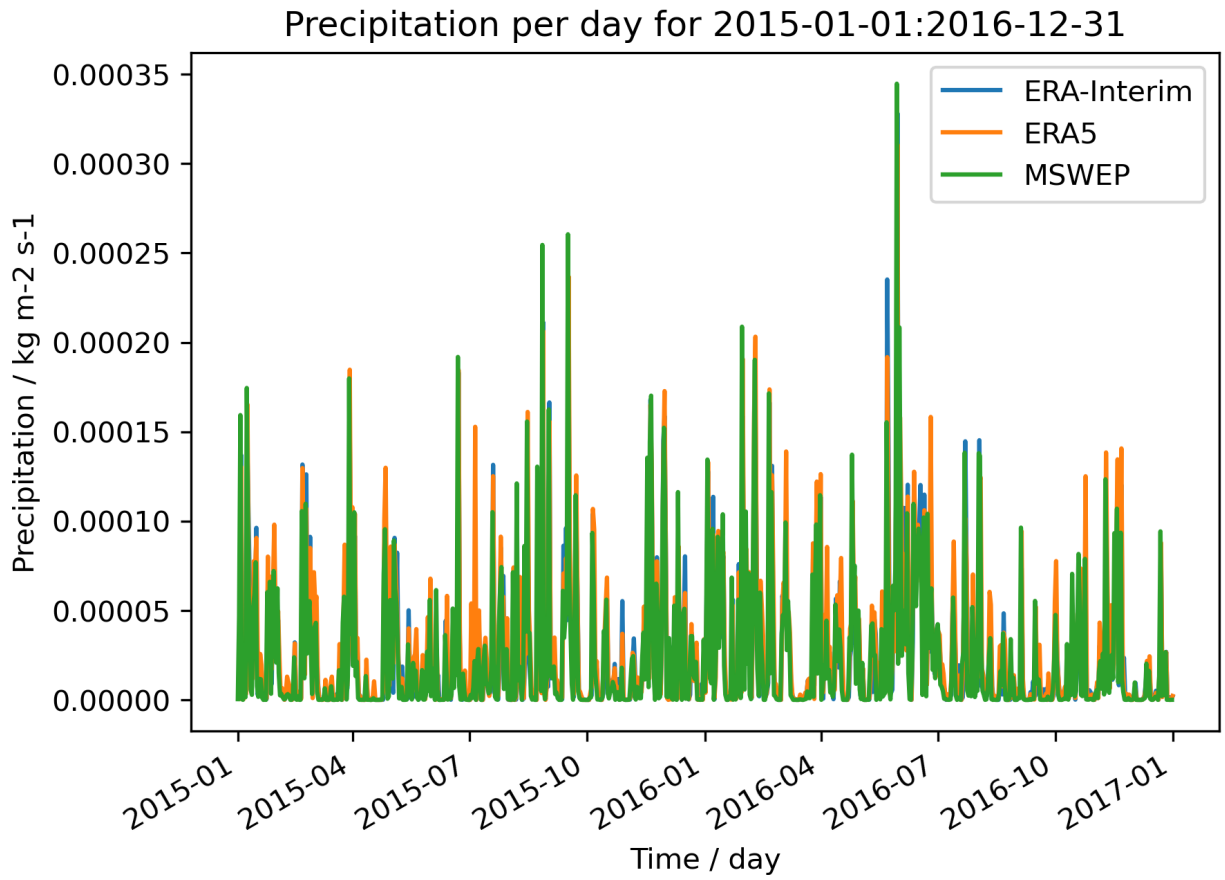


Fig. 45: *Precipitation per day for 2015-01-01:2016-12-31.*

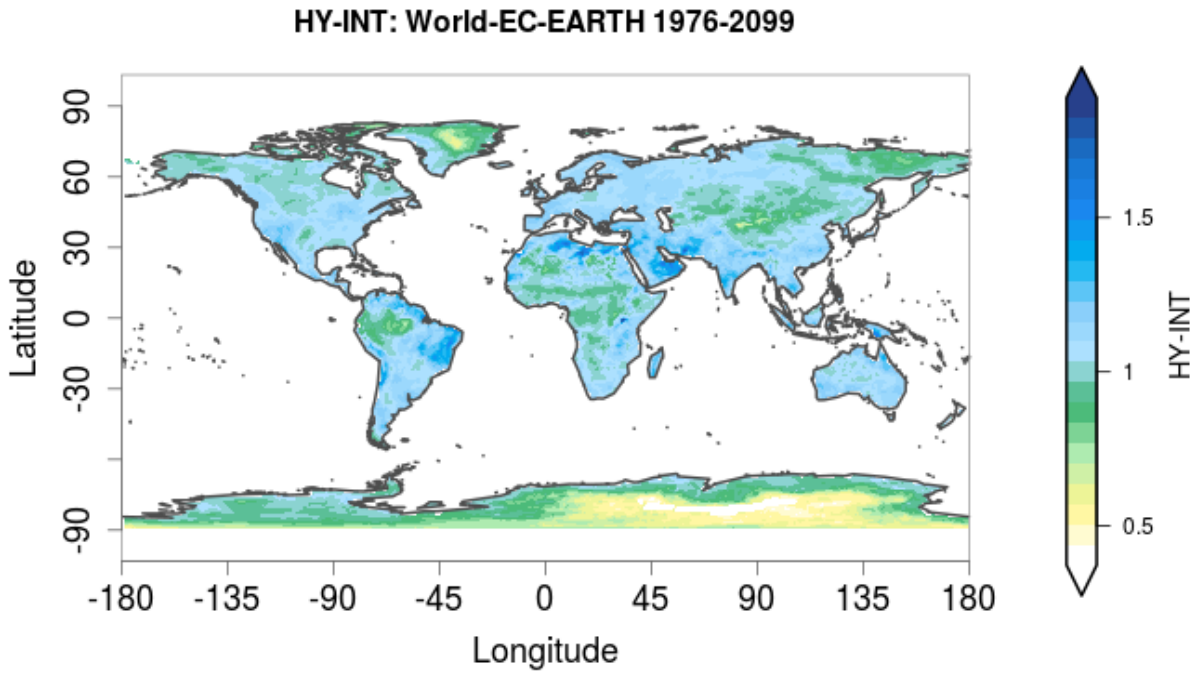


Fig. 46: *Mean hydroclimatic intensity for the EC-EARTH model, for the historical + RCP8.5 projection in the p…*

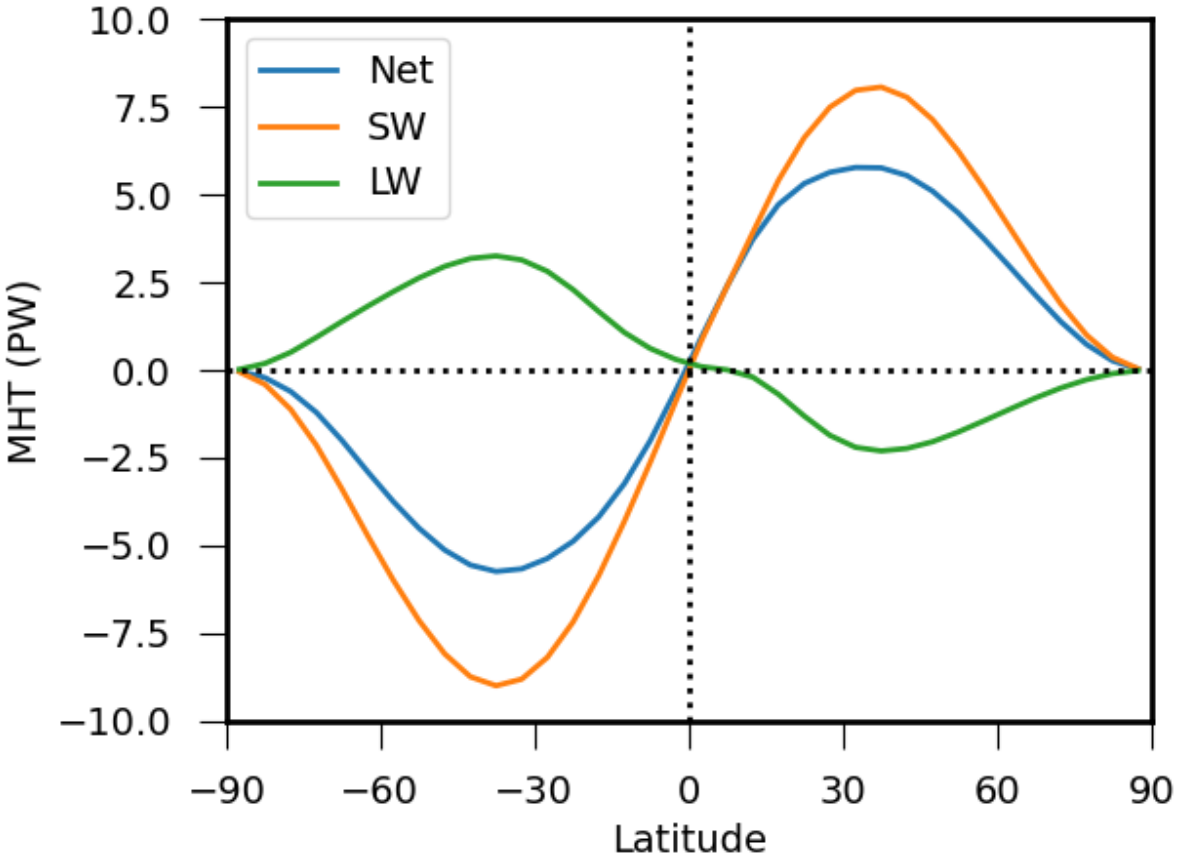


Fig. 47: *The implied heat transport due to the total net flux (blue), split into the contributions from the S…*

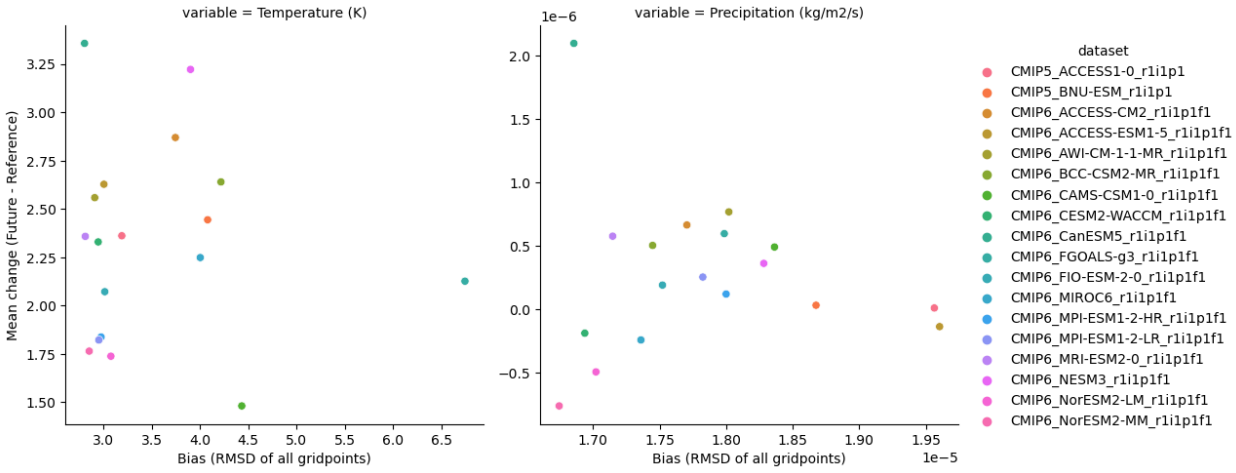


Fig. 48: *Bias and change for each variable*

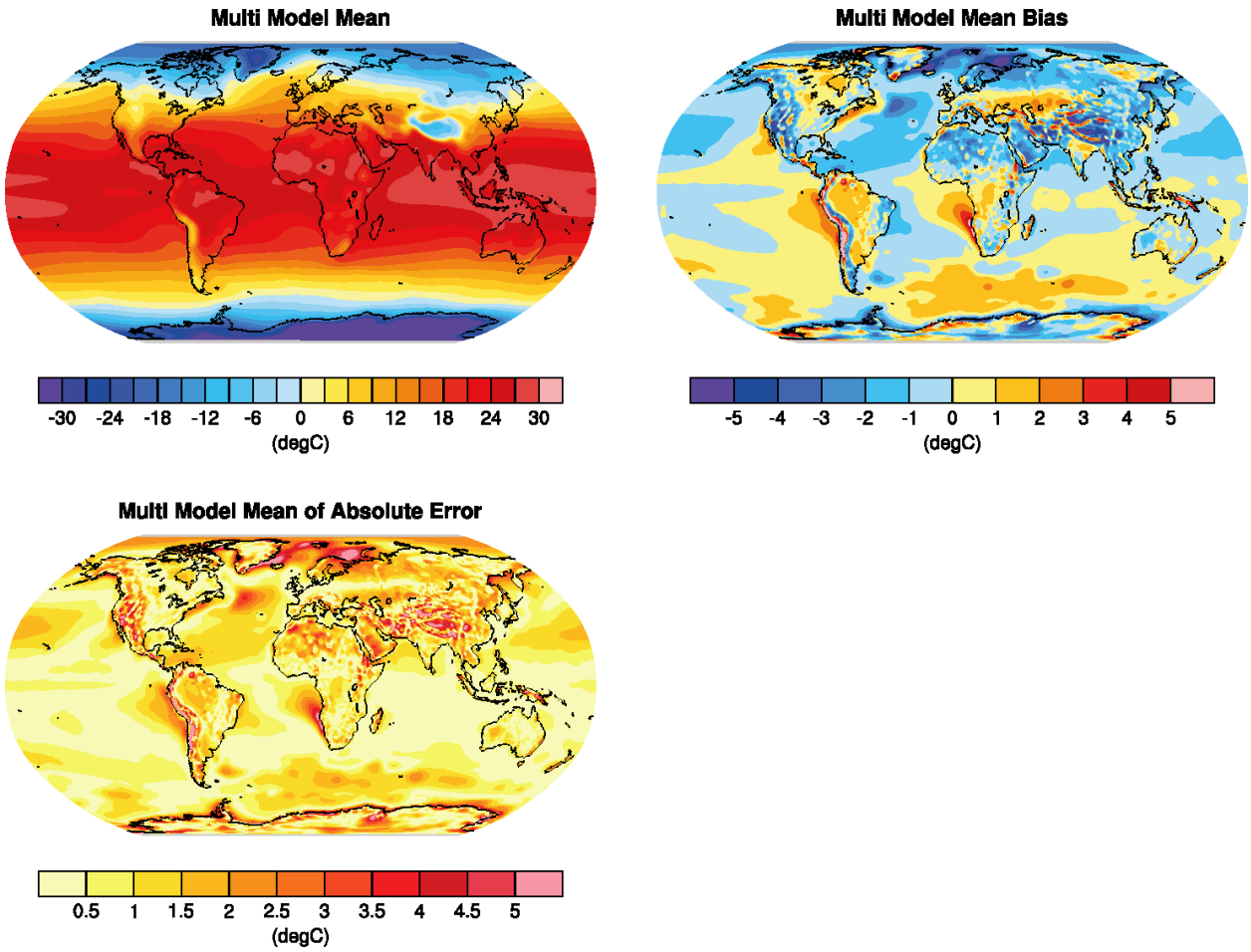


Fig. 49: *Figure 9.2 a,b,c: Annual-mean surface air temperature for the period 1980-2005. a) multi-model mean,…*

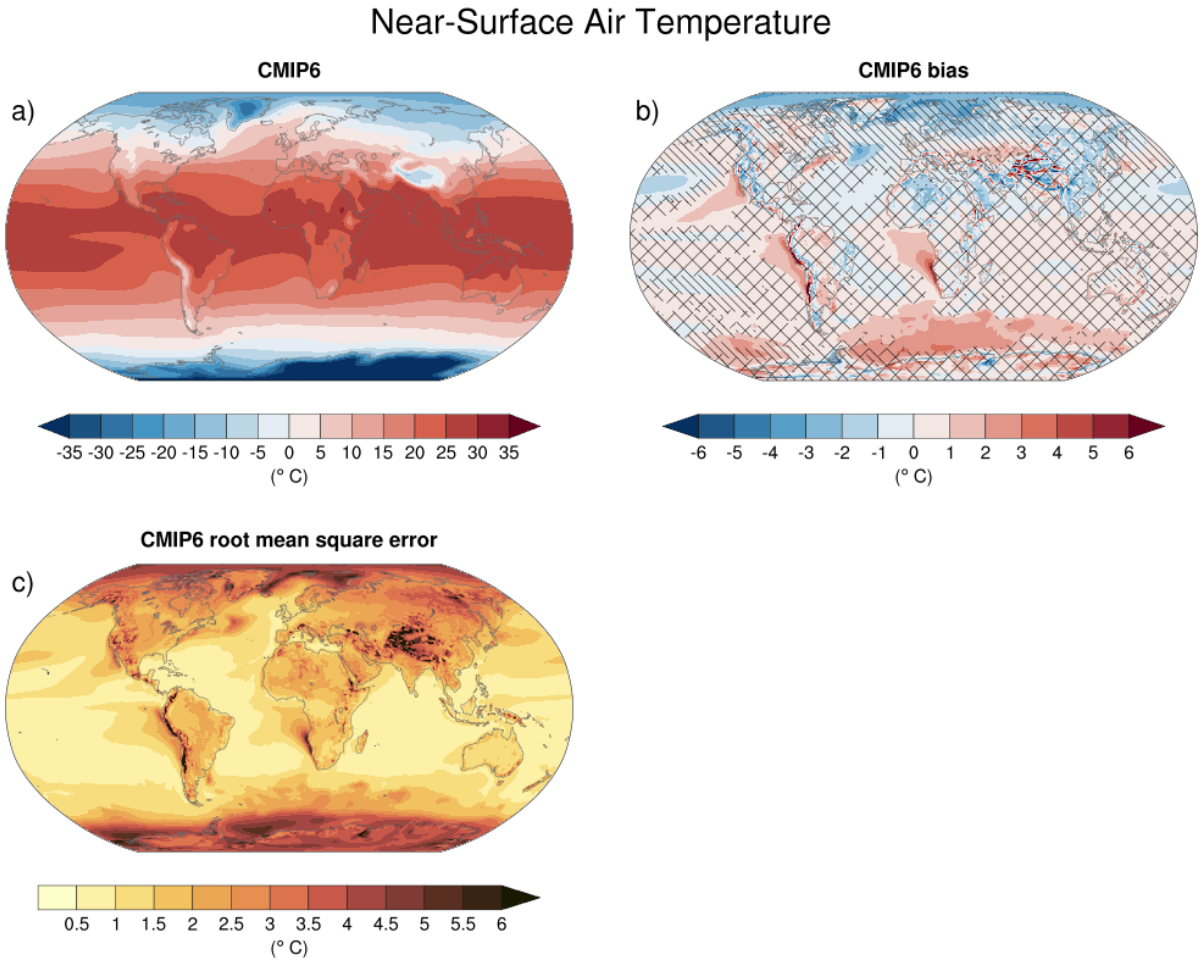


Fig. 50: *Figure 3.3: Annual mean near-surface (2 m) air temperature (°C) for the period 1995-2014. (a) Multi-…*

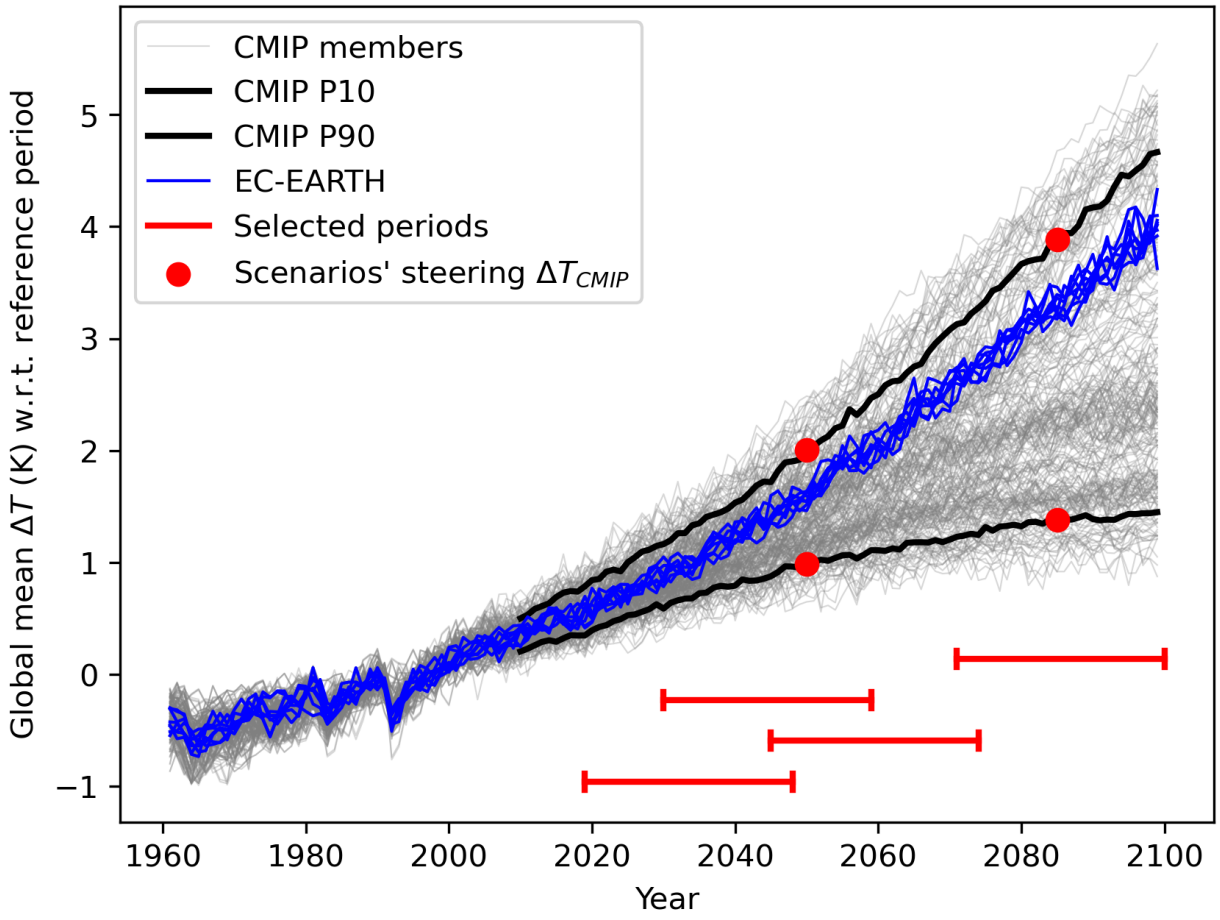


Fig. 51: *CMIP spread in global temperature change, highlighting selected steering parameters and resampling p…*

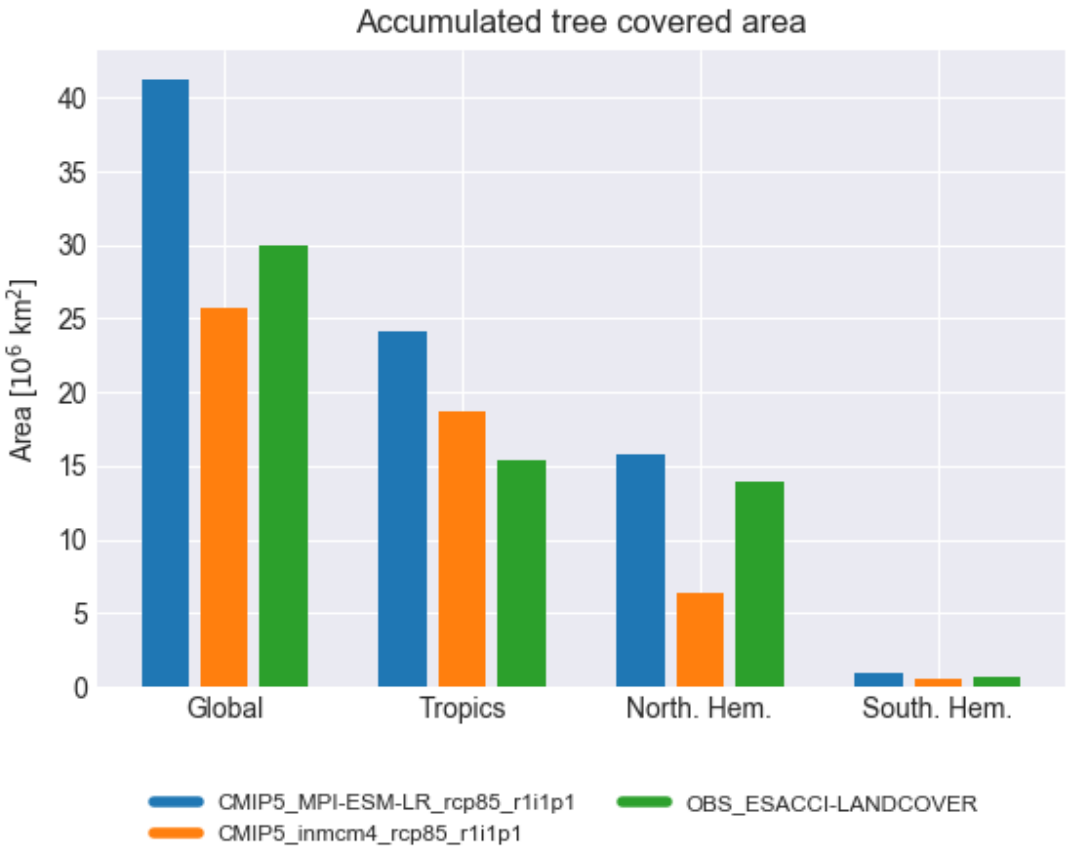


Fig. 52: *Accumulated tree covered area for different regions and experiments.*

CHAPTER

FOUR

GETTING STARTED

4.1 Installation

ESMValTool supports Python 3.12 and later and requires Linux or MacOS. Successful usage on Windows has been reported by following the Linux installation instructions with WSL.

ESMValTool can be installed in multiple ways.

Recommended installation method:

Install the *mamba package manager* and then follow the instructions for

- *ESMValTool installation on Linux*
- *ESMValTool installation on MacOS*.

Further options for installation are:

- *From the source code* available at https://github.com/ESMValGroup/ESMValTool;
- *From pre-installed versions on HPC clusters*;
- *Deployment through a Docker container* (see https://www.docker.com);
- *Deployment through a Singularity container* (see https://sylabs.io/guides/latest/user-guide/);
- *Installation with pip* (see https://pypi.org);

The next sections will detail the procedure to install ESMValTool through each of these methods.

There is also a lesson available in the ESMValTool tutorial that describes the installation of the ESMValTool in more detail. It can be found here.

See *common installation issues* if you run into trouble.

4.1.1 Mamba/Conda installation

In order to install ESMValTool and its dependencies from conda-forge, you will first need to install the mamba package manager. We recommend using mamba as a package manager for your conda environments instead of conda because it is faster.

For a minimal mamba installation (recommended) go to https://mamba.readthedocs.io/en/latest/installation/micromamba-installation.html.

Note

It is recommended that you always use the latest version of mamba, as problems have been reported when trying to use older versions.

Note

Some systems provide a pre-installed version of conda or mamba (e.g. via the module environment). However, several users reported problems when installing with such versions. It is therefore preferable to use a local, fully user-controlled mamba installation.

First download the installation file for Linux or MacOSX. After downloading the installation file from one of the links above, execute it by running (Linux example):

```
bash Miniforge3-Linux-x86_64.sh
```

and follow the instructions on your screen.

Note

Make sure to choose an installation location where you have at least 10 GB of disk space available.

During installation, mamba will ask you if you want `mamba` to be automatically loaded from your `.bashrc` or `.bash-profile` files. It is recommended that you answer yes. If you answered no, you can load the correct paths and environment variables later by running:

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

where `<prefix>` is the installation location of mamba (e.g. `/home/$USER/miniforge3` if you chose the default installation path).

If you use another shell than Bash, have a look at the available configurations in the `<prefix>/etc/profile.d` directory.

You can check that mamba installed correctly by running

```
which mamba
```

this should show the path to your mamba executable, e.g. `~/miniforge3/bin/mamba`.

It is recommended to update both mamba and conda after installing:

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

ESMValTool installation on Linux

Once you have installed the mamba package manager, you can install the entire ESMValTool package by running:

```
mamba create --name esmvaltool esmvaltool
```

It is also possible to install just a subset of the ESMValTool dependencies by installing one or more of the *subpackages* described in the next section.

The command above will create a new conda environment called `esmvaltool`, and install ESMValTool in it. Of course it is also possible to choose a different name than `esmvaltool` for the environment.

Note

Creating a new conda environment is often much faster and more reliable than trying to update an existing conda environment. Therefore it is recommended that you create a new environment when you want to upgrade to the latest version.

The next step is to check that the installation works properly.

First activate the environment with the command:

```
conda activate esmvaltool
```

and then run the tool with the command:

```
esmvaltool --help
```

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

Installation of subpackages

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

There are three language specific packages available:

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

The main `esmvaltool` package contains all three 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 (`.py` for Python, `.ncl` for NCL, `.R` for R diagnostics).

To install support for diagnostics written in Python and NCL into an existing environment, run

```
mamba install esmvaltool-python esmvaltool-ncl
```

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

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.

ESMValTool installation on MacOS

The Python diagnostics of the ESMValTool are supported on MacOS, but NCL, and R are not. If any of these are needed, deployment through a *Docker* container is advised.

The `esmvaltool-python` diagnostics can be installed as follows:

First, ensure mamba is installed (see *install_with_mamba* for more details).

Create a new environment with the `esmvaltool-python` package:

```
mamba create --name esmvaltool esmvaltool-python
```

Activate the new environment:

```
conda activate esmvaltool
```

Confirm that the ESMValTool is working with:

```
esmvaltool --help
```

Note that some recipes may depend on the OpenMP library, which does not install via mamba on MacOS. To install this library, run:

```
brew install libomp
```

to install the library with Homebrew. In case you do not have Homebrew, follow installation instructions here.

4.1.2 Install from source

Installing the tool from source is recommended if you need the very latest features or if you would like to contribute to its development.

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
```

or

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

if you prefer to connect to the repository over SSH.

The command above will create a folder called `ESMValTool` containing the source code of the tool in the current working directory.

Note

Using SSH is much more convenient if you push to the repository regularly (recommended to back up your work), because then you do not need to type your password over and over again. See this guide for information on how to set it up if you have not done so yet. If you are developing ESMValTool on a shared compute cluster, you can set up SSH agent forwarding to use your local SSH keys also from the remote machine.

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

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

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.

Install ESMValTool

It is recommended to use pixi to manage ESMValTool dependencies. Follow the pixi installation instructions to install it.

To create and activate a software environment to develop ESMValTool, go to the directory containing the ESMValTool source code that you just downloaded. It is called `ESMValTool` if you did not choose a different name.

```
cd ESMValTool
```

and run the command:

```
pixi shell --frozen
```

This will install all of the required dependencies for running and developing Python diagnostics.

Tip

We recommend running all pixi commands with the `--frozen` flag. This flag instructs pixi to use the versions of the packages listed in the pixi lock file (`pixi.lock`). Using this option makes all pixi commands run faster because they do not need to "solve" the environment (i.e. find out which combination of package versions is compatible and can be installed) and makes your analyses more reproducible because you know exactly which version of which software package was used. We recommend adding `export PIXI_FROZEN=true` to your `~/.bashrc` file to make this the default behavior.

Tip

To exit the pixi environment, run `exit` or press `Ctrl+D`.

If you are on Linux, it is also possible to install the dependencies for NCL and R diagnostics by running

```
pixi shell -e esmvaltool-dev
```

or just the dependencies for NCL diagnostics with `pixi shell -e esmvaltool-ncl-dev` and for R diagnostics with `pixi shell -e esmvaltool-r-dev`. There are also environments with the dependencies for running diagnostics in specific languages without the development dependencies, i.e. `pixi shell -e esmvaltool-python` for running Python diagnostics, `pixi shell -e esmvaltool-ncl` for running NCL diagnostics, `pixi shell -e esmvaltool-r` for running R diagnostics, and `pixi shell -e esmvaltool` for running any diagnostic. The environment names correspond to the ESMValTool subpackages described in *Installation of subpackages* (except that `esmvaltool-python-dev` is called `default`) and the `-dev` suffix indicates that additional development dependencies are included.

Tip

If you find that solving the environments uses too much memory, you can set the number of parallel solves to one by running `pixi config set concurrency.solves 1`.

If you are planning to do any coding, install the *Pre-commit* hooks by running:

```
pre-commit install
```

these will make sure that when you commit your changes, they will be formatted correctly.

Check your ESMValTool installation

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.

Note

MacOS users: some recipes may depend on the OpenMP library, which does not install via mamba on MacOS. Instead run

```
brew install libomp
```

to install the library with Homebrew. In case you do not have Homebrew, follow installation instructions here.

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

```
pytest -m "not installation"
```

or if you want to run the full test suite remove the `-m "not installation"` flag; also if you want to run the tests on multiple threads, making the run faster, use the *-n N* flag where N is the number of available threads e.g:

```
pytest -n 4
```

This concludes the installation from source guide. However, if you would like to do development work on ESMValCore, please read on.

Using the development version of the ESMValCore package

If you need the latest developments of the ESMValCore package, you can install it from source into the same pixi environment.

Attention

The recipes and diagnostics in the ESMValTool repository are compatible with the latest released version of ESMValCore. Using the development version of the ESMValCore package is only recommended if you are planning to develop new features for ESMValCore, e.g. you want to implement a new preprocessor function.

First follow the steps in the section above to *install ESMValTool from source*.

If you just want to use the latest features of ESMValCore, you can create a new pixi environment with the command:

```
pixi shell -e esmvalcore-dev
```

and verify that you now have a development version of ESMValCore installed by running

```
esmvaltool version
```

If you are planning to do development work on ESMValCore, you will need to clone the ESMValCore repository and tell pixi to use it.

Go to the place where you would like to keep the ESMValCore source code and clone the ESMValCore github repository:

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

or

```
git clone git@github.com:ESMValGroup/ESMValCore
```

The command above will create a folder called `ESMValCore` containing the source code of the tool in the current working directory.

Edit the `pyproject.toml` file in the ESMValTool source code and remove this line:

```
"ESMValCore" = { git = "https://github.com/ESMValGroup/ESMValCore.git", branch = "main" }
```

and uncomment this line:

```
# "ESMValCore" = { path = "../ESMValCore", editable = true }
```

Make sure the path in the line you just uncommented points to the ESMValCore repository you just cloned.

Finally, also install the ESMValCore *Pre-commit* hooks by going into the directory containing the ESMValCore source code and running:

```
pre-commit install
```

these will make sure that when you commit your changes, they will be formatted correctly.

4.1.3 Pre-installed versions on HPC clusters / other servers

ESMValTool is available on the HPC clusters CEDA-JASMIN and DKRZ-Levante, and on the Met Office Linux estate, so there is no need to install ESMValTool if you are just running recipes:

- CEDA-JASMIN: *esmvaltool* is available on the scientific compute nodes (*sciX.jasmin.ac.uk* where *X = 1, 2, 3, 4, 5*) after login and module loading via *module load esmvaltool*; see the helper page at CEDA .
- DKRZ-Levante: *esmvaltool* is available on login nodes (*levante.dkrz.de*) after login and module loading via *module load esmvaltool*; the command *module help esmvaltool* provides some information about the module. A Jupyter kernel based on the latest module is available from DKRZ-JupyterHub.
- Met Office: *esmvaltool* is available on the Linux estate after login and module loading via *module load*; see the ESMValTool Community of Practice SharePoint site for more details.
- NSC-Tetralith and Freja: *esmvaltool* is available after login and module loading via *module load esmvaltool*.

The ESMValTool Tutorial provides a quickstart guide that is particularly suited for new users that have an access to pre-installed version of ESMValTool.

Information on how to request an account at CEDA-JASMIN and DKRZ-Levante and to get started with these HPC clusters can be found on the setup page of the tutorial here.

4.1.4 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 main 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.

4.1.5 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
```

4.1.6 Pip installation

It is also possible to install ESMValTool from PyPI. However, this requires first installing dependencies that are not available on PyPI in some other way. The list of required dependencies can be found in `pyproject.toml`.

Warning

It is recommended to use the installation with mamba instead, as it may not be easy to install the correct versions of all dependencies.

After installing the dependencies that are not available from PyPI, install ESMValTool and any remaining Python dependencies with the command:

```
pip install esmvaltool
```

4.1.7 Common installation problems and their solutions

Problems with proxies

If you are installing ESMValTool from source from behind a proxy that does not trust the usual PyPI 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 --no-deps -e .[develop]
```

If R packages fail to download, you might be able to solve this by setting the environment variable `http_proxy` to the correct value, e.g. in bash:

```
export http_proxy=http://user:pass@proxy_server:port
```

the username and password can be omitted if they are not required. See e.g. here for more information.

Anaconda servers connection issues

HTTP connection errors (of e.g. type 404) to the Anaconda servers are rather common, and usually a retry will solve the problem.

Installation of R packages fails

Problems have been reported if the R interpreter was made available through the `module load` command in addition to installation from mamba. If your ESMValTool conda environment is called `esmvaltool` and you want to use the R interpreter installed from mamba, the path to the R interpreter should end with `mamba/envs/esmvaltool/bin/R` or `conda/envs/esmvaltool/bin/R`. When the conda environment for ESMValTool is activated, you can check which R interpreter is used by running

```
which R
```

The Modules package is often used by system administrators to make software available to users of scientific compute clusters. To list any currently loaded modules run `module list`, run `module help` or `man module` for more information about the Modules package.

Problems when using ssh

If you log in to 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`.

Problems when updating the conda environment

Usually mamba is much better at solving new environments than updating older environments, so it is often a good idea to create a new environment if updating does not work.

Do not run `mamba 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 tool.

4.2 Configuration

The `esmvaltool` command is provided by the ESMValCore package, the documentation on configuring ESMValCore can be found here. An overview of all configuration options can be found here. In particular, it is recommended to read the section on how to specify configuration options and the section on data sources.

To install the default configuration in the default location, run

```
esmvaltool config copy defaults/config-user.yml
```

Note that this 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.

There is a lesson available in the ESMValTool tutorial that describes how to personalize the configuration. It can be found at this site.

4.3 Running

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

4.3.1 Running your first recipe

There is a step-by-step tutorial available in the ESMValTool tutorial on how to run your first recipe. It can be found here.

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

This recipe finds data from BCC-ESM1 and CanESM2 and creates two plot types:

- a global map plot that shows the monthly mean 2m surface air temperature in January 2000.
- a time series plot that shows the globally averaged annual mean 2m surface air temperature and compares it to the one in Amsterdam.

To run this recipe and automatically download the required climate data from ESGF to the local directory, you need to configure the data sources ESMValCore can use so it can find the data. Next, you can run the recipe with the command:

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

Recall that the chapter on *configuring ESMValTool* provides an explanation of how to set up the configuration.

See running esmvaltool in the ESMValCore documentation for a more complete introduction to the `esmvaltool` command.

4.3.2 Available diagnostics and metrics

Although ESMValTool can be used to download data, analyze it using ESMValCore's preprocessing modules, and the creation of your own analysis code, its main purpose is the continuously growing set of diagnostics and metrics that it directly provides to the user. These metrics and diagnostics are provided as a set of preconfigured recipes that users can run or customize for their own analysis. The latest list of available recipes can be found *here*.

In order to make the management of these installed recipes easier, ESMValTool provides the `recipes` command group with utilities that help the users in discovering and customizing the provided recipes.

The first command in this group allows users to get the complete list of installed recipes printed to the console:

```
esmvaltool recipes list
```

If the user then wants to explore any one of these recipes, they can be printed using the following command

```
esmvaltool recipes show recipe_name.yml
```

Note that there is no `recipe_name.yml` shipped with ESMValTool, replace this with a recipes that is available, for example examples/recipe_python.yml. Finally, to get a local copy that can then be customized and run, users can run the following command

```
esmvaltool recipes get recipe_name.yml
```

Note that the `esmvaltool run recipe_name.yml` command will first look if `recipe_name.yml` is the path to an existing file. If this is the case, it will run that recipe. If not, it will look if it is a relative path to an existing recipe with respect to the recipes directory in your ESMValTool installation and run that.

4.3.3 Running multiple recipes

Have a look at *Running multiple recipes* if you are interested in running multiple recipes in parallel.

4.4 Output

ESMValTool automatically generates a new output directory with every run. The location is determined by the configuration option `output_dir`, 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.

4.4.1 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 configuration option `save_intermediary_cubes` is set to `true` , then the intermediary cubes will also be saved here. This option is set to `false` by default.

If the configuration option `remove_preproc_dir` is set to `true` , then the preproc directory will be deleted after the run completes. This option is set to `true` by default.

4.4.2 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.

4.4.3 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.

4.4.4 Plots

The plots directory is where diagnostics save their output figures. These plots are saved in the format requested by the configuration option `output_file_type`.

4.4.5 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 configuration (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.

4.4.6 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 available in the shared directory in the diagnostics directory.

CHAPTER

FIVE

RECIPES

A website displaying results produced with the latest release of ESMValTool for all available recipes can be accessed here.

5.1 General-purpose recipes

Recipes that use highly customizable diagnostics which are designed to plot a large variety of input data.

5.1.1 Model Benchmarking

Overview

These recipes and diagnostics are based on *recipe_monitor* that allow plotting arbitrary preprocessor output, i.e., arbitrary variables from arbitrary datasets. An extension of these diagnostics is used to benchmark a model simulation with other datasets (e.g. CMIP6). The benchmarking features are described in Lauer et al..

Available recipes and diagnostics

Recipes are stored in *recipes/model_evaluation*

- recipe_model_benchmarking_annual_cycle.yml
- recipe_model_benchmarking_boxplots.yml
- recipe_model_benchmarking_diurnal_cycle.yml
- recipe_model_benchmarking_maps.yml
- recipe_model_benchmarking_timeseries.yml
- recipe_model_benchmarking_zonal.yml

Diagnostics are stored in *diag_scripts/monitor/*

- *multi_datasets.py*: Monitoring diagnostic to show multiple datasets in one plot (incl. biases).

Recipe settings

See *multi_datasets.py* for a list of all possible configuration options that can be specified in the recipe.

Note

Please note that exactly one dataset (the dataset to be benchmarked) needs to specify the facet `benchmark_dataset: true` in the dataset entry of the recipe. For line plots (i.e. annual cycle, diurnal cycle, time series), it is recommended, to specify a particular line color and line style in the `scripts` section of the

recipe for the dataset to be benchmarked (`benchmark_dataset: true`) so that this dataset is easy to identify in the plot. In the example below, MIROC6 is the dataset to be benchmarked and ERA5 is used as a reference dataset.

```yaml
scripts:
  allplots:
    script: monitor/multi_datasets.py
    plot_folder: '{plot_dir}'
    plot_filename: '{plot_type}_{real_name}_{mip}'
    group_variables_by: variable_group
    facet_used_for_labels: alias
    plots:
      diurnal_cycle:
        legend_kwargs:
          loc: upper right
        plot_kwargs:
          'MIROC6':
            color: red
            label: '{alias}'
            linestyle: '-'
            linewidth: 2
            zorder: 4
          ERA5:
            color: black
            label: '{dataset}'
            linestyle: '-'
            linewidth: 2
            zorder: 3
          MultiModelPercentile10:
            color: gray
            label: '{dataset}'
            linestyle: '--'
            linewidth: 1
            zorder: 2
          MultiModelPercentile90:
            color: gray
            label: '{dataset}'
            linestyle: '--'
            linewidth: 1
            zorder: 2
          default:
            color: lightgray
            label: null
            linestyle: '-'
            linewidth: 1
            zorder: 1
```

Variables

Any, but the variables' number of dimensions should match the ones expected by each plot.

References

- Lauer, A., L. Bock, B. Hassler, P. Jöckel, L. Ruhe, and M. Schlund: Monitoring and benchmarking Earth System Model simulations with ESMValTool v2.12.0, EGUsphere [preprint], https://doi.org/10.5194/egusphere-2024-1518, 2024.

Example plots

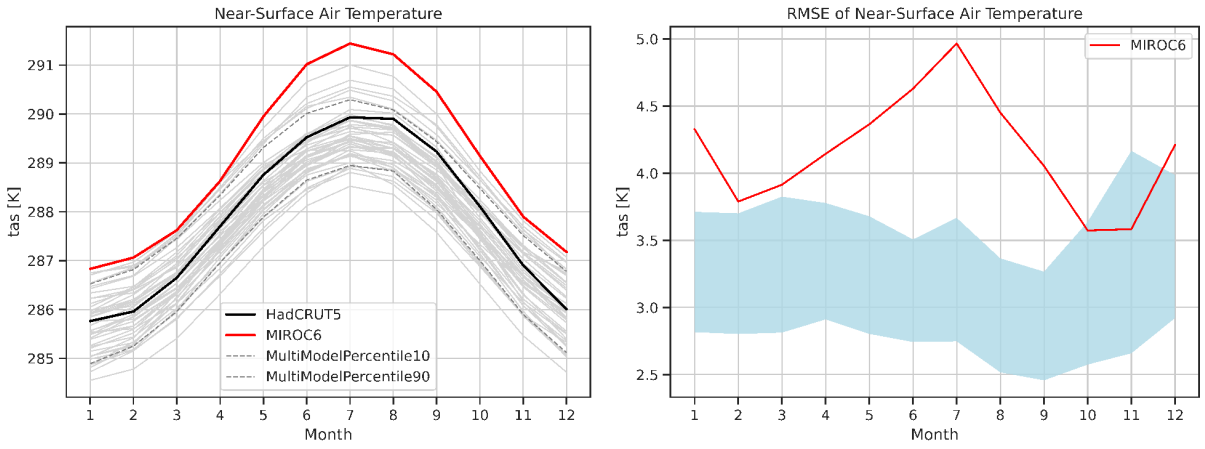


Fig. 1: (Left) Multi-year global mean (2000-2004) of the seasonal cycle of near-surface temperature in K from a simulation of MIROC6 and the reference dataset HadCRUT5 (black). The thin gray lines show individual CMIP6 models used for comparison, the dashed gray lines show the 10% and 90% percentiles of these CMIP6 models. (Right) same as (left) but for area-weighted RMSE of near-surface temperature. The light blue shading shows the range of the 10% to 90% percentiles of RMSE values from the ensemble of CMIP6 models used for comparison. Created with recipe_model_benchmarking_annual_cycle.yml.

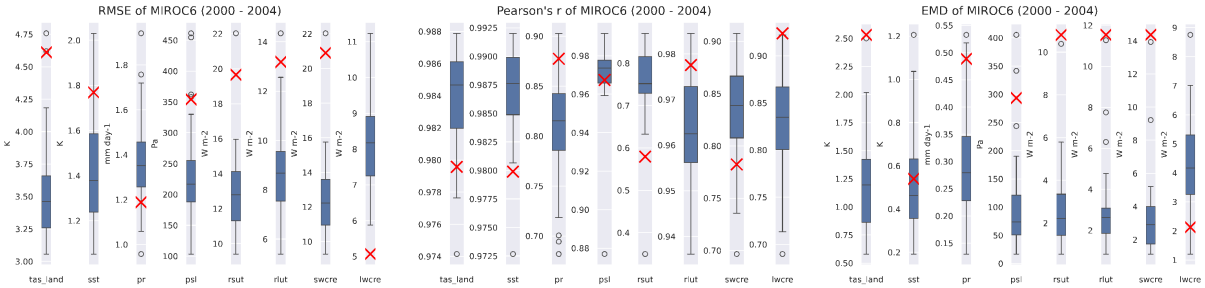


Fig. 2: (Left) Global area-weighted RMSE (smaller=better), (middle) weighted Pearson's correlation coefficient (higher=better) and (right) weighted Earth mover's distance (smaller=better) of the geographical pattern of 5-year means of different variables from a simulation of MIROC6 (red cross) in comparison to the CMIP6 ensemble (boxplot). Reference datasets for calculating the three metrics are: near-surface temperature (tas): HadCRUT5, surface temperature (ts): HadISST, precipitation (pr): GPCP-SG, air pressure at sea level (psl): ERA5, shortwave (rsut) longwave (rlut) radiative fluxes at TOA and shortwave (swcre) and longwave (lwcre) cloud radiative effects: CERES-EBAF. Each box indicates the range from the first quartile to the third quartile, the vertical lines show the median, and the whiskers the minimum and maximum values, excluding the outliers. Outliers are defined as being outside 1.5 times the interquartile range. Created with recipe_model_benchmarking_boxplots.yml.

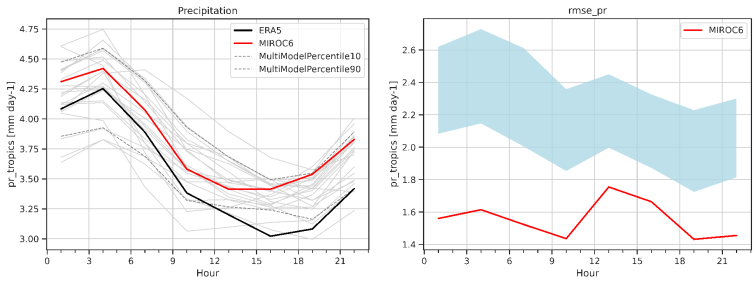


Fig. 3: Area-weighted RMSE of the annual mean diurnal cycle (year 2000) of precipitation averaged over the tropical ocean (ocean grid cells in the latitude belt 30°S to 30°N) from a simulation of MIROC6 averaged compared with ERA5 data (black). The light blue shading shows the range of the 10% to 90% percentiles of RMSE values from the ensemble of CMIP6 models used for comparison. Created with recipe_benchmarking_diurnal_cycle.yml.

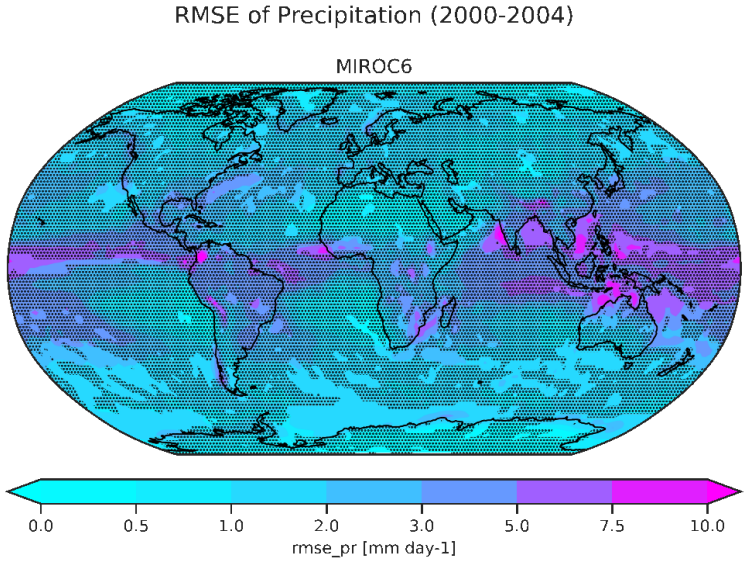


Fig. 4: 5-year annual mean (2000-2004) area-weighted RMSE of the precipitation rate in mm day-1 from a simulation of MIROC6 compared with GPCP-SG data. The stippled areas mask grid cells where the RMSE is smaller than the 90% percentile of RMSE values from an ensemble of CMIP6 models. Created with recipe_model_benchmarking_maps.yml

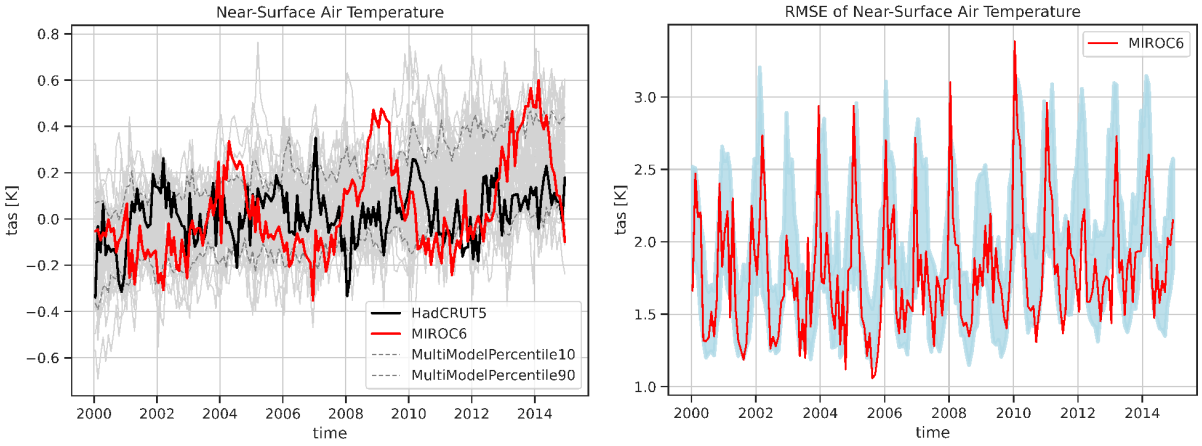


Fig. 5: (Left) Time series from 2000 through 2014 of global average monthly mean temperature anomalies (reference period 2000-2009) of the near-surface temperature in K from a simulation of MIROC6 (red) and the reference dataset HadCRUT5 (black). The thin gray lines show individual CMIP6 models used for comparison, the dashed gray lines show the 10% and 90% percentiles of these CMIP6 models. (Right) same as (left) but for area-weighted RMSE of the near-surface air temperature. The light blue shading shows the range of the 10% to 90% percentiles of RMSE values from the ensemble of CMIP6 models used for comparison. Created with recipe_model_benchmarking_timeseries.yml.

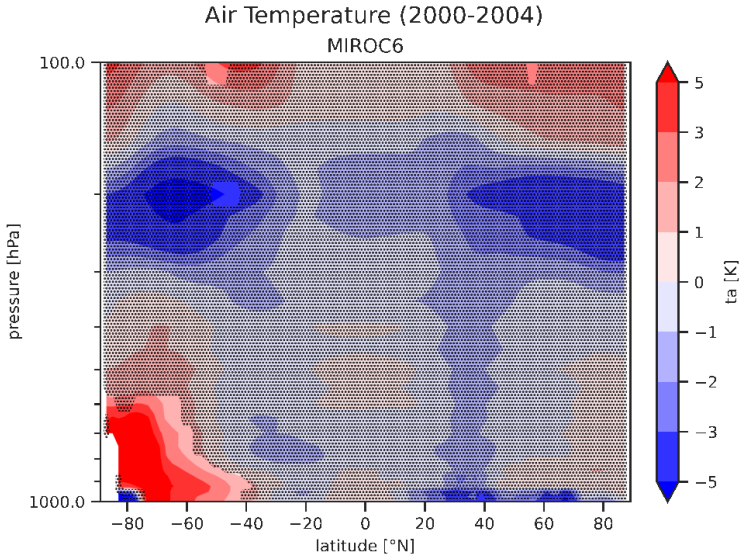


Fig. 6: 5-year annual mean bias (2000-2004) of the zonally averaged temperature in K from a historical simulation of MIROC6 compared with ERA5 reanalysis data. The stippled areas mask grid cells where the absolute BIAS ($|BIAS|$) is smaller than the maximum of the absolute 10% ($|p10|$) and the absolute 90% ($|p90|$) percentiles from an ensemble of CMIP6 models, i.e. $|BIAS| \geq max(|p10|, |p90|)$. Created with recipe_model_benchmarking_zonal.yml.

5.1.2 General model evaluation

Overview

These recipes and diagnostics provide a basic climate model evaluation with observational data. This is especially useful to get an overview of the performance of a simulation. The diagnostics used here allow plotting arbitrary preprocessor output, i.e., arbitrary variables from arbitrary datasets.

Available recipes and diagnostics

Recipes are stored in *recipes/model_evaluation*

- recipe_model_evaluation_basics.yml
- recipe_model_evaluation_clouds_clim.yml
- recipe_model_evaluation_clouds_cycles.yml
- recipe_model_evaluation_precip_diurnal_cycle.yml
- recipe_model_evaluation_precip_hist.yml
- recipe_model_evaluation_precip_zonal.yml

Diagnostics are stored in *diag_scripts/monitor/*

- *multi_datasets.py*: Monitoring diagnostic to show multiple datasets in one plot (incl. biases).
- *plot_histogram.py*: Plotting of 1D histograms with precomputed histogram data (`histogram` preprocessor).
- *hydrology/hour_of_max_precipitation.py*: Calculating and plotting hour of maximum precipitation.

User settings

It is recommended to use a vector graphic file type (e.g., SVG) for the output format when running this recipe, i.e., run the recipe with the configuration options `output_file_type: svg`. Note that map and profile plots are rasterized by default. Use `rasterize: false` in the recipe to disable this.

Recipe settings

A list of all possible configuration options that can be specified in the recipe is given for each diagnostic individually (see links given for the available diagnostics in the previous section).

Variables

Any, but the variables' number of dimensions should match the ones expected by each diagnostic (see links given for the available diagnostics in the previous section).

Example plots

5.1.3 Monitor

Overview

These recipes and diagnostics allow plotting arbitrary preprocessor output, i.e., arbitrary variables from arbitrary datasets. In addition, a *base class* is provided that allows a convenient interface for all monitoring diagnostics.

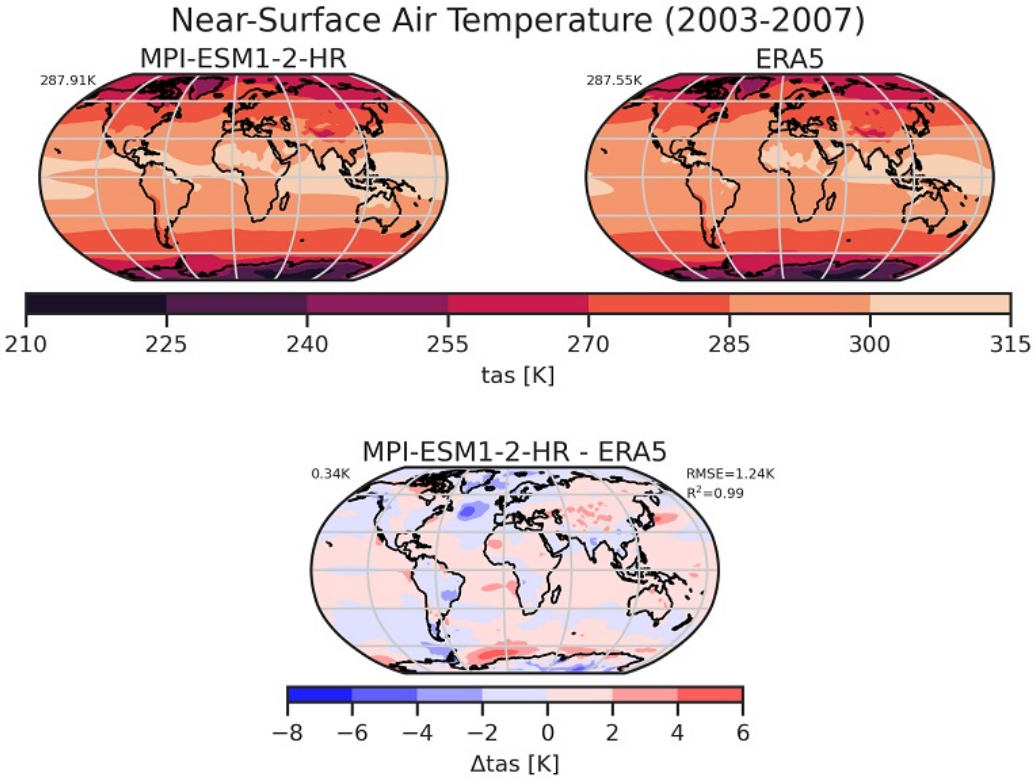


Fig. 7: Global climatology of 2m near-surface air temperature.

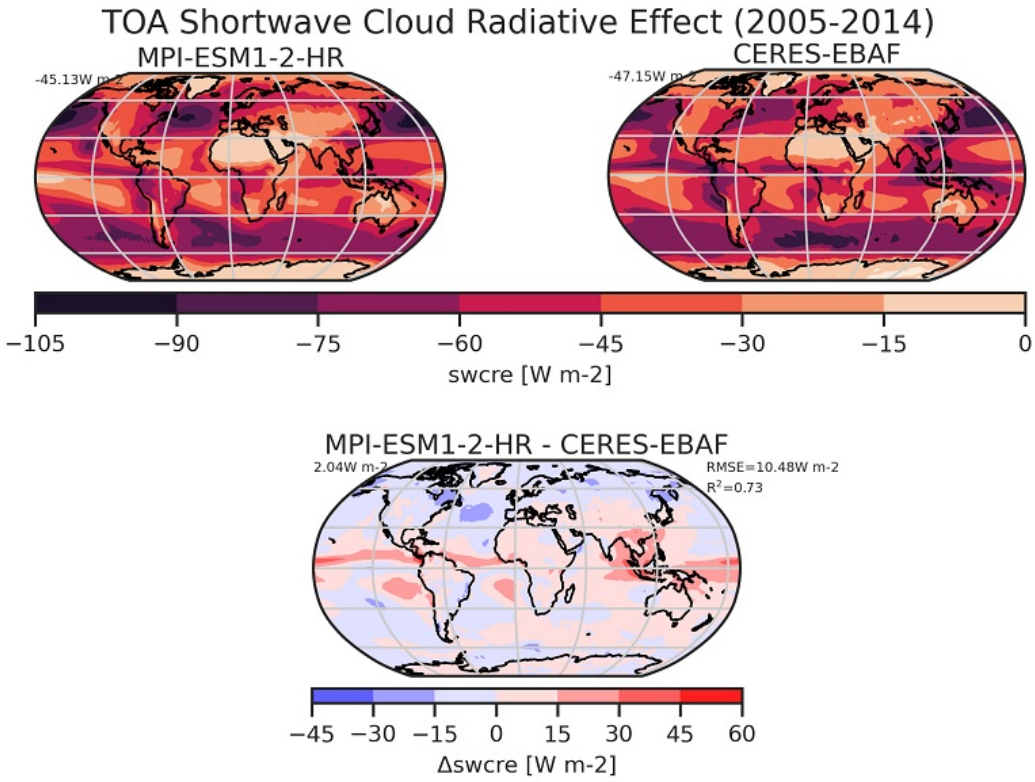


Fig. 8: Global climatology of the shortwave cloud radiative effect (SWCRE).

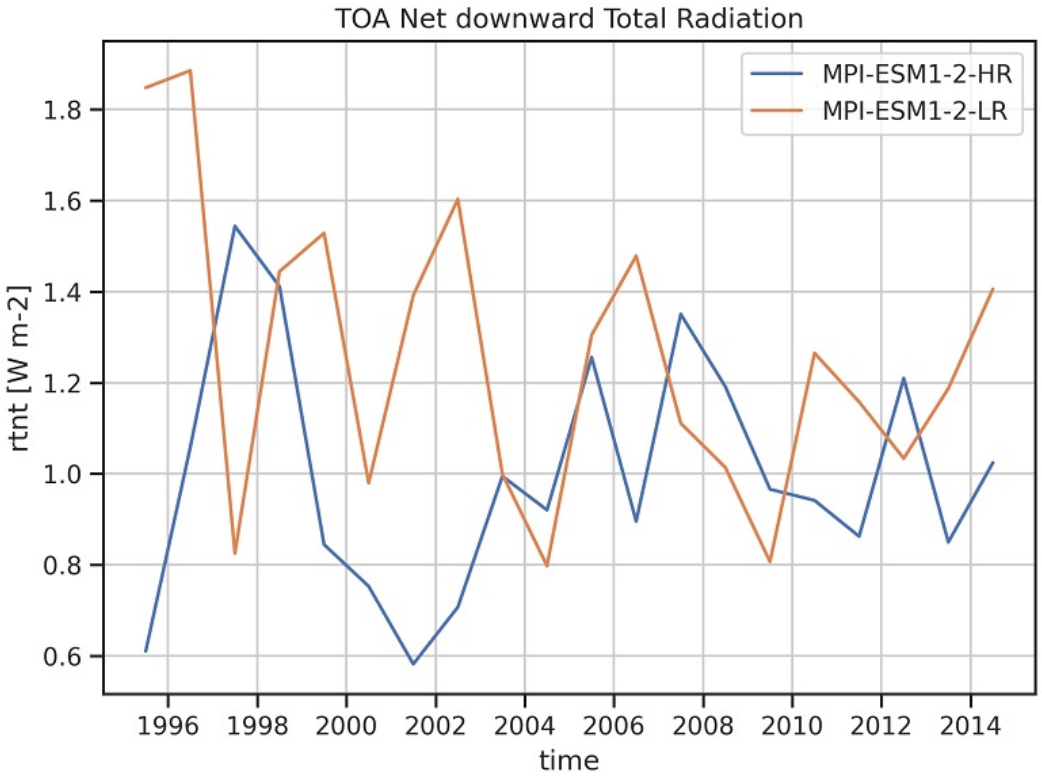


Fig. 9: Time series of the global mean top-of-the-atmosphere net radiative flux.

Available recipes and diagnostics

Recipes are stored in *recipes/monitor*

- recipe_monitor.yml
- recipe_monitor_with_refs.yml

Diagnostics are stored in *diag_scripts/monitor/*

- *monitor.py*: Monitoring diagnostic to plot arbitrary preprocessor output.
- *compute_eofs.py*: Monitoring diagnostic to plot EOF maps and associated PC timeseries.
- *multi_datasets.py*: Monitoring diagnostic to show multiple datasets in one plot (incl. biases).

User settings

It is recommended to use a vector graphic file type (e.g., SVG) for the output format when running this recipe, i.e., run the recipe with the configuration options `output_file_type: svg`. Note that map and profile plots are rasterized by default. Use `rasterize_maps: false` or `rasterize: false` (see *Recipe settings*) in the recipe to disable this.

Recipe settings

A list of all possible configuration options that can be specified in the recipe is given for each diagnostic individually (see previous section).

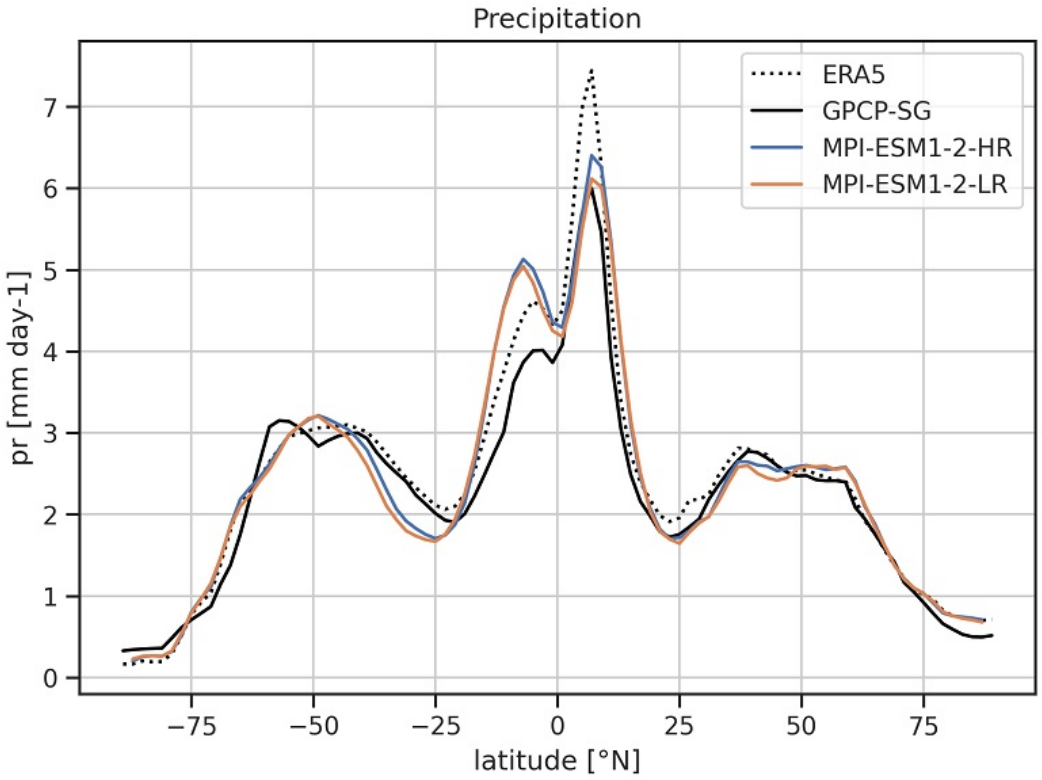


Fig. 10: Zonal mean precipitation.

Monitor configuration file

In addition, the following diagnostics support the use of a dedicated monitor configuration file:

- monitor.py
- compute_eofs.py

This file is a yaml file that contains map and variable specific options in two dictionaries `maps` and `variables`.

Each entry in `maps` corresponds to a map definition. Example:

```
maps:
   global: # Map name, choose a meaningful one
      projection: PlateCarree # Cartopy projection to use
      projection_kwargs: # Dictionary with Cartopy's projection keyword arguments.
         central_longitude: 285
      smooth: true # If true, interpolate values to get smoother maps. If not, all␣
↪points in a cells will get the exact same color
      lon: [-120, -60, 0, 60, 120, 180] # Set longitude ticks
      lat: [-90, -60, -30, 0, 30, 60, 90] # Set latitude ticks
      colorbar_location: bottom
      extent: null # If defined, restrict the projection to a region. Format [lon1, lon2,
↪ lat1, lat2]
      suptitle_pos: 0.87 # Title position in the figure.
```

Each entry in `variables` corresponds to a variable definition. Use the default entry to apply generic options to all variables. Example:

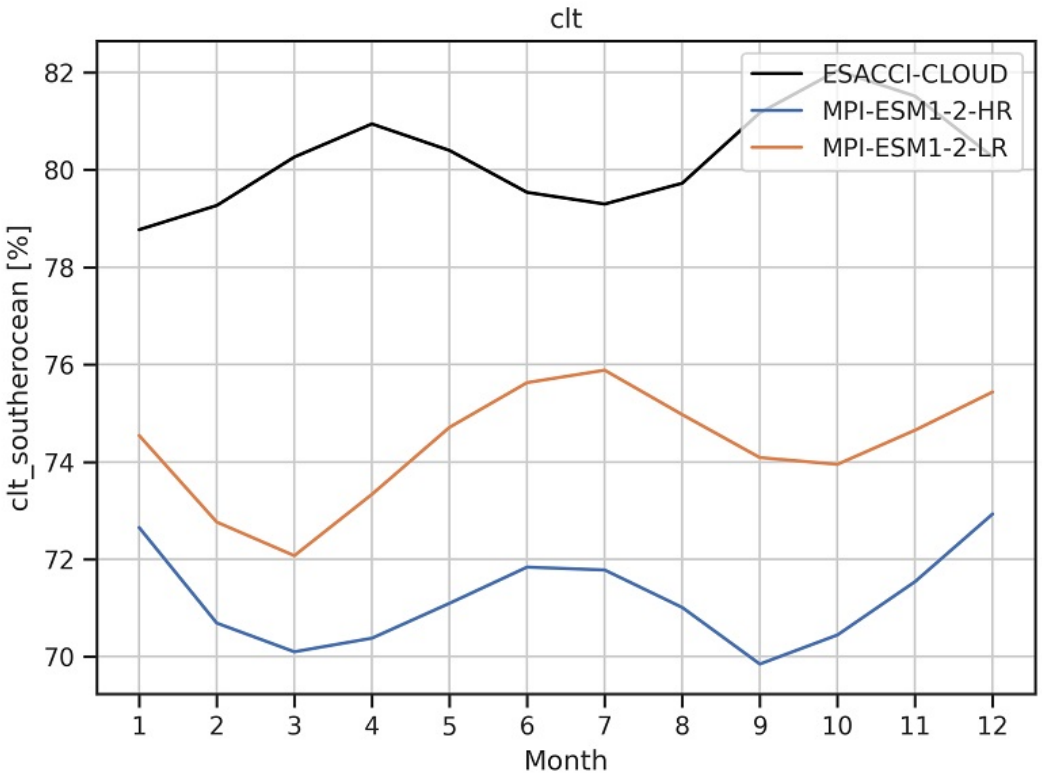


Fig. 11: Annual cycle of Southern Ocean total cloud cover.

```yaml
variables:
   # Define default. Variable definitions completely override the default
   # not just the values defined. If you want to override only the defined
   # values, use yaml anchors as shown
   default: &default
      colors: RdYlBu_r # Matplotlib colormap to use for the colorbar
      N: 20 # Number of map intervals to plot
      bad: [0.9, 0.9, 0.9] # Color to use when no data
   pr:
      <<: *default
      colors: gist_earth_r
      # Define bounds of the colorbar, as a list of
      bounds: 0-10.5,0.5 # Set colorbar bounds, as a list or in the format min-max,
↪interval
      extend: max # Set extend parameter of mpl colorbar. See https://matplotlib.org/
↪stable/api/_as_gen/matplotlib.pyplot.colorbar.html
   sos:
      # If default is defined, entries are treated as map specific option.
      # Missing values in map definitionas are taken from variable's default
      # definition
      default:
         <<: *default
         bounds: 25-41,1
         extend: both
```

(continues on next page)

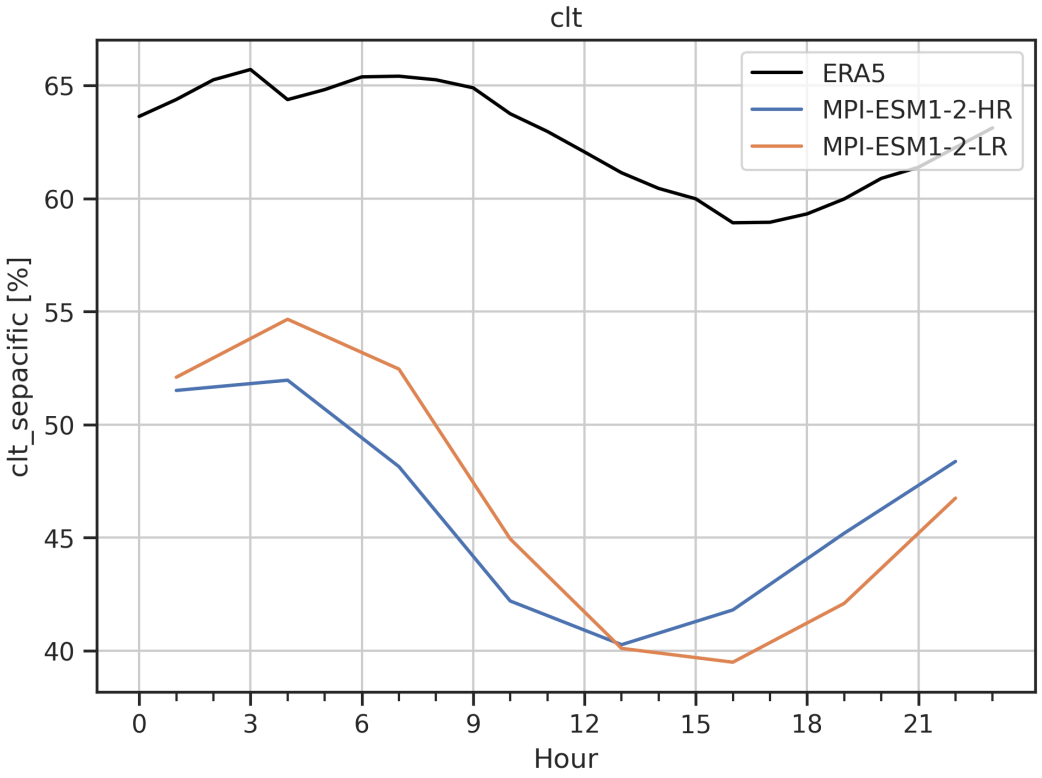


Fig. 12: Diurnal cycle of Southeast Pacific total cloud cover.

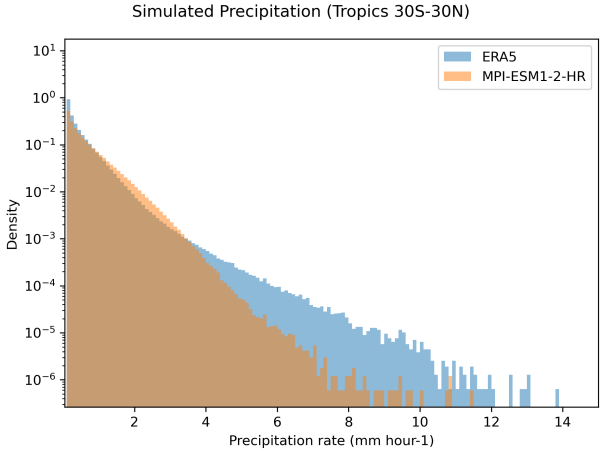


Fig. 13: Histogram of precipitation rates from MPI-ESM1-2-HR and ERA5 reanalysis 3-hourly data over the Tropics (30°S-30°N) for the years 1979-1980.

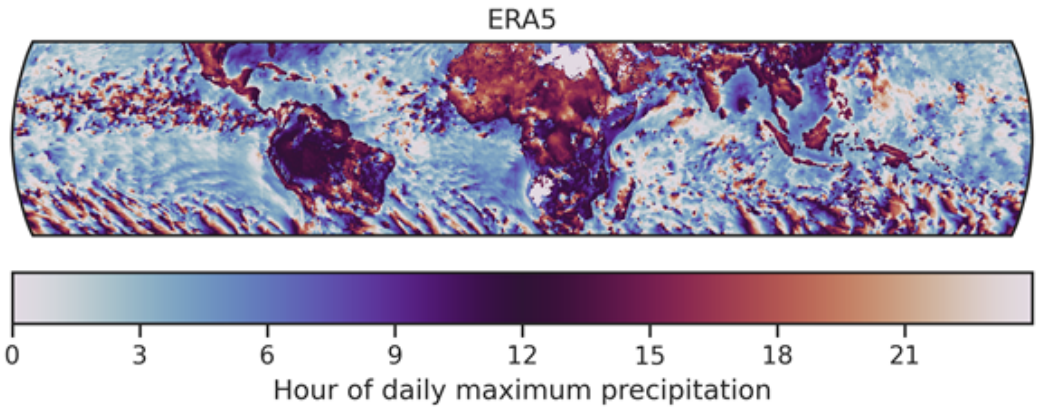


Fig. 14: Hour of maximum precipitation from ERA5 in the tropics (30°S-30°N) for the months June, July, August 2000.

(continued from previous page)

```
      arctic:
         bounds: 25-40,1
      antarctic:
         bounds: 30-40,0.5
   nao: &nao
      <<: *default
      extend: both
      # Variable definitions can override map parameters. Use with caution.
      bounds: [-0.03, -0.025, -0.02, -0.015, -0.01, -0.005, 0., 0.005, 0.01, 0.015, 0.02,
↪ 0.025, 0.03]
      projection: PlateCarree
      smooth: true
      lon: [-90, -60, -30, 0, 30]
      lat: [20, 40, 60, 80]
      colorbar_location: bottom
      suptitle_pos: 0.87
   sam:
      <<: *nao
      lat: [-90, -80, -70, -60, -50]
      projection: SouthPolarStereo
      projection_kwargs:
         central_longitude: 270
      smooth: true
      lon: [-120, -60, 0, 60, 120, 180]
```

Variables

Any, but the variables' number of dimensions should match the ones expected by each plot.

Example plots

5.1.4 Portrait plot

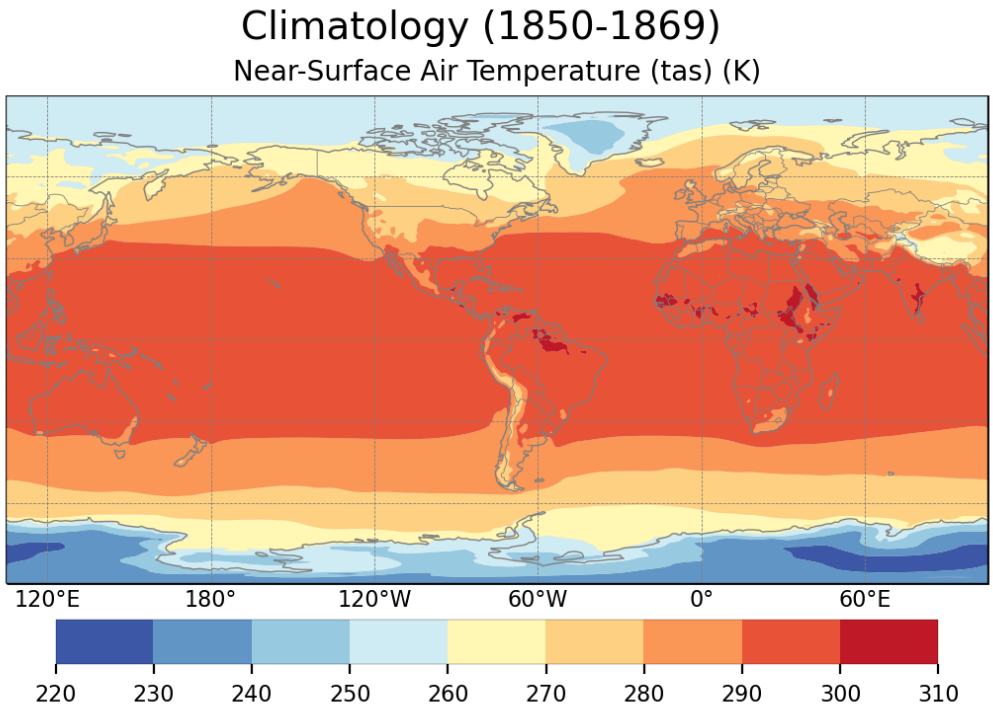


Fig. 15: Global climatology of tas.

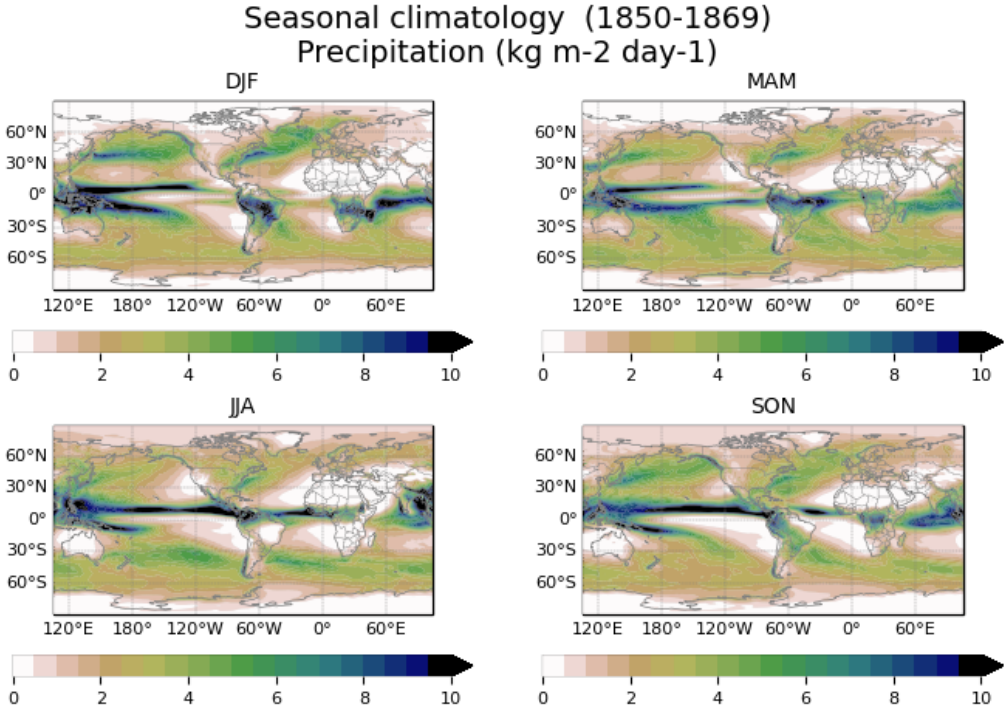


Fig. 16: Seasonal climatology of pr, with a custom colorbar.

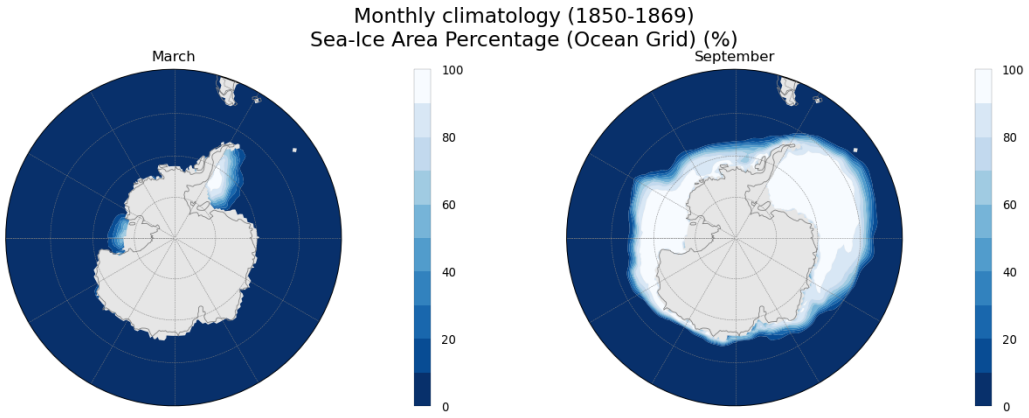


Fig. 17: Monthly climatology of sivol, only for March and September.

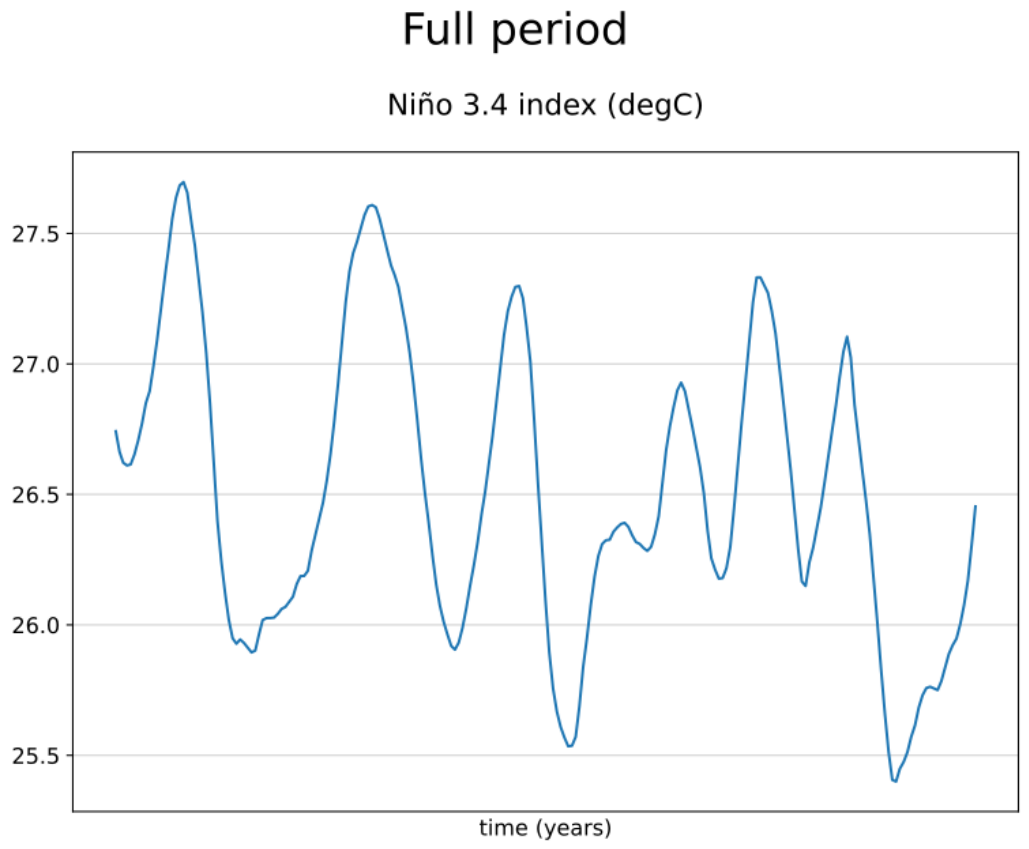


Fig. 18: Timeseries of Niño 3.4 index, computed directly with the preprocessor.

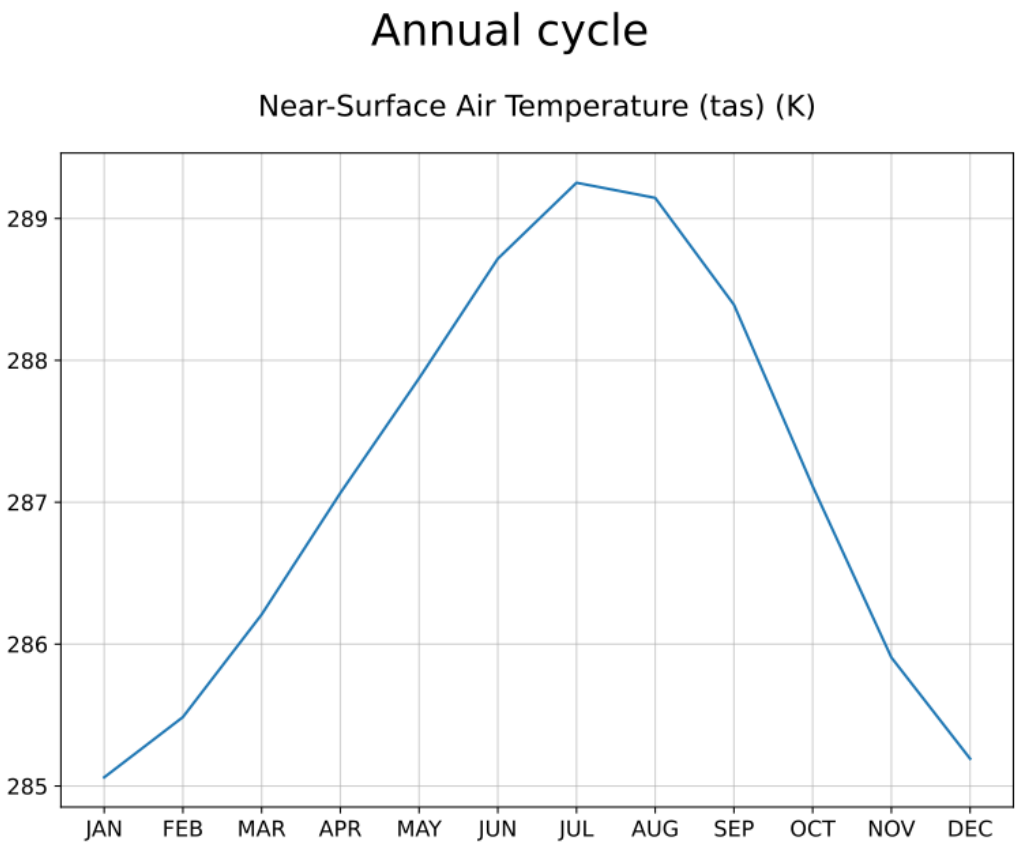


Fig. 19: Annual cycle of tas.

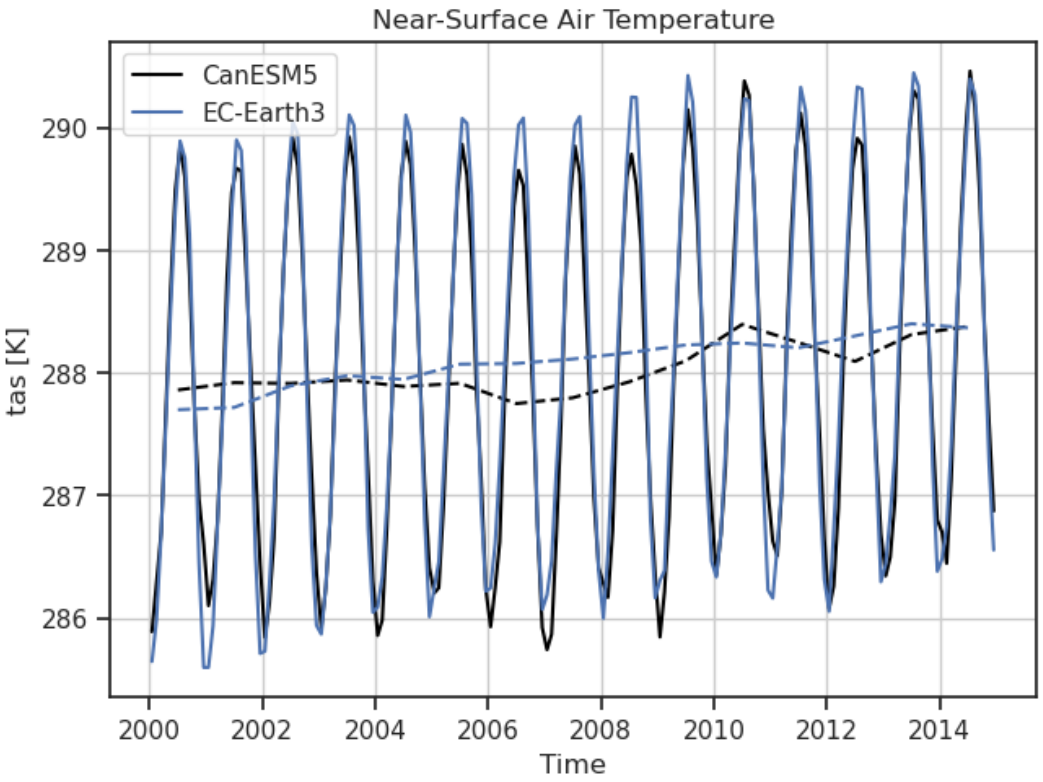


Fig. 20: Timeseries of tas including a reference dataset.

Overview

Portrait plots are a flexible way to visualize performance metrics for multiple datasets and up to four references. In this recipe `recipe_portrait_CMIP.yml` the normalized Root Mean Squared Deviation (RMSD) of global mean seasonal climatologies is calculated for a selection of CMIP models. In the example recipe, for each variable up to two observation based datasets are used as reference. See *Variables and Datasets* for complete list of references. The recipe uses preprocessor functions (distance metrics, global mean, climate statistics) to calculate a scalar metric for each combination of dataset, variable and reference, which is plotted by the `portrait_plot.py` diagnostic script.

User settings in recipe

By default cells are plotted for combinations of `short_name`, `dataset`, `project` and `split`, where `split` is an optional extra_facet for variables. However, this can be customized using the `x_by`, `y_by`, `group_by` and `split_by` script settings. For a complete and detailed list of settings, see the *diagnostic documentation*. While this allows very flexible use for any kind of data, there are some limitations as well: The grouping (subplots) and normalization is always applied along the x-axis. With default settings this means normalizing all metrics for each variable and grouping all datasets by project.

To plot distance metrics like RMSE, pearson R, bias etc. the `distance_metric` preprocessor or custom diagnostics can be used.

Variables and Datasets

Note

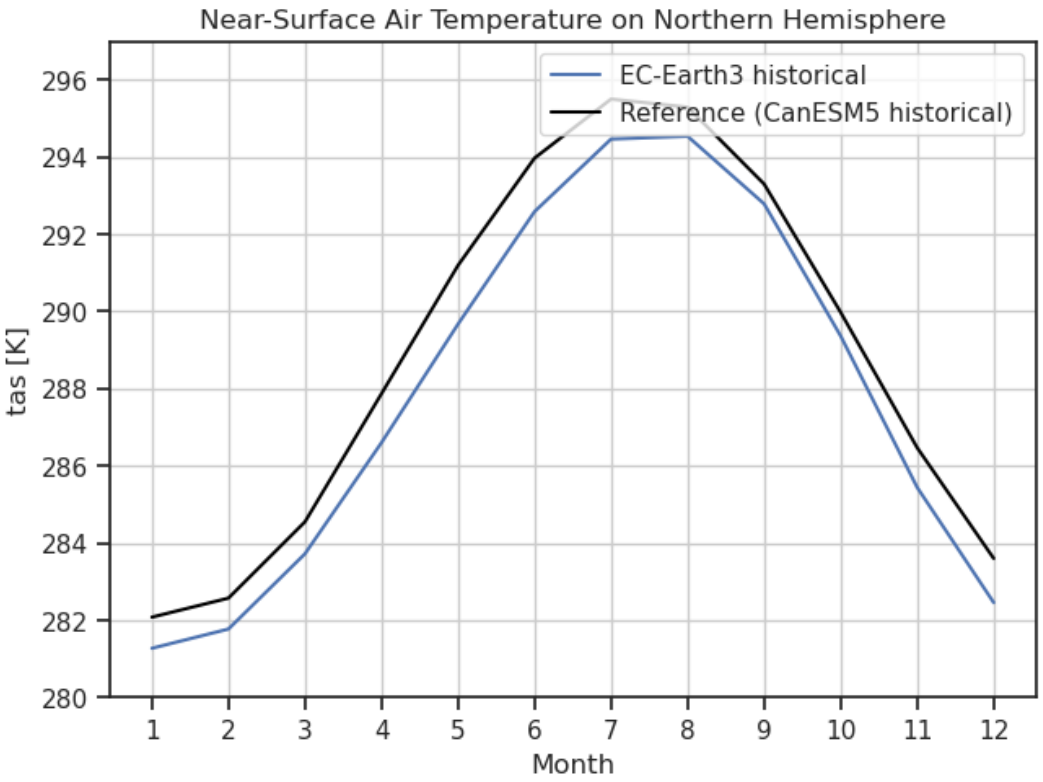


Fig. 21: Annual cycle of tas including a reference dataset.

The recipe generally works for any variable that is preprocessed correctly. To use different preprocessors or reference datasets it could be useful to create different variable groups and link them with the same extra_facet like `variable_name`. See recipe for examples. Listed below are the variables used to produce the example figure.

The following list shows which observational dataset is used as reference for each variable in this recipe. All variables are atmospheric monthly means. For 3D variables the selected pressure level is specified.

- clt (Ref1: ESACCI-CLOUD, Ref2: PATMOS-x)
- pr (Ref1: GPCP-V2.2)
- rlut, rsut (Ref1: CERES-EBAF)
- tas (Ref1: ERA-Interim, Ref2: NCEP-NCAR-R1)
- ua (200 hPa, Ref1: ERA-Interim, Ref2: NCEP-NCAR-R1)
- zg (500 hPa, Ref1: ERA-Interim, Ref2: NCEP-NCAR-R1)

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 ozone and selected climate parameters in a set of EMAC simulations, Geosci. Model Dev., 8, 733, doi: 10.5194/gmd-8-733-2015 (2015).

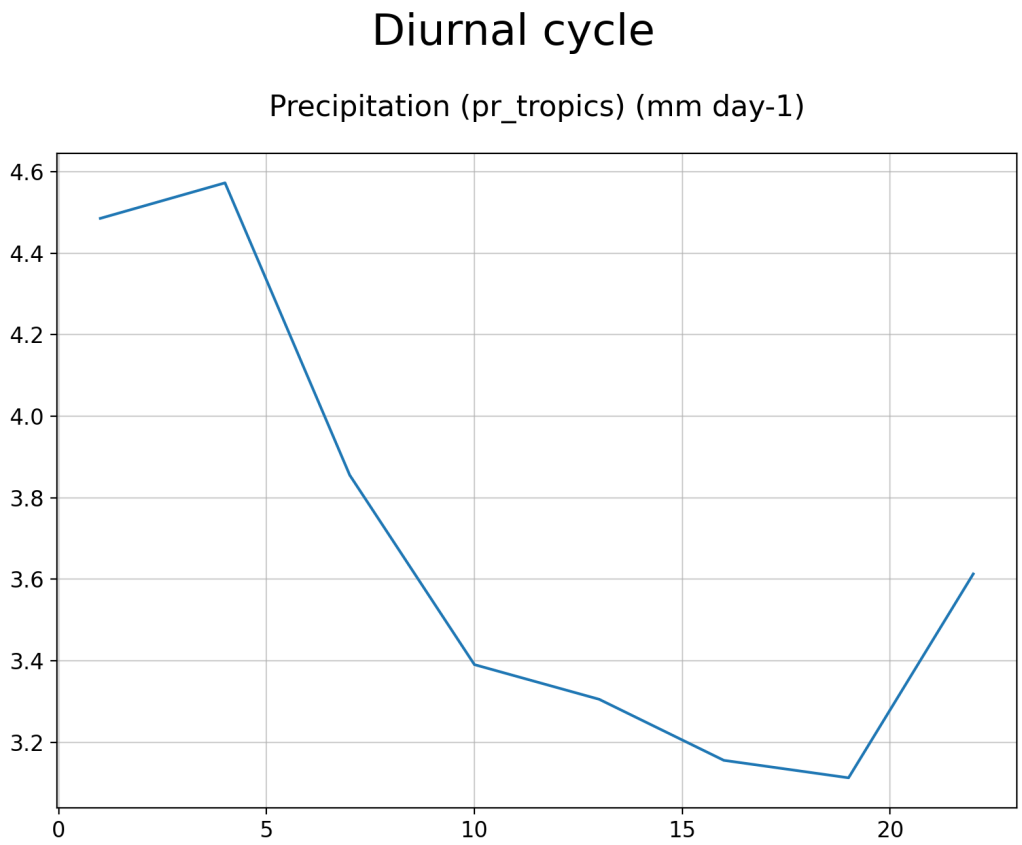


Fig. 22: Diurnal cycle of precipitation in the Tropics from EC-Earth3.

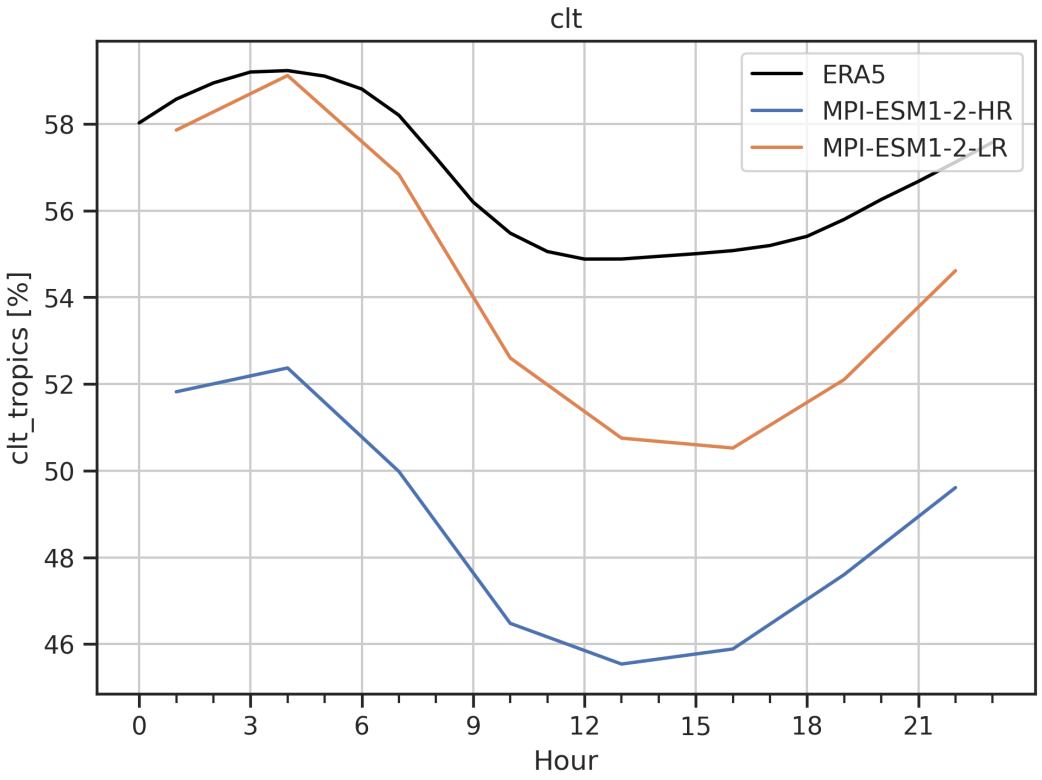


Fig. 23: Diurnal cycle of clt including a reference dataset.

Example plots

5.1.5 Sanity checks

Overview

The sanity check recipe (`recipe_sanity_checks.yml`) is intended to perform some basic sanity checks for new simulations during model development. A subset of variables is selected to check whether the model is performing in reasonable way. The recipe plots global monthly means as well as maximum and minimum across all individual grid cells for each time step. This can be used to check for example for negative mass concentrations occuring at any grid cell at any time (minimum is less than zero) or whether individual grid cells exceed physically reasonable values (e.g. total cloud fraction greater than 100%). The global monthly means can be compared to the minimum and maximum values found in reference datasets (observations, reanalyses) across all months and all reference datasets. The "reasonable" limits are shown as red lines. A good model simulations would be expected to not exceed these limits at any time. The minimum and maximum global averages across all months and all reference datasets is calculated with `recipe_create_ranges_obs.yml`.

Available recipes and diagnostics

Recipes are stored in *recipes/sanity_checks*

- recipe_sanity_checks.yml
- recipe_create_ranges_obs.yml

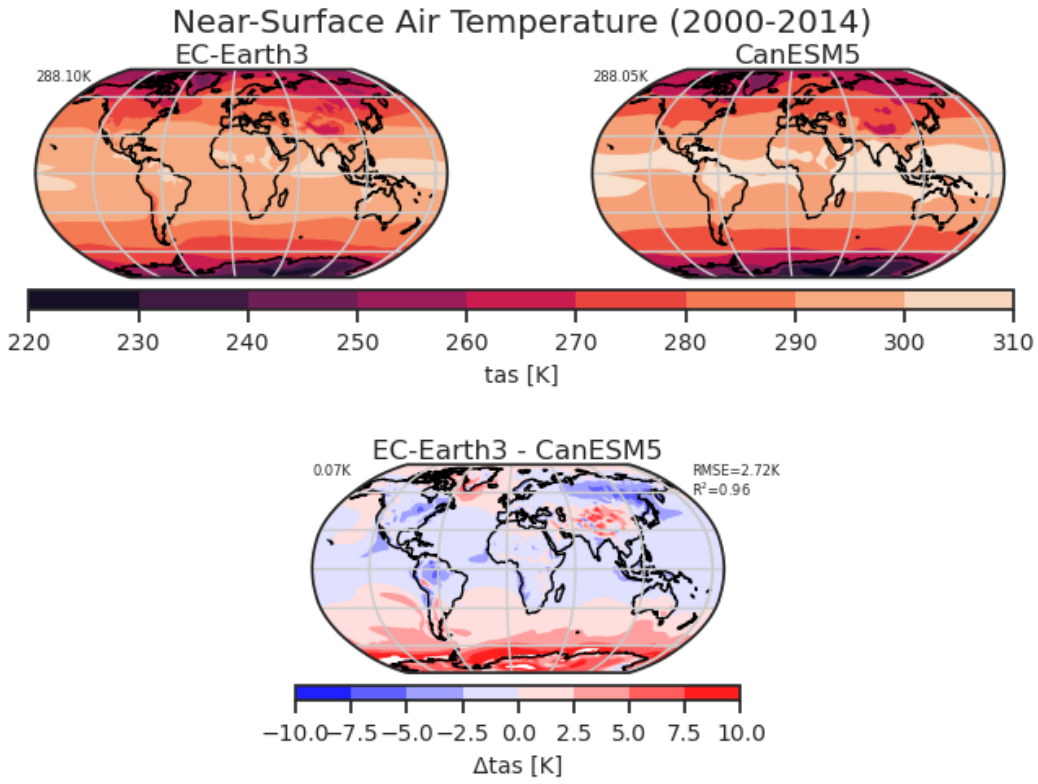


Fig. 24: Global climatology of tas including a reference dataset.

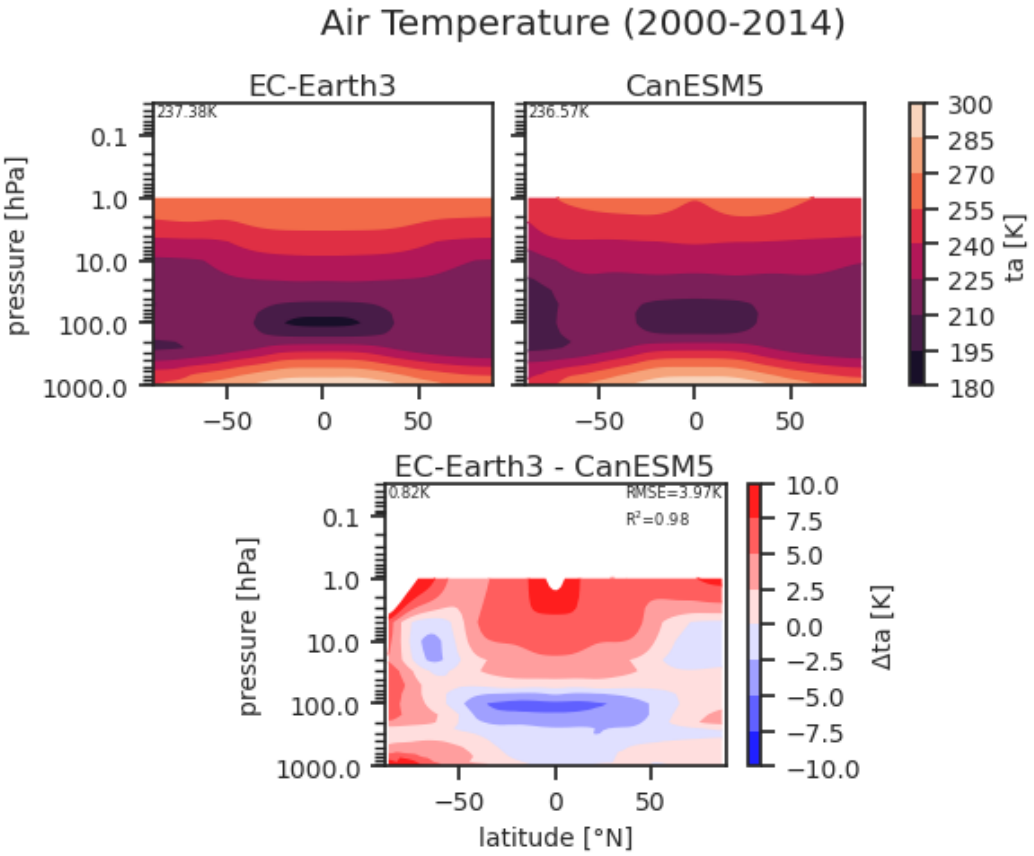


Fig. 25: Zonal mean profile of ta including a reference dataset.

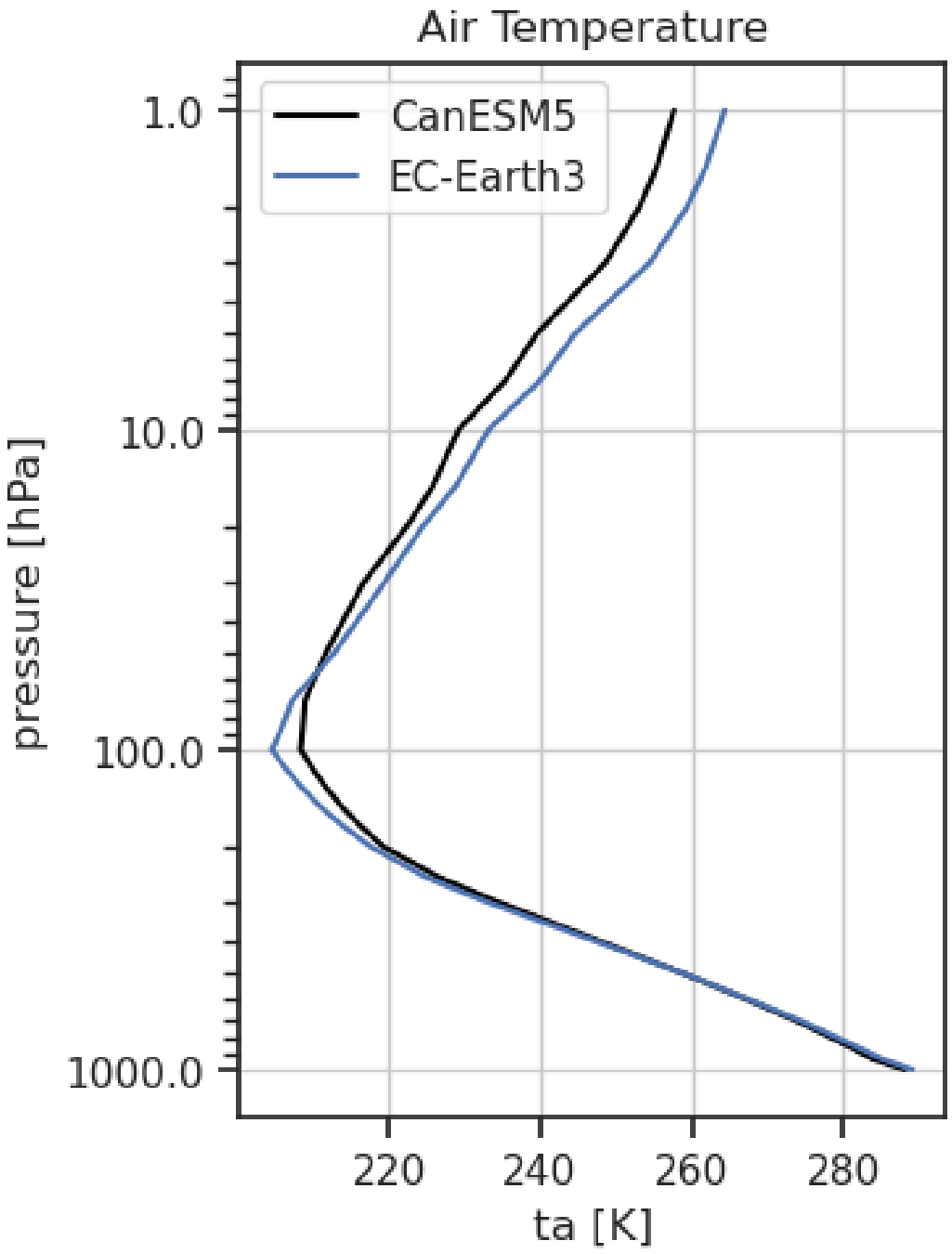


Fig. 26: 1D profile of ta including a reference dataset.

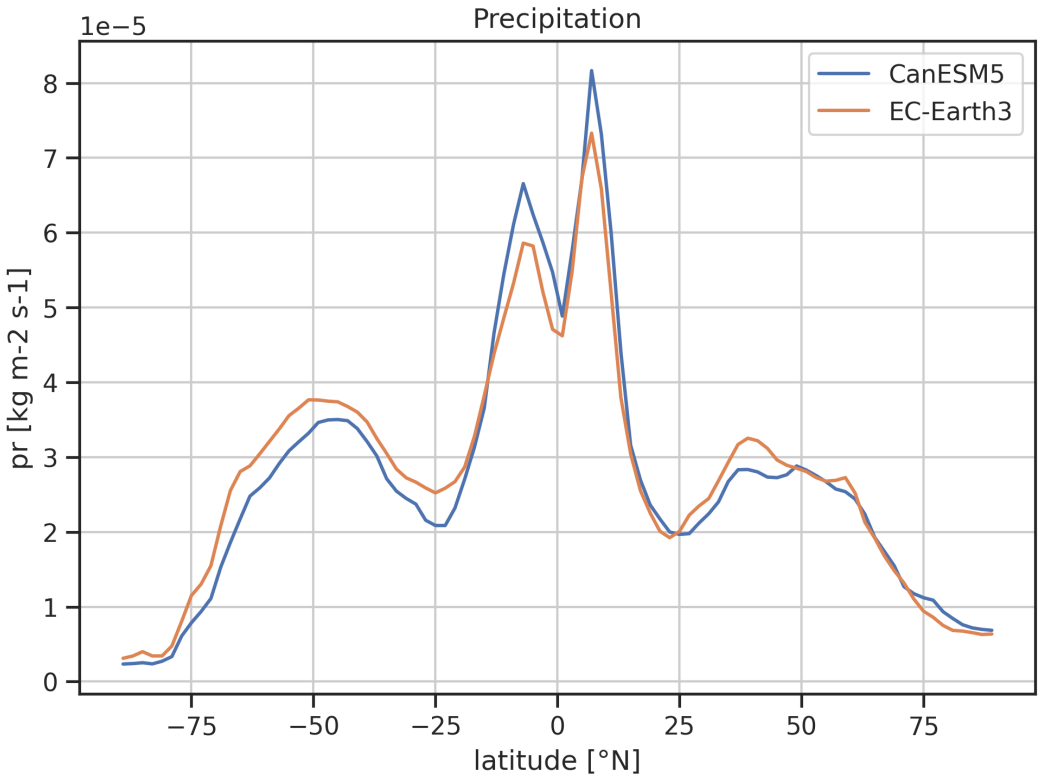


Fig. 27: Zonal mean pr including a reference dataset.

Note

The time frequency used in `recipe_create_ranges_obs.yml` (monthly, daily, etc. values) to calculate "reasonable" minimum and maximum values must match the time frequency used in the sanity check recipe (`recipe_sanity_checks.yml`).

Diagnostics are stored in *diag_scripts/monitor/*

- *multi_datasets.py*: Monitoring diagnostic to show multiple datasets in one plot (incl. biases).

Variables:

- ps (air mass)
- qep (moisture flux)
- prw (water vapor mass)
- asr
- clivi
- clt
- hfls
- hfss
- lwcre

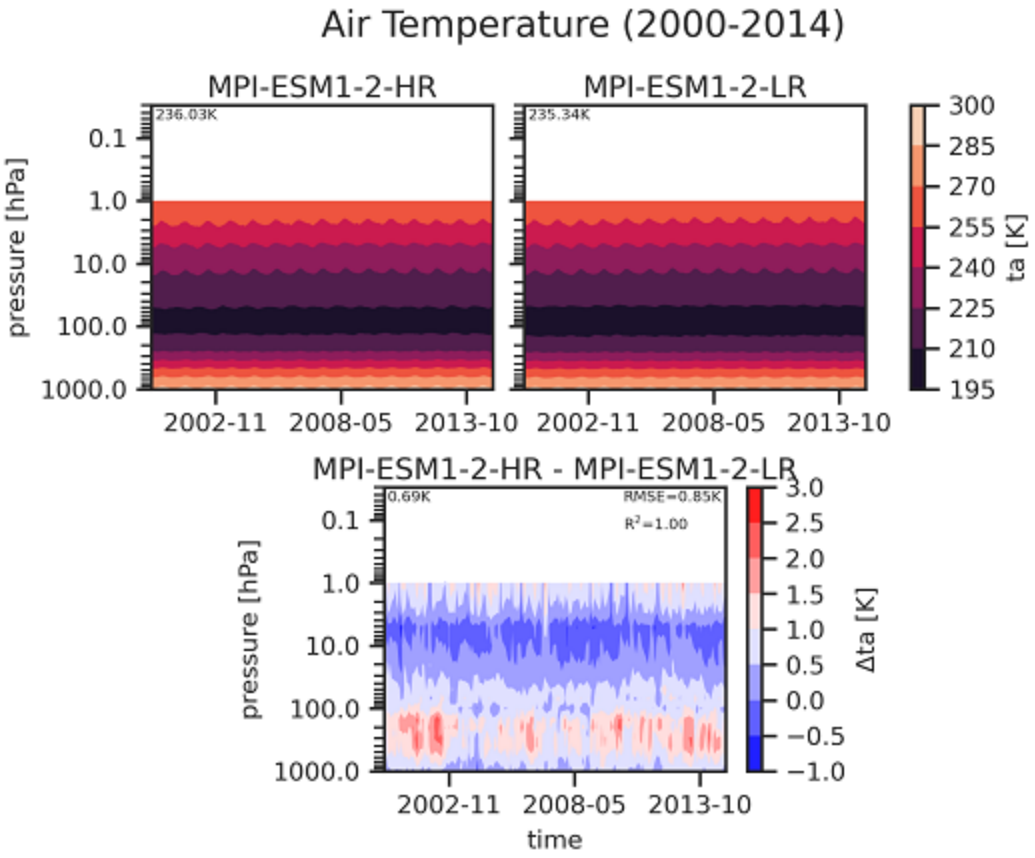


Fig. 28: Hovmoeller plot (pressure vs. time) of ta including a reference dataset.

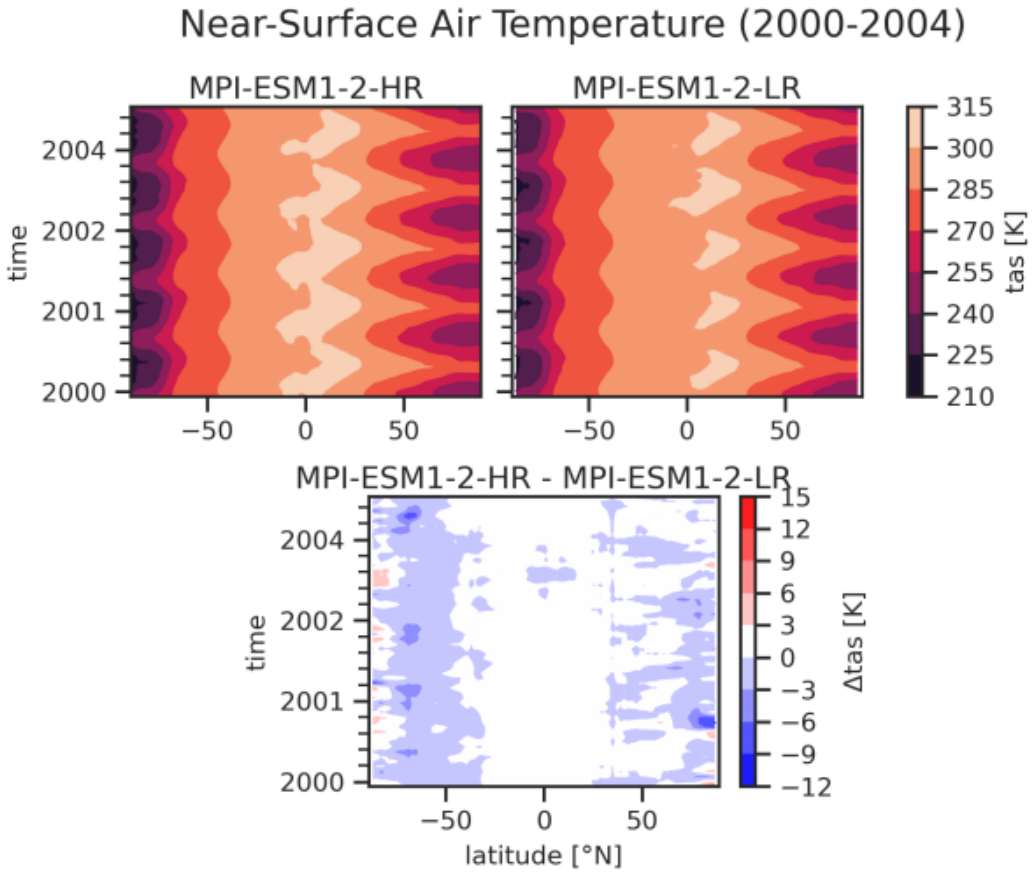


Fig. 29: Hovmoeller plot (time vs. latitude) of tas including a reference dataset

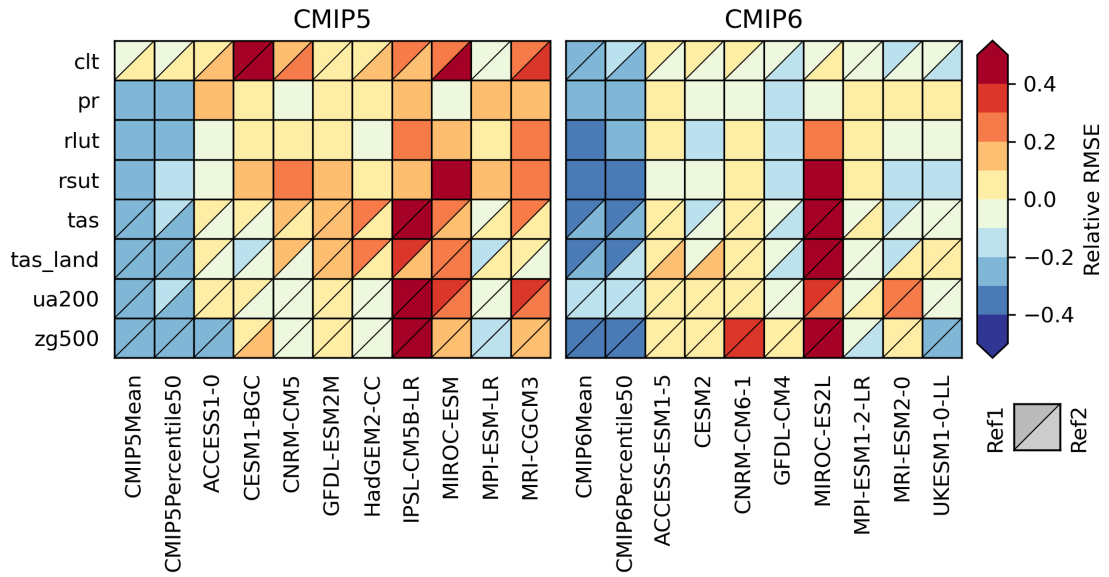


Fig. 30: Relative space-time root-mean-square deviation (RMSD) calculated from the climatological seasonal cycle of CMIP5 and CMIP6 simulations. A relative performance is displayed, with blue shading indicating better and red shading indicating worse performance than the median of all model results. A diagonal split of a grid square shows the relative error with respect to the reference data set.

- lwp
- netcre
- pr
- prc
- prw
- rlds
- rlut
- rtnt
- rsds
- rsut
- swcre
- tas
- tauu
- tauv

Example plots

5.1.6 Seaborn Diagnostics

Overview

These recipes showcase the use of the Seaborn diagnostic that provides a high-level interface to Seaborn for ESMValTool recipes.

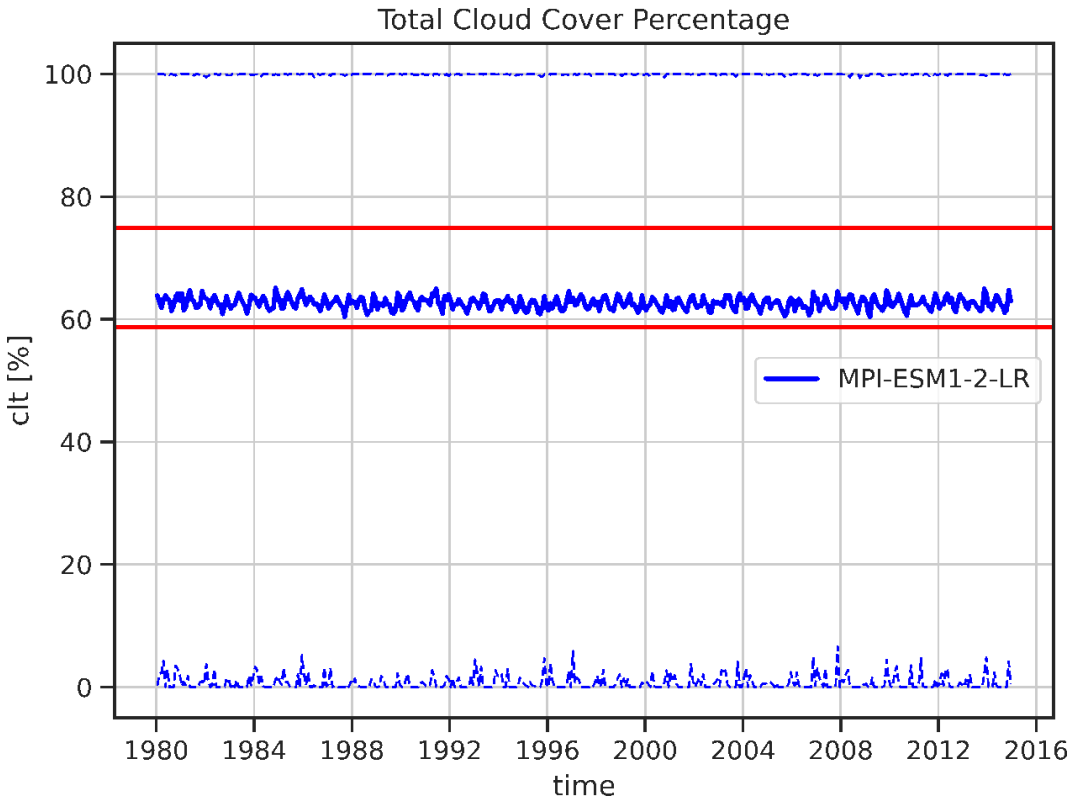


Fig. 31: Time series of monthly global average (solid line) and minimum / maximum (dashed lines) total cloud cover from MPI-ESM1-2-LR. The red horizontal lines show the monthly minimum and maximum global average across multiple reference dataset (here: ESACCI-CLOUD, CLARA-AVHRR, PATMOS-x, MODIS, ERA5) calculated with recipe_create_ranges_obs.yml. Plot created with recipe_sanity_checks.yml.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_seaborn.yml

Diagnostics are stored in diag_scripts/

- *seaborn_diag.py*

Variables

Arbitrary variables are supported.

Observations and reformat scripts

Arbitrary datasets are supported.

References

- Waskom, M. L. (2021), seaborn: statistical data visualization, Journal of Open Source Software, 6(60), 3021, doi:10.21105/joss.03021.

Example plots

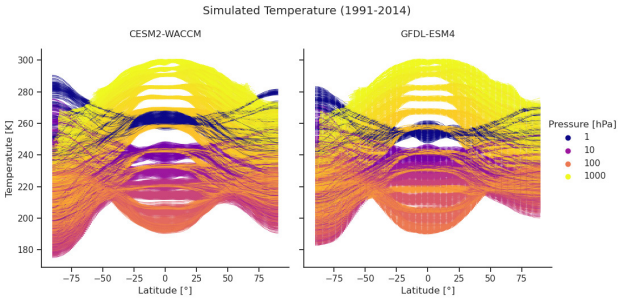


Fig. 32: Monthly and zonal mean temperatures vs. latitude in the period 1991-2014 for two Earth system models (CESM2-WACCM and GFDL-ESM4). Colors visualize the corresponding pressure levels.

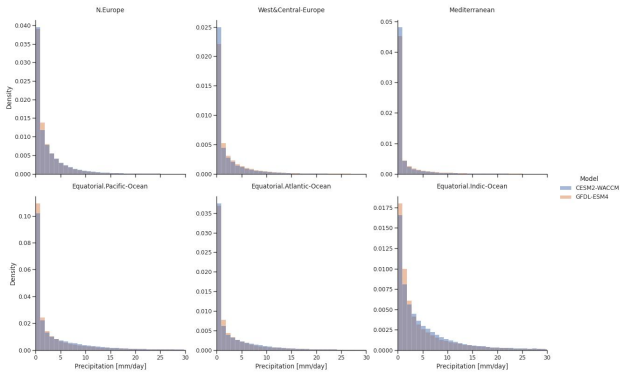


Fig. 33: Spatiotemporal distribution of daily precipitation in the period 2005-2014 for six IPCC AR6 regions simulated by two Earth system models (CESM2-WACCM and GFDL-ESM4). Each day in each grid cell in the corresponding regions is considered with equal weight.

5.1.7 CMIP Rapid Evaluation Framework (REF)

Overview

Here ESMValTool recipes are collected which will be used in the CMIP Rapid Evaluation Framework (REF).

Available recipes

Recipes are stored in *recipes/*

- *recipe_ecs.yml*: Calculate equilibrium climate sensitivity (ECS)
- *recipe_tcr.yml*: Calculate transient climate response (TCR)
- *recipe_tcre.yml*: Calculate transient climate response to cumulative CO2 emissions (TCRE)
- ref/recipe_ref_cre.yml: Maps and zonal means of longwave and shortwave cloud radiative effect
- ref/recipe_ref_timeseries_region.yml: Time series in comparison with reference data for a selected IPCC region defined through a shape file, based on *recipe_ref_timeseries.yml*
- ref/recipe_ref_annual_cycle_region.yml: Annual cycle in comparison with reference data for a selected IPCC region defined through a shape file, based on *recipe_ref_annual_cycle.yml*
- ref/recipe_ref_trend_regions.yml: Linear Trends for all IPCC land regions compared with reference data, based on *recipe_ref_trend_regions.yml*
- ref/recipe_ref_scatterplot.yml: 2D histograms with focus on clouds
- ref/recipe_ref_sea_ice_area_basic.yml: Seasonal cycle of Arctic (NH) and Antarctic (SH) sea ice area, time series of Arctic September (NH) and Antarctic February (SH) sea ice area
- ref/recipe_ref_ozone.yml: NH/SH polar cap (60 degrees to 90 degrees) March/September total column ozone time series, zonal mean total column ozone vs. time map plot, zonal mean total column ozone vs. annual cycle map plot, altitude vs. zonal mean ozone profile climatology map plot
- ***ref/recipe_ref_fire.yml*:**
  Maps of burnt area fraction, fire weather control, and fuel load continuity control.

Example plots:

5.2 Atmosphere

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

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:

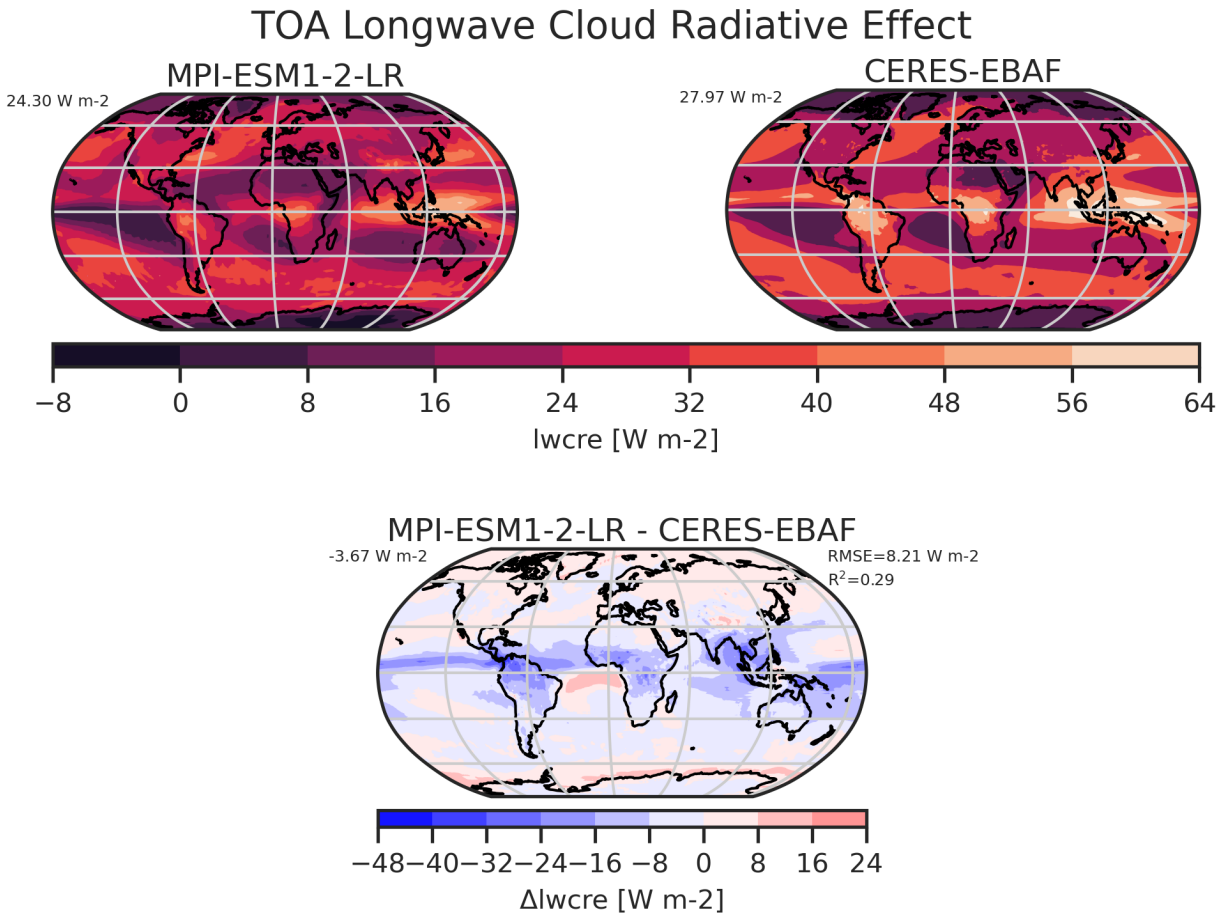


Fig. 34: Geographical map of the climatological mean longwave cloud radiative effect from MPI-ESM1-2-LR and CERES-EBAF Ed4.2 and their difference.

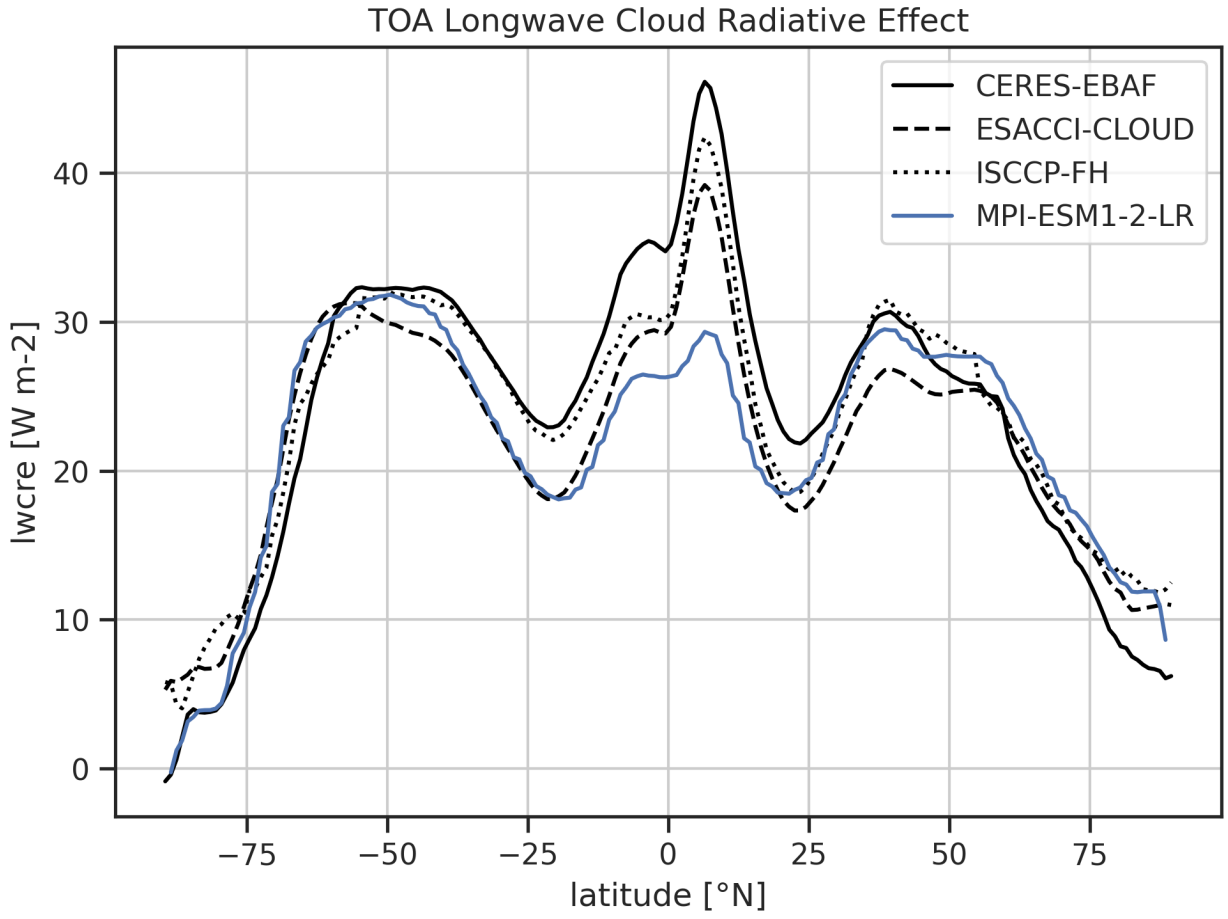


Fig. 35: Zonal averages of the climatological mean longwave cloud radiative effect from CERES-EBAF Ed4.2 (solid black), ESACCI-CLOUD (dashed black), ISCCP-FH (dotted black) and the MPI-ESM1-2-LR model (blue).

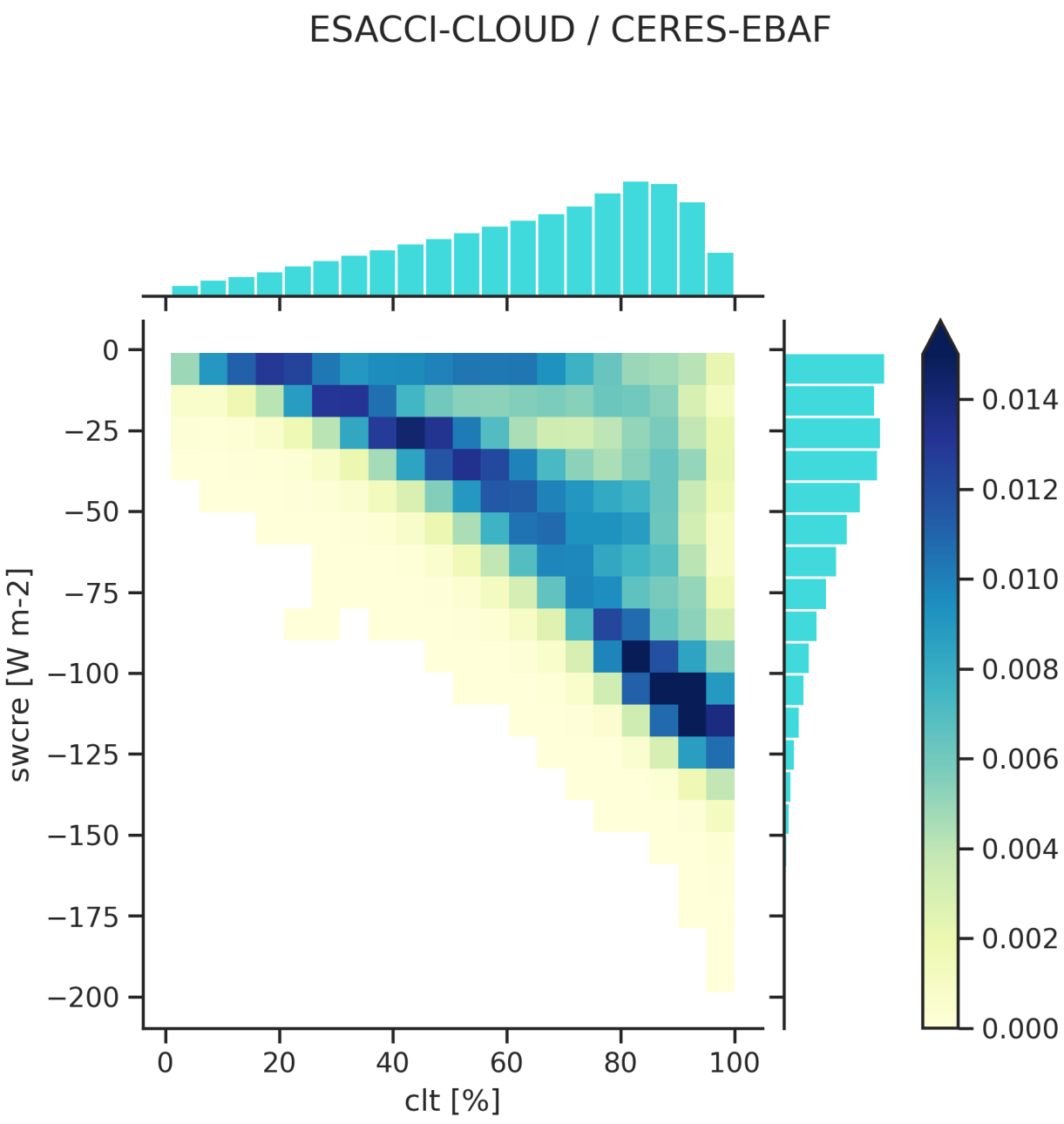


Fig. 36: 2D histogram of total cloud fraction (ESACCI-CLOUD) and shortwave cloud radiative effect (CERES-EBAF) for the years 2001-2016 with 1D histograms attached.

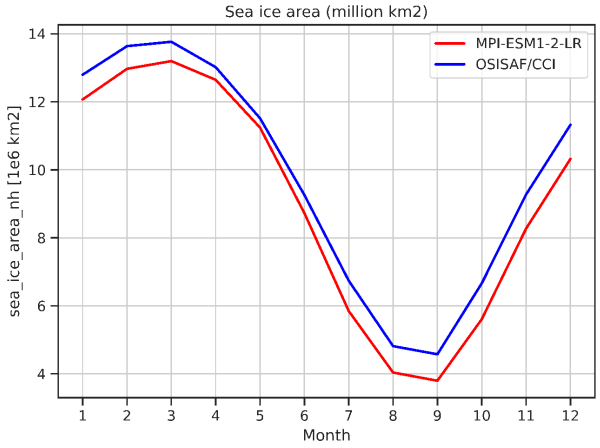


Fig. 37: Average seasonal cycle of the Arctic (NH) sea ice area from MPI-ESM1-2-LR (red line) compared with OSISAF/CCI (blue line). Created with recipe_ref_sea_ice_area_basic.yml.

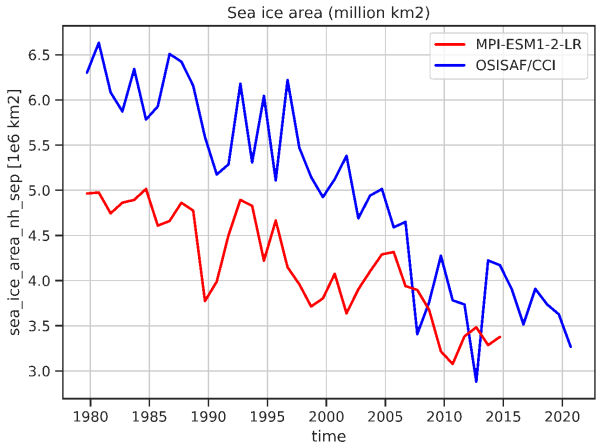


Fig. 38: Time series of Arctic (NH) September (NH) sea ice area from MPI-ESM1-2-LR (red line) compared with OSISAF/CCI (blue line). Created with recipe_ref_sea_ice_area_basic.yml.

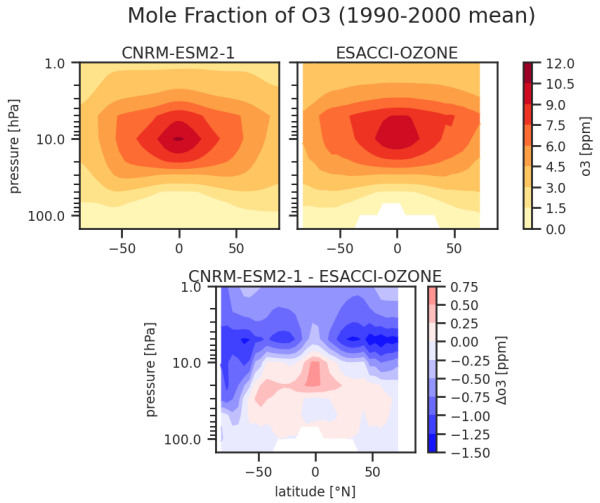


Fig. 39: Zonal mean vertically resolved ozone climatology from CNRM-ESM2-1 compared with ESACCI-OZONE for the years 1990 to 2000. Created with recipe_ref_ozone.yml.

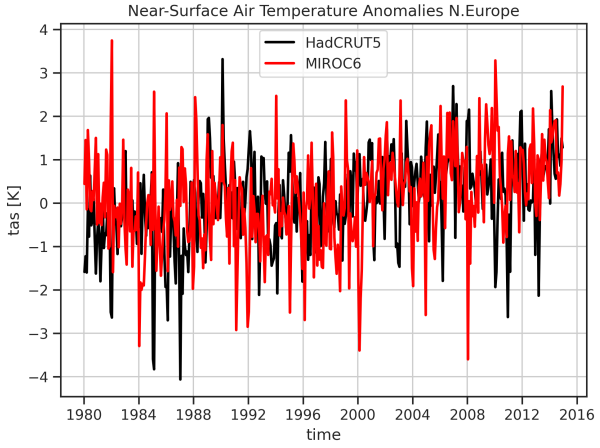


Fig. 40: Time series of near-surface air temperature anomalies from MIROC6 compared with HadCRUT5 for N.Europe for the years 1980 to 2014 (reference period 1980 to 2009). Created with recipe_ref_timeseries_region.yml.

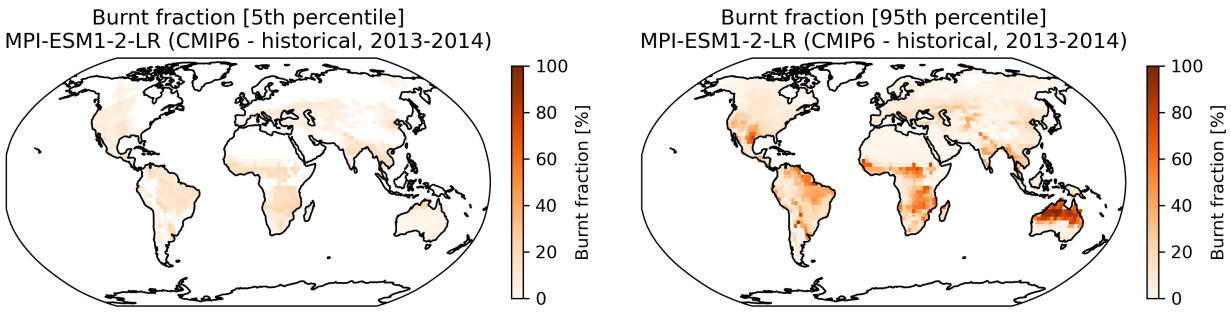


Fig. 41: Burnt area fraction for the MPI-ESM1-2-LR model (CMIP-historical experiment) for the time period 2013-2014 as computed with the ConFire model *Jones et al. (2024)*.

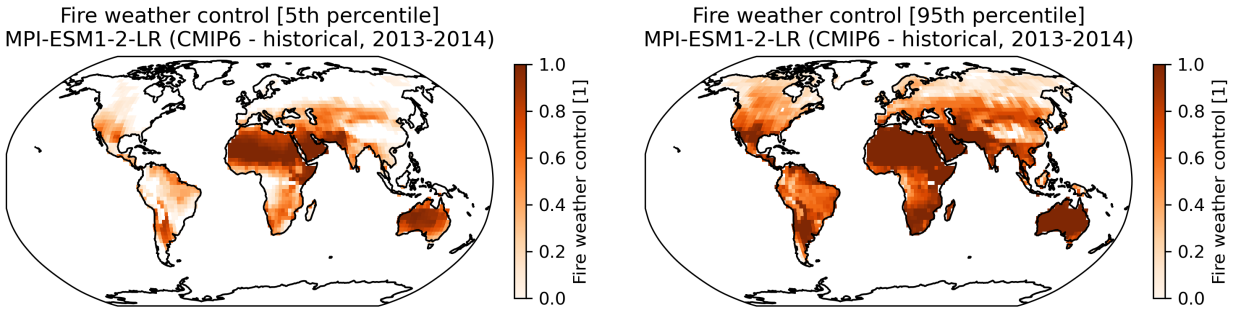


Fig. 42: Fire weather control for the MPI-ESM1-2-LR model (CMIP-historical experiment) for the time period 2013-2014 as computed with the ConFire model *Jones et al. (2024)*.

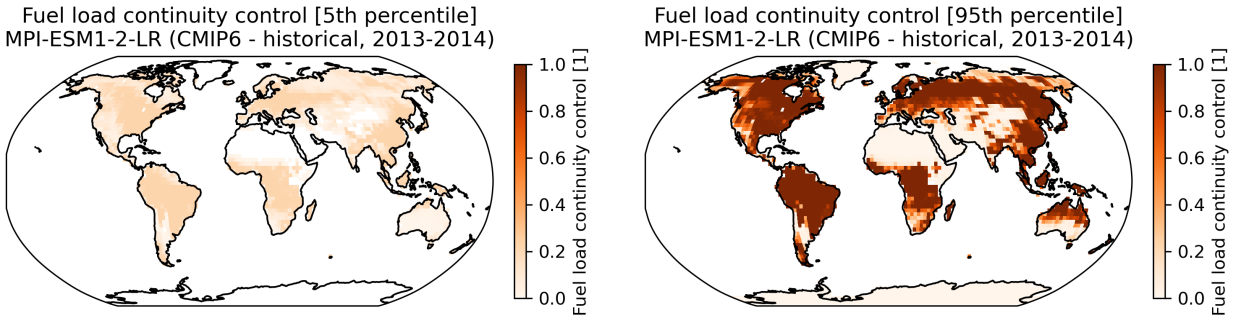


Fig. 43: Fuel load continuity control for the MPI-ESM1-2-LR model (CMIP-historical experiment) for the time period 2013-2014 as computed with the ConFire model *Jones et al. (2024)*.

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.

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.

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)

Variables

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

Observations and reformat scripts

- ERA-INTERIM

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/10.3402/tellusa.v2i3.8546
- 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

Example plots

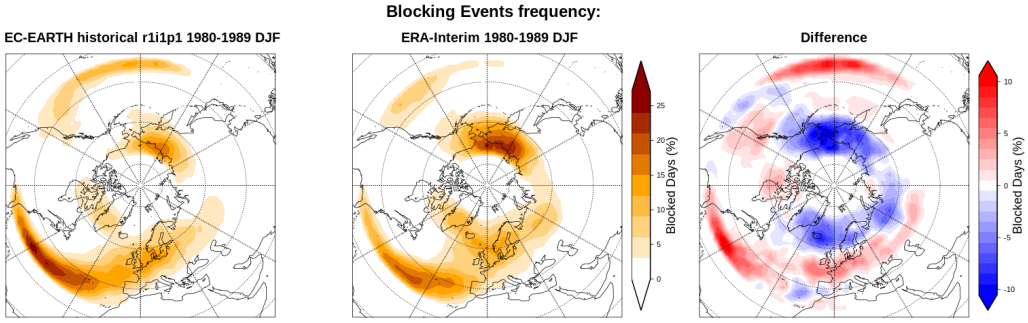


Fig. 44: Blocking Events frequency for a CMIP5 EC-Earth historical run (DJF 1980-1989), compared to ERA-Interim. Units are percentage of blocked days per season.

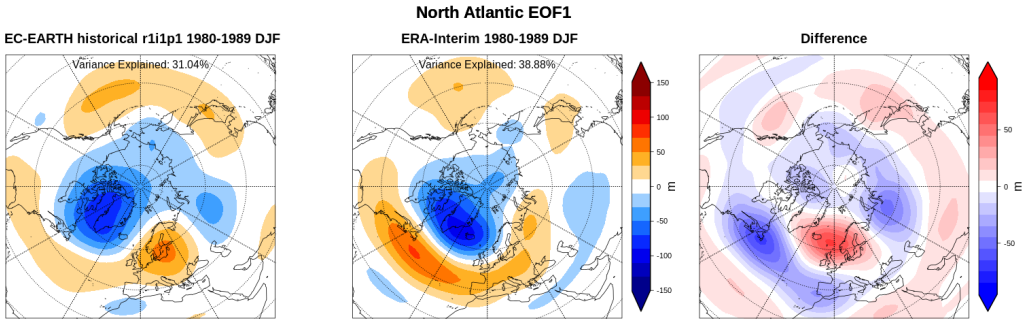


Fig. 45: North Atlantic Oscillation for a CMIP5 EC-Earth historical run (DJF 1980-1989) compared to ERA-Interim, shown as the linear regression of the monthly Z500 against the first Principal Component (PC1) of the North Atlantic region.

5.2.2 Generating Climate Patterns from CMIP6 Models

Overview

The recipe recipe_climate_patterns generates climate patterns from CMIP6 model datasets.

> **Note**
>
> The regrid setting in the recipe is set to a 2.5x3.75 grid. This is done to match the current resolution in the IMOGEN-JULES model, but can be adjusted with no issues for a finer/coarser patterns grid.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_climate_patterns.yml

Diagnostics are stored in esmvaltool/diag_scripts/climate_patterns/

- climate_patterns.py: generates climate patterns from input datasets
- sub_functions.py: set of sub functions to assist with driving scripts
- plotting.py: contains all plotting functions for driving scripts

User settings in recipe

1. Script climate_patterns.py

   Required settings for script

   None

   Optional settings for script

   - jules_mode: output jules-specific var names + .nc files
   - parallelise: parallelise over models or not
   - area: calculate the patterns globally, or over land only

   Required settings for variables

   - short_name
   - additional_datasets

   Optional settings for variables

   None

   Required settings for preprocessor

   - monthly_statistics: converts data to mean monthly data

   Optional settings for preprocessor

   - regrid: regrids data

Variables

1. Script climate_patterns.py

- tasmax (atmos, monthly, longitude latitude time)
- tasmin (atmos, monthly, longitude latitude time)
- tas (atmos, monthly, longitude latitude time)
- huss (atmos, monthly, longitude latitude time)
- pr (atmos, monthly, longitude latitude time)
- sfcWind (atmos, monthly, longitude latitude time)
- ps (atmos, monthly, longitude latitude time)
- rsds (atmos, monthly, longitude latitude time)
- rlds (atmos, monthly, longitude latitude time)

Observations and reformat scripts

None

References

- Huntingford, C., Cox, P. An analogue model to derive additional climate change scenarios from existing GCM simulations. Climate Dynamics 16, 575–586 (2000). https://doi.org/10.1007/s003820000067
- Mathison, C. T. et al. A rapid application emissions-to-impacts tool for scenario assessment: Probabilistic Regional Impacts from Model patterns and Emissions (PRIME). EGUsphere [preprint], (2024). https://doi.org/10.5194/egusphere-2023-2932

Example plots

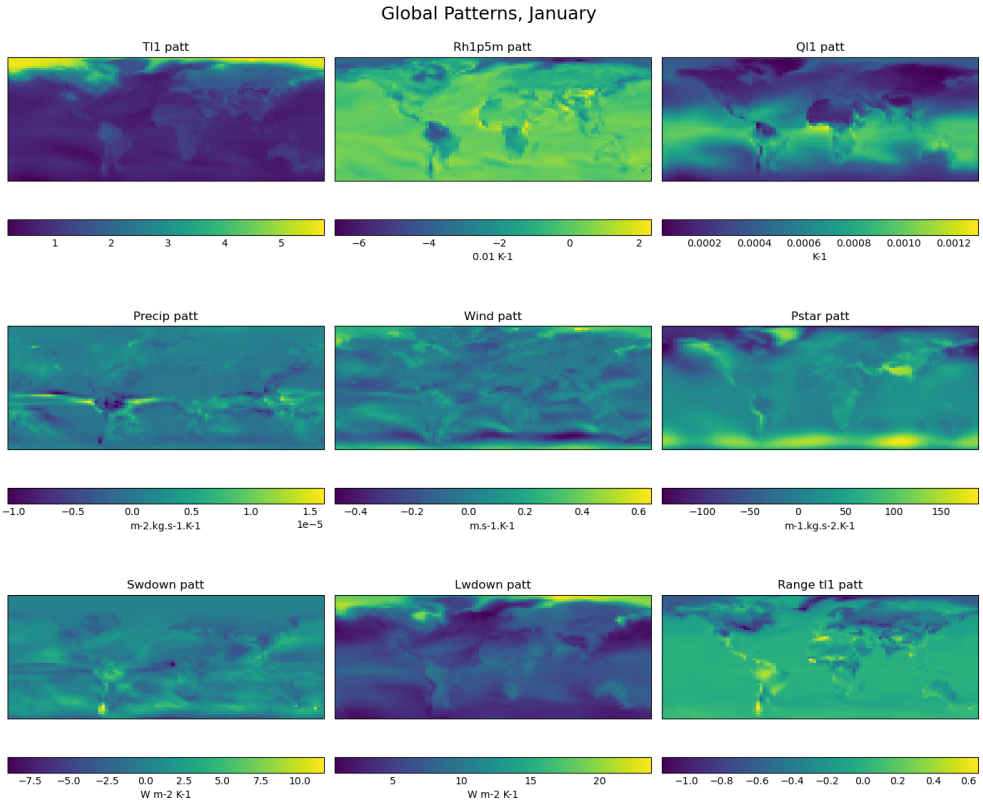


Fig. 46: Patterns generated for CMIP6 models, gridded view. Patterns are shown per variable, for the month of January.

5.2.3 Clouds

Overview

Four recipes are available to evaluate cloud climatologies from CMIP models.

1) recipe_clouds_bias.yml computes climatologies and creates map plots of multi-model mean, mean bias, absolute bias and relative bias of a given variable. Similar to IPCC AR5 (ch. 9) fig. 9.2 a/b/c (Flato et al., 2013).

2) recipe_clouds_ipcc.yml computes multi-model mean bias and zonal means of the clouds radiative effect (shortwave, longwave and net). Similar to IPCC AR5 (ch. 9) fig. 9.5 (Flato et al., 2013).

3) 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. Note that the satellite observations used in the original recipe (UWisc) is not maintained anymore and has been superseeded by MAC-LWP (Elsaesser et al., 2017). We recommend using MAC-LWP.

4) Recipe family recipe_lauer22jclim_*.yml is an extension of recipe_lauer13jclim.yml for evaluation of cloud radiative forcing (CRE), liquid water path (lwp), ice water path (clivi), total cloud amount (clt), cloud liquid water content (clw), cloud ice water content (cli), cloud fraction (cl) and water vapor path (prw) from CMIP6 models in comparison to CMIP5 results and satellite observations. Wherever possible, the diagnostics use multi-observational products as reference datasets. The recipe family reproduces all figures from Lauer et al. (2023): maps of the geographical distribution of multi-year averages, Taylor diagrams for multi-year annual averages, temporal variability, seasonal cycle amplitude, cloud ice fraction as a function of temperature, zonal means of 3-dim cloud liquid/ice content and cloud fraction, matrices of cloud cover and total cloud water path as a function of SST and 500 hPa vertical velocity, shortwave CRE and total cloud water path binned by total cloud cover and pdfs of total cloud cover for selected regions.

Available recipes and diagnostics

Recipes are stored in recipes/clouds

- recipe_clouds_bias.yml
- recipe_clouds_ipcc.yml
- recipe_lauer13jclim.yml
- recipe_lauer22jclim_*.yml (* = fig1_clim_amip, fig1_clim, fig2_taylor_amip, fig2_taylor, fig3-4_zonal, fig5_lifrac, fig6_interannual, fig7_seas, fig8_dyn, fig9-11ab_scatter, fig9-11c_pdf)

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_dyn_matrix.ncl: cloud properties by dynamical regime (SST, omega500)
- clouds_interannual.ncl: global maps of the interannual variability
- clouds_ipcc.ncl: global maps of multi-model mean minus observations + zonal averages of individual models, multi-model mean and observations
- clouds_lifrac_scatter.ncl: cloud liquid water fraction as a function of temperature
- clouds_lifrac_scatter_postproc.ncl: additional plots and diagnostics using the output of clouds_lifrac_scatter.ncl for given CMIP5/CMIP6 model pairs
- clouds_pdf.ncl: pdf of cloud parameters
- clouds_seasonal_cycle.ncl: seasonal cycle amplitude
- clouds_taylor.ncl: Taylor diagrams as in Lauer and Hamilton (2013)
- clouds_taylor_double.ncl: Taylor diagrams as in Lauer et al. (2023)

- clouds_zonal.ncl: zonal means of 3-dim variables

User settings in recipe

1. Script clouds.ncl

   Required settings (scripts)

   none

   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
   - multiobs_exclude: list of *observational* datasets to be excluded when calculating uncertainty estimates from multiple observational datasets (see also multiobs_uncertainty)
   - multiobs_uncertainty: calculate uncertainty estimates from multiple observational datasets (true, false); by default, all "obs", "obs6", "obs4mips" and "native6" datasets are used; any of such datasets can be explicitly excluded when also specifying "multiobs_exclude"
   - 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)
   - showyears: add start and end years to the plot titles (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)
   - var: short_name of variable to process (default = "" - use first variable in variable list)

   Required settings (variables)

   none

   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)

   none

   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_dyn_matrix.ncl

   Required settings (scripts)

   - var_x: short name of variable on x-axis
   - var_y: short name of variable on y-axis
   - var_z: short name of variable to be binned
   - xmin: min x value for generating x bins
   - xmax: max x value for generating x bins
   - ymin: min y value for generating y bins
   - ymax: max y value for generating y bins

   Optional settings (scripts)

   - clevels: explicit values for probability labelbar (array)
   - filename_add: optionally add this string to plot filesnames
   - nbins: number of equally spaced bins (var_x), default = 100
   - sidepanels: show/hide side panels (default = False)
   - xlabel: label overriding variable name for x-axis (e.g. SST)
   - ylabel: label overriding variable name for y-axis (e.g. omega500)
   - zdmin: min z value for labelbar (difference plots)
   - zdmax: max z value for labelbar (difference plots)
   - zmin: min z value for labelbar
   - zmax: max z value for labelbar

Required settings (variables)

Optional settings (variables)

- reference_dataset: reference dataset

4. Script clouds_interannual.ncl

   Required settings (scripts)

   none

   Optional settings (scripts)

   - colormap: e.g., WhiteBlueGreenYellowRed, rainbow
   - epsilon: "epsilon" value to be replaced with missing values
   - explicit_cn_levels: use these contour levels for plotting
   - filename_add: optionally add this string to plot filesnames
   - projection: map projection, e.g., Mollweide, Mercator
   - var: short_name of variable to process (default = "" - use first variable in variable list)

   Required settings (variables)

   none

   Optional settings (variables)

   - long_name: description of variable
   - reference_dataset: name of reference datatset

5. Script clouds_ipcc.ncl

   Required settings (scripts)

   none

   Optional settings (scripts)

   - explicit_cn_levels: contour levels
   - highlight_dataset: name of dataset to highlight (default = "MultiModelMean")
   - 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 clouds_lifrac_scatter.ncl

   Required settings (scripts)

   none

   Optional settings (scripts)

   - filename_add: optionally add this string to plot filesnames
   - min_mass: minimum cloud condensate (same units as clw, cli)
   - mm_mean_median: calculate multi-model mean and meadian
   - nbins: number of equally spaced bins (ta (x-axis)), default = 20
   - panel_labels: label individual panels (true, false)
   - PanelTop: manual override for "@gnsPanelTop" used by panel plot(s)s

   Required settings (variables)

   Optional settings (variables)

   - reference_dataset: reference dataset

7. Script clouds_lifrac_scatter_postproc.ncl

   Required settings (scripts)

   - models: array of CMIP5/CMIP6 model pairs to be compared
   - refname: name of reference dataset

   Optional settings (scripts)

   - nbins: number of bins used by clouds_lifrac_scatter.ncl (default = 20)
   - reg: region (string) (default = "")
   - t_int: array of temperatures for printing additional diagnostics

   Required settings (variables)

   none

   Optional settings (variables)

   none

8. Script clouds_pdf.ncl

   Required settings (scripts)

   - xmin: min value for bins (x axis)
   - xmax: max value for bins (y axis)

   Optional settings (scripts)

   - filename_add: optionally add this string to output filenames
   - plot_average: show average frequency per bin
   - region: show only selected geographic region given as latmin, latmax, lonmin, lonmax
   - styleset: "CMIP5", "DEFAULT"
   - ymin: min value for frequencies (%) (y axis)

- ymax: max value for frequencies (%) (y axis)

Required settings (variables)

Optional settings (variables)

- reference_dataset: reference dataset

9. Script clouds_seasonal_cycle.ncl

   Required settings (scripts)

   none

   Optional settings (scripts)

   - colormap: e.g., WhiteBlueGreenYellowRed, rainbow
   - epsilon: "epsilon" value to be replaced with missing values
   - explicit_cn_levels: use these contour levels for plotting
   - filename_add: optionally add this string to plot filesnames
   - projection: map projection, e.g., Mollweide, Mercator
   - showyears: add start and end years to the plot titles (default = false)
   - var: short_name of variable to process (default = "" i.e. use first variable in variable list)

   Required settings (variables)

   Optional settings (variables)

   - long_name: description of variable
   - reference_dataset: name of reference dataset

10. Script clouds_taylor.ncl

    Required settings (scripts)

    none

    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 = "")
    - legend_filter: do not show individual datasets in legend that are of project "legend_filter" (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"
    - multiobs_exclude: list of *observational* datasets to be excluded when calculating uncertainty estimates from multiple observational datasets (see also multiobs_uncertainty)

- multiobs_uncertainty: calculate uncertainty estimates from multiple observational datasets (true, false); by default, all "obs", "obs6", "obs4mips" and "native6" datasets are used; any of such datasets can be explicitly excluded when also specifying "multiobs_exclude"
- 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
- var: short_name of variable to process (default = "" - use first variable in variable list)

Required settings (variables)

- reference_dataset: name of reference data set

Optional settings (variables)

none

11. Script clouds_taylor_double.ncl

Required settings (scripts)

none

Optional settings (scripts)

- filename_add: legacy feature: arbitrary string to be added to all filenames of plots and netcdf output produced (default = "")
- multiobs_exclude: list of *observational* datasets to be excluded when calculating uncertainty estimates from multiple observational datasets (see also multiobs_uncertainty)
- multiobs_uncertainty: calculate uncertainty estimates from multiple observational datasets (true, false); by default, all "obs", "obs6", "obs4mips" and "native6" datasets are used; any of such datasets can be explicitely excluded when also specifying "multiobs_exclude"
- projectcolors: colors for each projectgroups (e.g. (/"(/0.0, 0.0, 1.0/)", "(/1.0, 0.0, 0.0/)"/)
- projectgroups: calculated mmm per "projectgroup" (e.g. (/"cmip5", "cmip6"/))
- styleset: "CMIP5", "DEFAULT" (if not set, CLOUDS_TAYLOR_DOUBLE 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)
- var: short_name of variable to process (default = "" - use first variable in variable list)

Required settings (variables)

- reference_dataset: name of reference data set

Optional settings (variables)

12. Script clouds_zonal.ncl

Required settings (scripts)

none

Optional settings (scripts)

- embracesetup: True = 2 plots per line, False = 4 plots per line (default)

- explicit_cn_levels: explicit contour levels for mean values (array)
- explicit_cn_dlevels: explicit contour levels for differences (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)
- showdiff: calculate and plot differences (default = False)
- rel_diff: if showdiff = True, then plot relative differences (%) (default = False)
- rel_diff_min: lower cutoff value in case of calculating relative differences (in units of input variable)
- t_test: perform t-test when calculating differences (default = False)
- timemean: time averaging - "seasonal" = DJF, MAM, JJA, SON), "annual" = annual mean
- units_to: target units (automatic conversion)

Required settings (variables)

none

Optional settings (variables)

- long_name: variable description
- reference_dataset: reference dataset; REQUIRED when calculating differences (showdiff = True)
- units: variable units (for labeling plot only)

Variables

- cl (atmos, monthly mean, longitude latitude time)
- clcalipso (atmos, monthly mean, longitude latitude time)
- cli (atmos, monthly mean, longitude latitude time)
- clw (atmos, monthly mean, longitude latitude time)
- 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)
- prw (atmos, monthly mean, longitude latitude time)
- rlut, rlutcs (atmos, monthly mean, longitude latitude time)
- rsut, rsutcs (atmos, monthly mean, longitude latitude time)
- ta (atmos, monthly mean, longitude latitude time)
- wap (atmos, monthly mean, longitude latitude time)

Observations/realanyses

- CALIPSO-GOCCP
- CALIPSO-ICECLOUD
- CERES-EBAF
- CLARA-AVHRR
- CLOUDSAT-L2
- ERA5
- ERA-Interim
- ESACCI-CLOUD
- ESACCI-WATERVAPOUR
- GPCP-SG
- ISCCP-FH
- MAC-LWP
- MODIS
- PATMOS-x
- UWisc

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.
- Lauer, A., L. Bock, B. Hassler, M. Schröder, and M. Stengel, Cloud climatologies from global climate models - a comparison of CMIP5 and CMIP6 models with satellite data, J. Climate, 36(2), doi: 10.1175/JCLI-D-22-0181.1, 2023.

Example plots

5.2.4 Evaluate water vapor short wave radiance absorption schemes of ESMs with the observations, including ESACCI data.

Overview

The recipe contains several diagnostics to use ESACCI water vapour data to evaluate CMIP models.

The diagnostic deangelisf3f4.py reproduces figures 3 and 4 from DeAngelis et al. (2015): See also doc/sphinx/source/recipes/recipe_deangelis15nat.rst 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.

The diagnostic diag_tropopause.py plots given variable at cold point tropopause height, here Specific Humidity (hus) is used. This will be calculated from the ESACCI water vapour data CDR-4, which are planed to consist of three-dimensional vertically resolved monthly mean water vapour data (in ppmv) with spatial resolution of 100 km, covering

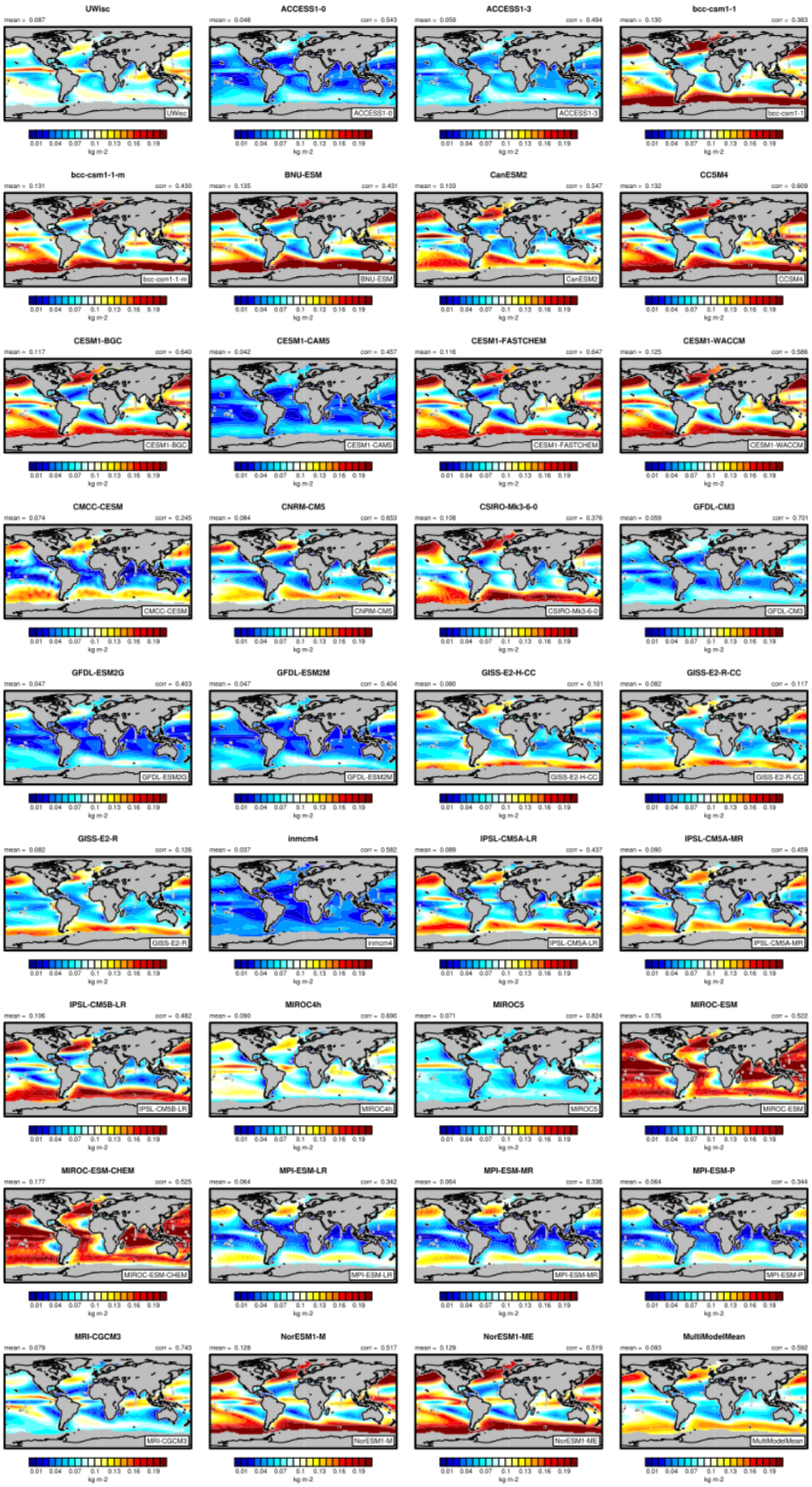


Fig. 47: The 20-yr average LWP (1986-2005) from the CMIP5 historical model runs and the multi-model mean in comparison with the UWisc satellite climatology (1988-2007) based on SSM/I, TMI, and AMSR-E (O'Dell et al. 2008). Produced with recipe_lauer13jclim.yml (diagnostic script clouds.ncl).

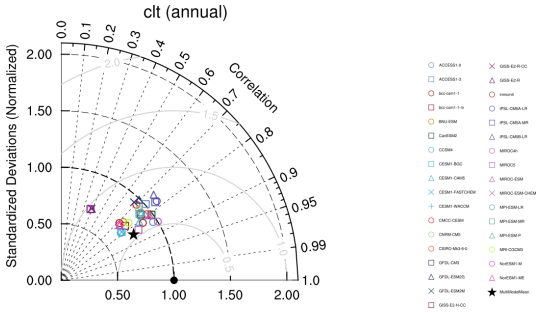


Fig. 48: Taylor diagram showing the 20-yr annual average performance of CMIP5 models for total cloud fraction as compared to MODIS satellite observations. Produced with recipe_lauer13jclim.yml (diagnostic script clouds_taylor.ncl).

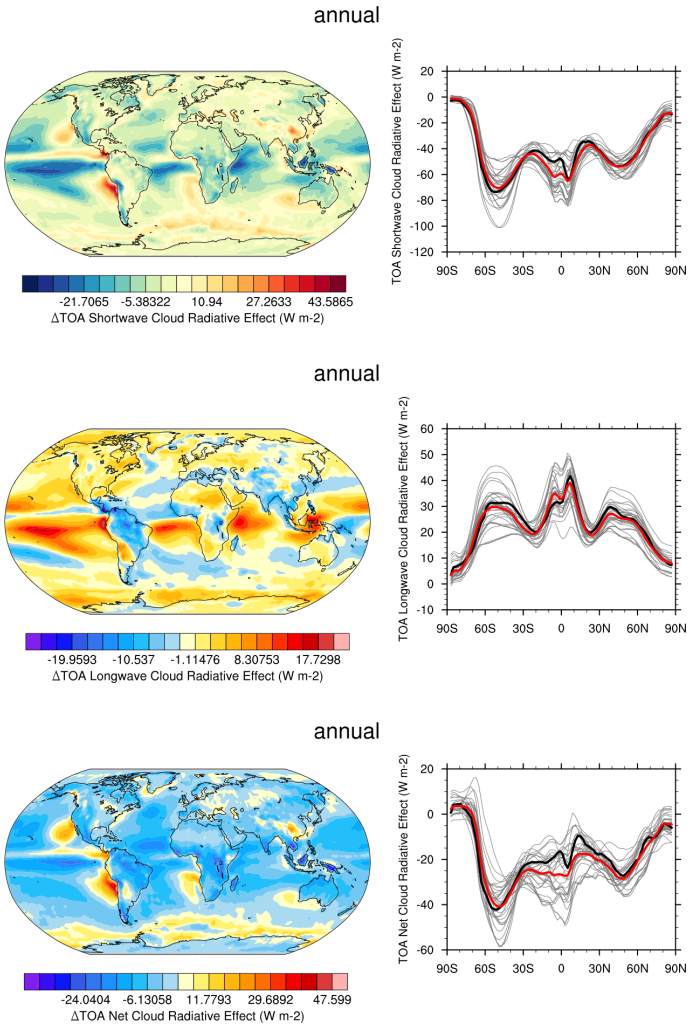


Fig. 49: 20-year average (1986-2005) annual mean cloud radiative effects of CMIP5 models against the CERES-EBAF (2001–2012). Top row shows the shortwave effect; middle row the longwave effect, and bottom row the net effect. Multi-model mean biases against CERES-EBAF are shown on the left, whereas the 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. Produced with recipe_clouds_ipcc.yml (diagnostic script clouds_ipcc.ncl).

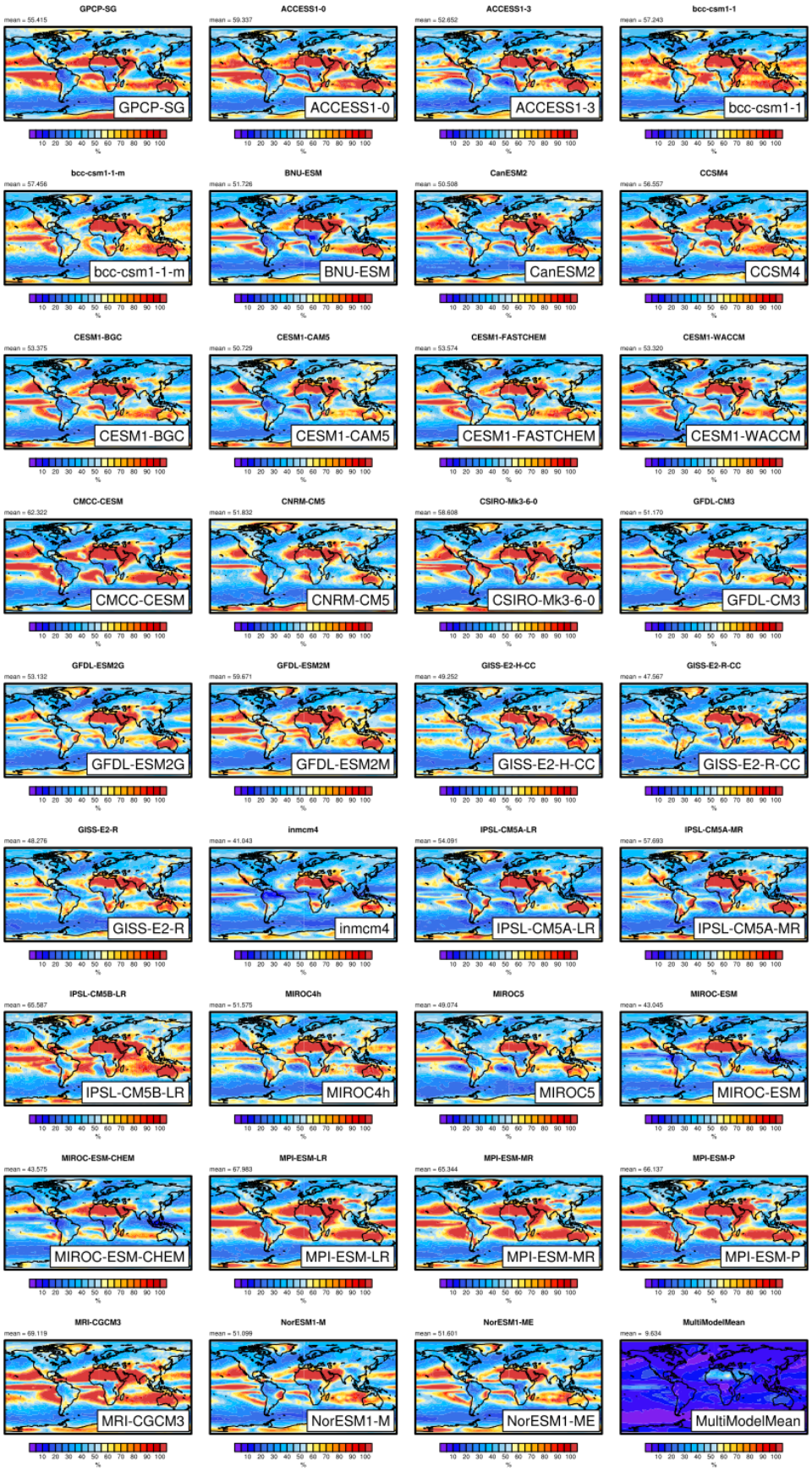


Fig. 50: Interannual variability of modeled and observed (GPCP) precipitation rates estimated as relative temporal standard deviation from 20 years (1986-2005) of data. The temporal standard deviations are calculated from monthly anomalies after subtracting the climatological mean seasonal cycle. Produced with recipe_lauer13jclim.yml (clouds_interannual.ncl).

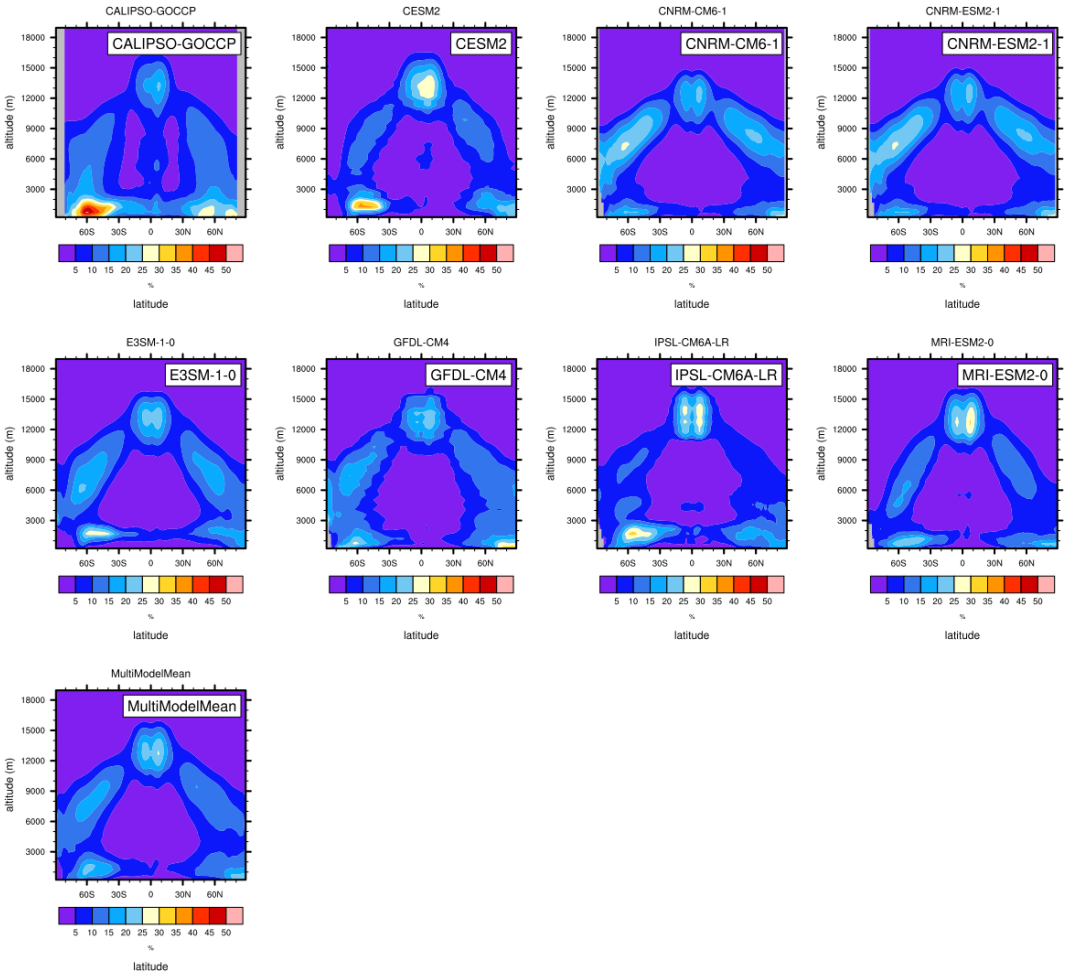


Fig. 51: Zonal mean of the multi-year annual mean cloud fraction as seen from CALIPSO from CMIP6 models in comparison to CALIPSO-GOCCP data. Produced with recipe_lauer22jclim_fig3-4_zonal.yml (diagnostic script clouds_zonal.ncl).

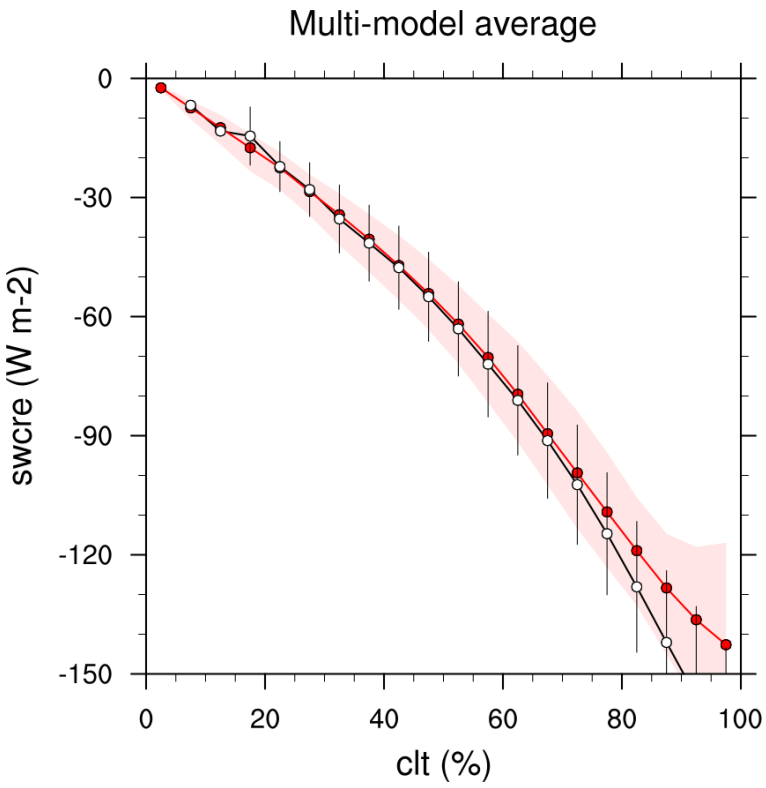


Fig. 52: Multi-year seasonal average (December-January-February) of cloud shortwave radiative effect (W m^{-2}) vs. total cloud fraction (clt, %) averaged over the Southern Ocean defined as latitude belt 30°S-65°S (ocean grid cells only). Shown are the CMIP6 multi-model mean (red filled circles and lines) and observational estimates from ESACCI-CLOUD (black circles and lines). The red shaded areas represent the range between the 10th and 90th percentiles of the results from all individual models. Produced with recipe_lauer22jclim_fig9-11ab_scatter.yml (diagnostic script clouds_scatter.ncl).

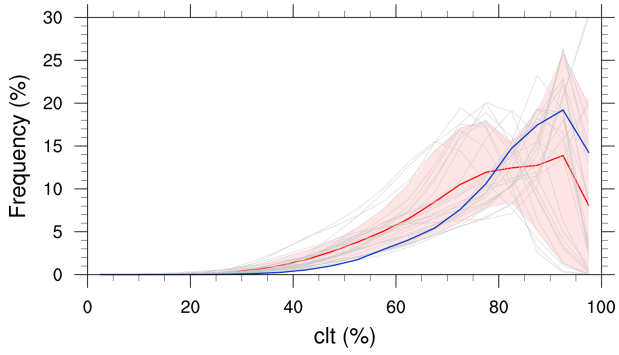


Fig. 53: Frequency distribution of monthly mean total cloud cover from CMIP6 models in comparison to ESACCI-CLOUD data. The red curve shows the multi-model average, the blue curve the ESACCI-CLOUD data and the thin gray lines the individual models. The red shading shows ±1 standard deviation of the inter-model spread. Produced with recipe_lauer22jclim_fig9-11c_pdf.yml (diagnostic script clouds_pdf.ncl).

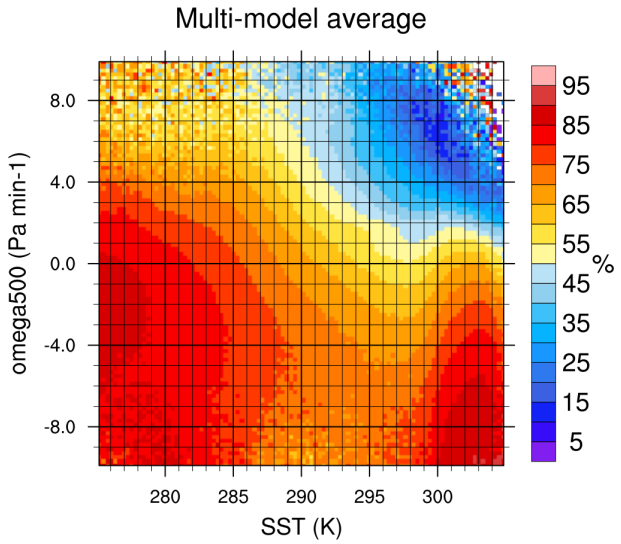


Fig. 54: 2-dimensional distribution of average total cloud cover (clt) binned by sea surface temperature (SST, x-axis) and vertical velocity at 500 hPa ($_{500}$, y-axis) averaged over 20 years and all grid cells over the ocean. Produced with recipe_lauer22jclim_fig8_dyn.yml (diagnostic script clouds_dyn_matrix.ncl).

the troposphere and lower stratosphere. The envisaged coverage is 2010-2014. The calculation of hus from water vapour in ppmv will be part of the cmorizer. Here, ERA-Interim data are used.

The diagnostic diag_tropopause_zonalmean.py plots zonal mean for given variable for all pressure levels between 250 and 1hPa and at cold point tropopause height. Here Specific Humidity (hus) is used. This will be calculated from the ESACCI water vapour data CDR-3, which are planed to contain the vertically resolved water vapour ECV in units of ppmv (volume mixing ratio) and will be provided as zonal monthly means on the SPARC Data Initiative latitude/pressure level grid (SPARC, 2017; Hegglin et al., 2013). It covers the vertical range between 250 hPa and 1 hPa, and the time period 1985 to the end of 2019. The calculation of hus from water vapour in ppmv will be part of the cmorizer. Here, ERA-Interim data are used.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_cmug_h2o.yml

Diagnostics are stored in diag_scripts/

- deangelis15nat/deangelisf3f4.py
- cmug_h2o/diag_tropopause.py
- cmug_h2o/diag_tropopause_zonalmean.py

User settings in recipe

The recipe can be run with different CMIP5 and CMIP6 models.

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. Currently, HOAPS data are included as place holder for expected ESACCI-WV data, type CDR-2: Gridded monthly time series of TCWV in units of kg/m2 (corresponds to prw) that cover the global land and ocean areas with a spatial resolution of 0.05° / 0.5° for the period July 2002 to December 2017.

Variables

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)

diag_tropopause.py: * *hus* (atmos, monthly, longitude, latitude, time, plev) * *ta* (atmos, monthly, longitude, latitude, time, plev)

diag_tropopause_zonalmean.py: * *hus* (atmos, monthly, longitude, latitude, time, plev) * *ta* (atmos, monthly, longitude, latitude, time, plev)

Observations and reformat scripts

deangelisf3f4.py:

- ***rsnstcs*:**
  CERES-EBAF
- ***prw***
  HOAPS, planed for ESACCI-WV data, type CDR-2

diag_tropopause.py:

- ***hus***
  ERA-Interim, ESACCI water vapour paned

diag_tropopause_zonalmean.py:

- ***hus***
  ERA-Interim, ESACCI water vapour paned

References

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

Example plots

5.2.5 Cloud Regime Error Metric (CREM)

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.

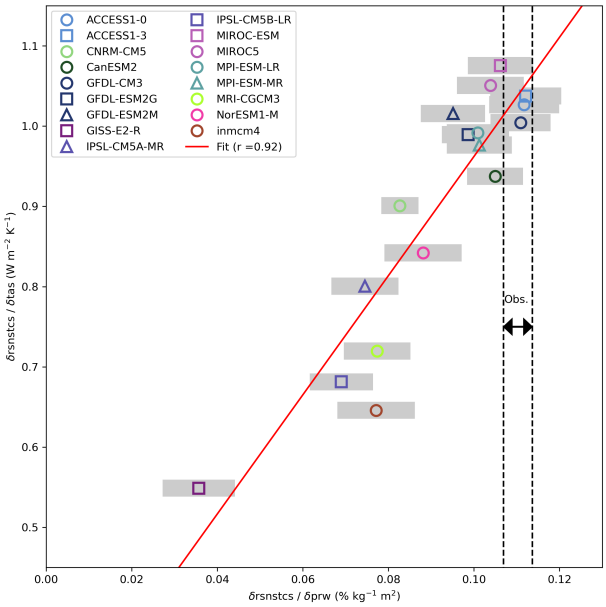


Fig. 55: Scatter plot and regression line computed between the ratio of the change of net short wave radiation (rsnst) and the change of the Water Vapor Path (prw) against the ratio of the change of netshort wave radiation for clear skye (rsnstcs) and the the change of surface temperature (tas). The 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 prw observations ESACCI CDR-2 data from 2003 to 2014 are used.

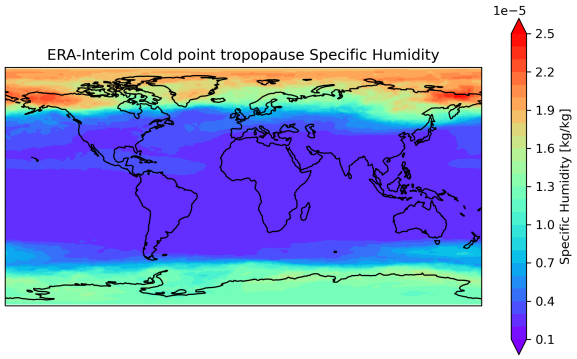


Fig. 56: Map of the average Specific Humidity (hus) at the cold point tropopause from ERA-Interim data. The diagnostic averages the complete time series, here 2010-2014.

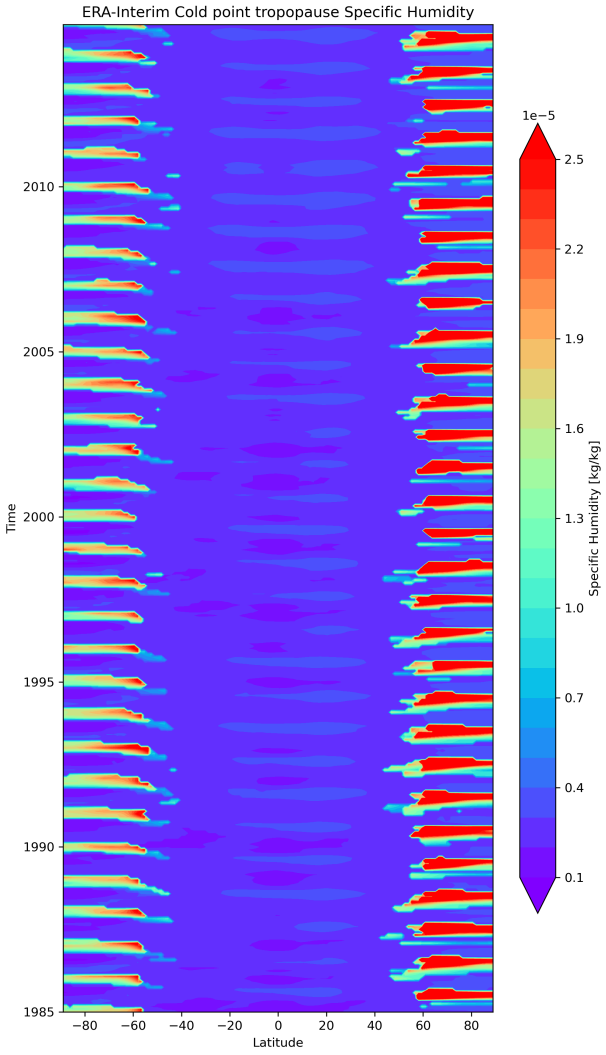


Fig. 57: Latitude versus time plot of the Specific Humidity (hus) at the cold point tropopause from ERA-Interim data.

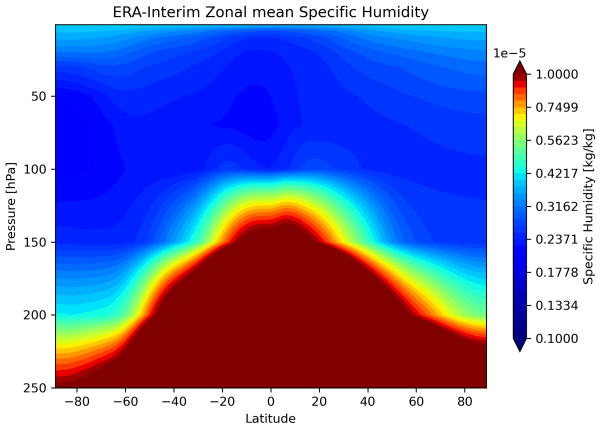


Fig. 58: Zonal average Specific Humidity (hus) between 250 and 1 hPa from ERA-Interim data. The diagnostic averages the complete time series, here 1985-2014.

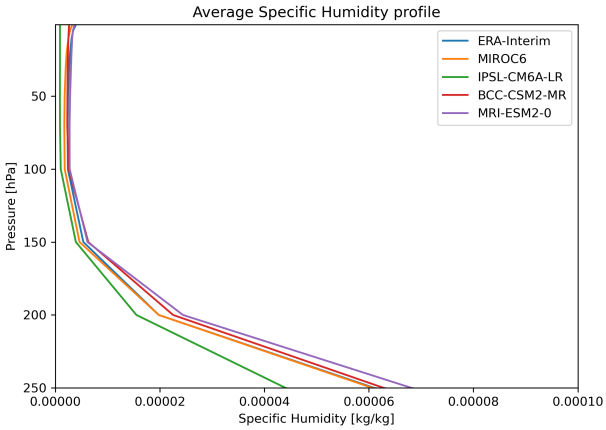


Fig. 59: Average Specific Humidity (hus) profile between 250 and 1 hPa from ERA-Interim and CMIP6 model data. The diagnostic averages the complete time series, here 1985-2014.

The original publication recommends to use sea ice fields from one model also for other models that do not provide daily sea ice concentration. This is possible as sea ice concentrations are prescribed in the AMIP simulations and has been done to produce the example figure shown below.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_williams09climdyn_CREM.yml

Diagnostics are stored in diag_scripts/crem/

- ww09_esmvaltool.py

User settings

None.

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.

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.

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.

Example plots

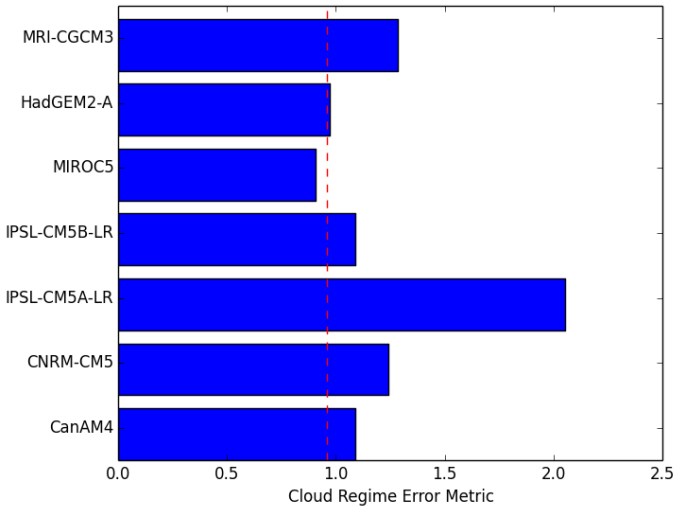


Fig. 60: Cloud Regime Error Metrics (CREMpd) from William and Webb (2009) applied to selected CMIP5 AMIP simulations. A perfect score with respect to ISCCP is zero; the dashed red line is an indication of observational uncertainty. Note: as daily sea ice concentration (sic) is not available for all models shown, the regridded fields from CanAM4 have been used for all models.

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

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.

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

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.

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)

Observations and reformat scripts

deangelisf1b.py: * None

deangelisf2ext.py: * None

deangelisf3f4.py:

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

References

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

Example plots

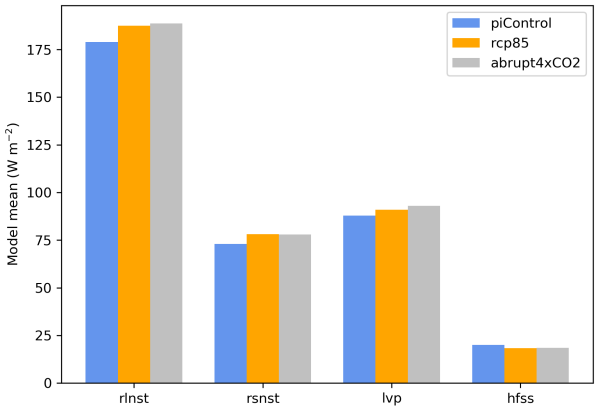


Fig. 61: Global average multi-model mean comparing different model experiments for the sum of upward long wave flux at TOA and net downward long wave flux at the surface (rlnst), heating from short wave absorption (rsnst), latent heat release from precipitation (lvp), and sensible heat flux (hfss). The 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.

5.2.7 Diurnal temperature range

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).

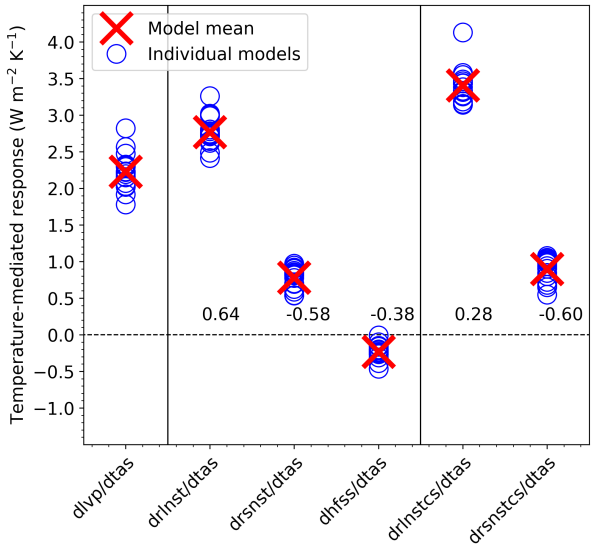


Fig. 62: The temperature-mediated response of each atmospheric energy budget term for each model as blue circles and the model mean as a red cross. The numbers above the abscissa are the cross-model correlations between dlvp/dtas and each other temperature-mediated response.'

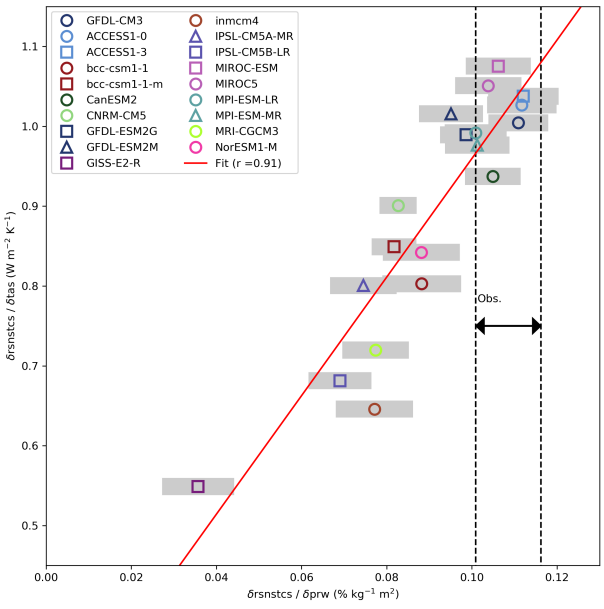


Fig. 63: Scatter plot and regression line the between the ratio of the change of net short wave radiation (rsnst) and the change of the Water Vapor Path (prw) against the ratio of the change of netshort wave radiation for clear skye (rsnstcs) and the the change of surface temperature (tas). The width 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.

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.

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.

User settings

User setting files are stored in recipes/

1. recipe_diurnal_temperature_index.yml

   Required settings for script

   - None

Variables

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

Observations and reformat scripts

None

References

- Amiri, S. (2013). Economic and Environmental Benefits of CHP-based District Heating Systems in Sweden. Retrieved from https://www.diva-portal.org/smash/get/diva2:622636/FULLTEXT01.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), and Planton, S.(CNRMGAME)(2014). Impact du changement climatique sur la gestion des réseaux de chaleur. DALKIA, Étude réalisée pour l'entreprise DALKIA. Last access 24.02.2021. https://docplayer.fr/9496504-Impact-du-changement-climatique-sur-la-gestion-des-reseaux-de-chaleur.html
- 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

Example plots

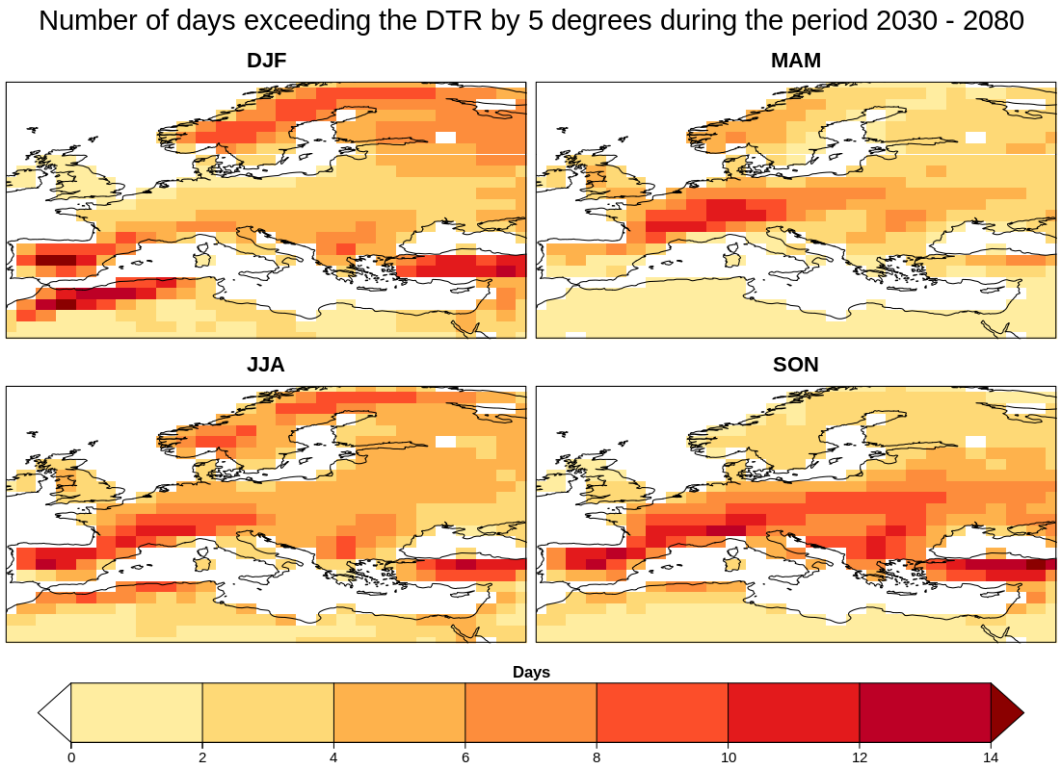


Fig. 64: Mean number of days exceeding the Diurnal Temperature Range (DTR) simulated during the historical period (1961-1990) by 5 degrees during the period 2030-2080. The result is derived from one RCP 8.5 scenario simulated by MPI-ESM-MR.

5.2.8 Droughts

Consecutive Dry Days (CDD)

Meteorological drought can in its simplest form be described by a lack of precipitation. The Consecutive Dry Days (CDD) diagnostic calculates the longest period of dry days based on user defined thresholds.

More details and usage examples can be found in the *CDD recipe documentation*.

Potential or Reference Evapotranspiration (PET, ET0)

The Potential Evapotranspiration (PET) is a measure of the evaporative demand of the atmosphere. It represents the amount of water that would evaporate from a reference surface, i.e. fully watered grass land. `pet.R` is able to calculate PET based on a method of users choice (`pet_type`). The methods require different input variables for example:

- Penman: tas, tasmin, tasmax, sfcWind, rsds, clt, hurs, ps
- Hargreaves: tasmin, tasmax, (rsds, pr)
- Thornthwaite: tas

A complete list and more details can be found in *SPEI recipe documentation*.

Standardized Precipitation-Evapotranspiration Index (SPEI)

Meteorological droughts are often described using the Standardized Precipitation Index (SPI; McKee et al, 1993), which in a standardized way describes local precipitation anomalies.

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 or agricultural drought. 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, by estimating the Potential Evapotranspiration (PET).

More details and usage examples can be found in the *SPEI recipe documentation*.

Available recipes and diagnostics

Consecutive dry days

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 a 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.

Available recipes and diagnostics

Recipes are stored in recipes/droughts/

- recipe_cdd.yml

Diagnostics are stored in diag_scripts/droughts/

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

User settings in recipe

Script cdd.py

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

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

Under `plot`:

cmap: str, optional
: the name of a colormap. cmocean colormaps are also supported.

other keyword arguments to *esmvaltool.diag_scripts.shared.plot.global_pcolormesh()* can also be supplied.

Variables

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

Example plots

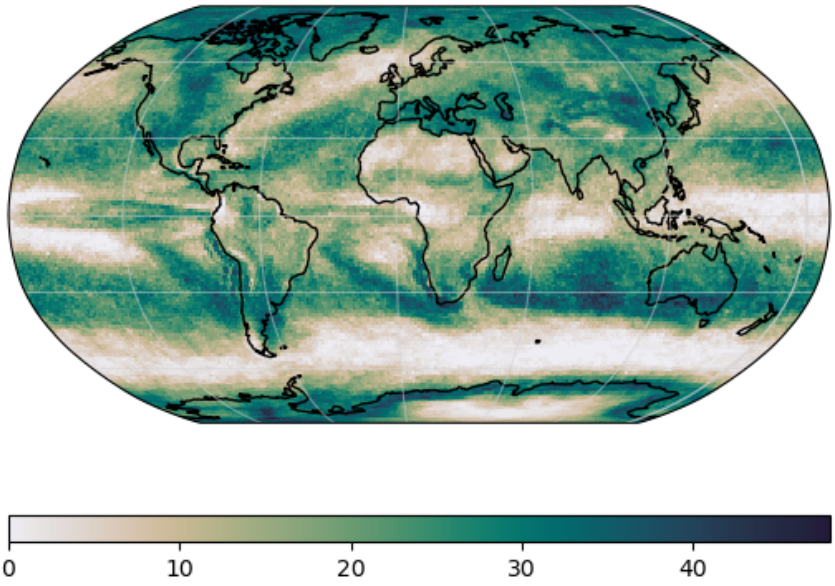


Fig. 65: Example of the number of occurrences with consecutive dry days of more than five days in the period 2001 to 2002 for the CMIP5 model bcc-csm1-1-m.

Standardized Precipitation-Evapotranspiration Index (SPEI)

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 droughts.

A hydrological drought occurs when low water supply becomes evident, especially in streams, reservoirs, and groundwater levels, usually after extended periods of meteorological drought.

ESMs 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. To consider evapotranspiration and plant water stress, potential evapotranspiration (PET) is calculated based on atmospheric variables. Different methods to derive PET are described below.

This page documents a set of R diagnostics based on the SPEI.R library. `recipes/roughts/recipe_spei.yml` is an example how to calculate and plot SPEI using `diag_scripts/droughts/pet.R` and `diag_scripts/droughts/spei.R`.

pet.R

The potential evapotranspiration (PET) is a measure of the evaporative demand of the atmosphere. It represents the amount of water that would evaporate from a reference surface, i.e. fully watered grass land. `pet.R` is able to calculate PET based on a method of the users choosing (`pet_type`) using the SPEI.R library. The approximations require different input variables. To control which variables are available to which diagnostic script in a more complex recipe, they can be set explicitly as ancestors.

- Thornthwaite: tas
- Hargreaves: tasmin, tasmax, (rsdt, pr)
- Penman: tasmin, tasmax, sfcWind, ps, rsds, (rsdt, clt, hurs)

The Thornthwaite equation (Thornthwaite, 1948) is the simplest one based solely on temperature. Hargreaves (1994) provides an equation based on daily minimum (tasmin) and maximum temperature (tasmax) and external radiation (rsdt). If precipitation data (pr) is provided and `use_pr: TRUE` it will be used as a proxy for irradiation to correct PET following Droogers and Allen (2002). The Penman-Monteith formula additionally considers surface windspeed (sfcWind), pressure (ps), and relative humidity (hurs). Some of these variables can be approximated if not available (for example by providing clt instead of rsds). There are further modifications to the Penman-Monteith equation, that can be selected using the `method` key for `pet_type: Penman`. Details about the different methods can be found in the SPEI.R package documentation (Beguería and Vicente-Serrano, 2011).

User settings

pet_type: str
: The method used to calculate the potential evapotranspiration. Some settings and required variables depend on the calculation method. Options are: Penman, Thornthwaite, Hargreaves

use_pr: boolean, optional
: Use precipitation as proxy for irradation to correct PET. Only used for `pet_type: Hargreaves`. By default `FALSE`.

method: str, optional
: Method used for PET calculation. Only used for `pet_type: Penman`. Options are: `"ICID"` (Allen et al., 1994), `"FAO"` (Allen et al., 1998), `"ASCE"` (Walter et al. 2002). By default: `"ICID"`.

crop: str, optional
: Crop type for PET calculation. Only used for `pet_type: Penman`. Options are: `"short"`, `"tall"`. By default: `"tall"`.

spei.R

The Standardized Precipitation-Evapotranspiration Index (SPEI) is calculated by fitting a probability distribution to the accumulated water budget (pr-PET) for each grid cell and month of the year. The accumulation period is configurable using the `smooth_month` setting. The PDF is transformed to a normal distribution. Based on given percentiles, index values from -2 (extreme droughts) to 2 (extreme wet spells) are assigned. The distribution used to fit the water budget can be configured through the `distribution` setting. The required PET can be provided by setting the pet.R diagnostic as an ancestor or adding an `evspsblpot` variable. To use alternative variables (i.e. actual evapotranspiration) change the `short_name` setting accordingly.

> SPI is considered as a special case of the SPEI. Provide only precipitation as input variable or ancestor and set `distribution="Gamma"`, to calculate SPI.

User settings

smooth_month: int
The number of months (scale) to accumulate. Common choices are 3, 6, 9, 12.

write_coeffs: boolean, optional
Save fitting coefficients. By default `FALSE`.

write_wb: boolean, optional
Write water balance to netcdf file. By default `FALSE`.

short_name_pet: string, optional
Short name of the variable to use as PET. By default `"evspsblpot"`.

distributionn: string, optional
Type of distribution used for SPEI calibration. Possible options are: "Gamma", "log-Logistic", "Pearson III". By default `"log-Logistic"`.

refstart_year: int, optional
First year of the reference period. By default `null` (first year of time series).

refstart_month: int, optional
First month of reference period. By default `1`.

refend_year: int, optional
Last year of the reference period. By default `null` (last year of time series).

refend_month: integer, optional
Last month of reference period. By default `12`.

References

- Allen, R. G., Pereira, L. S., Raes, D., and Smith, M.: Crop Evapotranspiration Guidelines for Computing Crop Water Requirements, no. 56 in FAO Irrigation and Drainage Paper, Food and Agriculture Organization of the United Nations, Rome, 1998.
- Allen, Richard. G., Smith, M., Perrier, A. and P., Luis S., & others. (1994). An update for thedefinition of reference evapotranspiration. ICID Bulletin, 43(2), 1-34.
- Beguería, S., & Vicente-Serrano, S. M. (2011). SPEI: Calculation of the Standardized Precipitation-Evapotranspiration Index (p. 1.8.1) [Dataset]. https://doi.org/10.32614/CRAN.package.SPEI
- Droogers P., Allen R. G., (2002). Estimating reference evapotranspiration under inaccurate data conditions. Irrigation and Drainage Systems 16: 33-45.
- Hargreaves G.H., (1994). Defining and using reference evapotranspiration. Journal of Irrigation and Drainage Engineering 120: 1132-1139.
- 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.
- Monteith, J.L., 1965. Evaporation and Environment. 19th Symposia of the Society for Experimental Biology, University Press, Cambridge, 19:205-234.
- Thornthwaite, C. W., (1948). An approach toward a rational classification of climate. Geogr. Rev., 38, 55-94. https://doi.org/10.1097/00010694-194807000-00007
- 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.

- Walter I.A. and 14 co-authors, 2002. The ASCE standardized reference evapotranspiration equation. Rep. Task Com. on Standardized Reference Evapotranspiration July 9, 2002, EWRI-Am. Soc. Civil Engr., Reston, VA, 57 pp.

Example plots

Note

To recreate figure 3 and 4 from Weigel et al. (2021) an older version of ESMValTool is required: *Legacy Recipe: Standardized Precipitation-Evapotranspiration Index (SPEI)*.

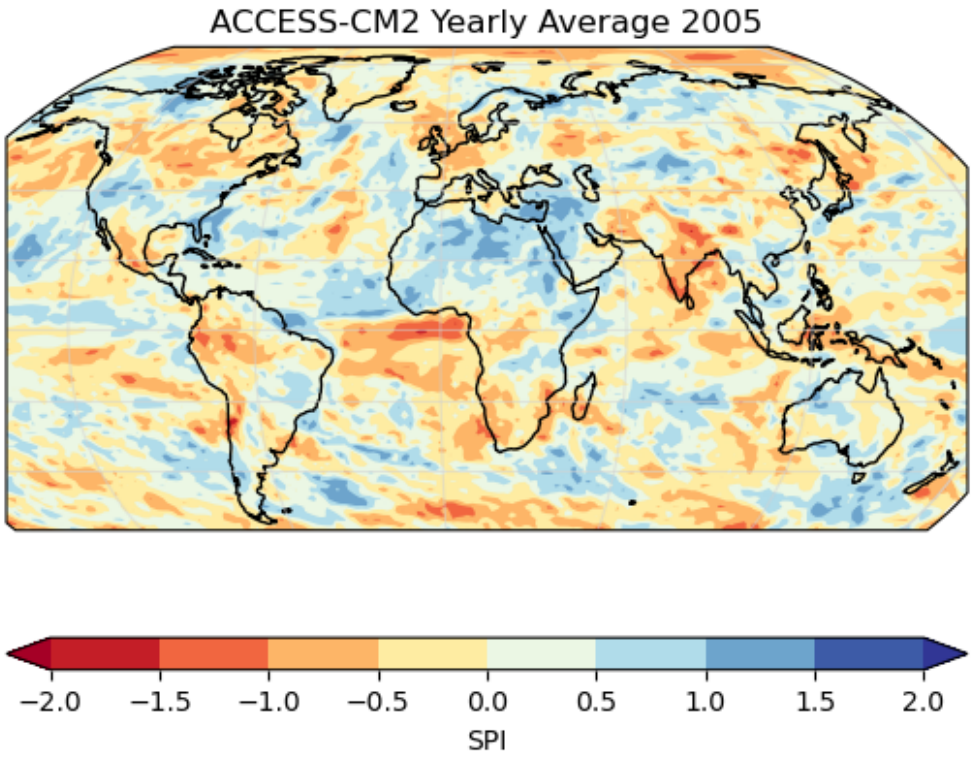


Fig. 66: Example plot of SPI averaged over the year 2005. The reference period for index calibration is 2000-2005.

Drought characteristics following Martin (2018)

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.

Available recipes and diagnostics

Recipes are stored in `recipes/droughts/`

- recipe_martin18grl.yml

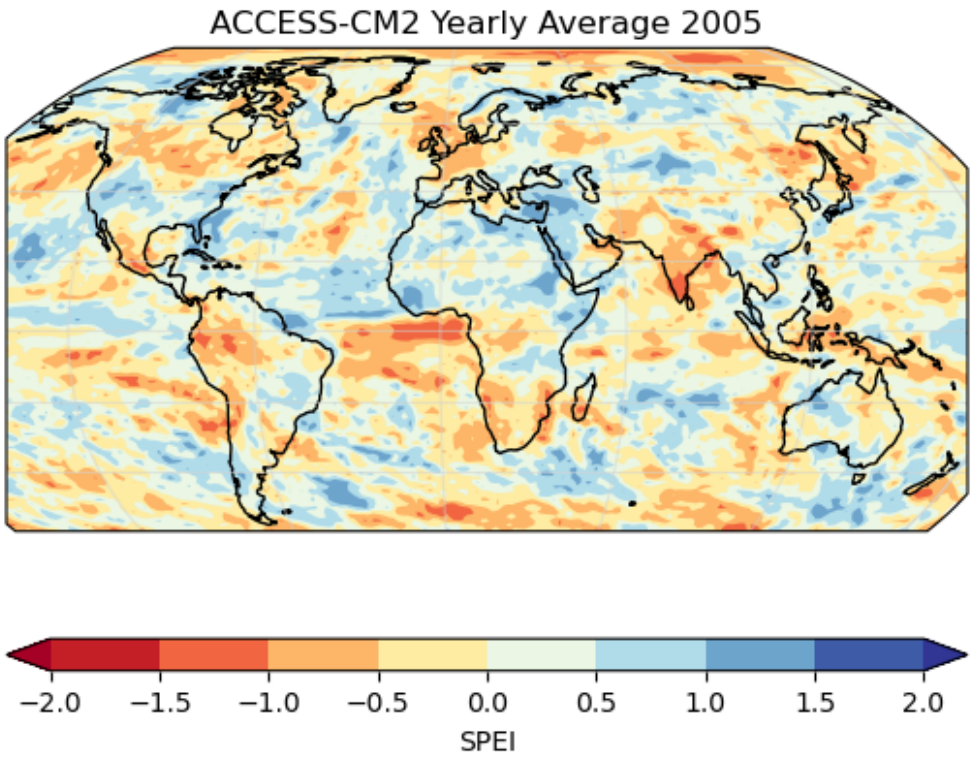


Fig. 67: Example plot of SPEI averaged over the year 2005. The reference period for index calibration is 2000-2005.

Diagnostics are stored in `diag_scripts/droughts/`

- *droughts/spei.R*
- *droughts/collect_drought.py*

Functions for metric calculation, plots and utility can be found in `droughts/collect_drought_func.py`.

User settings in recipe

The recipe calculates SPI for two different time periods. The first part compares CMIP5 models with the CRU observational dataset. The second part compares the last 51 years of the 21st century with the historical period (1950-2000) for the RCP8.5 scenario.

The recipe can be run with different CMIP5 and CMIP6 models and one observational or reanalysis data set. The latter is specified as `reference_dataset` in the recipe.

The distribution (`distribution: Gamma`) and the representative time scale (`smooth_month: 6`) can be changed by the user too. A complete list of settings and their description can be found in the *SPEI recipe* and *Collect drought API documentation*.

Variables

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

Observations and reformat scripts

None

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.

Example plots

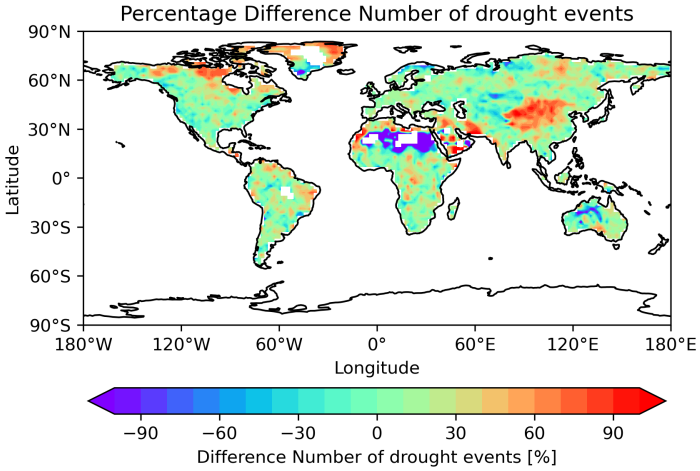


Fig. 68: Global map of the percentage difference between multi-model mean of 15 CMIP models and the CRU data for the number of drought events [%] based on SPI.

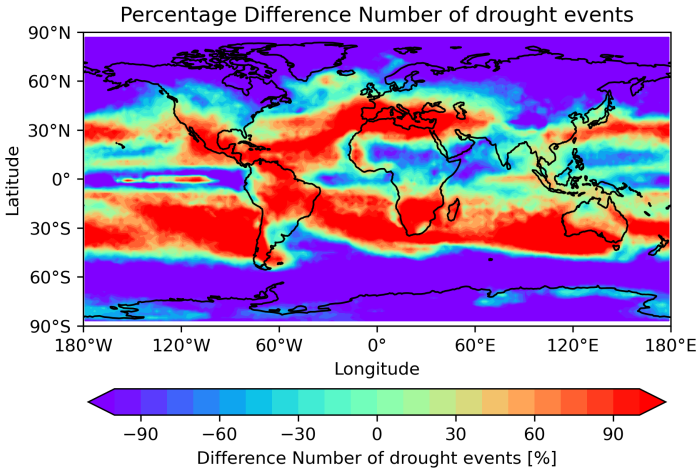


Fig. 69: Global map of the percentage difference between multi-model mean for RCP8.5 scenarios (2050-2100) runs and historical data (1950-2000) for 15 CMIP models for the number of drought events [%] based on SPI.

Diagnostics are stored in `diag_scripts/droughts/`. An incomplete list of diagnostics that are used in different recipes is shown below. Some recipes might use additional diagnostics. See the corresponding recipe documentation for more details.:

- cdd.py: calculate Consecutive Dry Days (CDD)
- pet.R: calculate Potential Evaporanspiration (PET)
- spei.R: calculate Standardized Precipitation-Evapotranspiration Index (SPEI)

User settings

The configuration that can be used in the recipes is documented in the corresponding recipe or diagnostics API documentation linked in the list above. *SPEI recipe documentation*.

Variables

- pr (atmos, daily/monthly mean, time latitude longitude)
- tas (atmos, monthly mean, time latitude longitude)
- tasmin (atmos, monthly mean, time latitude longitude)
- tasmax (atmos, monthly mean, time latitude longitude)
- sfcWind (atmos, monthly mean, time latitude longitude)
- rsds (atmos, monthly mean, time latitude longitude)
- rsdt (atmos, monthly mean, time latitude longitude)
- clt (atmos, monthly mean, time latitude longitude)
- hurs (atmos, monthly mean, time latitude longitude)
- ps (atmos, monthly mean, time latitude longitude)

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.
- 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.

5.2.9 Eady growth rate

Overview

This recipe computes the maximum Eady Growth Rate and performs the annual and seasonal means, storing the results for each dataset. For the seasonal means, the results are plotted over the North-Atlantic region for the selected pressure levels.

Available recipes and diagnostics

Recipes are stored in `esmvaltool/recipes/`

- `recipe_eady_growth_rate.yml`

Diagnostics are stored in `esmvaltool/diag_scripts/eady_growth_rate/`

- `eady_growth_rate.py`: Computes and stores the eady growth rate. Plots can be produced for the seasonal mean over the North Atlantic region.

User settings in recipe

1. Script `eady_growth_rate.py`

   Required settings for script

   - `time_statistic`: Set to *'annual'* to compute the annual mean. Set to *'seasonal'* to compute the seasonal mean.

   Optional settings for script

   - `plot_levels`: list of pressure levels to be plotted for the seasonal mean. If not specified, all levels will be plotted.

Variables

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

References

- Moreno-Chamarro, E., Caron, L-P., Ortega, P., Loosveldt Tomas, S., and Roberts, M. J., Can we trust CMIP5/6 future projections of European winter precipitation?. Environ. Res. Lett. 16 054063
- Brian J Hoskins and Paul J Valdes. On the existence of storm-tracks. Journal of the atmospheric sciences, 47(15):1854–1864, 1990.

Example plots

5.2.10 Extreme Events Indices (ETCCDI)

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.

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

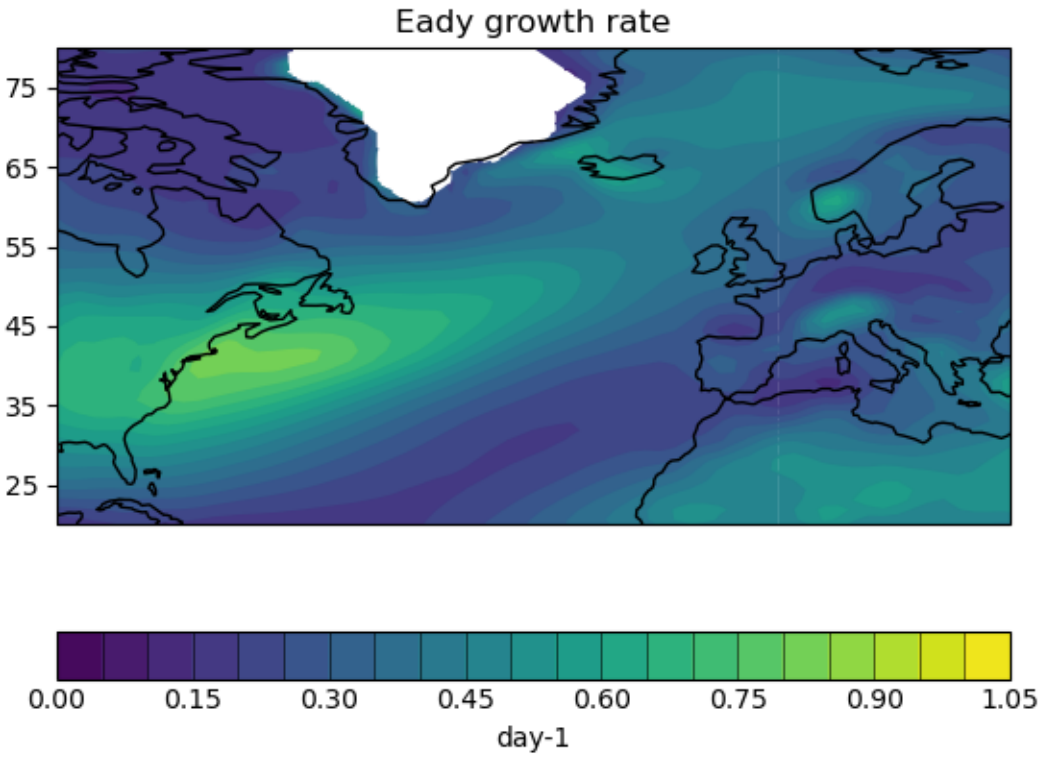


Fig. 70: Eady Growth Rate values over the North-Atlantic region at 70000 Pa.

- make_timeseries_plot.r
- cfg_climdex.r
- cfg_extreme.r

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)

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)

Observations and reformat scripts

None.

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

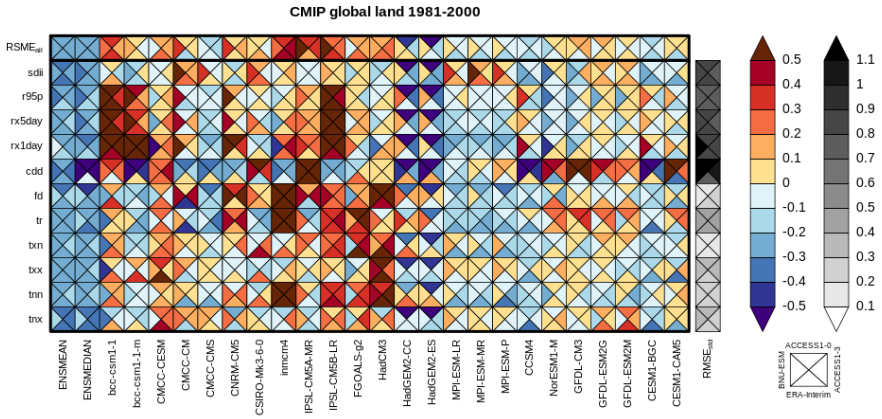


Fig. 71: Portrait plot of relative error metrics for the CMIP5 temperature and precipitation extreme indices evaluated over 1981-2000. Reproduces Fig. 9.37 of the IPCC AR5 report, Chapter 9.

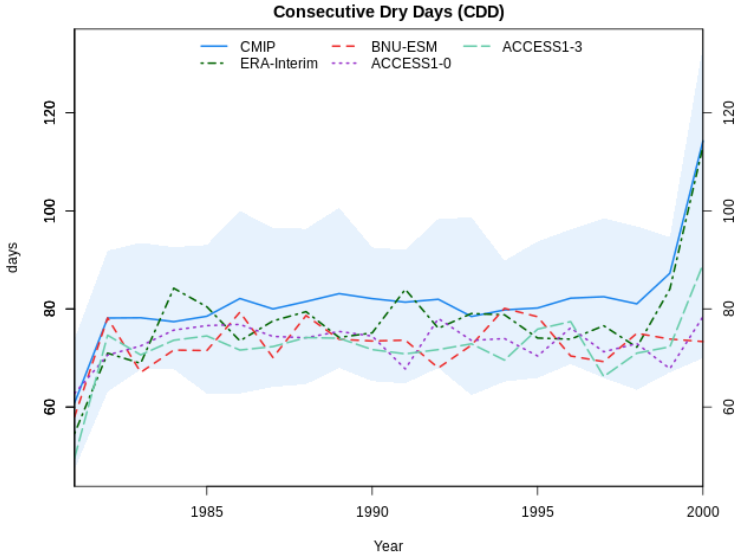


Fig. 72: Timeseries of the Consecutive Dry Days index over 1981-2000 for a selection of CMIP5 models, the CMIP5 multi-model mean (CMIP) and ERA-Interim. Shading is used to reproduce the multi-model spread.

Example plots

5.2.11 Combined Climate Extreme Index

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 expects data of daily 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).

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.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_extreme_index.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- extreme_index.R

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)

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)

Observations and reformat scripts

None

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

Example plots

5.2.12 Spatially resolved evaluation of ESMs with satellite column-averaged CO_2

Overview

This recipe reproduces the figures of Gier et al. (2020). It uses satellite column-averaged CO_2 data to evaluate ESMs by plotting several quantities such as timeseries, seasonal cycle and growth rate in different areas.

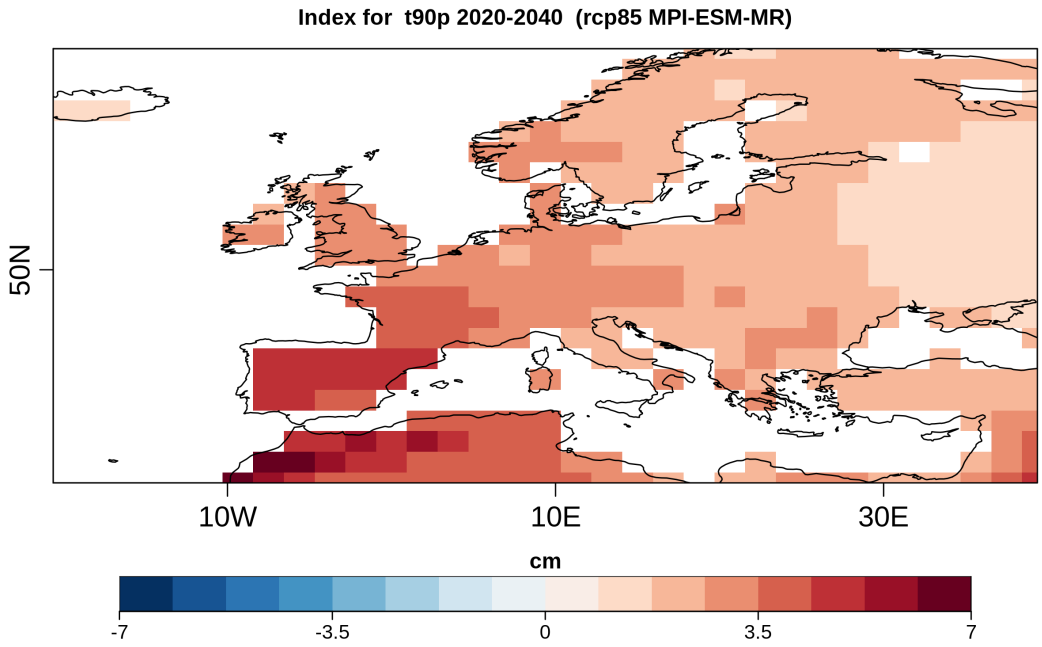


Fig. 73: Average change in the heat component (t90p metric) of the Combined Climate Extreme Index for the 2020-2040 compared to the 1971-2000 reference period for the RCP 8.5 scenario simulated by MPI-ESM-MR.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_gier20bg.yml

Diagnostics are stored in diag_scripts/

Diagnostics are stored in esmvaltool/diag_scripts/xco2_analysis/

- carbon_plots.ncl: plot script for panel plots
- delta_T.ncl: IAV of growth rate against growing season temperature - Figure C1
- global_maps.ncl: global maps for seasonal cycle amplitude - Figures 5, 6
- main.ncl: Timeseries and histogram - Figures 3, 4
- panel_plots.ncl: scatter plot of SCA/GR vs variable - Figures 7, 9, B1, B2
- sat_masks.ncl: data coverage of input data - Figures 1, 8
- stat.ncl: auxiliary functions for GR, SCA computation
- station_comparison.ncl: - comparison of surface and column data - Figure 2

User settings in recipe

1. Preprocessor
    - `conv_units`: converts units to plot-units
    - `mmm_ref`: calculates multi-model mean and regrids to ref dataset
    - `mmm_2x2`: computes multi-model mean on 2x2 grid
    - `mmm`: computes multi-model mean for 3D variable, 5x5 grid with specific pressure levels

2. Script xco2_analysis/delta_T.ncl
    - **Required diag_script_info attributes:**
        - `region`: region to average over
        - `masking`: the kind of masking to apply prior to region average (possible options: obs, land, sciamachy, gosat, none)
        - `var_order`: First main variable, then temperature variable to compare
    - **Optional diag_script_info attributes:**
        - `styleset`: styleset for color coding panels
        - `output_file_type`: output file type for plots, default: png
        - `var_plotname`: NCL string formatting how variable should be named in plots defaults to short_name if not assigned.
3. Script xco2_analysis/global_maps.ncl:
    - **Required diag_script_info attributes:**
        - `contour_max_level`: maximum value displayed for seasonal cycle amplitude contour plot
    - **Optional diag_script_info attributes:**
        - `output_file_type`: output file type for plots, default: png
4. Script xco2_analysis/main.ncl:
    - **Required diag_script_info attributes:**
        - `styleset`: styleset to use for plotting colors, linestyles...
        - `region`: latitude range for averaging
        - `masking`: different masking options are available to use on dataset: (possible options: none, obs)
        - `ensemble_mean`: if true calculates multi-model mean only accounting for the ensemble member named in "ensemble_refs"
    - **Optional diag_script_info attributes:**
        - `output_file_type`: output file type for plots, default: png
        - `ensemble_refs`: list of model-ensemble pairs to denote which ensemble member to use for calculating multi-model mean. required if ensemble_mean = true
        - `var_plotname`: String formatting how variable should be named in plots defaults to short_name if not assigned
5. Script xco2_analysis/panel_plots.ncl:
    - **Required diag_script_info attributes:**
        - `styleset`: styleset to use for plotting colors, linestyles...
        - `region`: latitude range for averaging
        - `masking`: different masking options are available to use on dataset: (possible options: obs, land, sciamachy, gosat, none)
        - `obs_in_panel`: True if observations should be included in plot
        - `area_avg`: Type of area averaging: "full-area" normal area-average "lat-first" calculate zonal means first, then average these

  - `plot_var2_mean`: If True adds mean of seasonal cycle to panel as string.
- **Optional diag_script_info attributes:**
  - `output_file_type`: output file type for plots, default: png
  - `var_plotname`: String formatting how variable should be named in plots defaults to short_name if not assigned

6. Script xco2_analysis/sat_masks.ncl:
   - **Optional diag_script_info attributes:**
     - `output_file_type`: output file type for plots, default: png
     - `var_plotname`: String formatting how variable should be named in plots defaults to short_name if not assigned
     - `c3s_plots`: Missing value plots separated by timeseries of c3s satellites
7. Script xco2_analysis/station_comparison.ncl:
   - **Required diag_script_info attributes:**
     - `var_order`: in this case xco2, co2, co2s - column averaged with obs dataset first, then 2D variable, followed by surface stations
   - **Optional diag_script_info attributes:**
     - `output_file_type`: output file type for plots, default: png
     - `var_plotnames`: String formatting how variables should be named in plots defaults to short_name if not assigned
     - `overwrite_altitudes`: Give other altitude values than the ones attached in the station data. Valid if altitude changes and timeseries spans range with different sample altitude. Caveat: If used, need to give altitude values for all stations.
     - `output_map`: boolean if stations to be displayed on map. As this requires finetuning, currently only implemented for station set of (ASK, CGO, HUN, LEF, WIS) following the paper. Change for different plot inset locations, if others are desired.

Variables

- *xco2* (atmos, monthly, longitude, latitude, time)
- *co2s* (atmos, monthly, longitude, latitude, time)
- *co2* (atmos, monthly, pressure, longitude, latitude, time)
- *tas* (atmos, monthly, longitude, latitude, time)
- *tasa* (atmos, monthly, longitude, latitude, time)

Observations and reformat scripts

- CDS-XCO2 (*xco2*)
- ESRL (*co2s*)
- GISTEMP (*tasa*)
- MODIS (land cover map, auxiliary data folder)

References

- Gier, B. K., Buchwitz, M., Reuter, M., Cox, P. M., Friedlingstein, P., and Eyring, V.: Spatially resolved evaluation of Earth system models with satellite column-averaged CO2, Biogeosciences, 17, 6115–6144, https://doi.org/10.5194/bg-17-6115-2020, 2020.

Example plots

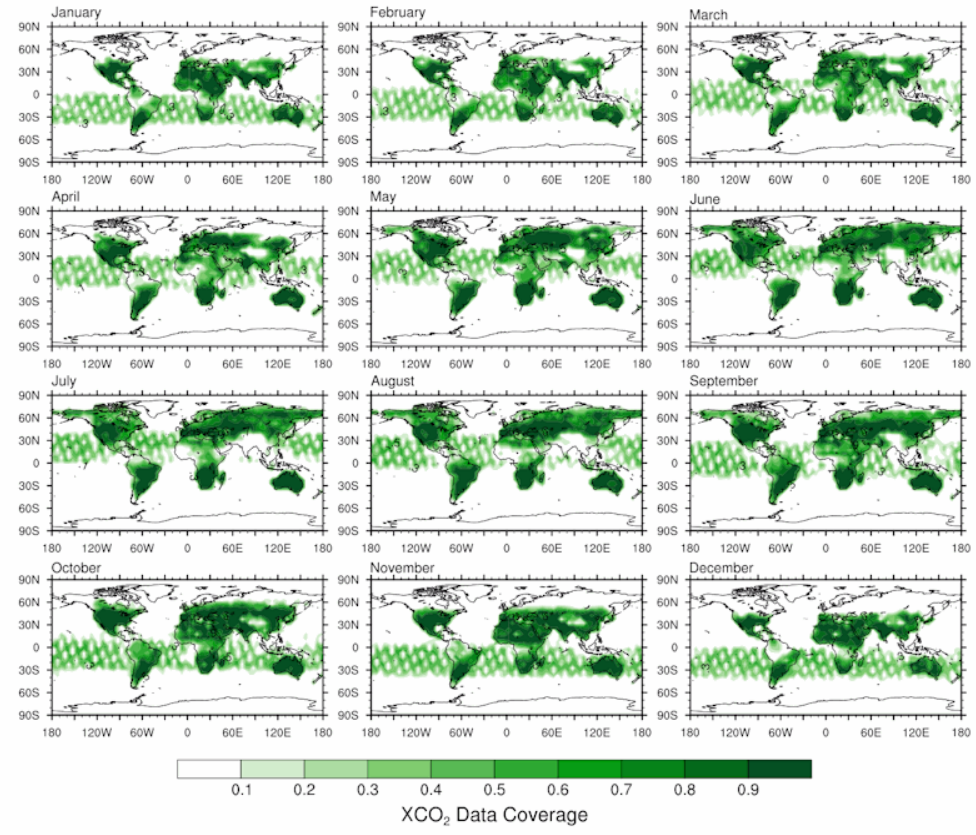


Fig. 74: Mean fractional coverage of monthly satellite data.

5.2.13 Heat wave and cold wave duration

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.

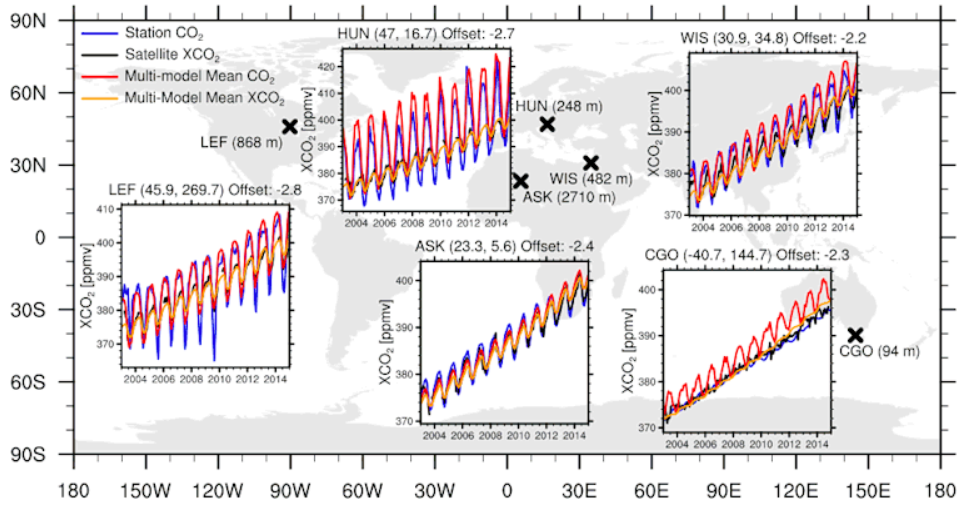


Fig. 75: Comparison of time series from satellite, in situ, and models sampled accordingly. Caveat: inset plot positions are hardcoded.

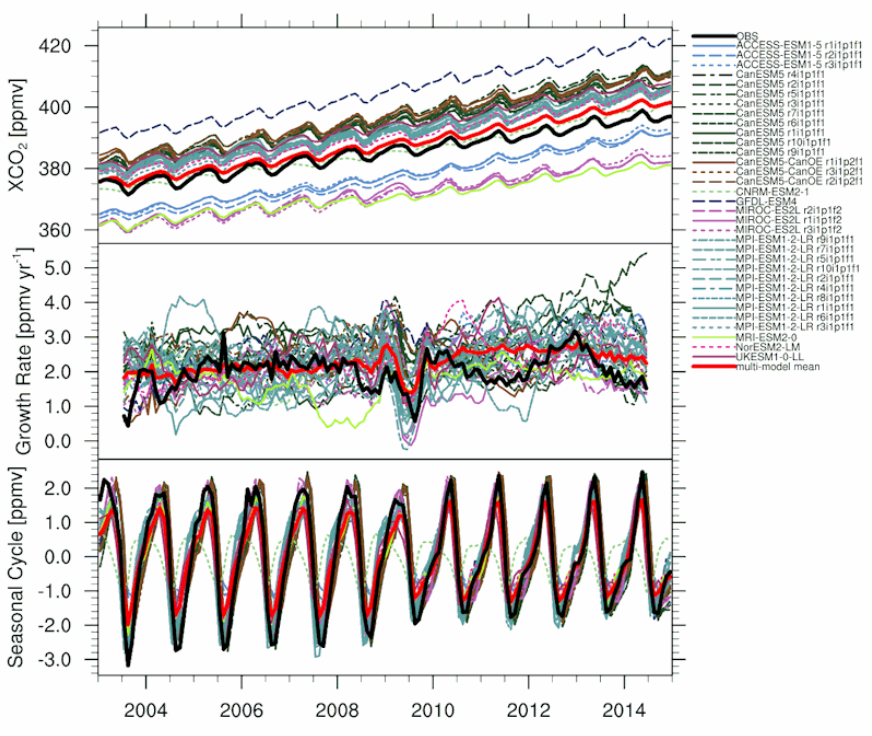


Fig. 76: Timeseries with panels depicting growth rate and seasonal cycle.

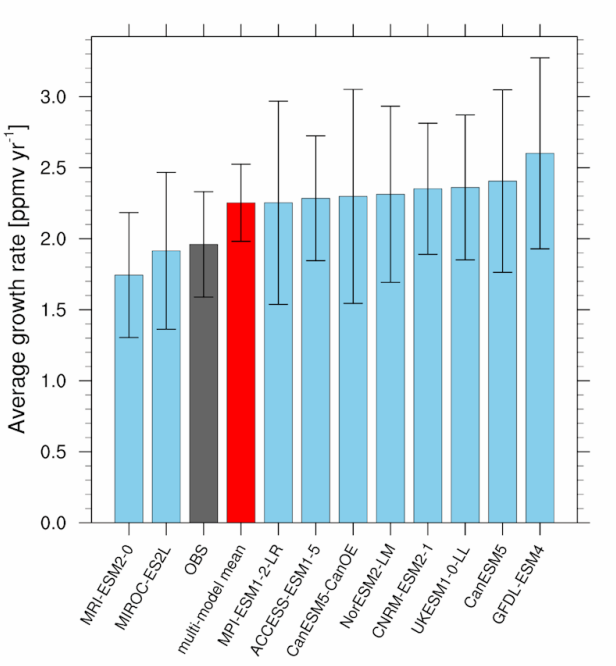


Fig. 77: Barplot of the growth rate, averaged over all years, with standard deviation of interannual variability.

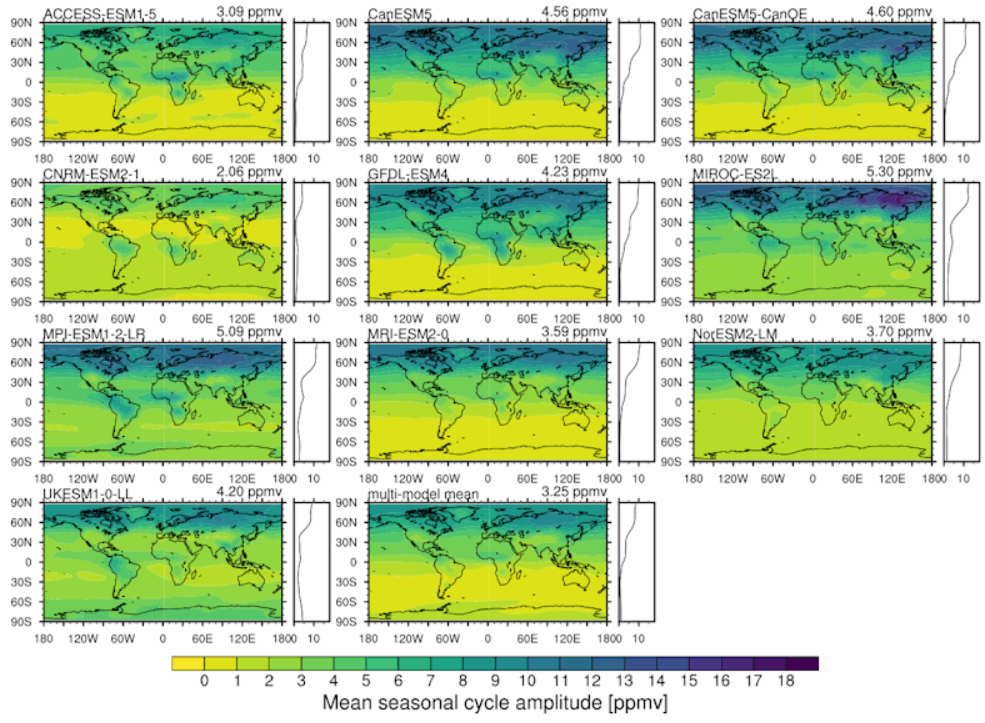


Fig. 78: Panel plot of spatially resolved seasonal cycle amplitude for all models, including a zonal average sidepanel.

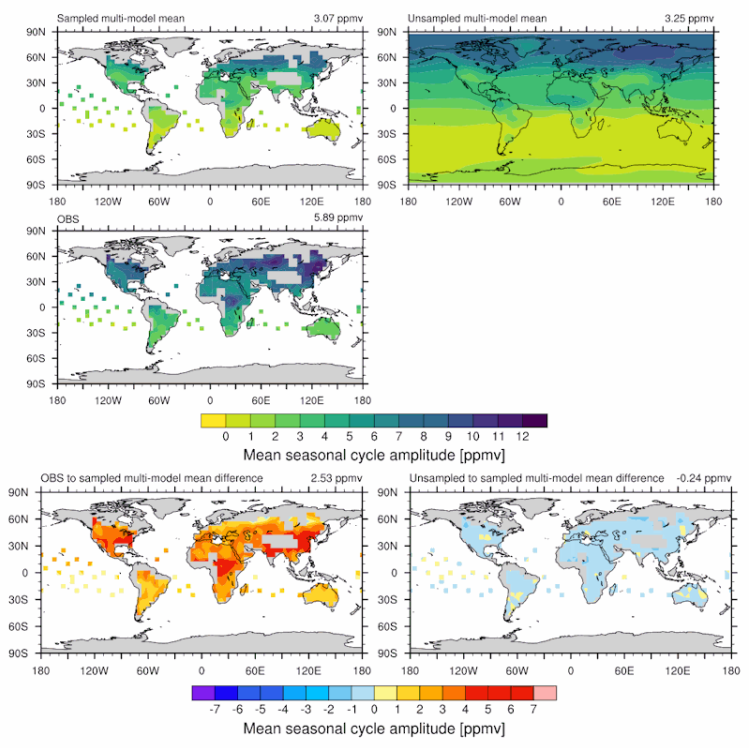


Fig. 79: Seasonal cycle amplitude map comparing influence of sampling, and difference to observations.

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.

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

Variables

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

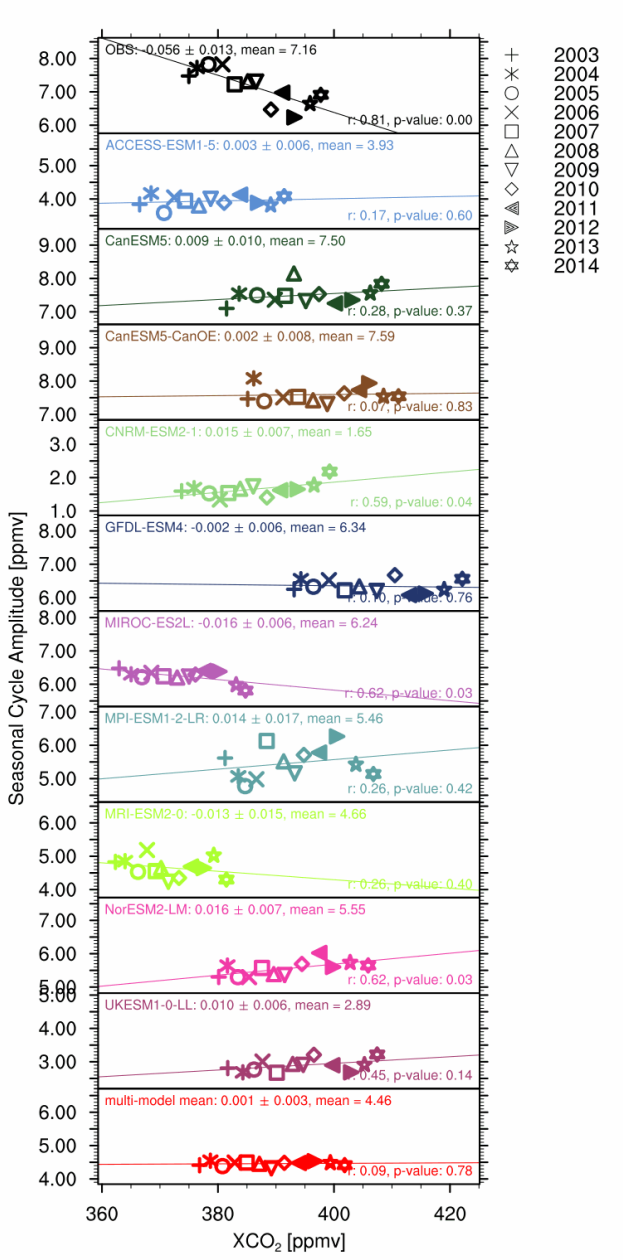


Fig. 80: Panel plots showing seasonal cycle amplitude against XCO_2, includes regression line and p-value.

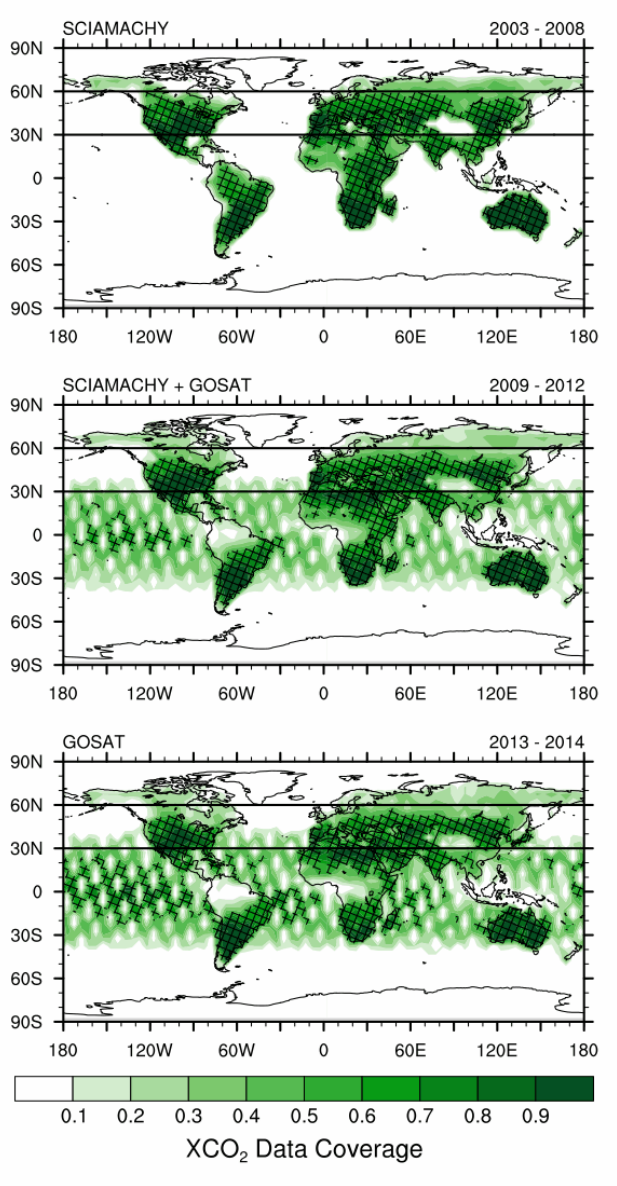


Fig. 81: Mean spatial data coverage for different satellites.

Observations and reformat scripts

None

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

Example plots

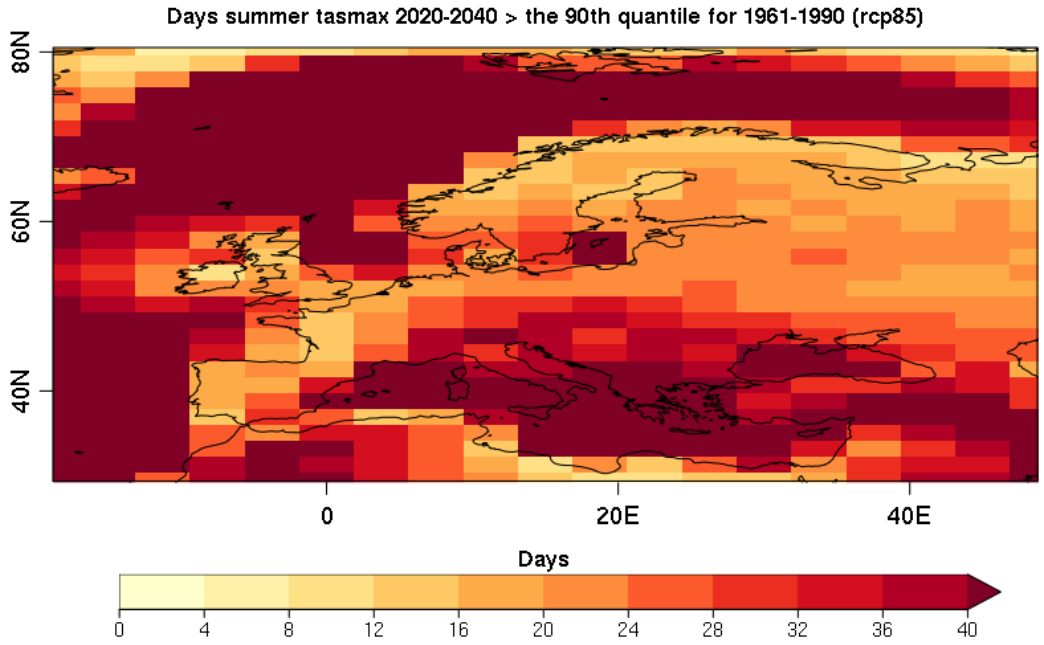


Fig. 82: Mean number of summer days during the period 2060-2080 when the daily maximum near-surface air temperature exceeds the 80th quantile of the 1971-2000 reference period. The results are based on one RCP 8.5 scenario simulated by BCC-CSM1-1.

5.2.14 Hydroclimatic intensity and extremes (HyInt)

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.

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.

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)

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

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)

Observations and reformat scripts

None.

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

Example plots

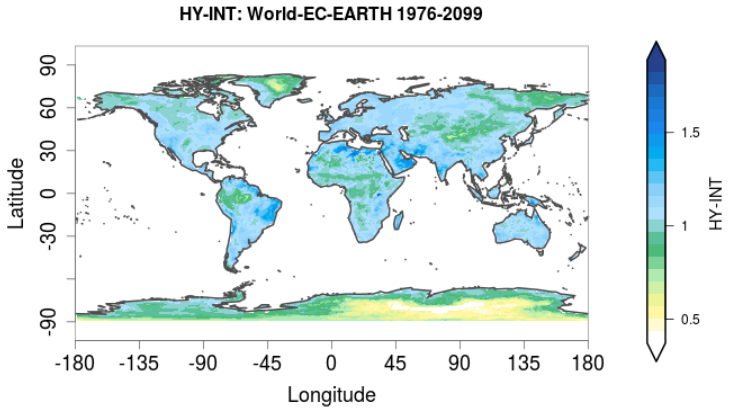


Fig. 83: Mean hydroclimatic intensity for the EC-EARTH model, for the historical + RCP8.5 projection in the period 1976-2099

5.2.15 Implied heat transport from Top of Atmosphere fluxes

Overview

This recipe calculates the implied horizontal heat transport (IHT) due to the spatial anomalies of radiative fluxes at the top of the atmosphere (TOA). The regional patterns of implied heat transport for different components of the TOA fluxes are calculated by solving the Poisson equation with the flux components as source terms. It reproduces the plots in Pearce and Bodas-Salcedo (2023) when the input data is CERES EBAF.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_iht_toa.yml calculates the IHT maps for the following radiative fluxes:
  - Total net, SW net, LW net (Figure 2).
  - Total CRE, SW CRE, LW CRE (Figure 4).
  - All-sky and clear-sky reflected SW (Figure 5).
  - The meridional heat transports (MHT) of the fluxes above (Figures 1 and 3).

Diagnostics are stored in esmvaltool/diag_scripts/iht_toa/

- single_model_diagnostics.py: driver script that produces the plots.
- poisson_solver.py: solver that calculates the IHTs.

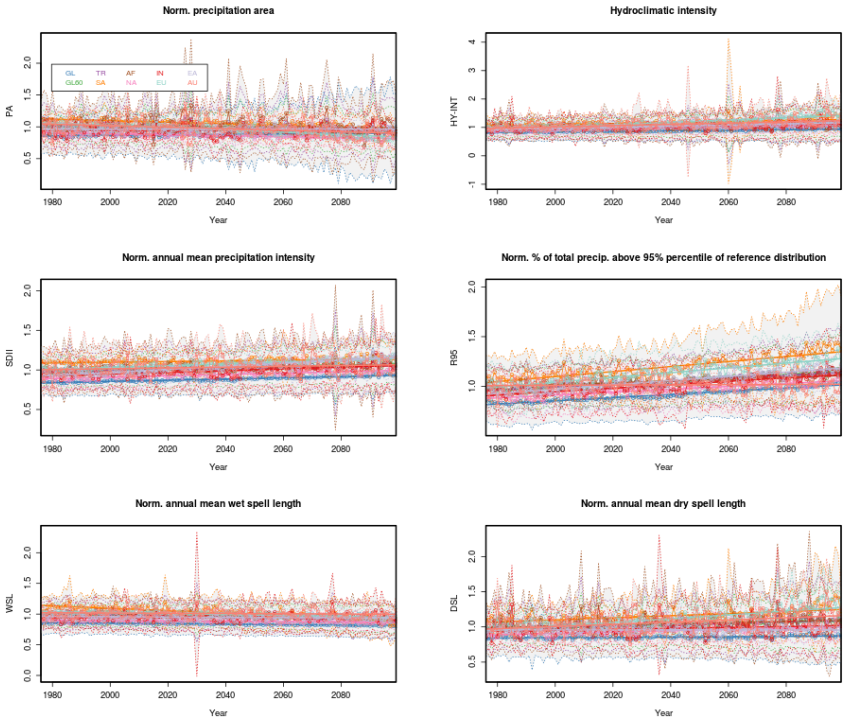


Fig. 84: Timeseries for multiple indices and regions for the ACCESS1-0 model, for the historical + RCP8.5 projection in the period 1976-2099, normalized to the 1976-2005 historical period.

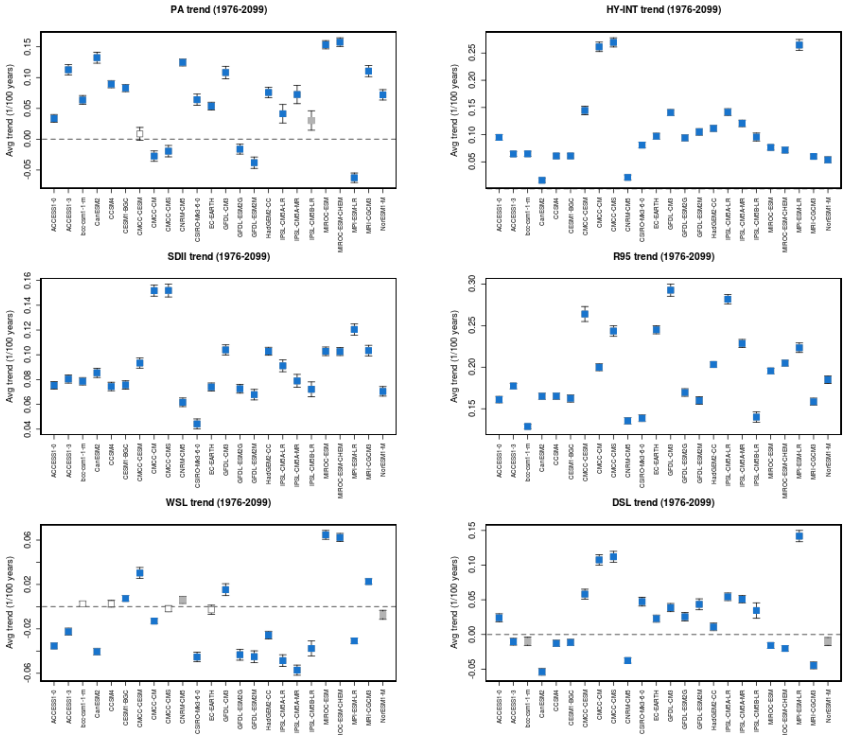


Fig. 85: Multi-model trend coefficients over selected indices for CMIP5 models in the RCP8.5 2006-2099 projection, normalized to the 1976-2005 historical period.

User settings in recipe

There are no user settings in this recipe.

Variables

- rlut (atmos, monthly, longitude latitude time)
- rlutcs (atmos, monthly, longitude latitude time)
- rsutcs (atmos, monthly, longitude latitude time)
- rsut (atmos, monthly, longitude latitude time)
- rsdt (atmos, monthly, longitude latitude time)

Observations and reformat scripts

- CERES-EBAF

References

- Pearce, F. A., and A. Bodas-Salcedo, 2023: Implied Heat Transport from CERES Data: Direct Radiative Effect of Clouds on Regional Patterns and Hemispheric Symmetry. J. Climate, 36, 4019–4030, doi: 10.1175/JCLI-D-22-0149.1.

Example plots

5.2.16 Quick insights for climate impact researchers

Overview

Many impact researchers do not have the time and finances to use a large ensemble of climate model runs for their impact analysis. To get an idea of the range of impacts of climate change it also suffices to use a small number of climate model runs. In case a system is only sensitive to annual temperature, one can select a run with a high change and one with a low change of annual temperature, preferably both with a low bias.

This recipe calculates the bias with respect to observations, and the change with respect to a reference period, for a wide range of (CMIP) models. These metrics are tabulated and also visualized in a diagram.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_impact.yml

Diagnostics are stored in esmvaltool/diag_scripts/

- impact/bias_and_change.py: tabulate and visualize bias and change.

User settings in recipe

1. Script `impact.py`

   Required settings for variables

   - tag: `'model'` or `'observations'`, so the diagnostic script knows which datasets to use for the bias calculation. This must be specified for each dataset.

   Optional settings for preprocessor

   - Region and time settings (both for the future and reference period) can be changed at will.

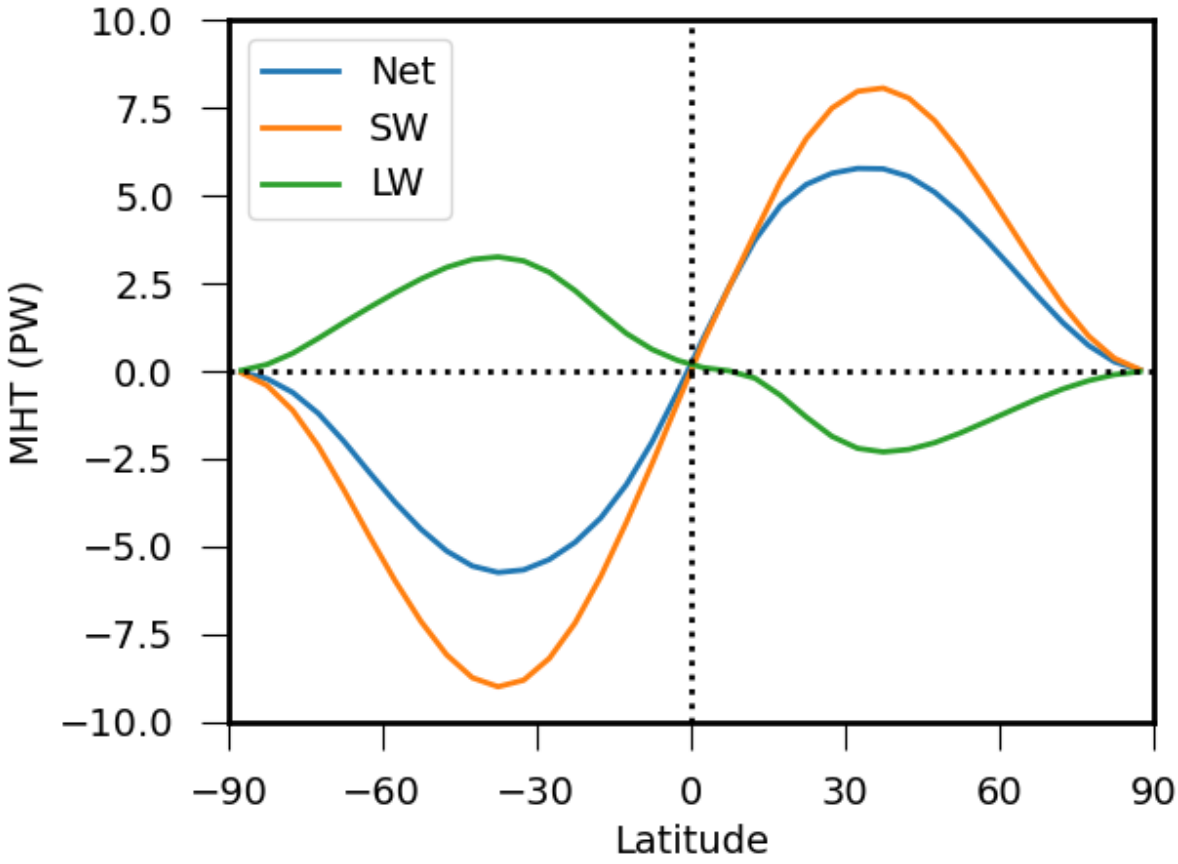


Fig. 86: The implied heat transport due to the total net flux (blue), split into the contributions from the SW (orange) and LW (green).

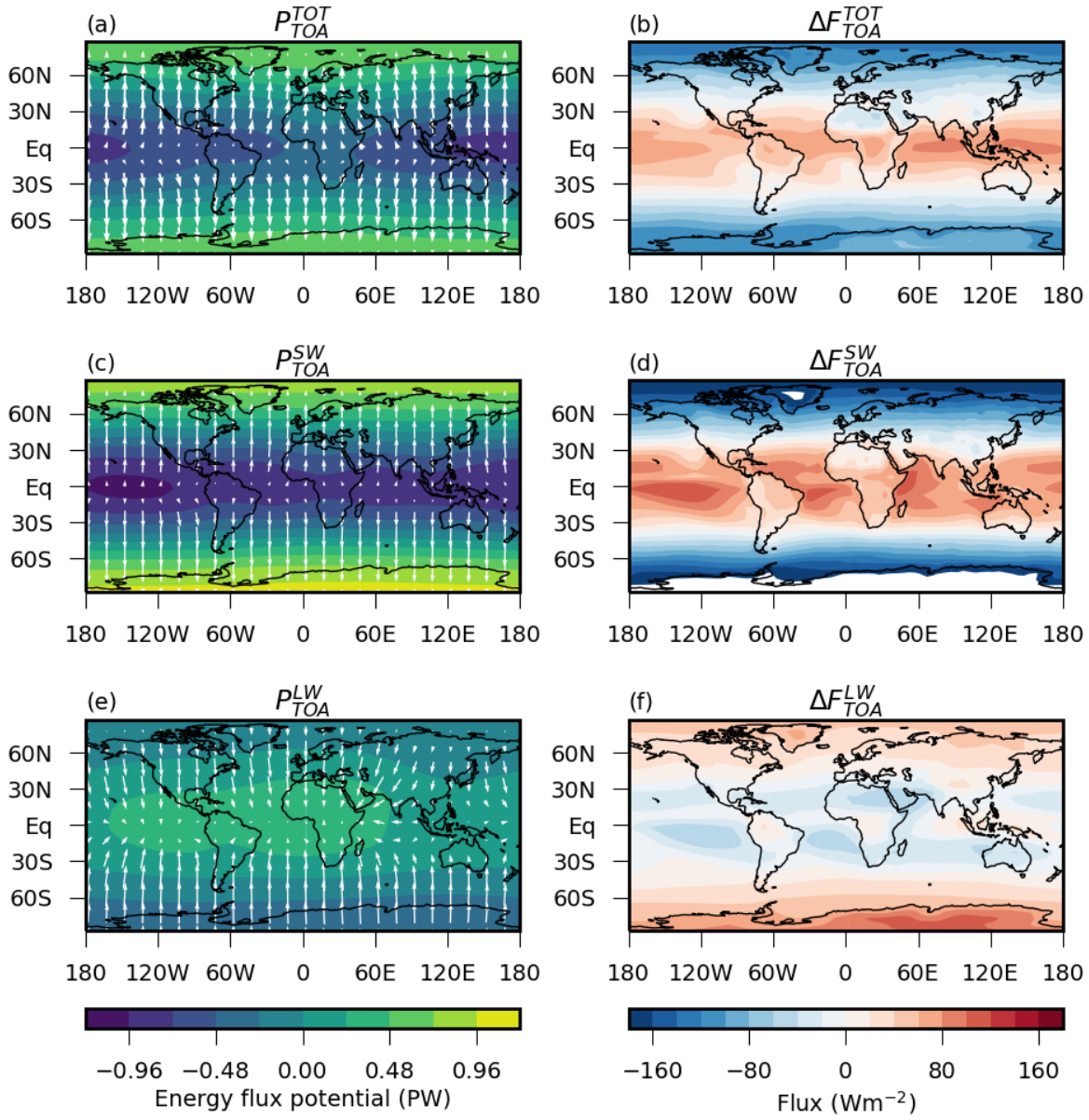


Fig. 87: The energy flux potentials for (a) TOT, (c) SW, and (e) LW fluxes, alongside maps of the spatial anomalies of the fluxes [(b),(d),(f) flux minus global average flux, respectively]. The implied heat transport is calculated as the gradient of the energy flux potential, shown by the white vector arrows for a subset of points to give the overall transport pattern. Heat is directed from the blue minima of the potential field to yellow maxima, with the magnitude implied by the density of contours. All maps of the same type share the same color bar at the bottom of the column.

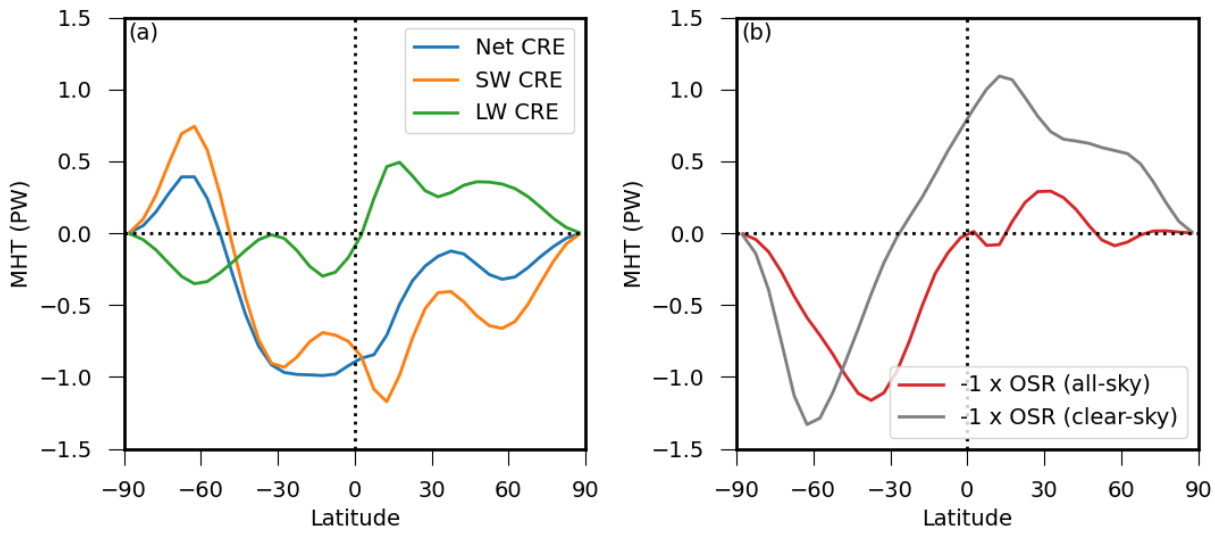


Fig. 88: Direct radiative effects of clouds on the meridional heat transport. (a) Contributions from TOT CRE (blue), SW CRE (orange), and LW CRE (green) fluxes. (b) Contributions from all-sky and clear-sky OSR. In (b), both curves have been multiplied by 1 such that positive heat transport is northward.

Variables

- tas (atmos, mon, longitude latitude time)
- pr (atmos, mon, longitude latitude time)
- any other variables of interest

Observations and reformat scripts

- ERA5 data can be used via the native6 project.

References

- None

Example plots

5.2.17 Lifetime

Overview

The diagnostic calculates the lifetime of CH4 (or potentially other chemically active trace gases) and is able to plot it in several different representations.

Available recipes and diagnostics

Recipes are stored in *recipes/lifetime*

- recipe_lifetime.yml

Diagnostics are stored in *diag_scripts/lifetime/*

- *lifetime.py*

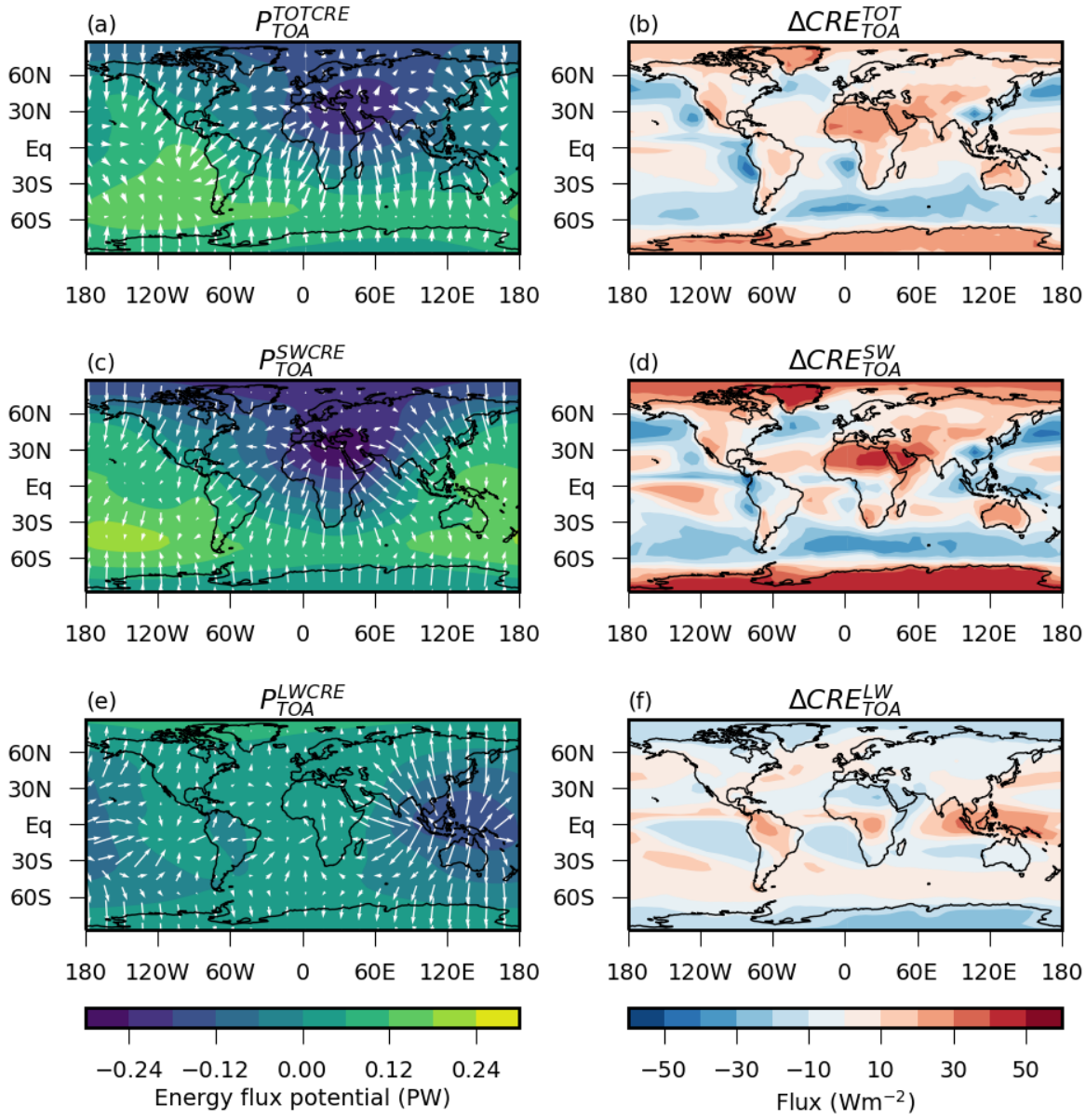


Fig. 89: As in Fig. 87, but for the implied heat transport associated with (a),(b) TOT CRE, (c),(d) SW CRE, and (e),(f) LW CRE fluxes.

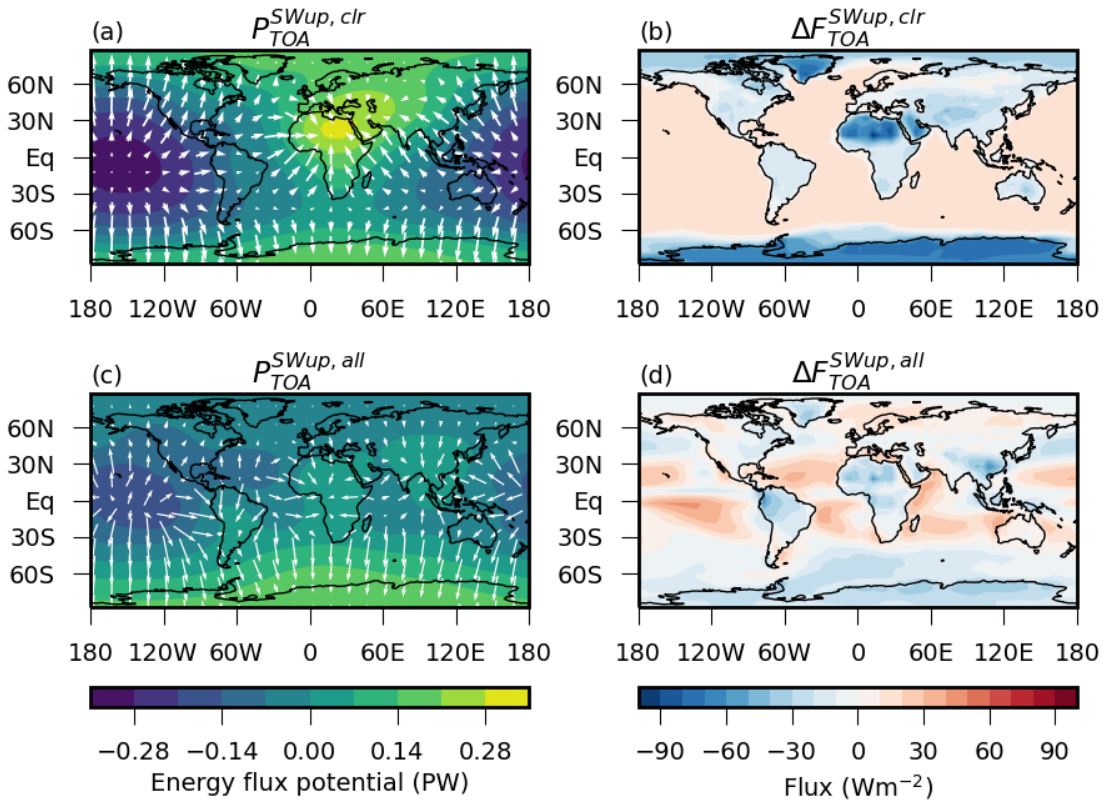


Fig. 90: As in Fig. 87, but for (a), (b) clear-sky and (c), (d) all-sky reflected SW flux.

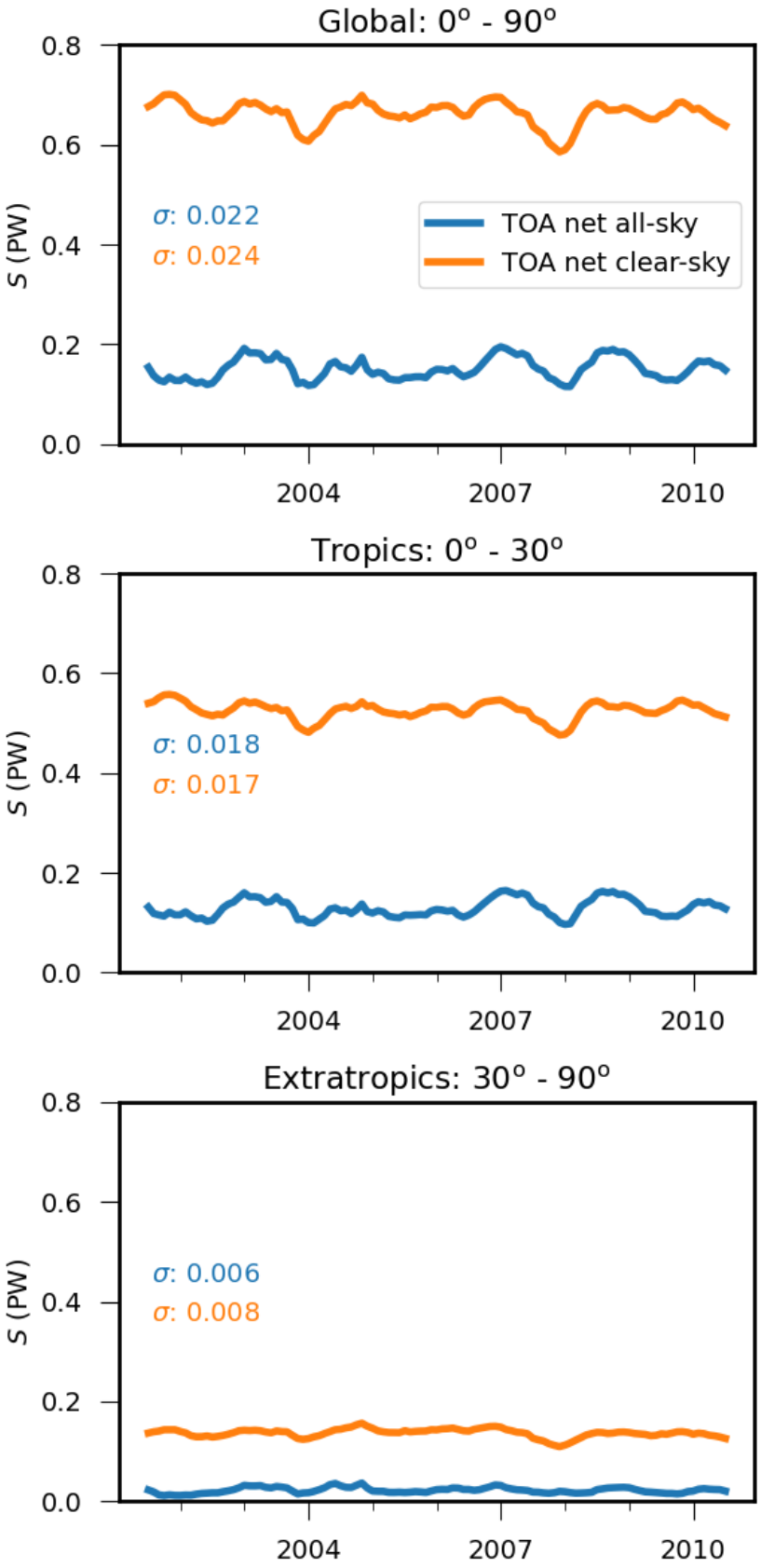


Fig. 91: A measure of the symmetry between heat transport in the Northern and Southern Hemispheres, calculated for the 12-month running mean of TOT MHT in the regions: (a) the full hemisphere, (b) from the equator to 30°, and (c) 30° to 90°. Symmetry values obtained when including (blue) and excluding (orange) the effect of clouds. The climatological symmetry values for the two cases are shown as the black lines in each subplot, dashed and dotted, respectively. The standard deviations of the time series are shown in each plot.

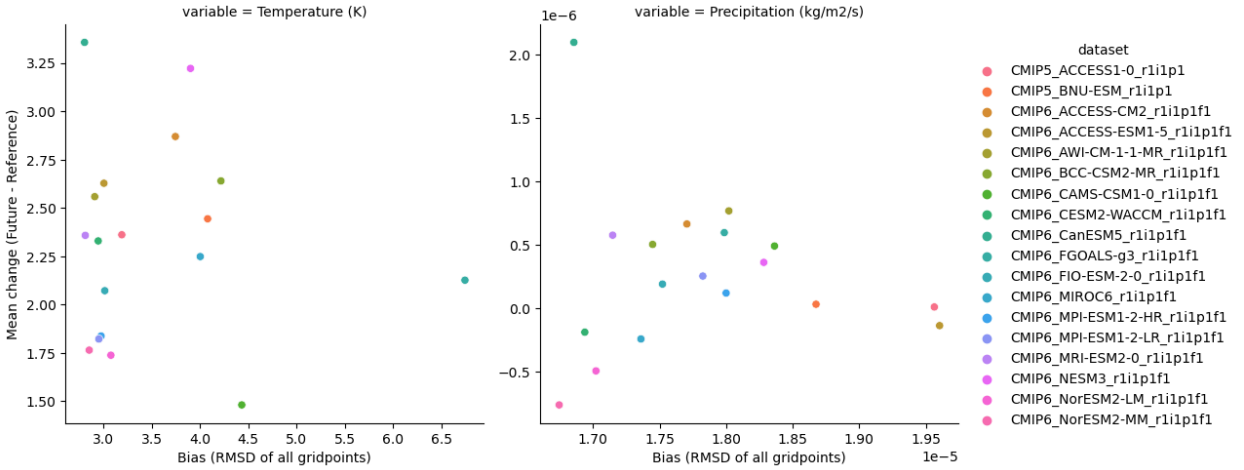


Fig. 92: Bias and change for each variable

Recipe settings

A list of all possible configuration options that can be specified in the recipe is given for each diagnostic individually (see previous section).

Example plots

Time series of CH4 lifetime in the troposphere.

5.2.18 Modes of variability

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.

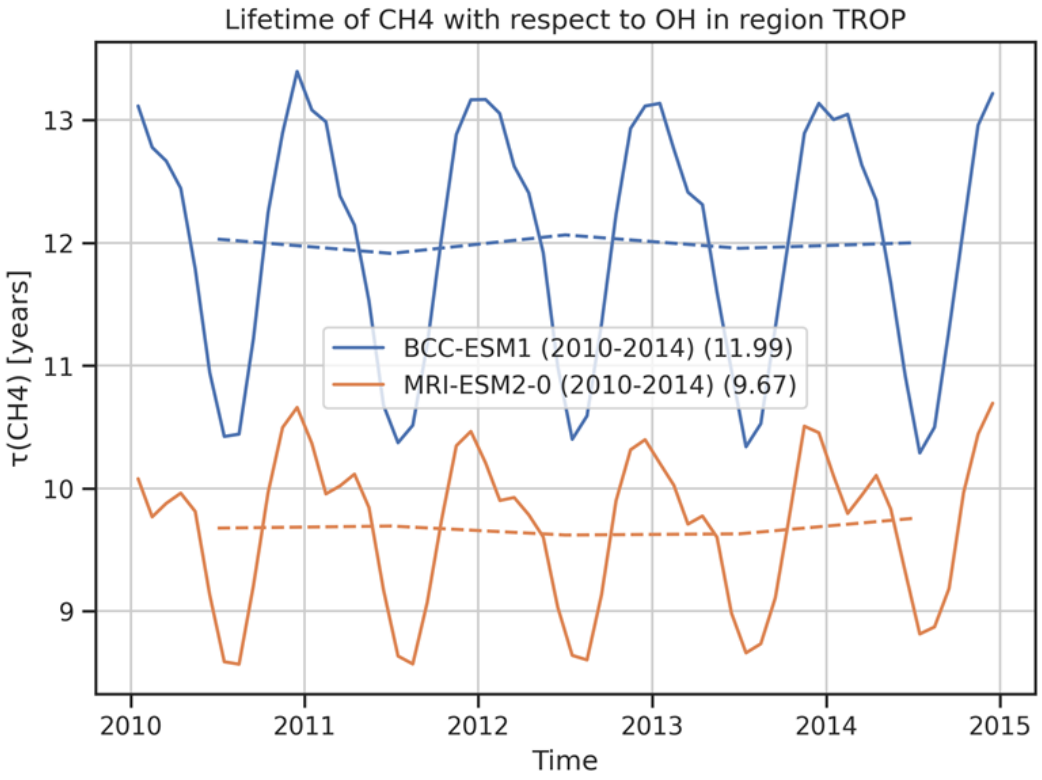
Available recipes and diagnostics

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



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).

Variables

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

Observations and reformat scripts

None

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.

Example plots

5.2.19 Diagnostics of integrated atmospheric methane (XCH4)

Overview

This recipe `recipe_mpqb_xch4.yml` allows the comparison of integrated atmospheric methane between CMIP6 model simulations and observations, and produces lineplots of monthly mean methane values, annual cycles and annual growth rates:

- Monthly mean time series of XCH4 for pre-defined regions (global, Northern Hemisphere, Southern Hemisphere)
- Annual cycles of XCH4 for pre-defined regions (global, Northern Hemisphere, Southern Hemisphere)
- Annual growth rates of XCH4 for pre-defined regions (global, Northern Hemisphere, Southern Hemisphere)

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/mpqb/

- recipe_mpqb_xch4.yml

Diagnostics are stored in esmvaltool/diag_scripts/mpqb/

- mpqb_lineplot.py
- mpqb_lineplot_anncyc.py
- mpqb_lineplot_growthrate.py

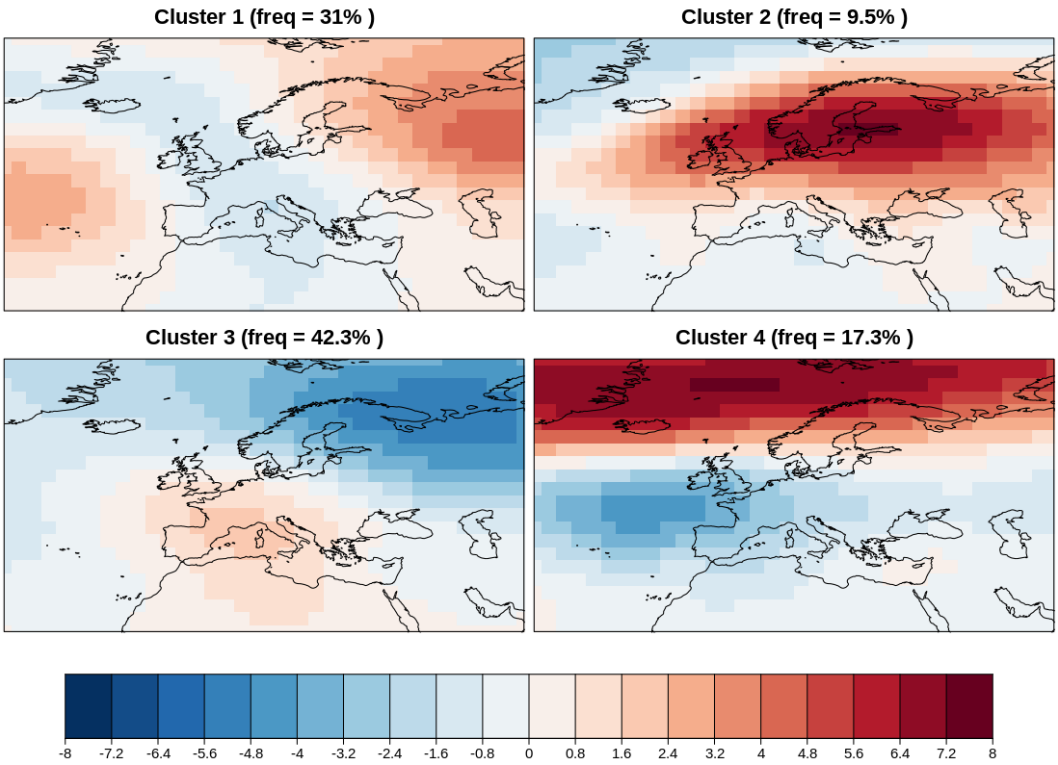


Fig. 93: Four modes of variability for autumn (September-October-November) in the North Atlantic European Sector for the RCP 8.5 scenario using BCC-CSM1-1 future projection during the period 2020-2075. The frequency of occurrence of each variability mode is indicated in the title of each map.

Observations and reformat scripts

Observations used in this recipe are:

- CDS-XCH4 (ESA CCI dataset served on the Copernicus Climate data store)

A cmorizing script for this dataset is available (`cmorize_obs_cds_xch4.ncl`).

XCH4 is a derived variable that needs to be calculated from four different variables (ch4, hus, zg, ps). A derivation script is included in the ESMValCore.

User settings in recipe

1. Preprocessor

   - `pp_lineplots_xx_mon`: Regridding, masking all missing values from all used datasets, area-mean ('xx' can ge replaced by 'gl'=global, 'sh'=southern hemisphere, 'nh'=northern hemisphere), units converted to [ppbv] to obtain one time series of monthly mean values for the selected region (global, southern hemisphere, northern hemisphere)
   - `pp_lineplots_xx_ann`: Regridding, masking all missing values from all used datasets, area-mean ('xx' can ge replaced by 'gl'=global, 'sh'=southern hemisphere, 'nh'=northern hemisphere), units converted to [ppbv] to obtain one time series of annual mean values for the selected region (global, southern hemisphere, northern hemisphere)
   - `pp_lineplots_anncyc_xx:` : Regridding, masking all missing values from all used datasets, area-mean ('xx' can ge replaced by 'gl'=global, 'sh'=southern hemisphere, 'nh'=northern hemisphere), units converted to [ppbv], monthly climate statistics applied to one annual cycle for the whole chosen time period and for the selected region (global, southern hemisphere, northern hemisphere)
   - `xch4_def_xx`: defining the time period over which the analysis should be calculated; options are "cmip6" which overlapping period of the observations and the CMIP6 historical simulations, and "future" which covers the time period of CMIP6 scenarios

2. Additional needed files

   - `mpqb_cfg_xch4.yml`: In this file additional information for the used datasets are defined and stored, e.g. alias of the dataset name and the color that is used to display the dataset in the figures
   - `mpqb_utils.yml`: In this file the preparations for the dataset displays are made.

3. Script <mpqb_lineplot.py>

   Required settings for script

   - no additional settings required

   Optional settings for script

   - no optional settings available

   Required settings for variables

   - no settings for the variable required

4. Script <mpqb_lineplot_anncyc.py>

   Required settings for script

   - no additional settings required

   Optional settings for script

   - no optional settings available

   Required settings for variables

- no settings for the variable required

5. Script <mpqb_lineplot_growthrate.py>

   Required settings for script

   - no additional settings required

   Optional settings for script

   - no optional settings available

   Required settings for variables

   - no settings for the variable required

Variables

- ch4 (atmos, monthly mean, longitude latitude level time)
- hus (atmos, monthly mean, longitude latitude level time)
- zg (atmos, monthly mean, longitude latitude level time)
- ps (atmos, monthly mean, longitude latitude time)

All variables are necessary to calculate the derived variable xch4.

Example plots

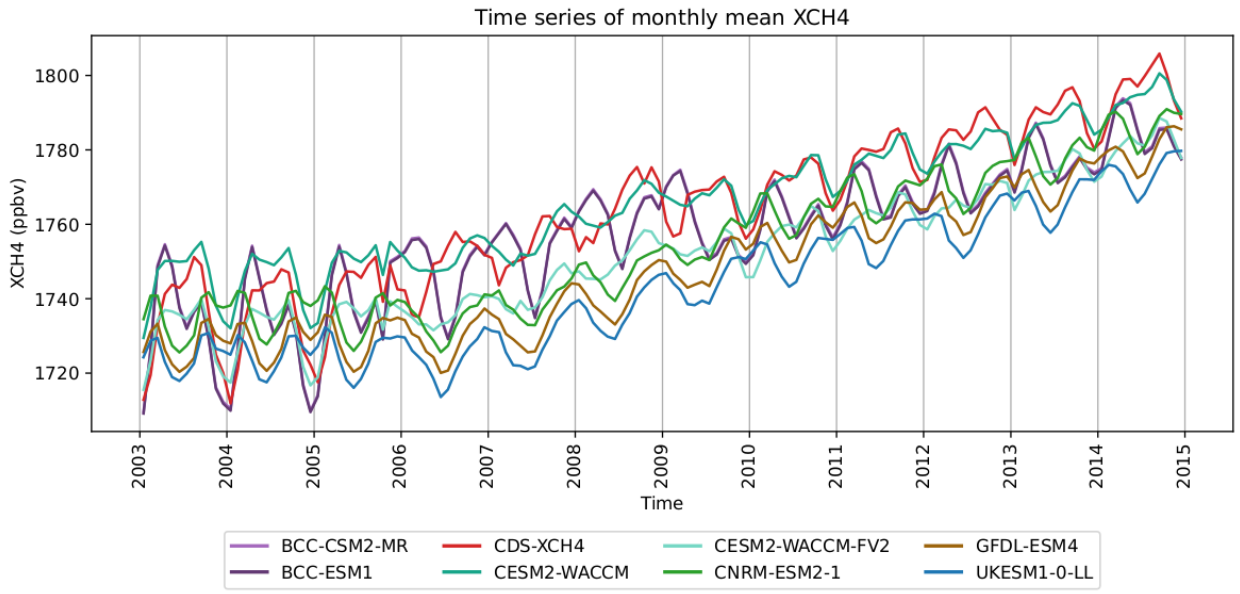


Fig. 94: Monthly mean time series of XCH4, calculated over the whole globe, for individual CMIP6 model simulations.

5.2.20 Precipitation quantile bias

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 specified 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.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_quantilebias.yml

Diagnostics are stored in diag_scripts/quantilebias/

- quantilebias.R

User settings

Required settings for script

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

Variables

- pr (atmos, monthly, longitude latitude time)

Observations and reformat scripts

- GPCP-SG observations (accessible via the obs4MIPs project)

References

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

Example plots

5.2.21 Quantifying progress across different CMIP phases

Overview

The recipe recipe_bock20jgr.yml generates figures to quantify the progress across different CMIP phases.

Note

The current recipe uses a horizontal 5x5 grid for figure 10, while the original plot in the paper shows a 2x2 grid. This is solely done for computational reasons (running the recipe with a 2x2 grid for figure 10 takes considerably more time than running it with a 5x5 grid) and can be easily changed in the preprocessor section of the recipe if necessary.

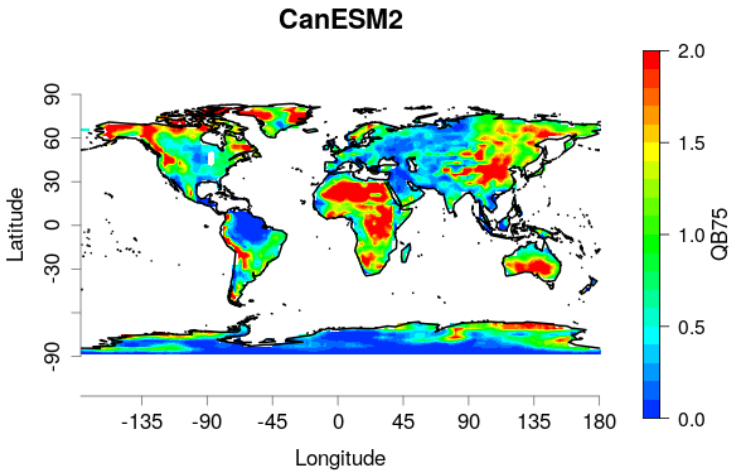


Fig. 95: Quantile bias, as defined in Mehran et al. 2014, with threshold t=75th percentile, evaluated for the CanESM2 model over the 1979-2005 period, adopting GPCP-SG v 2.3 gridded precipitation as a reference dataset. The optimal reference value is 1. Both datasets have been regridded onto a 2° regular grid.

Available recipes and diagnostics

Recipes are stored in recipes/bock20jgr

- recipe_bock20jgr_fig_1-4.yml
- recipe_bock20jgr_fig_6-7.yml
- recipe_bock20jgr_fig_8-10.yml

Diagnostics are stored in diag_scripts/

Fig. 1:

- bock20jgr/tsline.ncl: timeseries of global mean surface temperature anomalies

Fig. 2:

- bock20jgr/tsline_collect.ncl: collect different timeseries from tsline.ncl to compare different models ensembles

Fig. 3 and 4:

- bock20jgr/model_bias.ncl: global maps of the multi-model mean and the multi-model mean bias

Fig. 6:

- perfmetrics/main.ncl
- perfmetrics/collect.ncl

Fig. 7:

- bock20jgr/corr_pattern.ncl: calculate pattern correlation
- bock20jgr/corr_pattern_collect.ncl: create pattern correlation plot

Fig. 8:

- climate_metrics/ecs.py
- climate_metrics/create_barplot.py

Fig. 9:

- clouds/clouds_ipcc.ncl

Fig. 10:

- climate_metrics/feedback_parameters.py

User settings in recipe

1. Script tsline.ncl

   Required settings (scripts)

   - styleset: as in diag_scripts/shared/plot/style.ncl functions

   Optional settings (scripts)

   - time_avg: type of time average (currently only "yearly" and "monthly" are available).
   - ts_anomaly: calculates anomalies with respect to the defined reference period; for each grid 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
   - header: if true, use region name as header
   - write_stat: if true, write multi-model statistics to nc-file

   Required settings (variables)

   none

   - Optional settings (variables)

   none

2. Script tsline_collect.ncl

   Required settings (scripts)

   - styleset: as in diag_scripts/shared/plot/style.ncl functions

   Optional settings (scripts)

   - time_avg: type of time average (currently only "yearly" and "monthly" are available).
   - ts_anomaly: calculates anomalies with respect to the defined period
   - ref_start: start year of reference period for anomalies
   - ref_end: end year of reference period for anomalies

- region: name of domain
- plot_units: variable unit for plotting
- y_min: set min of y-axis
- y_max: set max of y-axis
- order: order in which experiments should be plotted
- header: if true, region name as header
- stat_shading: if true: shading of statistic range
- ref_shading: if true: shading of reference period

Required settings (variables)

none

- Optional settings (variables)

none

3. Script model_bias.ncl

   Required settings (scripts)

   none

   Optional settings (scripts)

   - 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 dataset

   Optional settings (variables)

   - long_name: description of variable

   Color tables

   - variable "tas": diag_scripts/shared/plot/rgb/ipcc-ar6_temperature_div.rgb,
   - variable "pr-mmday": diag_scripts/shared/plots/rgb/ipcc-ar6_precipitation_seq.rgb diag_scripts/shared/plot/rgb/ipcc-ar6_precipitation_div.rgb

4. Script perfmetrics_main.ncl

   See *here*.

5. Script perfmetrics_collect.ncl

   See *here*.

6. Script corr_pattern.ncl

   Required settings (scripts)

   none

   Optional settings (scripts)

   - plot_median

   Required settings (variables)

- reference_dataset

Optional settings (variables)

- alternative_dataset

7. Script corr_pattern_collect.ncl

   Required settings (scripts)

   none

   Optional settings (scripts)

   - diag_order

   Color tables

   - diag_scripts/shared/plot/rgb/ipcc-ar6_line_03.rgb

8. Script ecs.py

   See *here*.

9. Script create_barplot.py

   See *here*.

10. Script clouds_ipcc.ncl

    See *here*.

11. Script feedback_parameters.py

    Required settings (scripts)

    none

    Optional settings (scripts)

    - calculate_mmm: *bool* (default: `True`). Calculate multi-model means.
    - only_consider_mmm: *bool* (default: `False`). Only consider multi-model mean dataset. This automatically sets `calculate_mmm` to `True`. For large multi-dimensional datasets, this might significantly reduce the computation time if only the multi-model mean dataset is relevant.
    - output_attributes: *dict*. Write additional attributes to netcdf files.
    - seaborn_settings: *dict*. Options for `seaborn.set_theme()` (affects all plots).

Variables

- clt (atmos, monthly, longitude latitude time)
- hus (atmos, monthly, longitude latitude lev time)
- pr (atmos, monthly, longitude latitude time)
- psl (atmos, monthly, longitude latitude time)
- rlut (atmos, monthly, longitude latitude time)
- rsdt (atmos, monthly, longitude latitude time)
- rsut (atmos, monthly, longitude latitude time)
- rtmt (atmos, monthly, longitude latitude time)
- rlutcs (atmos, monthly, longitude latitude time)

- rsutcs (atmos, monthly, longitude latitude time)
- ta (atmos, monthly, longitude latitude lev time)
- tas (atmos, monthly, longitude latitude time)
- ts (atmos, monthly, longitude latitude time)
- ua (atmos, monthly, longitude latitude lev time)
- va (atmos, monthly, longitude latitude lev time)
- zg (atmos, monthly, longitude latitude time)

Observations and reformat scripts

- AIRS (obs4MIPs) - specific humidity
- CERES-EBAF (obs4MIPs) - CERES TOA radiation fluxes (used for calculation of cloud forcing)
- ERA-Interim - reanalysis of surface temperature, sea surface pressure

  Reformat script: recipes/cmorizers/recipe_era5.yml
- ERA5 - reanalysis of surface temperature

  Reformat script: recipes/cmorizers/recipe_era5.yml
- ESACCI-CLOUD - total cloud cover

  Reformat script: cmorizers/data/formatters/datasets/esacci_cloud.ncl
- ESACCI-SST - sea surface temperature

  Reformat script: cmorizers/data/formatters/datasets/esacci_sst.py
- GHCN - Global Historical Climatology Network-Monthly gridded land precipitation

  Reformat script: cmorizers/data/formatters/datasets/ghcn.ncl
- GPCP-SG (obs4MIPs) - Global Precipitation Climatology Project total precipitation
- HadCRUT4 - surface temperature anomalies

  Reformat script: cmorizers/data/formatters/datasets/hadcrut4.ncl
- HadISST - surface temperature

  Reformat script: cmorizers/data/formatters/datasets/hadisst.ncl
- JRA-55 (ana4mips) - reanalysis of sea surface pressure
- NCEP-NCAR-R1 - reanalysis of surface temperature

  Reformat script: cmorizers/data/formatters/datasets/ncep_ncar_r1.py
- PATMOS-x - total cloud cover

  Reformat script: cmorizers/data/formatters/datasets/patmos_x.ncl

References

- Bock, L., Lauer, A., Schlund, M., Barreiro, M., Bellouin, N., Jones, C., Predoi, V., Meehl, G., Roberts, M., and Eyring, V.: Quantifying progress across different CMIP phases with the ESMValTool, Journal of Geophysical Research: Atmospheres, 125, e2019JD032321. https://doi.org/10.1029/2019JD032321
- Copernicus Climate Change Service (C3S), 2017: ERA5: Fifth generation of ECMWF atmospheric reanalyses of the global climate, edited, Copernicus Climate Change Service Climate Data Store (CDS). https://doi.org/10.24381/cds.143582cf

- 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.
- Morice, C. P., Kennedy, J. J., Rayner, N. A., & Jones, P., 2012: Quantifying uncertainties in global and regional temperature change using an ensemble of observational estimates: The HadCRUT4 data set, Journal of Geophysical Research, 117, D08101. https://doi.org/10.1029/2011JD017187

Example plots

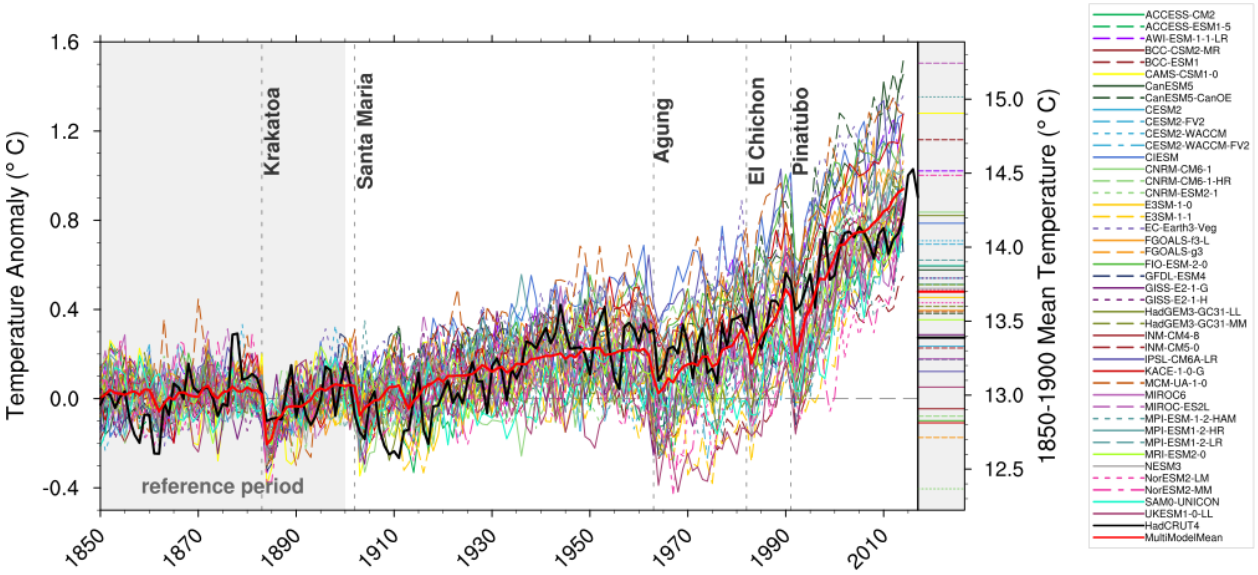


Fig. 96: Observed and simulated time series of the anomalies in annual and global mean surface temperature. All anomalies are differences from the 1850-1900 time mean of each individual time series (Fig. 1).

5.2.22 Stratosphere - Autoassess diagnostics

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.

Performance metrics:

- Polar night jet: northern hem (January) vs. ERA Interim
- Polar night jet: southern hem (July) vs. ERA Interim

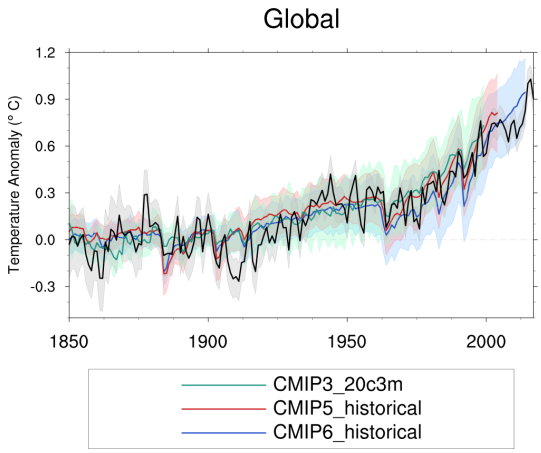


Fig. 97: Observed and simulated time series of the anomalies in annual and global mean surface temperature as in Figure 1; all anomalies are calculated by subtracting the 1850-1900 time mean from the time series. Displayed are the multimodel means of all three CMIP ensembles with shaded range of the respective standard deviation. In black the HadCRUT4 data set (HadCRUT4; Morice et al., 2012). Gray shading shows the 5% to 95% confidence interval of the combined effects of all the uncertainties described in the HadCRUT4 error model (measurement and sampling, bias, and coverage uncertainties) (Morice et al., 2012) (Fig. 2).

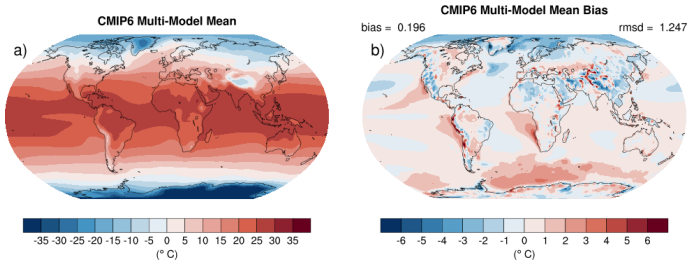


Fig. 98: Annual mean near-surface (2 m) air temperature (°C). (a) Multimodel (ensemble) mean constructed with one realization of CMIP6 historical experiments for the period 1995-2014. Multimodel-mean bias of (b) CMIP6 (1995-2014) compared to the corresponding time period of the climatology from ERA5 (Copernicus Climate Change Service (C3S), 2017). (Fig. 3)

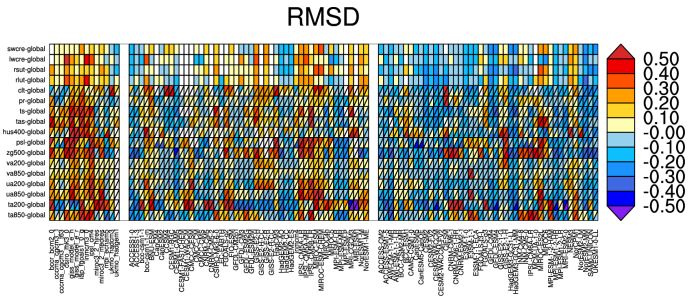


Fig. 99: Relative space-time root-mean-square deviation (RMSD) calculated from the climatological seasonal cycle of the CMIP3, CMIP5, and CMIP6 simulations (1980-1999) compared to observational data sets (Table 5). A relative performance is displayed, with blue shading being better and red shading worse than the median RMSD of all model results of all ensembles. 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) which are marked in Table 5. White boxes are used when data are not available for a given model and variable (Fig. 6).

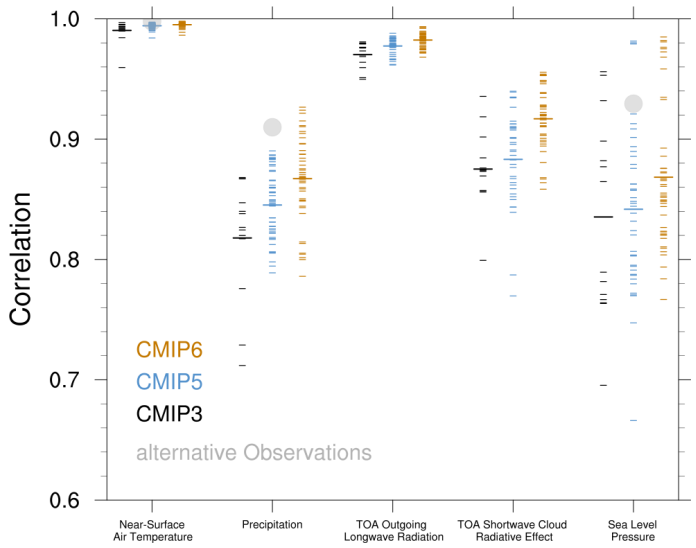


Fig. 100: Centered pattern correlations between models and observations for the annual mean climatology over the period 1980–1999 (Fig. 7).

- 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

Diagnostic plot:

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

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_autoassess_stratosphere.yml

Diagnostics are stored in esmvaltool/diag_scripts/autoassess/

- autoassess_area_base.py: wrapper for autoassess scripts
- stratosphere/strat_metrics_1.py: calculation of metrics
- stratosphere/age_of_air.py: calculate age of stratospheric air

- stratosphere/plotting.py: zonal mean wind and QBO plots
- plot_autoassess_metrics.py: plot normalised assessment metrics

User settings in recipe

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 have been ported to ESMValTool 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
```

Variables

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.

Observations and reformat scripts

ERA-Interim (ta, ua, hus - cmorizers/data/formatters/datasets/era_interim.py)

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.

Example metrics and plots

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 .

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

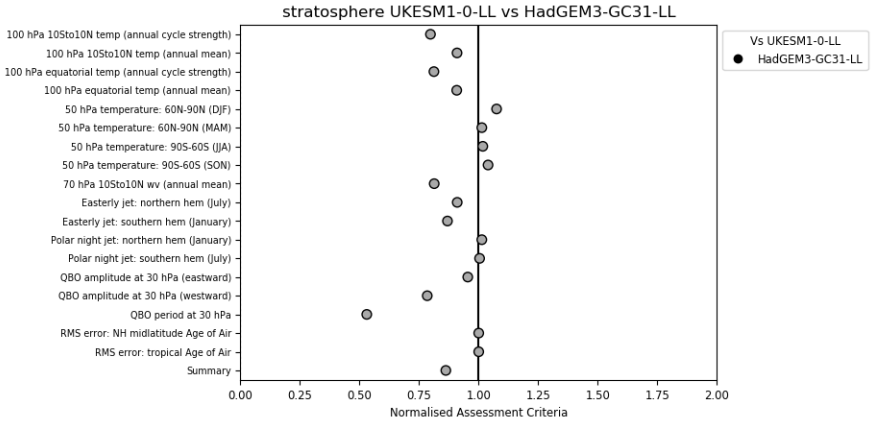


Fig. 101: Standard metrics plot comparing standard metrics from UKESM1-0-LL and HadGEM3-GC31.

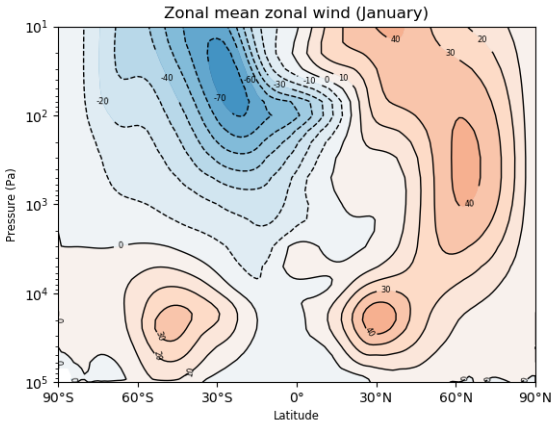


Fig. 102: Zonal mean zonal wind in January for UKESM1-0-LL.

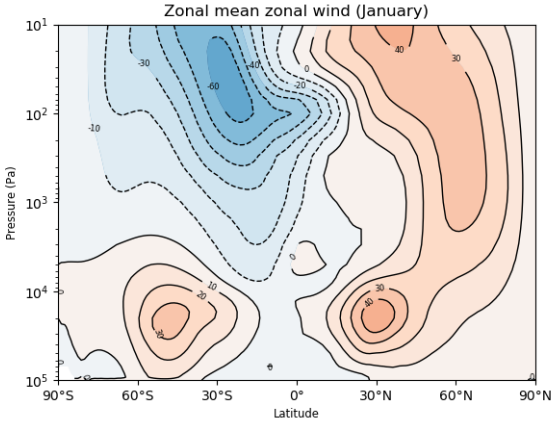


Fig. 103: Zonal mean zonal wind in January for HadGEM3-GC31-LL.

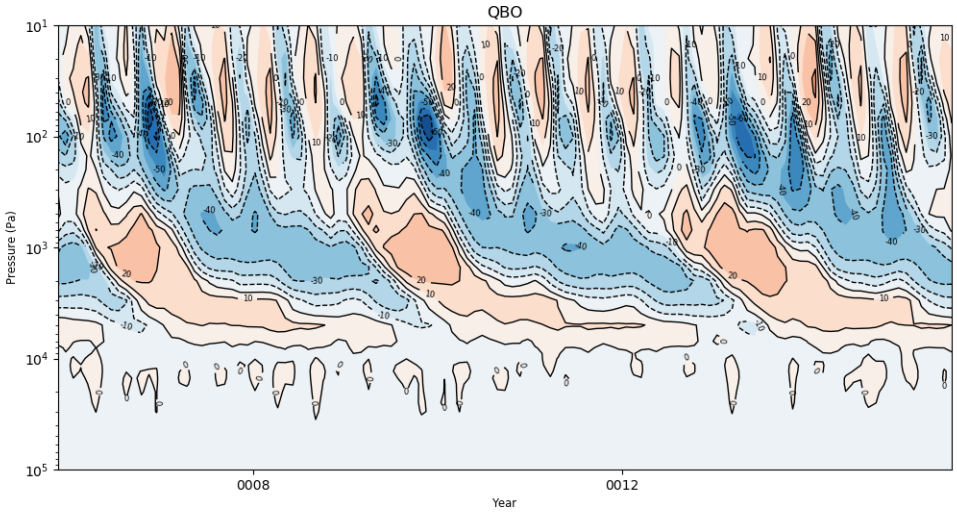


Fig. 104: QBO for UKESM1-0-LL.

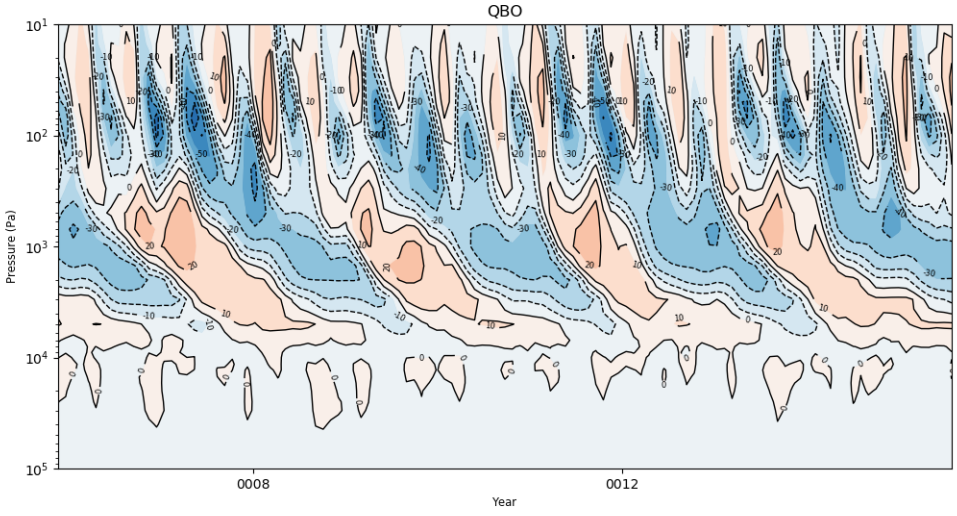


Fig. 105: QBO for HadGEM3-GC31-LL.

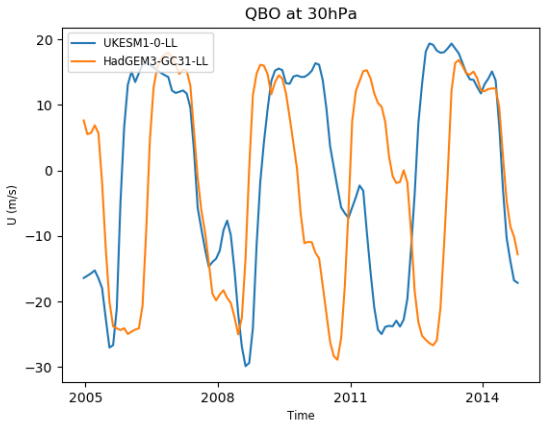


Fig. 106: QBO at 30hPa comparison between UKESM1-0-LL and HadGEM3-GC31-LL.

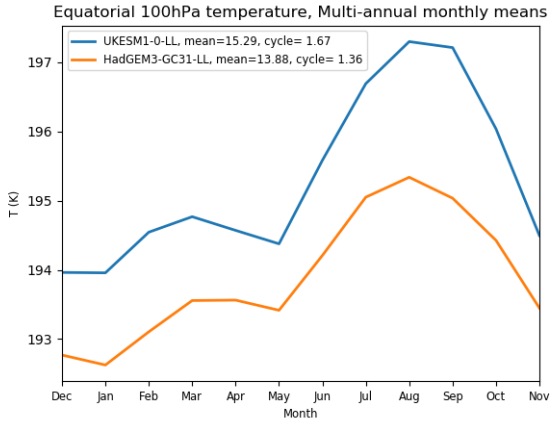


Fig. 107: Equatorial temperature at 100hPa, multi annual means.

Developers

Met Office:

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

ESMValTool:

- Since April 2018: Valeriu Predoi

5.2.23 Land-surface Permafrost - Autoassess diagnostics

Overview

Permafrost thaw is an important impact of climate change, and is the source of a potentially strong Earth system feedback through the release of soil carbon into the atmosphere. This recipe provides metrics that evaluate the climatological performance of models in simulating soil temperatures that control permafrost. Performance metrics (with observation-based estimates in brackets):

- permafrost area (17.46 million square km)
- fractional area of permafrost northwards of zero degree isotherm (0.47)
- soil temperature at 1m minus soil temperature at surface (-0.53 degrees C)
- soil temperature at surface minus air temperature (6.15 degrees C)
- annual amplitude at 1m / annual amplitude at the surface (0.40 unitless)
- annual amplitude at the surface / annual air temperature (0.57 unitless)

Plots:

- Maps of permafrost extent and zero degC isotherm
- Normalised assessment metrics plot comparing control and experiment

The recipe takes as input a control model and experimental model, comparisons being made with these two models.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_autoassess_landsurface_permafrost.yml

Diagnostics are stored in esmvaltool/diag_scripts/autoassess/

- autoassess_area_base.py: wrapper for autoassess scripts
- land_surface_permafrost/permafrost.py: script to calculate permafrost metrics
- plot_autoassess_metrics.py: plot normalised assessment metrics

User settings in recipe

1. Script autoassess_area_base.py

   Required settings for script

   - area: must equal land_surface_permafrost to select this diagnostic
   - control_model: name of model to be used as control
   - exp_model: name of model to be used as experiment

- start: date (YYYY/MM/DD) at which period begins (see note on time gating)
- end: date (YYYY/MM/DD) at which period ends (see note on time gating)

Optional settings for script

- title: arbitrary string with name of diagnostic
- obs_models: unused for this recipe

Required settings for variables

none

Optional settings for variables

none

2. Script plot_autoassess_metrics.py

   Required settings for script

   - area: must equal land_surface_permafrost to select this diagnostic
   - control_model: name of model to be used as control in metrics plot
   - exp_model: name of model to be used as experiment in metrics plot
   - title: string to use as plot title

   Optional settings for script

   none

   Required settings for variables

   none

   Optional settings for variables

   none

Variables

- tas (atmos, monthly mean, longitude latitude time)
- tsl (land, monthly mean, longitude latitude time)
- mrsos (land, monthly mean, longitude latitude time)
- sftlf (mask, fixed, longitude latitude)

Observations and reformat scripts

None

References

- Observed permafrost extent is from https://nsidc.org/data/ggd318/versions/2: Brown, J., O. Ferrians, J. A. Heginbottom, and E. Melnikov. 2002. Circum-Arctic Map of Permafrost and Ground-Ice Conditions, Version 2. Boulder, Colorado USA. NSIDC: National Snow and Ice Data Center. When calculating the global area of permafrost the grid cells are weighted by the proportion of permafrost within them.
- Annual mean air temperature is from: Legates, D. R., and C. J. Willmott, 1990: Mean seasonal and spatial variability in global surface air temperature. Theor. Appl. Climatol., 41, 11-21. The annual mean is calculated from the seasonal mean data available at the Met Office.

- The soil temperature metrics are calcuated following: Charles D. Koven, William J. Riley, and Alex Stern, 2013: Analysis of Permafrost Thermal Dynamics and Response to Climate Change in the CMIP5 Earth System Models. J. Climate, 26. (Table 3) http://dx.doi.org/10.1175/JCLI-D-12-00228.1 The locations used for Table 3 were extracted from the model and the modelled metrics calculated.

Example plots

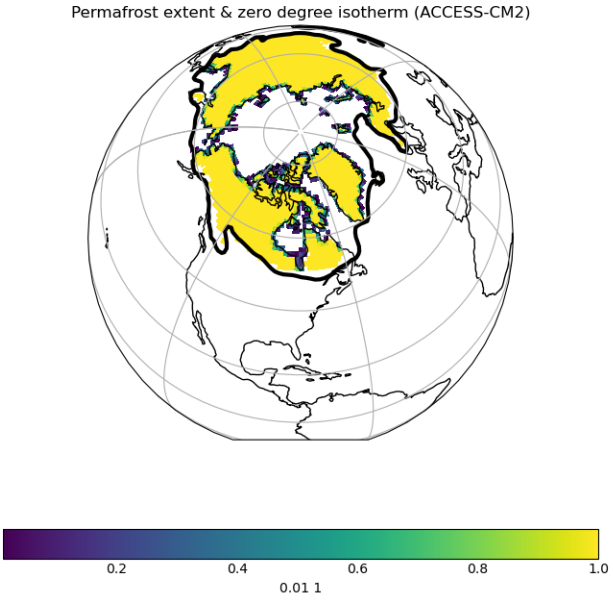


Fig. 108: Permafrost extent and zero degC isotherm, showing North America

Additional notes on usage

The `landsurface_permafrost` 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 Autoassess 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

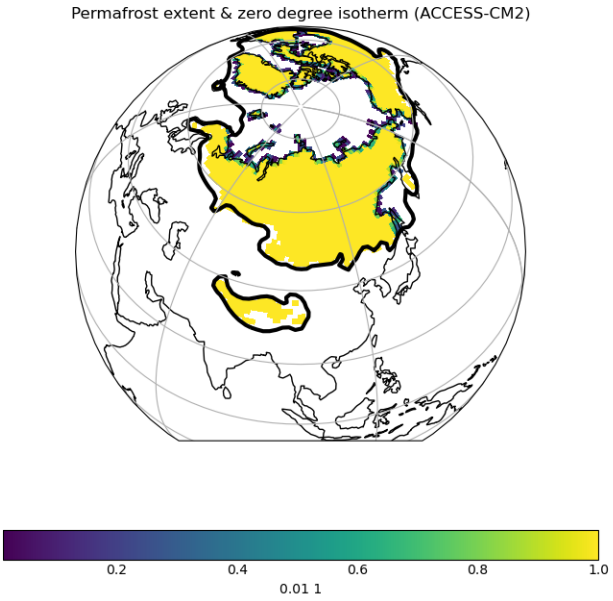


Fig. 109: Permafrost extent and zero degC isotherm, showing Asia and Europe

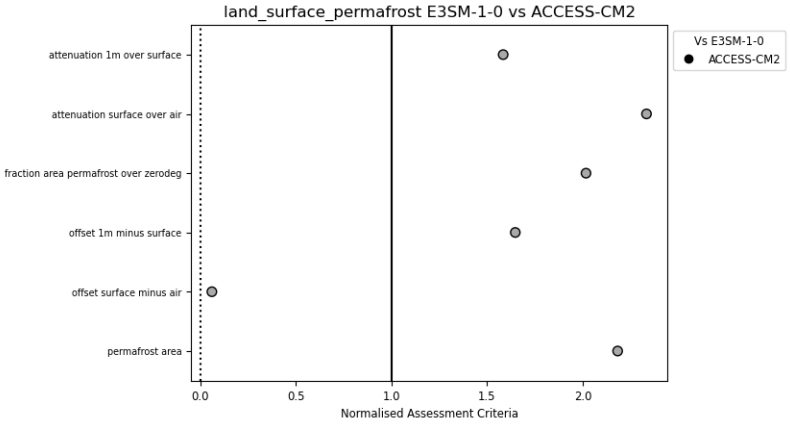


Fig. 110: Normalised metrics plot comparing a control and experiment simulation

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 have been ported to ESMValTool review has completed.

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

```
scripts:
  plot_landsurf_permafrost: &plot_landsurf_permafrost_settings
    <<: *autoassess_landsurf_permafrost_settings
    control_model: MPI-ESM-LR
    exp_model: MPI-ESM-MR
    script: autoassess/plot_autoassess_metrics.py
    ancestors: ['*/autoassess_landsurf_permafrost']
    title: "Plot Land-Surface Permafrost Metrics"
    plot_name: "Permafrost_Metrics"
    diag_tag: aa_landsurf_permafrost
    diag_name: autoassess_landsurf_permafrost
```

5.2.24 Land-surface Surface Radiation - Autoassess diagnostics

Overview

The simulation of surface radiation is central to all aspects of model performance, and can often reveal compensating errors which are hidden within top of atmosphere fluxes. This recipe provides metrics that evaluate the skill of models' spatial and seasonal distribution of surface shortwave and longwave radiation against the CERES EBAF satellite dataset.

Performance metrics:

- median absolute error (model minus observations) net surface shortwave (SW) radiation
- median absolute error (model minus observations) net surface longwave (LW) radiation

Metrics are calculated using model and observation multi-year climatologies (seasonal means) for meteorological seasons: * December-January-February (djf) * March-April-May (mam) * June-July-August (jja) * September-October-November (son) * Annual mean (ann)

Plots:

- Normalised assessment metrics plot comparing control and experiment

The recipe takes as input a control model and experimental model, comparisons being made with these two models.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_autoassess_landsurface_surfrad.yml

Diagnostics are stored in esmvaltool/diag_scripts/autoassess/

- autoassess_area_base.py: wrapper for autoassess scripts
- land_surface_surfrad/surfrad.py: script to calculate surface radiation metrics
- plot_autoassess_metrics.py: plot normalised assessment metrics

User settings in recipe

1. Script autoassess_area_base.py

   Required settings for script

   - area: must equal land_surface_surfrad to select this diagnostic
   - control_model: name of model to be used as control
   - exp_model: name of model to be used as experiment
   - start: date (YYYY/MM/DD) at which period begins (see note on time gating)
   - end: date (YYYY/MM/DD) at which period ends (see note on time gating)
   - climfiles_root: path to observation climatologies

   Optional settings for script

   - title: arbitrary string with name of diagnostic
   - obs_models: unused for this recipe

   Required settings for variables

   none

   Optional settings for variables

   none

2. Script plot_autoassess_metrics.py

   Required settings for script

   - area: must equal land_surface_surfrad to select this diagnostic
   - control_model: name of model to be used as control in metrics plot
   - exp_model: name of model to be used as experiment in metrics plot
   - title: string to use as plot title

   Optional settings for script

   none

   Required settings for variables

   none

   Optional settings for variables

   none

Variables

- rsns (atmos, monthly mean, longitude latitude time)
- rlns (atmos, monthly mean, longitude latitude time)
- sftlf (mask, fixed, longitude latitude)

Observations and reformat scripts

2001-2012 climatologies (seasonal means) from CERES-EBAF Ed2.7.

References

- Loeb, N. G., D. R. Doelling, H. Wang, W. Su, C. Nguyen, J. G. Corbett, L. Liang, C. Mitrescu, F. G. Rose, and S. Kato, 2018: Clouds and the Earth's Radiant Energy System (CERES) Energy Balanced and Filled (EBAF) Top-of-Atmosphere (TOA) Edition-4.0 Data Product. J. Climate, 31, 895-918, doi: 10.1175/JCLI-D-17-0208.1.
- Kato, S., F. G. Rose, D. A. Rutan, T. E. Thorsen, N. G. Loeb, D. R. Doelling, X. Huang, W. L. Smith, W. Su, and S.-H. Ham, 2018: Surface irradiances of Edition 4.0 Clouds and the Earth's Radiant Energy System (CERES) Energy Balanced and Filled (EBAF) data product, J. Climate, 31, 4501-4527, doi: 10.1175/JCLI-D-17-0523.1

Example plots

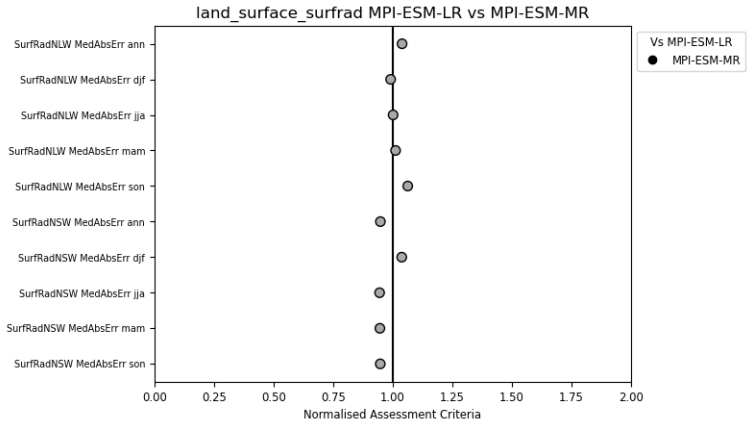


Fig. 111: Normalised metrics plot comparing a control and experiment simulation

Inputs and usage

The `landsurface_soilmoisture` 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 Autoassess 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 have been ported to ESMValTool review has completed.

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

```
scripts:
  autoassess_landsurf_surfrad: &autoassess_landsurf_surfrad_settings
    script: autoassess/autoassess_area_base.py
    title: "Autoassess Land-Surface Diagnostic Surfrad Metric"
    area: land_surface_surfrad
    control_model: UKESM1-0-LL
    exp_model: UKESM1-0-LL
    obs_models: [CERES-EBAF]
    obs_type: obs4MIPs
    start: 1997/12/01
    end: 2002/12/01
```

5.2.25 Land-surface Soil Moisture - Autoassess diagnostics

Overview

Soil moisture is a critical component of the land system, controling surface energy fluxes in many areas of the world. This recipe provides metrics that evaluate the skill of models' spatial and seasonal distribution of soil moisture against the ESA CCI soil moisture ECV.

Performance metrics:

- median absolute error (model minus observations)

Metrics are calculated using model and observation multi-year climatologies (seasonal means) for meteorological seasons:

- December-January-February (djf)
- March-April-May (mam)
- June-July-August (jja)
- September-October-November (son)

Plots:

- Normalised assessment metrics plot comparing control and experiment

The recipe takes as input a control model and experimental model, comparisons being made with these two models.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_autoassess_landsurface_soilmoisture.yml

Diagnostics are stored in esmvaltool/diag_scripts/autoassess/

- land_surface_soilmoisture/soilmoisture.py: script to calculate soil moisture metrics
- plot_autoassess_metrics.py: plot normalised assessment metrics

User settings in recipe

1. Script soilmoisture.py

   Required settings for script

   - area: must equal land_surface_soilmoisture to select this diagnostic

- control_model: name of model to be used as control
- exp_model: name of model to be used as experiment

Optional settings for script

none

Required settings for variables

none

Optional settings for variables

none

2. Script plot_autoassess_metrics.py

   Required settings for script

   - area: must equal land_surface_soilmoisture to select this diagnostic
   - control_model: name of model to be used as control in metrics plot
   - exp_model: name of model to be used as experiment in metrics plot
   - title: string to use as plot title

   Optional settings for script

   none

   Required settings for variables

   none

   Optional settings for variables

   none

Variables

- mrsos (from models: land, monthly mean, longitude latitude time)
- sm (from observations: land, monthly mean, longitude latitude time)

Observations and reformat scripts

1999-2008 climatologies (seasonal means) from ESA ECV Soil Moisture Dataset v1. Produced by the ESA CCI soil moisture project: https://www.esa-soilmoisture-cci.org/node/93

References

- Dorigo, W.A., Wagner, W., Albergel, C., Albrecht, F., Balsamo, G., Brocca, L., Chung, D., Ertl, M., Forkel, M., Gruber, A., Haas, E., Hamer, D. P. Hirschi, M., Ikonen, J., De Jeu, R. Kidd, R. Lahoz, W., Liu, Y.Y., Miralles, D., Lecomte, P. (2017). ESA CCI Soil Moisture for improved Earth system understanding: State-of-the art and future directions. In Remote Sensing of Environment, 2017, ISSN 0034-4257, https://doi.org/10.1016/j.rse.2017.07.001.
- Gruber, A., Scanlon, T., van der Schalie, R., Wagner, W., Dorigo, W. (2019). Evolution of the ESA CCI Soil Moisture Climate Data Records and their underlying merging methodology. Earth System Science Data 11, 717-739, https://doi.org/10.5194/essd-11-717-2019

Example plots

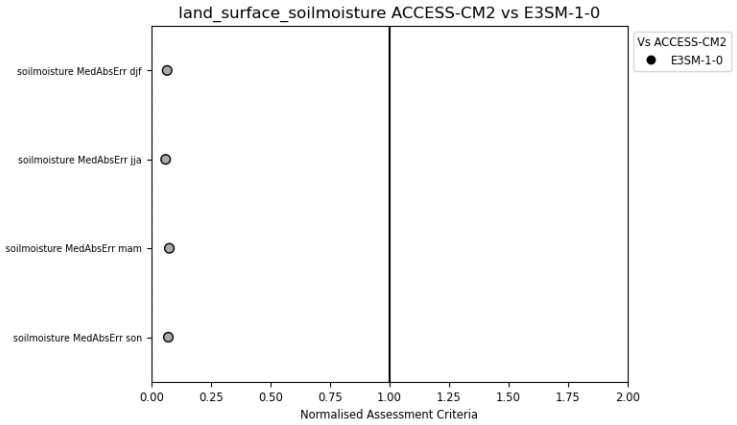


Fig. 112: Normalised metrics plot comparing a control and experiment simulation

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

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 Annular Modes (AM, 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 AM index. The regression of the monthly mean geopotential height onto this monthly averaged index represents the AM pattern for each selected pressure level.

The outputs of the procedure are the monthly time series and the histogram of the daily zonal mean AM 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, the Northern or Southern hemisphere (NH or SH) and a subset of pressure levels of interest.

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

User settings

Hemisphere of interest (NH or SH)

Variables

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

Observations and reformat scripts

None.

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.

Example plots

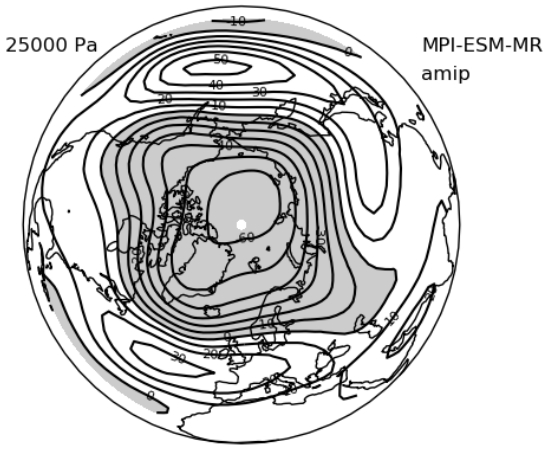


Fig. 113: Regression map of the Northern Hemisphere zonal mean AM index onto geopotential height, for a selected pressure level (250 hPa) for the MPI-ESM-MR model (CMIP5 AMIP experiment, period 1979-2008). Negative values are shaded in grey.

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

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.

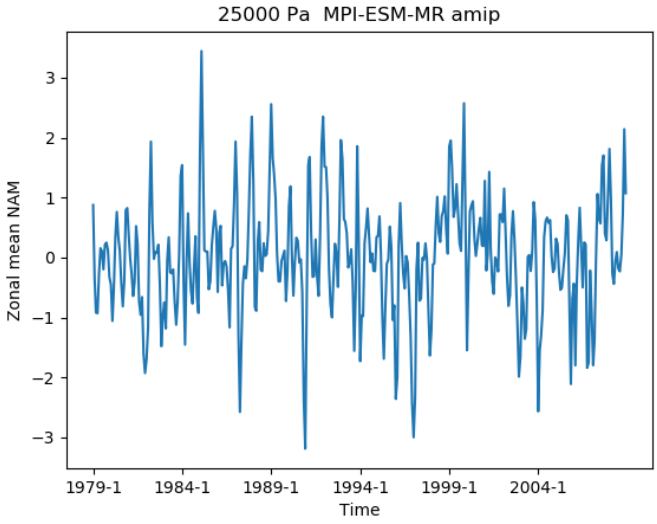


Fig. 114: Time series of the Northern Hemisphere zonal mean AM index for a selected pressure level (250 hPa) for the MPI-ESM-MR model (CMIP5 AMIP experiment, period 1979-2008).

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 are needed. These are required to perform a vertical interpolation, substituting data in pressure levels where surface pressure is lower than the respective level and fields are not stored as an output of the analysed model.

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.

In order to account for possible gaps in pressure levels, the daily fields of 2D near-surface temperature and horizontal velocities.'

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;

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

Variables

Default variables needed for computation of energy budgets and transports:

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

Additional variables needed for water mass and latent energy computation (optional, with 'wat' set to 'true'):

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

Additional variable needed for LEC computations (optional, with 'lec' set to 'true'):

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

Additional variables needed for material entropy production computations with direct method (optional, with 'entr' set to 'true' and 'mep' to '2' or '3'):

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

Additional variables needed for material entropy production computations with indirect method (optional, with 'entr' set to 'true' and 'mep' to '1' or '3'):

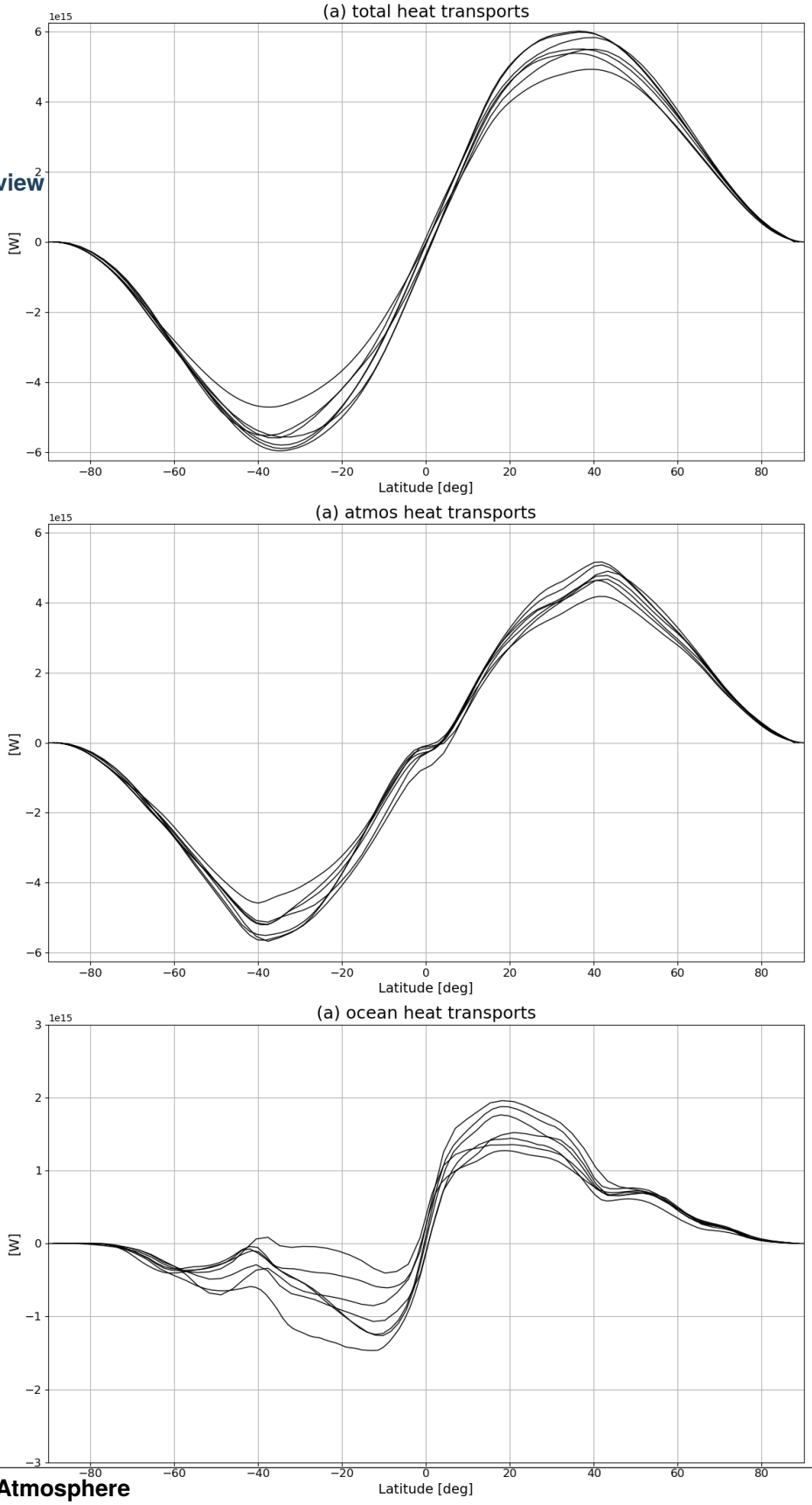
- tas (atmos, daily mean, time latitude longitude)
- uas (atmos, daily mean, time latitude longitude)
- vas (atmos, daily mean, time latitude longitude)

Depending on the user's options, variables listed above must be provided. All other variables shall be commented in the recipe file.

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-12-3805-2019
- 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

Example plots



5.2.28 Zonal and Meridional Means

Overview

This functional diagnostic 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

Fig. 115: Meridional transports.

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 regridding on a common grid for all model and observational datasets should be performed in preprocessing (if datasets are on different grids). Also note that it is allowed to use the same dataset (with varying parameters like experiment or ensemble or mip) for both CONTROL and EXPERIMENT (as long as at least one data parameter is different).

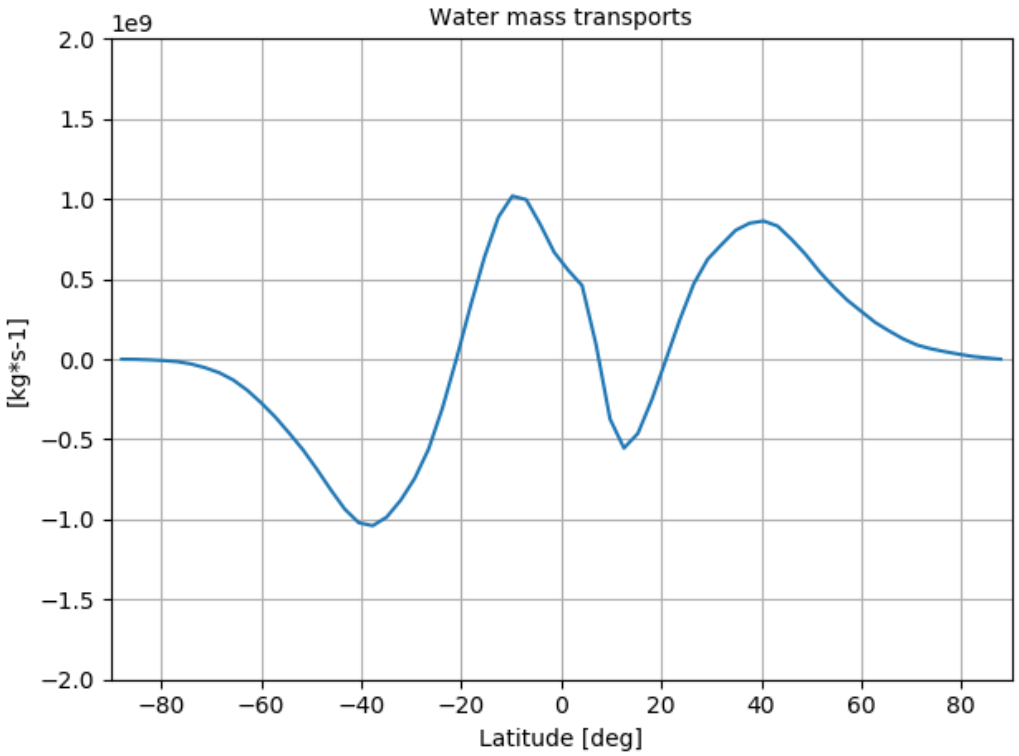


Fig. 116: Water mass transports.

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

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`;

Variables

- any variable

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 preprocessing as model data

References

- none, basic technical analysis

Example plots

5.2.29 Radiation Budget

Overview

The aim of monitoring the energy budget is to understand the (im)balance of energy flux between the atmosphere and the surface of a model due to its link with the hydrological cycle and climate change.

This diagnostic analyses the radiation budget by separating top-of-atmosphere fluxes into clear-sky and cloud forcing components, and surface fluxes into downwelling and upwelling components. Model predictions are compared against three observational estimates, one of which (Stephens et al. 2012) includes uncertainty estimates. When the black error bars overlap the zero line, the model is consistent with observations according to Stephens et al. (2012).

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_radiation_budget.yml

Diagnostics are stored in esmvaltool/diag_scripts/radiation_budget/

- radiation_budget.py: Plot the global radiation budget.
- seasonal_radiation_budget.py: Write the global climatological seasonal radiation budget to a text file.

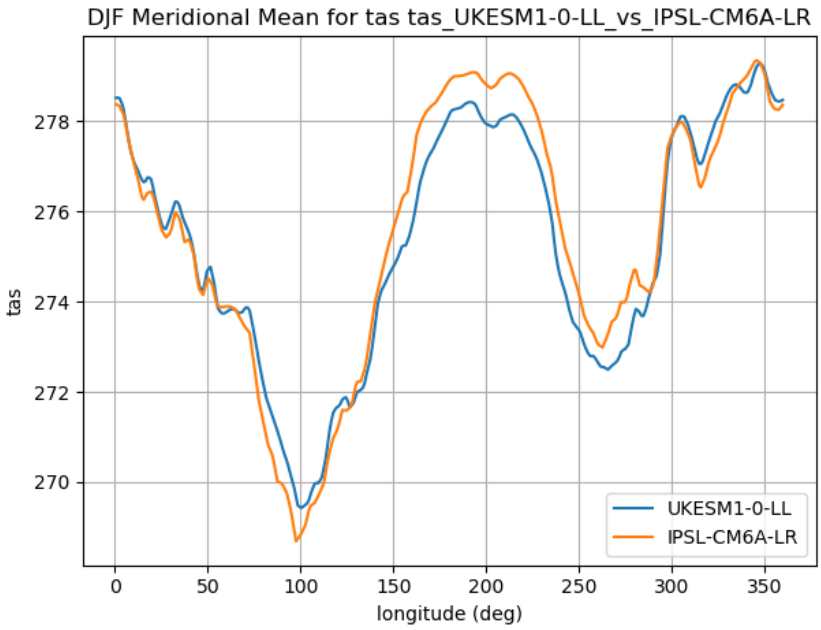


Fig. 117: Meridional seasonal mean for winter (DJF) comparison beween CMIP6 UKESM1 and IPSL models.

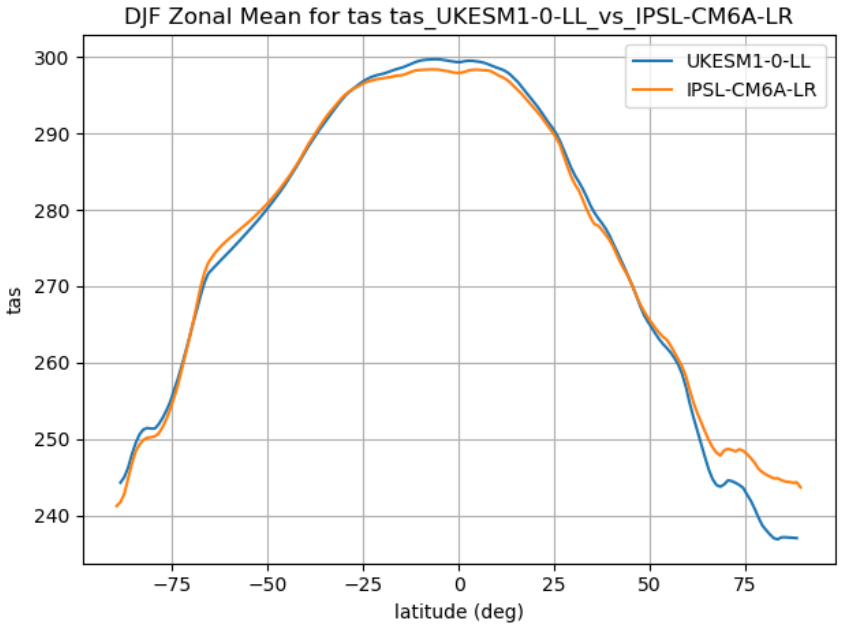


Fig. 118: Zonal seasonal mean for winter (DJF) comparison beween CMIP6 UKESM1 and IPSL models.

User settings in recipe

None

Variables

- rss (atmos, monthly mean, longitude latitude time)
- rsdt (atmos, monthly mean, longitude latitude time)
- rsut (atmos, monthly mean, longitude latitude time)
- rsutcs (atmos, monthly mean, longitude latitude time)
- rsds (atmos, monthly mean, longitude latitude time)
- rls (atmos, monthly mean, longitude latitude time)
- rlut (atmos, monthly mean, longitude latitude time)
- rlutcs (atmos, monthly mean, longitude latitude time)
- rlds (atmos, monthly mean, longitude latitude time)
- hfss (atmos, monthly mean, longitude latitude time)
- hfls (atmos, monthly mean, longitude latitude time)

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)
- Demory observations can be found in esmvaltool/diag_scripts/radiation_budget/Demory_et_al_2014_obs_Energy_Budget.yml and are from Figure 2 in Demory et al. (2014).
- Stephens observations can be found in esmvaltool/diag_scripts/radiation_budget/Stephens_et_al_2012_obs_Energy_Budget.yml from figure 1b in Stephens et al. (2012).

References

- Demory, ME., Vidale, P.L., Roberts, M.J. et al. The role of horizontal resolution in simulating drivers of the global hydrological cycle. Clim Dyn 42, 2201–2225 (2014). https://doi.org/10.1007/s00382-013-1924-4
- Stephens, G., Li, J., Wild, M. et al. An update on Earth's energy balance in light of the latest global observations. Nature Geosci 5, 691–696 (2012). https://doi.org/10.1038/ngeo1580

Example plots

5.2.30 AOD AeroNET Assess

Overview

This diagnostic evaluates model aerosol optical depth (AOD) against ground based observations from the AeroNET measurement network. Monthly mean AOD data is downloaded from the AeroNET website and formatted (CMORized) using the AERONET downloader and formatter within ESMValTool.

Multiannual seasonal means are calculated from the model output and compared with a multiannual seasonal mean climatology generated from AeroNET observational data. At each AeroNET station the data are screened for validity according to the following default criteria:

- 1. Monthly means must be generated from at least one AOD observation in that month.

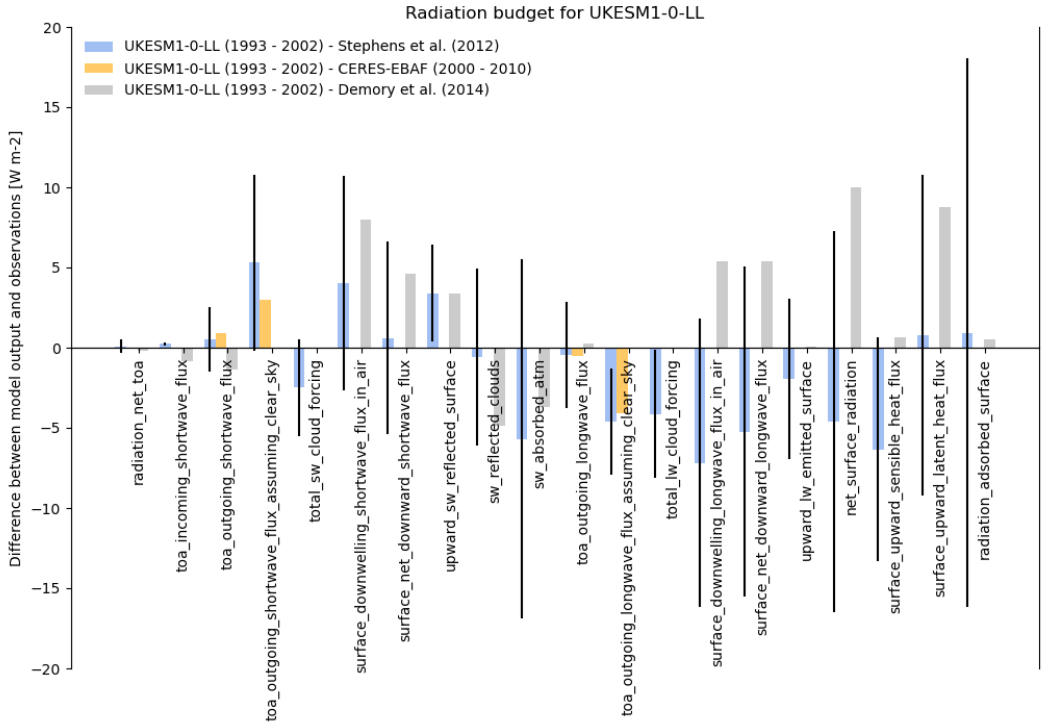


Fig. 119: Radiation budget for UKESM1-0-LL

- 2. Seasonal means for DJF, MAM, JJA and SON must be calculated from three monthly means, i.e. a monthly mean from December January and Feburary.
- 3. For a given year to be valid, there must be a seasonal mean for each climate season i.e. DJF, MAM, JJA and SON.
- 4. For a multiannual seasonal means there must be at least five seasonaal means over the time range of interest.

NOTE: The code is designed to be flexible and the default criteria can be changed according to the user's requirements (see the user settings below).

The evaluation is visualised by plotting model output as 2D filled contours and overlaying AeroNET observations at model grid cells co-located with the AeroNET measurement stations. Statistical data (root mean square error) is generated using AeroNET observations at model grid cells co-located with the AeroNET measurement stations.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_aod_aeronet_assess.yml

Diagnostics are stored in esmvaltool/diag_scripts/aerosols/

- aod_aeronet_assess.py: Plot the AOD evaluation.
- aero_utils.py: Utility functions commonly used by aerosol assessment routines.

User settings in recipe

1. Script aod_aeronet_assess.py

   Required settings for script

   - wavel: The wavelength of interest for the evaluation. Defaults to 440nm.
   - min_days_per_mon: The minimum number of days used to calculate the AOD monthly mean. Defaults to 1.
   - min_mon_per_seas: The minimum number of seasons used to calculate each seasonal mean. This must be between 1 and 3. Defaults to 3.
   - min_seas_per_year: The minimum number of seasonal means in each year. This must be between 1 and 4. Defaults to 4.
   - min_seas_per_clim: The minimum number of seasonal means used to calculate the multiannual seasonal mean. This must be btween 1 and the number of years of available AeroNET data. Defaults to 5.

   Optional settings for script

   - None

   Required settings for variables

   - None

   Optional settings for variables

   - None

   Required settings for preprocessor

   - None

   Optional settings for preprocessor

   - None

Color tables

- brewer_Spectral_11

Variables

- od440aer (atmos, monthly mean, longitude latitude time)
- od550aer (atmos, monthly mean, longitude latitude time)
- od870aer (atmos, monthly mean, longitude latitude time)

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.
- The AeroNET data is downloaded from the AeroNET website using the downloader:

```
$ esmvaltool data download AERONET.
```

- The AeroNET data is formatteed (CMORized) using the formatter:

```
$ esmvaltool data format AERONET.
```

References

- Holben B.N., T.F.Eck, I.Slutsker, D.Tanre, J.P.Buis, A.Setzer, E.Vermote, J.A.Reagan, Y.Kaufman, T.Nakajima, F.Lavenu, I.Jankowiak, and A.Smirnov, 1998: AERONET - A federated instrument network and data archive for aerosol characterization, Rem. Sens. Environ., 66, 1-16.
- Holben, B.N., D.Tanre, A.Smirnov, T.F.Eck, I.Slutsker, N.Abuhassan, W.W.Newcomb, J.Schafer, B.Chatenet, F.Lavenue, Y.J.Kaufman, J.Vande Castle, A.Setzer, B.Markham, D.Clark, R.Frouin, R.Halthore, A.Karnieli, N.T.O'Neill, C.Pietras, R.T.Pinker, K.Voss, and G.Zibordi, 2001: An emerging ground-based aerosol climatology: Aerosol Optical Depth from AERONET, J. Geophys. Res., 106, 12 067-12 097.
- Mulcahy, J. P., Johnson, C., Jones, C. G., Povey, A. C., Scott, C. E., Sellar, A., Turnock, S. T., Woodhouse, M. T., Abraham, N. L., Andrews, M. B., Bellouin, N., Browse, J., Carslaw, K. S., Dalvi, M., Folberth, G. A., Glover, M., Grosvenor, D. P., Hardacre, C., Hill, R., Johnson, B., Jones, A., Kipling, Z., Mann, G., Mollard, J., O'Connor, F. M., Palmiéri, J., Reddington, C., Rumbold, S. T., Richardson, M., Schitgens, N. A. J., Stier, P., Stringer, M., Tang, Y., Walton, J., Woodward, S., and Yool. A.: Description and evaluation of aerosol in UKESM1 and HadGEM3-GC3.1 CMIP6 historical simulations, Geosci. Model Dev., 13, 6383–6423, 2020

Example plots

5.2.31 Surface trace gases

Overview

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_surface_trace_gas_ch4.yml
- recipe_surface_trace_gas_co2.yml
- recipe_surface_trace_gas_n2o.yml

Diagnostics are stored in esmvaltool/diag_scripts/surface_trace_gases/

- diagnostic_surface_trace_gases.py : Plots the different surface trace gas evaluations.

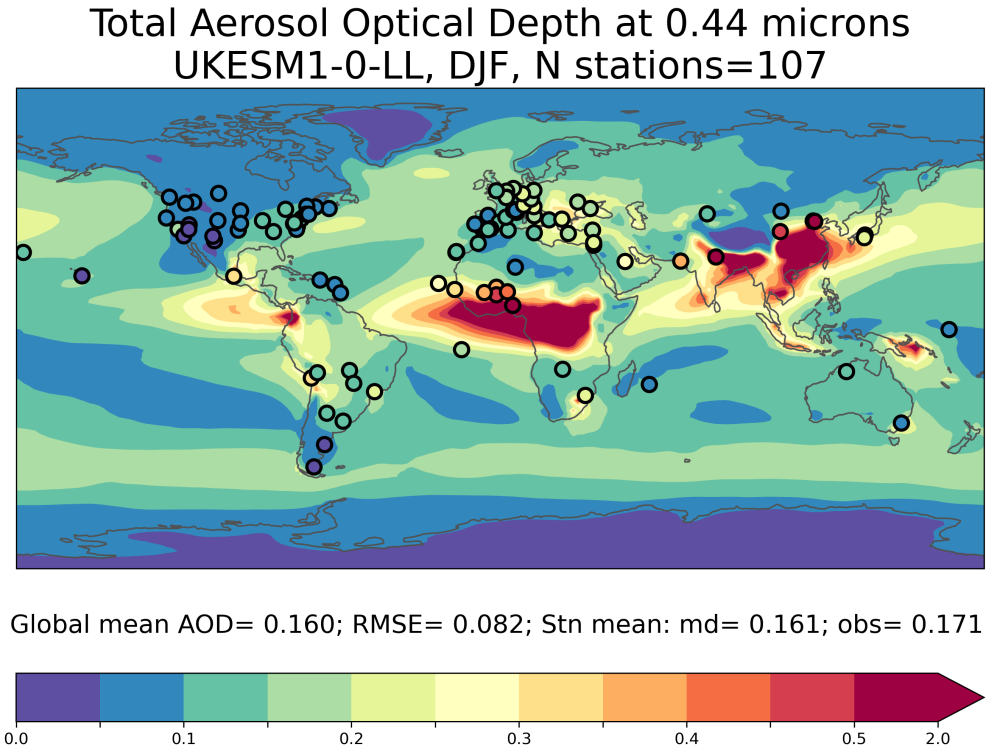


Fig. 120: Evaluation of AOD at 440 nm from UKESM1 historical ensemble member r1i1p1f2 against the AeroNET climatology from ground-based observations for Dec-Jan-Feb. The multiannual seasonal mean is calculated for the model data for the period 1994-2014. The model output is overlaid with the observational climatology.

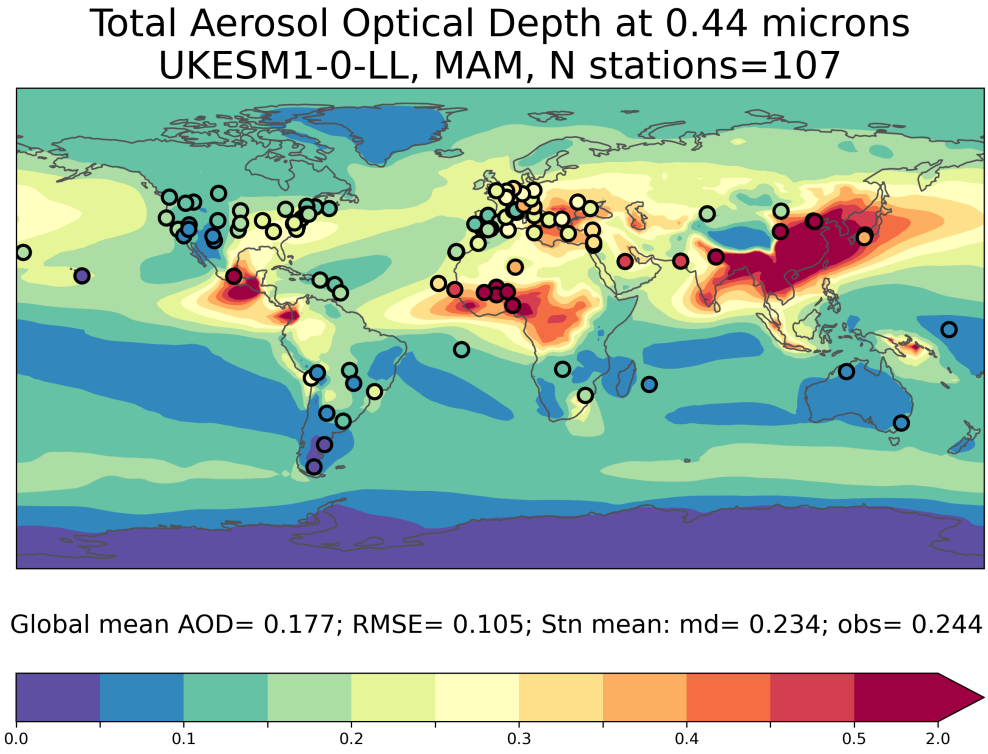


Fig. 121: Evaluation of AOD at 440 nm from UKESM1 historical ensemble member r1i1p1f2 against the AeroNET climatology from ground-based observations for Mar_Apr_May. The multiannual seasonal mean is calculated for the model data for the period 1994-2014. The model output is overlaid with the observational climatology.

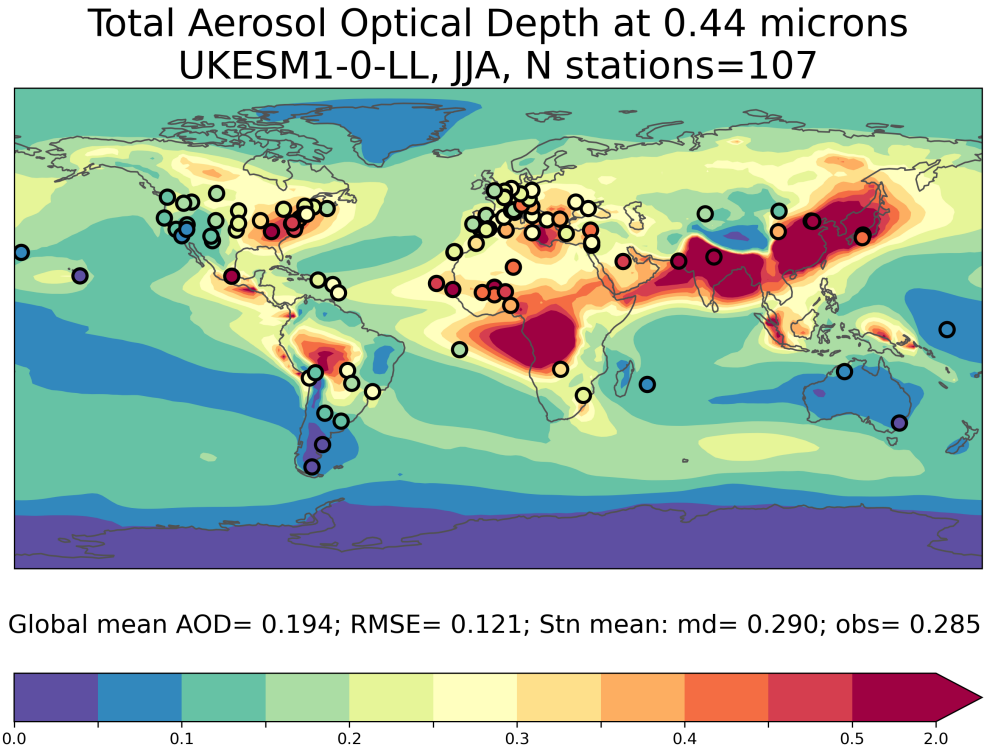


Fig. 122: Evaluation of AOD at 440 nm from UKESM1 historical ensemble member r1i1p1f2 against the AeroNET climatology from ground-based observations for Jun-Jul-Aug. The multiannual seasonal mean is calculated for the model data for the period 1994-2014. The model output is overlaid with the observational climatology.

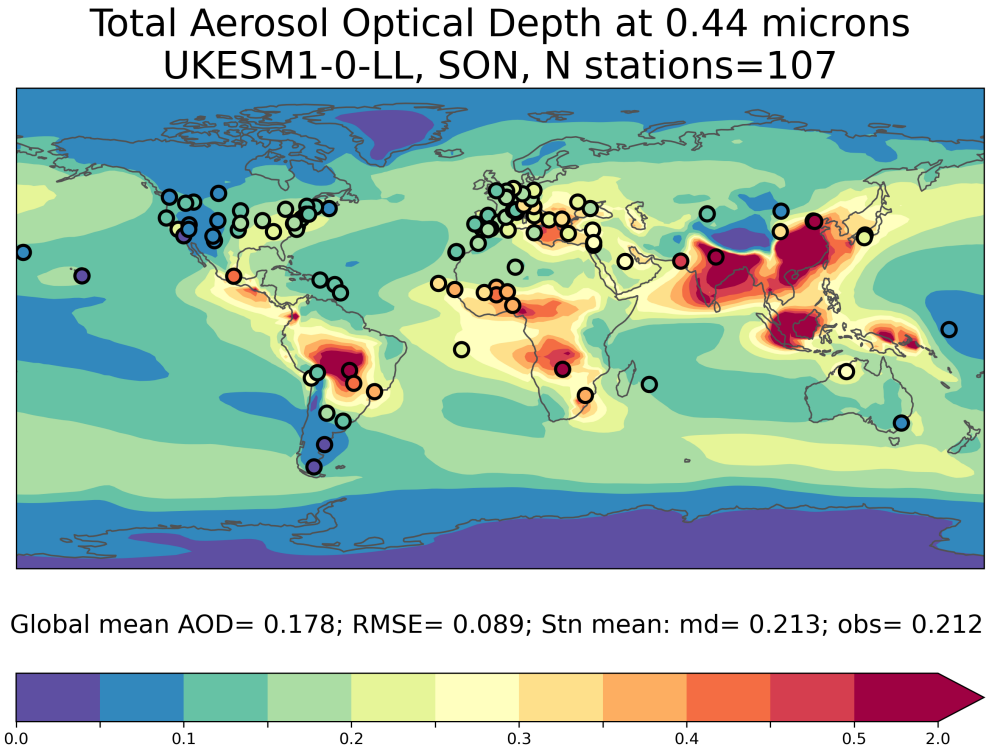


Fig. 123: Evaluation of AOD at 440 nm from UKESM1 historical ensemble member r1i1p1f2 against the AeroNET climatology from ground-based observations for Sep-Oct-Nov. The multiannual seasonal mean is calculated for the model data for the period 1994-2014. The model output is overlaid with the observational climatology.

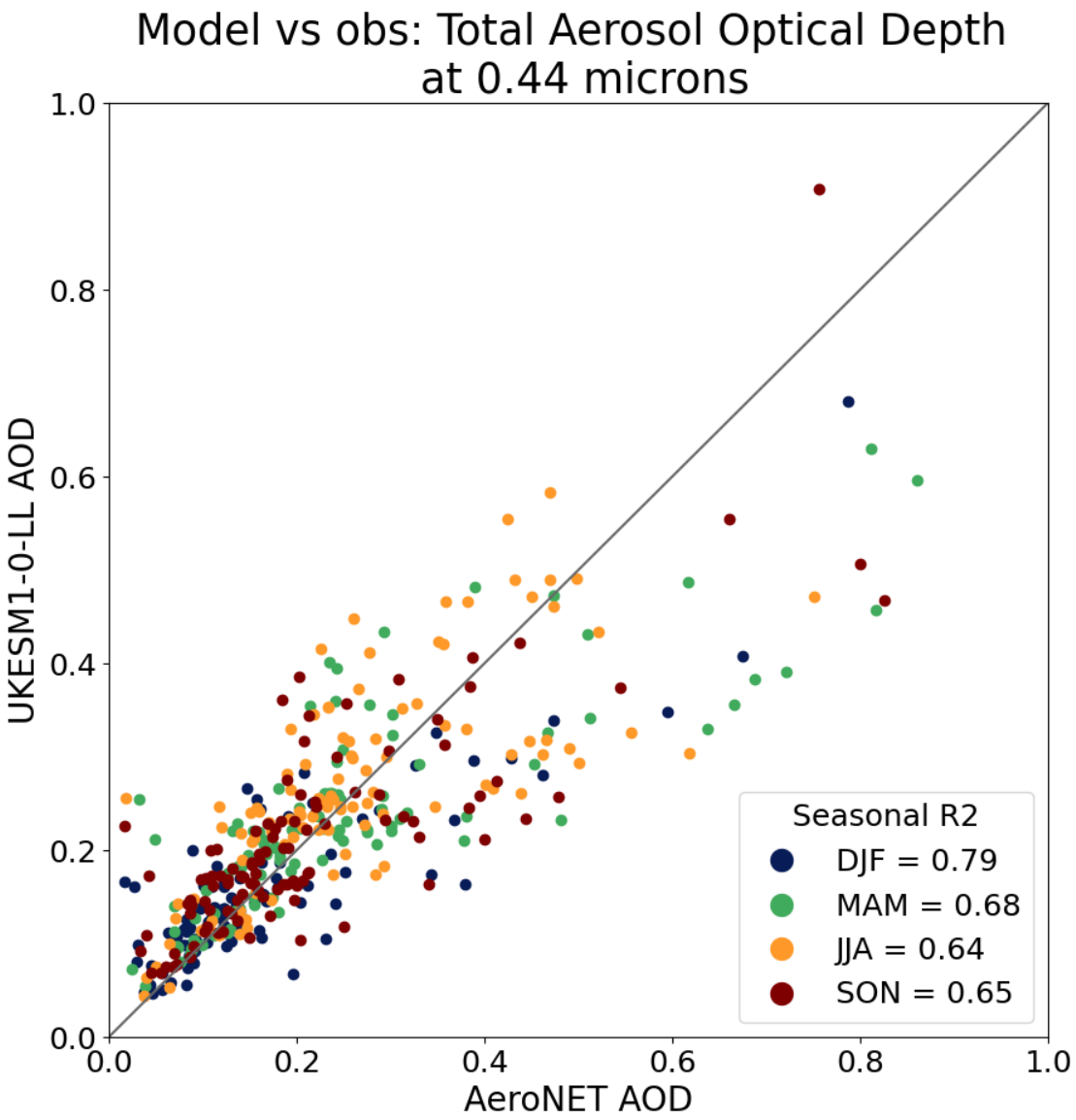


Fig. 124: Evaluation of AOD at 440 nm from UKESM1 historical ensemble member r1i1p1f2 against the AeroNET climatology from ground-based observations for Dec-Jan-Feb, Mar_Apr_May, Jun-Jul-Aug and Sep-Oct-Nov. The multiannual seasonal mean is calculated for the model data for the period 1994-2014.

- utils_surface_trace_gases.py: Utility functions used for the surface trace gas evaluation routines.

This diagnostic allows to produce 3 different types of analysis, typically comparing station data and model data. In the case of zonal plots, the latitude ranges considered are 60N - 90N, 30N - 60N, 30S - 30N, 90S - 30S.

- Seasonal maps of the trace gas surface concentration overlaid with station data points. A complimentary scatter plot compares the surface concentration at the station locations with the model data extracted at these same locations for each season.
- Zonal plots of surface concentration time series and seasonal cycle. The left column includes monthly surface concentration time series. The center-left column includes yearly surface concentration scatter plots. The center-right column includes multi-annual monthly anomalies. The right column includes yearly min/max months time series. The plot is inspired from Figure 8 of Folberth et al. (2018).
- Zonal plots of surface concentration amplitude and growth. The left column includes yearly concentration amplitude time series. The center column includes yearly concentration growth time series. The right column includes yearly sensitivities of amplitude/growth.
- Taylor diagram of trace gas surface concentration for the observations and the different models from the recipe.

Each analysis can be turned on or off using the argument *plots* in the recipe. Furthermore, one can change how the station data is pre-processed before plotting by filtering stations with a minimum amount of data for months, seasons, and years, using the *min_mon_per_seas*, *min_seas_per_year*, and *min_seas_per_clim* arguments in the recipe (see next section).

User settings in recipe

1. Preprocessor

- `extract_surface_from_atm` : Extract surface from the input 3D atmospheric variable based on the surface pressure *ps*.

1. Script diagnostic_surface_trace_gases.py

   Required settings for script * trace_gas: The surface trace gas for which to run the evaluation. * min_mon_per_seas: The minimum number of months used to calculate each seasonal mean. This must be between 1 and 3. Recommended value is 3. * min_seas_per_year: The minimum number of seasonal means in each year. This must be between 1 and 4. Recommended value is 4. * min_seas_per_clim: The minimum number of seasonal means used to calculate the multiannual seasonal mean. This must be between 1 and the number of years of available NOAA GML Surface Flask data.

   Optional settings for script

   - plots: List of the diagnostics to plot among *seas_maps*, *timeserie_lat*, *sensitivity_ampl_trend* and *taylor_diag*. Default is all of them.

   Required settings for variables

   - None

   Optional settings for variables

   - None

   Required settings for preprocessor

   - None

   Optional settings for preprocessor

   - None

Variables

The trace gas depends on the recipe used and can be one of the following:

- *ch4* (atmos, monthly mean, height longitude latitude time)
- *co2* (atmos, monthly mean, height longitude latitude time)
- *n2o* (atmos, monthly mean, height longitude latitude time)
- *ps* (atmos, monthly mean, longitude latitude time) as a supplementary variable for the *extract_surface_from_atm* preprocessor.

Observations and reformat scripts

- The NOAA GML Surface Flask data is downloaded from the NOAA GML website using the downloaders:

```
$ esmvaltool data download NOAA-GML-SURFACE-FLASK-CH4.
$ esmvaltool data download NOAA-GML-SURFACE-FLASK-CO2.
$ esmvaltool data download NOAA-GML-SURFACE-FLASK-N2O.
```

- The NOAA GML Surface Flask data is formatted using the formatters:

```
$ esmvaltool data format NOAA-GML-SURFACE-FLASK-CH4.
$ esmvaltool data format NOAA-GML-SURFACE-FLASK-CO2.
$ esmvaltool data format NOAA-GML-SURFACE-FLASK-N2O.
```

References

- Folberth et al.: Description and Evaluation of an Emission-Driven and Fully Coupled Methane Cycle in UKESM1, 10.1029/2021MS002982, 2018.

Example plots

5.2.32 Lamb Weathertypes

Overview

A diagnostic to calculate Lamb weathertypes over a given region. Furthermore, correlations between weathertypes and precipitation patterns over a given area can be calculated and 'combined' or 'simplified' weathertypes can be derived. Additionally, mean fields, as well as anomalies and standard deviations can be plotted.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_weathertyping.yml

Diagnostics are stored in esmvaltool/diag_scripts/weathertyping/

- weathertyping.py: calculate lamb and simplified WT, plot mean, anomalies and std for each WT for psl, tas, pr

User settings in recipe

1. weathertyping.py

   Required settings for script

   Optional settings for script

   - correlation_threshold: correlation threshold for selecting similar WT pairs, only needed if automatic_slwt==True and predefined_slwt==False. default=0.9

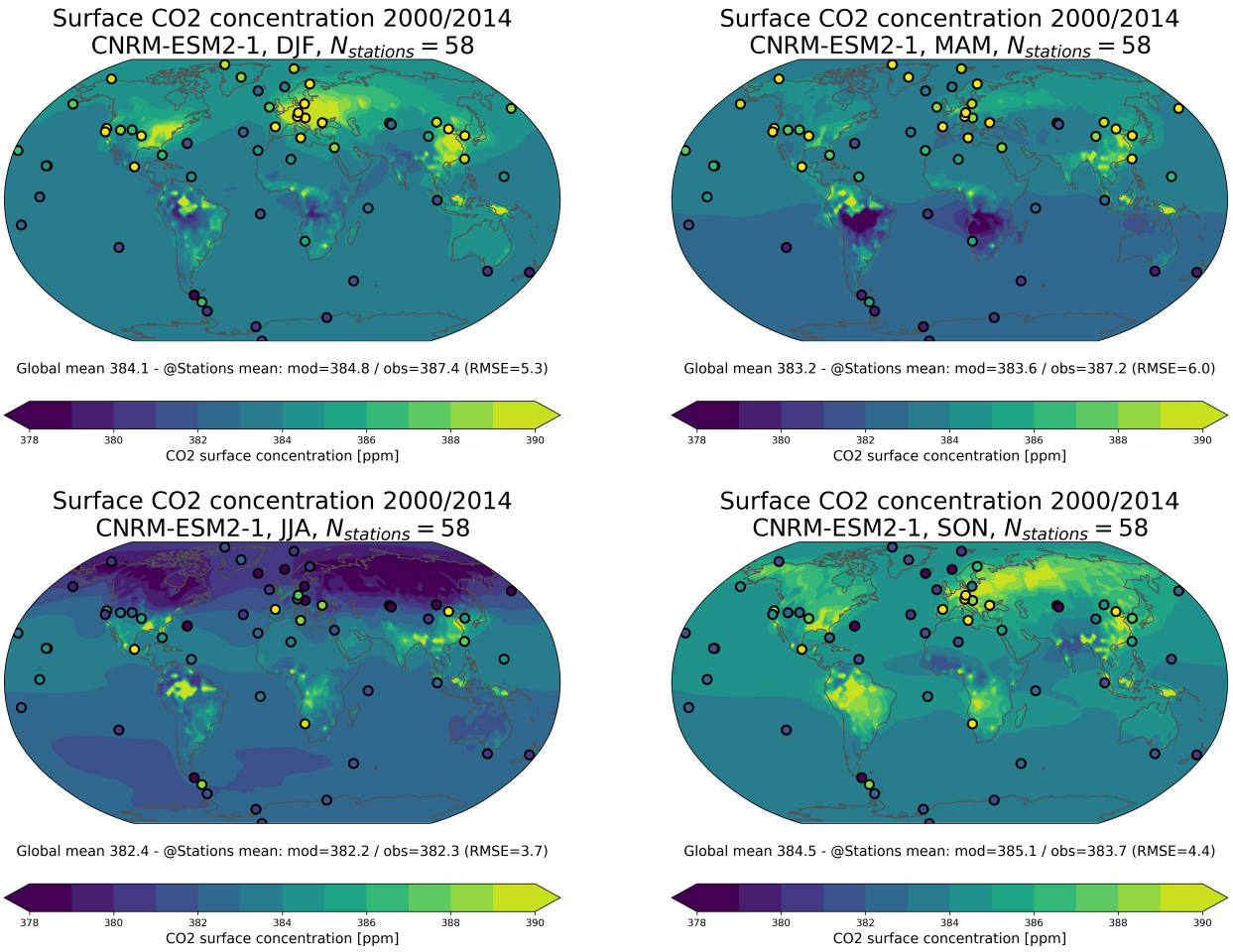


Fig. 125: Evaluation of seasonal surface concentration of CO2 from CNRM-ESM2-1 esm-hist member r1i1p1f3 against the NOAA GML climatology from ground-based observations. The multiannual seasonal mean is calculated for the model data for the period 2000-2014. The model output is overlaid with the observational climatology.

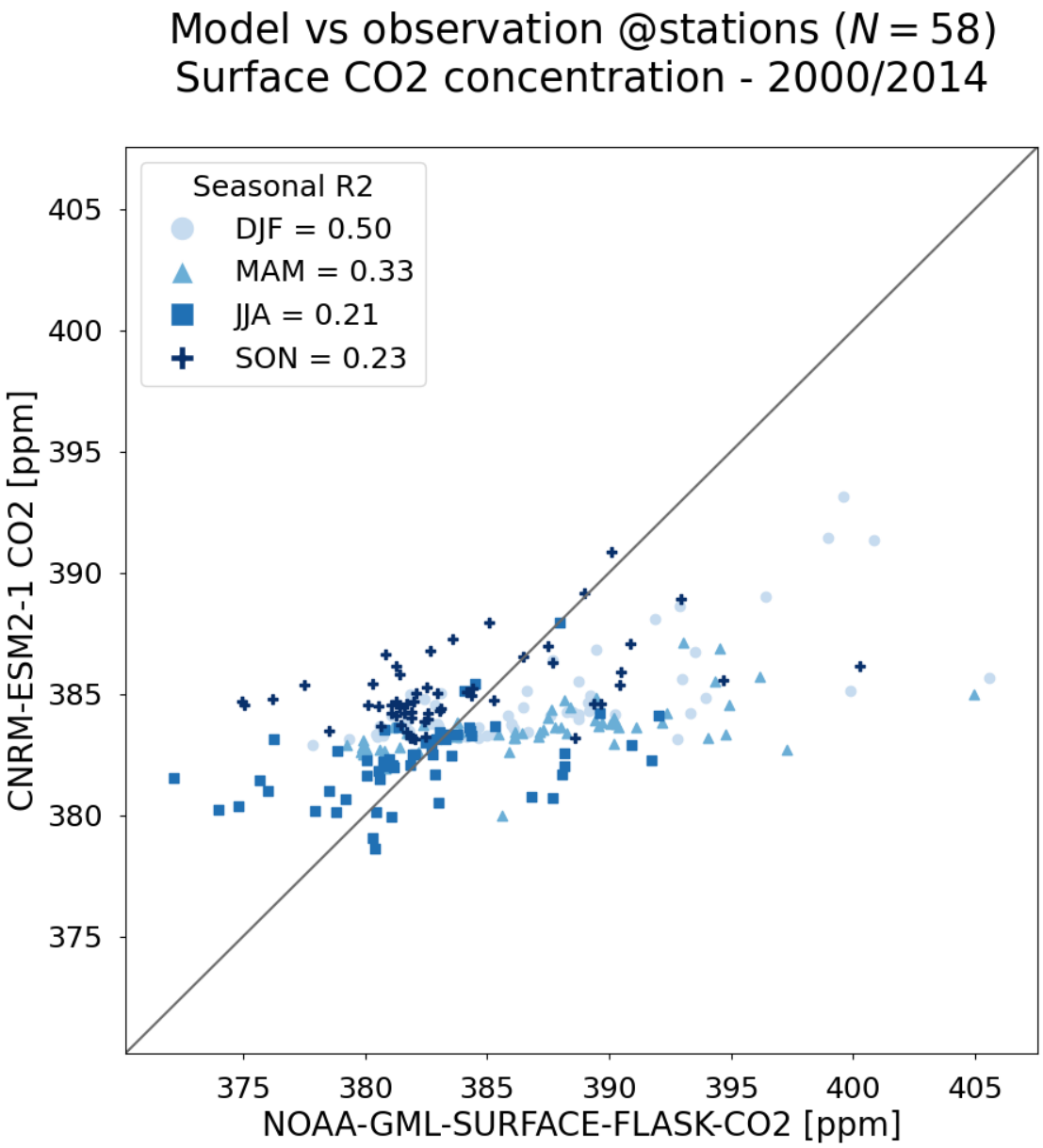


Fig. 126: Evaluation of seasonal surface concentration of CO2 from CNRM-ESM2-1 esm-hist member r1i1p1f3 against the NOAA GML climatology from ground-based observations. The multiannual seasonal mean is calculated for the model data for the period 2000-2014.

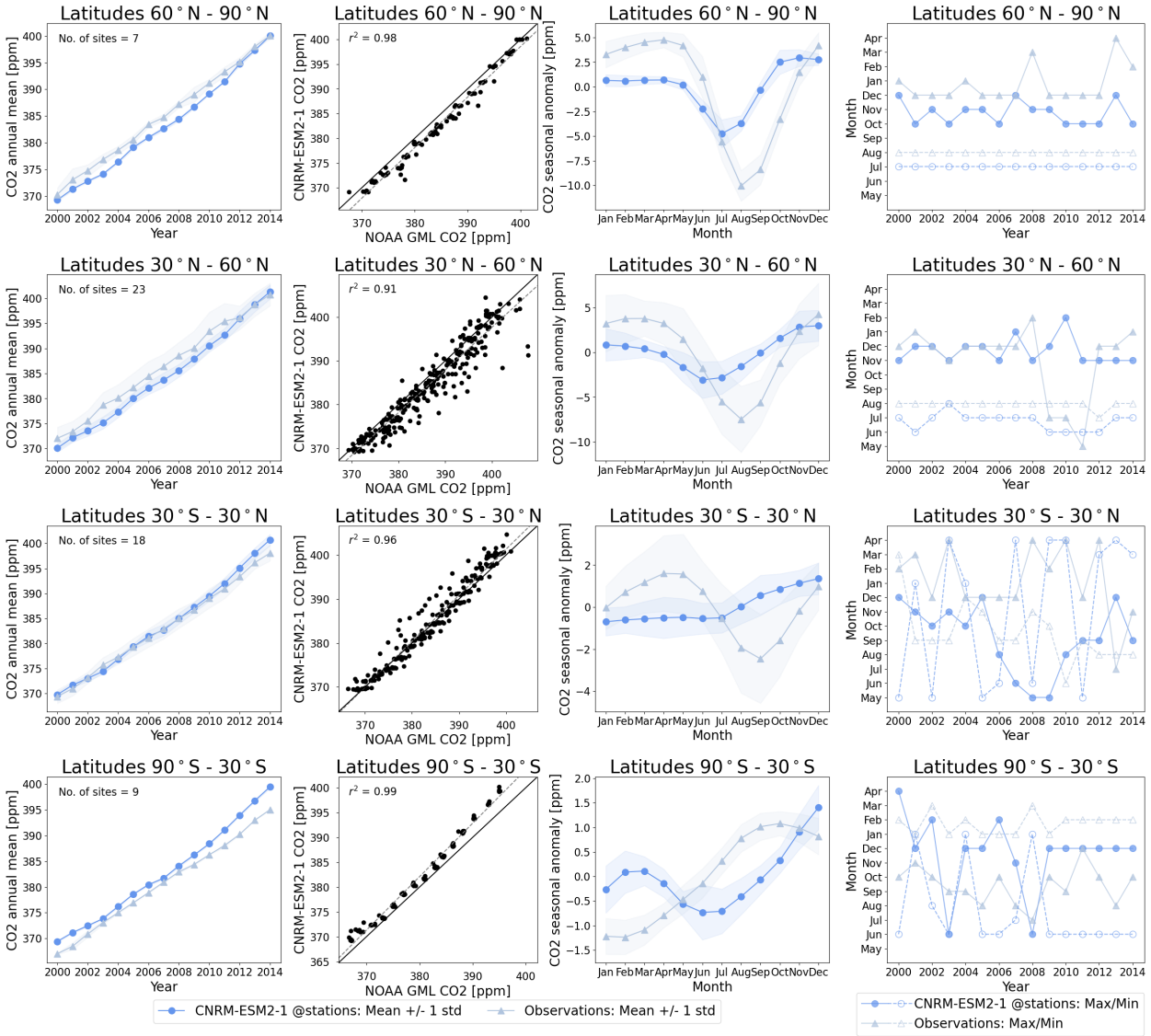


Fig. 127: Evaluation of surface concentration time series (monthly, seasonal, annual) of CO2 from CNRM-ESM2-1 esm-hist member r1i1p1f3 against the NOAA GML climatology from ground-based observations. The multiannual seasonal mean is calculated for the model data for the period 2000-2014.

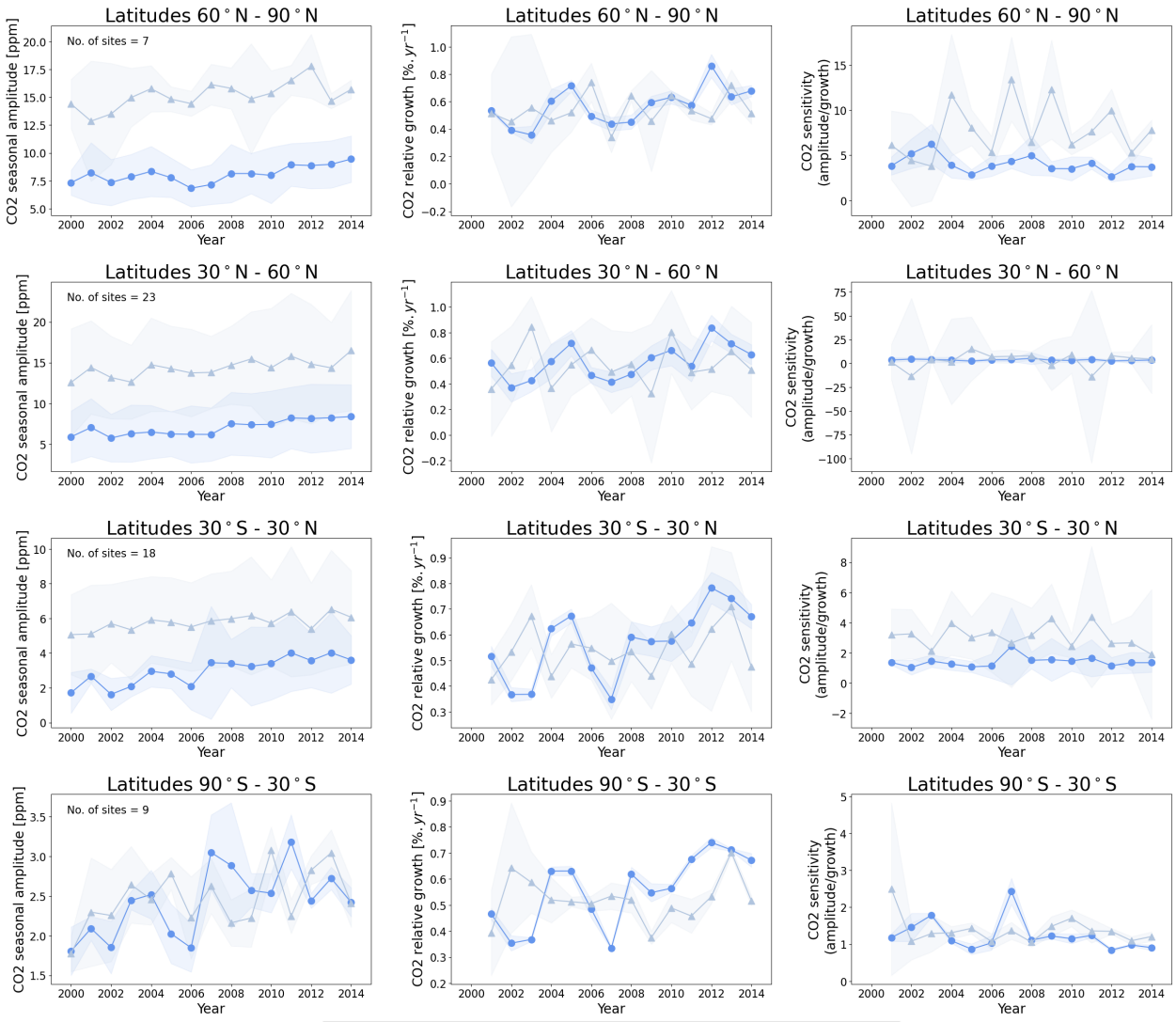


Fig. 128: Evaluation of surface concentration time series (amplitude, growth, sensitivity) of CO2 from CNRM-ESM2-1 esm-hist member r1i1p1f3 against the NOAA GML climatology from ground-based observations. The multiannual seasonal mean is calculated for the model data for the period 2000-2014.

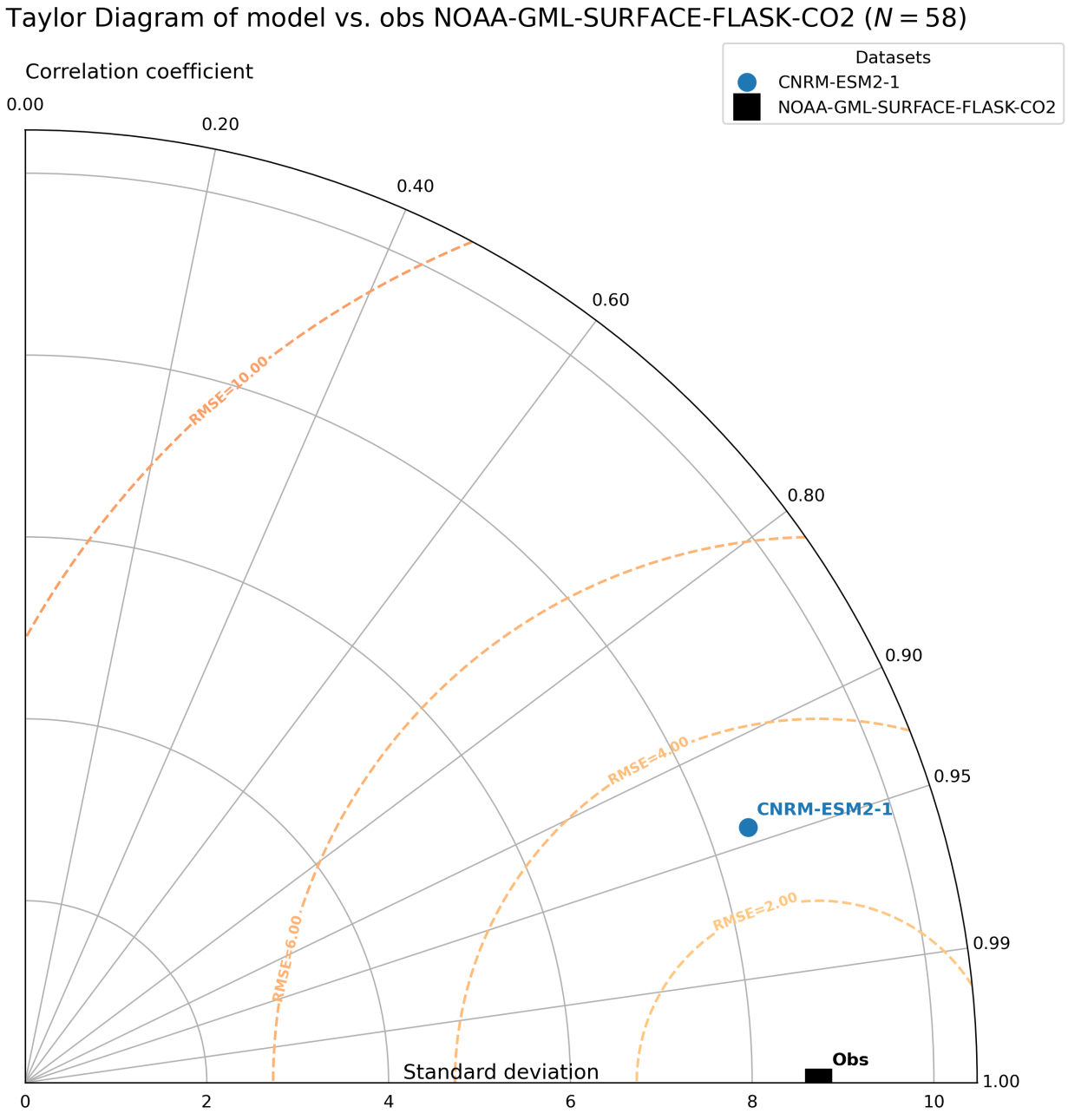


Fig. 129: Taylor diagram of surface concentration of CO2 from CNRM-ESM2-1 esm-hist member r1i1p1f3 against the NOAA GML climatology from ground-based observations. The multiannual seasonal mean is calculated for the model data for the period 2000-2014.

- rmse_threshold: rmse threshold for selecting similar WT pairs, only needed if automatic_slwt==True and predefined_slwt==False. default=0.002
- plotting: if true, create plots of means, anomalies and std for psl, tas, prcp
- automatic_slwt: if true, automatically combine WT with similar precipitation patterns over specified area (via thresholds of correlation and rmse OR via predefined_slwt)
- predefined_slwt: dictionary of mappings between weathertypes

Note

predefined_slwt can be a dictionary where keys are slwt and the values are arrays of lwt OR where keys are lwt and values are slwt

Required settings for variables

Optional settings for variables

Required settings for preprocessor

Optional settings for preprocessor

Color tables

Variables

- psl (atmos, day, time longitude latitude)
- tas (atmos, day, time longitude latitude)
- tp (atmos, day, time longitude latitude)
- pr (atmos, day, time longitude latitude)

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.

This recipe currently only works with the following reanalysis and observation datasets:

- E-OBS: European Climate Assessment & Dataset gridded daily precipitation sum
- ERA5: ECMWF reanalysis

References

- Maraun, D., Truhetz, H., & Schaffer, A. (2021). Regional climate model biases, their dependence on synoptic circulation biases and the potential for bias adjustment: A process-oriented evaluation of the Austrian regional climate projections. Journal of Geophysical Research: Atmospheres, 126, e2020JD032824. https://doi.org/10.1029/2020JD032824
- Jones, P.D., Hulme, M. and Briffa, K.R. (1993), A comparison of Lamb circulation types with an objective classification scheme. Int. J. Climatol., 13: 655-663. https://doi.org/10.1002/joc.3370130606

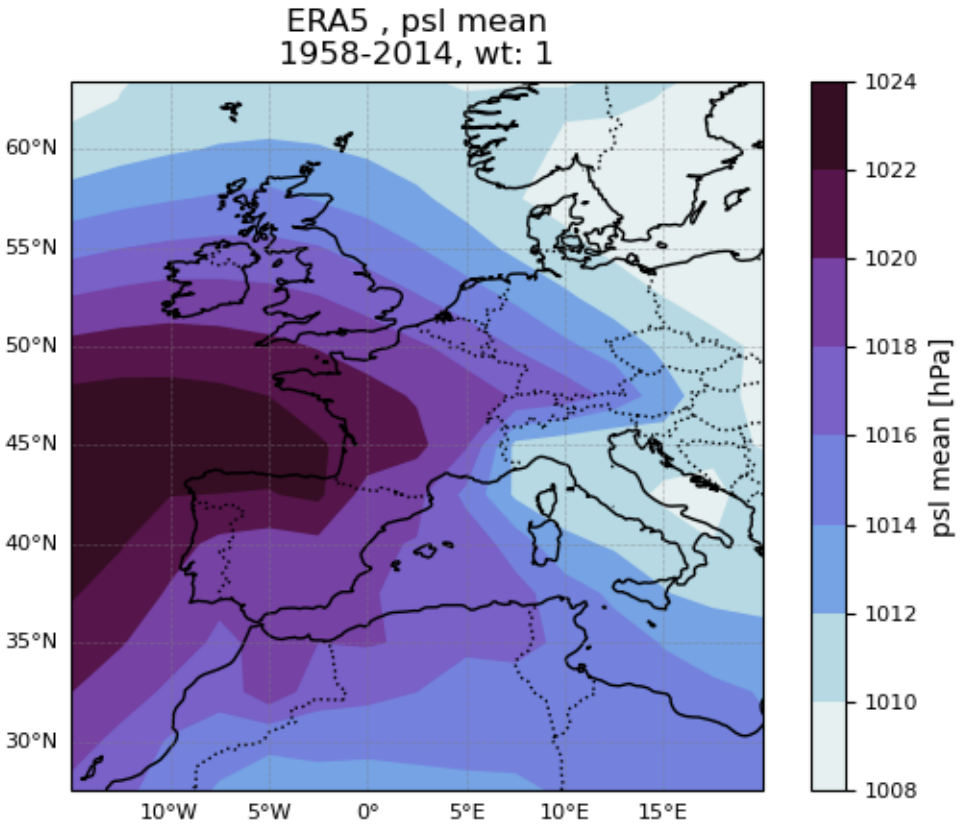


Fig. 130: PSL mean map of Lamb WT 1 for ERA5.

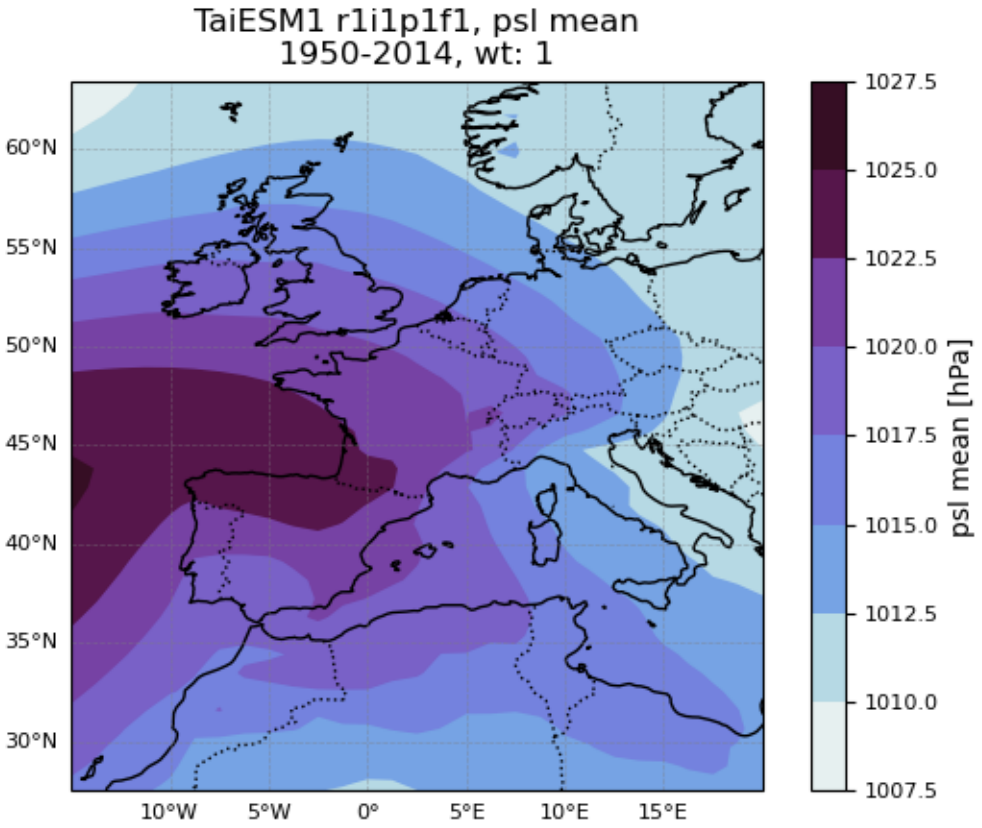


Fig. 131: PSL mean map of Lamb WT 1 for TaiESM1.

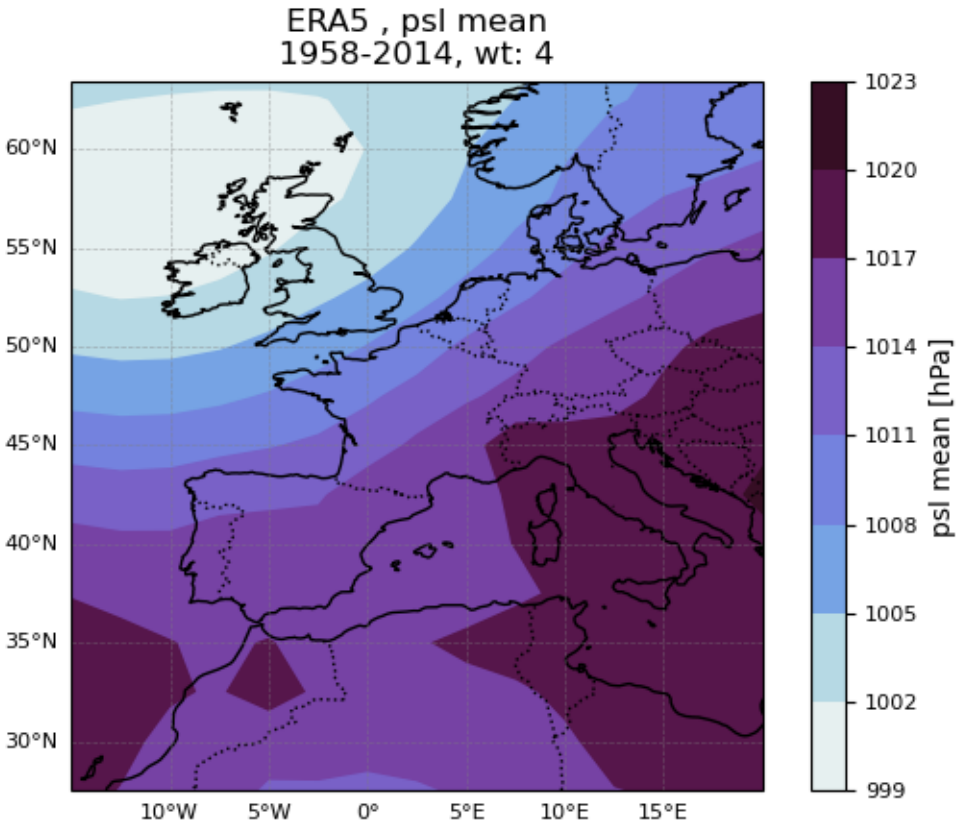


Fig. 132: PSL mean map of slwt_EOBS 4 for ERA5 (in this case combined Lamb WT 24 and 23).

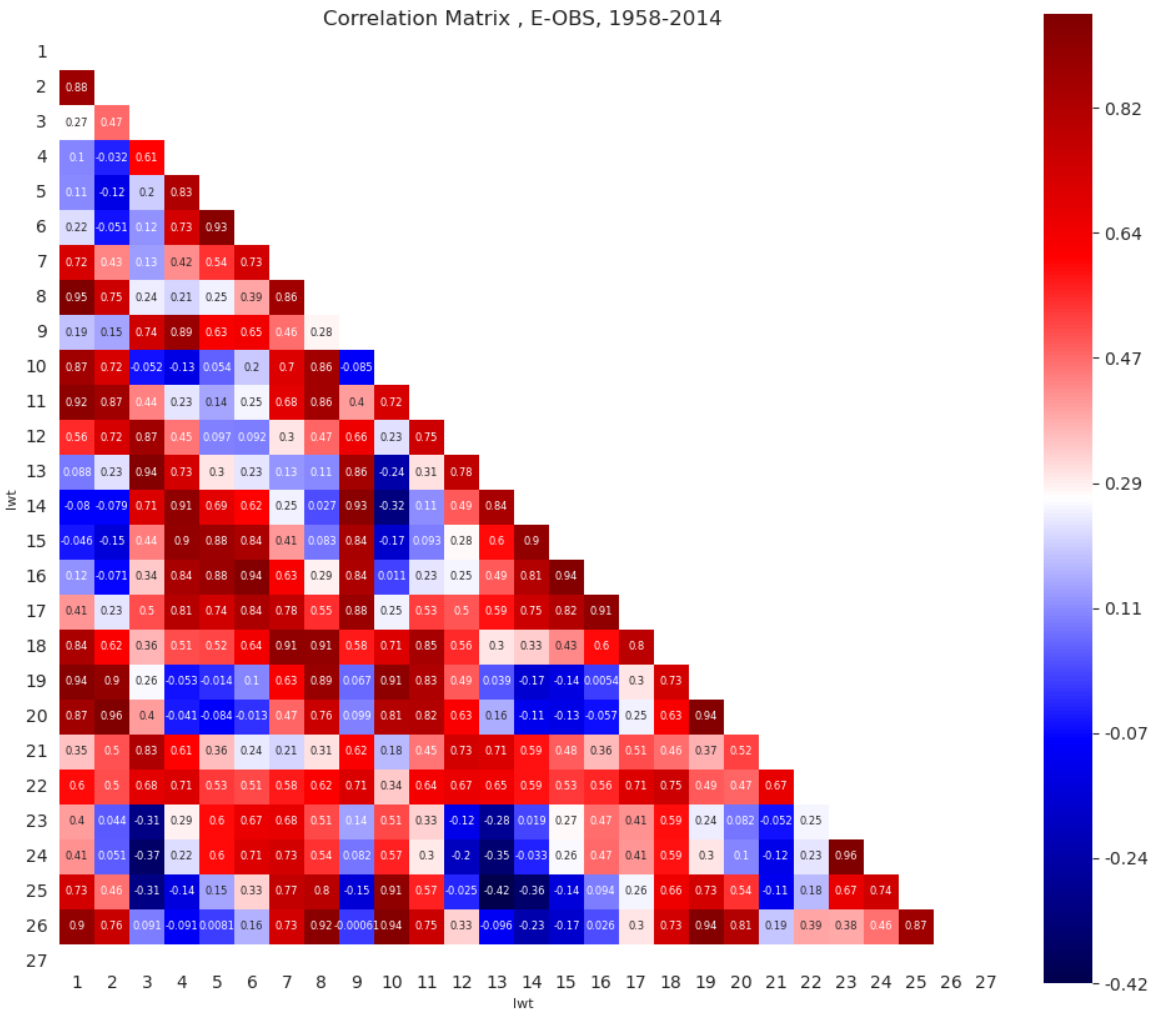


Fig. 133: Heatmap of correlation values for Lamb WTs 1-27.

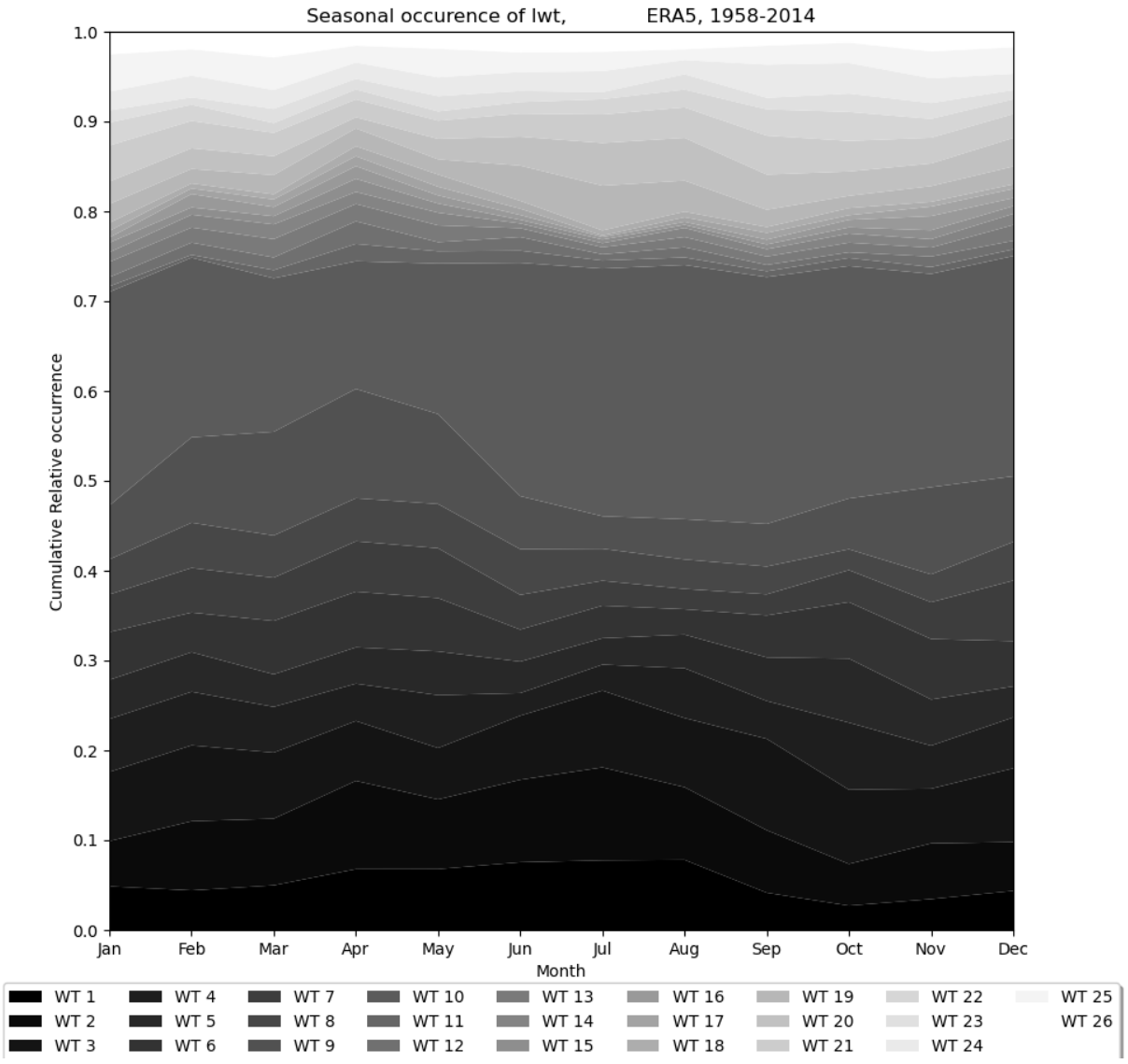


Fig. 134: Stackplot of seasonal relative occurrences of each WT for ERA5.

Example plots

5.3 Climate metrics

5.3.1 Performance metrics for essential climate parameters

Note

Some of the results of this diagnostics can also be reproduced utilizing python diagnostics: Portrait plot: *Portrait plot* Monitoring: *Monitor*

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).

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_perfmetrics_CMIP5.yml
- recipe_perfmetrics_CMIP5_4cds.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.

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: displays the global avaerage of the input field as string at the top-right of lat-lon plots (default: False)
- annots: choose the annotation style, e.g. `alias` which would display the alias of the dataset as title (applies to plot_type zonal and cycle_zonal)
- 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)
- add_tropopause: adds an outline of a climatological tropopause to the zonal plot (default: False)

Special optional plot configurations

It is possible to make some specific customizations to the plots (zonal only).

This includes for example specific tickmark labels of the axes.

Those special customizations can be done by adding ncl plotting resources combined with prefix `res_` as optional settings of the main script in the recipe.

Note that this requires to be familiar with the ncl plotting routines for pressure vs height plots (https://www.ncl.ucar.edu/Document/Graphics/Interfaces/gsn_csm_pres_hgt.shtml) and the corresponding resources.

The following shows an example on customizing the latitude tickmarks so that a degree sign and and empty space is used for the labels:

```
# copernicus style of latitude tickmarks
res_tmXBMode: "Explicit"
res_tmXBValues: [-60, -30, 0, 30, 60]
res_tmXBLabels: ["60~F35~J~F21~ S", "30~F35~J~F21~ S", "0~F35~J", "30~F35~J~F21~ N",
↪ "60~F35~J~F21~ N"]
```

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.

1. 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 triangle for values outside range
- label_hi: adds upper triangle 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

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)

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 reformatitng. For non-obs4MIPs data use *esmvaltool data info DATASET* or see headers of cmorization scripts (in /esmvaltool/cmorizers/data/formatters/datasets/) for downloading and processing instructions.

1. recipe_perfmetrics_CMIP5.yml
    - AIRS (hus - obs4MIPs)
    - CERES-EBAF (rlut, rlutcs, rsut, rsutcs - obs4MIPs)
    - ERA-Interim (tas, ta, ua, va, zg, hus - esmvaltool/cmorizers/data/formatters/datasets/era-interim.py)
    - ESACCI-AEROSOL (od550aer, od870aer, od550abs, od550lt1aer - esmvaltool/cmorizers/data/formatters/datasets/esacci-aerosol.ncl)
    - ESACCI-CLOUD (clt - esmvaltool/cmorizers/data/formatters/datasets/esacci-cloud.ncl)
    - ESACCI-OZONE (toz - esmvaltool/cmorizers/data/formatters/datasets/esacci-ozone.ncl)
    - ESACCI-SOILMOISTURE (sm - esmvaltool/cmorizers/data/formatters/datasets/esacci_soilmoisture.ncl)
    - ESACCI-SST (ts - esmvaltool/ucmorizers/data/formatters/datasets/esacci-sst.py)
    - GPCP-SG (pr - obs4MIPs)
    - HadISST (ts - esmvaltool/cmorizers/data/formatters/datasets/hadisst.ncl)
    - MODIS (od550aer - esmvaltool/cmorizers/data/formatters/datasets/modis.ncl)
    - NCEP-NCAR-R1 (tas, ta, ua, va, zg - esmvaltool/cmorizers/data/formatters/datasets/ncep_ncar_r1.py)
    - NIWA-BS (toz - esmvaltool/cmorizers/data/formatters/datasets/niwa_bs.ncl)
    - PATMOS-x (clt - esmvaltool/cmorizers/data/formatters/datasets/patmos_x.ncl)
2. recipe_perfmetrics_land_CMIP5.yml
    - CERES-EBAF (rlus, rlds, rsus, rsds - obs4MIPs)
    - ESACCI-SOILMOISTURE (sm - esmvaltool/cmorizers/data/formatters/datasets/esacci_soilmoisture.ncl)
    - FLUXCOM (gpp - esmvaltool/cmorizers/data/formatters/datasets/fluxcom.py)
    - JMA-TRANSCOM (nbp, fgco2 - esmvaltool/cmorizers/data/formatters/datasets/jma_transcom.py)
    - LAI3d (lai - esmvaltool/cmorizers/data/formatters/datasets/lai3g.py)
    - LandFlux-EVAL (et - esmvaltool/cmorizers/data/formatters/datasets/landflux_eval.py)
    - Landschuetzer2016 (fgco2 - esmvaltool/cmorizers/data/formatters/datasets/landschuetzer2016.py)

- MTE (gpp - esmvaltool/cmorizers/data/formatters/datasets/mte.py)

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 ozone and selected climate parameters in a set of EMAC simulations, Geosci. Model Dev., 8, 733, doi: 10.5194/gmd-8-733-2015 (2015).

Example plots

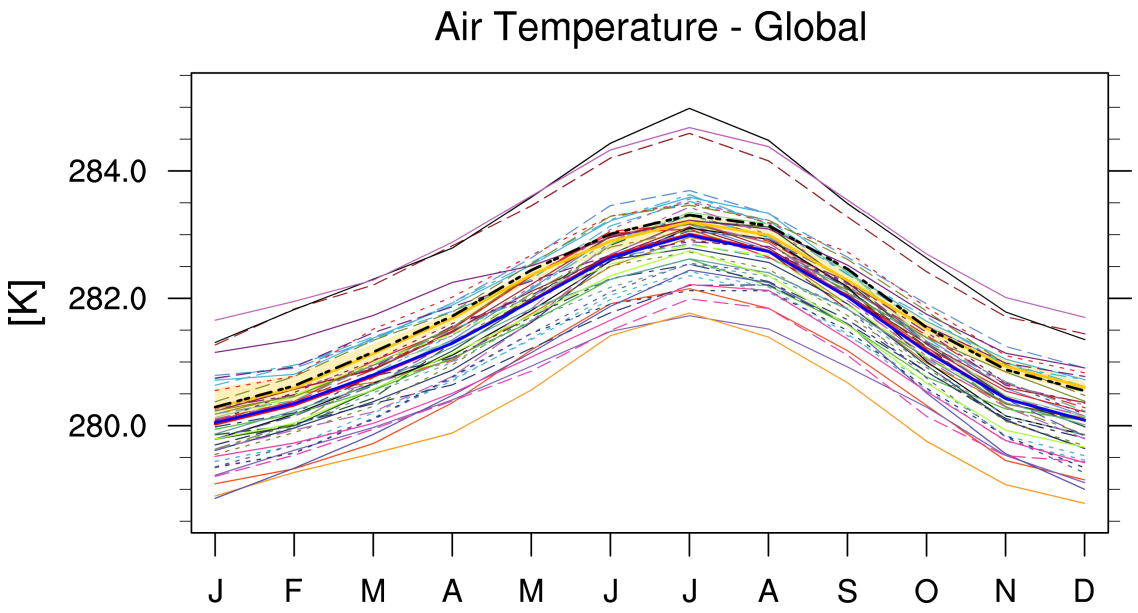


Fig. 135: Annual cycle of globally averaged temperature at 850 hPa (time period 1980-2005) for different CMIP5 models (historical simulation) (thin colored lines) in comparison to ERA-Interim (thick yellow line) and NCEP-NCAR-R1 (thick black dashed line) reanalysis data.

5.3.2 Single Model Performance Index (SMPI)

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.

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

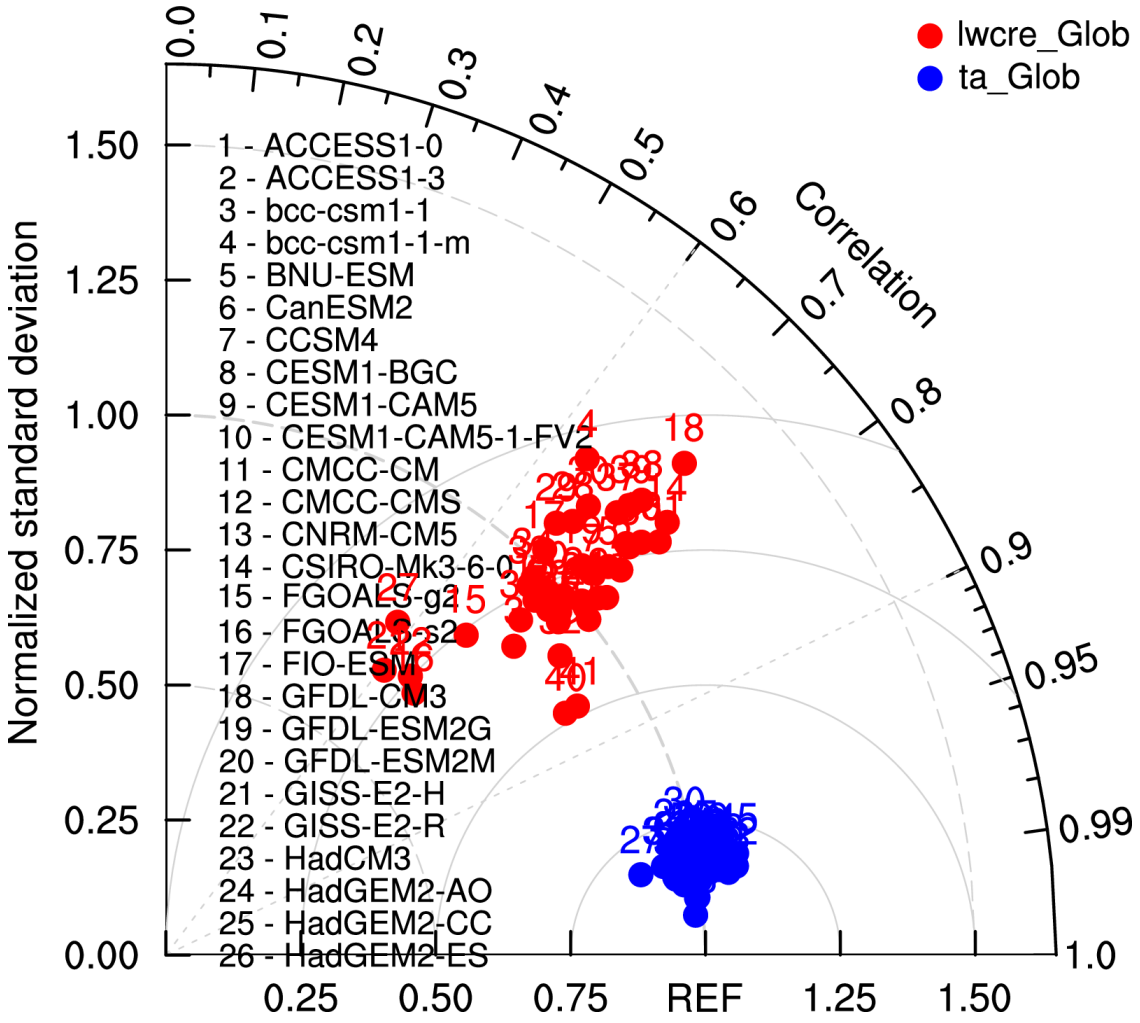


Fig. 136: Taylor diagram of globally averaged temperature at 850 hPa (ta) and longwave cloud radiative effect (lwcre) for different CMIP5 models (historical simulation, 1980-2005). Reference data (REF) are ERA-Interim for temperature (1980-2005) and CERES-EBAF (2001-2012) for longwave cloud radiative effect.

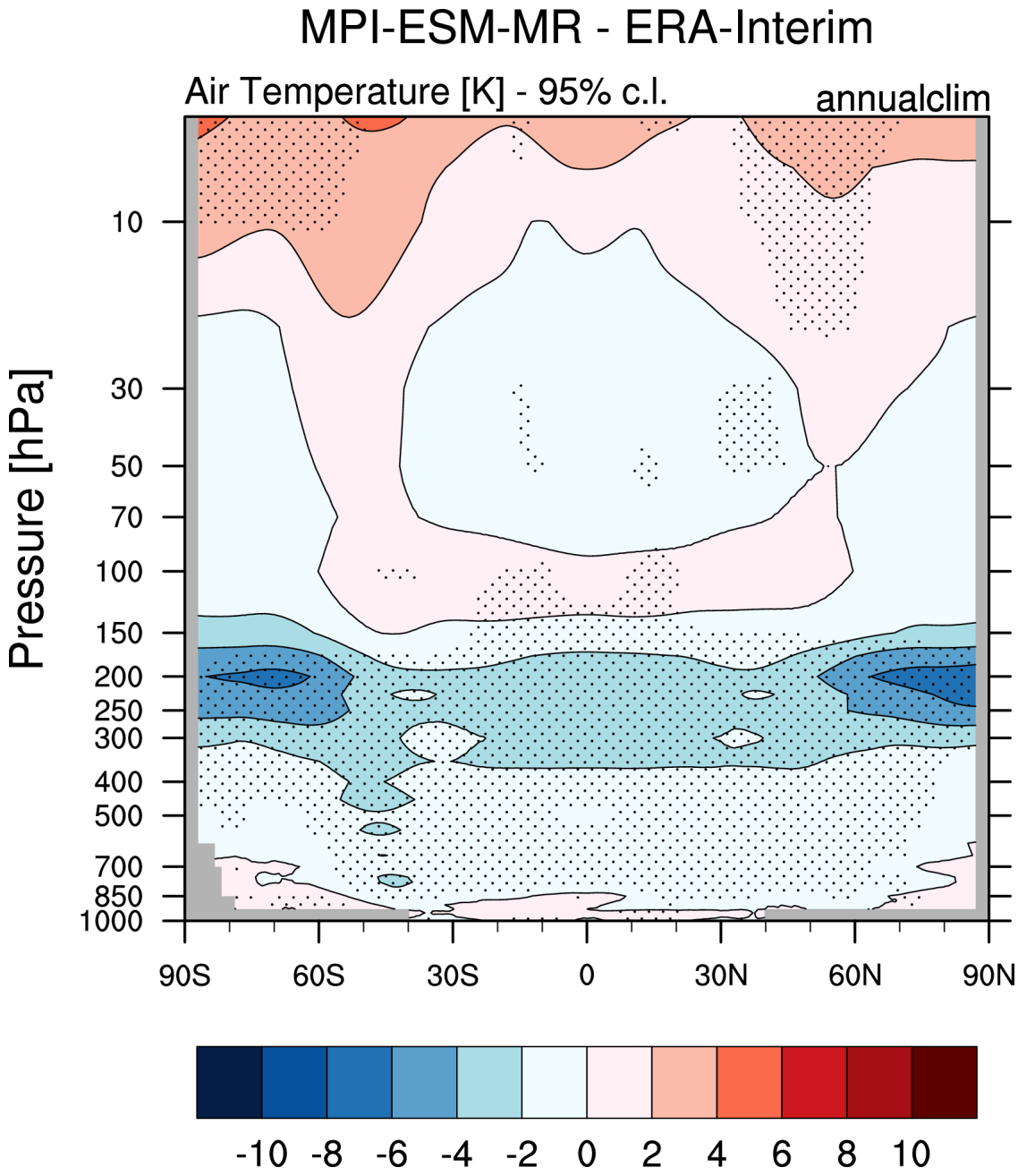


Fig. 137: Difference in annual mean of zonally averaged temperature (time period 1980-2005) between the CMIP5 model MPI-ESM-MR (historical simulation) and ERA-Interim. Stippled areas indicdate differences that are statistically significant at a 95% confidence level.

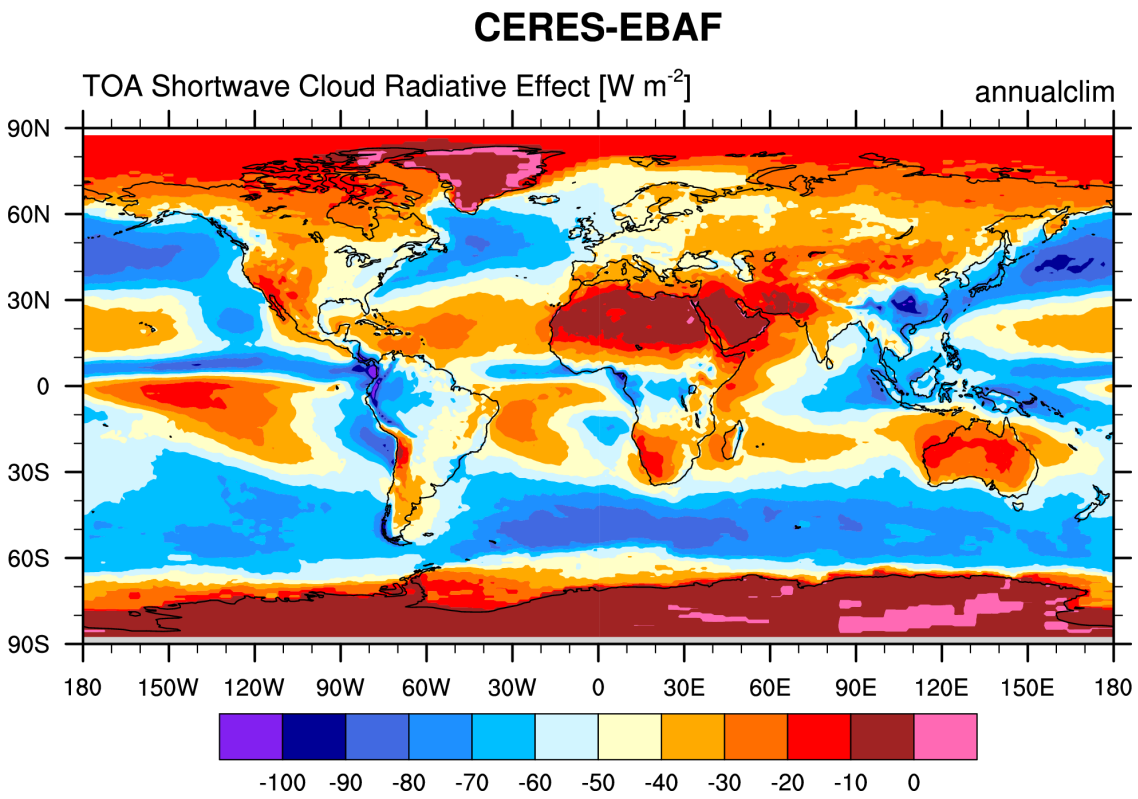


Fig. 138: Annual mean (2001-2012) of the shortwave cloud radiative effect from CERES-EBAF.

calculations, or the missing variable has to be removed from the diagnostics all together.

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 performance 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.

User settings

1. perfmetrics/main.ncl

   Required settings for script

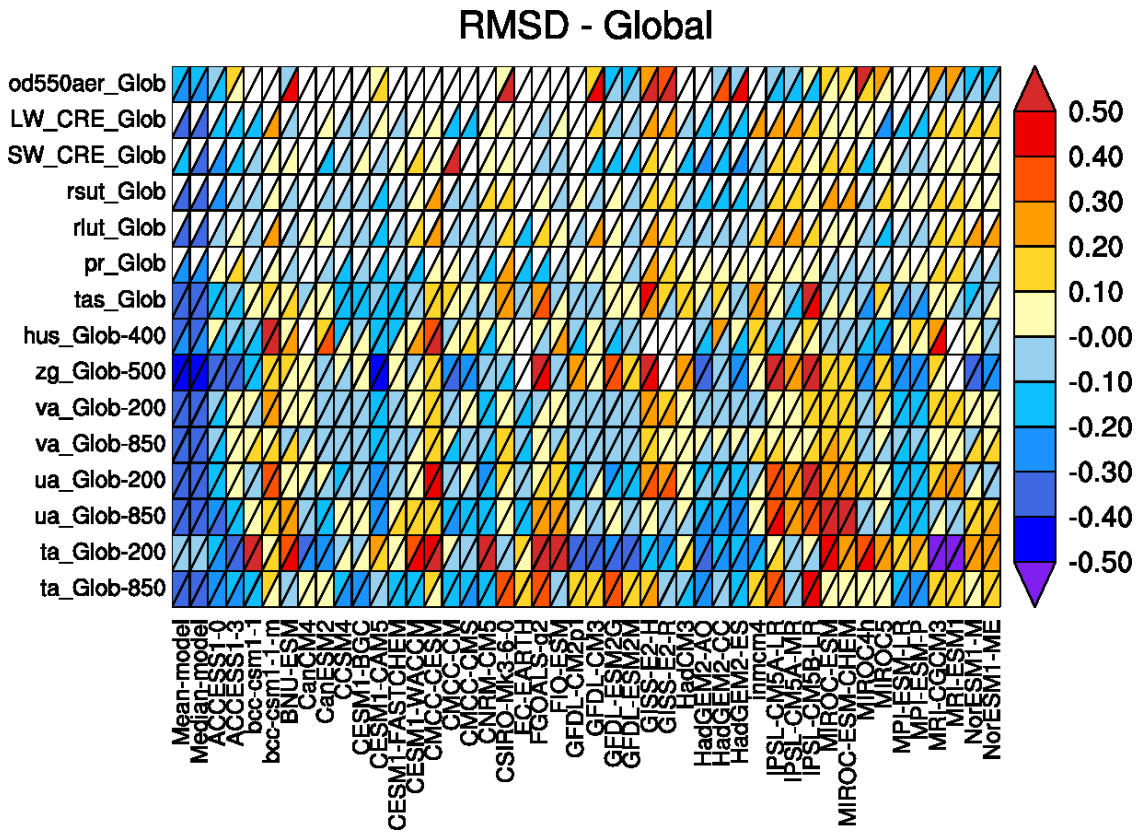


Fig. 139: Relative space-time root-mean-square deviation (RMSD) calculated from the climatological seasonal cycle of CMIP5 simulations. A relative performance is displayed, with blue shading indicating better and red shading indicating worse performance than the median of all model results. A diagonal 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.

- 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")

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)

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 reformatting. For non-obs4MIPs data use *esmvaltool data info DATASET* or see headers of cmorization scripts for downloading and processing instructions.

- ERA-Interim (hfds, hus, psl, ta, tas, tauu, tauv, ua, va - esmvaltool/data/formatters/datasets/era-interim.py)

- HadISST (sic, tos - esmvaltool/data/formatters/datasets/hadisst.ncl)
- GPCP-V2.2 (pr - obs4MIPs)

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.

Example plots

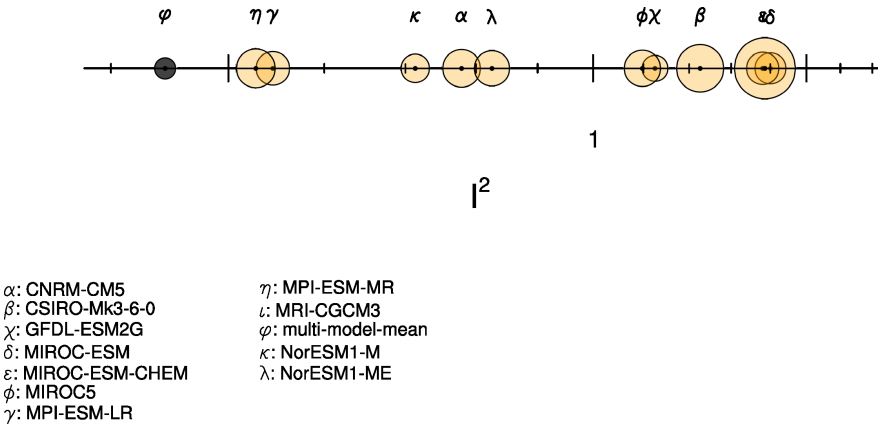


Fig. 140: Performance index I^2 for individual models (circles). Circle sizes indicate the length of the 95% confidence intervals. The black circle indicates the I^2 of the multi-model mean (similar to Reichler and Kim (2008), Figure 1).

5.4 Future projections

5.4.1 Climate model Weighting by Independence and Performance (ClimWIP)

Overview

Projections of future climate change are often based on multi-model ensembles of global climate models such as CMIP6. To condense the information from these models they are often combined into probabilistic estimates such as mean and a related uncertainty range (such as the standard deviation). However, not all models in a given multi-model ensemble are always equally 'fit for purpose' and it can make sense to weight models based on their ability to simulate observed quantities related to the target. In addition, multi-model ensembles, such as CMIP can contain several models based on a very similar code-base (sharing of components, only differences in resolution etc.) leading to complex inter-dependencies between the models. Adjusting for this by weighting models according to their independence helps to adjust for this.

This recipe implements the **Climate model Weighting by Independence and Performance (ClimWIP)** method. It is based on work by Knutti et al. (2017), Lorenz et al. (2018), Brunner et al. (2019), Merrifield et al. (2020), Brunner et al. (2020). Weights are calculated based on historical model performance in several metrics (which can be defined by the `performance_contributions` parameter) as well as by their independence to all the other models in the ensemble based on their output fields in several metrics (which can be defined by the `independence_contributions` parameter). These weights can be used in subsequent evaluation scripts (some of which are implemented as part of this diagnostic).

Note: this recipe is still being developed! A more comprehensive (yet older) implementation can be found on GitHub: https://github.com/lukasbrunner/ClimWIP

Using shapefiles for cutting scientific regions

To use shapefiles for selecting SREX or AR6 regions by name it is necessary to download them, e.g., from the sources below and reference the file using the *shapefile* parameter. This can either be a absolute or a relative path. In the example recipes they are stored in a subfolder *shapefiles* in the configuration option `auxiliary_data_dir`.

SREX regions (AR5 reference regions): http://www.ipcc-data.org/guidelines/pages/ar5_regions.html

AR6 reference regions: https://github.com/SantanderMetGroup/ATLAS/tree/v1.6/reference-regions

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- `recipe_climwip_test_basic.yml`: Basic sample recipe using only a few models
- `recipe_climwip_test_performance_sigma.yml`: Advanced sample recipe for testing the perfect model test in particular
- `recipe_climwip_brunner2020esd.yml`: Slightly modified results for Brunner et al. (2020)

Archived recipes (stored in archive/legacy_recipes):

Note

These recipes have been retired and moved to the folder "archive". These recipes are only included for documentation purposes and not maintained any more.

- `recipe_climwip_brunner2019_med.yml`: Slightly modified results for one region from Brunner et al. (2019) (to change regions see below) [last known ESMValTool version supporting this recipe is version 2.11.0.]

Diagnostics are stored in esmvaltool/diag_scripts/weighting/climwip/

- `main.py`: Compute weights for each input dataset
- `calibrate_sigmas.py`: Compute the sigma values on the fly
- `core_functions.py`: A collection of core functions used by the scripts
- `io_functions.py`: A collection of input/output functions used by the scripts

Plot scripts are stored in esmvaltool/diag_scripts/weighting/

- `weighted_temperature_graph.py`: Show the difference between weighted and non-weighted temperature anomalies as time series.
- `weighted_temperature_map.py`: Show the difference between weighted and non-weighted temperature anomalies on a map.
- `plot_utilities.py`: A collection of functions used by the plot scripts.

User settings in recipe

1. Script `main.py`

   Required settings for script

   - `performance_sigma` xor `calibrate_performance_sigma`: If `performance_contributions` is given exactly one of the two has to be given. Otherwise they can be skipped or not set.
     - `performance_sigma`: float setting the shape parameter for the performance weights calculation (determined offline).

  - `calibrate_performance_sigma`: dictionary setting the performance sigma calibration. Has to contain at least the key-value pair specifying `target`: `variable_group`. Optional parameters for adjusting the calibration are not yet implemented. **Warning:** It is highly recommended to visually inspect the graphical output of the calibration to check if everything worked as intended. In case the calibration fails, the best performance sigma will still be indicated in the figure (see example Fig. 145 below) but not automatically picked - the user can decide to use it anyway by setting it in the recipe (not recommenced).

- `independence_sigma`: float setting the shape parameter for the independence weights calculation (determined offline). Can be skipped or not set if `independence_contributions` is skipped or not set. A on-the-fly calculation of the independence sigma is not yet implemented
- `performance_contributions`: dictionary where the keys represent the variable groups to be included in the performance calculation. The values give the relative contribution of each group, with 0 being equivalent to not including the group. Can be skipped or not set then weights will be based purely on model independence (this is mutually exclusive with `independence_contributions` being skipped or not set).
- `independence_contributions`: dictionary where the keys represent the variable groups to be included in the independence calculation. The values give the relative contribution of each group, with 0 being equivalent to not including the group. If skipped or not set weights will be based purely on model performance (this is mutually exclusive with `performance_contributions` being skipped or not set).
- `combine_ensemble_members`: set to true if ensemble members of the same model should be combined during the processing (leads to identical weights for all ensemble members of the same model). Recommended if running with many (>10) ensemble members per model. If set to false, the model independence weighting will still (partly) account for the (very high) dependence between members of the same model. The success of this will depend on the case and the selected parameters. See Merrifield et al. (2020) for an in-depth discussion.
- `obs_data`: list of project names to specify which are the observational data. The rest is assumed to be model data.

Required settings for variables * This script takes multiple variables as input as long as they're available for all models * `start_year`: provide the period for which to compute performance and independence. * `end_year`: provide the period for which to compute performance and independence. * `mip`: typically Amon * `preprocessor`: e.g., climatological_mean * `additional_datasets`: this should be `*obs_data` and is only needed for variables used in `performance_contributions`.

Required settings for preprocessor

- Different combinations of preprocessor functions can be used, but the end result should always be aggregated over the time dimension, i.e. the input for the diagnostic script should be 2d (lat/lon).

Optional settings for preprocessor

- `extract_region` or `extract_shape` can be used to crop the input data.
- `extract_season` can be used to focus on a single season.
- different climate statistics can be used to calculate mean, (detrended) std_dev, or trend.

2. Script `weighted_temperature_graph.py`

   Required settings for script

   - `ancestors`: must include weights from previous diagnostic
   - `weights`: the filename of the weights: 'weights.nc'

- `settings`: a list of plot settings: `start_year` (integer), `end_year` (integer), `central_estimate` ('mean' or integer between 0 and 100 giving the percentile), `lower_bound` (integer between 0 and 100), `upper_bound` (integer between 0 and 100)

Required settings for variables

- This script only takes temperature (tas) as input
- `start_year`: provide the period for which to plot a temperature change graph.
- `end_year`: provide the period for which to plot a temperature change graph.
- `mip`: typically Amon
- `preprocessor`: temperature_anomalies

Required settings for preprocessor

- Different combinations of preprocessor functions can be used, but the end result should always be aggregated over the latitude and longitude dimensions, i.e. the input for the diagnostic script should be 1d (time).

Optional settings for preprocessor

- Can be a global mean or focus on a point, region or shape
- Anomalies can be calculated with respect to a custom reference period
- Monthly, annual or seasonal average/extraction can be used

3. Script `weighted_temperature_map.py`

Required settings for script

- `ancestors`: must include weights from previous diagnostic
- `weights`: the filename of the weights: 'weights_combined.nc'

Optional settings for script

- `model_aggregation`: how to aggregate the models: mean (default), median, integer between 0 and 100 representing a percentile
- `xticks`: positions to draw xticks at
- `yticks`: positions to draw yticks at

Required settings for variables

- This script takes temperature (tas) as input
- `start_year`: provide the period for which to plot a temperature change graph.
- `end_year`: provide the period for which to plot a temperature change graph.
- `mip`: typically Amon
- `preprocessor`: temperature_anomalies

Optional settings for variables

- A second variable is optional: temperature reference (tas_reference). If given, maps of temperature change to the reference are drawn, otherwise absolute temperatures are drawn.
- tas_reference takes the same fields as tas

Updating the Brunner et al. (2019) recipe for new regions

`recipe_climwip_brunner2019_med.yml` demonstrates a very similar setup to Brunner et al. (2019) but only for one region (the Mediterranean). To calculated weights for other regions the recipe needs to be updated in two places:

```
extract_shape:
   shapefile: shapefiles/srex.shp
   decomposed: True
   method: contains
   crop: true
   ids:
     - 'South Europe/Mediterranean [MED:13]'
```

The `ids` field takes any valid SREX region key or any valid AR6 region key (depending on the shapefile). Note that this needs to be the full string here (not the abbreviation).

The sigma parameters need to be set according to the selected region. The sigma values for the regions used in Brunner et al. (2019) can be found in table 1 of the paper.

```
performance_sigma: 0.546
independence_sigma: 0.643
```

Warning: if a new region is used the sigma values should be recalculated! This can be done by commenting out the sigma values (lines above) and commenting in the blocks defining the target of the weighting:

```
CLIM_future:
   short_name: tas
   start_year: 2081
   end_year: 2100
   mip: Amon
   preprocessor: region_mean
```

as well as

```
calibrate_performance_sigma:
   target: CLIM_future
```

In this case ClimWIP will attempt to perform an on-the-fly perfect model test to estimate the lowest performance sigma (strongest weighting) which does not lead to overconfident weighting. **Important:** the user should always check the test output for unusual behaviour. For most cases the performance sigma should lie around 0.5. In cases where the perfect model test fails (no appropriate performance sigma can be found) the test will still produce graphical output before raising a ValueError. The user can then decide to manually set the performance sigma to the most appropriate value (based on the output) - **this is not recommended** and should only be done with care! The perfect model test failing can be a hint for one of the following: (1) not enough models in the ensemble for a robust distribution (normally >20 models should be used) or (2) the performance metrics used are not relevant for the target.

An on-the-fly calibration for the independence sigma is not yet implemented. For most cases we recommend to use the same setup as in Brunner et al. (2020) or Merrifield et al. (2020) (global or hemispherical temperature and sea level pressure climatologies as metrics and independence sigma values between 0.2 and 0.5).

Warning: if a new region or target is used the provided metrics to establish the weights might no longer be appropriate. Using unrelated metrics with no correlation and/or physical relation to the target will reduce the skill of the weighting and ultimately render it useless! In such cases the perfect model test might fail. This means the performance metrics should be updated.

Brunner et al. (2020) recipe and example independence weighting

`recipe_climwip_brunner2020esd.yml` implements the weighting used in Brunner et al. (2020). Compared to the paper there are minor differences due to two models which had to be excluded due to errors in the ESMValTool pre-processor: CAMS-CSM1-0 and MPI-ESM1-2-HR (r2) as well as the use of only one observational dataset (ERA5).

The recipe uses an additional step between pre-processor and weight calculation to calculate anomalies relative to the global mean (e.g., tas_ANOM = tas_CLIM - global_mean(tas_CLIM)). This means we do not use the absolute temperatures of a model as performance criterion but rather the horizontal temperature distribution (see Brunner et al. 2020 for a discussion).

This recipe also implements a somewhat general independence weighting for CMIP6. In contrast to model performance (which should be case specific) model independence can largely be seen as only dependent on the multi-model ensemble in use but not the target variable or region. This means that the configuration used should be valid for similar subsets of CMIP6 as used in this recipe:

```
combine_ensemble_members: true
independence_sigma: 0.54
independence_contributions:
    tas_CLIM_i: 1
    psl_CLIM_i: 1
```

Note that this approach weights ensemble members of the same model with a 1/N independence scaling (combine_ensemble_members: true) as well as different models with an output-based independence weighting. Different approaches to handle ensemble members are discussed in Merrifield et al. (2020). Note that, unlike for performance, climatologies are used for independence (i.e., the global mean is **not** removed for independence). **Warning:** Using only the independence weighting without any performance weighting might not always lead to meaningful results! The independence weighting is based on model output, which means that if a model is very different from all other models as well as the observations it will get a very high independence weight (and also total weight in absence of any performance weighting). This might not reflect the actual independence. It is therefore recommended to use weights based on both independence and performance for most cases.

Variables

- pr (atmos, monthly mean, longitude latitude time)
- tas (atmos, monthly mean, longitude latitude time)
- psl (atmos, monthly mean, longitude latitude time)
- rsus, rsds, rlus, rlds, rsns, rlns (atmos, monthly mean, longitude latitude time)
- more variables can be added if available for all datasets.

Observations and reformat scripts

Observation data is defined in a separate section in the recipe and may include multiple datasets.

References

- Brunner et al. (2020), Earth Syst. Dynam., 11, 995-1012
- Merrifield et al. (2020), Earth Syst. Dynam., 11, 807-834
- Brunner et al. (2019), Environ. Res. Lett., 14, 124010
- Lorenz et al. (2018), J. Geophys. Res.: Atmos., 9, 4509-4526
- Knutti et al. (2017), Geophys. Res. Lett., 44, 1909-1918

Example plots

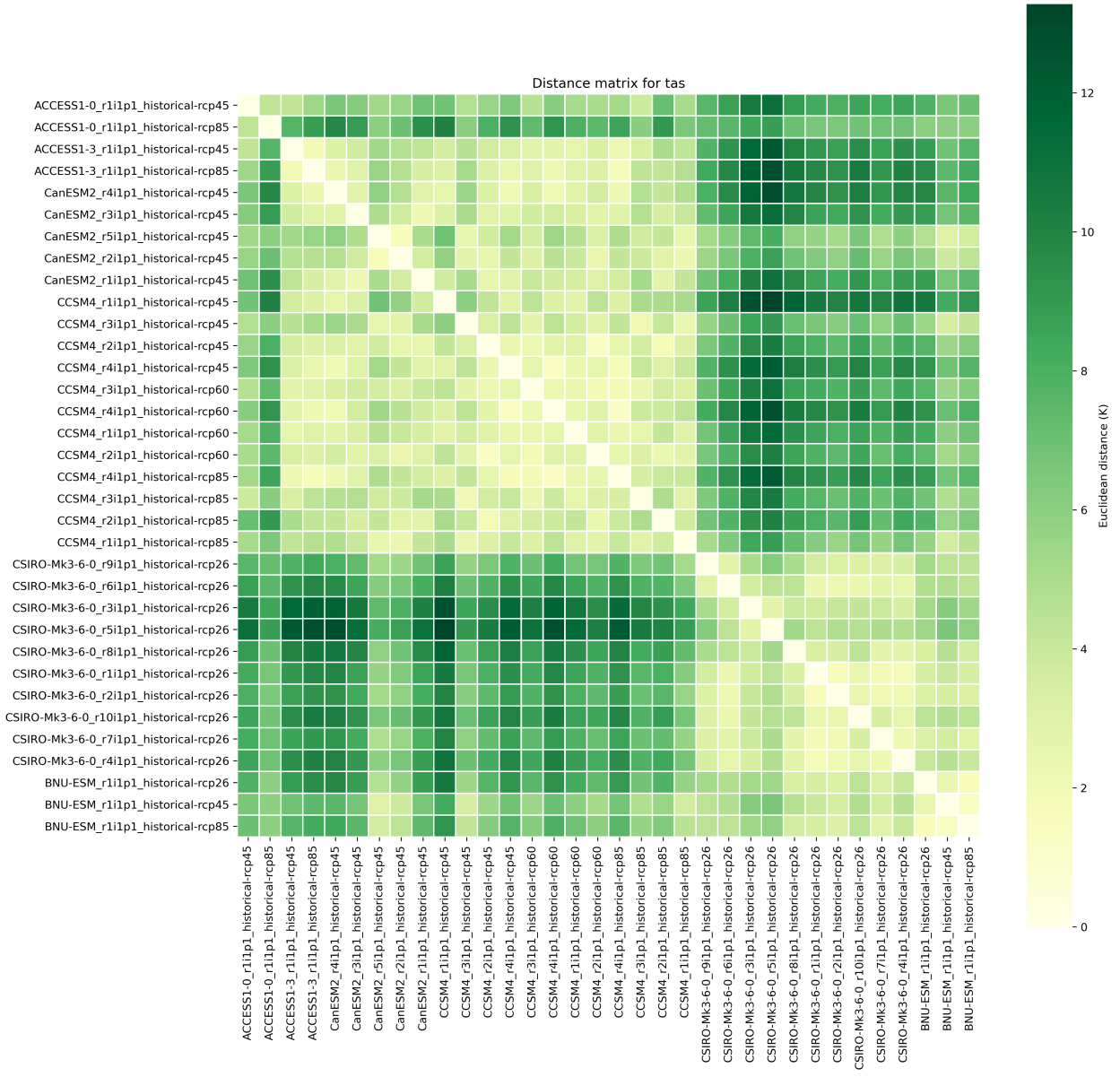


Fig. 141: Distance matrix for temperature, providing the independence metric.

5.4.2 Constraining future Indian Summer Monsoon projections with the present-day precipitation over the tropical western Pacific

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.

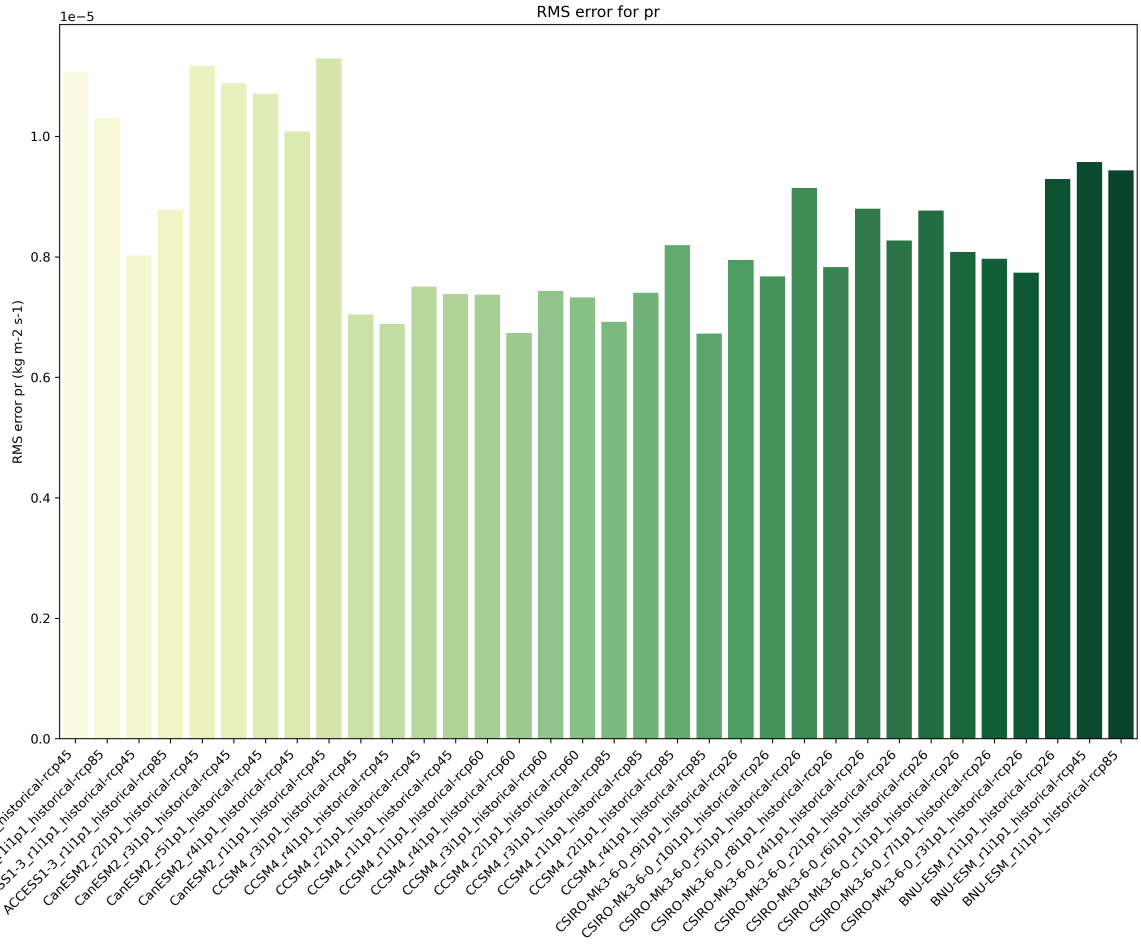


Fig. 142: Distance of preciptation relative to observations, providing the performance metric.

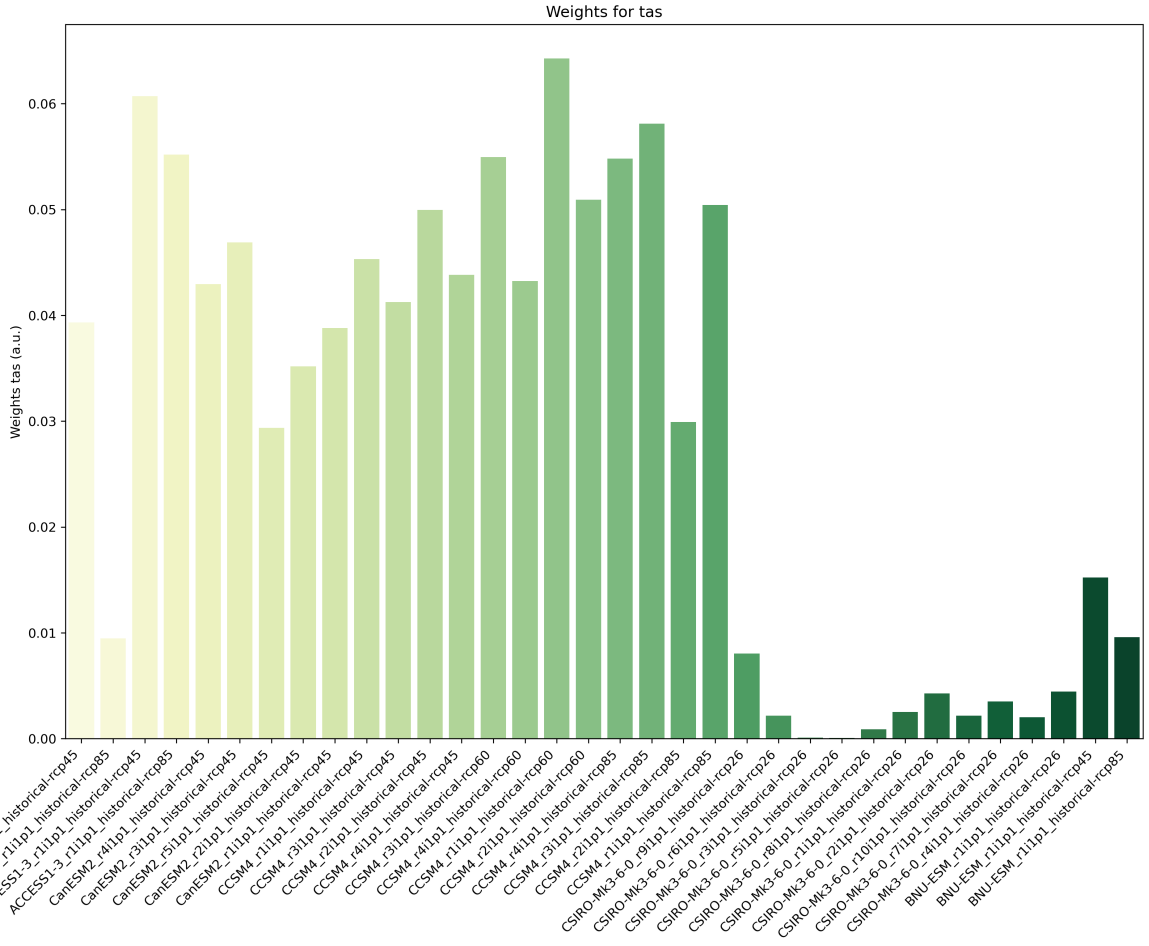


Fig. 143: Weights determined by combining independence and performance metrics for tas.

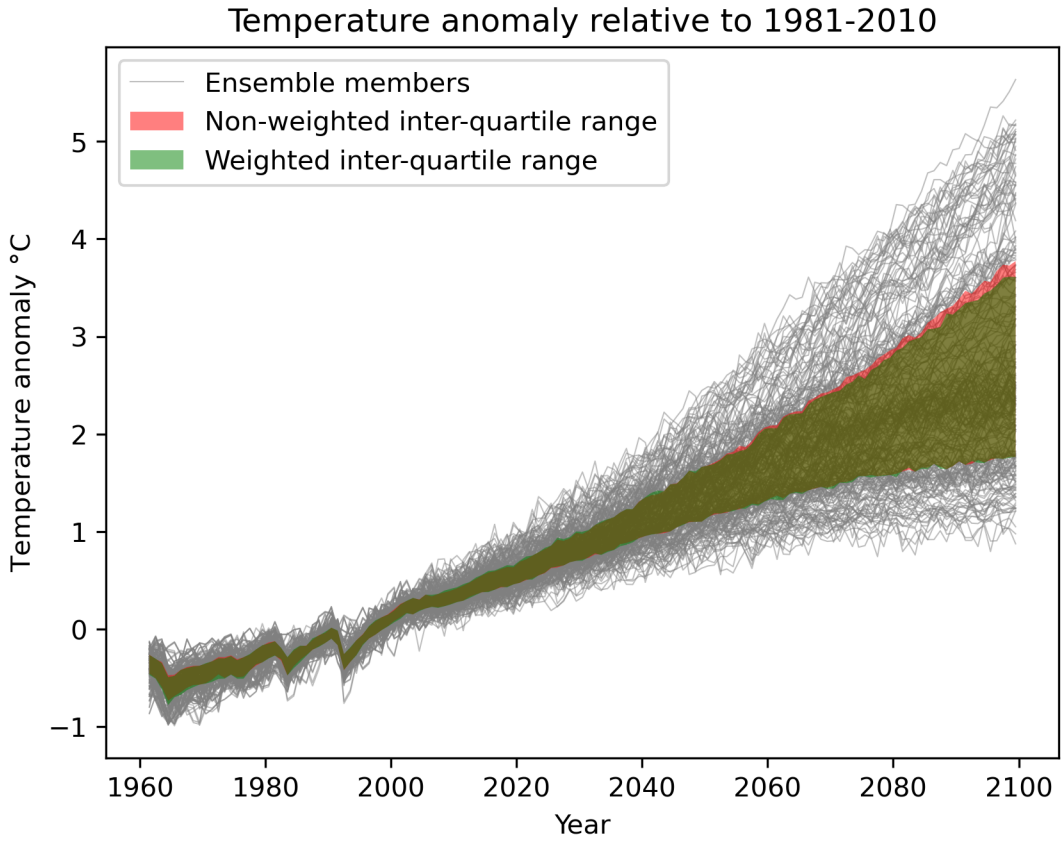


Fig. 144: Interquartile range of temperature anomalies relative to 1981-2010, weighted versus non-weighted.

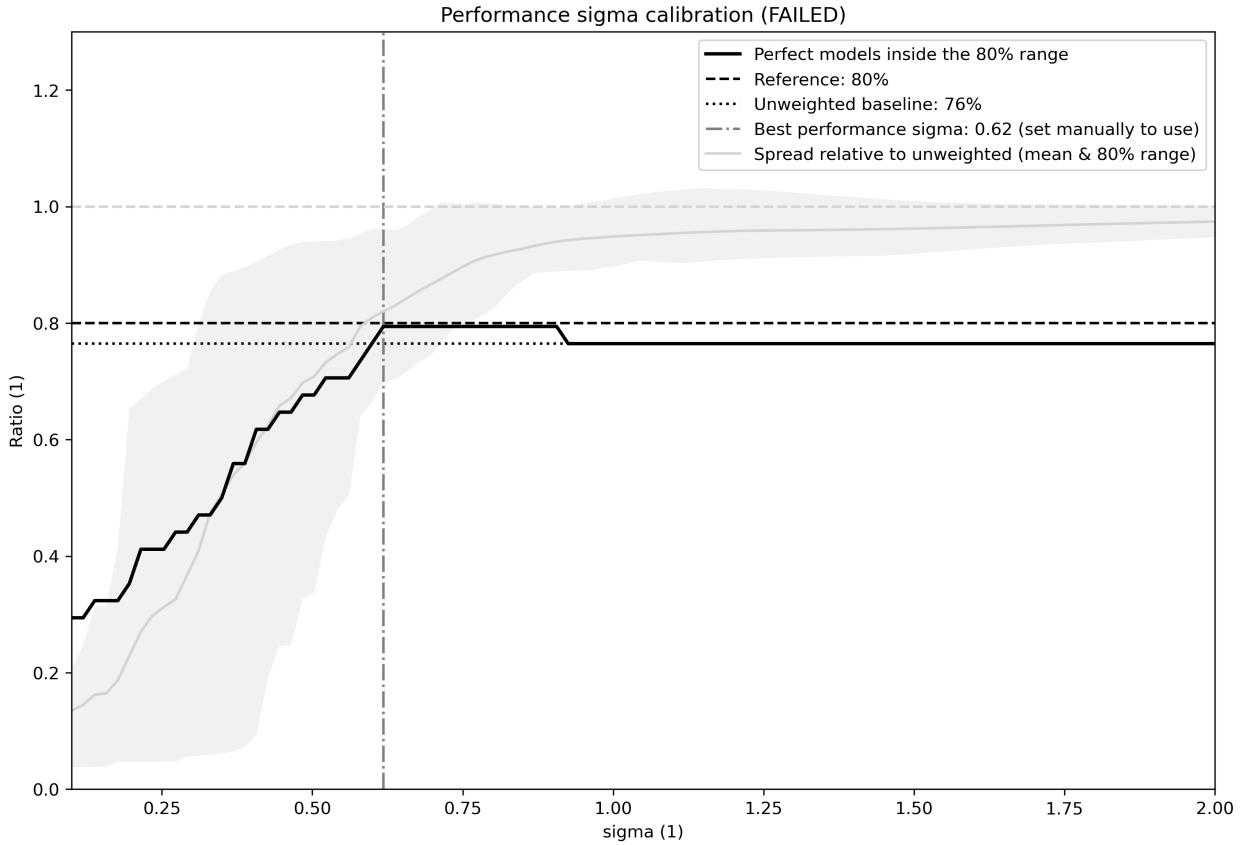


Fig. 145: Performance sigma calibration: The thick black line gives the reliability (c.f., weather forecast verification) which should reach at least 80%. The thick grey line gives the mean change in spread between the unweighted and weighted 80% ranges as an indication of the weighting strength (if it reaches 1, the weighting has no effect on uncertainty). The smallest sigma (i.e., strongest weighting) which is not overconfident (reliability >= 80%) is selected. If the test fails (like in this example) the smallest sigma which comes closest to 80% will be indicated in the legend (but NOT automatically selected).

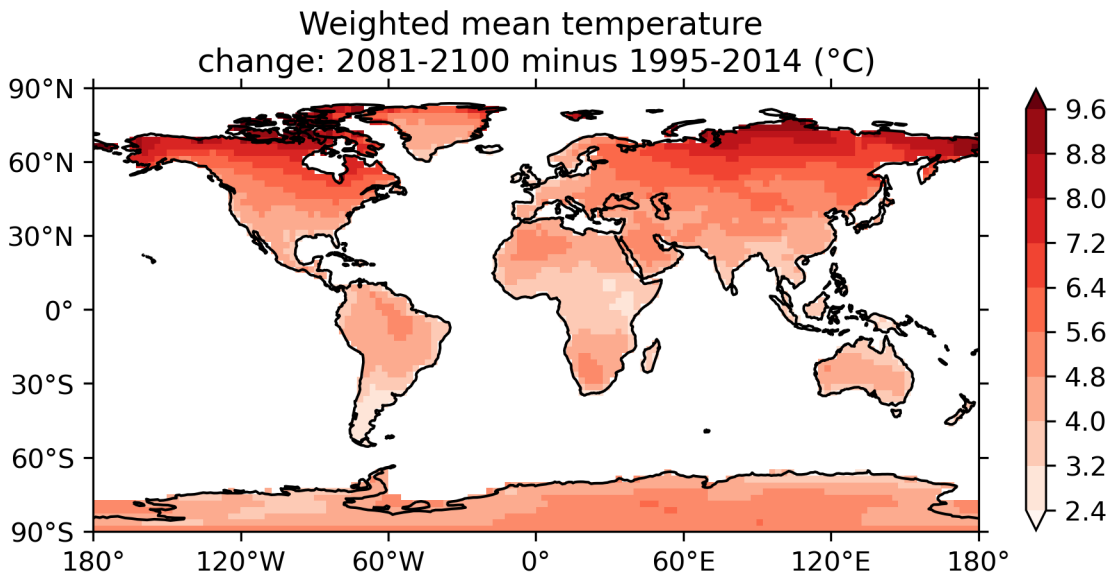


Fig. 146: Map of weighted mean temperature change 2081-2100 relative to 1995-2014

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_li17natcc.yml

Diagnostics are stored in diag_scripts/

- emergent_constraints/lif1f2.py

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.

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)

Observations and reformat scripts

None

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

Example plots

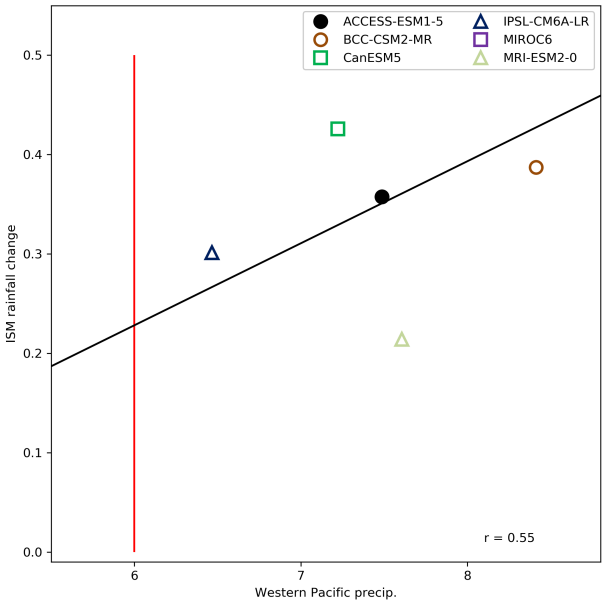


Fig. 147: Scatter plot of the simulated tropical western Pacific precipitation (mm d^{-1}) versus projected average ISM (Indian Summer Monsoon) rainfall changes under the ssp585 scenario. The red line denotes the observed present-day western Pacific precipitation and the inter-model correlation (r) is shown. (CMIP6).

5.4.3 Context for interpreting equilibrium climate sensitivity and transient climate response from the CMIP6 Earth system models

Overview

This recipe reproduces the analysis of Meehl et al., Sci. Adv. (2020). In this paper, the equilibrium climate sensitivity (ECS) and transient climate response (TCR) are evaluated for the CMIP6 models and put into historical context.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_meehl20sciadv.yml

Diagnostics are stored in diag_scripts/

- climate_metrics/ecs.py
- climate_metrics/tcr.py
- climate_metrics/create_table.py
- ipcc_ar5/ch09_fig09_42b.py

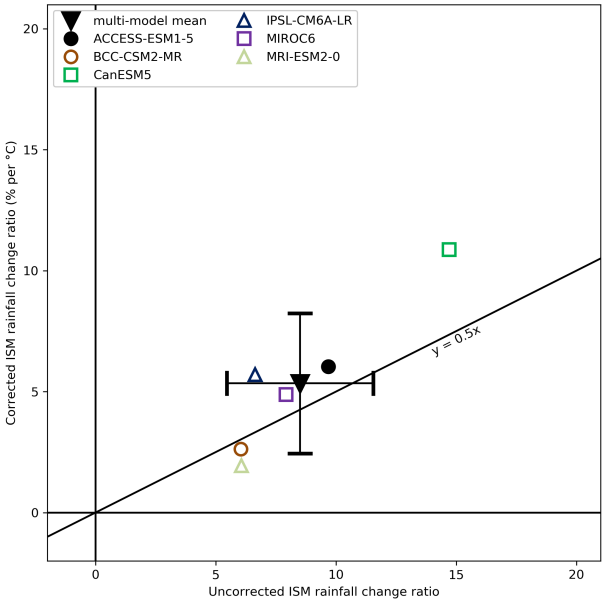


Fig. 148: Scatter plot of the uncorrected versus corrected average ISM (Indian Summer Monsoon) rainfall change ratios (% per degree Celsius of global SST warming). The error bars for the Multi-model mean indicate the standard deviation spread among models and the 2:1 line (y = 0.5x) is used to illustrate the Multi-model mean reduction in projected rainfall increase. (CMIP6).

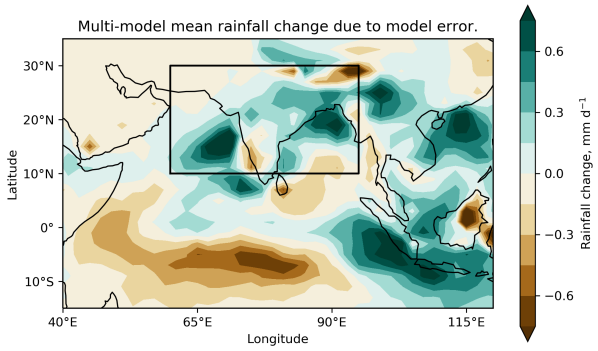


Fig. 149: Multi-model mean rainfall change due to model error. Box displays the area used to define the average ISM (Indian Summer Monsoon) rainfall. Precipitation changes are normalized by the corresponding global mean SST increase for each model. (CMIP6).

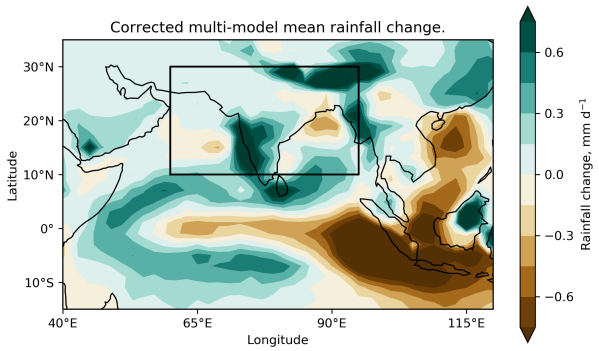


Fig. 150: Corrected multi-model mean rainfall change. Box displays the area used to define the average ISM (Indian Summer Monsoon) rainfall. Precipitation changes are normalized by the corresponding global mean SST increase for each model. (CMIP6).

User settings in recipe

- Script climate_metrics/ecs.py

  See *here*.

- Script climate_metrics/tcr.py

  See *here*.

- Script climate_metrics/create_table.py
  - `calculate_mean`, *bool*, optional (default: `True`): Calculate mean over all datasets and add it to table.
  - `calculate_std`, *bool*, optional (default: `True`): Calculate standard deviation over all datasets and add it to table.
  - `exclude_datasets`, *list of str*, optional (default: `['MultiModelMean']`): Exclude certain datasets when calculating statistics over all datasets and for assigning an index.
  - `patterns`, *list of str*, optional: Patterns to filter list of input data.
  - `round_output`, *int*, optional: If given, round output to given number of decimals.
- Script ipcc_ar5/ch09_fig09_42b.py

  See *here*.

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)

References

- Meehl, G. A., Senior, C. A., Eyring, V., Flato, G., Lamarque, J. F., Stouffer, R. J., Taylor, K. E. and Schlund, M., *Context for interpreting equilibrium climate sensitivity and transient climate response from the CMIP6 Earth system models*, Science Advances, 6(26), eaba1981, https://doi.org/10.1126/sciadv.aba1981, 2020.

Example plots

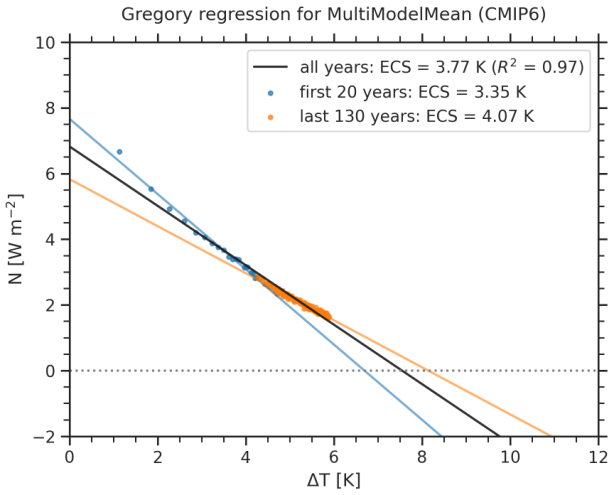


Fig. 151: ECS calculated for the CMIP6 models using the Gregory method over different time scales. Using the entire 150-year 4xCO2 experiment (black line), there is an ECS value of 3.8 K; using only the first 20 years (blue dots and blue line), there is an ECS of 3.4 K; and using the last 130 years, there is an ECS of 4.1 K (orange dots and orange line).

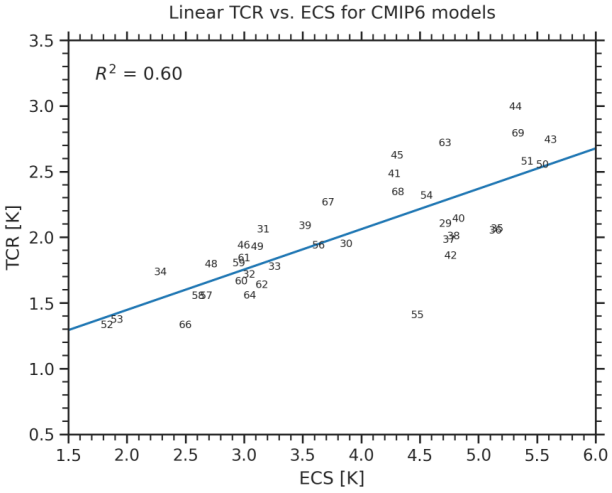


Fig. 152: TCR as a function of ECS for the CMIP6 models (black line is a linear fit). The R^2 values are given in the upper left parts of each panel. The numbers denote individual CMIP6 models.

5.4.4 Emergent constraints for equilibrium climate sensitivity

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)

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/single_constraint.py: create scatterplots for emergent constraints
- climate_metrics/psi.py: calculate temperature variabililty metric (Cox et al., 2018)

User settings in recipe

- 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)

  none

  Color tables

  none

- Script emergent_constraints/ecs_scatter.py

  See *here*.

- Script emergent_constraints/single_constraint.py

  See *here*.

- Script climate_metrics/psi.py

  See *here*.

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)

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

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.

Example plots

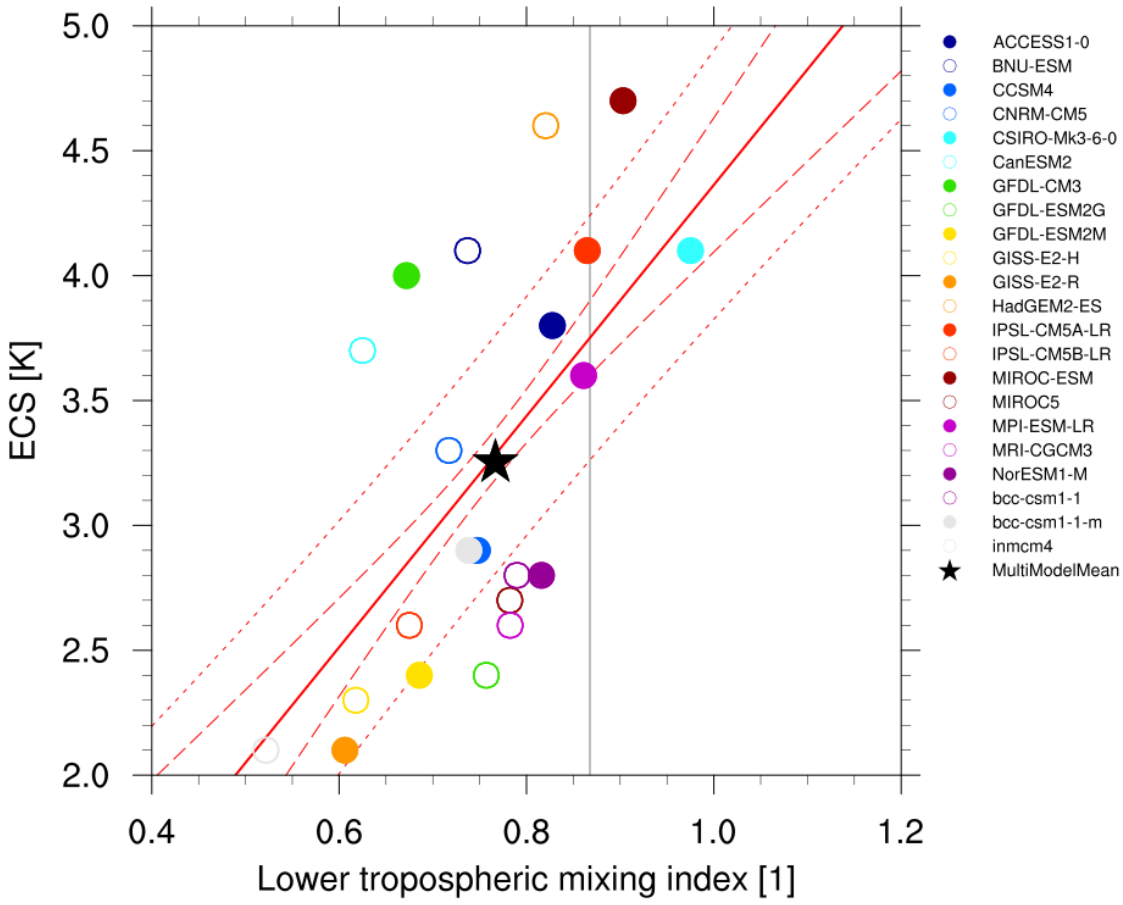


Fig. 153: Lower tropospheric mixing index (LTMI; Sherwood et al., 2014) vs. equilibrium climate sensitivity from CMIP5 models.

5.4.5 Emergent constraints on equilibrium climate sensitivity in CMIP5: do they hold for CMIP6?

Overview

This recipe reproduces the analysis of Schlund et al., Earth Sys. Dyn. (2020). In this paper, emergent constraints on the equilibrium climate sensitivity are evaluated on CMIP5 and CMIP6 models. Since none of the considered emergent constraints have been developed on the CMIP6 ensemble, this allows an out-of-sample testing of the emergent constraints. Most emergent constraints show a reduced skill in CMIP6 when compared to CMIP5.

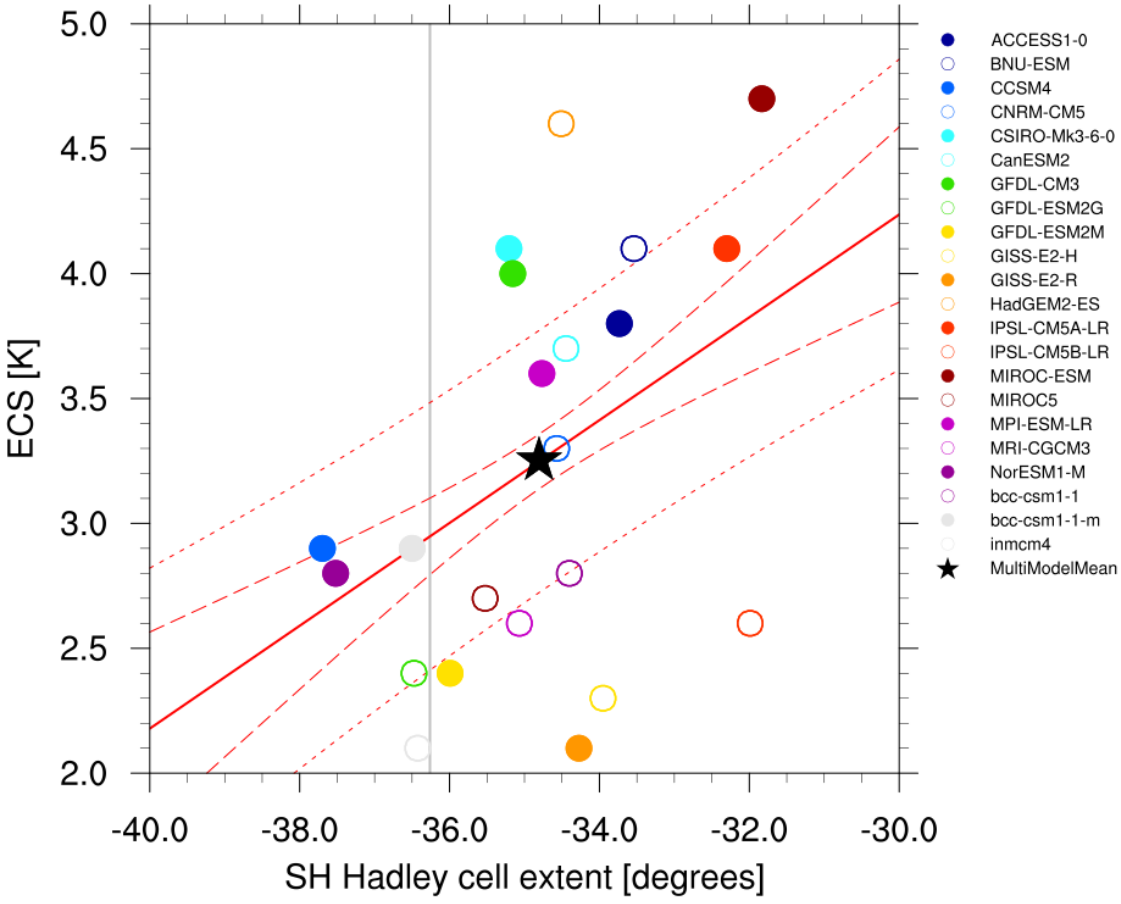


Fig. 154: Climatological Hadley cell extent (Lipat et al., 2017) vs. equilibrium climate sensitivity from CMIP5 models.

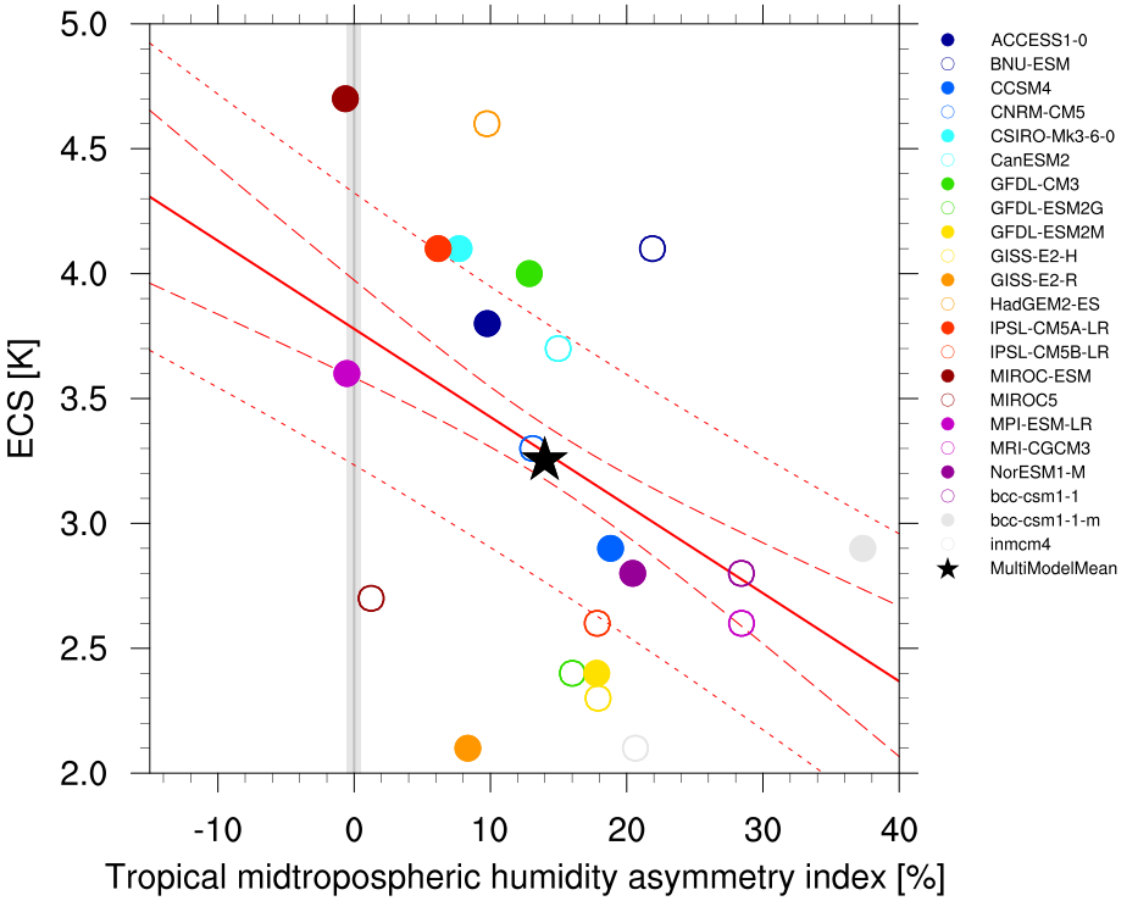


Fig. 155: Tropical mid-tropospheric humidity asymmetry index (Tian, 2015) vs. equilibrium climate sensitivity from CMIP5 models.

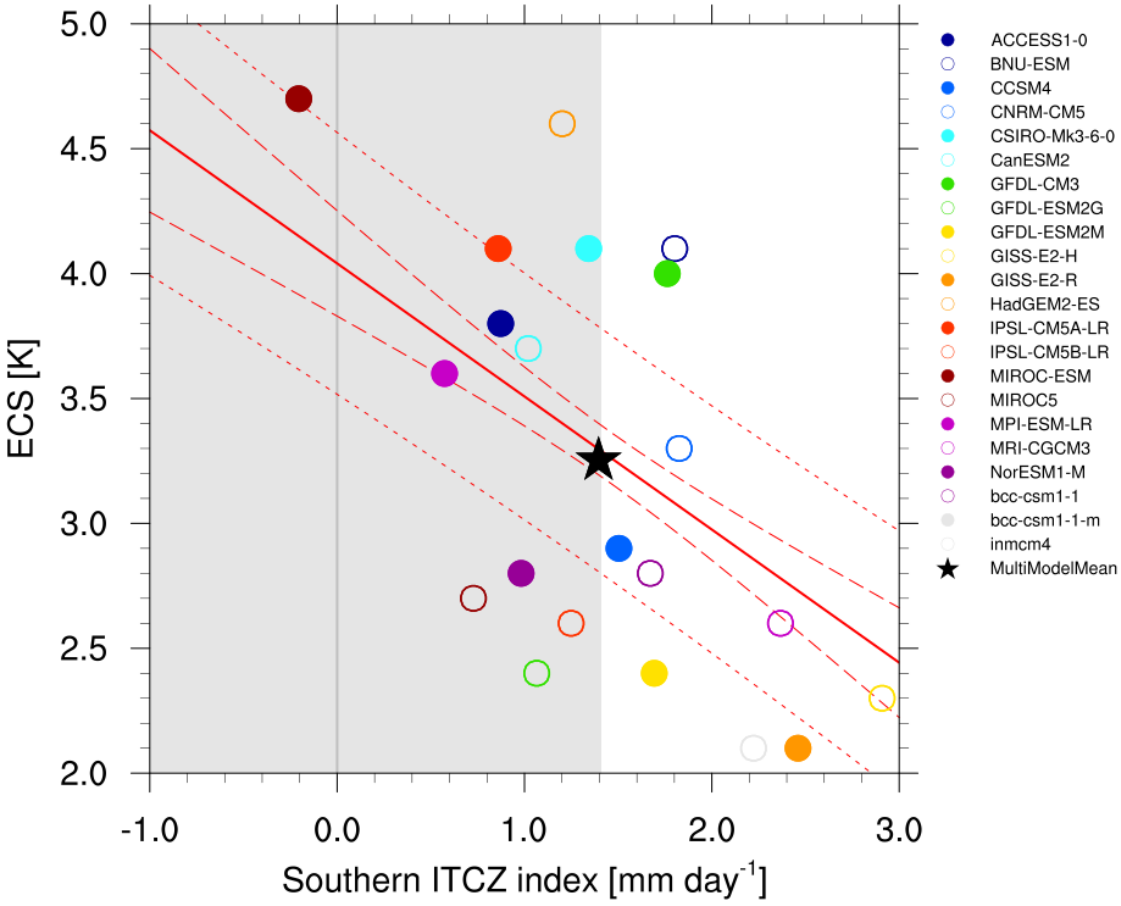


Fig. 156: Southern ITCZ index (Tian, 2015) vs. equilibrium climate sensitivity from CMIP5 models.

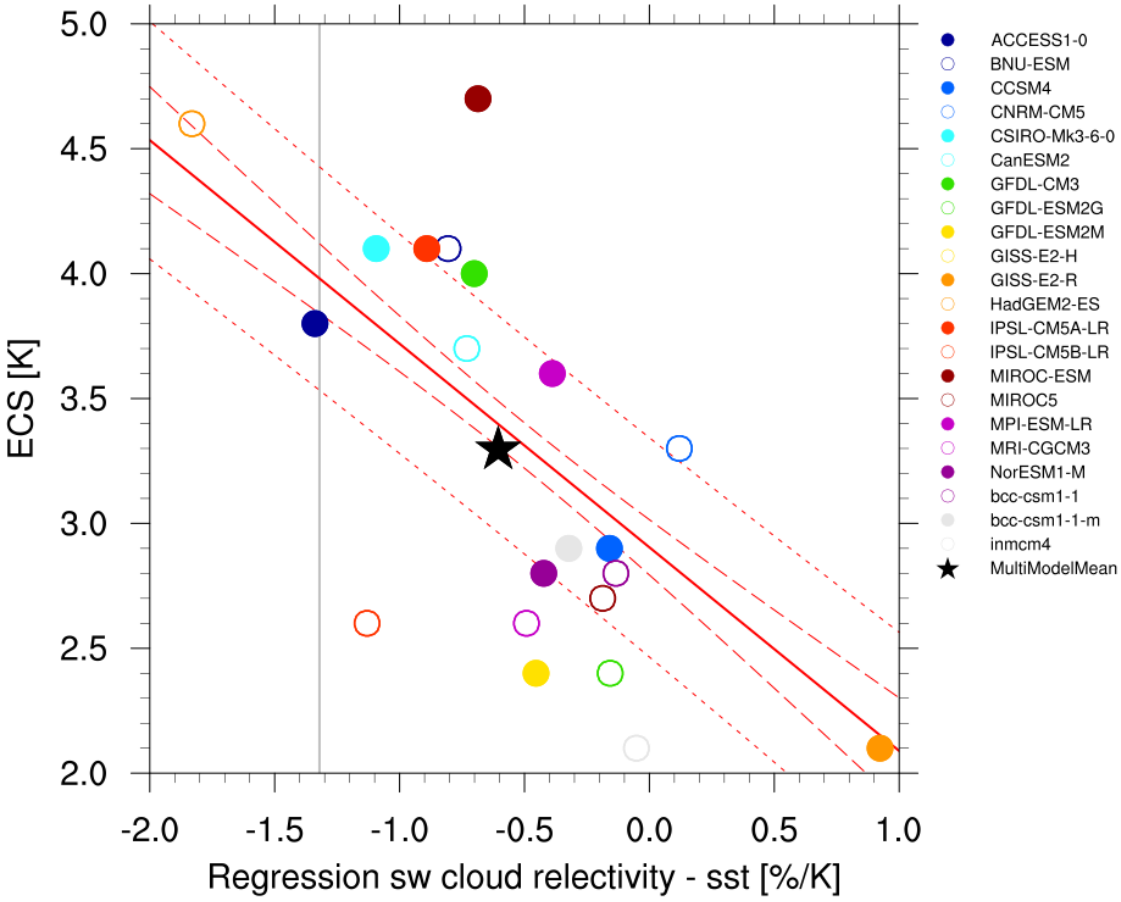


Fig. 157: Covariance of shortwave cloud reflection (Brient and Schneider, 2016) vs. equilibrium climate sensitivity from CMIP5 models.

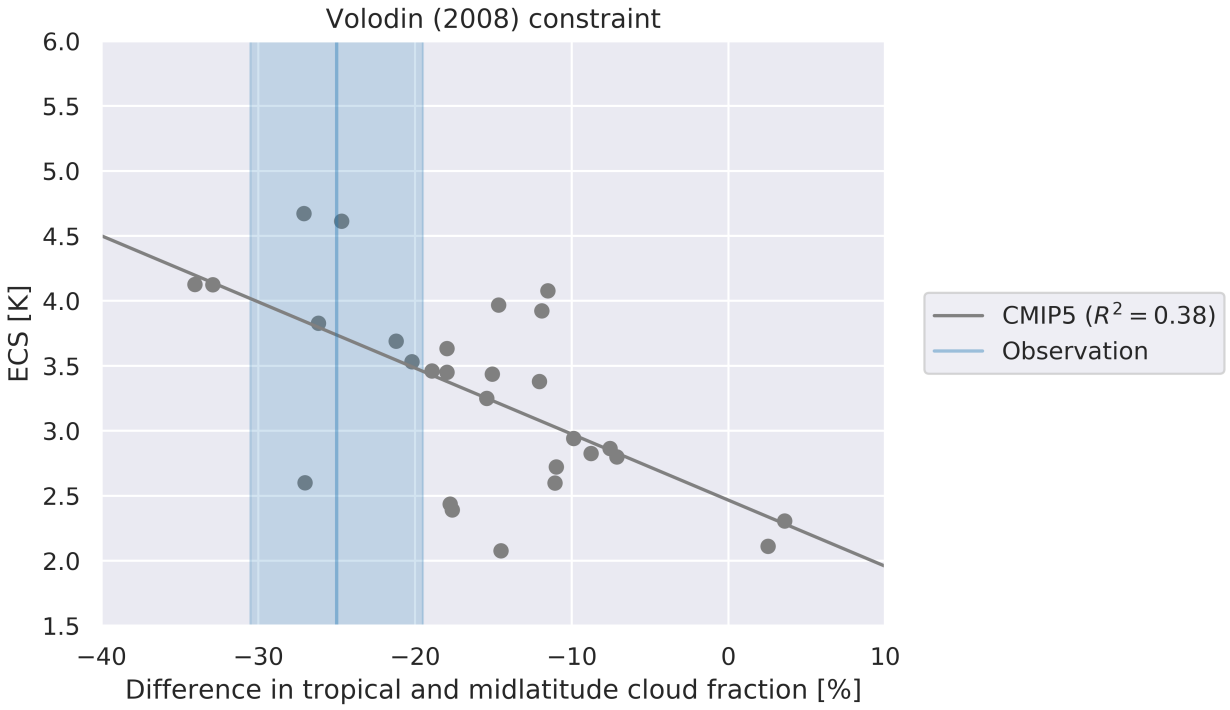


Fig. 158: Difference in total cloud fraction between tropics (28°S - 28°N) and Southern midlatitudes (56°S - 36°S) (Volodin, 2008) vs. equilibrium climate sensitivity from CMIP5 models.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_schlund20esd.yml

Diagnostics are stored in diag_scripts/

- *climate_metrics/ecs.py*
- *climate_metrics/psi.py*
- *emergent_constraints/ecs_scatter.ncl*
- *emergent_constraints/ecs_scatter.py*
- *emergent_constraints/multiple_constraints.py*

More details on the emergent constraint module are given in the API documentation which is available *here*.

Variables

- *cl* (atmos, monthly, longitude, latitude, level, time)
- *clt* (atmos, monthly, longitude, latitude, time)
- *hur* (atmos, monthly, longitude, latitude, level, time)
- *hus* (atmos, monthly, longitude, latitude, level, time)
- *pr* (atmos, monthly, longitude, latitude, time)
- *rsdt* (atmos, monthly, longitude, latitude, time)

- *rsut* (atmos, monthly, longitude, latitude, time)
- *rsutcs* (atmos, monthly, longitude, latitude, time)
- *rtnt* or *rtmt* (atmos, monthly, longitude, latitude, time)
- *ta* (atmos, monthly, longitude, latitude, level, time)
- *tas* (atmos, monthly, longitude, latitude, time)
- *tasa* (atmos, monthly, longitude, latitude, time)
- *tos* (atmos, monthly, longitude, latitude, time)
- *ts* (atmos, monthly, longitude, latitude, time)
- *va* (atmos, monthly, longitude, latitude, level, time)
- *wap* (atmos, monthly, longitude, latitude, level, time)

Observations and reformat scripts

- AIRS (*hur*, *hus*)
- CERES-EBAF (*rsut*, *rsutcs*, *rsdt*)
- ERA-Interim (*hur*, *ta*, *va*, *wap*)
- GPCP-SG (*pr*)
- HadCRUT4 (*tasa*)
- HadISST (*ts*)
- MLS-AURA (*hur*)

References

- Schlund, M., Lauer, A., Gentine, P., Sherwood, S. C., and Eyring, V.: Emergent constraints on equilibrium climate sensitivity in CMIP5: do they hold for CMIP6?, Earth Syst. Dynam., 11, 1233–1258, https://doi.org/10.5194/esd-11-1233-2020, 2020.

Example plots

5.4.6 Emergent constraint on equilibrium climate sensitivity from global temperature variability

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 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).

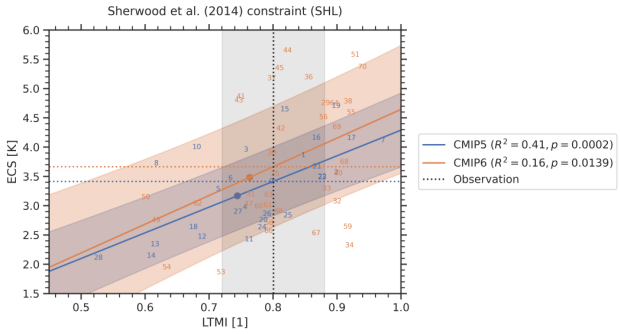


Fig. 159: Emergent relationship (solid blue and orange lines) of the Sherwood et al. (2014) emergent constraint, which is based on the lower tropospheric mixing index (LTMI). The numbers correspond to individual CMIP models. The shaded area around the regression line corresponds to the standard prediction error, which defines the error in the regression model itself. The vertical dashed black line corresponds to the observational reference with its uncertainty range given as standard error (gray shaded area). The horizontal dashed lines show the best estimates of the constrained ECS for CMIP5 (blue) and CMIP6 (orange). The colored dots mark the CMIP5 (blue) and CMIP6 (orange) multi-model means.

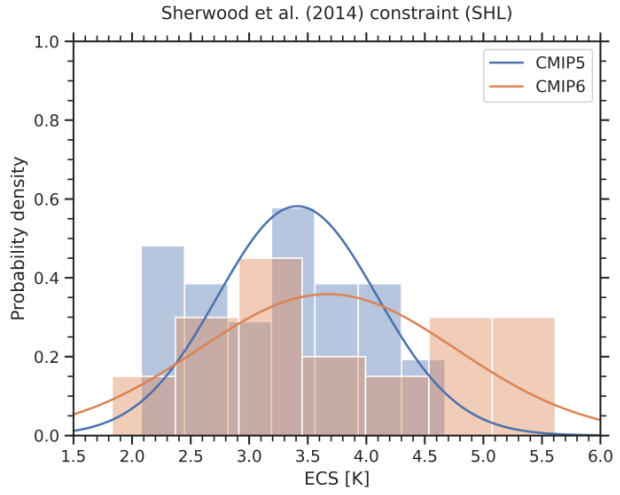


Fig. 160: Probability densities for the constrained ECS (solid lines) and the unconstrained model ensembles (histograms) of the emergent relationship shown in the figure above.

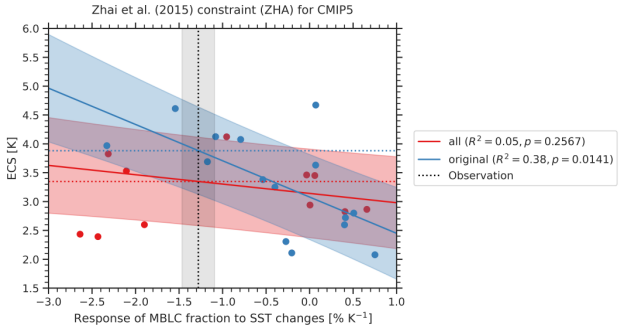


Fig. 161: Emergent relationship of the Zhai et al. (2015) emergent constraint for different subsets of CMIP5 models. Blue circles show the 15 CMIP5 models used in the original publication (except for CESM1-CAM5); the solid blue line and blue shaded area show the emergent relationships evaluated on these models including the uncertainty range. In this study, 11 more CMIP5 models have been added (red circles). The corresponding emergent relationship that considers all available CMIP5 models is shown in red colors. This relationship shows a considerably lower coefficient of determination (R^2) and higher p-value than the relationship using the original subset of CMIP5 models. The vertical dashed line and shaded area correspond to the observational reference, and the horizontal dashed lines show the corresponding ECS constraints using this observation.

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

User settings in recipe

- Preprocessor
  - `area_statistics` (*operation: mean*): Calculate global mean.
- Script emergent_constraints/cox18nature.py

  See *here*.
- Script climate_metrics/ecs.py

  See *here*.
- 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.

Variables

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

Observations and reformat scripts

- HadCRUT4 (*tasa*)

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.

Example plots

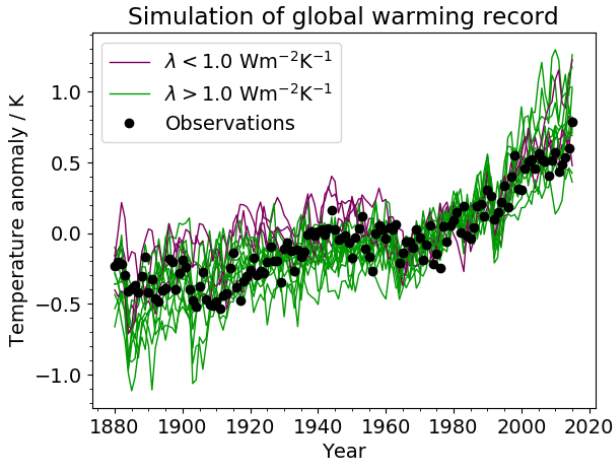


Fig. 162: Simulated change in global temperature from CMIP5 models (coloured lines), compared to the global temperature anomaly from the HadCRUT4 dataset (black dots). The anomalies are relative to a baseline period of 1961–1990. The model lines are colour-coded, with lower-sensitivity models (> 1 $Wm^{-2}K^{-1}$) shown by green lines and higher-sensitivity models (< 1 $Wm^{-2}K^{-1}$) shown by magenta lines.

5.4.7 Emergent constraint on snow-albedo effect

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.

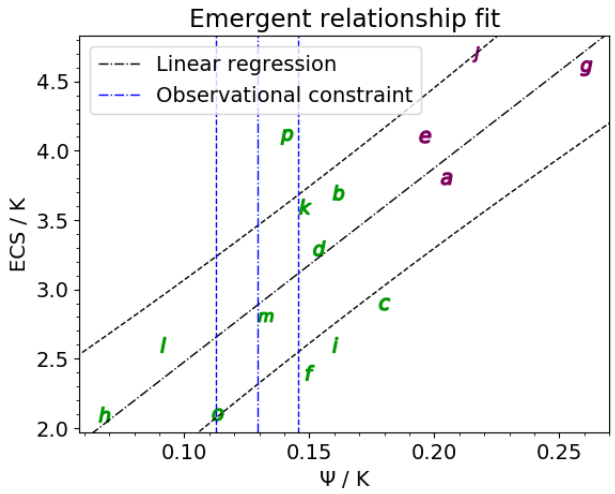


Fig. 163: Emergent relationship between ECS and the metric. The black dot-dashed line shows the best-fit linear regression across the model ensemble, with the prediction error for the fit given by the black dashed lines. The vertical blue lines show the observational constraint from the HadCRUT4 observations: the mean (dot-dashed line) and the mean plus and minus one standard deviation (dashed lines).

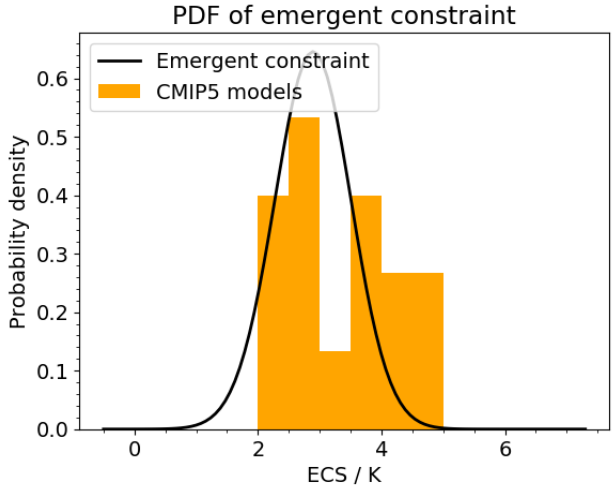


Fig. 164: The PDF for ECS. The orange histograms (both panels) show the prior distributions that arise from equal weighting of the CMIP5 models in 0.5 K bins.

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.

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

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

   none

   Required settings (scripts)

   none

   Optional settings (scripts)

Variables

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

Observations and reformat scripts

- ERA-Interim (tas - esmvaltool/cmorizers/data/formatters/datasets/era_interim.py)
- ISCCP-FH (rsuscs, rsdscs, rsdt - esmvaltool/cmorizers/data/formatters/datasets/isccp_fh.ncl)

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.

Example plots

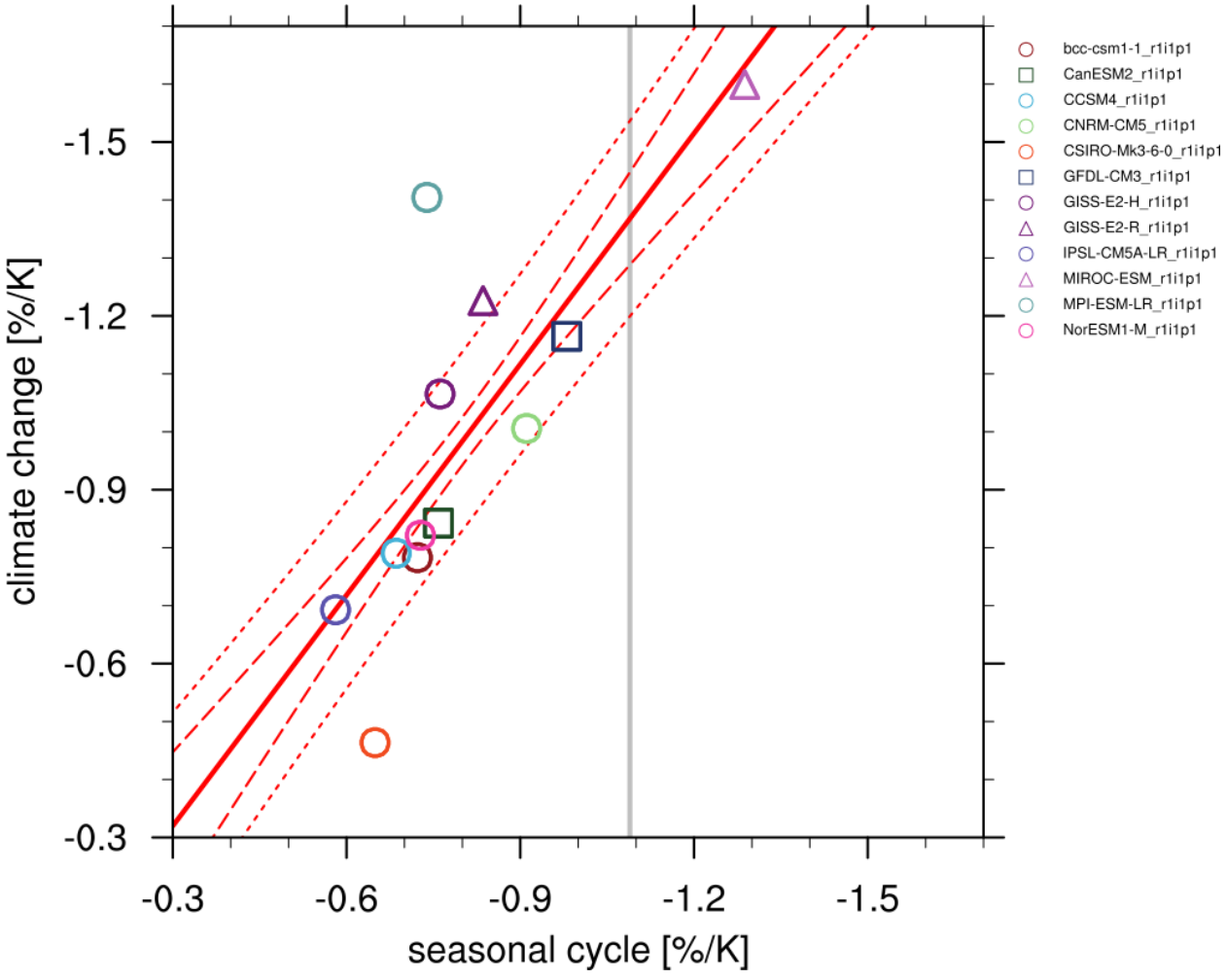


Fig. 165: Scatterplot of springtime snow-albedo effect values in climate change vs. springtime $\Delta\alpha_s/\Delta T_s$ values in the seasonal cycle in transient climate change experiments (CMIP5 historical experiments: 1901-2000, RCP4.5 experiments: 2101-2200). Similar to IPCC AR5 Chapter 9 (Flato et al., 2013), Figure 9.45a.

5.4.8 Equilibrium climate sensitivity

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.

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

User settings in recipe

- Preprocessor
  - `area_statistics` (*operation: mean*): Calculate global mean.
- Script climate_metrics/ecs.py
  - `calculate_mmm`, *bool*, optional (default: `True`): Calculate multi-model mean ECS.
  - `complex_gregory_plot`, *bool*, optional (default: `False`): Plot complex Gregory plot (also add response for first `sep_year` years and last 150 - `sep_year` years, default: `sep_year=20`) if `True`.
  - `output_attributes`, *dict*, optional: Write additional attributes to netcdf files.
  - `read_external_file`, *str*, optional: Read ECS and feedback parameters from external file. The path can be given relative to this diagnostic script or as absolute path.
  - `savefig_kwargs`, *dict*, optional: Keyword arguments for `matplotlib.pyplot.savefig()`.
  - `seaborn_settings`, *dict*, optional: Options for `seaborn.set_theme()` (affects all plots).
  - `sep_year`, *int*, optional (default: `20`): Year to separate regressions of complex Gregory plot. Only effective if `complex_gregory_plot` is `True`.
  - `x_lim`, *list of float*, optional (default: `[1.5, 6.0]`): Plot limits for X axis of Gregory regression plot (T).
  - `y_lim`, *list of float*, optional (default: `[0.5, 3.5]`): Plot limits for Y axis of Gregory regression plot (N).
- Script climate_metrics/create_barplot.py
  - `add_mean`, *str*, optional: Add a bar representing the mean for each class.
  - `label_attribute`, *str*, optional: Cube attribute which is used as label for different input files.
  - `order`, *list of str*, optional: Specify the order of the different classes in the barplot by giving the `label`, makes most sense when combined with `label_attribute`.
  - `patterns`, *list of str*, optional: Patterns to filter list of input data.
  - `savefig_kwargs`, *dict*, optional: Keyword arguments for `matplotlib.pyplot.savefig()`.
  - `seaborn_settings`, *dict*, optional: Options for `seaborn.set_theme()` (affects all plots).
  - `sort_ascending`, *bool*, optional (default: `False`): Sort bars in ascending order.

  - `sort_descending`, *bool*, optional (default: `False`): Sort bars in descending order.
  - `subplots_kwargs`, *dict*, optional: Keyword arguments for `matplotlib.pyplot.subplots()`.
  - `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.
- 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.set_theme()` (affects all plots).
  - `y_range`, *list of float*, optional: Range for the Y axis of the plot.

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)

Observations and reformat scripts

None

References

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

Example plots

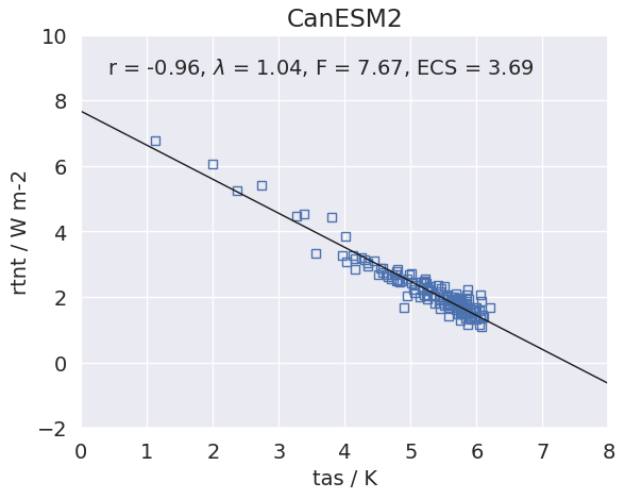


Fig. 166: Scatterplot between TOA radiance and global mean surface temperature anomaly for 150 years of the abrupt 4x CO2 experiment including linear regression to calculate ECS for CanESM2 (CMIP5).

5.4.9 KNMI Climate Scenarios 2014

Overview

This recipe implements the method described in Lenderink et al., 2014, to prepare the 2014 KNMI Climate Scenarios (KCS) for the Netherlands. A set of 8 global climate projections from EC-Earth were downscaled with the RACMO regional climate model. Since the EC-Earth ensemble is not readily representative for the spread in the full CMIP ensemble, this method recombines 5-year segments from the EC-Earth ensemble to obtain a large suite of "resamples". Subsequently, 8 new resamples are selected that cover the spread in CMIP much better than the original set.

The original method created 8 resampled datasets:

- 2 main scenarios: Moderate (M) and Warm (W) (Lenderink 2014 uses "G" instead of "M").
- 2 'sub'scenarios: Relatively high (H) or low (L) changes in seasonal temperature and precipitation
- 2 time horizons: Mid-century (MOC; 2050) and end-of-century (EOC; 2085)
- Each scenario consists of changes calculated between 2 periods: Control (e.g. 1981-2010) and future (variable).

The configuration settings for these resamples can be found in table 1 of Lenderink 2014's supplementary data.

Implementation

The implementation is such that application to other datasets, regions, etc. is relatively straightforward. The description below focuses on the reference use case of Lenderink et al., 2014, where the target model was EC-Earth. An external set of EC-Earth data (all RCP85) was used, for which 3D fields for downscaling were available as well. In the recipe shipped with ESMValTool, however, the target model is CCSM4, so that it works out of the box with ESGF data only.

In the first diagnostic, the spread of the full CMIP ensemble is used to obtain 4 values of a *global* ΔT_{CMIP}, corresponding to the 10th and 90th percentiles for the M and W scenarios, respectively, for both MOC and EOC. Subsequently, for each of these 4 *steering parameters*, 30-year periods are selected from the target model ensemble, where $\Delta T_{target} \approx \Delta T_{CMIP}$.

In the second diagnostic, for both the control and future periods, the N target model ensemble members are split into 6 segments of 5 years each. Out of all N^6 possible re-combinations of these 5-year segments, eventually M new 'resamples' are selected based on *local* changes in seasonal temperature and precipitation. This is done in the following steps:

1. Select 1000 samples for the control period, and 2 x 1000 samples for the future period (one for each subscenario). Step 1 poses a constraint on winter precipitation. For the control period, winter precipitation must still closely represent the average of the original ensemble. For the two future periods, the change in winter precipitation with respect to the control period must approximately equal 4% per degree ΔT (subscenario L) or 8% per degree ΔT (subscenario H).
2. Further constrain the selection by picking samples that represent either high or low changes in summer precipitation and summer and winter temperature, by limiting the remaining samples to certain percentile ranges: relatively wet/cold in the control and dry/warm in the future, or vice versa. The percentile ranges are listed in table 1 of Lenderink 2014's supplement. This should result is approximately 50 remaining samples for each scenario, for both control and future.
3. Use a Monte-Carlo method to make a final selection of 8 resamples with minimal reuse of the same ensemble member/segment.

Datasets have been split in two parts: the CMIP datasets and the target model datasets. An example use case for this recipe is to compare between CMIP5 and CMIP6, for example. The recipe can work with a target model that is not part of CMIP, provided that the data are CMOR compatible, and using the same data reference syntax as the CMIP data.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_kcs.yml

Diagnostics are stored in diag_scripts/kcs/

- global_matching.py
- local_resampling.py

Note

We highly recommend using the options described in *Re-running diagnostics*. The speed bottleneck for the first diagnostic is the preprocessor. In the second diagnostic, step 1 is most time consuming, whereas steps 2 and 3 are likely to be repeated several times. Therefore, intermediate files are saved after step 1, and the diagnostic will automatically detect and use them if the `-i` flag is used.

User settings

1. Script <global_matching.py>

   Required settings for script

   - `scenario_years`: a list of time horizons. Default: `[2050, 2085]`
   - `scenario_percentiles`: a list of percentiles for the steering table. Default: `[p10, p90]`

   Required settings for preprocessor This diagnostic needs global mean temperature anomalies for each dataset, both CMIP and the target model. Additionally, the multimodel statistics preprocessor must be used to produce the percentiles specified in the setting for the script above.

2. Script <local_resampling.py>

   Required settings for script

   - `control_period`: the control period shared between all scenarios. Default: `[1981, 2010]`
   - `n_samples`: the final number of recombinations to be selected. Default: `8`
   - `scenarios`: a scenario name and list of options. The default setting is a single scenario:

   ```yaml
   scenarios:
     ML_MOC:  # scenario name; can be chosen by the user
       description: "Moderate / low changes in seasonal temperature &␣
   ↪precipitation"
       global_dT: 1.0
       scenario_year: 2050
       resampling_period: [2021, 2050]
       dpr_winter: 4
       pr_summer_control: [25, 55]
       pr_summer_future: [45, 75]
       tas_winter_control: [50, 80]
       tas_winter_future: [20, 50]
       tas_summer_control: [0, 100]
       tas_summer_future: [0, 50]
   ```

   These values are taken from table 1 in the Lenderink 2014's supplementary material. Multiple scenarios can be processed at once by appending more configurations below the default one. For new

applications, `global_dT`, `resampling_period` and `dpr_winter` are informed by the output of the first diagnostic. The percentile bounds in the scenario settings (e.g. `tas_winter_control` and `tas_winter_future`) are to be tuned until a satisfactory scenario spread over the full CMIP ensemble is achieved.

Required settings for preprocessor

This diagnostic requires data on a single point. However, the `extract_point` preprocessor can be changed to `extract_shape` or `extract_region`, in conjunction with an area mean. And of course, the coordinates can be changed to analyze a different region.

Variables

Variables are precipitation and temperature, specified separately for the target model and the CMIP ensemble:

- pr_target (atmos, monthly mean, longitude latitude time)
- tas_target (atmos, monthly mean, longitude latitude time)
- pr_cmip (atmos, monthly mean, longitude latitude time)
- tas_cmip (atmos, monthly mean, longitude latitude time)

References

- Lenderink et al. 2014, Environ. Res. Lett., 9, 115008.

Example output

The diagnostic `global_matching` produces a scenarios table like the one below

```
   year percentile  cmip_dt period_bounds  target_dt  pattern_scaling_factor
0  2050        P10     0.98  [2019, 2048]       0.99                    1.00
1  2050        P90     2.01  [2045, 2074]       2.02                    0.99
2  2085        P10     1.38  [2030, 2059]       1.38                    1.00
3  2085        P90     3.89  [2071, 2100]       3.28                    1.18
```

which is printed to the log file and also saved as a csv-file `scenarios.csv`. Additionally, a figure is created showing the CMIP spread in global temperature change, AND highlighting the selected steering parameters and resampling periods:

The diagnostic `local_resampling` produces a number of output files:

- `season_means_<scenario>.nc`: intermediate results, containing the season means for each segment of the original target model ensemble.
- `top1000_<scenario>.csv`: intermediate results, containing the 1000 combinations that have been selected based on winter mean precipitation.
- `indices_<scenario>.csv`: showing the final set of resamples as a table:

```
                control                                                 future
               Segment 0 Segment 1 Segment 2 Segment 3 Segment 4 Segment 5 Segment␣
↪0 Segment 1 Segment 2 Segment 3 Segment 4 Segment 5
Combination 0           5         7         6         3         1         3        ␣
↪2         4         2         4         7         7
Combination 1           0         3         0         4         3         2        ␣
↪4         1         6         1         3         0
Combination 2           2         4         3         7         4         2        ␣
```

(continues on next page)

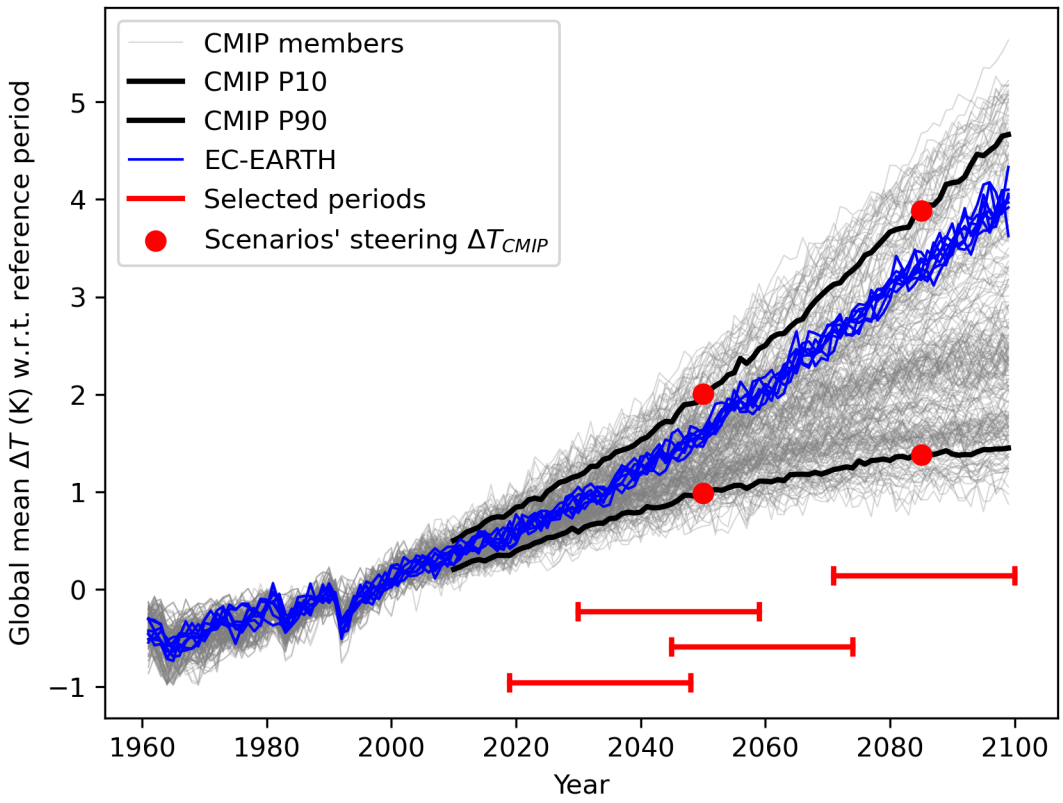


Fig. 167: CMIP spread in global temperature change, highlighting selected steering parameters and resampling periods

(continued from previous page)

```
↪5          4          6          6          4          2
Combination 3          1          4          7          2          3          6          ␣
↪5          3          1          7          4          1
Combination 4          5          7          6          3          1          3          ␣
↪2          3          0          6          1          7
Combination 5          7          2          1          4          5          1          ␣
↪6          0          4          2          3          3
Combination 6          7          2          2          0          6          6          ␣
↪5          2          1          5          4          2
Combination 7          6          3          2          1          6          1          ␣
↪2          1          0          2          1          3
```

- `resampled_control_<scenario>.nc`: containing the monthly means for the control period according to the final combinations.
- `resampled_future_<scenario>.nc`: containing the monthly means for the future period according to the final combinations.
- Provenance information: bibtex, xml, and/or text files containing citation information are stored alongside the final result and the final figure. The final combinations only derive from the target model data, whereas the figure also uses CMIP data.
- A figure used to validate the final result, reproducing figures 5 and 6 from Lenderink et al.:

5.4.10 Transient Climate Response

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).

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

User settings in recipe

- Preprocessor
  - `area_statistics` (*operation: mean*): Calculate global mean.
- Script climate_metrics/tcr.py
  - `calculate_mmm`, *bool*, optional (default: `True`): Calculate multi-model mean TCR.
  - `plot`, *bool*, optional (default: `True`): Plot temperature vs. time.

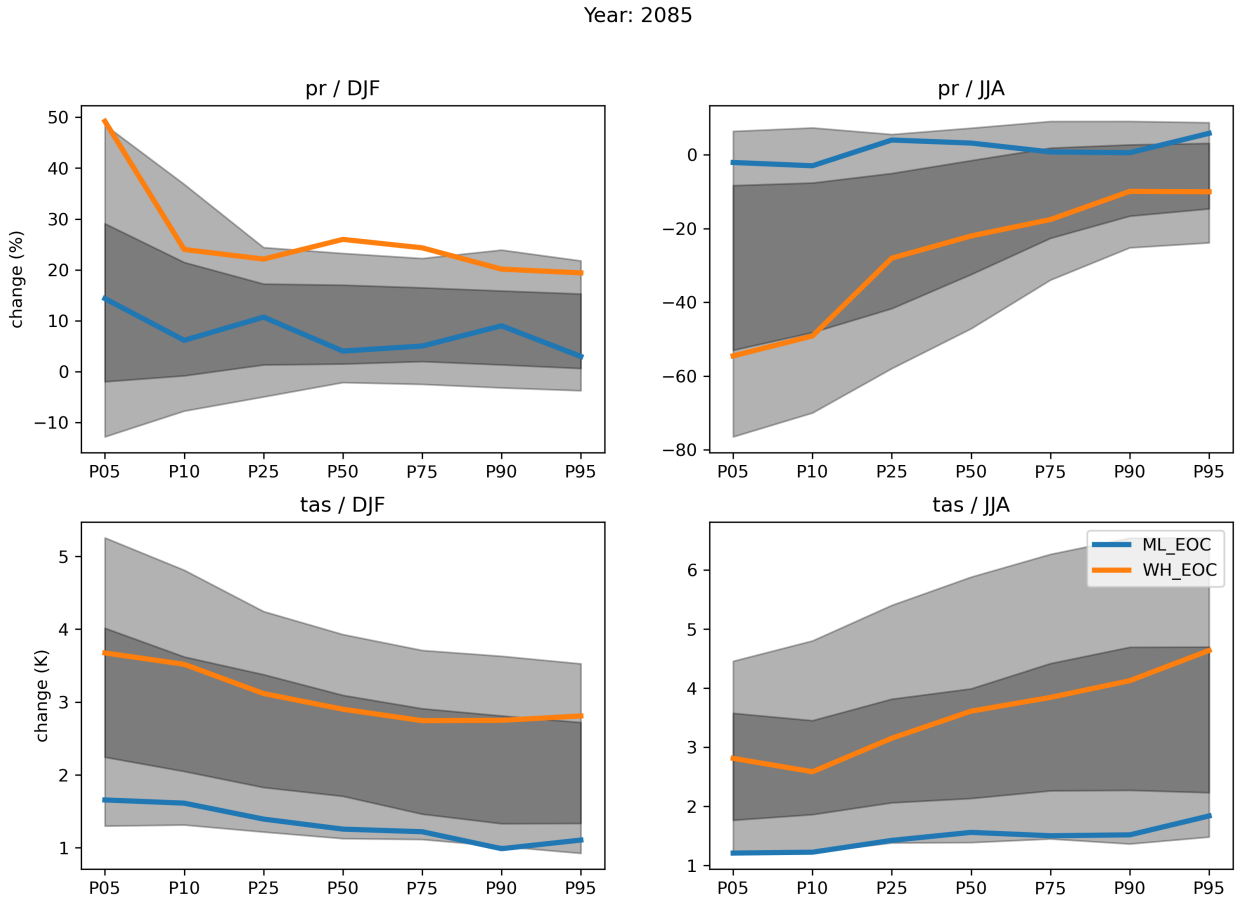


Fig. 168: Validation figure, reproducing figures 5 and 6 from Lenderink et al. (2014)

  - `read_external_file`, *str*, optional: Read TCR from external file. The path can be given relative to this diagnostic script or as absolute path.
  - `savefig_kwargs`, *dict*, optional: Keyword arguments for `matplotlib.pyplot.savefig()`.
  - `seaborn_settings`, *dict*, optional: Options for `seaborn.set_theme()` (affects all plots).
- Script climate_metrics/create_barplot.py

  See *here*.
- Script climate_metrics/create_scatterplot.py

  See *here*.

Variables

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

Observations and reformat scripts

None

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).

Example plots

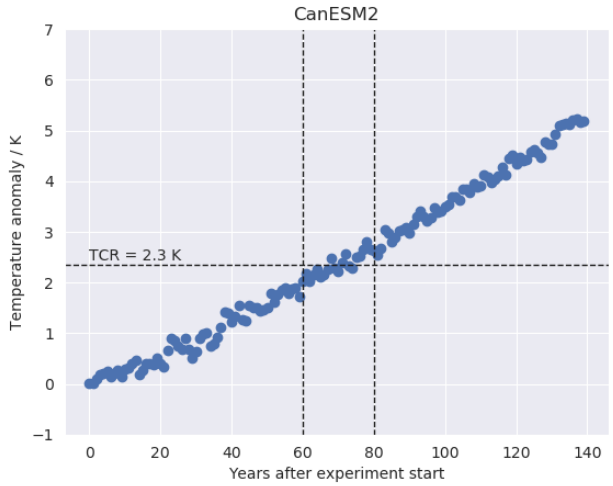


Fig. 169: Time series of the global mean surface air temperature anomaly (relative to the linear fit of the pre-industrial control run) of CanESM2 (CMIP5) for the 1% CO_2 increase per year experiment. The horizontal dashed line indicates the transient climate response (TCR) defined as the 20 year average temperature anomaly centered at the time of CO_2 doubling (vertical dashed lines).

5.4.11 Transient Climate Response to Cumulative CO_2 Emissions (TCRE)

Overview

The idea that global temperature rise is directly proportional to the total amount of carbon dioxide (CO_2) released into the atmosphere is fundamental to climate policy. The concept stems from research showing a clear linear relationship

between cumulative CO_2 emissions and global temperature change in climate models (Allen et al. 2009; Matthews et al. 2009; Zickfeld et al. 2009). This relationship is called the Transient Climate Response to Cumulative CO_2 Emissions (TCRE), which represents the amount of global warming caused by each trillion tonnes of carbon emitted. This simple yet powerful tool allows policymakers to directly link emission budgets to specific temperature targets and compare the long-term effects of different emissions scenarios.

For CMIP7, TCRE should be calculated from the `esm-flat10` experiment, following the experimental protocol of Sanderson et al (2024).

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_tcre.yml
- ref/recipe_tcre.yml

Diagnostics are stored in diag_scripts/

- *climate_metrics/tcre.py*
- *climate_metrics/barplot.py*

Variables

- *tas* (atmos, monthly, longitude, latitude, time)
- *fco2antt* (atmos, monthly, longitude, latitude, time)

Observations and reformat scripts

None

References

- Sanderson, B. M., Brovkin, V., Fisher, R., Hohn, D., Ilyina, T., Jones, C., Koenigk, T., Koven, C., Li, H., Lawrence, D., Lawrence, P., Liddicoat, S., Macdougall, A., Mengis, N., Nicholls, Z., O'Rourke, E., Romanou, A., Sandstad, M., Schwinger, J., Seferian, R., Sentman, L., Simpson, I., Smith, C., Steinert, N., Swann, A., Tjiputra, J., and Ziehn, T.: flat10MIP: An emissions-driven experiment to diagnose the climate response to positive, zero, and negative CO_2 emissions, EGUsphere [preprint], https://doi.org/10.5194/egusphere-2024-3356, 2024.
- Allen, M., Frame, D., Huntingford, C. et al.: Warming caused by cumulative carbon emissions towards the trillionth tonne, Nature 458, 1163–1166, https://doi.org/10.1038/nature08019, 2009.
- Matthews, H., Gillett, N., Stott, P. et al.: The proportionality of global warming to cumulative carbon emissions, Nature 459, 829–832, https://doi.org/10.1038/nature08047, 2009.
- K. Zickfeld, M. Eby, H. D. Matthews, & A. J. Weaver: Setting cumulative emissions targets to reduce the risk of dangerous climate change, Proc. Natl. Acad. Sci. U.S.A., 106 (38), 16129-16134, https://doi.org/10.1073/pnas.0805800106, 2009.

Example plots

5.4.12 Climate model projections from the ScenarioMIP of CMIP6

Overview

This recipe is implemented into ESMValTool to evaluate the temperature and precipitation changes from the ScenarioMIP of CMIP6. It produces the original plots and tables of Tebaldi et al. (2021), https://doi.org/10.5194/esd-12-253-2021

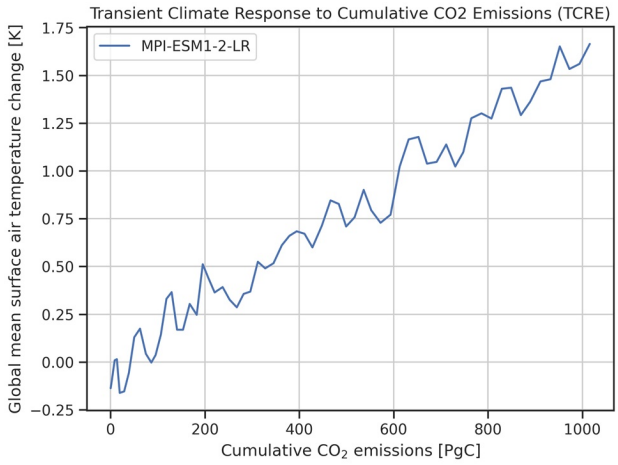


Fig. 170: Global mean surface air temperature anomaly versus cumulative CO_2 emissions for MPI-ESM1-2-LR using the emission-driven 1% CO_2 increase per year experiment.

Available recipe and diagnostics

Recipe is stored in esmvaltool/recipes/

- recipe_tebaldi21esd.yml

Diagnostics are stored in esmvaltool/diag_scripts/tebaldi21esd/

- calc_timeseries_across_realization_stddev_runave.ncl: computes time series of ensemble spreads (i.e., inter-member standard deviations). One dataset is used for resampling subsets of 10 members.
- calc_cmip6_and_cmip5_pattern_diff_scaleT.ncl: computes the pattern difference between the CMIP6 multi-model mean change and the CMIP5 multi-model mean change.
- calc_IAV_hatching.ncl: computes the interannual variability (IAV) over piControl runs, either over the whole time period or in chunks over some years.
- calc_pattern_diff_scaleT.ncl: computes the map of multi-model mean change scaled by global T change.
- calc_pattern_stippling_hatching.ncl: computes the map of multi-model mean change with stippling for significant region and hatching for non-significant region. 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.
- calc_pattern_intermodel_stddev_scaleT.ncl: computes the intermodel standard deviation of the change scaled by global T change standard deviation of the change scaled by global T change
- calc_pattern_interscenario_stddev_scaleT.ncl: computes the interscenario standard deviation of the change scaled by global T change
- calc_pattern_stddev_scaleT.ncl: computes the standard deviation of the change scaled by global T change
- calc_pattern_comparison.ncl: computes the difference between the patterns of multi-model mean change of two different scenarios (ex: SSP4-6.0 and SSP4-3.4)
- calc_table_changes.ncl: computes the changes (mean and spreads) for the specified scenarios and time periods relative to the historical baseline.
- calc_table_warming_level.ncl: computes the warming level crossing year (mean, five percent and ninety-five percent quantiles of crossing years) for specified scenarios and warming levels.

- calc_timeseries_mean_spread_runave.ncl: computes multi-model time series of change against historical baseline for specified scenarios with spread. A running average with specified window is performed.
- calc_timeseries_mean_spread_ssp4.ncl: computes multi-model time series of change against historical baseline for specified ssp434 and ssp460 with spread. A running average with specified window is performed.
- calc_timeseries_mean_spread_ssp5.ncl: computes multi-model time series of change against historical baseline for ssp534-over and ssp585 with spread. A running average with specified window is performed.
- plot_pattern.ncl: plots a pattern.
- plot_table_changes: plots a table of the multi-model mean and spread for specified scenarios and periods.
- plot_table_warming_level.ncl: plots a table of warming level crossing years for specified scenarios (columns) and warming levels (rows).
- plot_timeseries_mean_spread_3scenarios.ncl: plots time series (multi- model mean and spread) for 3 scenarios.
- plot_timeseries_mean_spread_constrained_projections.ncl: plot time series with brackets for constrained projections.
- plot_timeseries_mean_spread.ncl: plot time series (multi-model mean and spread) for 5 scenarios.
- plot_timeseries_mean_spread_rightaxis_5scen.ncl: plot time series (multi-model mean and spread) for 5 scenarios and with an additional right axis.
- plot_timeseries_mean_spread_ssp4.ncl: plot time series for two ssp4 scenarios.
- plot_timeseries_mean_spread_ssp5.ncl: plot time series for two ssp5 scenarios.
- plot_timeseries_across_realization_stddev_runave.ncl: plot time series of inter-member standard deviation.

User settings in recipe

1. Script calc_timeseries_across_realization_stddev_runave.ncl

   Required settings for script

   - scenarios: list with scenarios included in figure
   - syears: list with start years in time periods (e.g. start of historical period and SSPs)
   - eyears: list with end years in time periods (end year of historical runs and SSPs)
   - begin_ref_year: start year of reference period (e.g. 1995)
   - end_ref_year: end year of reference period (e.g. 2014)
   - n_samples: number of samples of size 10 to draw among all the ensembles of sampled_model
   - sampled_model: name of dataset on which to sample
   - runave_window: size window used for the centered running average

2. Script calc_cmip6_and_cmip5_pattern_diff_scaleT.ncl

   Required settings for script

   - scenarios_cmip5: list of CMIP5 scenarios included in figure
   - scenarios_cmip6: list of CMIP6 scenarios included in figure
   - periods: list with start years of periods to be included
   - time_avg: time_avg: time averaging ("annualclim", "seasonalclim")

   Optional settings for script

   - percent: determines if difference expressed in percent (0, 1, default= 0)

3. Script calc_IAV_hatching.ncl

   Required settings for script

   - time_avg: time_avg: time averaging ("annualclim", "seasonalclim") needs to be consistent with calc_pattern_stippling_hatching.ncl

   Optional settings for 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 calc_pattern_stippling_hatching.ncl

4. Script calc_pattern_diff_scaleT.ncl

   Required settings for script

   - scenarios: list with scenarios included in figure
   - periods: list with start years of periods to be included
   - time_avg: time_avg: time averaging ("annualclim", "seasonalclim")

5. Script calc_pattern_stippling_hatching.ncl

   Required settings for script

   - ancestors: variable and diagnostics that calculated interannual variability for stiplling and hatching
   - time_avg: time_avg: time averaging ("annualclim", "seasonalclim") needs to be consistent with calc_IAV_hatching.ncl
   - scenarios: list with scenarios to be included
   - periods: list with start years of periods to be included
   - labels: list with labels to use in legend depending on scenarios
   - sig: plot stippling for significance? (True, False)
   - not_sig: plot hatching for uncertainty? (True, False)

   Optional settings for script

   - seasons: list with season index if time_avg is "seasonalclim" (then seasons is 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 calc_IAV_hatching.ncl
   - percent: determines if difference expressed in percent (0, 1, default = 0)

6. Script calc_pattern_intermodel_stddev_scaleT.ncl

   Required settings for script

   - scenarios: list with scenarios included in figure
   - periods: list with start years of periods to be included
   - time_avg: time_avg: time averaging ("annualclim", "seasonalclim")

7. Script calc_pattern_interscenario_stddev_scaleT.ncl

   Required settings for script

   - scenarios: list with scenarios included in figure
   - periods: list with start years of periods to be included

- time_avg: time_avg: time averaging ("annualclim", "seasonalclim")

8. Script calc_pattern_stddev_scaleT.ncl

   Required settings for script

   - scenarios: list with scenarios included in figure
   - periods: list with start years of periods to be included
   - time_avg: time_avg: time averaging ("annualclim", "seasonalclim")

9. Script calc_pattern_comparison.ncl

   Required settings for script

   - scenarios: list with two scenarios included in figure. The last scenario is taken as reference. For example to compute the difference of pattern between SSP4-6.0 and SSP4-3.4, the scenario ssp460 should be the last element of the list.
   - periods: list with start years of periods to be included
   - time_avg: time_avg: time averaging ("annualclim", "seasonalclim")
   - label: label of periods

10. Script calc_table_changes.ncl

    Required settings for script

    - scenarios: list with scenarios included in the table
    - syears: list with start years of time periods to include in the table
    - eyears: list with end years of the time periods to include in the table
    - begin_ref_year: start year of historical baseline period (e.g. 1995)
    - end_ref_year: end year of historical baseline period (e.g. 2014)
    - spread: multiplier of standard deviation to calculate spread with (e.g. 1.64)
    - label: list of scenario names included in the table

11. Script calc_table_warming_level.ncl

    Required settings for script

    - scenarios: list with scenarios included in the table
    - warming_levels: list of warming levels to include in the table
    - syears: list with start years of time periods (historical then SSPs)
    - eyears: list with end years of the time periods (historical then SSPs)
    - begin_ref_year: start year of historical baseline period (e.g. 1995)
    - end_ref_year: end year of historical baseline period (e.g. 2014)
    - offset: offset between current historical baseline and 1850-1900 period
    - label: list of scenario names included in the table

12. Script calc_timeseries_mean_spread_runave.ncl

    Required settings for script

    - scenarios: list of scenarios to include
    - syears: list with start years of time periods (historical then SSPs)

- eyears: list with end years of the time periods (historical then SSPs)
- begin_ref_year: start year of historical baseline period (e.g. 1986)
- end_ref_year: end year of historical baseline period (e.g. 2005)

Optional settings for script

- runave_window: size of the window used to perform running average (default 11)
- spread: how many standard deviations to calculate the spread with (default 1)
- label: list of scenario names included in the legend
- percent: determines if difference expressed in percent (0, 1, default = 0)
- model_nr: whether to save number of models used for each scenario

13. Script calc_timeseries_mean_spread_ssp4.ncl

Required settings for script

- scenarios: list of scenarios to include: ssp434 and ssp460
- syears: list with start years of time periods (historical then SSPs)
- eyears: list with end years of the time periods (historical then SSPs)
- begin_ref_year: start year of historical baseline period (e.g. 1986)
- end_ref_year: end year of historical baseline period (e.g. 2005)

Optional settings for script

- runave_window: size of the window used to perform running average (default 11)
- spread: how many standard deviations to calculate the spread with (default 1)
- label: list of scenario names included in the legend
- percent: determines if difference expressed in percent (0, 1, default = 0)
- model_nr: whether to save number of models used for each scenario

14. Script calc_timeseries_mean_spread_ssp5.ncl

Required settings for script

- scenarios: list of scenarios to include: ssp534-over, ssp585
- syears: list with start years of time periods (historical then SSPs)
- eyears: list with end years of the time periods (historical then SSPs)
- begin_ref_year: start year of historical baseline period (e.g. 1986)
- end_ref_year: end year of historical baseline period (e.g. 2005)

Optional settings for script

- runave_window: size of the window used to perform running average (default 11)
- spread: how many standard deviations to calculate the spread with (default 1)
- label: list of scenario names included in the legend
- percent: determines if difference expressed in percent (0, 1, default = 0)
- model_nr: whether to save number of models used for each scenario

15. Script plot_pattern.ncl

    Required settings for script

    - scenarios: list of scenarios
    - periods: list with start years of periods
    - ancestors: variable and diagnostics that calculated field to be plotted

    Optional settings for 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)
    - pltname: alternative name for output plot, default is diagnostic + varname + time_avg
    - units: units written next to colorbar, e.g. (~F35~J~F~C)
    - sig: plot stippling for significance? (True, False)
    - not_sig: plot hatching for uncertainty? (True, False)
    - label: label to add in the legend

16. Script plot_table_changes.ncl

    Required settings for script

    - ancestors: variable and diagnostics that calculated field to be plotted
    - scenarios: list of scenarios included in the figure
    - syears: list of start years of periods of interest
    - eyears: list of end years of periods of interest
    - label: list of labels of the scenarios

    Optional settings for script

    - title: title of the plot

17. Script plot_table_warming_level.ncl

    Required settings for script

    - scenarios: list of scenarios included in the figure
    - warming_levels: list of warming levels
    - syears: list of start years of historical and SSPs scenarios
    - eyears: list of end years of historical and SSPs scenarios
    - begin_ref_year: start year of reference period
    - end_ref_year: end year of reference period

- label: list of labels of the scenarios
- offset: offset between reference baseline and 1850-1900

18. Script plot_timeseries_mean_spread_3scenarios.ncl

    Required settings for script

    - ancestors: variable and diagnostics that calculated field to be plotted
    - scenarios: list of scenarios included in the figure
    - syears: list of start years of historical and SSPs scenarios
    - eyears: list of end years of historical and SSPs scenarios
    - begin_ref_year: start year of reference period
    - end_ref_year: end year of reference period
    - label: list of labels of the scenarios

    Optional settings for 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
    - model_nr: save number of model runs per period and scenario
    - styleset: color style
    - spread: how many standard deviations to calculate the spread with, default is 1, ipcc tas is 1.64

19. Script plot_timeseries_mean_spread_constrained_projections.ncl

    Required settings for script

    - ancestors: variable and diagnostics that calculated field to be plotted
    - scenarios: list of scenarios included in the figure
    - syears: list of start years of historical and SSPs scenarios
    - eyears: list of end years of historical and SSPs scenarios
    - begin_ref_year: start year of reference period
    - end_ref_year: end year of reference period
    - label: list of labels of the scenarios
    - baseline_offset: offset between reference period (baseline) and 1850-1900
    - lower_constrained_projections: list of lower bounds of the constrained projections for the scenarios included in the same order as the scenarios
    - upper_constrained_projections: list of upper bounds of the constrained projections for the scenarios included in the same order as the scenarios
    - mean_constrained_projections: list of means of the constrained projections for the scenarios included in the same order as the scenarios

    Optional settings for 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
- model_nr: save number of model runs per period and scenario
- styleset: color style
- spread: how many standard deviations to calculate the spread with, default is 1, ipcc tas is 1.64

20. Script plot_timeseries_mean_spread.ncl

    Required settings for script

    - ancestors: variable and diagnostics that calculated field to be plotted
    - scenarios: list of scenarios included in the figure
    - syears: list of start years of historical and SSPs scenarios
    - eyears: list of end years of historical and SSPs scenarios
    - begin_ref_year: start year of reference period
    - end_ref_year: end year of reference period
    - label: list of labels of the scenarios

    Optional settings for 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
    - model_nr: save number of model runs per period and scenario
    - styleset: color style
    - spread: how many standard deviations to calculate the spread with, default is 1, ipcc tas is 1.64

21. Script plot_timeseries_mean_spread_rightaxis_5scen.ncl

    Required settings for script

    - ancestors: variable and diagnostics that calculated field to be plotted
    - scenarios: list of scenarios included in the figure
    - syears: list of start years of historical and SSPs scenarios
    - eyears: list of end years of historical and SSPs scenarios
    - begin_ref_year: start year of reference period
    - end_ref_year: end year of reference period
    - rightaxis_offset: offset of the right axis relative to the left axis
    - label: list of labels of the scenarios

Optional settings for 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
- model_nr: save number of model runs per period and scenario
- styleset: color style
- spread: how many standard deviations to calculate the spread with, default is 1, ipcc tas is 1.64

22. Script plot_timeseries_mean_spread_ssp4.ncl

Required settings for script

- ancestors: variable and diagnostics that calculated field to be plotted
- scenarios: list of scenarios included in the figure
- syears: list of start years of historical and SSPs scenarios
- eyears: list of end years of historical and SSPs scenarios
- begin_ref_year: start year of reference period
- end_ref_year: end year of reference period
- label: list of labels of the scenarios

Optional settings for 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
- model_nr: save number of model runs per period and scenario
- styleset: color style
- spread: how many standard deviations to calculate the spread with, default is 1, ipcc tas is 1.64

23. Script plot_timeseries_mean_spread_ssp5.ncl

Required settings for script

- ancestors: variable and diagnostics that calculated field to be plotted
- scenarios: list of scenarios included in the figure
- syears: list of start years of historical and SSPs scenarios
- eyears: list of end years of historical and SSPs scenarios
- begin_ref_year: start year of reference period
- end_ref_year: end year of reference period
- label: list of labels of the scenarios

Optional settings for 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
- model_nr: save number of model runs per period and scenario
- styleset: color style
- spread: how many standard deviations to calculate the spread with, default is 1, ipcc tas is 1.64

24. Script plot_timeseries_across_realization_stddev_runave.ncl

Required settings for script

- ancestors: variable and diagnostics that calculated field to be plotted
- scenarios: list of scenarios included in the figure
- syears: list of start years of historical and SSPs scenarios
- eyears: list of end years of historical and SSPs scenarios
- begin_ref_year: start year of reference period
- end_ref_year: end year of reference period
- label: list of labels of the scenarios
- n_samples: number of samples of size 10 to draw among all the ensembles of sampled_model only
- sampled_model: name of dataset on which to sample

Optional settings for script

- trend: whether the trend is calculated and displayed
- runave_window: only used if trend is true, size window used for the centered running average
- 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

Variables

Note: These are the variables tested and used in the original paper.

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

However, the code is flexible and in theory other variables of the same kind can be used.

References

- Tebaldi, C., Debeire, K., Eyring, V., Fischer, E., Fyfe, J., Friedlingstein, P., Knutti, R., Lowe, J., O'Neill, B., Sanderson, B., van Vuuren, D., Riahi, K., Meinshausen, M., Nicholls, Z., Hurtt, G., Kriegler, E., Lamarque, J.-F., Meehl, G., Moss, R., Bauer, S. E., Boucher, O., Brovkin, V., Golaz, J.-C., Gualdi, S., Guo, H., John, J. G., Kharin, S., Koshiro, T., Ma, L., Olivié, D., Panickal, S., Qiao, F., Rosenbloom, N., Schupfner, M., Seferian, R., Song, Z., Steger, C., Sellar, A., Swart, N., Tachiiri, K., Tatebe, H., Voldoire, A., Volodin, E., Wyser, K., Xin, X., Xinyao, R., Yang, S., Yu, Y., and Ziehn, T.: Climate model projections from the Scenario Model Intercomparison Project (ScenarioMIP) of CMIP6, Earth Syst. Dynam., 12, 253-293, https://doi.org/10.5194/esd-12-253-2021

Example plots

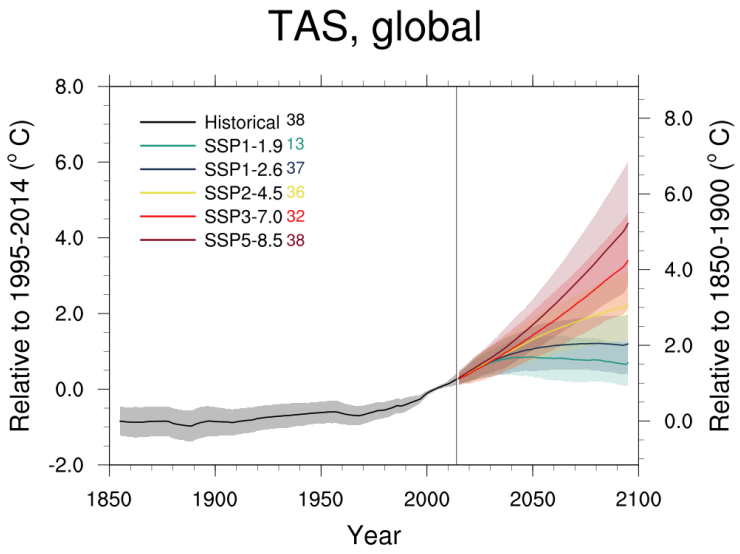


Fig. 171: Global average temperature time series (11-year running averages) of changes from current baseline (1995–2014, left axis) and pre-industrial baseline (1850–1900, right axis, obtained by adding a 0.84 ∘C offset) for SSP1-1.9, SSP1-2.6, SSP2-4.5, SSP3-7.0 and SSP5-8.5.

5.4.13 Climate Change Hotspot

Overview

In the context of a changing climate, it is found that not all regions change at the same pace and the same way. The regions that change at a faster rate than the rest of the globe are labelled as climate change hotspots. Estimating the location and magnitude of the hotspots is important for climate change adaptation, and it is usually computed using the projected climate variables' differences between the regional and larger scales.

One issue when trying to evaluate projections of climate change is the vast amount of information available from the Coupled Model Intercomparison Project (CMIP) exercises. Additionally, results from the CMIP phases 5 and 6 can be quite different, therefore a comparison between the two multi-model ensembles can be made to evaluate their differences and similarities. To account for the projections scenario uncertainty, data from three different end-of-the-century radiative forcings is given in the recipe.

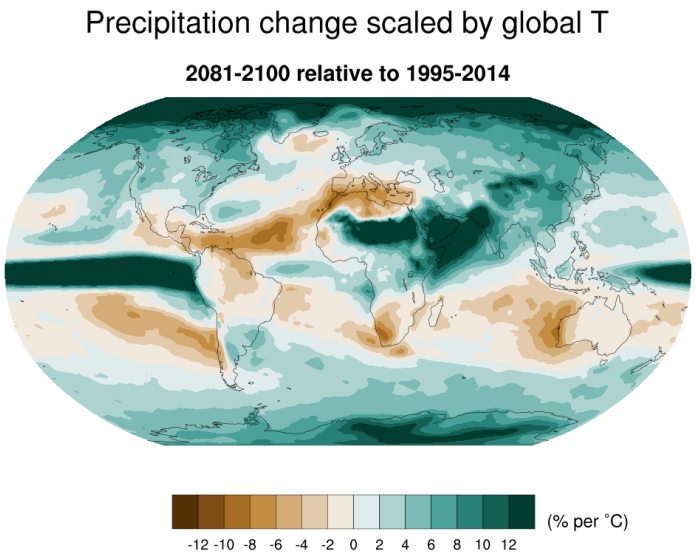


Fig. 172: Patterns of temperature (a) and percent precipitation change (b) normalized by global average temperature change (averaged across CMIP6 models and all Tier 1 plus SSP1-1.9 scenarios).

This recipe compares regional surface temperature and precipitation against larger scale means to obtain the hotspot magnitudes for both CMIP5 and CMIP6 in the 2.6, 4.5 and 8.5 Wm^-2 radiative forcings by the year 2100 against the preindustrial Era (RCP2.6, RCP4.5, RCP8.5 for CMIP5 and SSP1-2.6, SSP2-4.5, SSP5-8.5 for CMIP6). Recipe based on the work by Cos et al. (2022).

Note: This recipe is currently set to evaluate the Mediterranean hotspot (with bounds start_longitude: -10, end_longitude: 40, start_latitude: 30, end_latitude: 45) but it can be set to any other rectangular region.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- **`recipe_climate_change_hotspot.yml`: Loads and ensembles the data,**
  computes the necessary climate change hotspot diagnostics and plots the results figures.

Diagnostics are stored in esmvaltool/diag_scripts/cos22esd/

- `climate_change_hotspot.py`: Calculates the regional field hotspot for temperature and precipitation and the 10-year rolling mean timeseries for regional and large-scale temperature and precipitation.
- `hotspot_plotter.py`: Gathers the data output from the `climate_change_hotspot.py` script and plots the hotspot fields and the rolling mean timeseries [Figures 2, 3, S2 and S4 by Cos et al. (2022).].

User settings in the recipe

1. Script `climate_change_hotspot.py`

   Required settings for script

   - `baseline_period`: Historical period that serves as a reference to compute the time anomalies.

Warming level crossings relative to 1850-1900

	SSP1-1.9	SSP1-2.6	SSP2-4.5	SSP3-7.0	SSP5-8.5
1.5°C warming level	2029 [2021,NA] (11/13)	2028 [2020,NA] (30/31)	2028 [2020,2047] (31/31)	2028 [2020,2045] (31/31)	2026 [2020,2040] (31/31)
2.0°C warming level	NA [2036,NA] (2/13)	2064 [2032,NA] (17/31)	2046 [2032,2082] (31/31)	2043 [2031,2064] (31/31)	2039 [2030,2055] (31/31)
3.0°C warming level	NA [NA,NA] (0/13)	NA [NA,NA] (0/31)	2094 [2058,NA] (16/31)	2069 [2052,NA] (31/31)	2060 [2048,2083] (31/31)
4.0°C warming level	NA [NA,NA] (0/13)	NA [NA,NA] (0/31)	NA [NA,NA] (1/31)	2091 [2071,NA] (17/31)	2078 [2062,NA] (27/31)
5.0°C warming level	NA [NA,NA] (0/13)	NA [NA,NA] (0/31)	NA [NA,NA] (0/31)	NA [2088,NA] (3/31)	2094 [2075,NA] (15/31)

Fig. 173: Times (best estimate and range – in square brackets – based on the 5%–95% range of the ensemble after smoothing the trajectories by 11-year running means) at which various warming levels (defined as relative to 1850–1900) are reached according to simulations following, from left to right, SSP1-1.9, SSP1-2.6, SSP2-4.5, SSP3-7.0 and SSP5-8.5. Crossing of these levels is defined by using anomalies with respect to 1995–2014 for the model ensembles and adding the offset of 0.84 to derive warming from pre-industrial values. We use a common subset of 31 models for the Tier 1 scenarios and all available models (13) for SSP1-1.9, while Table A7 shows the result of using all available models under each scenario. The number of models available under each scenario and the number of models reaching a given warming level are shown in parentheses. However, the estimates are based on the ensemble means and ranges computed from all the models considered (13 or 31 in this case), not just from the models that reach a given level. An estimate marked as "NA" is to be interpreted as "not reaching that warming level by 2100". In cases where the ensemble average remains below the warming level for the whole century, it is possible for the central estimate to be NA, while the earlier time of the confidence interval is not, since it is determined by the warmer end of the ensemble range.

- `future_periods`: List of the two future periods given in years ("YYYY-YYYY") where the hotspot will be computed. Following the format [future period #1, future period #2].
- `region`: list of longitudes and latitudes that enclose a rectangular region. In the form of [start_longitude, end_longitude, start_latitude, end_latitude].
- `region_name`: Name of the region to be included in the provenance record.

2. Script `hotspot_plotter.py`

   Required settings for script

   - `baseline_period`: Historical period displayed in the figures' titles.
   - `future_periods`: List of the two future periods given in years ("YYYY-YYYY"), following the format [future period #1, future period #2], used to identify the ancestor files and in the figure titles.
   - `region`: List of longitudes and latitudes that enclose a region. In the form of [start_longitude, end_longitude, start_latitude, end_latitude]. Used in the title to identify the precipitation large-scale region.
   - `region_name`: Name of the region used in the plot titles.

Modifying the datasets and scenarios used

`recipe_climate_change_hotspot.yml` can be modified to use different scenario combinations. The standard recipe uses data from scenarios with the radiative forcings 2.6, 4.5 and 8.5 *Wm^{-2}* (referred to as 26, 45 and 85), but any combination of three scenarios from the following list can be used:

```
26: "RCP2.6/SSP1-2.6"
45: "RCP4.5/SSP2-4.5"
60: "RCP6.0/SSP4-6.0"
85: "RCP8.5/SSP5-8.5"
```

To specify which datasets are available for each scenario, lists of datasets can be attributed to a specific CMIP project and scenario between the `documentation` and `preprocessor` sections of the recipe as follows:

```
cmip6_85: &cmip6_85
   - {...dataset keys...}
   - {...dataset keys...}
cmip5_85: &cmip5_85
   - {...dataset keys...}
   - {...dataset keys...}
cmip6_45: &cmip6_45
   - {...dataset keys...}
   - {...dataset keys...}
cmip5_45: &cmip5_45
   - {...dataset keys...}
   - {...dataset keys...}
```

These different dataset sections will be called at each diagnostic as `additional_datasets` using the anchors `*cmip6_85`, `*cmip5_85`, etc. as in the example:

```
pr_cmip6_85:
 variables:
   pr:
     mip: Amon
     short_name: pr
```

(continues on next page)

(continued from previous page)

```
    preprocessor: ensemble_members
    additional_datasets: *cmip6_85
 scripts:
   pr_cmip6_85:
     <<: *script_input
```

In case of wanting to use other scenarios, the datasets and diagnostics must be changed maintaining the format `cmip{phase}_{scenario}` and `{variable}_cmip{phase}_{scenario}`. For example, if we want scenario 60 instead of scenario 85, we would need to include the files available for `cmip6_60` and `cmip5_60`, and the previous diagnostic would change to:

```
pr_cmip6_60:
 variables:
   pr:
     mip: Amon
     short_name: pr
     preprocessor: ensemble_members
     additional_datasets: *cmip6_60
 scripts:
   pr_cmip6_60:
     <<: *script_input
```

Finally, if the datasets that need to be included in the multi-model means are common for all scenarios, the datasets could be simplified to:

```
cmip6: &cmip6
   - {...dataset keys...}
   - {...dataset keys...}
cmip5: &cmip5
   - {...dataset keys...}
   - {...dataset keys...}
```

Note that the diagnostics' `additional_datasets` will need to be modified accordingly.

Variables

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

References

- Cos et al. 2022, Earth Syst. Dynam., 13, 321–340

Example plots

5.4.14 Cloud properties and their projected changes in CMIP models with low to high climate sensitivity

Overview

The recipes recipe_bock24acp_*.yml reproduce figures (Fig. 3, 4, 6 and 7) from the publication Bock and Lauer, 2024 investigating cloud properties and their projected changes in CMIP models with low to high climate sensitivity.

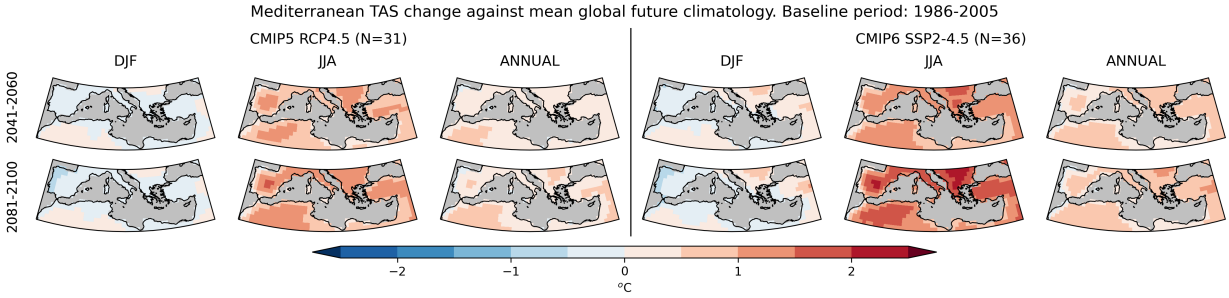


Fig. 174: Mediterranean region temperature change differences against the mean global temperature change. The changes for the periods 2041–2060 (first and third row) and 2081–2100 (second and fourth row) are evaluated against 1986–2005 mean. The differences are shown for the CMIP5 (left) and CMIP6 (right) DJF, JJA and annual mean projections (columns) under the high emission scenario RCP8.5 and SSP5-8.5 respectively. N indicates the number of models included in the ensemble mean.

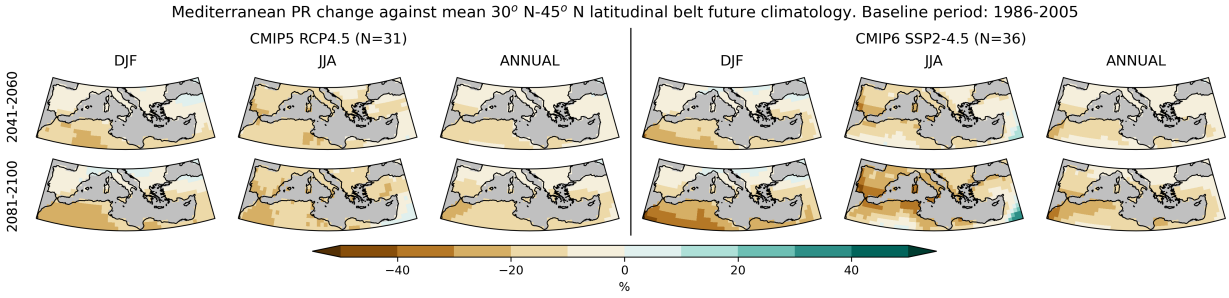


Fig. 175: Mediterranean region precipitation change differences against the mean 30–45º N latitudinal belt precipitation change respectively. The changes for the periods 2041–2060 (first and third row) and 2081–2100 (second and fourth row) are evaluated against 1986–2005 mean. The differences are shown for the CMIP5 (left) and CMIP6 (right) DJF, JJA and annual mean projections (columns) under the high emission scenario RCP8.5 and SSP5-8.5 respectively. N indicates the number of models included in the ensemble mean.

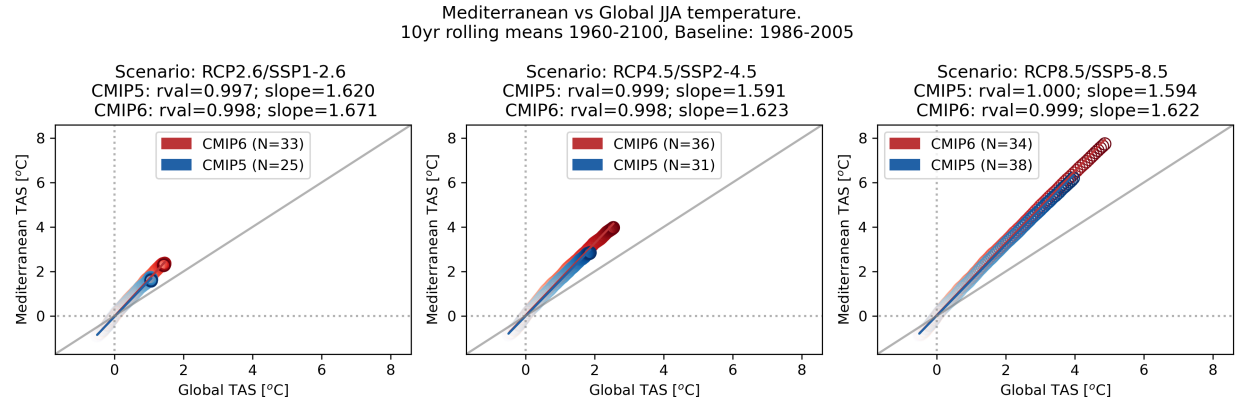


Fig. 176: Mediterranean region warming against global warming for the summer 2.6, 4.5 and 8.5 *Wm^{-2}* RCP and SSP scenarios for the CMIP5 and CMIP6 ensemble means. Each dot represents a 10-year mean change beginning from the period 1960-1969 (light colouring) until 2091-2100 (opaque coloring). The changes are computed with 1986-2005 as the baseline. An ordinary least squares linear regression is computed and the slope and r values are shown. N indicates the number of models included in the ensemble mean.

Available recipes and diagnostics

Recipes are stored in recipes/clouds

- recipe_bock24acp_fig3-4_maps.yml
- recipe_bock24acp_fig6_zonal.yml
- recipe_bock24acp_fig7_boxplots.yml

Diagnostics are stored in diag_scripts/

Fig. 3 and 4:

- clouds/clouds_ecs_groups_maps.py: Geographical maps of the multi-year annual means for group means of historical CMIP simulations from all three ECS groups.

Fig. 6:

- clouds/clouds_ecs_groups_zonal.py: Zonally averaged group means.

Fig. 7:

- clouds/clouds_ecs_groups_boxplots.py: Boxplots of relative changes for all groups.

User settings in recipe

1. Script clouds_ecs_groups_maps.py

   Required settings (scripts)

   reference: if true, a reference dataset is given within 'variable_group' equal 'OBS'

   Optional settings (scripts)

   plot_each_model: one figure for each single model

2. Script clouds/clouds_ecs_groups_zonal.py

   Required settings (scripts)

   group_by: list of 'variable_group's to have the order plot_type: 'zonal' and 'height' plots are available

   Optional settings (scripts)

   filename_attach: attachment to the output files

3. Script clouds/clouds_ecs_groups_boxplots.py

   Required settings (scripts)

   exclude_datasets: list of datasets which are not used for the statistics, default is ['MultiModelMean', 'MultiModelP5', 'MultiModelP95'] group_by: list of 'variable_group's to have the order plot_type: 'zonal' and 'height' plots are available

   Optional settings (scripts)

   filename_attach: attachment to the output files title: set title of figure y_range: set range of the y-axes

Variables

- clt (atmos, monthly, longitude latitude time)
- clivi (atmos, monthly, longitude latitude time)
- clwvi (atmos, monthly, longitude latitude time)
- rlut (atmos, monthly, longitude latitude time)

- rsut (atmos, monthly, longitude latitude time)
- rlutcs (atmos, monthly, longitude latitude time)
- rsutcs (atmos, monthly, longitude latitude time)
- tas (atmos, monthly, longitude latitude time)

Observations and reformat scripts

- CERES-EBAF (Ed4.2) - TOA radiation fluxes (used for calculation of the cloud radiative effects)

  Reformat script: cmorizers/data/formatters/datasets/ceres_ebaf.py

References

- Bock, L. and Lauer, A.: Cloud properties and their projected changes in CMIP models with low to high climate sensitivity, Atmos. Chem. Phys., 24, 1587–1605, https://doi.org/10.5194/acp-24-1587-2024, 2024.

Example plots

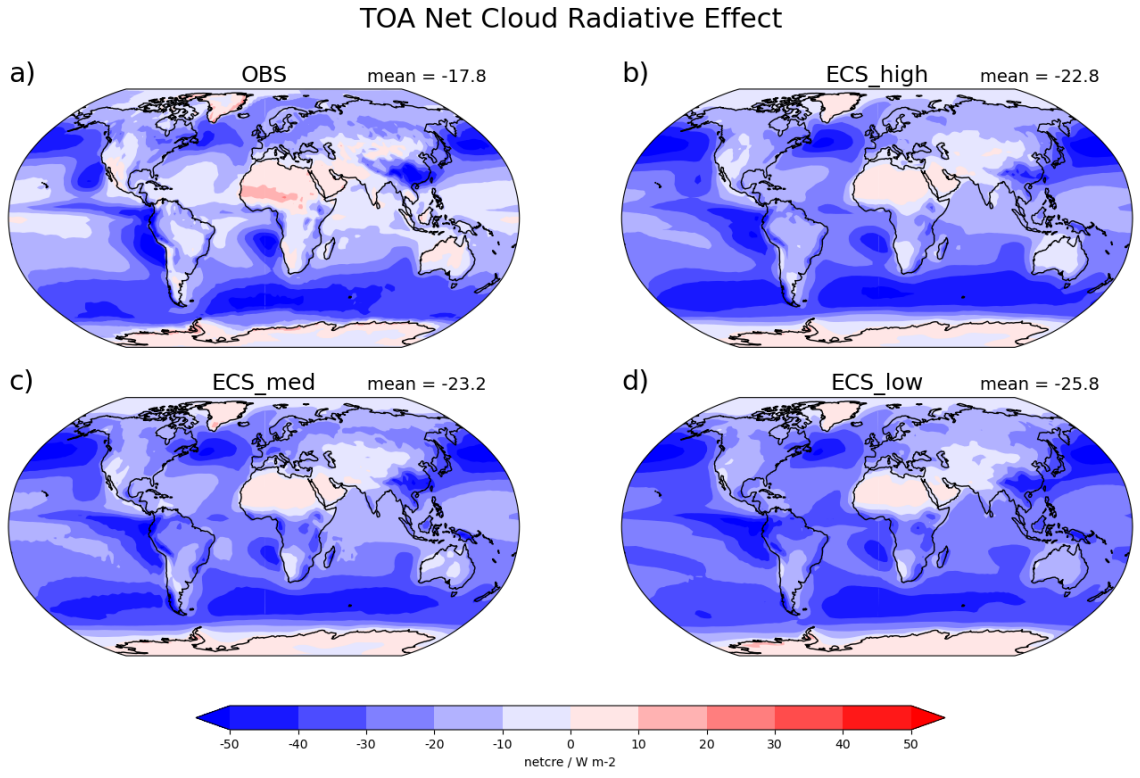


Fig. 177: Geographical map of the multi-year annual mean net cloud radiative effect from (a) CERES–EBAF Ed4.2 (OBS) and (b–d) group means of historical CMIP simulations from all three ECS groups (Fig. 4).

5.4.15 Zero Emissions Commitment (ZEC)

Overview

The Zero Emissions Commitment (ZEC) quantifies the change in global mean temperature expected to occur after net carbon dioxide (CO_2) emissions cease. ZEC is therefore important to consider when estimating the remaining carbon

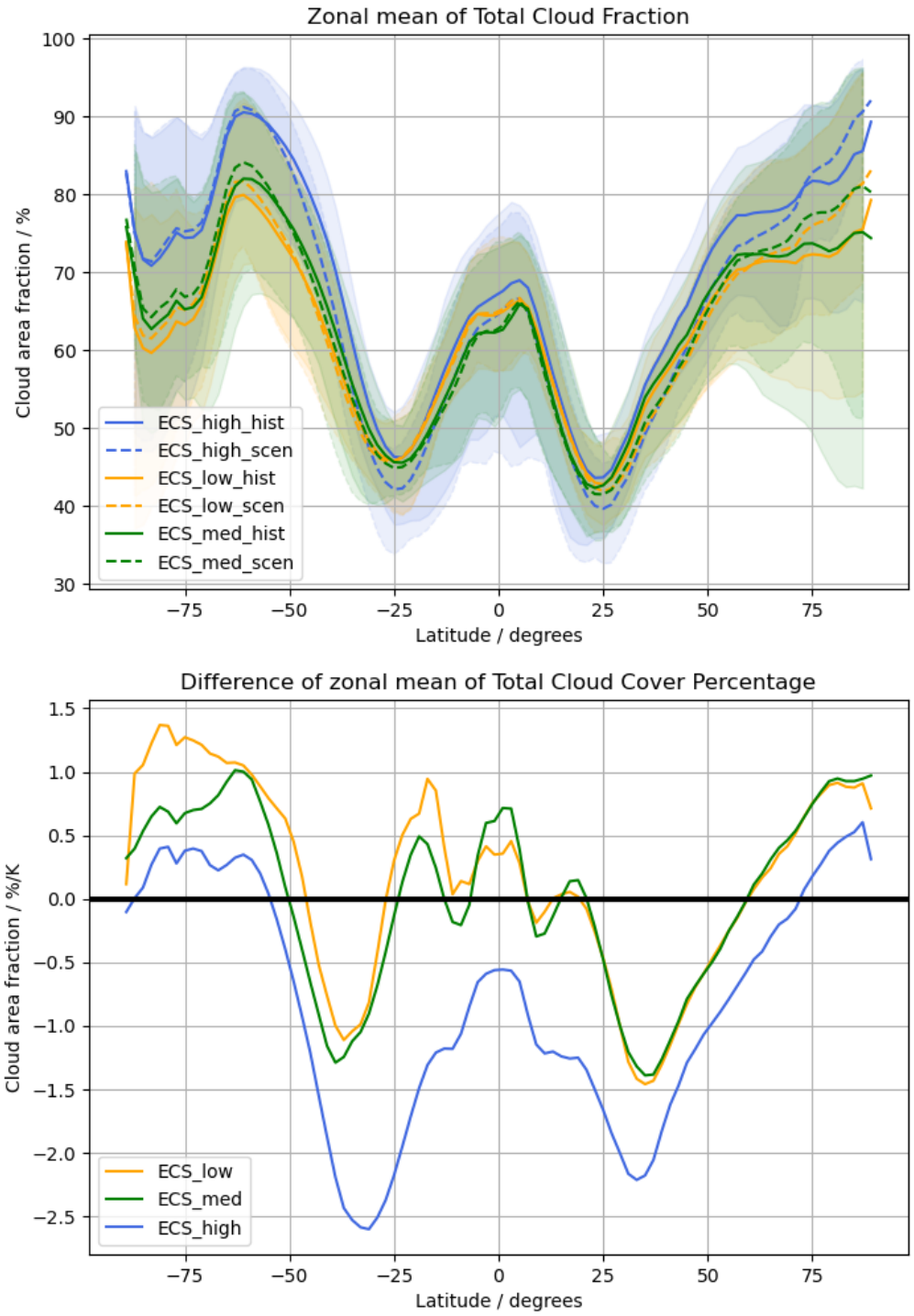


Fig. 178: The upper panel show the zonally averaged group means of total cloud fraction from historical simulations (solid lines) and RCP8.5/SSP5-8.5 scenarios (dashed lines) for the three different ECS groups. Lower panels show the corresponding relative differences of all zonally averaged group means between the RCP8.5/SSP5-8.5 scenarios and the corresponding historical simulations. Shading indicates the 5% and 95% quantiles of the single model results (Fig. 6a).

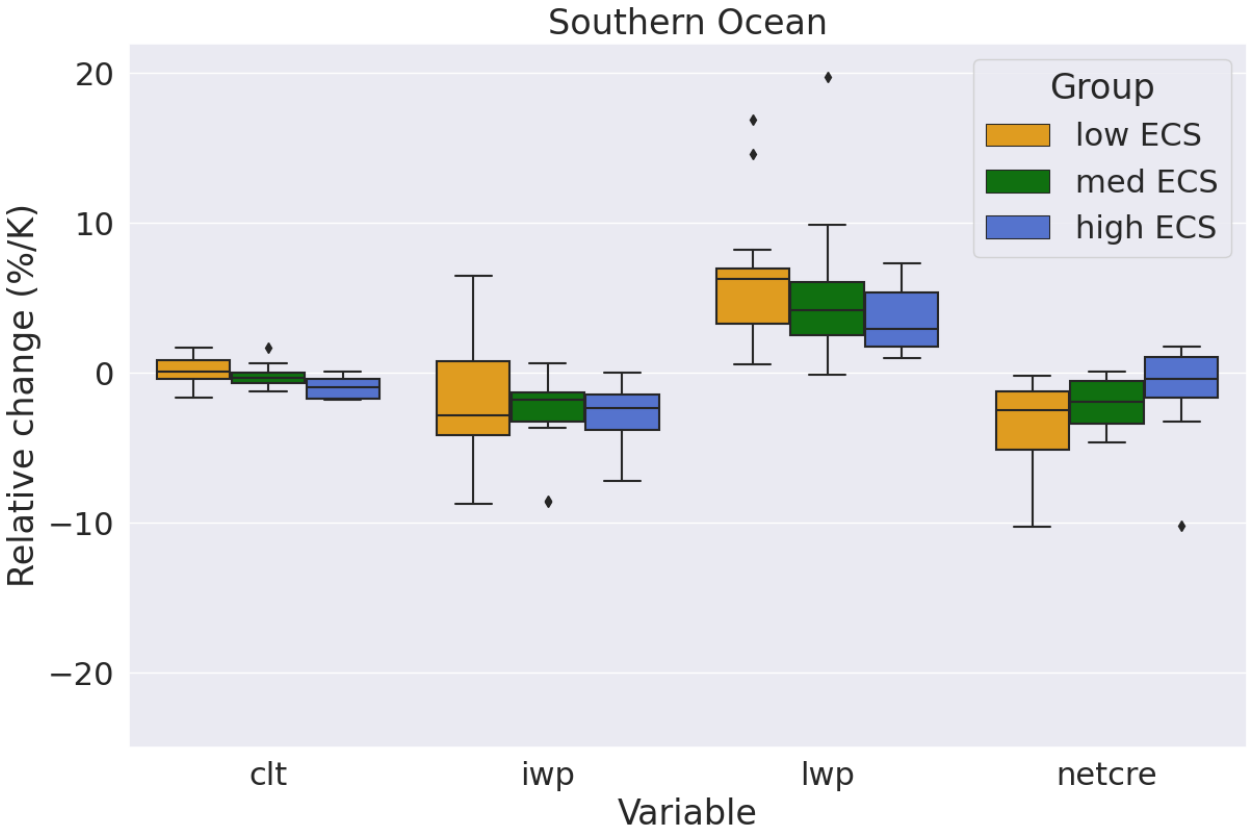


Fig. 179: Relative change (calculated as the difference between the scenario value and the historical value divided by the historical value) of total cloud fraction (clt), ice water path (iwp), liquid water path (lwp), and net cloud radiative effect (netcre) per degree of warming averaged over the Southern Ocean (30–65°S). In the box plot, each box indicates the range from the first quartile to the third quartile, the vertical line shows the median, and the whiskers the minimum and maximum values, excluding the outliers. Outliers are defined as being outside 1.5 times the interquartile range (Fig. 7b).

budget. Calculation of ZEC requires dedicated simulations with emissions set to zero, branching off a base simulation with emissions. In CMIP6 the simulations were part of ZECMIP, with the simulations called `esm-1pct-brch-xPgC` branching off the `1pctCO2` simulation when emissions reach x PgC. The default x was 1000PgC, with additional simulations for 750PgC and 2000PgC. In CMIP7, ZEC simulations (`esm-flat10-zec`) are part of the fast track and branch off (`esm-flat10`) with constant emissions of 10GtC/yr at year 100 (Sanderson 2024).

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_zec.yml

Diagnostics are stored in diag_scripts/

- *climate_metrics/zec.py*

User settings in recipe

Note: The preprocessor settings should not be changed. When changing the recipe, ensure that the reference simulation period (generally `1pctCO2` or `esm-flat10`, using the `anomaly_base` preprocessor block) is the 20-year period around the time the ZEC simulation branches off of. This point may vary in CMIP6 for 1pctCO2, while in CMIP7 it should be at year 100 of `esm-flat10`.

- Preprocessor
  - `area_statistics` (*operation: mean*): Calculate global mean.
  - `annual_statistics` (*operation: mean*): For the ZEC simulation, compute annual statistics.
  - `climate_statistics` (*operation: mean, period: full*): For the reference simulation, compute the 20-year time average centered around the time when the ZEC simulation starts.
- Script climate_metrics/zec.py
  - `zec_year`, *list*, optional (default: `[50]`): Calculate ZEC for the 20-year average centered around year x, multiple values are possible. Barplots are generated for all ZEC_x.
  - `experiments`, *dict*, optional: When using non-default experiments to calculate ZEC, the experiment setting is required with values for `reference` and `simulation`. Accepted by default are: `reference: esm-flat10, 1pctCO2 simulation: esm-flat10-zec, esm-1pct-brch-1000PgC`

Variables

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

Observations and reformat scripts

None

References

- MacDougall, A. H., Frölicher, T. L., Jones, C. D., Rogelj, J., Matthews, H. D., Zickfeld, K., Arora, V. K., Barrett, N. J., Brovkin, V., Burger, F. A., Eby, M., Eliseev, A. V., Hajima, T., Holden, P. B., Jeltsch-Thömmes, A., Koven, C., Mengis, N., Menviel, L., Michou, M., Mokhov, I. I., Oka, A., Schwinger, J., Séférian, R., Shaffer, G., Sokolov, A., Tachiiri, K., Tjiputra, J., Wiltshire, A., and Ziehn, T.: Is there warming in the pipeline? A multi-model analysis of the Zero Emissions Commitment from CO_2, Biogeosciences, 17, 2987–3016, https://doi.org/10.5194/bg-17-2987-2020, 2020.
- Sanderson, B. M., Brovkin, V., Fisher, R., Hohn, D., Ilyina, T., Jones, C., Koenigk, T., Koven, C., Li, H., Lawrence, D., Lawrence, P., Liddicoat, S., Macdougall, A., Mengis, N., Nicholls, Z., O'Rourke, E., Romanou,

A., Sandstad, M., Schwinger, J., Seferian, R., Sentman, L., Simpson, I., Smith, C., Steinert, N., Swann, A., Tjiputra, J., and Ziehn, T.: flat10MIP: An emissions-driven experiment to diagnose the climate response to positive, zero, and negative CO_2 emissions, EGUsphere [preprint], https://doi.org/10.5194/egusphere-2024-3356, 2024.

Example plots

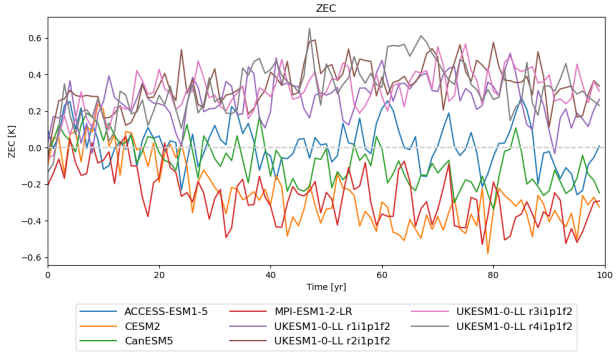


Fig. 180: Time series of ZEC - temperature change after cessation of emissions.

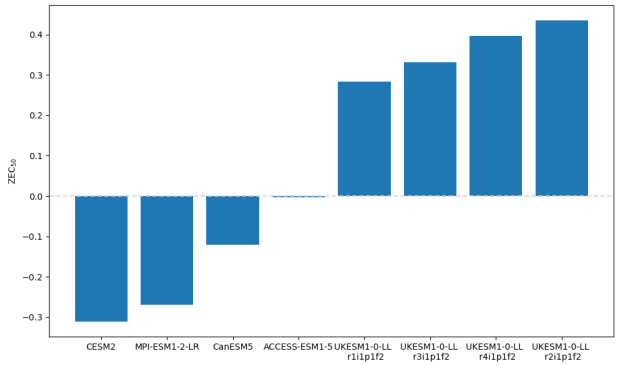


Fig. 181: Barplot for ascending values of Zec_x, with x = 50 in this case, symbolizing the Zero Emissions Commitment 50 years after emissions cease.

5.4.16 Physical Climate at Global Warming Levels (GWLs)

Overview

This recipe calculates years of Global Warming Level (GWL) exceedances in CMIP models as described in *Swaminathan et al (2022)*. Time series of the anomalies in annual global mean surface air temperature (GSAT) are calculated with respect to the 1850-1900 time-mean of each individual time series. To limit the influence of short-term variability, a 21-year centered running mean is applied to the time series. The year at which the time series exceeds warming levels or temperatures such as 1.5C is then recorded for the specific model ensemble member and future scenario. Once the years of exceedance are calculated, the time averaged global mean and standard deviation for the multimodel ensemble over the 21-year period around the year of exceedance are plotted. By selecting specific scenarios, the multimodel mean and spread for a single future scenario can be plotted as shown in the examples below.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_calculate_gwl_exceedance_stats.yml

Diagnostics are stored in esmvaltool/diag_scripts/

- gwls/calculate_gwl_exceedance_years.py
- gwls/plot_gwl_exceedance_mm_stats.py

User settings in recipe

- Preprocessors
  - `calculate_anomalies`
    - `custom_order` (*bool: True*) : Follow order of preprocessors as given.
    - `area_statistics` (*operator: mean*) : Calculate area averaged means.
    - `annual_statistics` (*operator: mean*): Calculate annual means.
    - **`anomalies`: Calculate anomalies for the full time period and with** respect to the reference period.
    - `extract_time` : Extracting time period to calculate time series for.
  - `multi_model_gwl_stats`
    - `custom_order` (*bool: True*) : Follow order of preprocessors as given.
    - `extract_time` : Extract time for the period of the time series calculation.
    - `annual_statistics` (*operator: mean*): Calculate annual means.
    - `regrid`: Regrid to a common resolution so multimodel means can be calculated.
- Script calculate_gwl_exceedance_years.py
  - `window_size` : Number of years to average over to smooth the time series.
  - `gwls`: Global warming levels for which years of exceedances are to be calculated.
- Script plot_gwl_exceedance_mm_stats.py
  - **`ancestors`**
    [Output file from the GWL exceedance calculation step and] preprocessed output for the physical variable.
  - `quickplot` : Plotting options.
    - `plot_type` : For recording provenance information on type of plot.
    - `cmap_mean`: Colormap for the mean map contour plot.
    - `cmap_stdev`: Colormap for the standard deviation map contour plot.
    - `title_var` : Variable name for the plot title.
    - `mean_level_params`: Start, end and step size for the levels in the contour map plot. Values are floats and used by the numpy arange function.
    - `stdev_level_params`: As above but for the standard deviation plots.

Variables

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

Observations and reformat scripts

- None used but can be included.

References

- Swaminathan, R., R. J. Parker, C. G. Jones, R. P. Allan, T. Quaife, D. I. Kelley, L. de Mora, and J. Walton, 2022: The Physical Climate at Global Warming Thresholds as Seen in the U.K. Earth System Model. J. Climate, 35, 29–48, https://doi.org/10.1175/JCLI-D-21-0234.1.

Example plots

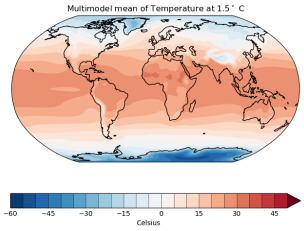


Fig. 182: Multimodel mean of temperature under SSP1-2.6 at 1.5 degC warming.

5.5 IPCC

5.5.1 IPCC AR6 Chapter 3 (selected figures)

Overview

This recipe collects selected diagnostics used in IPCC AR6 WGI Chapter 3: Human influence on the climate system (Eyring et al., 2021). Plots from IPCC AR6 can be readily reproduced and compared to previous versions. The aim is to be able to start with what was available now the next time allowing us to focus on developing more innovative analysis methods rather than constantly having to "re-invent the wheel".

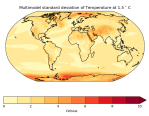


Fig. 183: Multimodel standard deviation of temperature under SSP1-2.6 at 1.5 degC warming.

Processing of CMIP3 models currently works only in serial mode, due to an issue in the input data still under investigation. To run the recipe for Fig 3.42a and Fig. 3.43 set the configuration option `max_parallel_tasks: 1`.

The plots are produced collecting the diagnostics from individual recipes. The following figures from Eyring et al. (2021) can currently be reproduced:

- Figure 3.3 a,b,c,d: Surface Air Temperature - Model Bias
- Figure 3.4: Anomaly Of Near-Surface Air Temperature
- Figure 3.5: Temporal Variability Of Near-Surface Air Temperature
- Figure 3.9: Anomaly Of Near-Surface Air Temperature - Attribution
- Figure 3.13: Precipitation - Model Bias
- Figure 3.15: Precipitation Anomaly
- Figure 3.19: Speed-Up Of Zonal Mean Wind
- Figure 3.24: Biases In Zonal Mean And Equatorial Sea Surface Temperature
- Figure 3.42: Relative Model Performance
- Figure 3.43: Correlation Pattern

To reproduce Fig. 3.9 you need the shapefile of the AR6 reference regions (Iturbide et al., 2020). Please download the file IPCC-WGI-reference-regions-v4_shapefile.zip, unzip and store it in *<auxiliary_data_dir>/IPCC-regions/* (where `auxiliary_data_dir` is given as configuration option).

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/ipccwg1ar6ch3/

- recipe_ipccwg1ar6ch3_atmosphere.yml
- recipe_ipccwg1ar6ch3_fig_3_9.yml
- recipe_ipccwg1ar6ch3_fig_3_19.yml
- recipe_ipccwg1ar6ch3_fig_3_24.yml
- recipe_ipccwg1ar6ch3_fig_3_42_a.yml
- recipe_ipccwg1ar6ch3_fig_3_42_b.yml
- recipe_ipccwg1ar6ch3_fig_3_43.yml

Diagnostics are stored in esmvaltool/diag_scripts/

Fig. 3.3:

- ipcc_ar5/ch12_calc_IAV_for_stippandhatch.ncl: See *here:*.
- ipcc_ar6/model_bias.ncl

Fig. 3.4:

- ipcc_ar6/tas_anom.ncl
- ipcc_ar6/tsline_collect.ncl

Fig. 3.5:

- ipcc_ar6/zonal_st_dev.ncl

Fig. 3.9:

- ipcc_ar6/tas_anom_damip.ncl

Fig. 3.13:

- ipcc_ar5/ch12_calc_IAV_for_stippandhatch.ncl: See *here:*.
- ipcc_ar6/model_bias.ncl

Fig. 3.15:

- ipcc_ar6/precip_anom.ncl

Fig. 3.19:

- ipcc_ar6/zonal_westerly_winds.ncl

Fig. 3.24: * ocean/diagnostic_biases.py

Fig. 3.42:

- perfmetrics/main.ncl
- perfmetrics/collect.ncl

Fig. 3.43:

- ipcc_ar6/corr_pattern.ncl

- ipcc_ar6/corr_pattern_collect.ncl

User settings in recipe

1. Script ipcc_ar5/ch12_calc_IAV_for_stippandhatch.ncl

   See *here*.

2. Script ipcc_ar6/model_bias.ncl

   Optional settings (scripts)

   - plot_abs_diff: additionally also plot absolute differences (true, false)
   - plot_rel_diff: additionally also plot relative differences (true, false)
   - plot_rms_diff: additionally also plot root mean square 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 dataset

   Color tables

   - variable "tas" and "tos": diag_scripts/shared/plot/rgb/ipcc-ar6_temperature_div.rgb, diag_scripts/shared/plot/rgb/ipcc-ar6_temperature_10.rgb, diag_scripts/shared/plot/rgb/ipcc-ar6_temperature_seq.rgb
   - variable "pr": diag_scripts/shared/plots/rgb/ipcc-ar6_precipitation_seq.rgb, diag_scripts/shared/plot/rgb/ipcc-ar6_precipitation_10.rgb
   - variable "sos": diag_scripts/shared/plot/rgb/ipcc-ar6_misc_seq_1.rgb, diag_scripts/shared/plot/rgb/ipcc-ar6_misc_div.rgb

3. Script ipcc_ar6/tas_anom.ncl

   Required settings for script

   - styleset: as in diag_scripts/shared/plot/style.ncl functions

   Optional settings for script

   - blending: if true, calculates blended surface temperature
   - 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
   - header: if true, region name as header
   - volcanoes: if true, adds volcanoes to the plot
   - write_stat: if true, write multi model statistics in nc-file

Optional settings for variables

- reference_dataset: reference dataset; REQUIRED when calculating anomalies

Color tables

- e.g. diag_scripts/shared/plot/styles/cmip5.style

4. Script ipcc_ar6/tas_anom_damip.ncl

   Required settings for script

   - start_year: start year in figure
   - end_year: end year in figure
   - panels: list of variable blocks for each panel

   Optional settings for script

   - ref_start: start year of reference period for anomalies
   - ref_end: end year of reference period for anomalies
   - ref_mask: if true, model fields will be masked by reference fields
   - plot_units: variable unit for plotting
   - y-min: set min of y-axis
   - y-max: set max of y-axis
   - header: title for each panel
   - title: name of region as part of filename
   - legend: set labels for optional output of a legend in an extra file

5. Script ipcc_ar6/tsline_collect.ncl

   Optional settings for script

   - blending: if true, then var="gmst" otherwise "gsat"
   - ref_start: start year of reference period for anomalies
   - ref_end: end year of reference period for anomalies
   - region: name of domain
   - plot_units: variable unit for plotting
   - y-min: set min of y-axis
   - y-max: set max of y-axis
   - order: order in which experiments should be plotted
   - stat_shading: if true: shading of statistic range
   - ref_shading: if true: shading of reference period

   Optional settings for variables

   - reference_dataset: reference dataset; REQUIRED when calculating anomalies

6. Script ipcc_ar6/zonal_st_dev.ncl

   Required settings for script

   - styleset: as in diag_scripts/shared/plot/style.ncl functions

Optional settings for script

- plot_legend: if true, plot legend will be plotted
- plot_units: variable unit for plotting
- multi_model_mean: if true, multi-model mean and uncertainty will be plotted

Optional settings for variables

- reference_dataset: reference dataset; REQUIRED when calculating anomalies

7. Script ipcc_ar6/precip_anom.ncl

   Required settings for script

   - panels: list of variables plotted in each panel
   - start_year: start of time coordinate
   - end_year: end of time coordinate

   Optional settings for script

   - anomaly: true if anomaly should be calculated
   - ref_start: start year of reference period for anomalies
   - ref_end: end year of reference period for anomalies
   - ref_mask: if true, model fields will be masked by reference fields
   - region: name of domain
   - plot_units: variable unit for plotting
   - header: if true, region name as header
   - stat: statistics for multi model nc-file (MinMax,5-95,10-90)
   - y_min: set min of y-axis
   - y_max: set max of y-axis

8. Script ipcc_ar6/zonal_westerly_winds.ncl

   Optional settings for variables

   - reference_dataset: reference dataset; REQUIRED when calculating anomalies

   Optional settings for script

   - e13fig12_start_year: year when the climatology calculation starts (default: start_year of var)
   - e13fig12_end_year: year when the climatology calculation ends (default: end_year of var)
   - e13fig12_multimean: multimodel mean (default: False)
   - e13fig12_exp_MMM: name of the experiments for the MMM (required if @e13fig12_multimean = True)
   - e13fig12_season: season (default: ANN)

9. Script perfmetrics/perfmetrics_main.ncl

   See *here*.

10. Script perfmetrics/perfmetrics_collect.ncl

    See *here*.

11. Script ipcc_ar6/corr_pattern.ncl

    Required settings for variables

    - reference_dataset: name of reference observation

    Optional settings for variables

    - alternative_dataset: name of alternative observations

12. Script ipcc_ar6/corr_pattern_collect.ncl

    Optional settings for script

    - diag_order: give order of plotting variables on the x-axis
    - labels: List of labels for each variable on the x-axis
    - model_spread: if True, model spread is shaded
    - plot_median: if True, median is plotted
    - project_order: give order of projects

13. Script ocean/diagnostic_biases.py

    Required settings for variables

    - reference_dataset: name of reference observation

    Required settings for script

    - data_statistics: a dictionary with the statistics to be calculated for each variable group. Should contain keywords 'best_guess' and 'borders'. 'borders' should be a list with two statistics. The statistics values are the same as 'operator' as in the preprocessors.

    Optional settings for script

    - bias: boolean flag, indicating, if bias should be calculated. If none provided, absolute values will be used.
    - mask: a dictionary with the mask information. The accepted keywords are 'flag' (required), 'type' (required) and 'group' (optional). 'flag' is a boolean flag if the mask should be used. 'type' accepts two values: 'simple' and 'resolved'. If 'simple' option is used, the data will be masked to the existing mask from the reference dataset. If 'resolved' is used, the values along the dimension of the data will be masked using the data from the variable group 'group'.
    - mpl_style: name of the matplotlib style file. If none provided, the default style will be used.
    - caption: figure caption. If none, an empty string will be used.
    - color_style: a name of the color_style to be used. If none provided, the default style file will be used.

Variables

- et (land, monthly mean, longitude latitude time)
- fgco2 (ocean, monthly mean, longitude latitude time)
- gpp (land, monthly mean, longitude latitude time)
- hfds (land, monthly mean, longitude latitude time)
- hus (land, monthly mean, longitude latitude level time)
- lai (land, monthly mean, longitude latitude time)
- lwcre (atmos, monthly mean, longitude latitude time)
- nbp (land, monthly mean, longitude latitude time)

- pr (atmos, monthly mean, longitude latitude time)
- psl (atmos, monthly mean, longitude latitude time)
- rlds (atmos, monthly mean, longitude latitude time)
- rlus (atmos, monthly mean, longitude latitude time)
- rlut (atmos, monthly mean, longitude latitude time)
- rsds (atmos, monthly mean, longitude latitude time)
- rsus (atmos, monthly mean, longitude latitude time)
- rsut (atmos, monthly mean, longitude latitude time)
- sm (land, monthly mean, longitude latitude time)
- sic (seaice, monthly mean, longitude latitude time)
- siconc (seaice, monthly mean, longitude latitude time)
- swcre (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)
- ua (atmos, monthly mean, longitude latitude level time)
- va (atmos, monthly mean, longitude latitude level time)
- zg (atmos, monthly mean, longitude latitude level time)

Observations and reformat scripts

- AIRS (hus - obs4MIPs)
- ATSR (tos - obs4MIPs)
- BerkeleyEarth (tasa - esmvaltool/cmorizers/data/formatters/datasets/berkeleyearth.py)
- CERES-EBAF (rlds, rlus, rlut, rlutcs, rsds, rsus, rsut, rsutcs - obs4MIPs)
- CRU (pr - esmvaltool/cmorizers/data/formatters/datasets/cru.py)
- ESACCI-SOILMOISTURE (sm - esmvaltool/cmorizers/data/formatters/datasets /esacci_soilmoisture.py)
- ESACCI-SST (ts - esmvaltool/cmorizers/data/formatters/datasets/esacci_sst.py)
- ERA5 (hus, psl, ta, tas, ua, va, zg - ERA5 data can be used via the native6 project)
- ERA-Interim (hfds - cmorizers/data/formatters/datasets/era_interim.py)
- FLUXCOM (gpp - cmorizers/data/formatters/datasets/fluxcom.py)
- GHCN (pr - esmvaltool/cmorizers/data/formatters/datasets/ghcn.ncl)
- GPCP-SG (pr - obs4MIPs)
- HadCRUT5 (tasa - esmvaltool/cmorizers/data/formatters/datasets/hadcrut5.py)
- HadISST (sic, tos, ts - esmvaltool/cmorizers/data/formatters/datasets/hadisst.ncl)
- JMA-TRANSCOM (fgco2, nbp - esmvaltool/cmorizers/data/formatters/datasets/jma_transcom.py)

- JRA-55 (psl - ana4MIPs)
- Kadow2020 (tasa - esmvaltool/cmorizers/data/formatters/datasets/kadow2020.py)
- LandFlux-EVAL (et - esmvaltool/cmorizers/data/formatters/datasets/landflux_eval.py)
- Landschuetzer2016 (fgco2 - esmvaltool/cmorizers/data/formatters/datasets/landschuetzer2016.py)
- LAI3g (lai - esmvaltool/cmorizers/data/formatters/datasets/lai3g.py)
- MTE (gpp - esmvaltool/cmorizers/data/formatters/datasets/mte.py)
- NCEP-NCAR-R1 (ta, tas, ua, va, zg - esmvaltool/cmorizers/data/formatters/datasets/ncep_ncar_r1.py)
- NOAAGlobalTemp (tasa - esmvaltool/cmorizers/data/formatters/datasets/noaaglobaltemp.py)

References

- Eyring, V., N.P. Gillett, K.M. Achuta Rao, R. Barimalala, M. Barreiro Parrillo, N. Bellouin, C. Cassou, P.J. Durack, Y. Kosaka, S. McGregor, S. Min, O. Morgenstern, and Y. Sun, 2021: Human Influence on the Climate System. In Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Masson-Delmotte, V., P. Zhai, A. Pirani, S.L. Connors, C. Péan, S. Berger, N. Caud, Y. Chen, L. Goldfarb, M.I. Gomis , M. Huang, K. Leitzell, E. Lonnoy, J.B.R. Matthews, T.K. Maycock, T. Waterfield, O. Yelekçi, R. Yu, and B. Zhou (eds.)]. Cambridge Universiy Press, Cambridge, United Kingdom and New York, NY, USA, pp. 423-552, doi: 10.1017/9781009157896.005.

Example plots

5.5.2 IPCC AR5 Chapter 9 (selected figures)

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".

Note

Please note that most recipes have been modified to include only models that are (still) readily available via ESGF. Plots produced may therefore look different than the original figures from IPCC AR5.

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 itself.

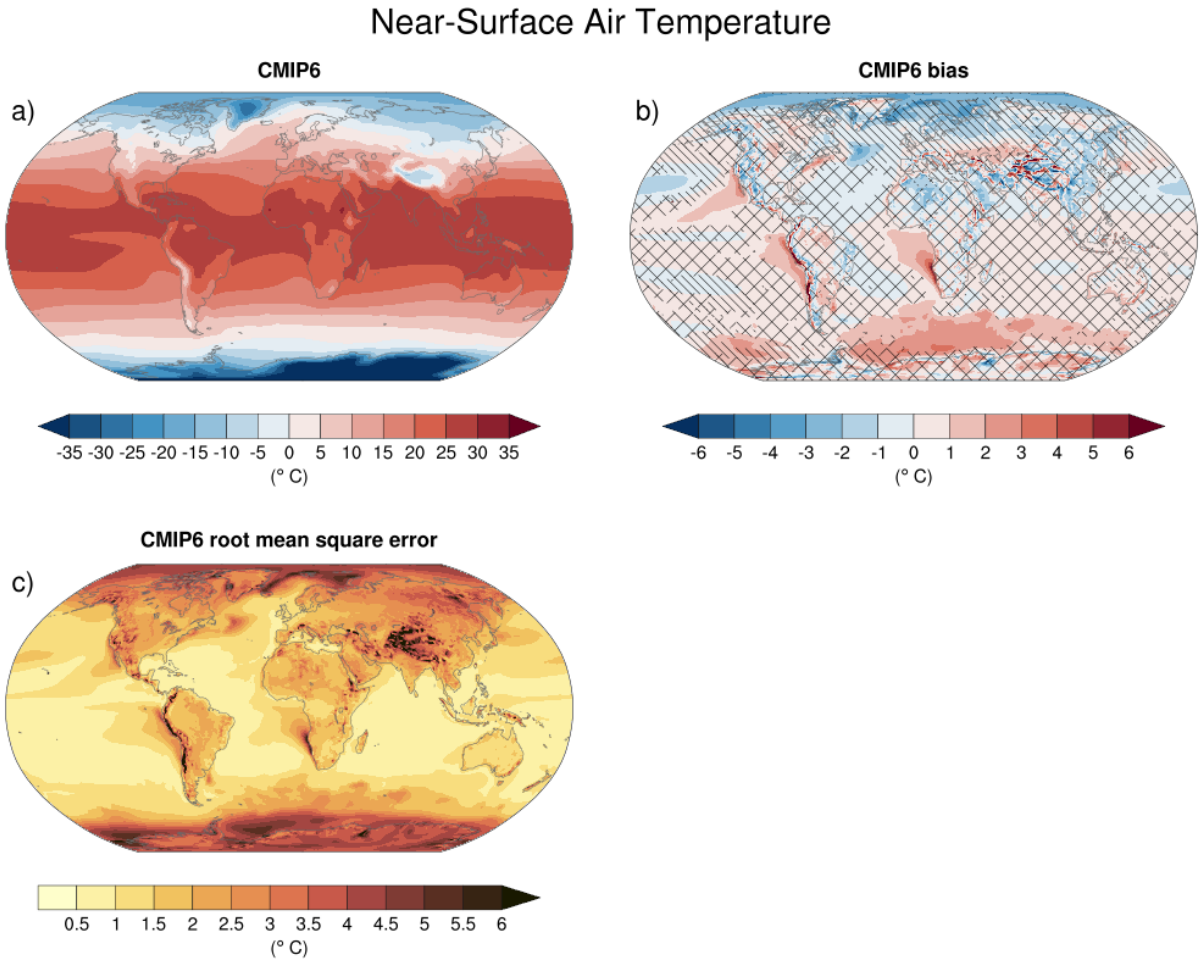


Fig. 184: Figure 3.3: Annual mean near-surface (2 m) air temperature (°C) for the period 1995-2014. (a) Multi-model (ensemble) mean constructed with one realization of the CMIP6 historical experiment from each model. (b) Multi-model mean bias, defined as the difference between the CMIP6 multi-model mean and the climatology of the fifth generation European Centre for Medium-Range Weather Forecasts (ECMWF) atmospheric reanalysis of the global climate (ERA5). (c) Multi-model mean of the root mean square error calculated over all months separately and averaged, with respect to the climatology from ERA5. Uncertainty is represented using the advanced approach: No overlay indicates regions with robust signal, where >=66% of models show change greater than the variability threshold and >=80% of all models agree on sign of change; diagonal lines indicate regions with no change or no robust signal, where <66% of models show a change greater than the variability threshold; crossed lines indicate regions with conflicting signal, where >=66% of models show change greater than the variability threshold and <80% of all models agree on sign of change.

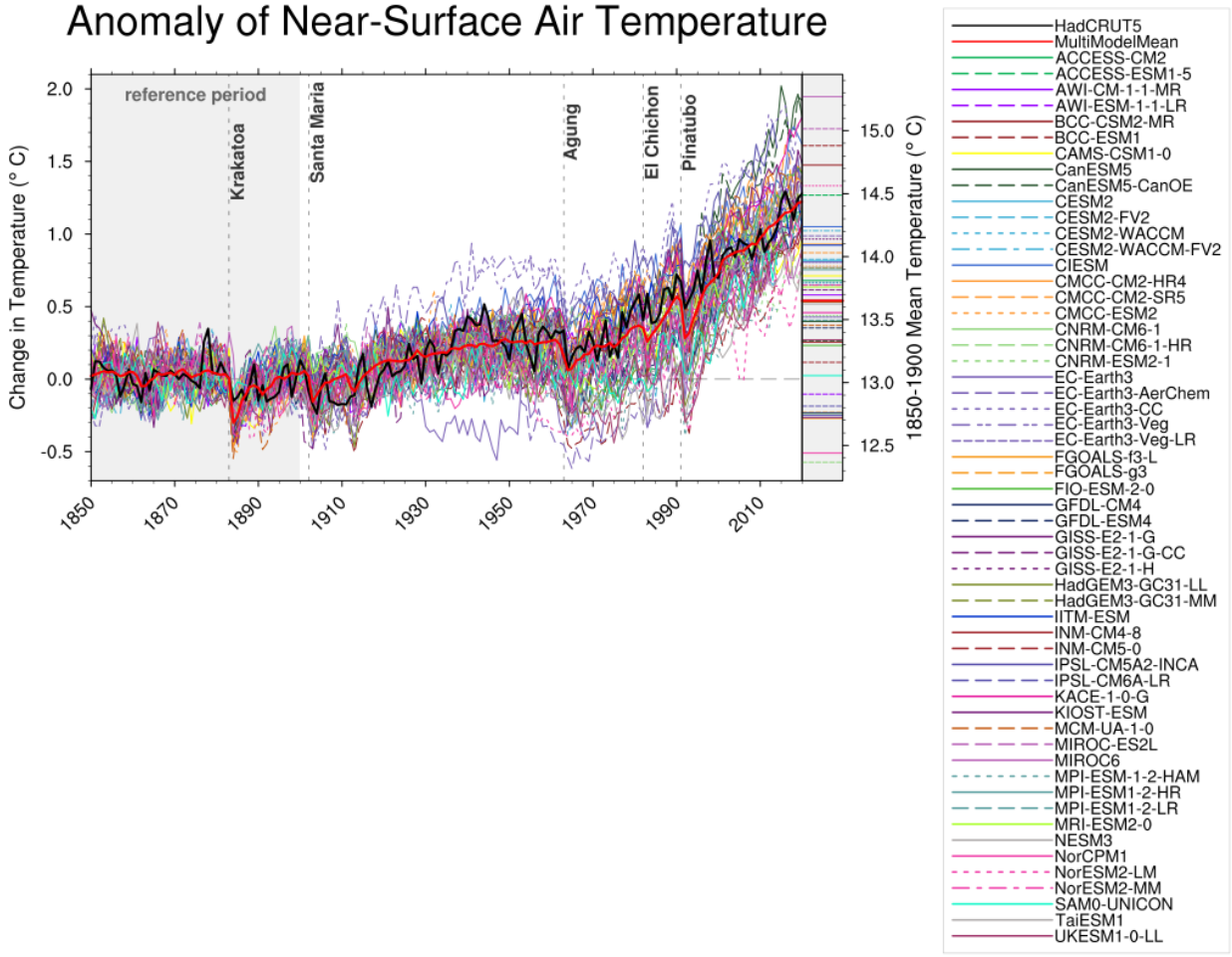


Fig. 185: Figure 3.4a: Observed and simulated time series of the anomalies in annual and global mean surface air temperature (GSAT). All anomalies are differences from the 1850-1900 time-mean of each individual time series. The reference period 1850-1900 is indicated by grey shading. (a) Single simulations from CMIP6 models (thin lines) and the multi-model mean (thick red line). Observational data (thick black lines) are from the Met Office Hadley Centre/Climatic Research Unit dataset (HadCRUT5), and are blended surface temperature (2 m air temperature over land and sea surface temperature over the ocean). All models have been subsampled using the HadCRUT5 observational data mask. Vertical lines indicate large historical volcanic eruptions. Inset: GSAT for each model over the reference period, not masked to any observations.

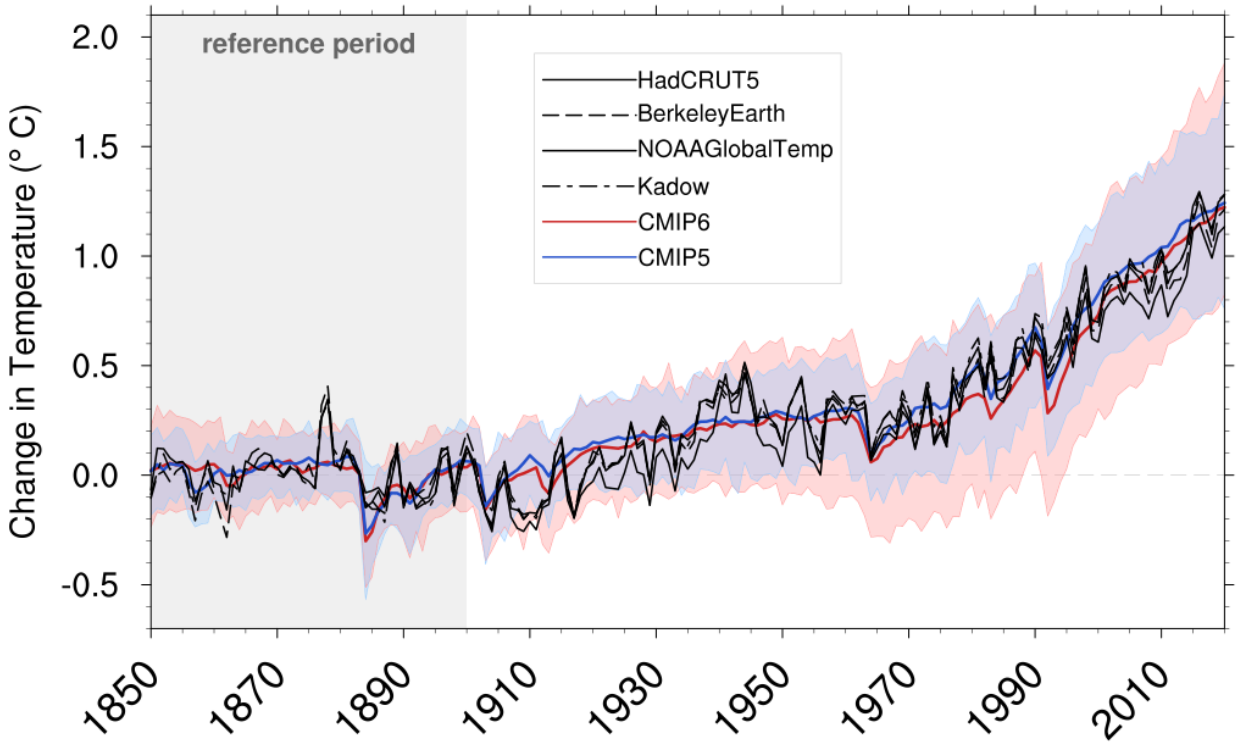


Fig. 186: Figure 3.4b: Observed and simulated time series of the anomalies in annual and global mean surface air temperature (GSAT). All anomalies are differences from the 1850-1900 time-mean of each individual time series. The reference period 1850-1900 is indicated by grey shading. (b) Multi-model means of CMIP5 (blue line) and CMIP6 (red line) ensembles and associated 5th to 95th percentile ranges (shaded regions). Observational data are HadCRUT5, Berkeley Earth, National Oceanic and Atmospheric Administration NOAAGlobalTemp and Kadow et al. (2020). Masking was done as in (a). CMIP6 historical simulations were extended with SSP2-4.5 simulations for the period 2015-2020 and CMIP5 simulations were extended with RCP4.5 simulations for the period 2006-2020. All available ensemble members were used. The multi-model means and percentiles were calculated solely from simulations available for the whole time span (1850-2020).

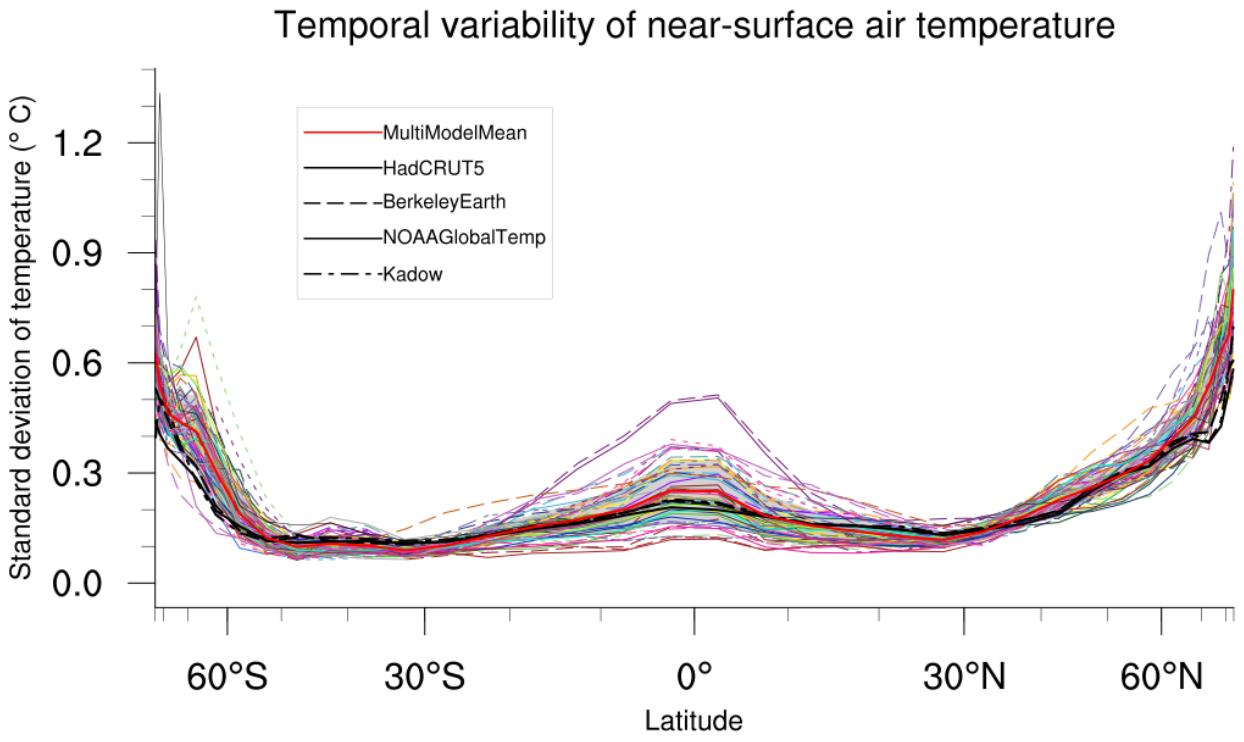


Fig. 187: Figure 3.5: The standard deviation of annually averaged zonal-mean near-surface air temperature. This is shown for four detrended observed temperature datasets (HadCRUT5, Berkeley Earth, NOAAGlobalTemp and Kadow et al. (2020), for the years 1995-2014) and 59 CMIP6 pre-industrial control simulations (one ensemble member per model, 65 years) (after Jones et al., 2013). For line colours see the legend of Figure 3.4. Additionally, the multi-model mean (red) and standard deviation (grey shading) are shown. Observational and model datasets were detrended by removing the least-squares quadratic trend.

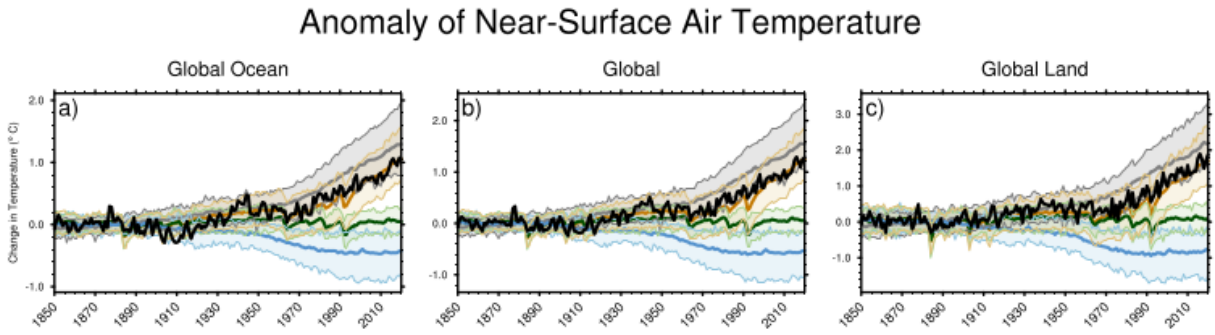


Fig. 188: Figure 3.9: Global, land and ocean annual mean near-surface air temperature anomalies in CMIP6 models and observations. Timeseries are shown for CMIP6 historical anthropogenic and natural (brown) natural-only (green), greenhouse gas only (grey) and aerosol only (blue) simulations (multi-model means shown as thick lines, and shaded ranges between the 5th and 95th percentiles) and for HadCRUT5 (black). All models have been subsampled using the HadCRUT5 observational data mask. Temperature anomalies are shown relative to 1950-2010 for Antarctica and relative to 1850-1900 for other continents. CMIP6 historical simulations are expanded by the SSP2-4.5 scenario simulations. All available ensemble members were used. Regions are defined by Iturbide et al. (2020).

- 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: Centered 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

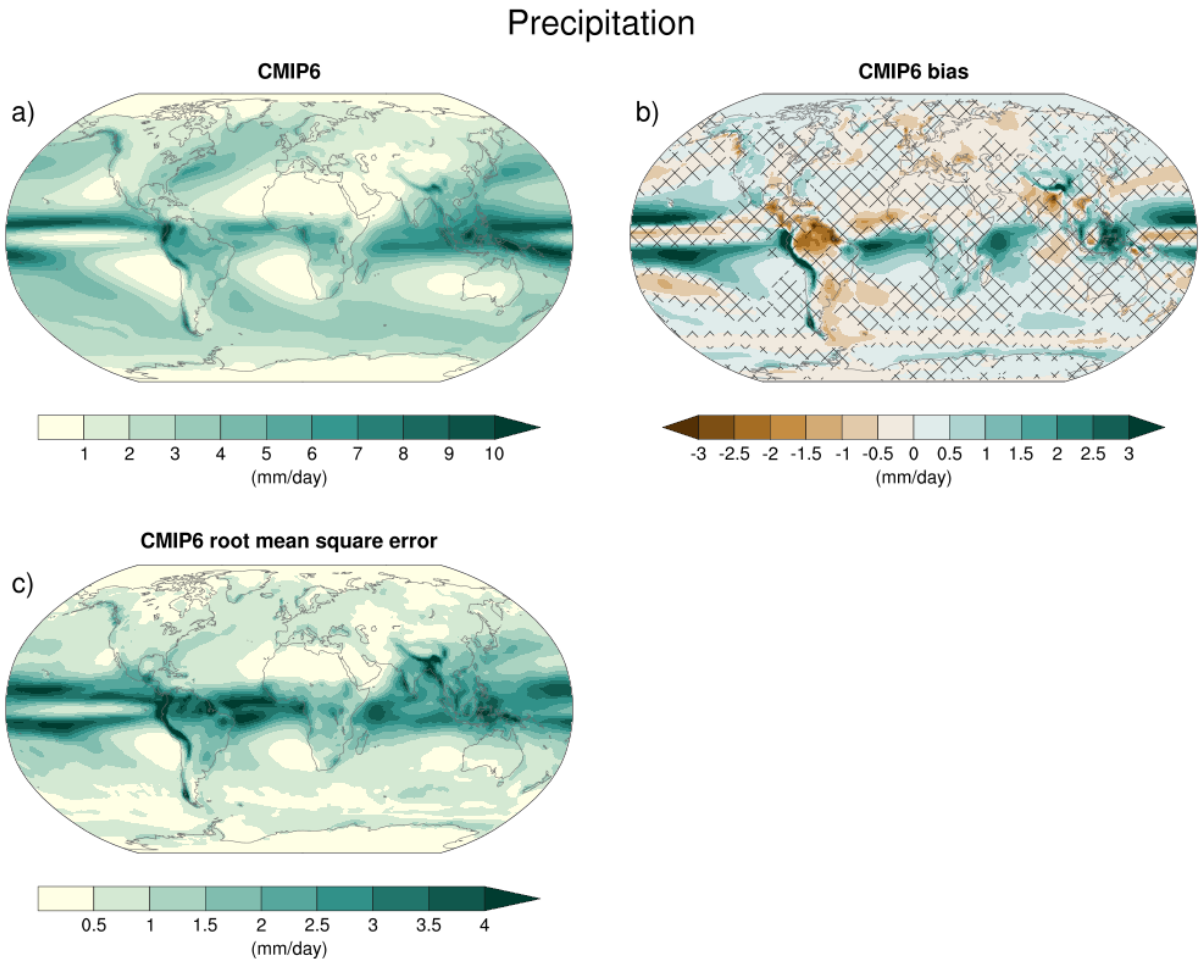


Fig. 189: Figure 3.13: Annual-mean precipitation rate (mm day-1) for the period 1995-2014. (a) Multi-model (ensemble) mean constructed with one realization of the CMIP6 historical experiment from each model. (b) Multi-model mean bias, defined as the difference between the CMIP6 multi-model mean and precipitation analysis from the Global Precipitation Climatology Project (GPCP) version 2.3 (Adler et al., 2003). (c) Multi-model mean of the root mean square error calculated over all months separately and averaged with respect to the precipitation analysis from GPCP version 2.3. Uncertainty is represented using the advanced approach. No overlay indicates regions with robust signal, where >=66% of models show change greater than the variability threshold and >=80% of all models agree on sign of change; diagonal lines indicate regions with no change or no robust signal, where <66% of models show a change greater than the variability threshold; crossed lines indicate regions with conflicting signal, where >=66% of models show change greater than the variability threshold and <80% of all models agree on the sign of change.

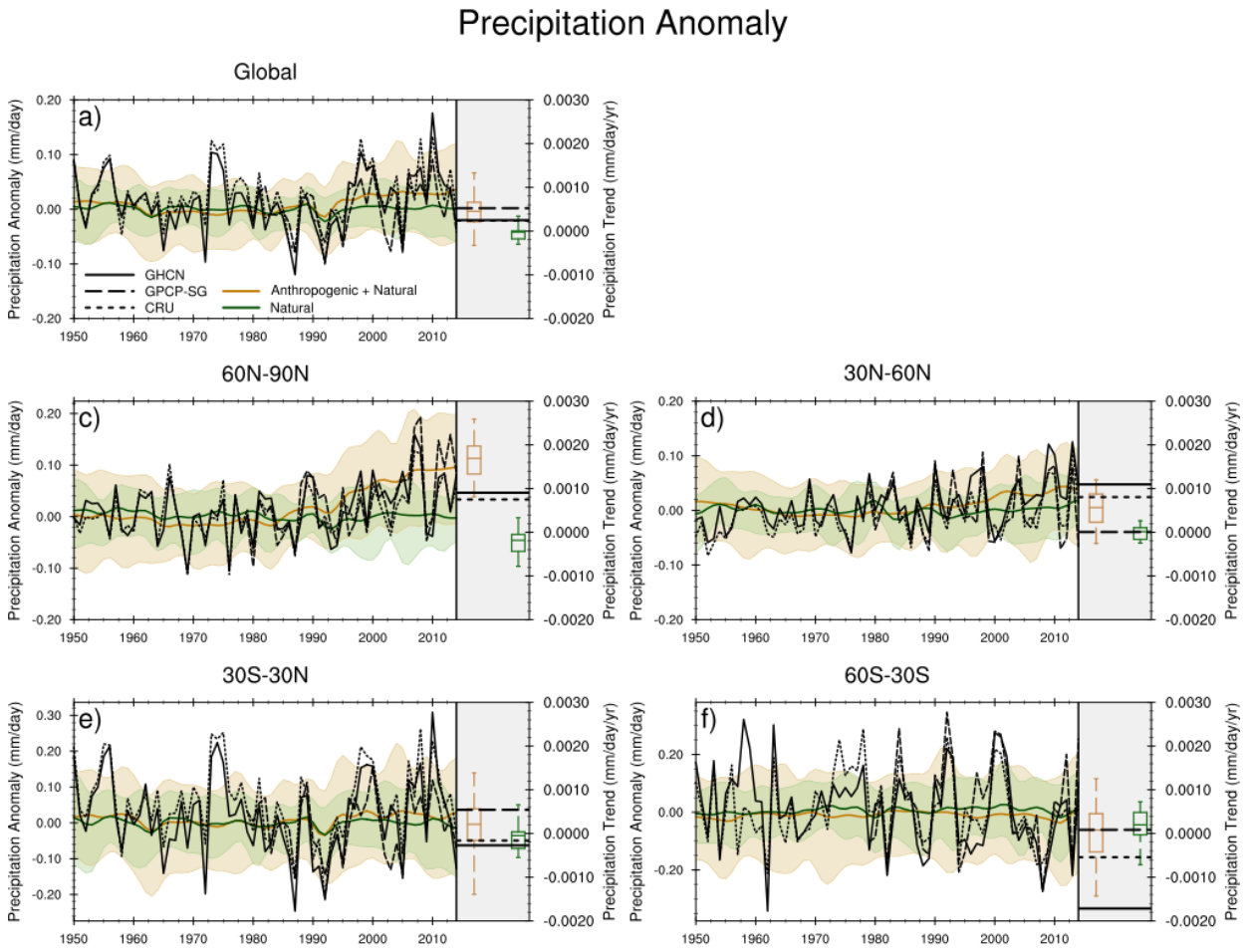


Fig. 190: Figure 3.15: Observed and simulated time series of anomalies in zonal average annual mean precipitation. (a), (c-f) Evolution of global and zonal average annual mean precipitation (mm day-1) over areas of land where there are observations, expressed relative to the base period of 1961-1990, simulated by CMIP6 models (one ensemble member per model) forced with both anthropogenic and natural forcings (brown) and natural forcings only (green). Multi-model means are shown in thick solid lines and shading shows the 5-95% confidence interval of the individual model simulations. The data is smoothed using a low pass filter. Observations from three different datasets are included: gridded values derived from Global Historical Climatology Network (GHCN version 2) station data, updated from Zhang et al. (2007), data from the Global Precipitation Climatology Product (GPCP L3 version 2.3, Adler et al. (2003)) and from the Climate Research Unit (CRU TS4.02, Harris et al. (2014)). Also plotted are boxplots showing interquartile and 5-95% ranges of simulated trends over the period for simulations forced with both anthropogenic and natural forcings (brown) and natural forcings only (blue). Observed trends for each observational product are shown as horizontal lines. Panel (b) shows annual mean precipitation rate (mm day-1) of GHCN version 2 for the years 1950-2014 over land areas used to compute the plots.

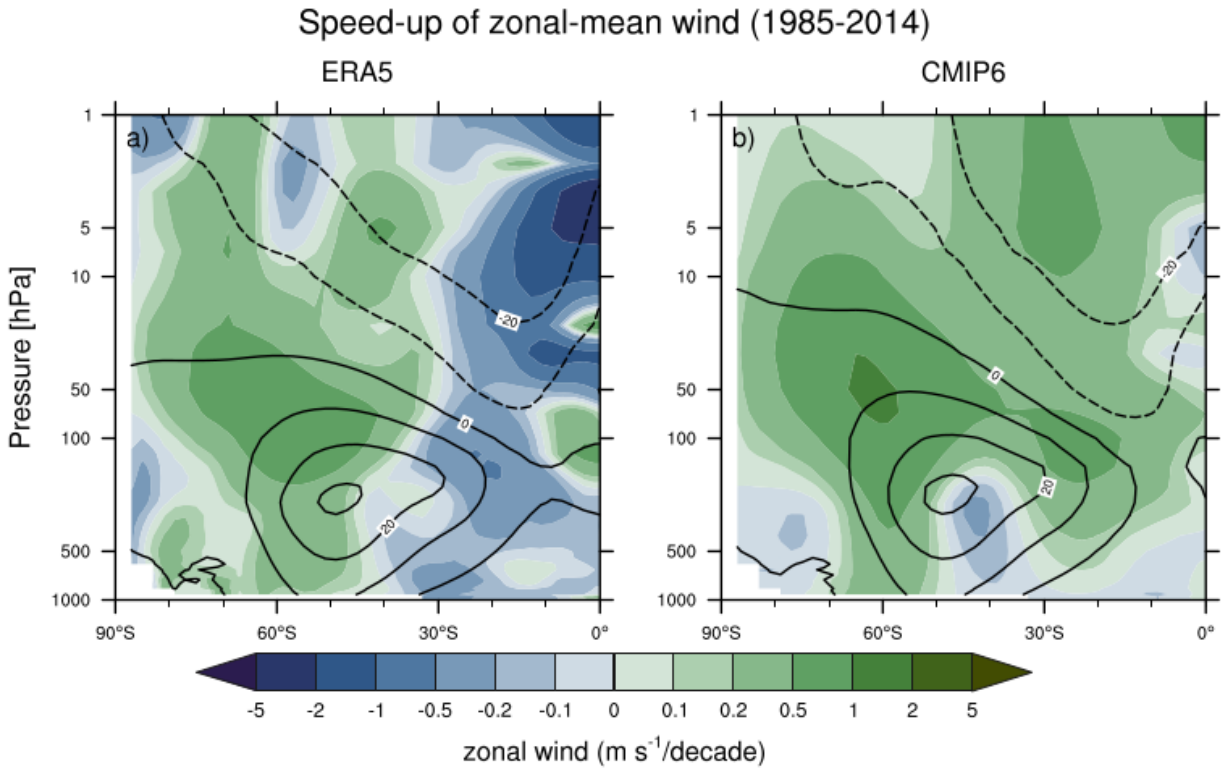


Fig. 191: Figure 3.19: Long-term mean (thin black contours) and linear trend (colour) of zonal mean December-January-February zonal winds from 1985 to 2014 in the Southern Hemisphere. The figure shows (a) ERA5 and (b) the CMIP6 multi-model mean (58 CMIP6 models). The solid contours show positive (westerly) and zero long-term mean zonal wind, and the dashed contours show negative (easterly) long-term mean zonal wind. Only one ensemble member per model is included. Figure is modified from Eyring et al. (2013), their Figure 12.

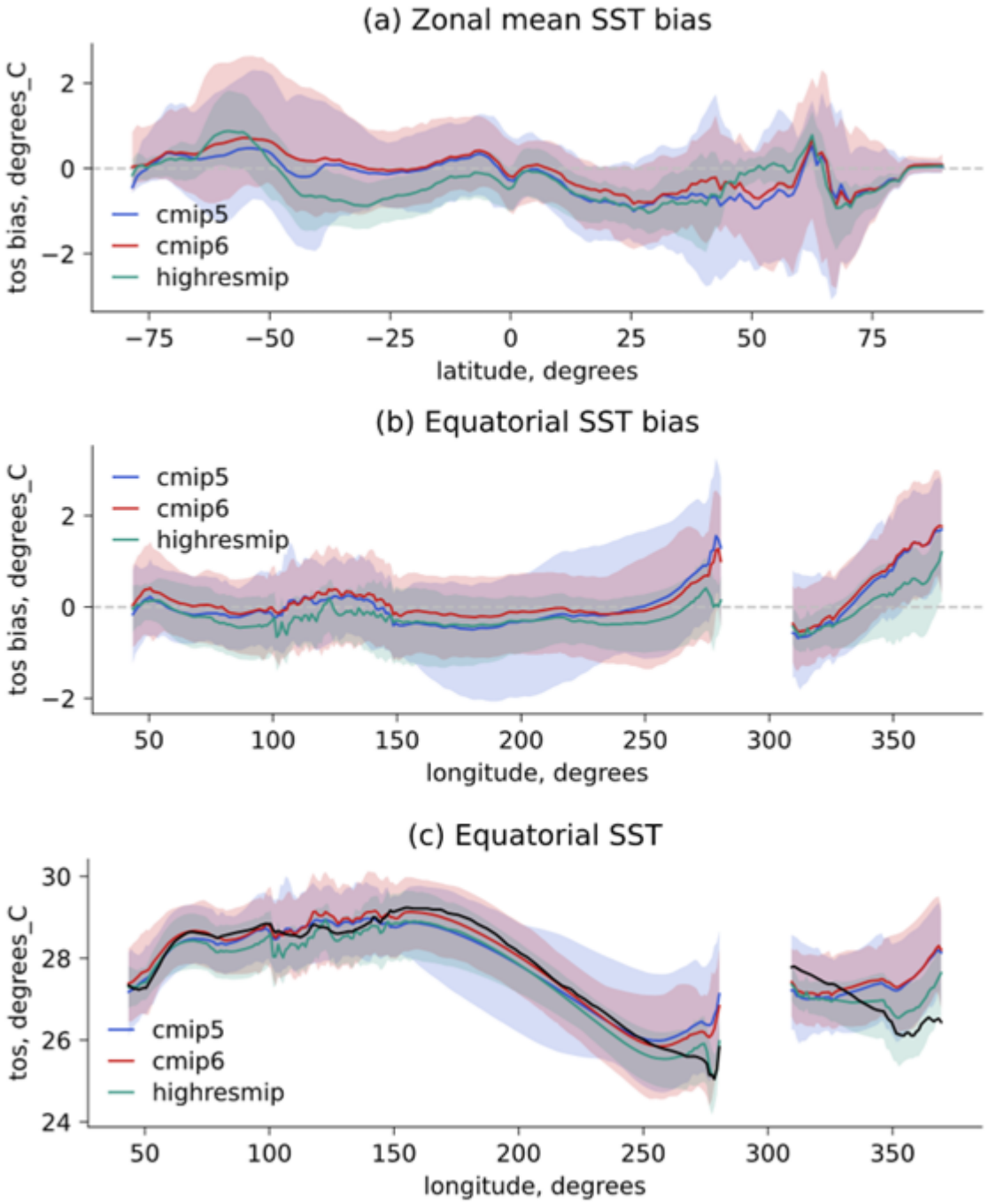


Fig. 192: Figure 3.24: Biases in zonal mean and equatorial sea surface temperature (SST) in CMIP5 and CMIP6 models. CMIP6 (red), CMIP5 (blue) and HighResMIP (green) multi-model mean (a) zonally averaged SST bias; (b) equatorial SST bias; and (c) equatorial SST compared to observed mean SST (black line) for 1979–1999. The inter-model 5th and 95th percentiles are depicted by the respective shaded range. Model climatologies are derived from the 1979–1999 mean of the historical simulations, using one simulation per model. The Hadley Centre Sea Ice and Sea Surface Temperature version 1 (HadISST) observational climatology for 1979–1999 is used as the reference for the error calculation in (a) and (b); and for observations in (c). (The panels were obtained individually and combined together.)

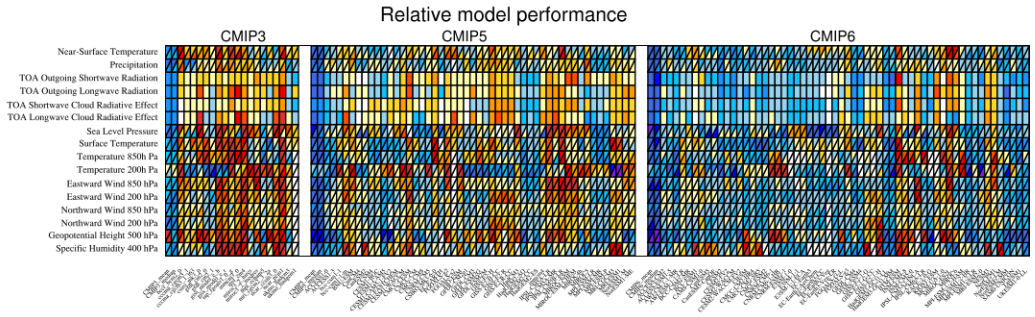


Fig. 193: Figure 3.42a: Relative space-time root-mean-square deviation (RMSD) calculated from the climatological seasonal cycle of the CMIP simulations (1980-1999) compared to observational datasets. A relative performance measure is displayed, with blue shading indicating better and red shading indicating worse performance than the median error 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 an additional data set (upper left triangle). Reference/additional datasets are from top to bottom in (a): ERA5/NCEP, GPCP-SG/GHCN, CERES-EBAF, CERES-EBAF, CERES-EBAF, CERES-EBAF, JRA-55/ERA5, ESACCI-SST/HadISST, ERA5/NCEP, ERA5/NCEP, ERA5/NCEP, ERA5/NCEP, ERA5/NCEP, ERA5/NCEP, AIRS/ERA5, ERA5/NCEP. White boxes are used when data are not available for a given model and variable. Figure is updated and expanded from Bock et al. (2020).

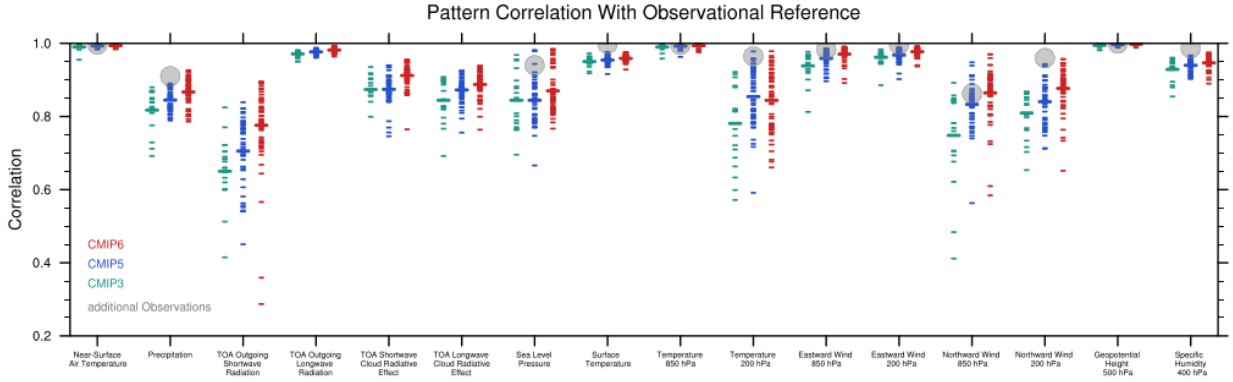


Fig. 194: Figure 3.43 | Centred pattern correlations between models and observations for the annual mean climatology over the period 1980-1999. Results are shown for individual CMIP3 (green), CMIP5 (blue) and CMIP6 (red) models (one ensemble member from each model is used) as short lines, along with the corresponding multi-model ensemble averages (long lines). Correlations are shown between the models and the primary reference observational data set (from left to right: ERA5, GPCP-SG, CERES-EBAF, CERES-EBAF, CERES-EBAF, CERES-EBAF, JRA-55, ESACCI-SST, ERA5, ERA5, ERA5, ERA5, ERA5, ERA5, AIRS, ERA5). In addition, the correlation between the primary reference and additional observational datasets (from left to right: NCEP, GHCN, -, -, -, -, ERA5, HadISST, NCEP, NCEP, NCEP, NCEP, NCEP, NCEP, NCEP, ERA5) are shown (solid grey circles) if available. To ensure a fair comparison across a range of model resolutions, the pattern correlations are computed after regridding all datasets to a resolution of 4° in longitude and 5° latitude.

fluxes, calculated as the standard deviation of the annual means over the period 1986–2005.

- Figure 9.38: Seasonal cycle for the surface temperature or precipitation over land within defined regions multi-model mean and difference to reference dataset or absolute annual cycle can be chosen.
- Figure 9.39: Seasonal bias box and whiskers plot for surface temperature or precipitation within SREX (IPCC Special Report on Managing the Risks of Extreme Events and Disasters to Advance Climate Change Adaptation) regions.
- Figure 9.40: Seasonal bias box and whiskers plot for surface temperature or precipitation within defined polar and ocean regions.
- Figure 9.41b: Comparison between observations and models for variable values within defined regions.
- 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).

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/recipe_ipccwg1ar5ch9

- recipe_flato13ipcc_figures_92_95.yml: Figures 9.2, 9.3, 9.4, 9.5
- recipe_flato13ipcc_figure_96.yml: Figure 9.6
- recipe_flato13ipcc_figure_98.yml: Figure 9.8
- recipe_flato13ipcc_figure_914.yml: Figure 9.14
- recipe_flato13ipcc_figure_924.yml: Figure 9.24
- recipe_flato13ipcc_figures_926_927.yml: Figures 9.26 and 9.27
- recipe_flato13ipcc_figure_942.yml: Figure 9.42
- recipe_flato13ipcc_figure_945a.yml: Figure 9.45a
- recipe_flato13ipcc_figures_938_941_cmip3.yml: Figures 9.38, 9.39, 9.40, and 9.41
- recipe_flato13ipcc_figures_938_941_cmip6.yml: Figures 9.38, 9.39, 9.40, and 9.41 CMIP6 instead of CMIP3
- recipe_weigel21gmd_figures_13_16.yml: ESMValTool paper version (Weigel et al., 2021) of Figures 9.38, 9.39, 9.40, and 9.41, only CMIP5

Diagnostics are stored in esmvaltool/diag_scripts/

- carbon_cycle/main.ncl: See *here*.
- climate_metrics/ecs.py: See *here*.
- clouds/clouds_bias.ncl: global maps of the multi-model mean and the multi-model mean bias (Fig. 9.2, 9.4)
- clouds/clouds_ipcc.ncl: 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)
- regional_downscaling/Figure9_38.ncl (Fig 9.38a (variable tas) and Fig 9.38b (variable pr))
- regional_downscaling/Figure9_39.ncl (Fig 9.39a/c/e (variable tas) and Fig 9.39b/d/f (variable pr))
- regional_downscaling/Figure9_40.ncl (Fig 9.40a/c/e (variable tas) and Fig 9.40b/d/f (variable pr))
- regional_downscaling/Figure9_41.ncl (Fig 9.41b)
- 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)

User settings in recipe

1. Script carbon_cycle/main.ncl

   See *here*.

2. Script climate_metrics/ecs.py

   See *here*.

3. Script clouds/clouds_bias.ncl

   Required settings (scripts)

   none

   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 dataset

   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

4. Script clouds/clouds_ipcc.ncl

   Required settings (scripts)

   none

   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 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 grid 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

none

Optional settings for variables

- reference_dataset: reference dataset; REQUIRED when calculating anomalies

Color tables

- e.g. diag_scripts/shared/plot/styles/cmip5.style

6. Script ipcc_ar5/ch09_fig09_3.ncl

   Required settings for script

   none

   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

7. 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

8. Script ipcc_ar5/ch09_fig09_6_collect.ncl

   Required settings for script

   none

   Optional settings for script

   - diag_order: List of diagnostic names in the order variables should appear on x-axis

9. 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

10. 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)

11. Script regional_downscaling/Figure9.38.ncl

    Required settings for script

    none

    Optional settings (scripts)

    - styleset: for plot_type cycle (e.g. CMIP5, CMIP6), default "CMIP5"
    - fig938_region_label: Labels for regions, which should be included (["WNA", "ENA", "CAM", "TSA", "SSA", "EUM", "NAF","CAF", "SAF", "NAS", "CAS", "EAS", "SAS", "SEA", "AUS"]), default "WNA"
    - fig938_project_MMM: projects to average, default "CMIP5"
    - fig938_experiment_MMM: experiments to average, default "historical"
    - fig938_mip_MMM: mip to average, default "Amon"
    - fig938_names_MMM: names in legend i.e. (["CMIP5","CMIP3"]), default fig938_project_MMM
    - fig938_colors_MMM: Color for multi-model mean (e.g. ["red"]), default "red"
    - If set fig938_mip_MMM, fig938_experiment_MMM, fig938_project_MMM, fig938_names_MMM, and fig938_colors_MMM must have the same number of elements
    - fig938_refModel: Reference data set for differences default "ERA-Interim"
    - fig938_MMM: Plot multi-model mean (true/false), default "true"
    - fig938_YMin: minimum Y Axis
    - fig938_YMax: maximum Y Axis
    - fig938_diff: Difference to reference data (true) or absolute annual cycle (false), default "true"

12. Script regional_downscaling/Figure9.39.ncl

    Required settings (scripts)

    - reference_dataset: reference dataset name (set of variables)

    Optional settings (scripts)

    - styleset: for plot_type cycle (e.g. CMIP5, CMIP6), default "CMIP5"
    - fig939_season: seasons i.e. (["DJF","JJA","ANN"]), default "DJF"
    - fig939_region_label: Labels for regions, which should be included (["ALAs","CGIs","WNAs","CNAs","ENAs","CAMs","AMZs","NEBs","WSAs","SSAs", "NEUs","CEUs","MEDs","SAHs","WAFs","EAFs","SAFs","NASs","WASs","CASs", "TIBs","EASs","SASs","SEAs","NAUs","SAUs"]), default "ALAs"
    - fig939_project_MMM: projects to average, default "CMIP5"
    - fig939_experiment_MMM: experiments to average, default "historical"
    - fig939_mip_MMM: mip to average, default "Amon"
    - fig939_MMM: Plot multi-model mean (true/false)
    - fig939_names_MMM: names in legend i.e. (["CMIP5","CMIP3"])
    - fig939_YMin: minimum Y Axis

- fig939_YMax: maximum Y Axis
- fig939_vert_line_pos: i.e. ([6,10,13,17,24,26])
- fig939_vert_line_label: labels of vertical lines, i.e. (["North America", "South America","Europe","Africa","Asia","Australia"])
- fig939_mode: True= cumulative mode

13. Script regional_downscaling/Figure9.40.ncl

Required settings (scripts)

- reference_dataset: reference dataset name (set of variables)

Optional settings (scripts)

- styleset: for plot_type cycle (e.g. CMIP5, CMIP6), default "CMIP5"
- fig940_season: seasons i.e. (["DJF","JJA","ANN"]), default "DJF"
- fig940_region_label: Labels for regions, which should be included (["Arctic_land","Arctic_sea","Antarctic_land","Antarctic_sea", "Caribbean","WesternIndianOcean","NorthernIndianOcean", "NorthernTropicalPacific","EquatorialTropicalPacific", "SouthernTropicalPacific","World_land","World_sea","World"]), default "Arctic_land"
- fig940_project_MMM: projects to average, default "CMIP5"
- fig940_experiment_MMM: experiments to average, default "historical"
- fig940_mip_MMM: mip to average, default "Amon"
- fig940_MMM: Plot multi-model mean (true/false)
- fig940_names_MMM: names in legend i.e. (["CMIP5","CMIP3"])
- fig940_YMin: minimum Y Axis
- fig940_YMax: maximum Y Axis
- fig940_vert_line_pos: i.e. ([6,10,13,17,24,26])
- fig940_vert_line_label: labels of vertical lines, i.e. (["North America", "South America","Europe","Africa","Asia","Australia"])
- fig940_mode: True= cumulative mode

14. Script regional_downscaling/Figure9.41.ncl

Required settings (scripts)

- reference_dataset: reference dataset name (set of variables)

Optional settings (scripts)

- styleset: for plot_type cycle (e.g. CMIP5, CMIP6), default "CMIP5"
- fig941_region_label: Labels for regions, which should be included (["ALAs","CGIs","WNAs","CNAs","ENAs","CAMs","AMZs","NEBs","WSAs","SSAs", "NEUs","CEUs","MEDs","SAHs","WAFs","EAFs","SAFs","NASs","WASs","CASs", "TIBs","EASs","SASs","SEAs","NAUs","SAUs"]), default "MEDs"

15. Script ipcc_ar5/ch09_fig09_42a.py

Required settings for script

none

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.set_theme() (affects all plots).

1. Script ipcc_ar5/ch09_fig09_42b.py

   Required settings for script

   none

   Optional settings for script

   - dataset_style: Dataset style file (located in esmvaltool.diag_scripts.shared.plot.styles_python). The entry marker is ignored when marker_file is given.
   - log_x: Apply logarithm to X axis (ECS).
   - log_y: Apply logarithm to Y axis (TCR).
   - marker_column: Name of the column to look up markers in marker_file.
   - marker_file: CSV file with markers (can also be integers). Must have the columns dataset and marker (or the column specified by marker_column). If a relative path is given, assumes that this is a pattern to search for ancestor files.
   - savefig_kwargs: Keyword arguments for matplotlib.pyplot.savefig().
   - seaborn_settings: Options for seaborn.set_theme() (affects all plots).
   - x_lim: Plot limits for X axis (ECS).
   - y_lim: Plot limits for Y axis (TCR).

2. 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

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)

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/data/formatters/datasets/era-interim.py)
- GCP2018 (fgco2, nbp - esmvaltool/cmorizers/data/formatters/datasets/gcp2018.py)
- GPCP-SG (pr - obs4MIPs)
- JMA-TRANSCOM (fgco2, nbp - esmvaltool/cmorizers/data/formatters/datasets/jma_transcom.py)
- HadCRUT4 (tas - esmvaltool/cmorizers/data/formatters/datasets/hadcrut4.ncl)
- HadISST (sic, tos - esmvaltool/cmorizers/data/formatters/datasets/hadisst.ncl)
- ISCCP-FH (rsuscs, rsdscs, rsdt - esmvaltool/cmorizers/data/formatters/datasets/isccp_fh.ncl)

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.

- Seneviratne, S. I., Nicholls, N., Easterling, D., Goodess, C. M., Kanae, S., Kossin, J., Luo, Y., Marengo, J., McInnes, K., Rahimi, M., Reichstein, M., Sorteberg, A., Vera, C., and Zhang, X., 2012: Changes in climate extremes and their impacts on the naturalphysical environment. , in: Managing the Risks of Extreme Events and Disasters to Advance Climate Change Adaptation. A Special Report of Working Groups I and II of the Intergovernmental Panel on ClimateChange (IPCC), edited by: Field, C. B., Barros, V., Stocker, T. F., Qin, D., Dokken, D. J., Ebi, K. L., Mastrandrea, M. D., Mach, K. J., Plattner, G.-K., Allen, S. K., Tignor, M., and Midgley, P. M., Cambridge University Press, Cambridge, UK, and New York, NY, USA, 109-230.
- Weigel, K., Bock, L., Gier, B. K., Lauer, A., Righi, M., Schlund, M., Adeniyi, K., Andela, B., Arnone, E., Berg, P., Caron, L.-P., Cionni, I., Corti, S., Drost, N., Hunter, A., Lledó, L., Mohr, C. W., Paçal, A., Pérez-Zanón, N., Predoi, V., Sandstad, M., Sillmann, J., Sterl, A., Vegas-Regidor, J., von Hardenberg, J., and Eyring, V.: Earth System Model Evaluation Tool (ESMValTool) v2.0 - diagnostics for extreme events, regional and impact evaluation, and analysis of Earth system models in CMIP, Geosci. Model Dev., 14, 3159-3184, https://doi.org/10.5194/gmd-14-3159-2021, 2021.

Example plots

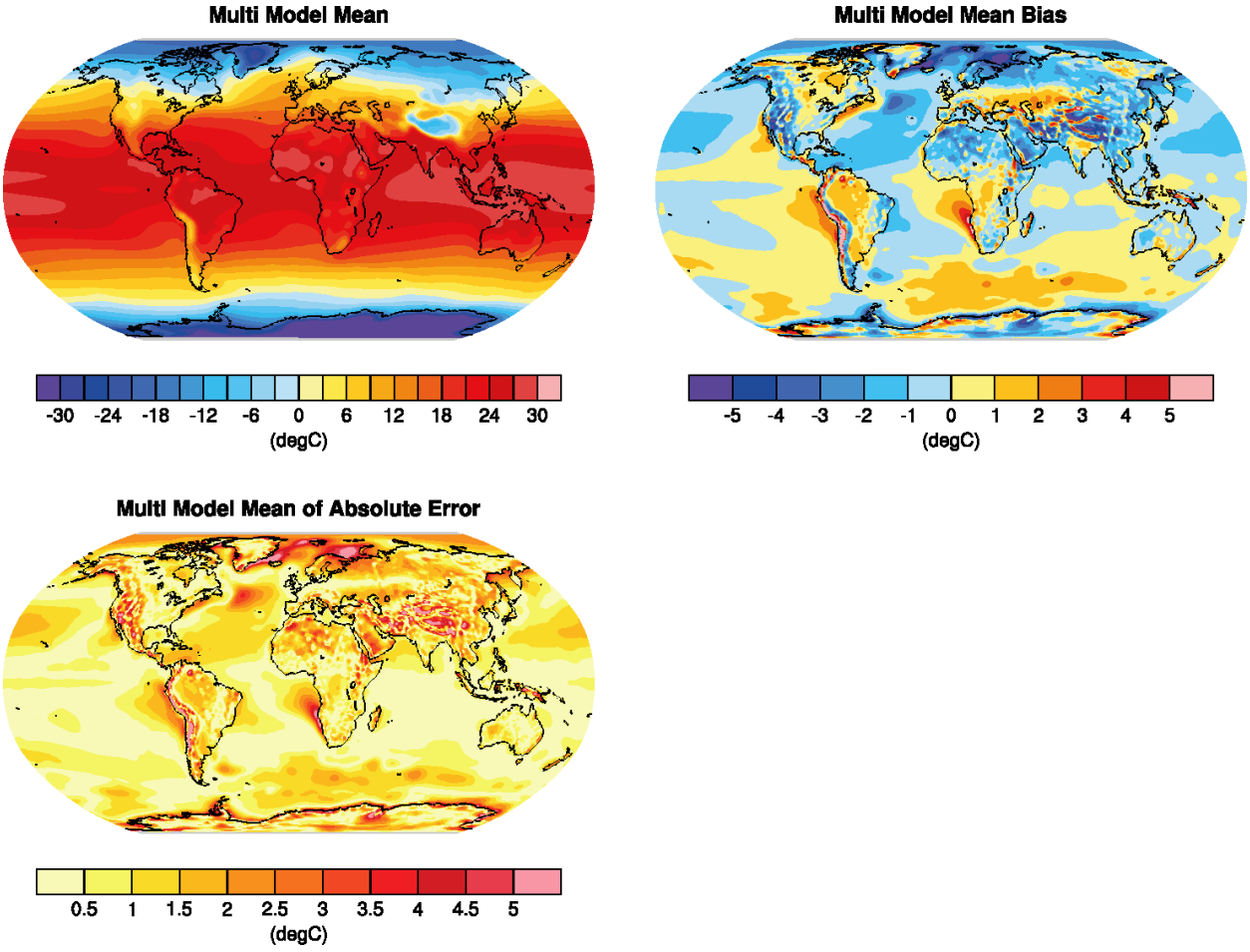


Fig. 195: Figure 9.2 a,b,c: Annual-mean surface air temperature for the period 1980-2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from ERA-Interim (Dee et al., 2011), c) mean absolute model error with respect to the climatology from ERA-Interim.

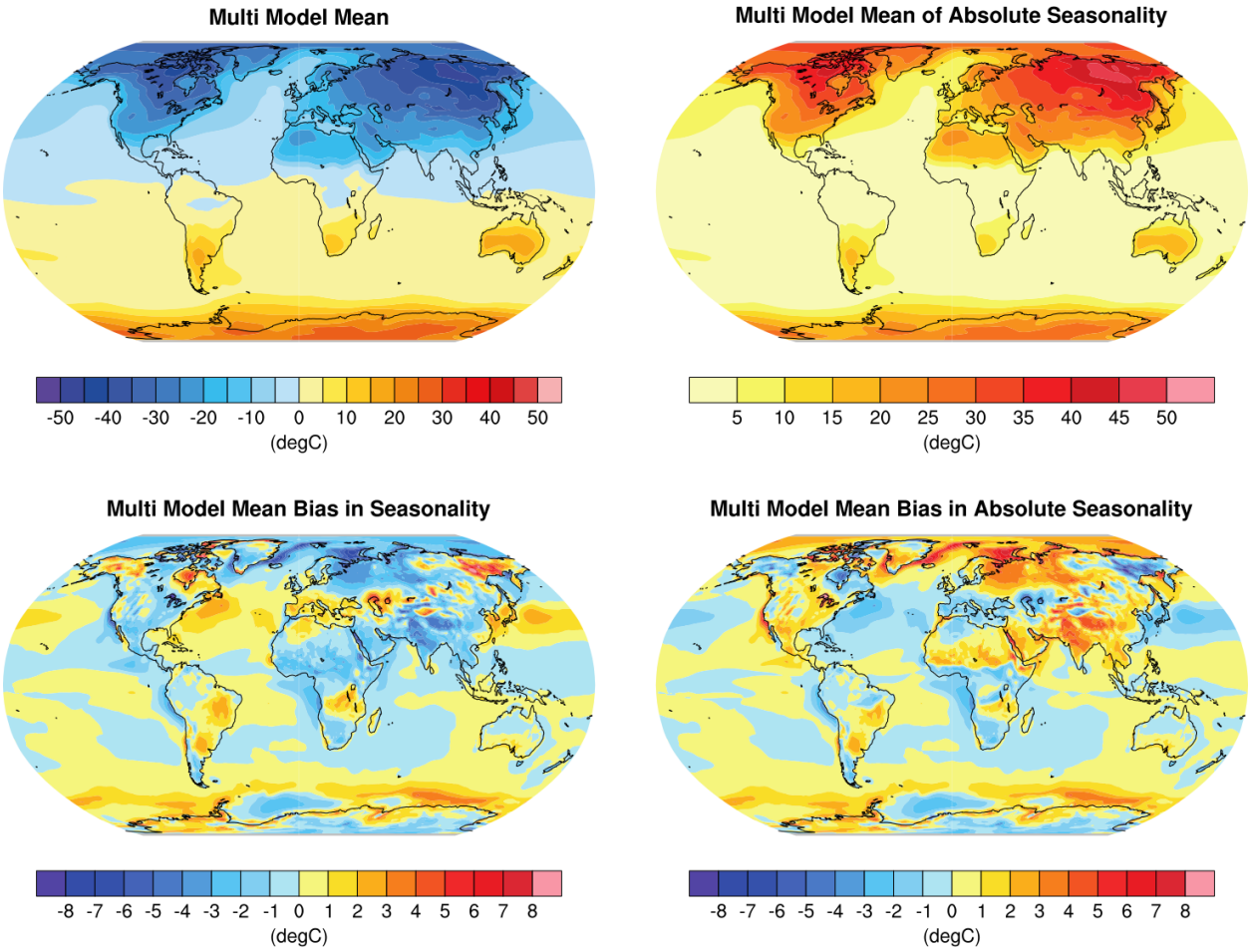


Fig. 196: Figure 9.3: Multi model values for seasonality of near-surface temperature, from top left to bottom right: mean, mean of absolute seasonality, mean bias in seasonality, mean bias in absolute seasonality. Reference dataset: ERA-Interim.

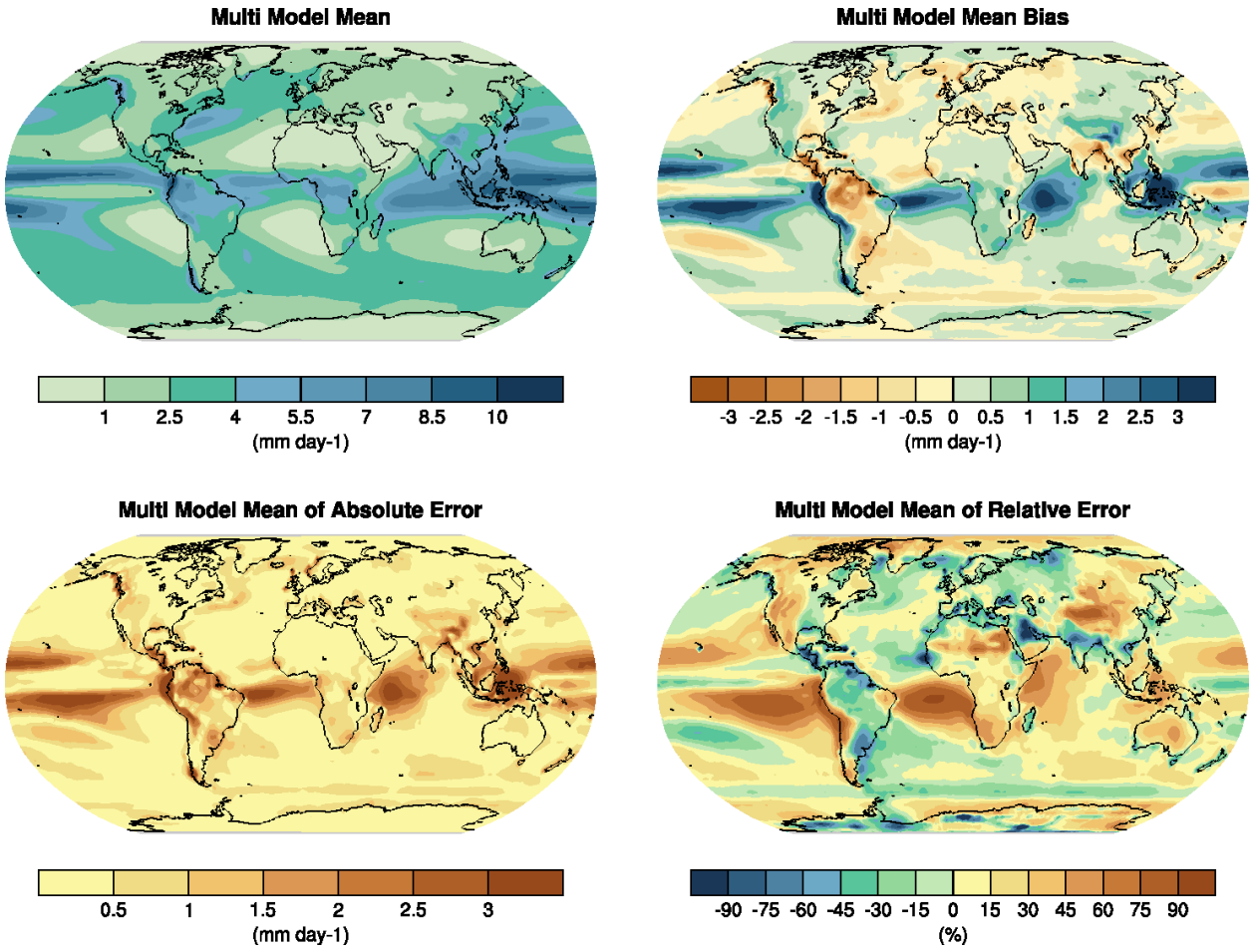


Fig. 197: Figure 9.4: Annual-mean precipitation rate (mm day-1) for the period 1980-2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from the Global Precipitation Climatology Project (Adler et al., 2003), c) multi-model mean absolute error with respect to observations, and d) multi-model mean error relative to the multi-model mean precipitation itself.

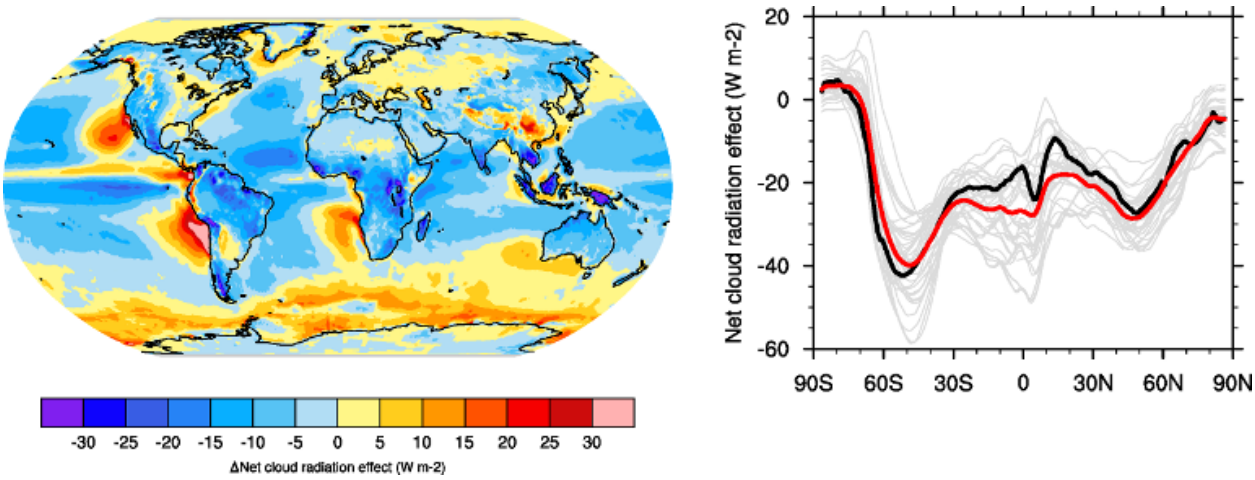


Fig. 198: Figure 9.5: Climatological (1985-2005) annual-mean cloud radiative effects in Wm-2 for the CMIP5 models against CERES EBAF (2001-2011) in Wm-2. Top row shows the shortwave effect; middle row the longwave effect, and bottom row the net effect. Multi-model-mean biases against CERES EBAF 2.6 are 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).

5.5.3 IPCC AR5 Chapter 12 (selected figures)

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.

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.

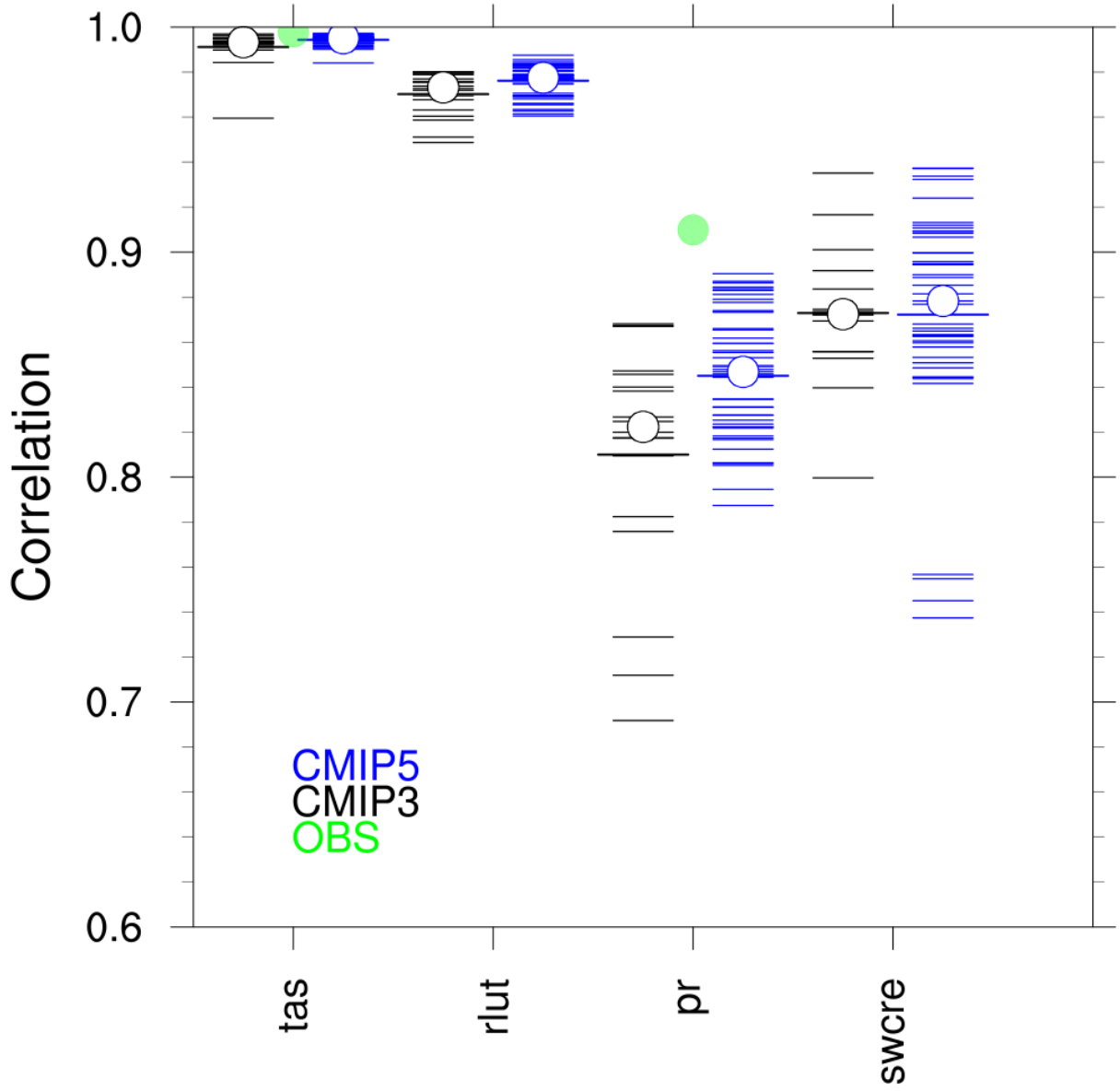


Fig. 199: Figure 9.6: Centered pattern correlations between models and observations for the annual mean climatology over the period 1980–1999. Results are shown for individual CMIP3 (black) and CMIP5 (blue) models as thin dashes, along with the corresponding ensemble average (thick dash) and median (open 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).

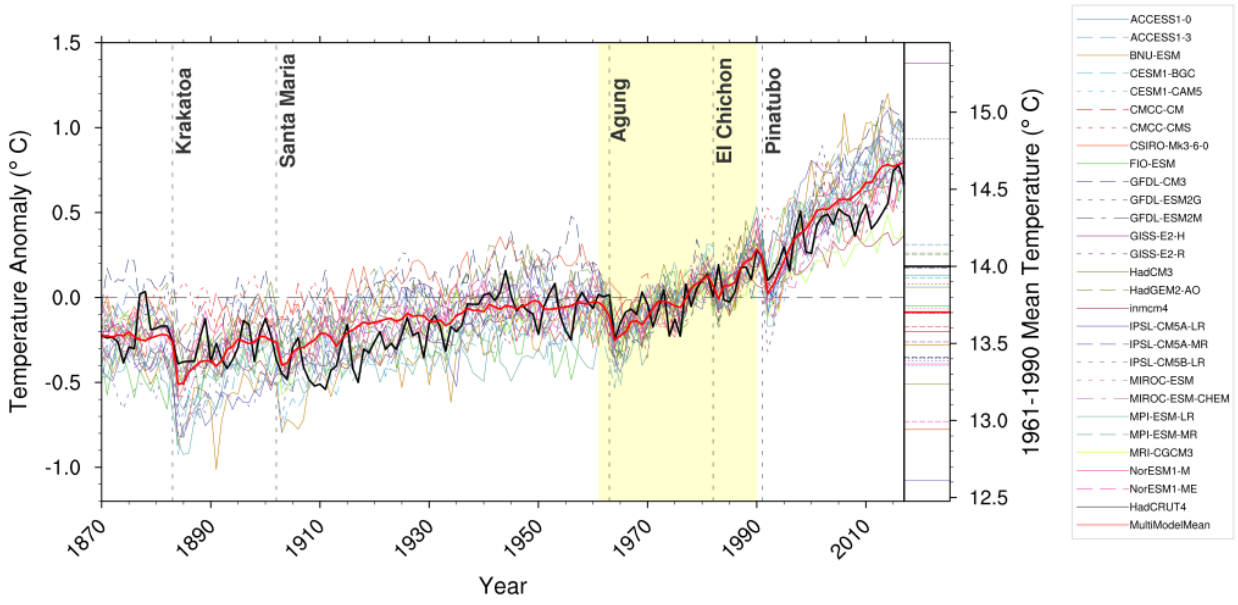


Fig. 200: Figure 9.8: Observed and simulated time series of the anomalies in annual and global mean surface temperature. All anomalies are differences from the 1961-1990 time-mean of each individual time series. The reference period 1961-1990 is indicated by yellow shading; vertical dashed grey lines 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.

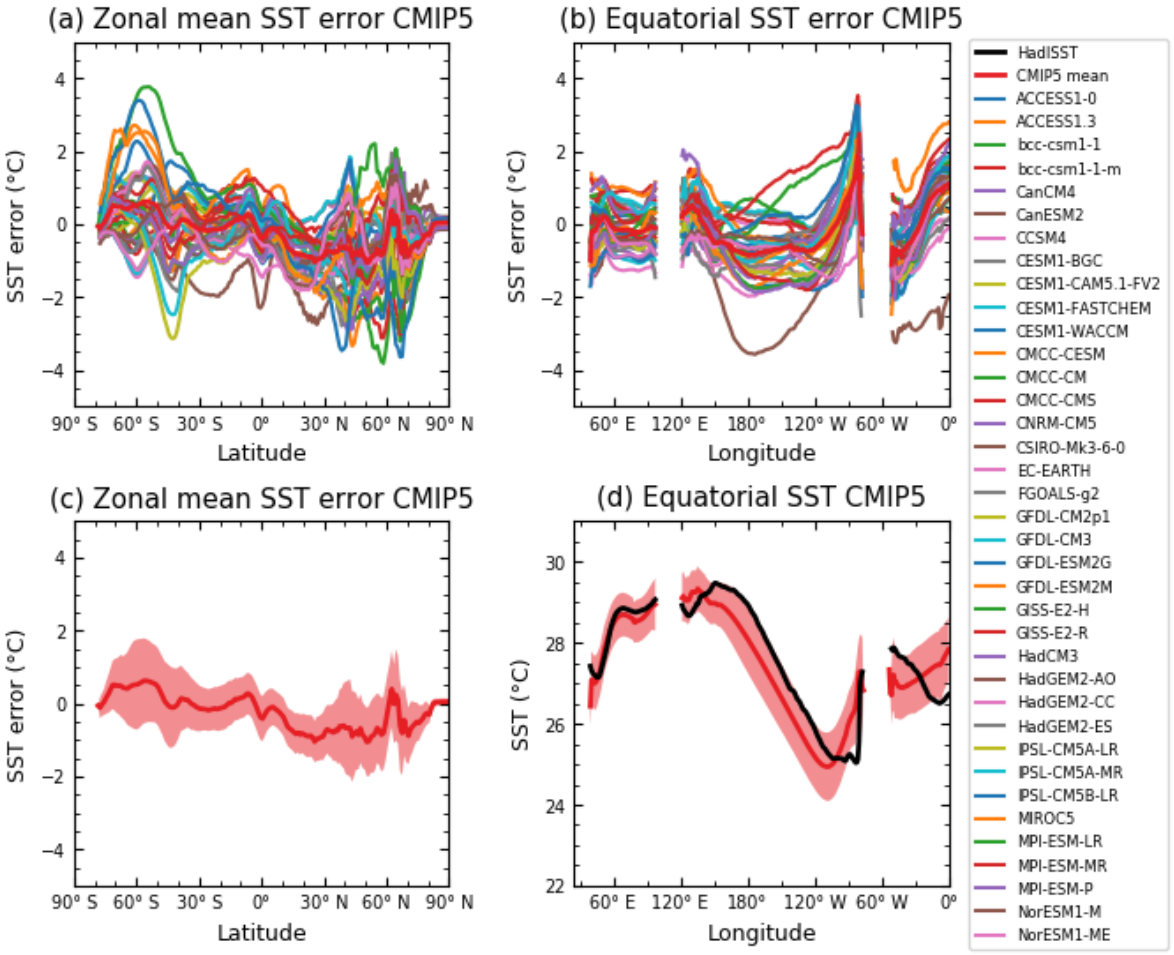


Fig. 201: Figure 9.14: (a) Zonally averaged sea surface temperature (SST) error in CMIP5 models. (b) Equatorial SST error in CMIP5 models. (c) Zonally averaged multi-model mean SST error for CMIP5 together with inter-model standard deviation (shading). (d) Equatorial multi-model mean SST in CMIP5 together 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).

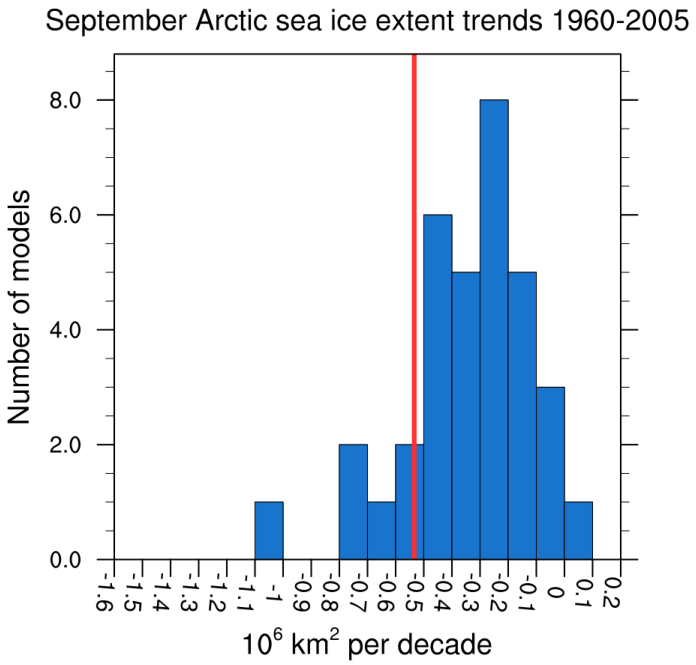


Fig. 202: Figure 9.24c: Sea ice extent trend distribution for the Arctic in September.

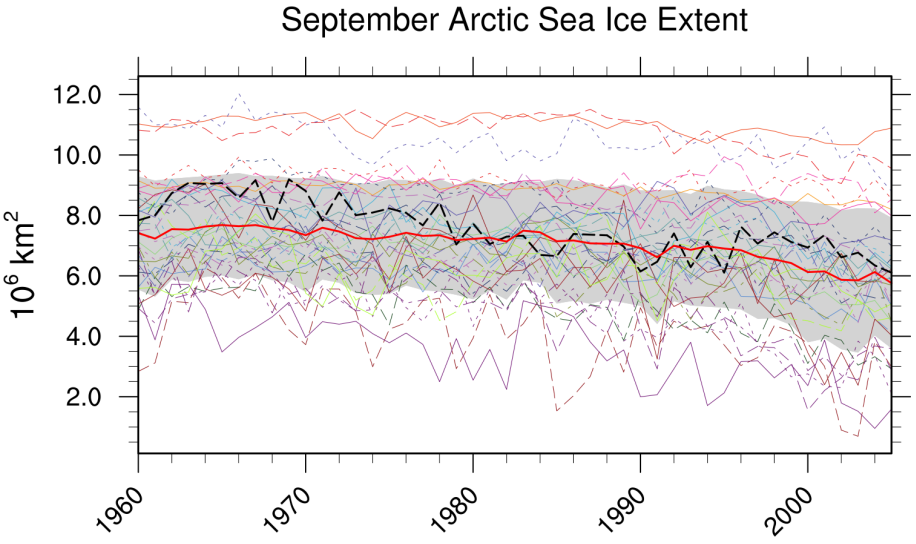


Fig. 203: Figure 9.24a: Time series of total sea ice area and extent (accumulated) for the Arctic in September including multi-model mean and standard deviation.

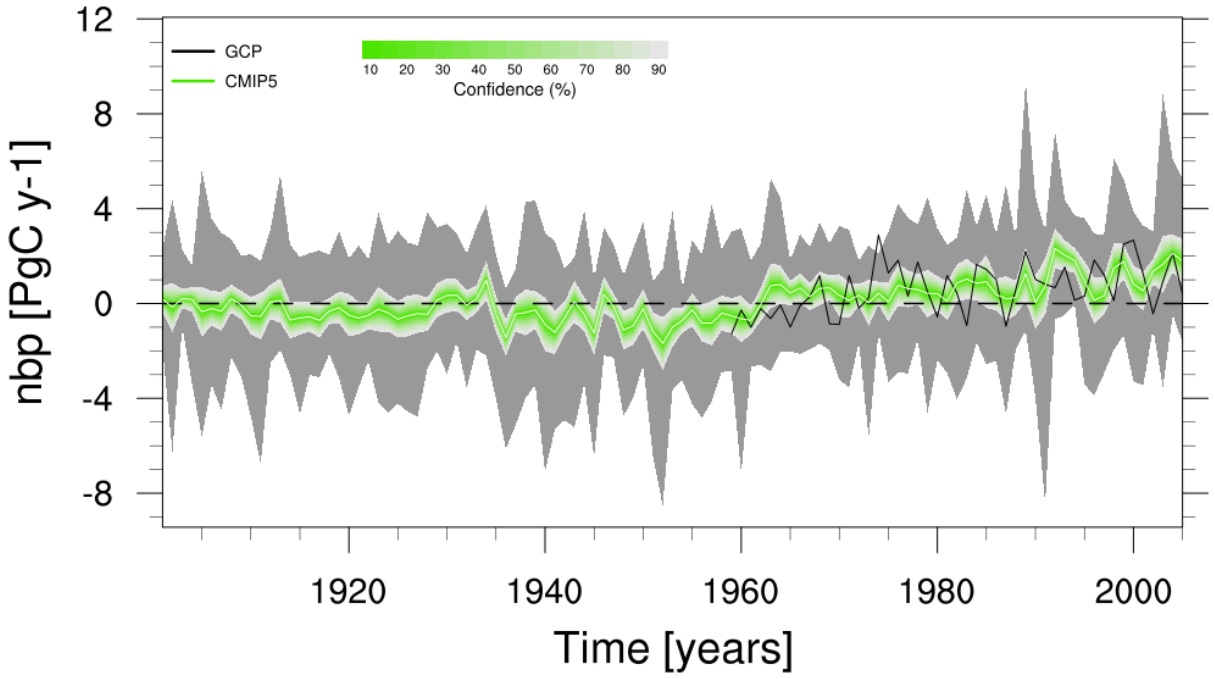


Fig. 204: Figure 9.26 (bottom): Ensemble-mean global land carbon uptake in the CMIP5 ESMs for the historical period 1900–2005. For comparison, the observation-based estimates provided by the Global Carbon Project (GCP) are also shown (black line). The confidence limits on the ensemble mean are derived 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.

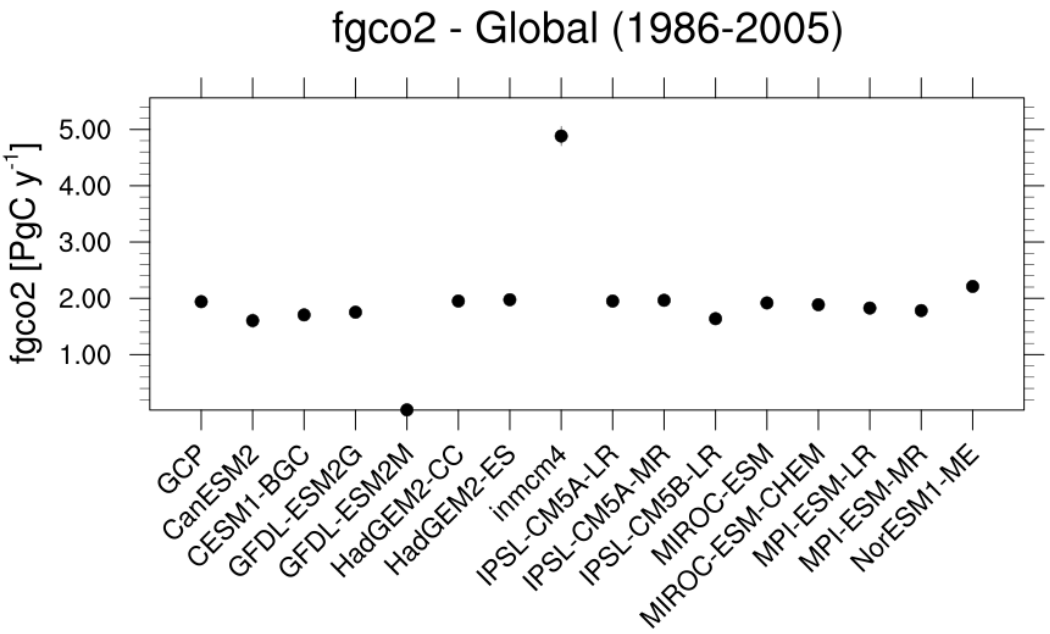


Fig. 205: Figure 9.27 (top): Simulation of global mean atmosphere–ocean CO2 fluxes ("fgCO2") by ESMs for the period 1986–2005. For comparison, the observation-based estimates provided by Global Carbon Project (GCP) are also shown. The error bars for the ESMs and observations represent interannual variability in the fluxes, calculated as the standard deviation of the annual means over the period 1986–2005.

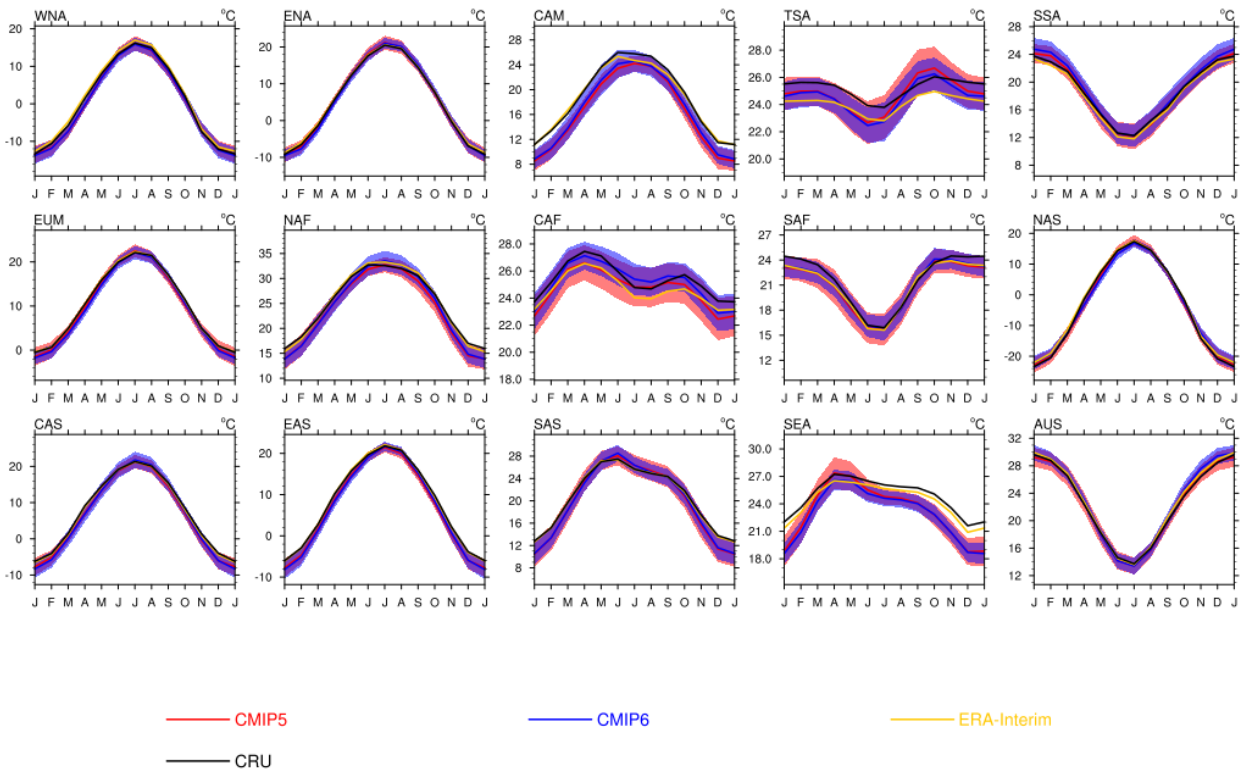


Fig. 206: Figure 9.38tas: Mean seasonal cycle for surface temperature (tas) as multi model mean of 38 CMIP5 and 22 CMIP6 models as well as CRU and ERA-Interim reanalysis data averaged for 1980-2005 over land in different regions: Western North America (WNA), Eastern North America (ENA), Central America (CAM), Tropical South America (TSA), Southern South America (SSA), Europe and Mediterranean (EUM), North Africa (NAF), Central Africa (CAF), South Africa (SAF), North Asia (NAS), Central Asia (CAS), East Asia (EAS), South Asia (SAS), Southeast Asia (SEA), and Australia (AUS). Similar to Fig. 9.38a from Flato et al. (2013), CMIP6 instead of CMIP3 and set of CMIP5 models used different.

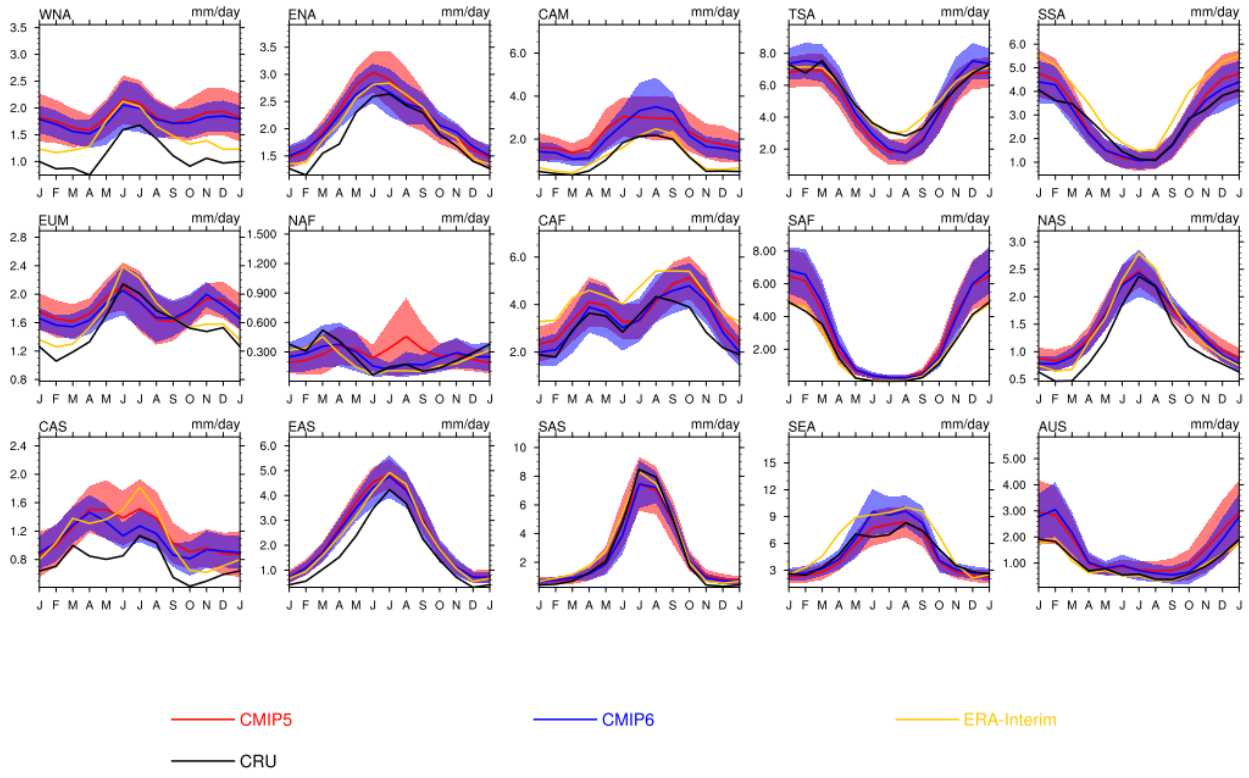


Fig. 207: Figure 9.38pr: Mean seasonal cycle for precipitation (pr) as multi model mean of 38 CMIP5 and 22 CMIP6 models as well as CRU and ERA-Interim reanalysis data averaged for 1980-1999 over land in different regions: Western North America (WNA), Eastern North America (ENA), Central America (CAM), Tropical South America (TSA), Southern South America (SSA), Europe and Mediterranean (EUM), North Africa (NAF), Central Africa (CAF), South Africa (SAF), North Asia (NAS), Central Asia (CAS), East Asia (EAS), South Asia (SAS), Southeast Asia (SEA), and Australia (AUS). Similar to Fig. 9.38b from Flato et al. (2013), CMIP6 instead of CMIP3 and set of CMIP5 models used different.

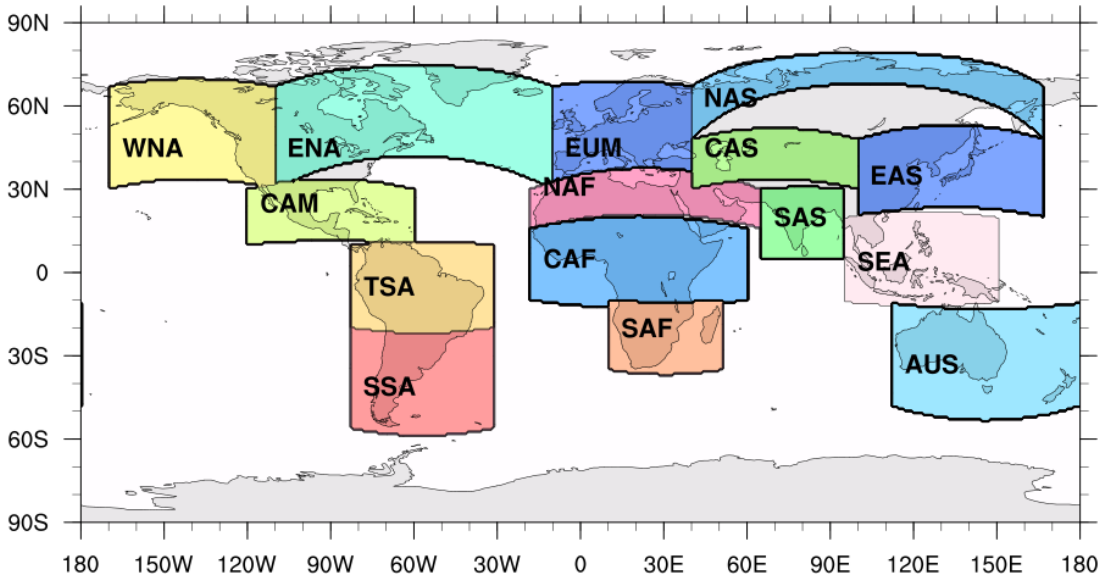


Fig. 208: Figure 9.38reg: Positions of the regions used in Figure 9.38.

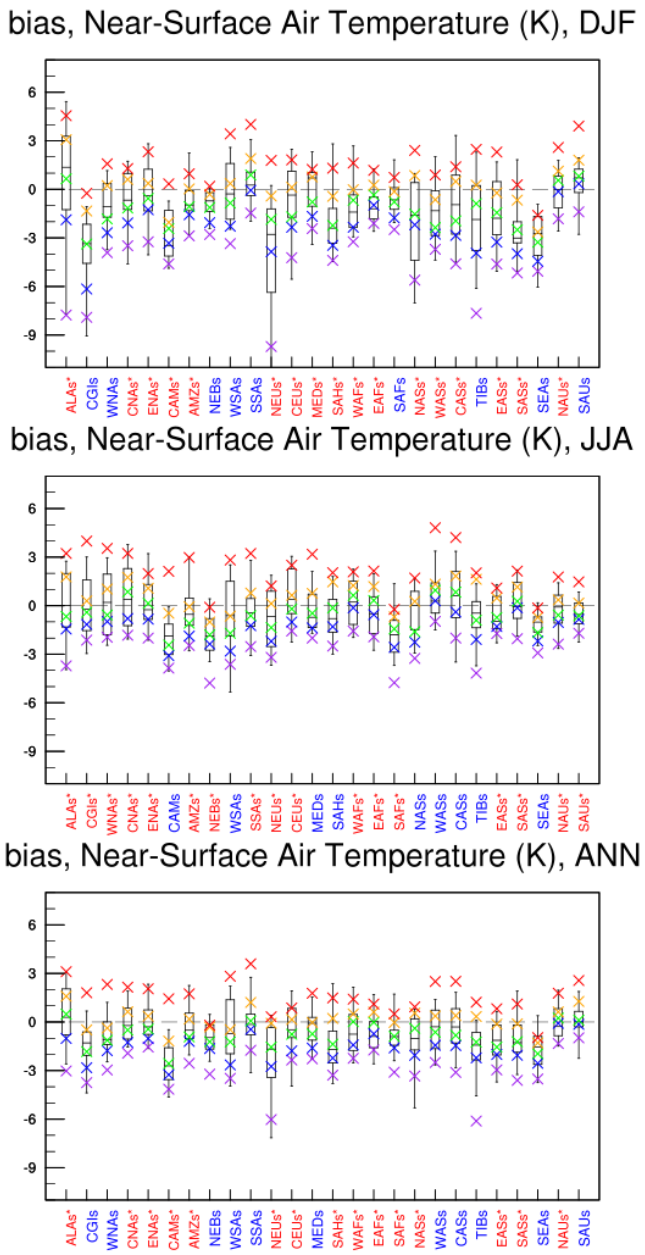


Fig. 209: Figure 9.39tas: Box and whisker plots showing the 5th, 25th, 50th, 75th and 95th percentiles of the seasonal- and annual mean biases for surface temperature (tas) for 1980-2005 between 38 CMIP5 models (box and whiskers) or 22 CMIP6 models (crosses) and CRU data. The regions are: Alaska/NW Canada (ALAs), Eastern Canada/Greenland/Iceland (CGIs), Western North America(WNAs), Central North America (CNAs), Eastern North America (ENAs), Central America/Mexico (CAMs), Amazon (AMZs), NE Brazil (NEBs), West Coast South America (WSAs), South-Eastern South America (SSAs), Northern Europe (NEUs), Central Europe (CEUs), Southern Europe/the Mediterranean (MEDs), Sahara (SAHs), Western Africa (WAFs), Eastern Africa (EAFs), Southern Africa (SAFs), Northern Asia (NASs), Western Asia (WASs), Central Asia (CASs), Tibetan Plateau (TIBs), Eastern Asia (EASs), Southern Asia (SASs), Southeast Asia (SEAs), Northern Australia (NASs) and Southern Australia/New Zealand (SAUs). The positions of these regions are defined following (Seneviratne et al., 2012) and differ from the ones in Fig. 9.38. Similar to Fig. 9.39 a,c,e from Flato et al. (2013), CMIP6 instead of CMIP3 and set of CMIP5 models used different.

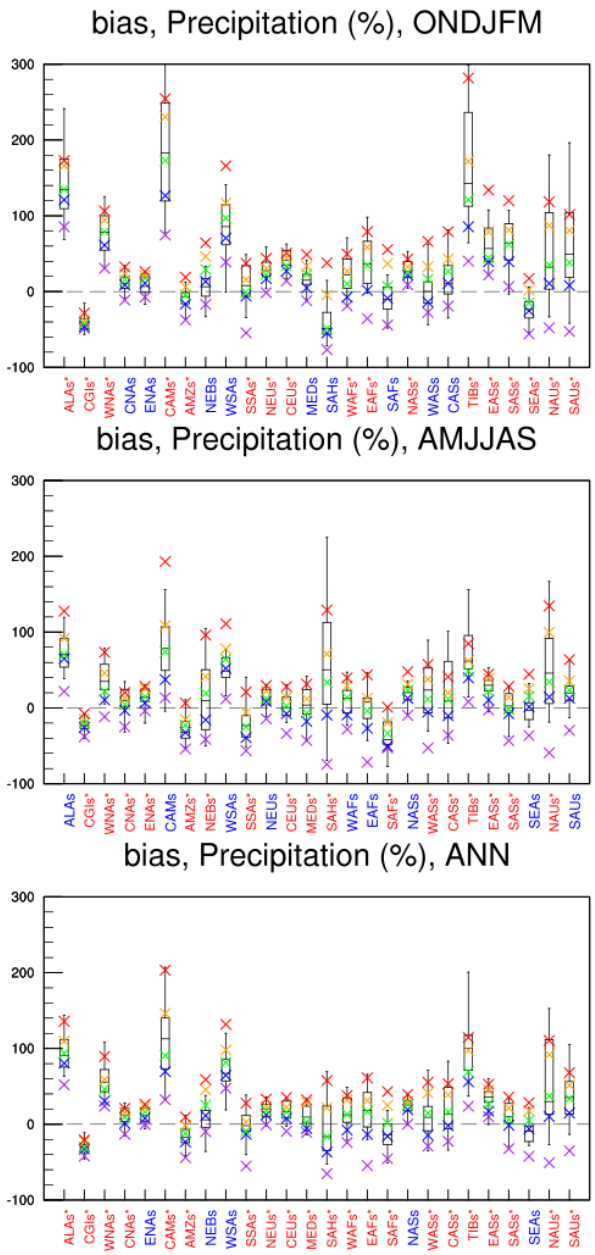


Fig. 210: Figure 9.39pr: Box and whisker plots showing the 5th, 25th, 50th, 75th and 95th percentiles of the seasonal- and annual mean biases for precipitation (pr) for 1980-2005 between 38 CMIP5 models (box and whiskers) or 22 CMIP6 models (crosses) and CRU data. The regions are: Alaska/NW Canada (ALAs), Eastern Canada/Greenland/Iceland (CGIs), Western North America(WNAs), Central North America (CNAs), Eastern North America (ENAs), Central America/Mexico (CAMs), Amazon (AMZs), NE Brazil (NEBs), West Coast South America (WSAs), South-Eastern South America (SSAs), Northern Europe (NEUs), Central Europe (CEUs), Southern Europe/the Mediterranean (MEDs), Sahara (SAHs), Western Africa (WAFs), Eastern Africa (EAFs), Southern Africa (SAFs), Northern Asia (NASs), Western Asia (WASs), Central Asia (CASs), Tibetan Plateau (TIBs), Eastern Asia (EASs), Southern Asia (SASs), Southeast Asia (SEAs), Northern Australia (NASs) and Southern Australia/New Zealand (SAUs). The positions of these regions are defined following (Seneviratne et al., 2012) and differ from the ones in Fig. 9.38. Similar to Fig. 9.39 b,d,f from Flato et al. (2013), CMIP6 instead of CMIP3 and set of CMIP5 models used different.

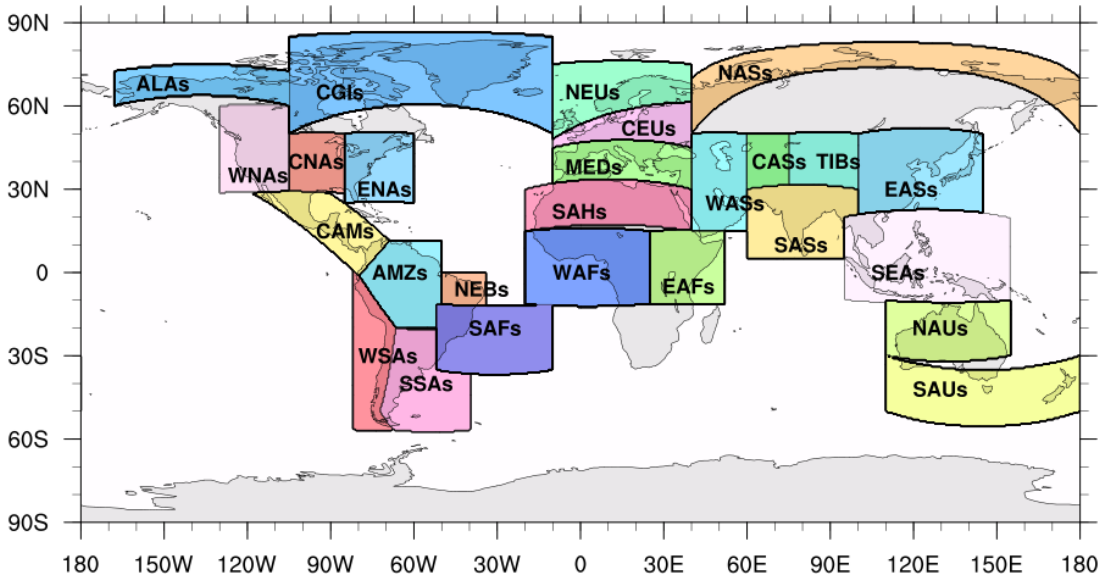


Fig. 211: Figure 9.39reg: Positions of the regions used in Figure 9.39.

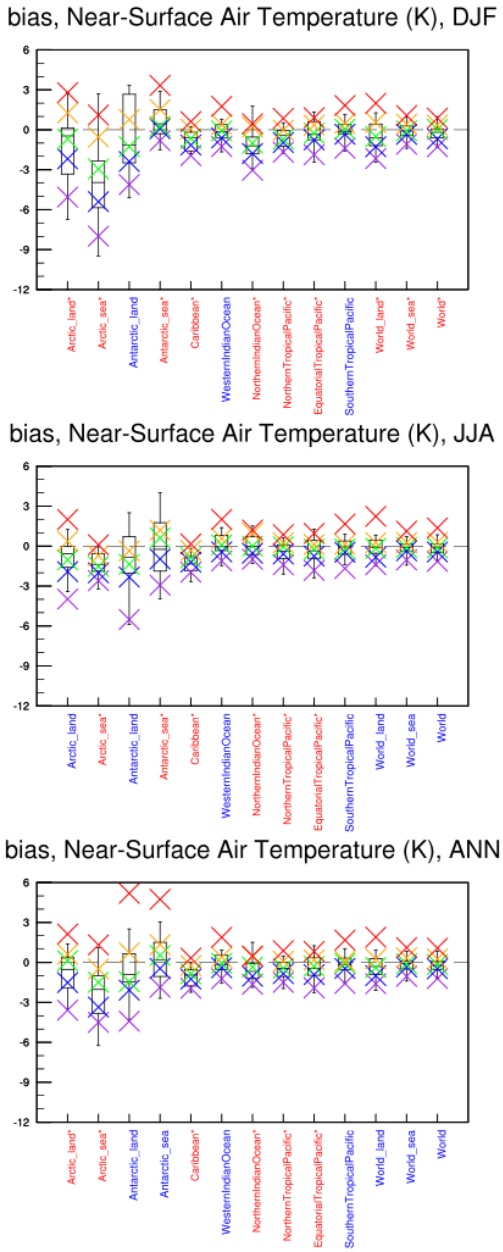


Fig. 212: Figure 9.40tas: Box and whisker plots showing the 5th, 25th, 50th, 75th and 95th percentiles of the seasonal- and annual mean biases for surface temperature (tas) for oceanic and polar regions between 38 CMIP5 (box and whiskers) or 22 CMIP6 (crosses) models and ERA-Interim data for 1980–2005.

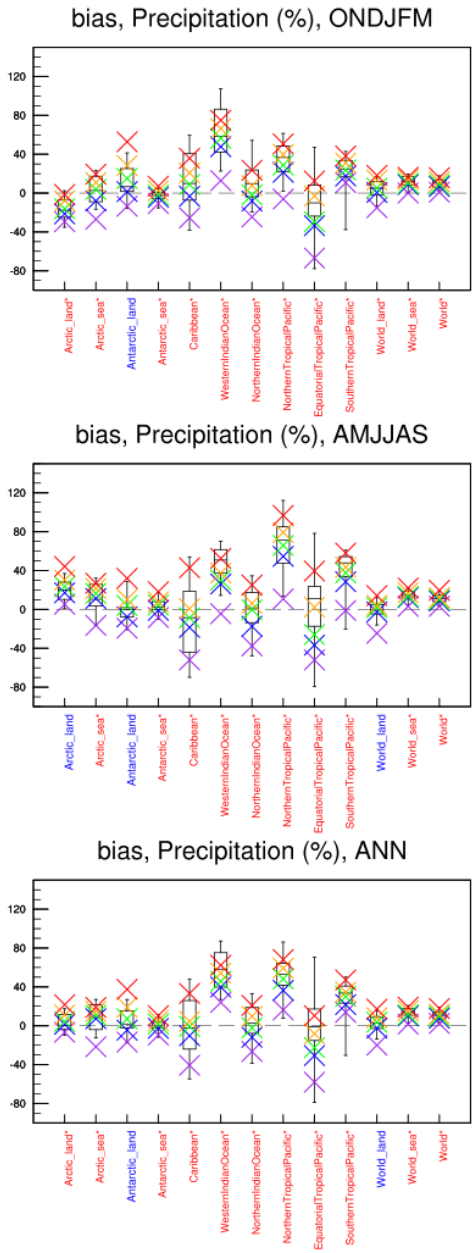


Fig. 213: Figure 9.40pr: Box and whisker plots showing the 5th, 25th, 50th, 75th and 95th percentiles of the seasonal- and annual mean biases for precipitation (pr) for oceanic and polar regions between 38 CMIP5 (box and whiskers) or 22 CMIP6 (crosses) models and Global Precipitation Climatology Project - Satellite-Gauge (GPCP-SG) data for 1980–2005.

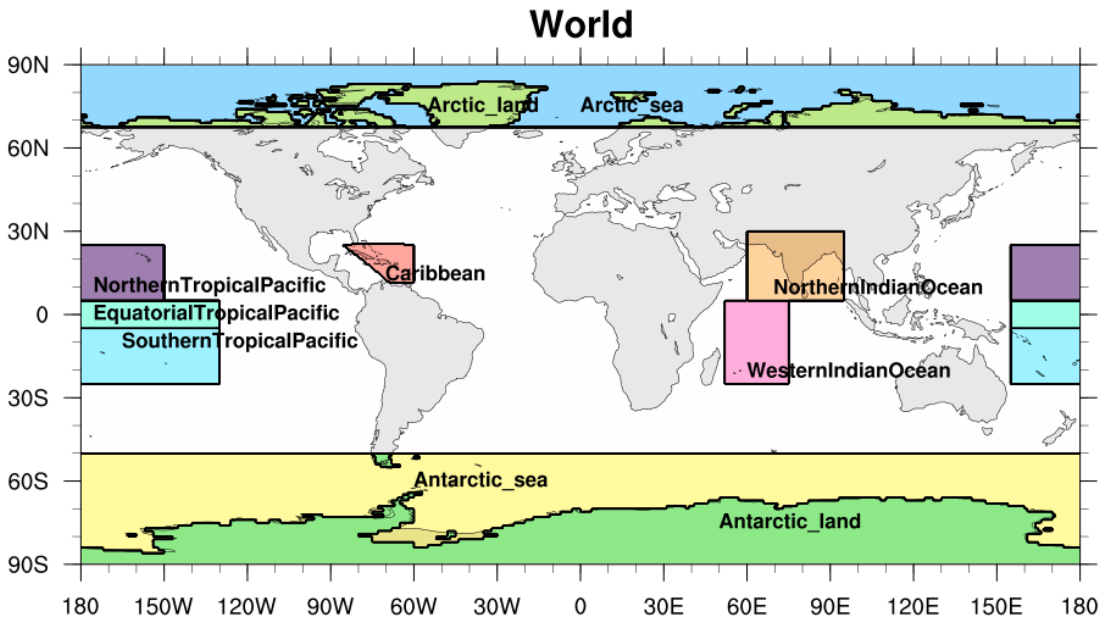


Fig. 214: Figure 9.40reg: Positions of the regions used in Figure 9.40.

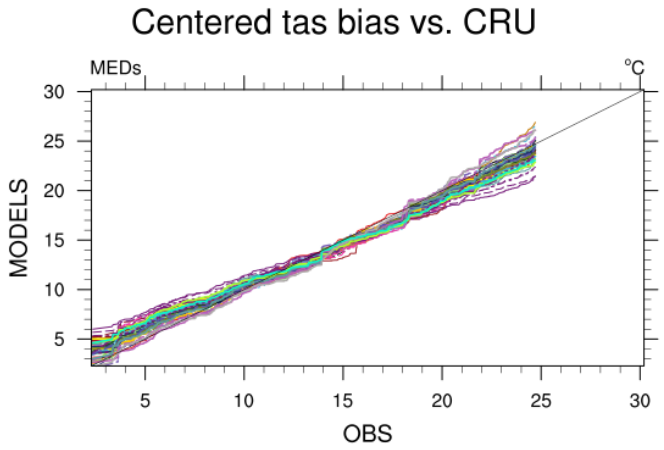


Fig. 215: Figure 9.41b: Ranked modelled versus ERA-Interim mean temperature for 38 CMIP5 models in the Mediterranean region for 1961–2000.

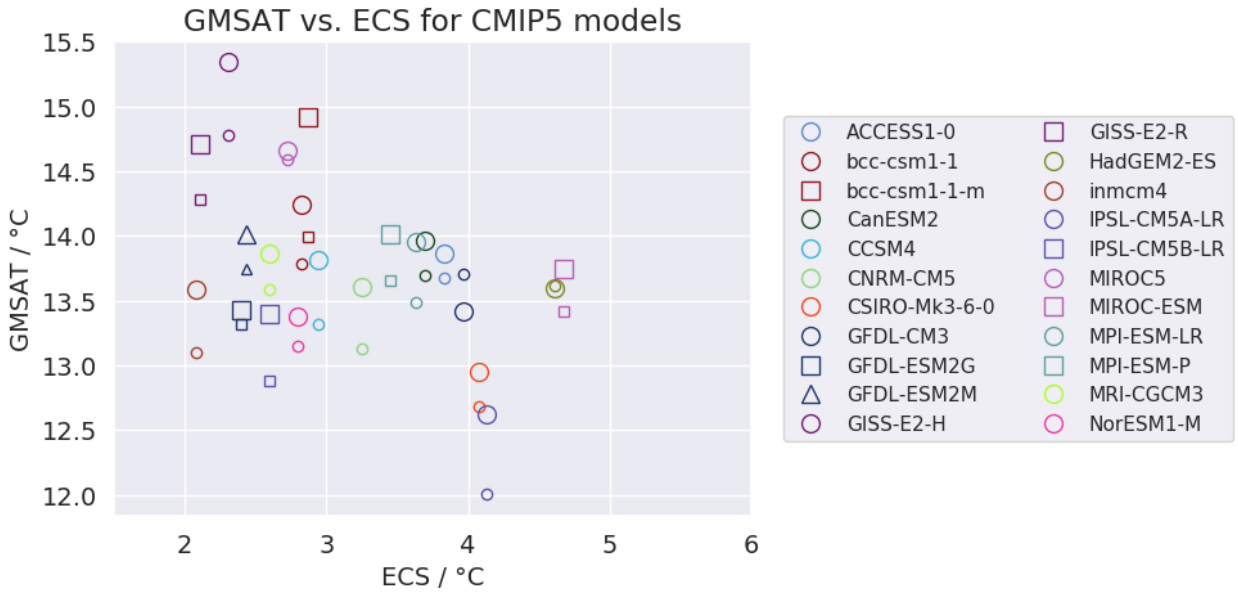


Fig. 216: Figure 9.42a: Equilibrium climate sensitivity (ECS) against the global mean surface air temperature of CMIP5 models, both for the period 1961-1990 (larger symbols) and for the pre-industrial control runs (smaller symbols).

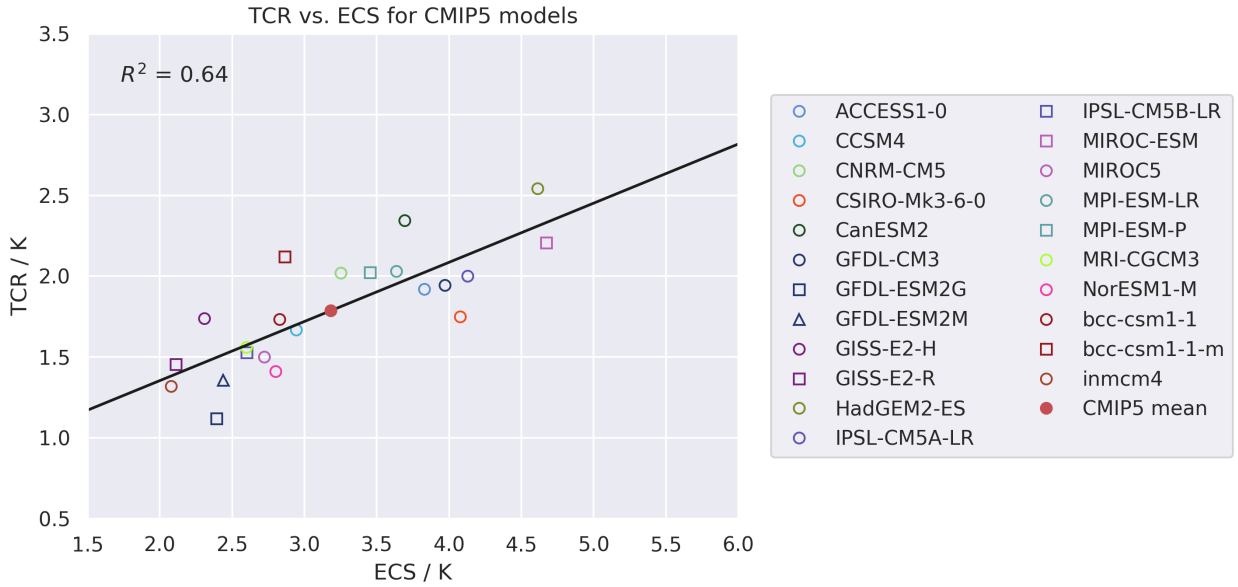


Fig. 217: Figure 9.42b: Transient climate response (TCR) against equilibrium climate sensitivity (ECS) for CMIP5 models.

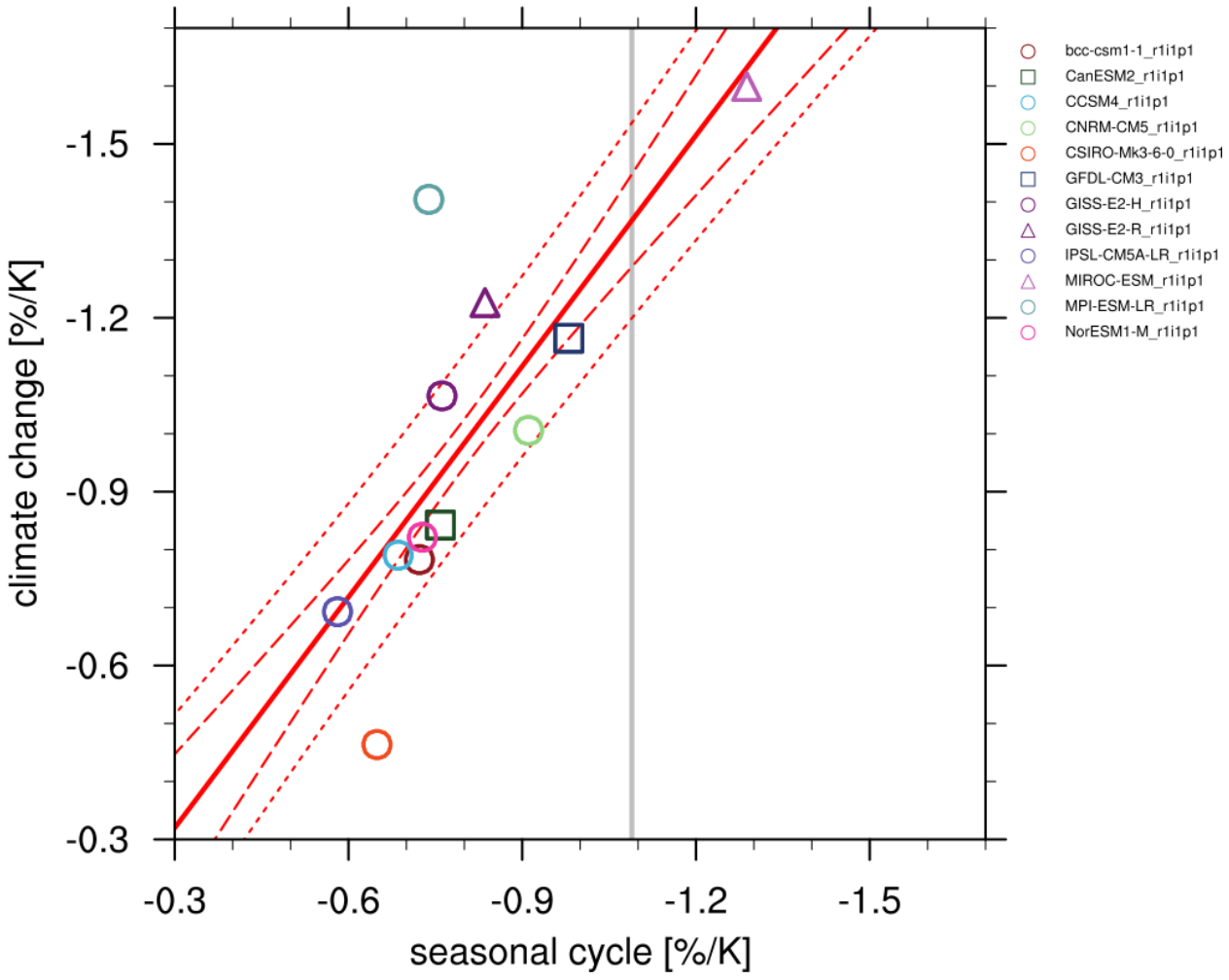


Fig. 218: Figure 9.45a: Scatterplot of springtime snow-albedo effect values in climate change vs. springtime $\Delta\alpha_s/\Delta T_s$ values in the seasonal cycle in transient climate change experiments (CMIP5 historical experiments: 1901-2000, RCP4.5 experiments: 2101-2200).

- ipcc_ar5/ch12_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.
- 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.

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

1. 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.

2. 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

3. 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)

4. 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

5. 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

6. 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

7. 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]

8. 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

9. 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

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)

Observations and reformat scripts

- HadISST (sic - esmvaltool/cmorizers/data/formatters/datasets/hadisst.ncl)

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.

Example plots

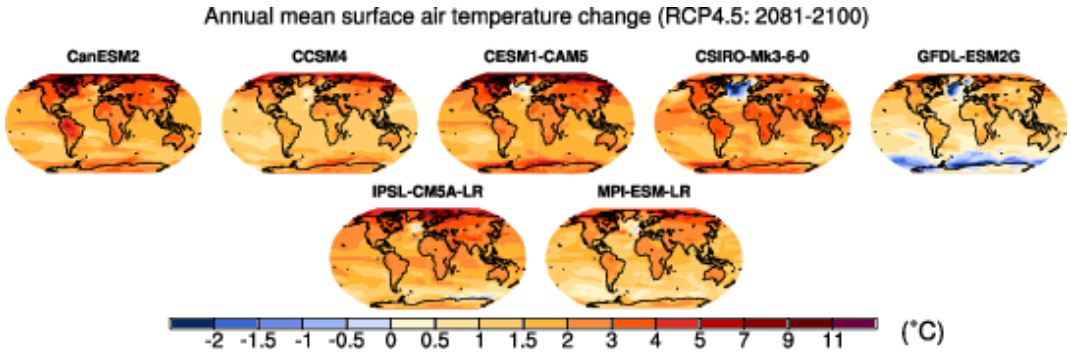


Fig. 219: Surface air temperature change in 2081–2100 displayed as anomalies with respect to 1986–2005 for RCP4.5 from individual CMIP5 models.

5.6 Land

5.6.1 Landcover - Albedo

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.

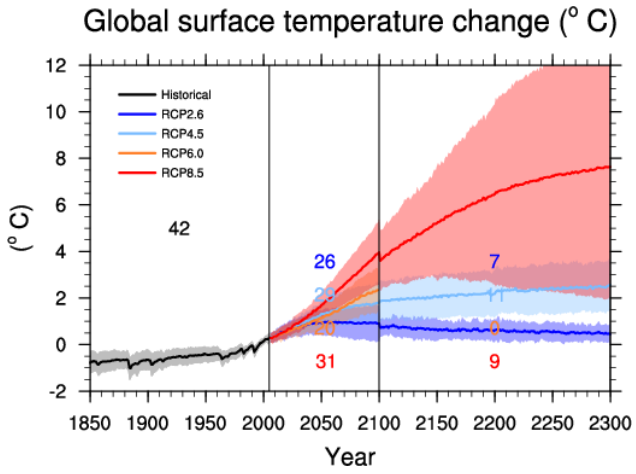


Fig. 220: Time series of global annual mean surface air temperature anomalie (relative to 1986–2005) from CMIP5 concentration-driven experiments.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_albedolandcover.yml

Diagnostics are stored in diag_scripts/landcover/

- albedolandcover.py

User settings

Several parameters can be set in the recipe

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)

Observations and reformat scripts

A reformatting script for observational data is available here:

- esmvaltool/cmorizers/data/formatters/datasets/duveiller2018.py

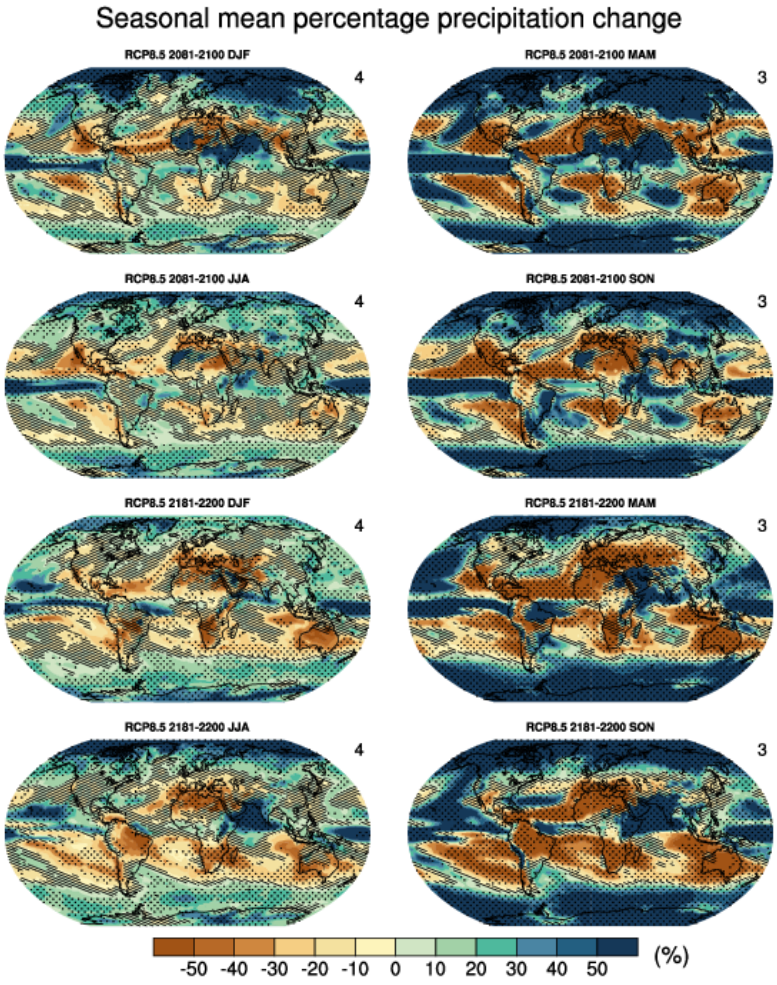


Fig. 221: Multi-model CMIP5 average percentage change in seasonal mean precipitation relative to the reference period 1986–2005 averaged over the periods 2081–2100 and 2181–2200 under the RCP8.5 forcing scenario. Hatching indicates regions where the multi-model mean change is less than one standard deviation 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

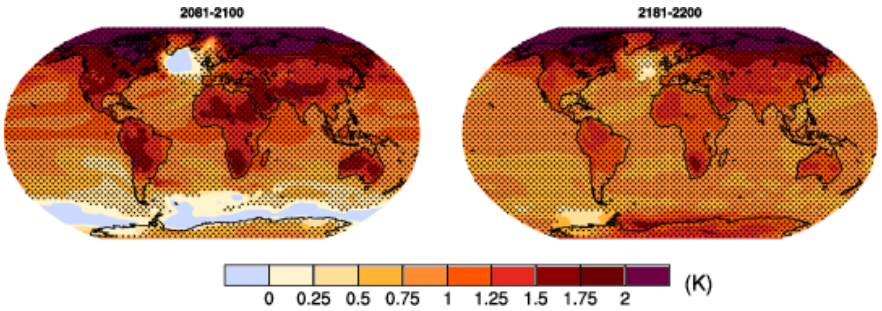


Fig. 222: Temperature change patterns scaled to 1°C of global mean surface temperature change.

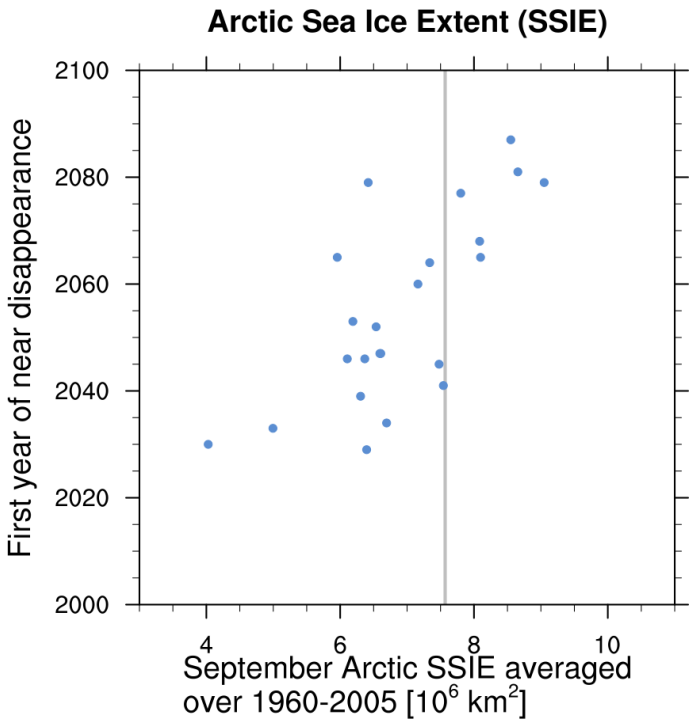


Fig. 223: Scatter plot of mean historical September Arctic sea ice extent vs 1st year of disappearance (RCP8.5) (similar to IPCC AR5 Chapter 12, Fig. 12.31a).

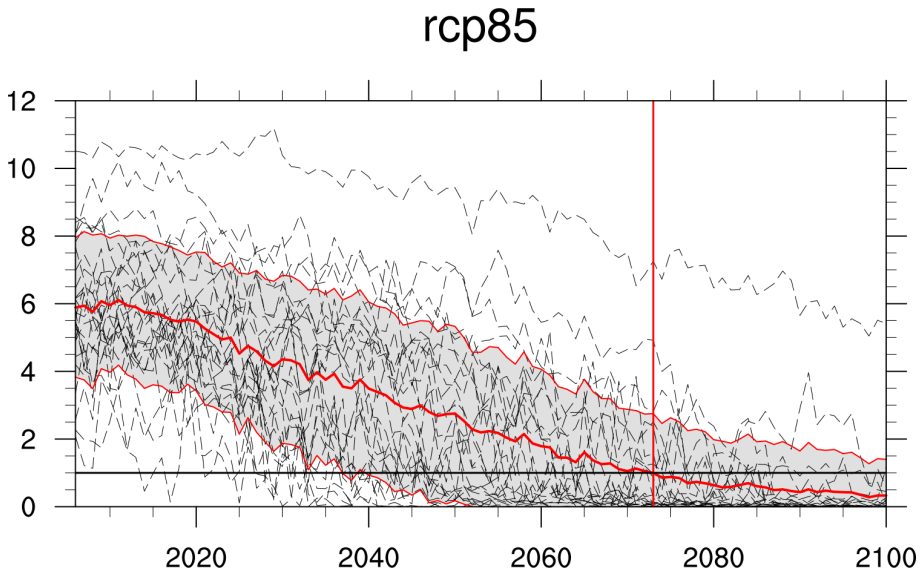


Fig. 224: Time series of September Arctic sea ice extent for individual CMIP5 models, multi-model mean and multi-model standard deviation, year of disappearance (similar to IPCC AR5 Chapter 12, Fig. 12.31e).

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.

Example plots

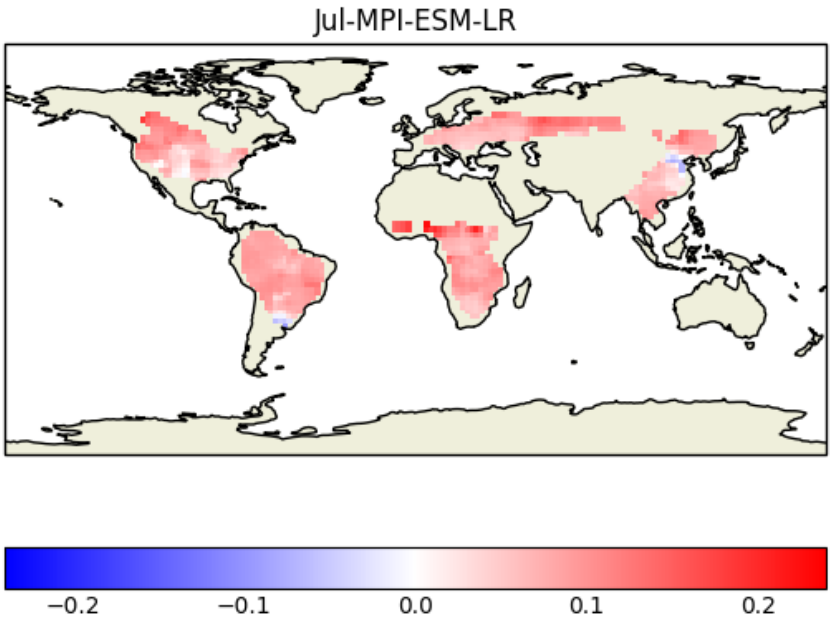


Fig. 225: Example of albedo change from tree to crop and grass for the CMIP5 model MPI-ESM-LR derived for the month of July and averaged over the years 2000 to 2004.

5.6.2 Turnover time of carbon over land ecosystems

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

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.

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

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 configuration) 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 between 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.

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)

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.

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.

Example plots

5.6.3 Hydrological models - data pre-processing

Overview

We provide a collection of scripts that pre-processes environmental data for use in several hydrological models:

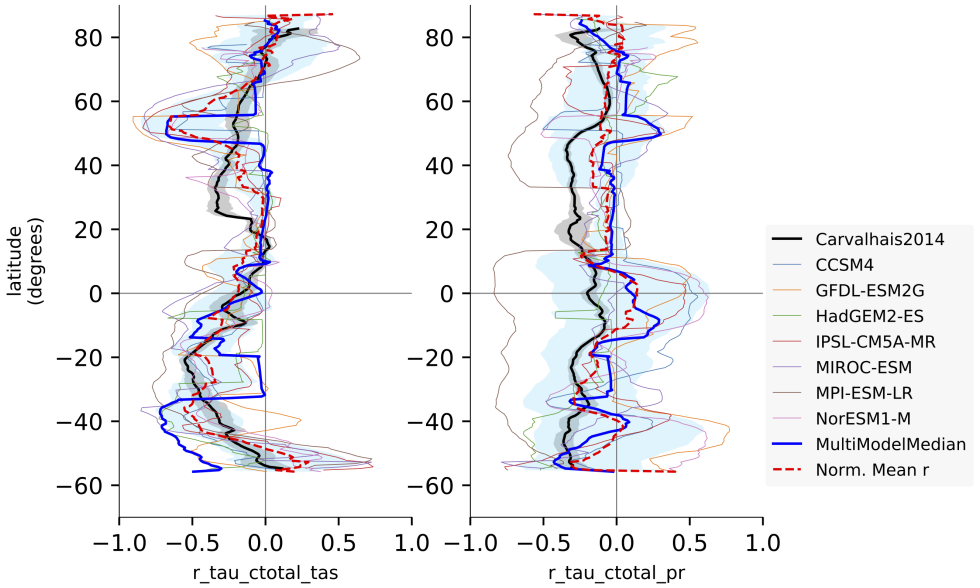


Fig. 226: Comparison of latitudinal (zonal) variations of pearson correlation between turnover time and climate: turnover time and precipitation, controlled for temperature (left) and vice-versa (right). Reproduces figures 2c and 2d in Carvalhais et al. (2014).

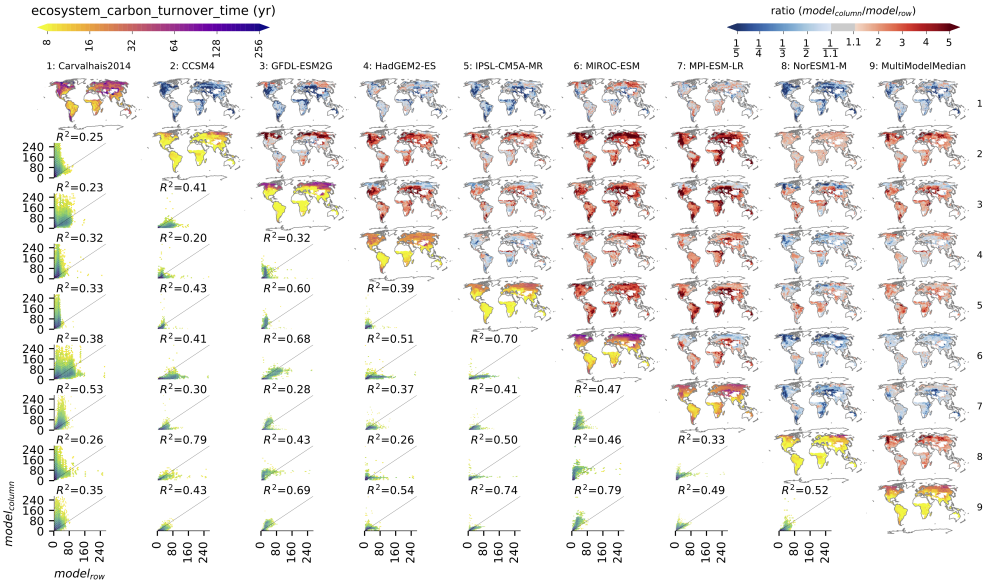


Fig. 227: Comparison of observation-based and modelled ecosystem carbon turnover time. Along the diagnonal, tau_ctotal are plotted, above the bias, and below density plots. The inset text in density plots indicate the correlation.

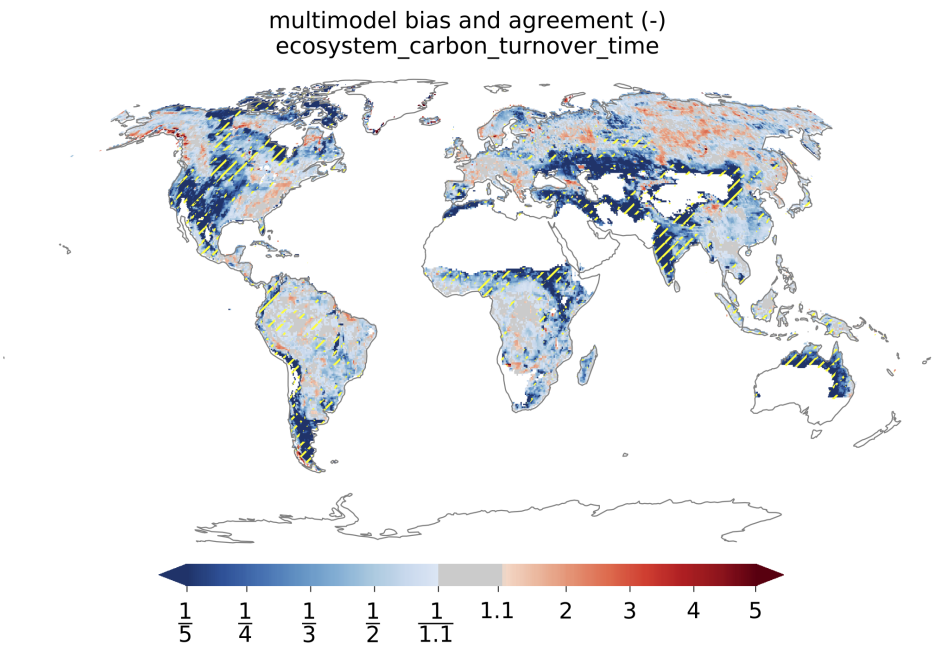


Fig. 228: Global distributions of multimodel bias and model agreement. Multimodel bias is calculated as the ratio of multimodel median turnover time and that from observation. Stippling indicates the regions where only less than one quarter of the models fall within the range of observational uncertainties (*5^{th}* and *95^{th}* percentiles). Reproduces figure 3 in Carvalhais et al. (2014).

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.

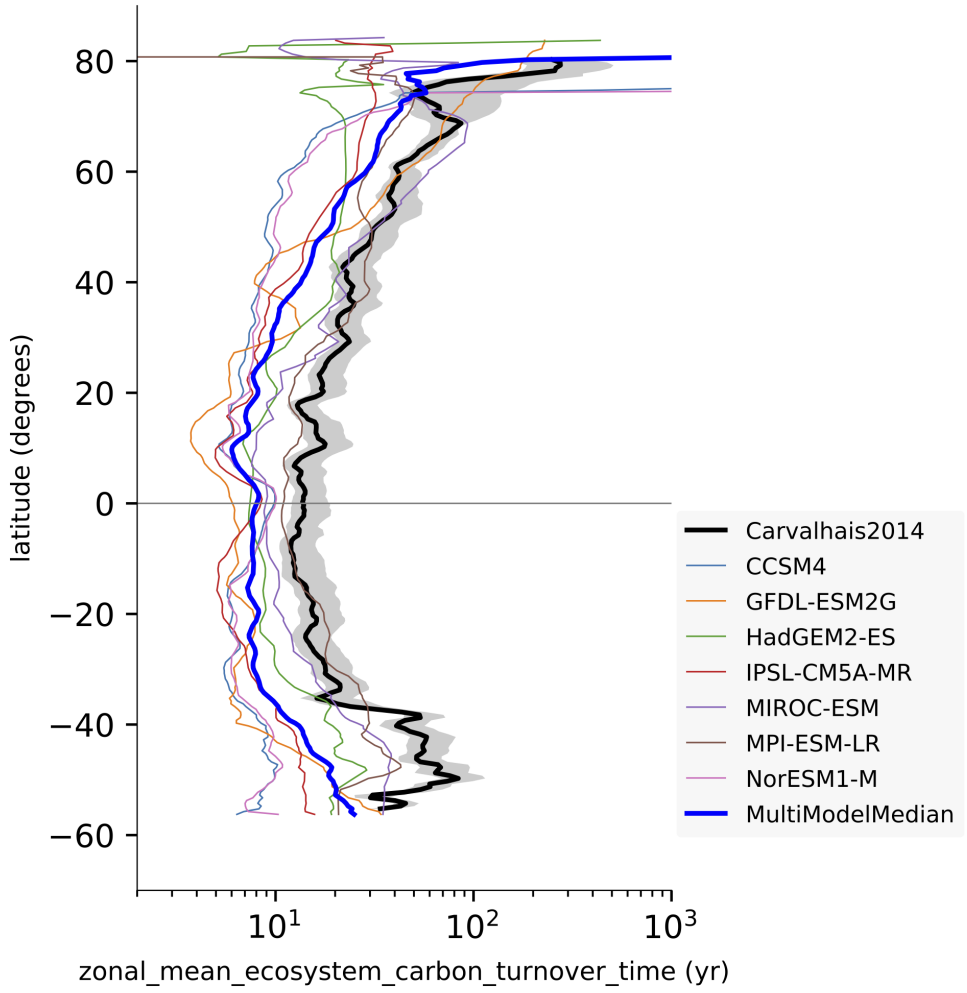


Fig. 229: Comparison of latitudinal (zonal) variations of observation-based and modelled ecosystem carbon turnover time. The zonal turnover time is calculated as the ratio of zonal *ctotal* and *gpp*. Reproduces figures 2a and 2b in Carvalhais et al. (2014).

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.

GlobWat

GlobWat is a soil water balance model that has been provided by the Food and Agriculture Organization (FAO) to assess water use in irrigated agriculture (http://www.fao.org/nr/water/aquamaps). The recipe pre-processes ERA-Interim and ERA5 reanalyses data for use in the GlobWat model. GlobWat requires potential evapotranspiration (evspsblpot) as input. The variable evspsblpot is not available in ERA-Interim. Thus, we use debruin function (De Bruin et al. 2016) or the langbein method (Langbein et al. 1949) to obtain evspsblpot using both ERA-Interim and ERA5. The Langbein function needs a variable tas and the debruin function besides that needs the variables psl, rsds, and rsdt as input. In order to calculate monthly/daily pet with Langbein method we assumed that tas is constant over time and the average value is equal to the annual average.

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
- recipe_globwat.yml

Diagnostics are stored in esmvaltool/diag_scripts/hydrology

- pcrglobwb.py
- marrmot.py
- wflow.py
- lisflood.py
- hype.py
- globwat.py

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 configuration option `auxiliary_data_dir`).
   - 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. A wflow example dataset is available at: https://github.com/openstreams/wflow/tree/master/examples/wflow_rhine_sbm The example dem_file can be obtained from https://github.com/openstreams/wflow/blob/master/examples/wflow_rhine_sbm/staticmaps/wflow_dem.map
   - 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

6. recipe_globwat.yml

   Required preprocessor settings:

   - start_year: 2004
   - end_year: 2004
   - target_grid_file: grid of globwat input files. A target file has been generated from one of the GlobWat models sample files (prc01wb.asc) for regridding ERA5 and ERA-Interim datasets. The ASCII file can be found at: https://storage.googleapis.com/fao-maps-catalog-data/geonetwork/aquamaps/GlobWat-InputP1_prec.zip. You can use the GDAL translator to convert the file from ASCII format to NetCDF format by entering the following command into the terminal: gdal_translate -of netCDF prc01wb.asc globwat_target_grid.nc

   Optional preprocessor settings:

   - area_selection: A region bounding box to extract the data for a specific region. The area selection preprocessor can be used by users to process the data for their desired region. The data will be processed at the global scale if the preprocessor in the recipe is commented.
   - regrid_scheme: The area-weighted regridding scheme is used as a default regridding scheme to ensure that the total volume of water is consistent before and after regridding.
   - langbein_pet: Can be set to True to use langbein function for calculating evspsblpot (default is de bruin method)

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)
6. recipe_globwat.yml
   - pr (atmos, daily or monthly, longitude, latitude, time)
   - tas (atmos, daily or monthly, longitude, latitude, time)
   - psl (atmos, daily or monthly, longitude, latitude, time)
   - rsds (atmos, daily or monthly, longitude, latitude, time)
   - rsdt (atmos, daily or monthly , longitude, latitude, time)

Observations and reformat scripts

Note: download instructions can be obtained with `esmvaltool data info DATASET` or in headers of cmorization scripts.

- ERA-Interim (esmvaltool/cmorizers/data/formatters/datasets/era_interim.py)
- ERA5 (esmvaltool/diag_scripts/cmorizers/era5.py)

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.

5. recipe_globwat.yml

   The forcing data, stored in separate files per timestep and variable.

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.
- Hoogeveen, J., Faurès, J. M., Peiser, L., Burke, J., de Giesen, N. V.: GlobWat–a global water balance model to assess water use in irrigated agriculture, Hydrology & Earth System Sciences Discussions, 2015 Jan 1;12(1), Doi:10.5194/hess-19-3829-2015.
- Langbein, W.B., 1949. Annual runoff in the United States. US Geol. Surv.(https://pubs.usgs.gov/circ/1949/0052/report.pdf)

5.6.4 Hydro forcing comparison

Overview

This recipe can be used to assess the agreement between forcing datasets (i.e. MSWEP, ERA5, ERA-Interim) for a defined catchment. The recipe can be used to:

1. Plot a timeseries of the raw daily data
2. Plot monthly aggregrated data over a defined period
3. Plot the monthly / daily climatology statistics over a defined period

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/hydrology

- `recipe_hydro_forcing.yml`

Diagnostics are stored in esmvaltool/diag_scripts/hydrology/

- `hydro_forcing.py`: Compares and plots precipitation for MSWEP / ERA5 / ERA-5 interim datasets

User settings in recipe

All hydrological recipes require a shapefile as an input to select 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 from HydroSHEDS 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 `hydrology/hydro_forcing.yml`

   Optional preprocessor settings:

   - `extract_shape`: The region specified here should match the catchment

   Required settings for script:

   - `plot_type`: Define which plot function to run. Choices:
     - `timeseries`: Plot a timeseries for the variable data over the defined period
     - `climatology`: Plot the climate statistics over the defined period

   Required settings for ``timeseries`` plots:

   - `time_period`: Defines the period of the output for the correct captions/labels. This value should match the period used for the preprocessor. Choices: `day`, `month`.

Variables

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

Observations

All data can be used directly without any preprocessing.

- ERA-Interim
- ERA5
- MSWEP

Example plots

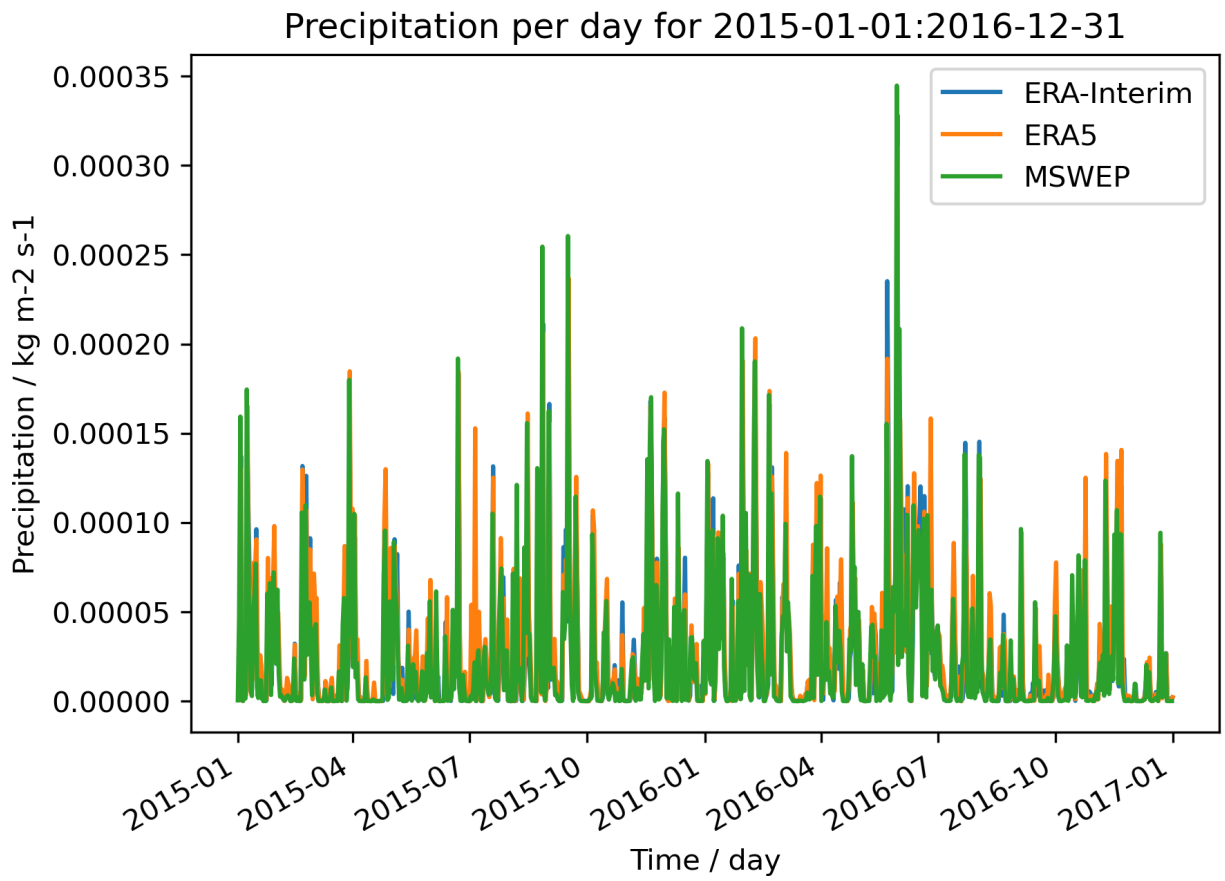


Fig. 230: Precipitation per day for 2015-01-01:2016-12-31.

5.6.5 Landcover diagnostics

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.

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.

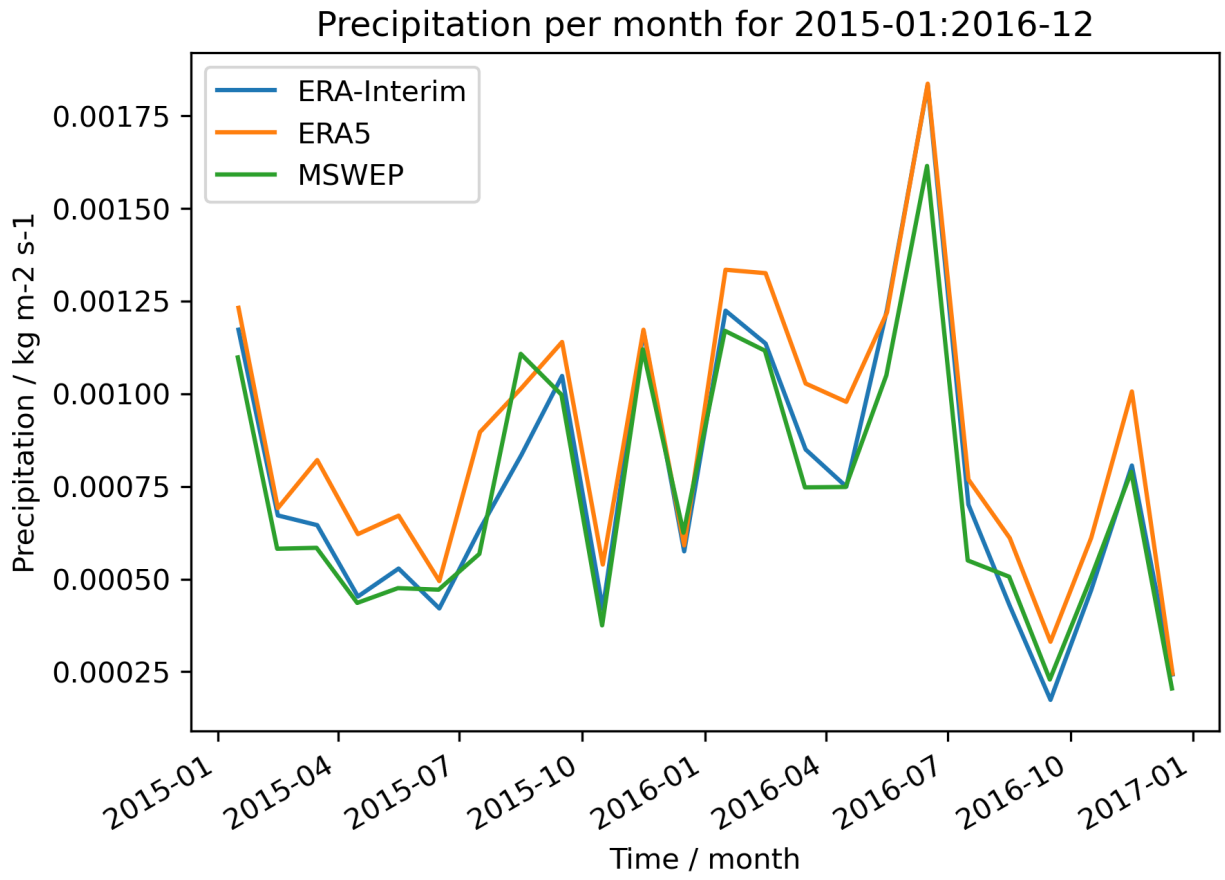


Fig. 231: Precipitation per month for 2015-01:2016-12.

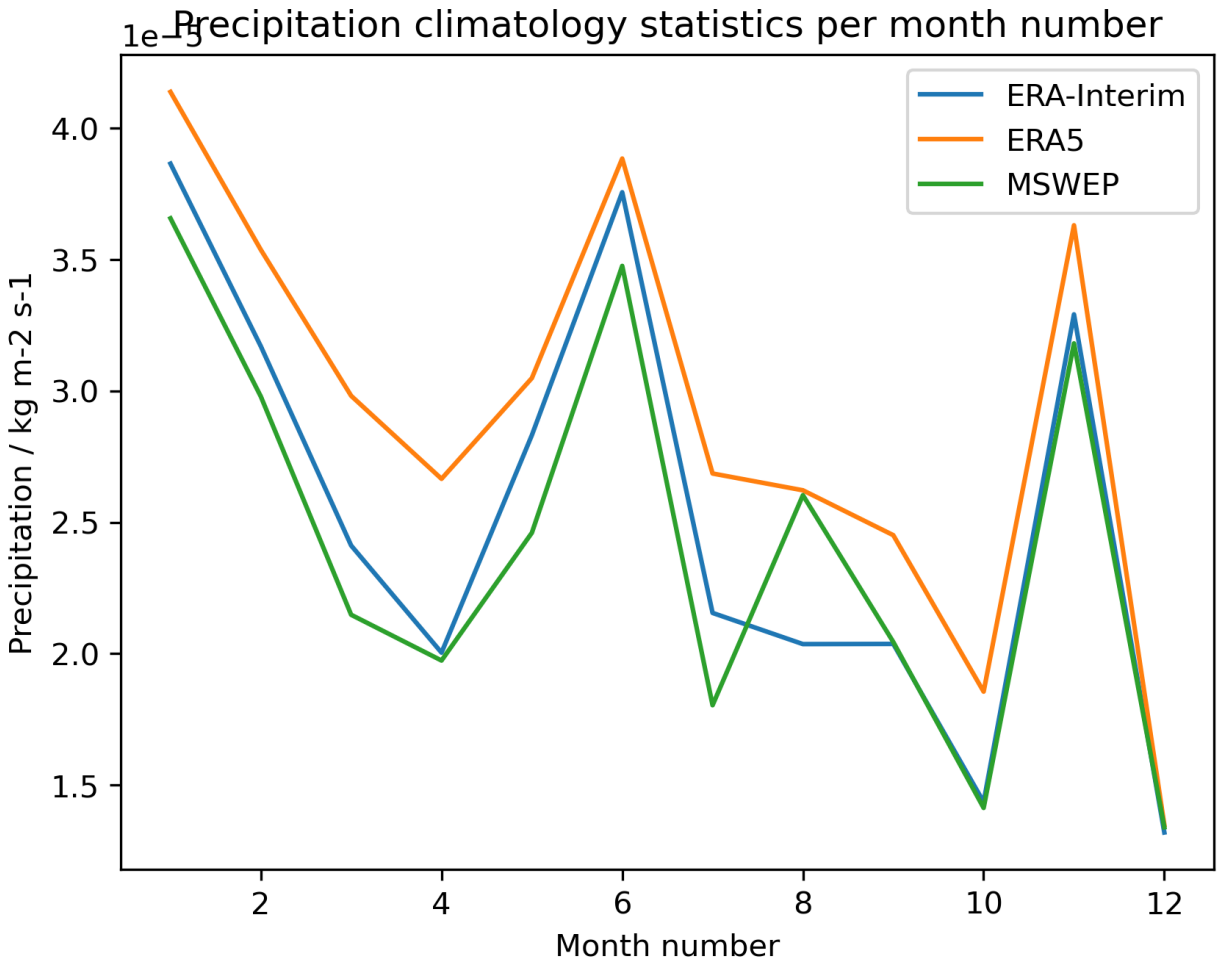


Fig. 232: Precipitation climatology statistics per month number.

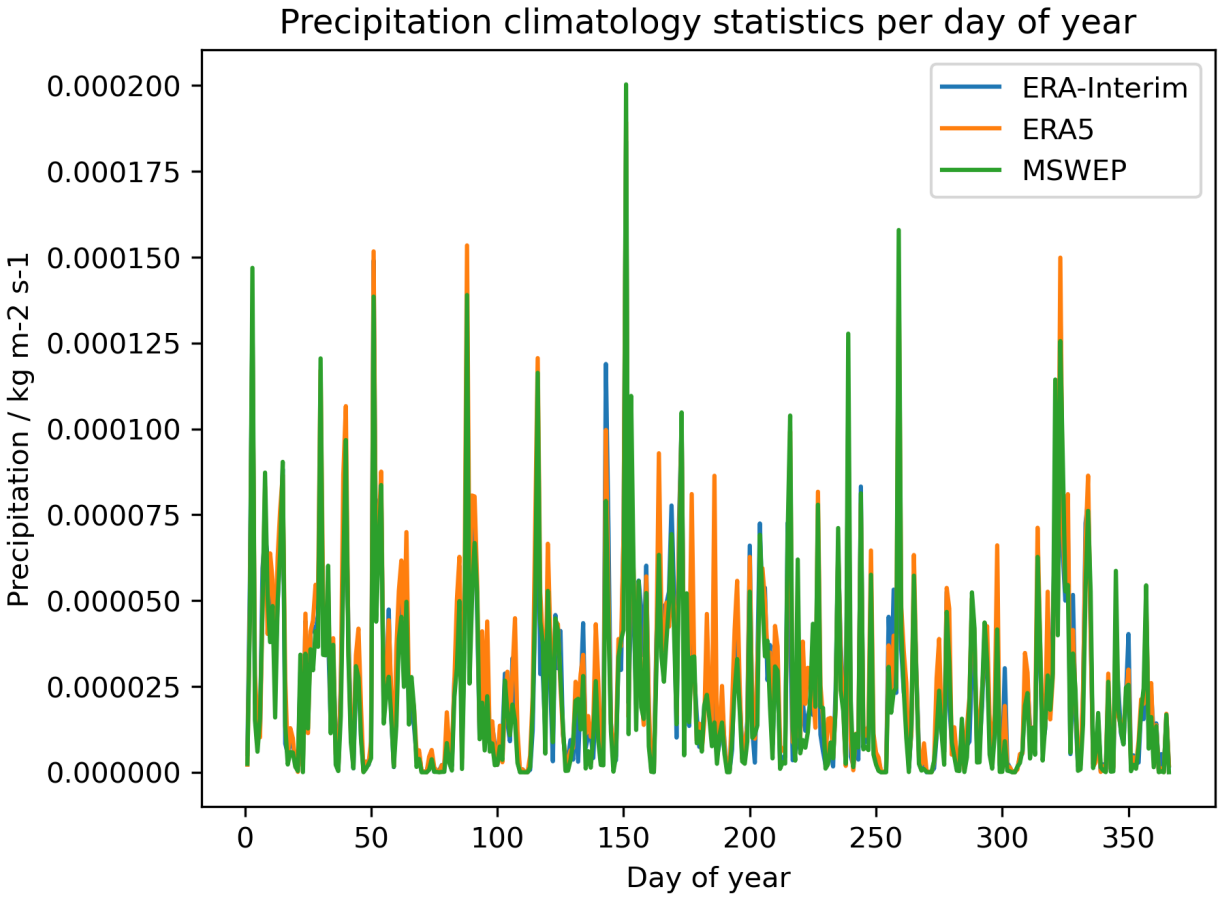


Fig. 233: Precipitation climatology statistics per day of year.

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.

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)

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
baresoilFrac	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.

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

Example plots

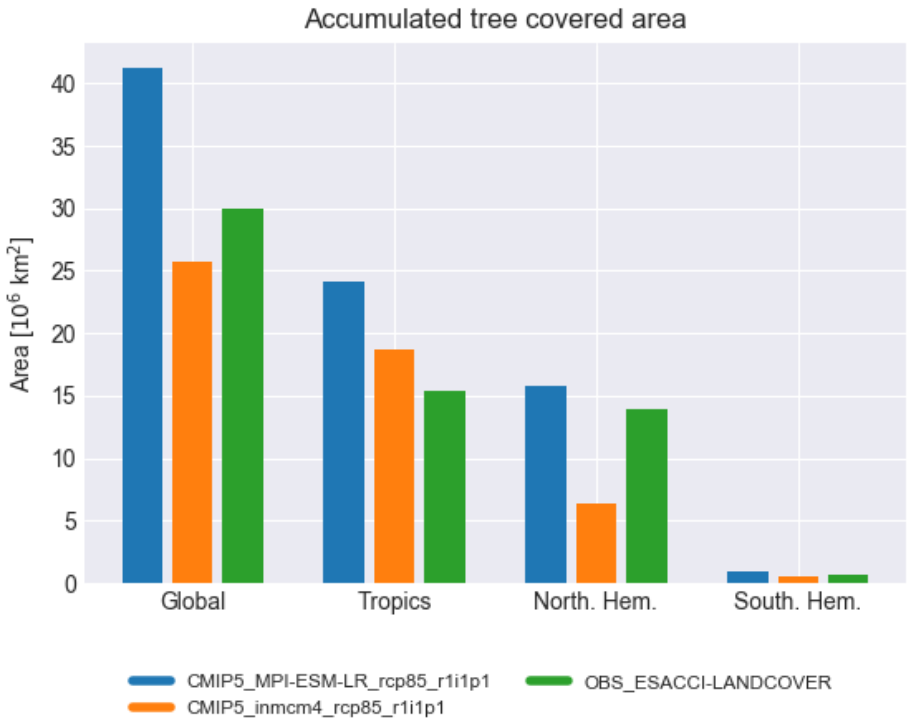


Fig. 234: Accumulated tree covered area for different regions and experiments.

5.6.6 Land and ocean components of the global carbon cycle

Overview

This recipe reproduces most of the figures of Anav et al. (2013):

- Timeseries plot for different regions
- Seasonal cycle plot for different regions
- 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)

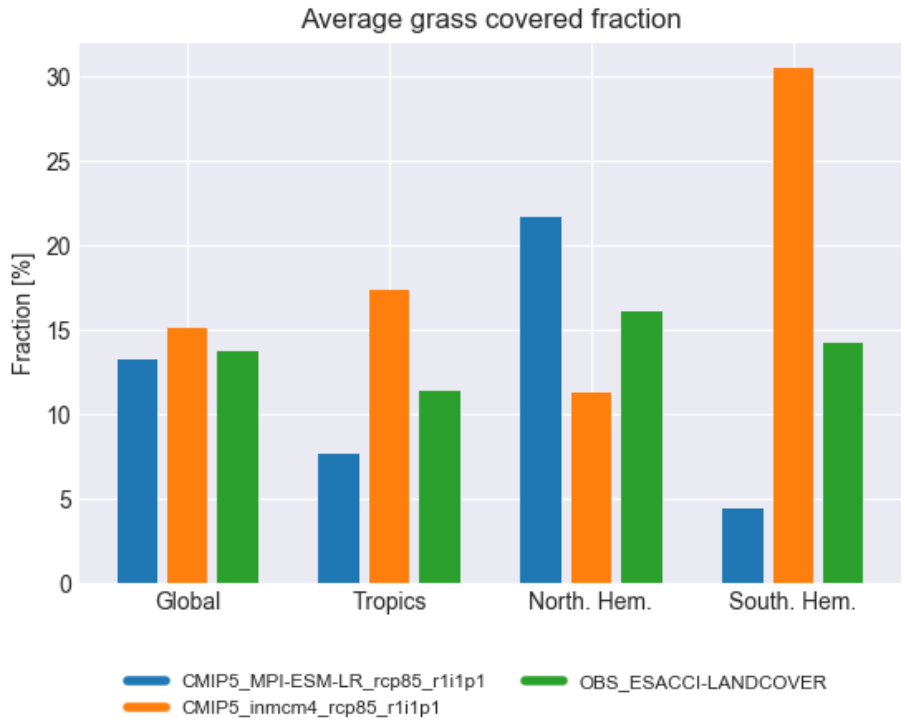


Fig. 235: Average grass cover fraction for different regions and experiments

- 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*).

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.

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*).

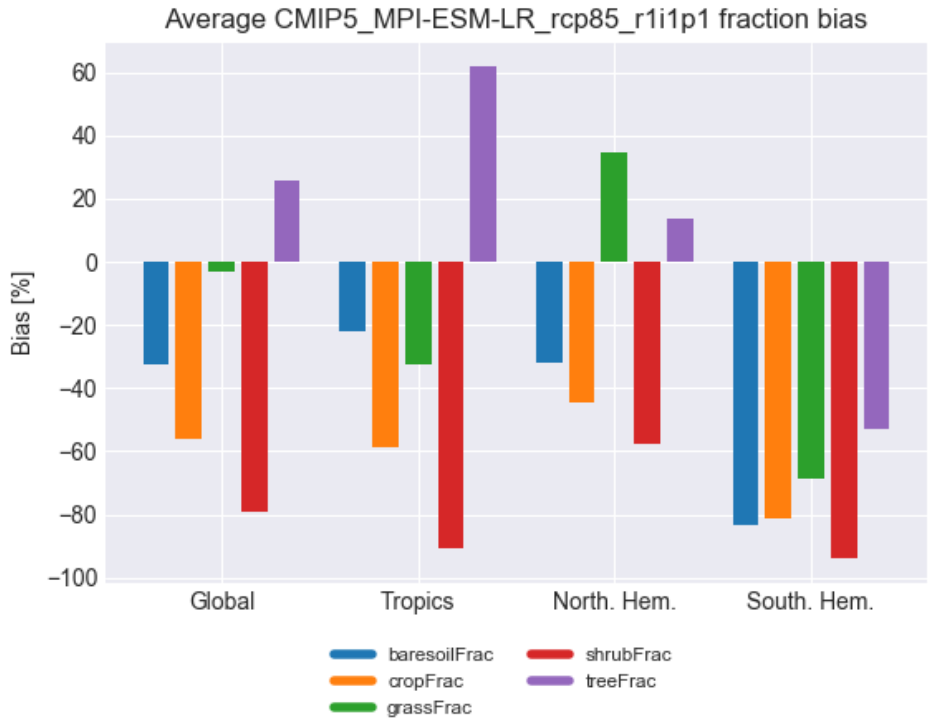


Fig. 236: Biases in five major land cover fractions for different regions and one experiment.

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

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*.

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)

Observations and reformat scripts

- CRU (*tas*, *pr*)
- JMA-TRANSCOM (*nbp*, *fgco2*)
- MTE (*gpp*)
- LAI3g (*lai*)
- NDP (*cveg*)
- HWSD (*csoil*)
- HadISST (*tos*)

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.

Example plots

5.6.7 Runoff, Precipitation, Evapotranspiration

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

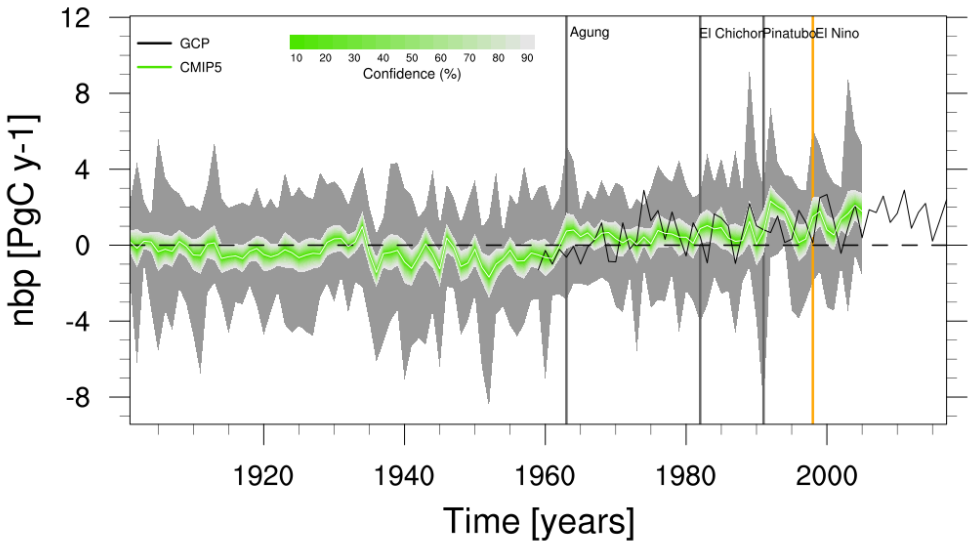


Fig. 237: Time series of global net biome productivity (NBP) over the period 1901-2005. Similar to Anav et al. (2013), Figure 5.

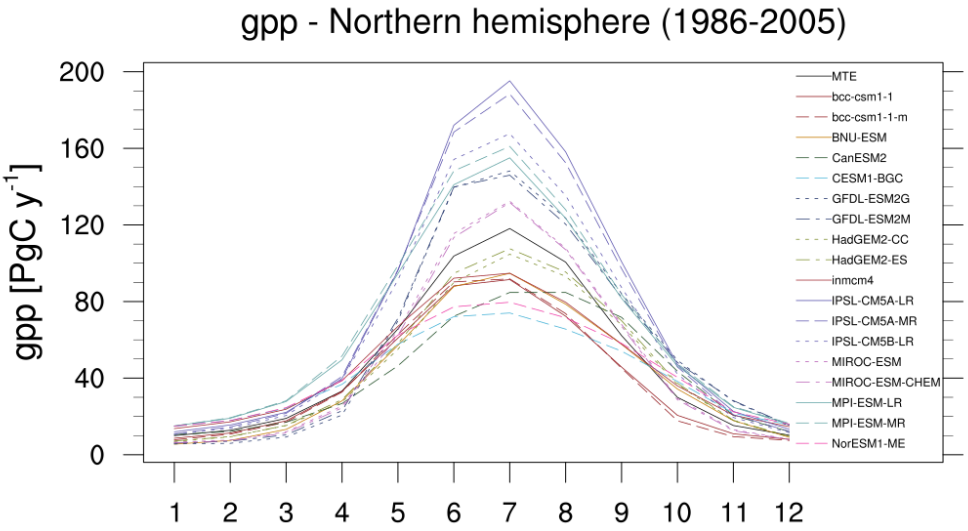


Fig. 238: Seasonal cycle plot for nothern hemisphere gross primary production (GPP) over the period 1986-2005. Similar to Anav et al. (2013), Figure 9.

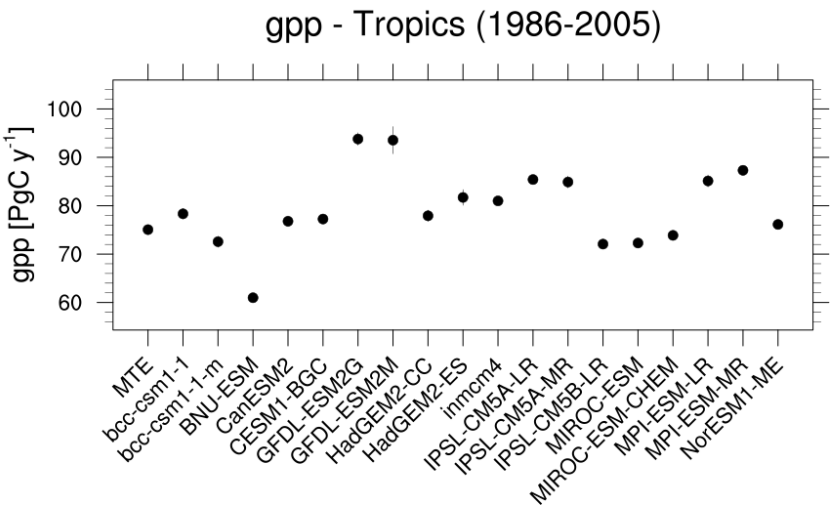


Fig. 239: Errorbar plot for tropical gross primary production (GPP) over the period 1986-2005.

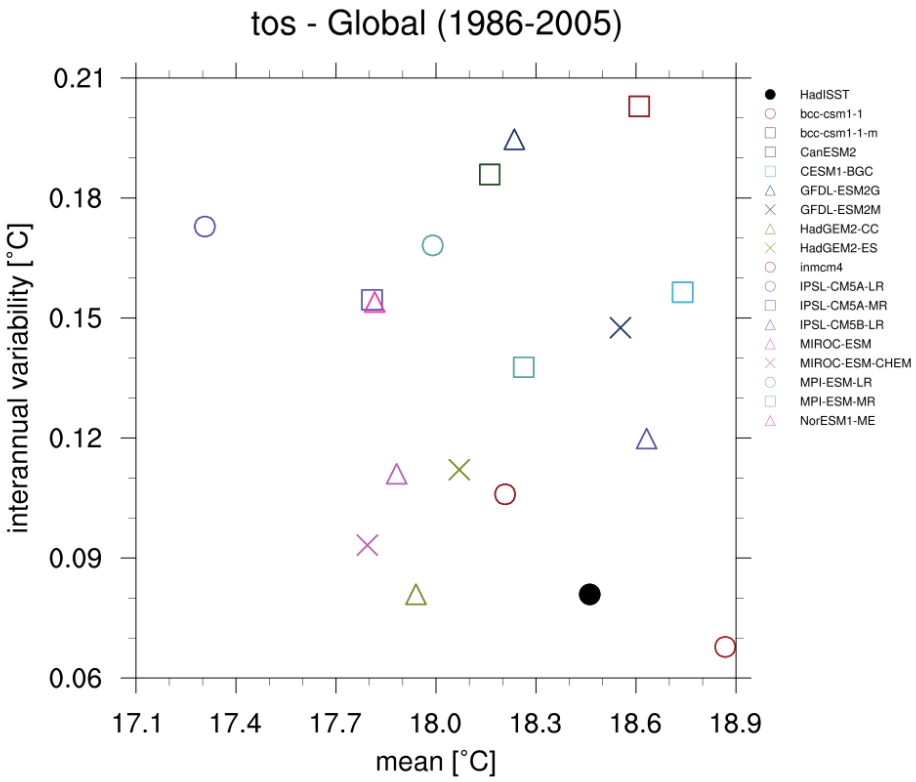


Fig. 240: Scatterplot for interannual variability and mean of global sea surface temperature (TOS) over the period 1986-2005.

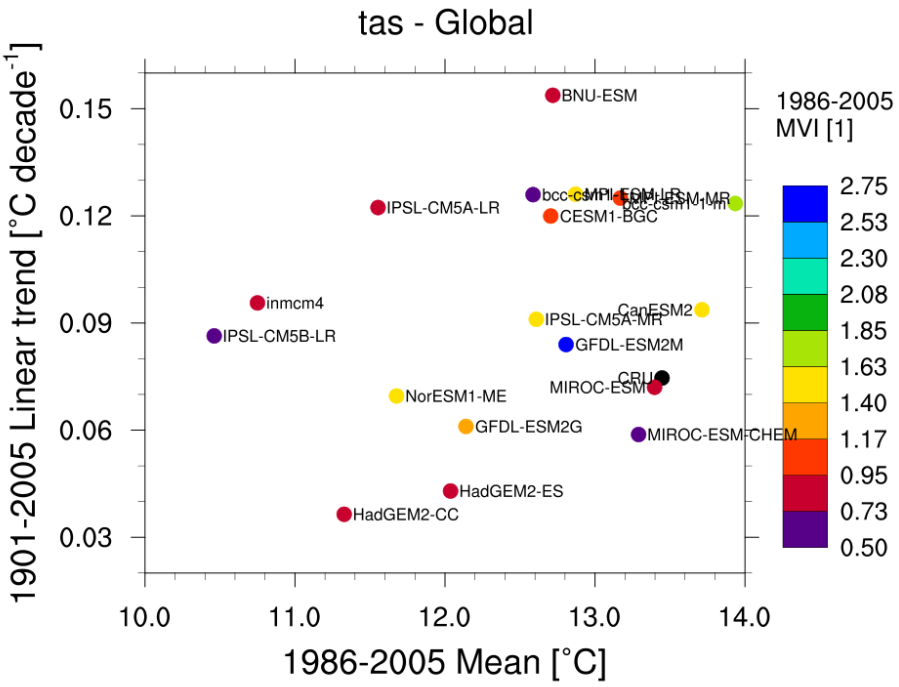


Fig. 241: Scatterplot for multiyear average of 2m surface temperature (TAS) in x axis, its linear trend in y axis, and MVI. Similar to Anav et al. (2013) Figure 1 (bottom).

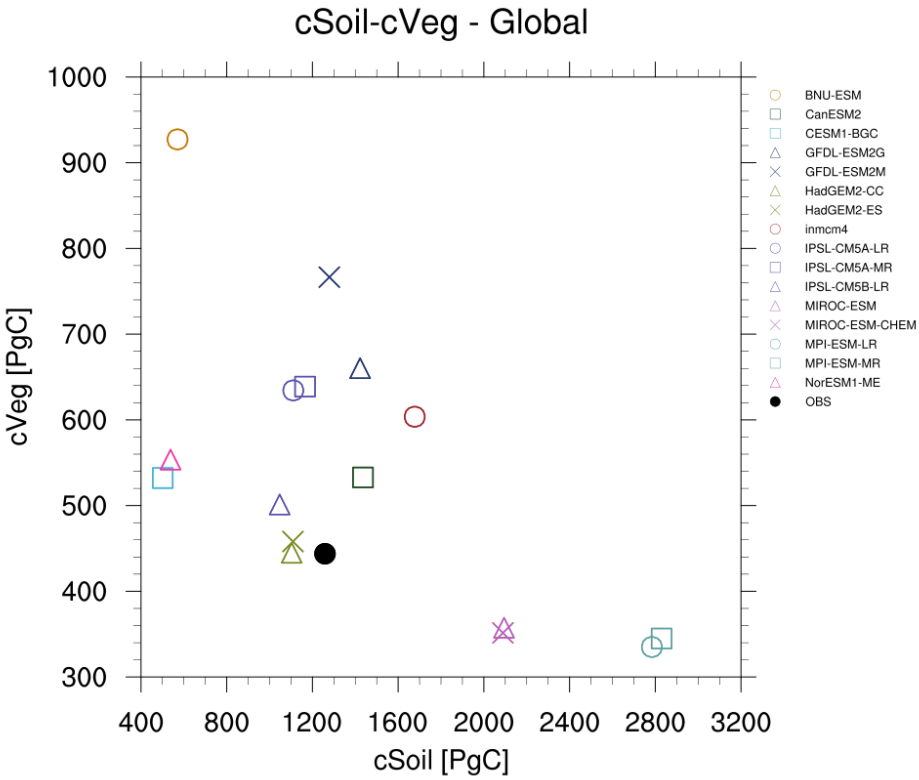


Fig. 242: Scatterplot for vegetation carbon content (cVeg) and soil carbon content (cSoil) over the period 1986-2005. Similar to Anav et al. (2013), Figure 12.

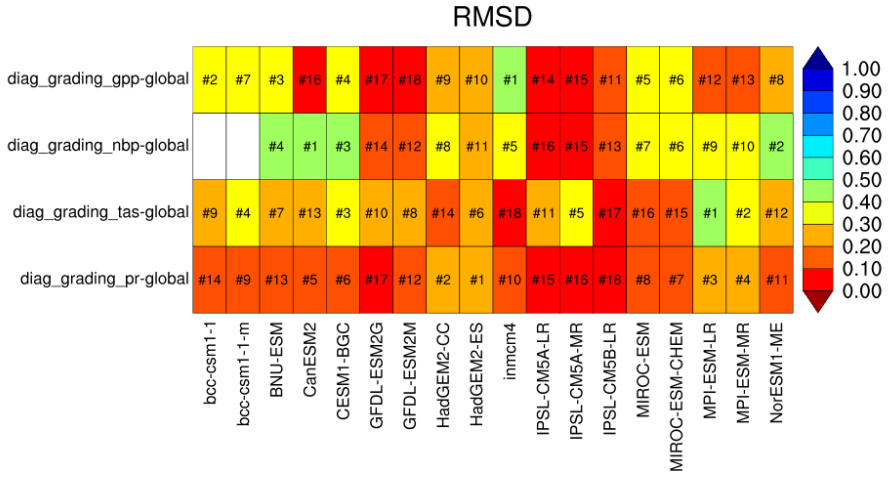


Fig. 243: Performance metrics plot for carbon-cycle-relevant diagnostics.

(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.

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

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.

Variables

- evspsbl (atmos, monthly mean, time latitude longitude)
- pr (atmos, monthly mean, time latitude longitude)
- mrro (land, monthly mean, time latitude longitude)

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.

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

Example plots

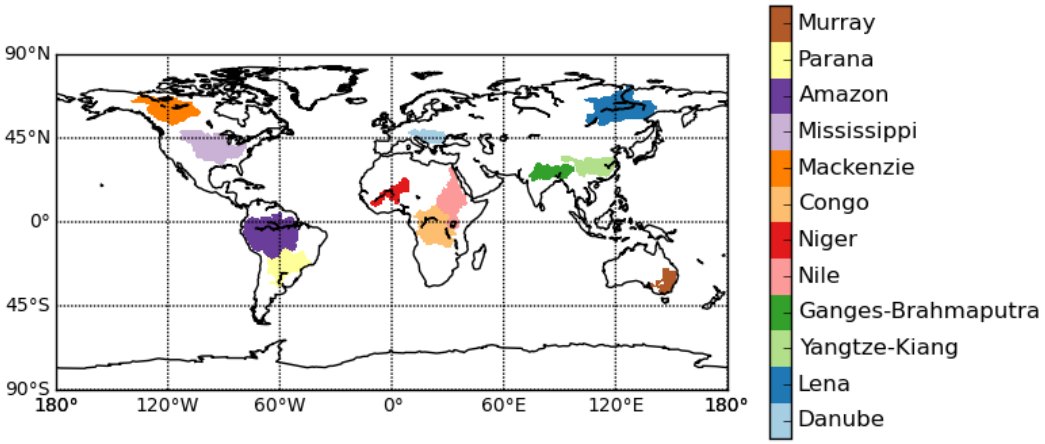


Fig. 244: Catchment definitions used in the diagnostics.

5.6.8 Climate drivers of fire

Overview

This diagnostic includes:

- burnt fraction
- fire weather control
- fuel load/continuity control

The diagnostic relies on the processing of fire climate drivers through the ConFire model (dedicated branch from the GitHub repository available at https://github.com/douglask3/Bayesian_fire_models/tree/AR7_REF) and is based on *Jones et al. (2024)*. The diagnostic computes the burnt fraction for each grid cell based on a number of drivers. Additionally, the respective controls due to fire weather and fuel load/continuity are computed. The stochastic control corresponds to the unmodelled processed influencing to fire occurrence. The ESMValTool diagnostic includes only the relevant part of the evaluation code (see https://github.com/douglask3/Bayesian_fire_models/blob/AR7_REF/fire_model/ConFire.py for now) as the model run is done offline beforehand. The corresponding result files are made available through a Zenodo archive which can be retrieved inside the diagnostic. The archive is published under the DOI 10.5281/zenodo.14917244, but a specific version of the files can be specified using the corresponding DOI, e.g. the default link to https://zenodo.org/records/14917245. The other option is to provide a local directory which contains the necessary model files.

The ConFire model relies on a variety of observational datasets (see references):

- Global Fire Emissions Database version 5
- MODIS MOD44B
- ESA CCI Biomass
- ISIMIP3a GSWP3-W5E5 dataset

Note: If custom ConFire run files are to be used for the diagnostic, they need to be present inside the auxiliary directory defined in the user configuration file *config_user.yml*.

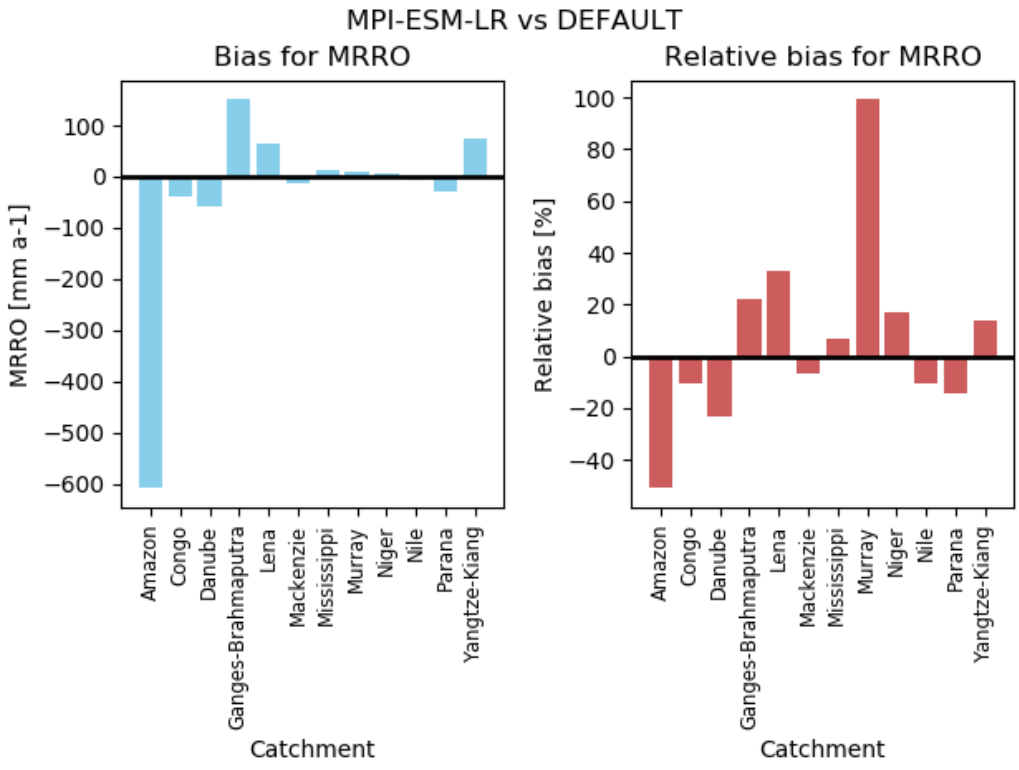


Fig. 245: Barplot indicating the absolute and relative bias in annual runoff between MPI-ESM-LR (1970-2000) and long term GRDC data for specific catchments.

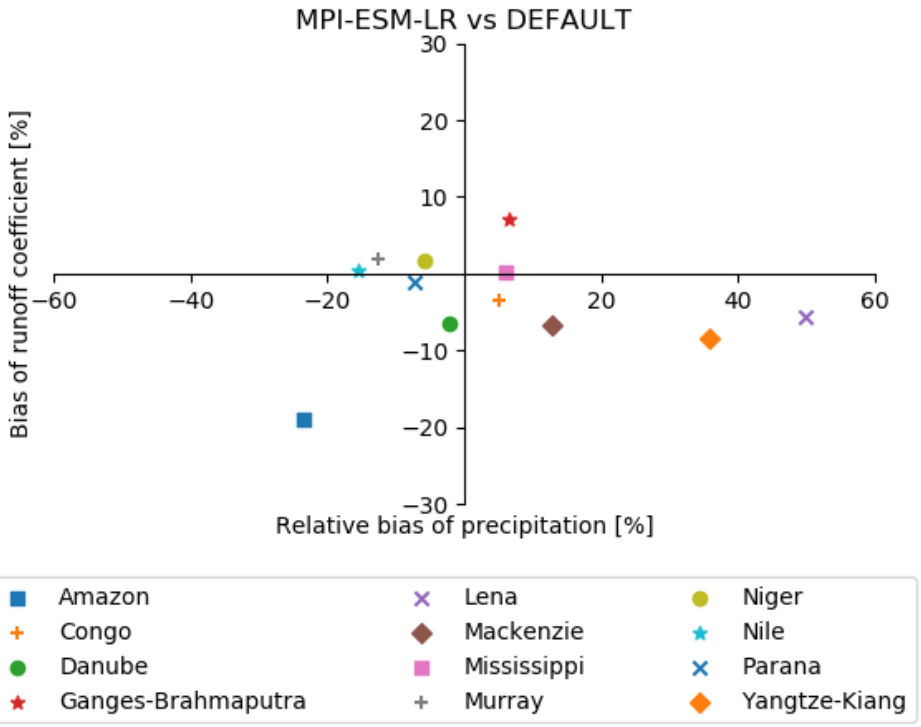


Fig. 246: Biases in runoff coefficient (runoff/precipitation) and precipitation for major catchments of the globe. The MPI-ESM-LR historical simulation (1970-2000) is used as an example.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/:

- recipe_ref_fire.yml

Diagnostics are stored in esmvaltool/diag_scripts/fire/:

- fire_diagnostic.py: main diagnostic script calling a util function from diagnostic_run_ConFire.py.
- diagnostic_run_ConFire.py: script containing utils functions to run the ConFire model.

User settings in recipe

1. Script fire_diagnostic.py

   Required settings for script

   - var_order: list of climate drivers in the order corresponding to the one specified in the corresponding file from the confire_param directory.

   Optional settings for script

   - confire_param: path to the directory containing the required files to run the ConFire model or Zenodo URL to retrieve files from a Zenodo archive. If custom files are used, the corresponding directory needs to be present inside the auxiliary data directory defined inside the user configuration. This defaults to the original Zenodo archive otherwise.
   - remove_vpd_files: Removing or not the computed vapor pressure deficit files. It will only apply if the vapor pressure deficit is part of var_order. This defaults to False.
   - remove_confire_files: Removing or not the files produced during the ConFire model evaluation. This defaults to False.

   Required settings for variables

   Optional settings for variables

   Required settings for preprocessor

   Optional settings for preprocessor

2. Script diagnostic_run_ConFire.py

   Required settings for script

   - confire_param: path to the directory containing the required files to run the ConFire model (or downloaded files from a Zenodo archive).
   - files_input: list containing tuples of variable name and file path for each climate driver present in var_order.

   Optional settings for script

   - model_name: string containing the input data model for legending the plots.
   - timerange: string containing the time range of the input data for legending the plots.
   - project: string or list containing the input data project(s).
   - experiment: string or list containing the input data experiment(s).

   Required settings for variables

   Optional settings for variables

   Required settings for preprocessor

   Optional settings for preprocessor

Variables

- pr (atmos, monthly mean, longitude latitude time)
- tasmax (atmos, monthly mean, longitude latitude time)
- treeFrac (land, monthly mean, longitude latitude time)
- vegFrac (land, monthly mean, longitude latitude time)
- cveg (land, monthly mean, longitude latitude time)
- tas (atmos, monthly mean, longitude latitude time): used to compute vpd.
- hurs (atmos, monthly mean, longitude latitude time): used to compue vpd.

References

- Jones, M. W., Kelley, D. I., Burton, C. A., Di Giuseppe, F., Barbosa, M. L. F., Brambleby, E., Hartley, A. J., Lombardi, A., Mataveli, G., McNorton, J. R., Spuler, F. R., Wessel, J. B., Abatzoglou, J. T., Anderson, L. O., Andela, N., Archibald, S., Armenteras, D., Burke, E., Carmenta, R., Chuvieco, E., Clarke, H., Doerr, S. H., Fernandes, P. M., Giglio, L., Hamilton, D. S., Hantson, S., Harris, S., Jain, P., Kolden, C. A., Kurvits, T., Lampe, S., Meier, S., New, S., Parrington, M., Perron, M. M. G., Qu, Y., Ribeiro, N. S., Saharjo, B. H., San-Miguel-Ayanz, J., Shuman, J. K., Tanpipat, V., van der Werf, G. R., Veraverbeke, S., and Xanthopoulos, G.: State of Wildfires 2023–2024, Earth Syst. Sci. Data, 16, 3601–3685, https://doi.org/10.5194/essd-16-3601-2024, 2024.
- Yang Chen, Joanne Hall, Dave van Wees, Niels Andela, Stijn Hantson, Louis Giglio, Guido R. van der Werf, Douglas C. Morton, & James T. Randerson. (2023). Global Fire Emissions Database (GFED5) Burned Area (0.1) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.7668424.
- DiMiceli, C., Sohlberg, R., Townshend, J. (2022). MODIS/Terra Vegetation Continuous Fields Yearly L3 Global 250m SIN Grid V061 [Data set]. NASA EOSDIS Land Processes Distributed Active Archive Center. Accessed 2025-04-01 from https://doi.org/10.5067/MODIS/MOD44B.061.
- Santoro, M.; Cartus, O. (2024): ESA Biomass Climate Change Initiative (Biomass_cci): Global datasets of forest above-ground biomass for the years 2010, 2015, 2016, 2017, 2018, 2019, 2020 and 2021, v5.01. NERC EDS Centre for Environmental Data Analysis, 22 August 2024. https://dx.doi.org/10.5285/bf535053562141c6bb7ad831f5998d77.
- Stefan Lange, Matthias Mengel, Simon Treu, Matthias Büchner (2022): ISIMIP3a atmospheric climate input data (v1.0). ISIMIP Repository. https://doi.org/10.48364/ISIMIP.982724.

Example plots

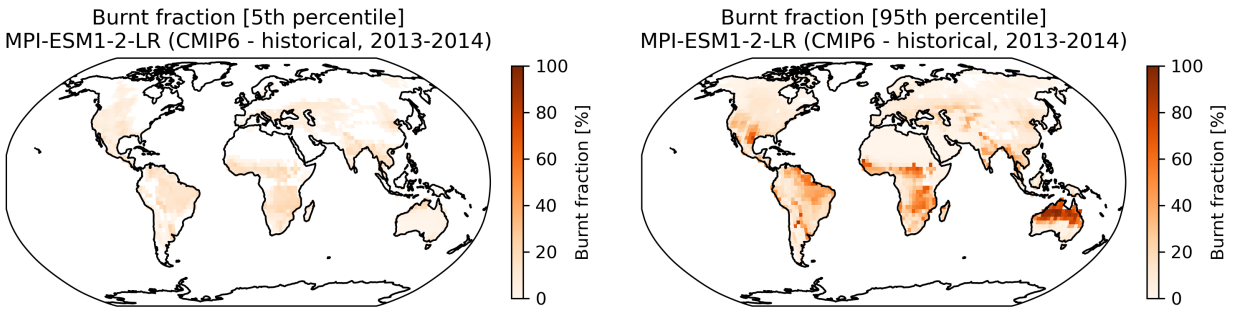


Fig. 247: Burnt area fraction for the MPI-ESM1-2-LR model (CMIP-historical experiment) for the time period 2013-2014 as computed with the ConFire model *Jones et al. (2024)*.

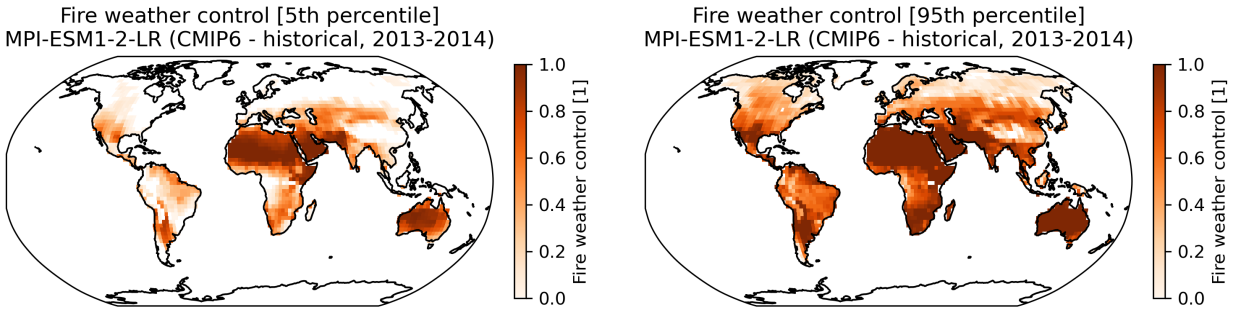


Fig. 248: Fire weather control for the MPI-ESM1-2-LR model (CMIP-historical experiment) for the time period 2013-2014 as computed with the ConFire model *Jones et al. (2024)*.

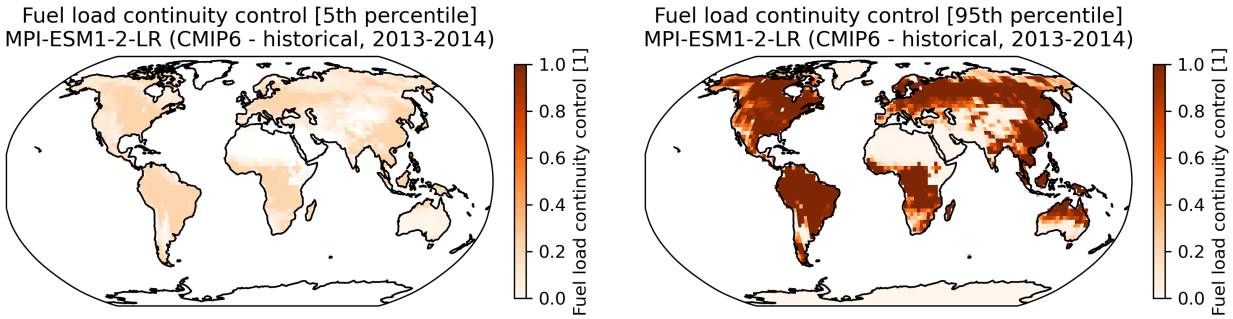


Fig. 249: Fuel load continuity control for the MPI-ESM1-2-LR model (CMIP-historical experiment) for the time period 2013-2014 as computed with the ConFire model *Jones et al. (2024)*.

5.7 Ocean

5.7.1 Recipe for evaluating Arctic Ocean

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.

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
```

(continues on next page)

(continued from previous page)

```
# 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
```

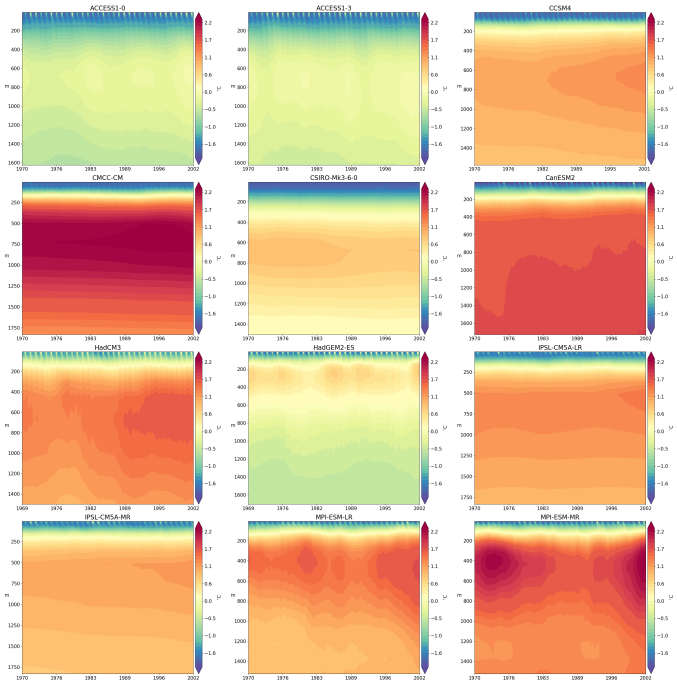


Fig. 250: Hovmoller diagram of monthly spatially averaged potential temperature in the Eurasian Basin of the Arctic Ocean for selected CMIP5 climate models (1970-2005).

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 lineraly 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
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
```

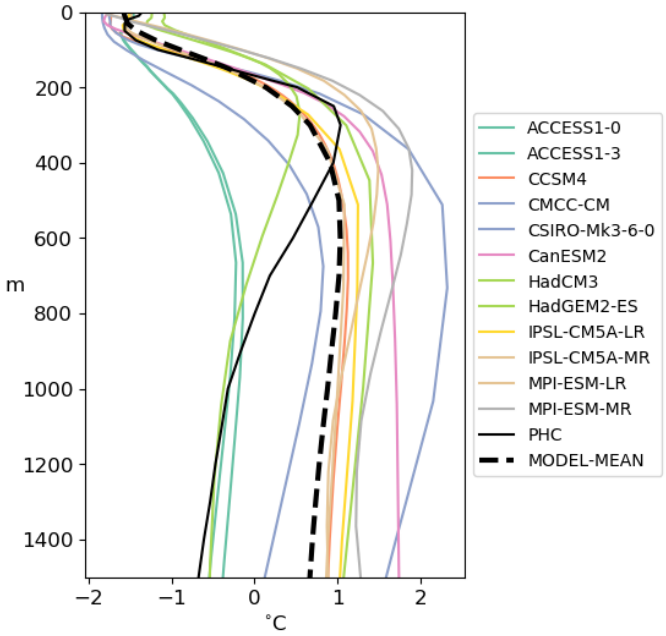


Fig. 251: Mean (1970-2005) vertical potential temperature distribution in the Eurasian basin for participating CMIP5 coupled ocean models, PHC3 climatology (dotted red line) and multi-model mean (dotted black line).

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
```

(continues on next page)

(continued from previous page)

```
plot2d_vars: ['thetao', 'so']
# Define colormap (as a list, same size as list with variables)
# Only cmaps from matplotlib and cmocean are supported.
# 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
```

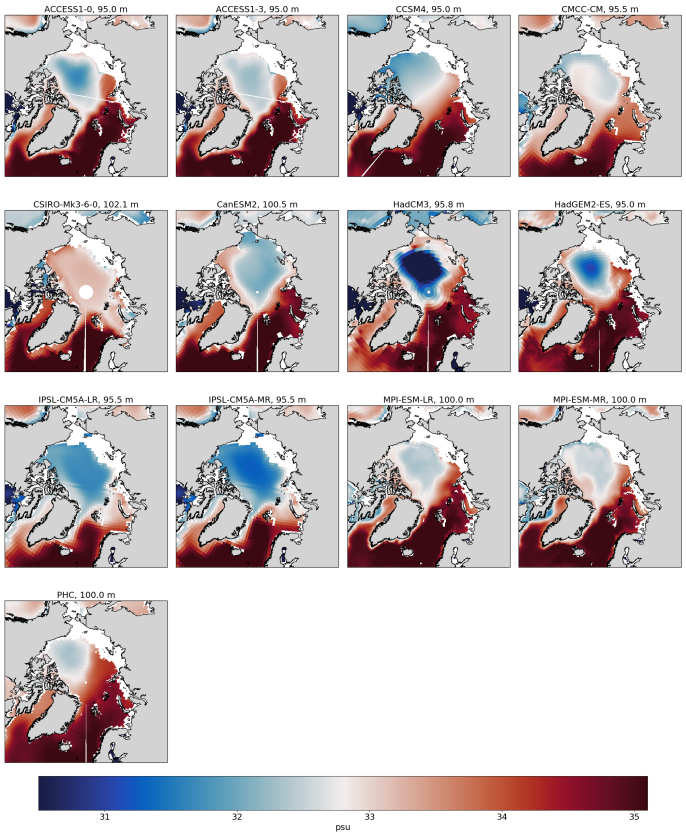


Fig. 252: Mean (1970-2005) salinity distribution at 100 meters.

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]
```

(continues on next page)

(continued from previous page)

```
# 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]]
# number of columns in the bias plots
plot2d_bias_ncol: 3
```

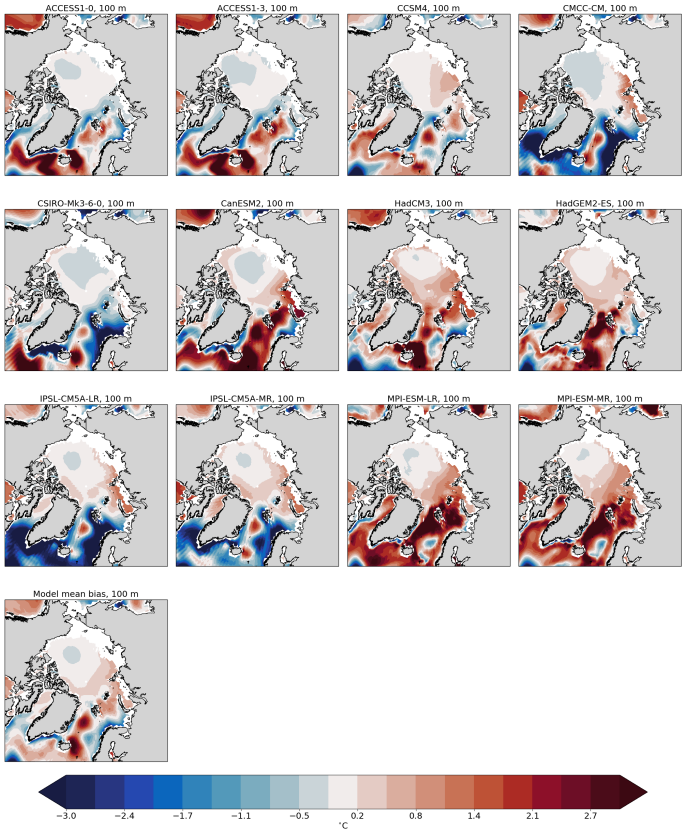


Fig. 253: Mean (1970-2005) salinity bias at 100m relative to PHC3 climatology

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
```

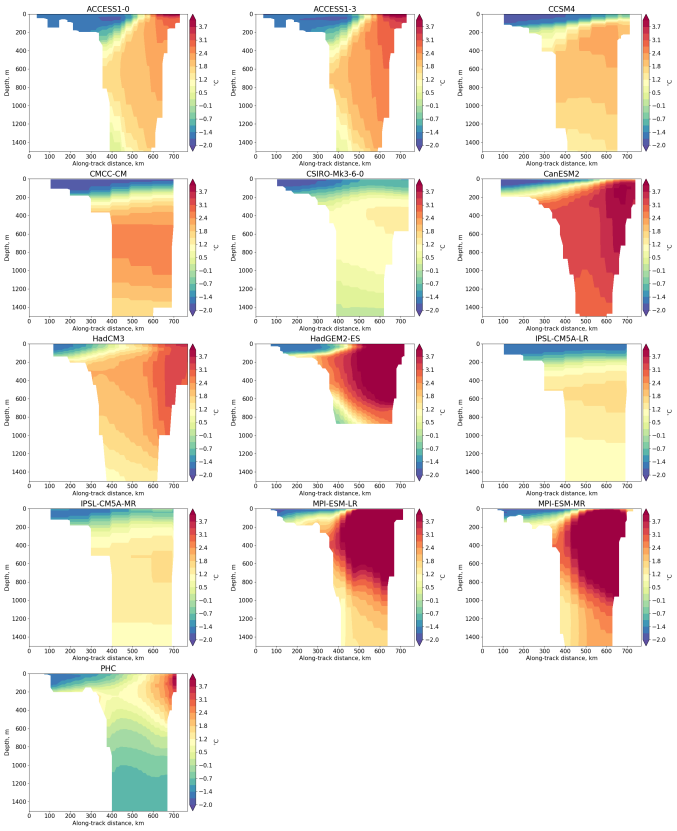


Fig. 254: Mean (1970-2005) potential temperature across the Fram strait.

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.

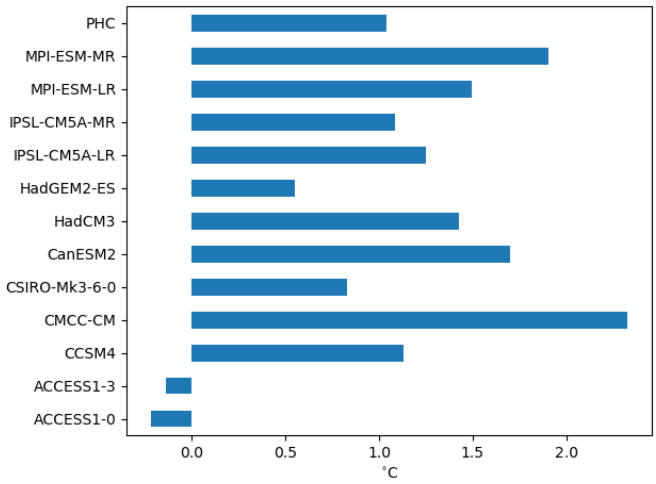


Fig. 255: Mean (1970-2005) Atlantic Water core temperature. PHC33 is an observed climatology.

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
```

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.

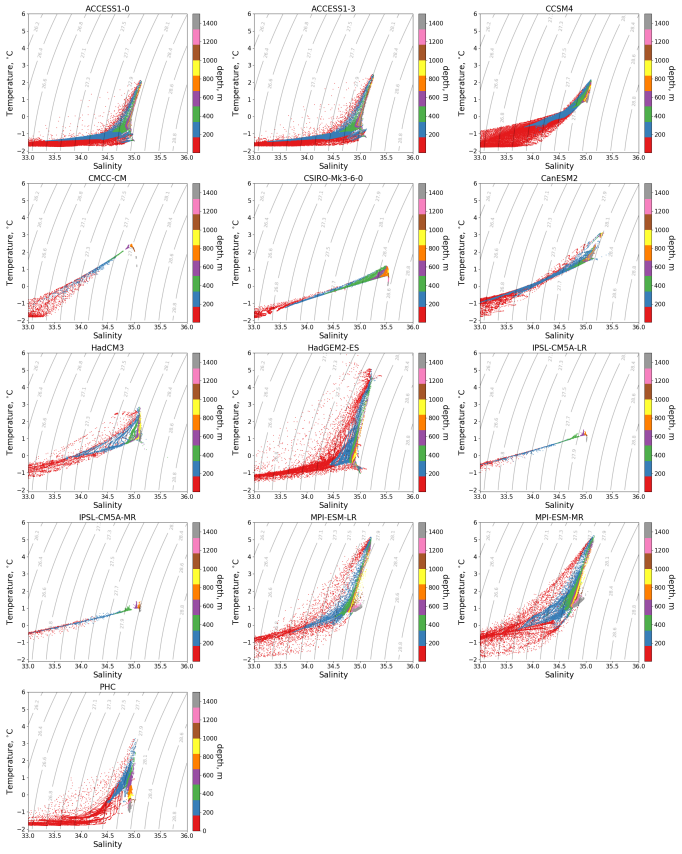


Fig. 256: Mean (1970-2005) T-S diagrams for Eurasian Basin of the Arctic Ocean.

- 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

5.7.2 Climate Variability Diagnostics Package (CVDP)

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].

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_cvdp.yml

Diagnostics are stored in diag_scripts/cvdp/

- cvdp_wrapper.py

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.

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)

Observations and reformat scripts

None.

References

[1] http://www.cesm.ucar.edu/working_groups/CVC/cvdp/

[2] https://github.com/NCAR/CVDP-ncl

Example plots

5.7.3 Nino indices, North Atlantic Oscillation (NAO), Souther Oscillation Index (SOI)

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)
- Southern Oscillation Index (SOI)

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

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

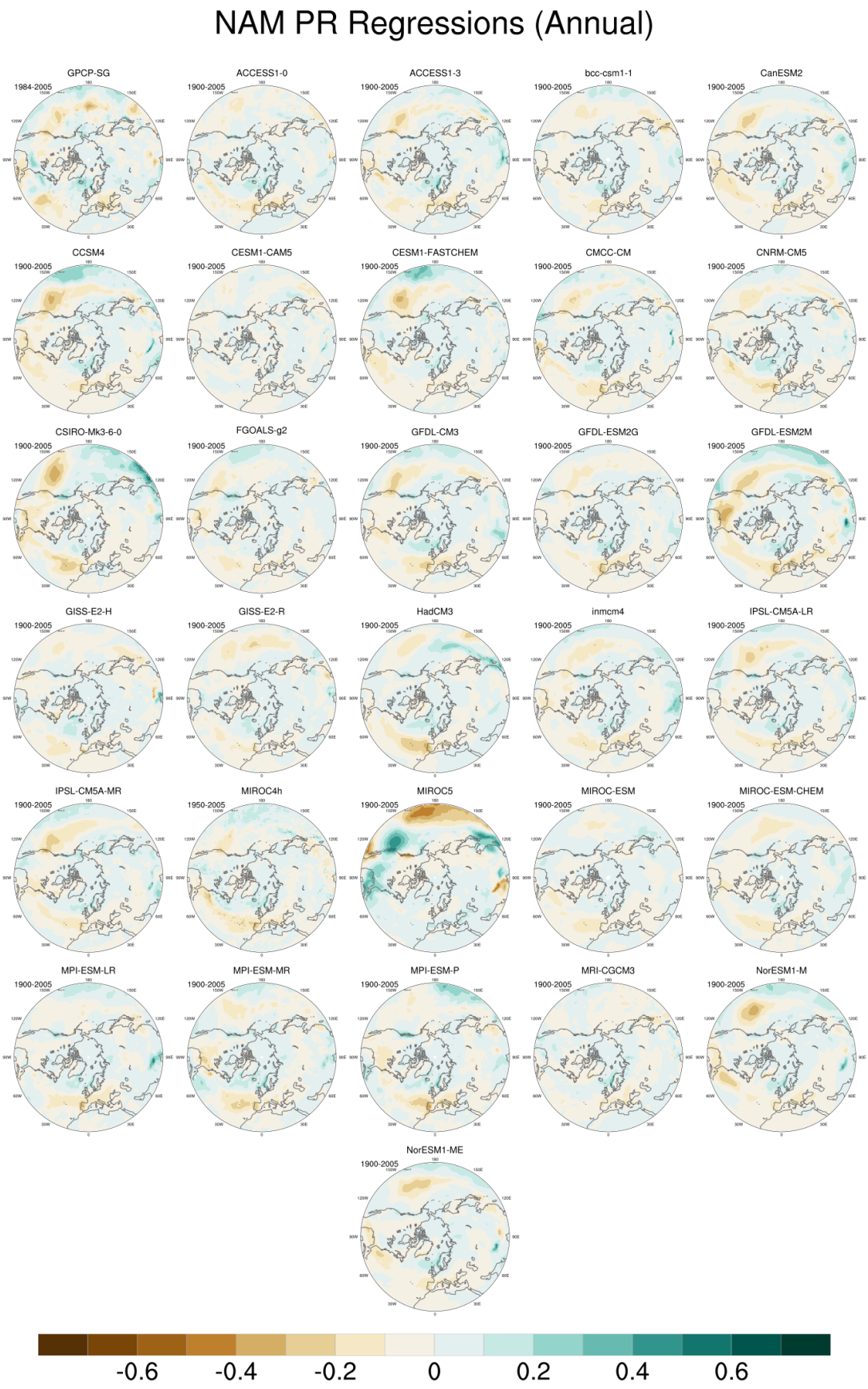


Fig. 257: Regression of the precipitation anomalies (PR) onto the Northern Annular Mode (NAM) index for the time period 1900-2005 for 30 CMIP5 models and observations (GPCP (pr) / IFS-Cy31r2 (psl); time period 1984-2005).

- 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

Variables

- all variables (atmos/ocean, monthly, longitude, latitude, time)

Observations and reformat scripts

None

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.

Example plots

5.7.4 Ocean chlorophyll in ESMs compared to ESA-CCI observations.

Overview

This recipe compares monthly surface chlorophyll from CMIP models to ESA CCI ocean colour chlorophyll (ESACCI-OC). The observations are the merged sensor geographic monthly L3S chlor_a data Sathyendranath et al. (2019). Multiple models and different observational versions can be used by the script.

The recipe_esacci_oc.yml produces an image showing four maps. Each of these four maps shows latitude vs longitude and the chlorophyll value. The four plots are: ESACCI-OC v5.0 chlorophyll, the CMIP6 model, the bias model-observation and the ratio model/observations. The script also produces a scatter plot for all coordinates with the model on the x-axis and the observations on the y axis and a line of best fit with the parameter values given in the panel.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/ocean/

- recipe_esacci_oc.yml

Diagnostics are stored in esmvaltool/diag_scripts/ocean/

- diagnostic_model_vs_obs.py

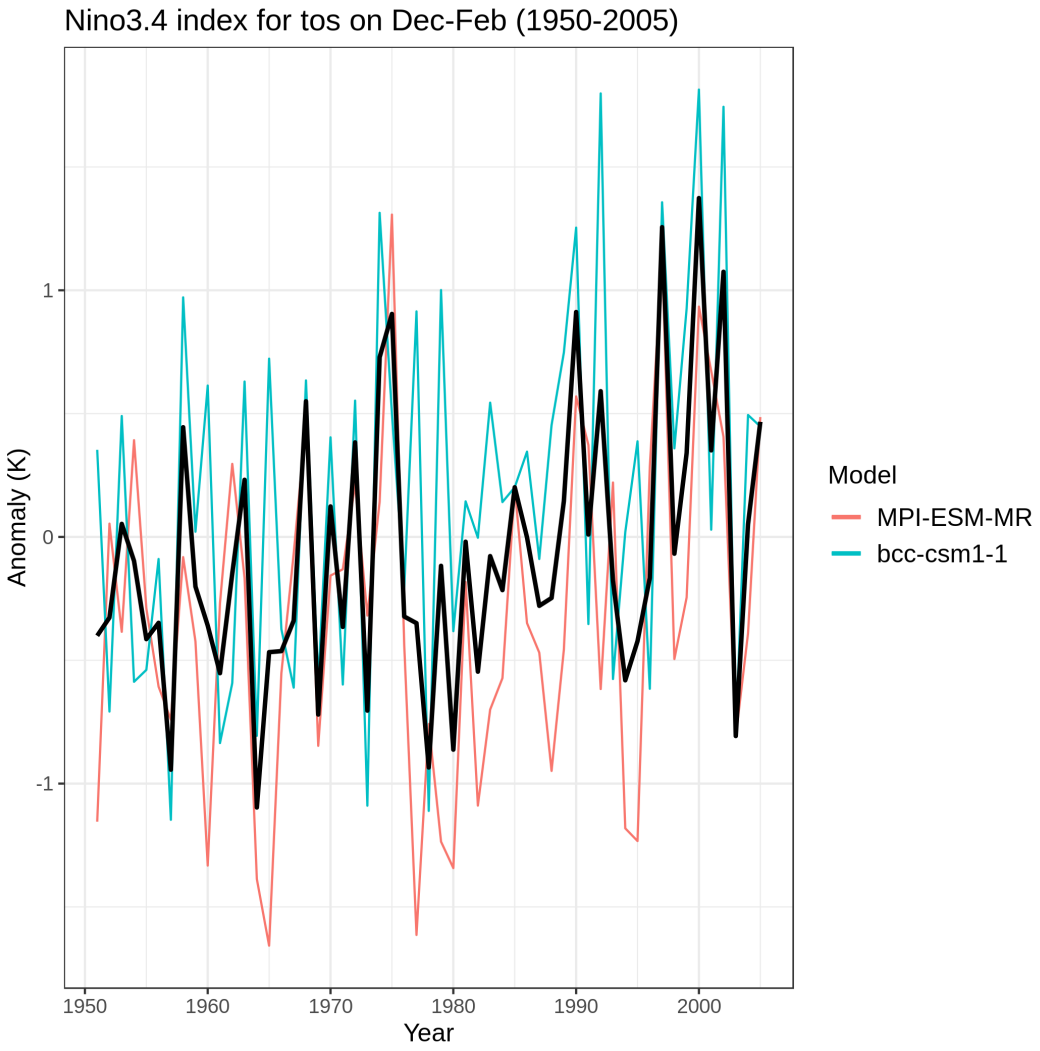


Fig. 258: Time series of the standardized sea surface temperature (tos) area averaged over the Nino 3.4 region during the boreal winter (December-January-February). The time series correspond to the MPI-ESM-MR (red) and BCC-CSM1-1 (blue) models and their mean (black) during the period 1950-2005 for the ensemble r1p1i1 of the historical simulations.

User settings in recipe

1. Script diagnostic_model_vs_obs.py

   Required settings for script

   - observational_dataset: name of reference dataset (e.g. {dataset: ESACCI-OC,})

Variables

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

Observations and reformat scripts

- ESACCI-OC (chl)

  Reformat script: reformat_scripts/obs/reformat_obs_esacci_oc.py

References

- Sathyendranath, S., et al. (2019), An ocean-colour time series for use in climate studies: the experience of the Ocean-Colour Climate Change Initiative (OC-CCI). Sensors: 19, 4285. doi:10.3390/s19194285.
- ESACCI-OC dataset: http://dx.doi.org/10.5285/00b5fc99f9384782976a4453b0148f49

Example plots

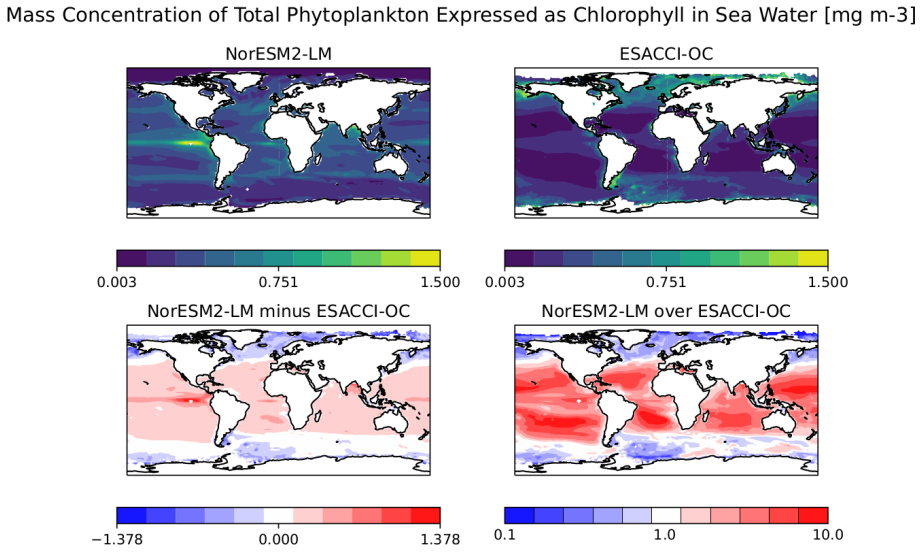


Fig. 259: Surface chlorophyll from ESACCI-OC ocean colour data version 5.0 and the CMIP6 model NorESM2-LM. This model overestimates chlorophyll compared to the observations.

5.7.5 ENSO CLIVAR metrics - reproducing background climatology and ENSO characteristics

Overview

Reproducing some ENSO metrics from the [CLIVAR package](https://pcmdi.llnl.gov/pmp-preliminary-results/interactive_plot/portrait_plot/enso_metric/enso_metrics_interactive_portrait_plots_v20231121.html)

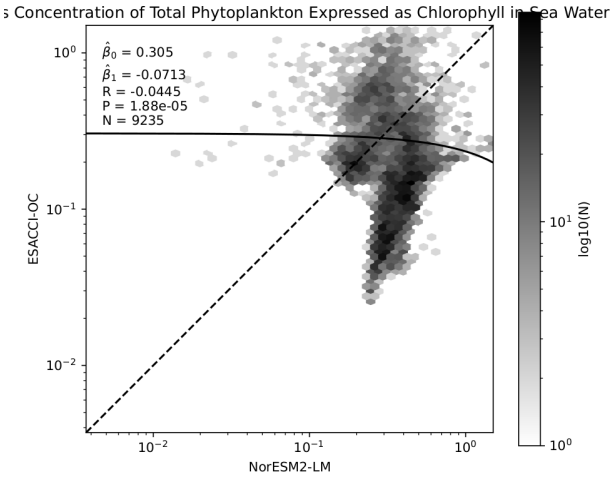


Fig. 260: Scatter plot of surface chlorophyll from ESACCI-OC ocean colour data version 5.0 and the CMIP6 model NorESM2-LM.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- ref/recipe_enso_characteristics.yml
- ref/recipe_enso_basicclimatology.yml

Diagnostics are stored in esmvaltool/diag_scripts/enso_metrics/

- **enso_diag1metrics.py**: metrics for basic ENSO characteristics.
- **climatology_diagnostic1.py**: metrics for background climatology.
- **climatology_diagnosticlevel2.py**: creating map plots for background climatology.

User settings in recipe

Variables

- tos (Omon, monthly)
- areacello (Ofx)
- pr (Amon, monthly)
- ts (Amon, monthly)
- tauu (Amon, monthly)

Observations and reformat scripts

- GPCP-SG
- TROPFLUX

References

- [1] https://pcmdi.llnl.gov/pmp-preliminary-results/interactive_plot/portrait_plot/enso_metric/enso_metrics_interactive_portrait_plots_v20231121.html
- [2] https://github.com/CLIVAR-PRP/ENSO_metrics/

Example plots

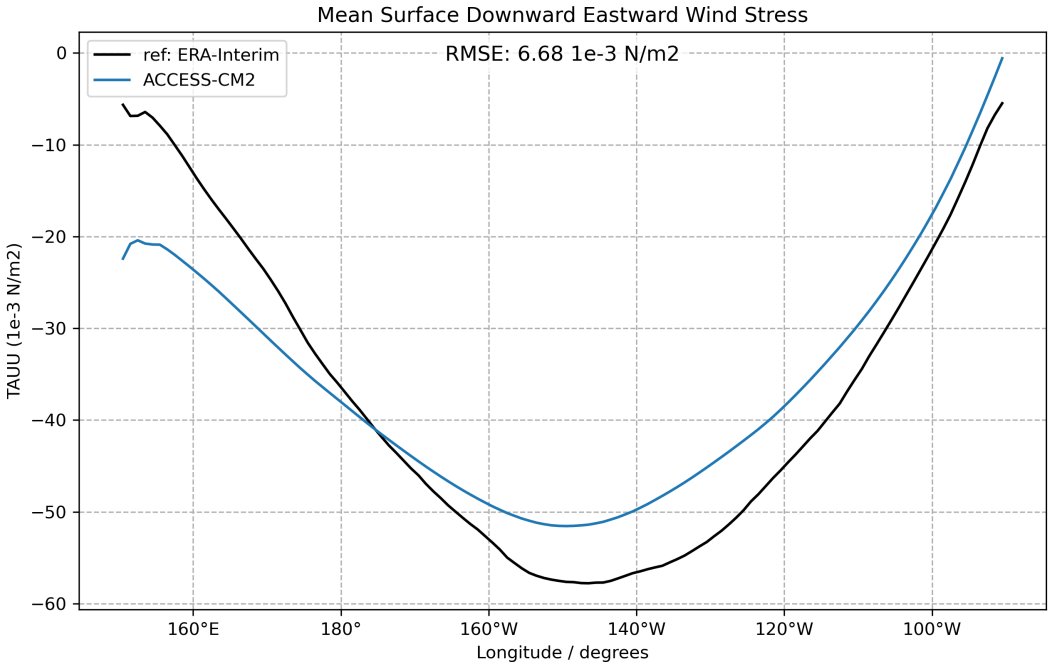


Fig. 261: Bias in the zonal structure of zonal wind stress (Taux) in the equatorial Pacific (5°S-5°N averaged), showing mainly trade winds bias (usually a decreased circulation in the central Pacific and an increased circulation in the western Pacific)

5.7.6 ENSO CLIVAR metrics - reproducing ENSO characteristics, lifecycle, seasonality, teleconnections

Overview

Reproducing ENSO metrics from the [CLIVAR package](https://pcmdi.llnl.gov/pmp-preliminary-results/interactive_plot/portrait_plot/enso_metric/enso_metrics_interactive_portrait_plots_v20231121.html) with dive down plots of ENSO diagnostics including characteristics, lifecycle, seasonality, feedbacks and teleconnections.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_enso_climatology1.yml
- recipe_enso_climatology_level2-3.yml
- recipe_enso_characteristics.yml
- recipe_enso_divedown9-10_14-15.yml
- recipe_enso_divedown11-13.yml
- recipe_enso_feedback.yml
- recipe_enso_feedbacknhf.yml
- recipe_enso_teleconnection.yml

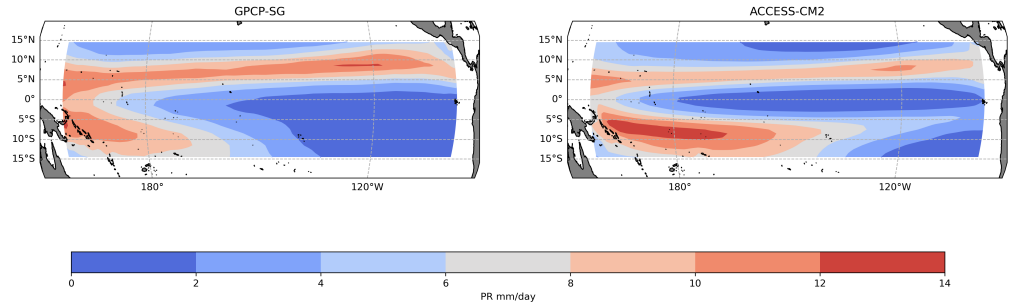


Fig. 262: Bias of time-mean precipitation (PR) in the equatorial Pacific, showing mainly the double intertropical convergence zone (ITCZ) bias. The left and right maps show respectively the reference and the model

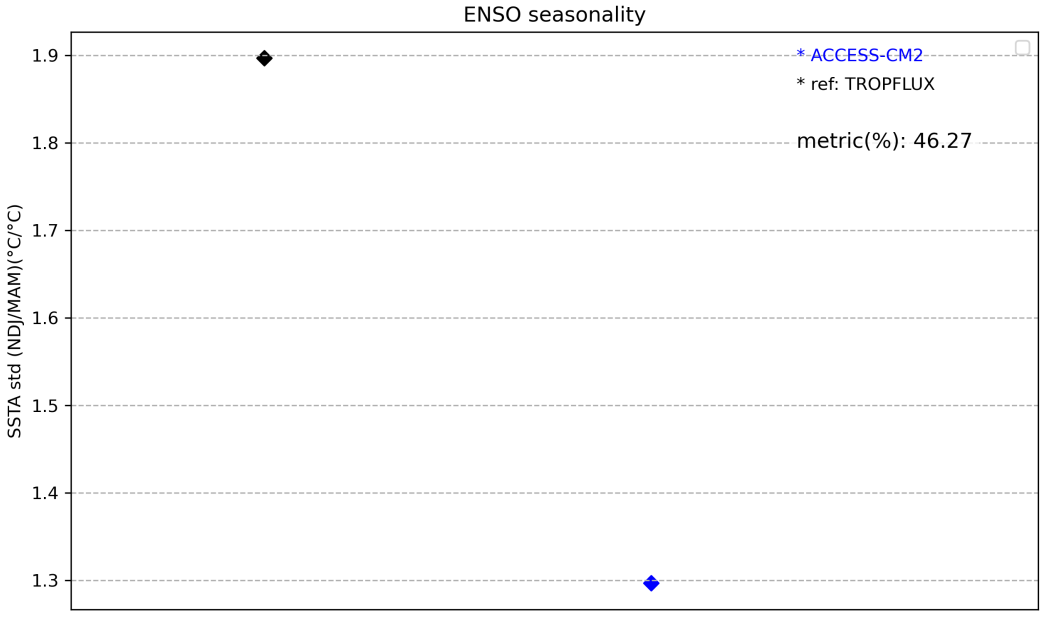


Fig. 263: Ratio of winter over spring's standard deviation of sea surface temperature anomalies (SSTA) in the central equatorial Pacific (Niño3.4 averaged), showing the seasonal timing of SSTA. All models are on the same line.

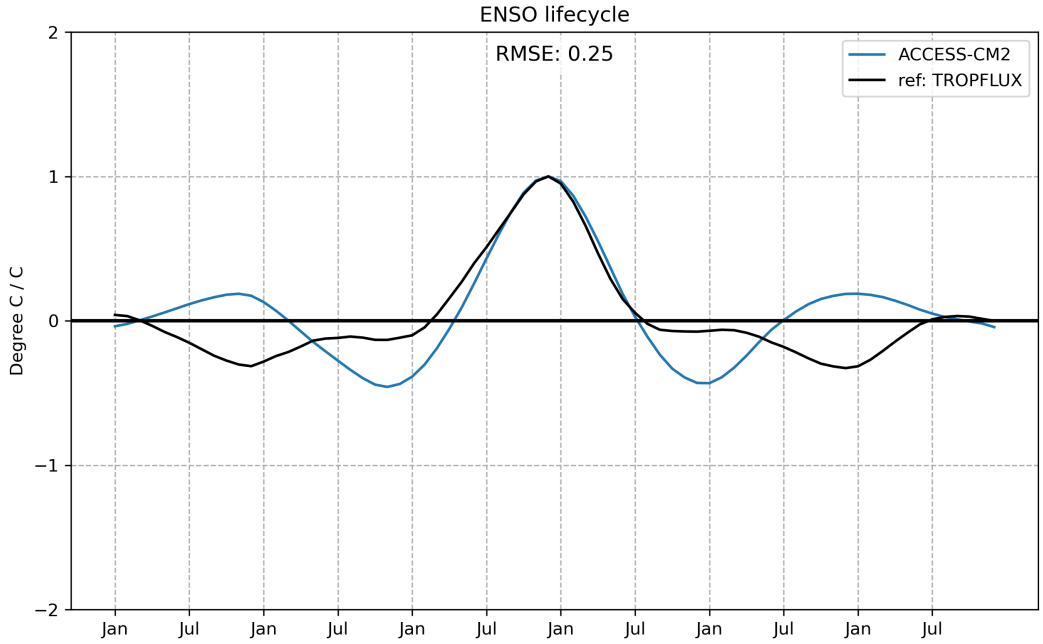


Fig. 264: Temporal structure of sea surface temperature anomalies (SSTA) in the central equatorial Pacific (Niño3.4 averaged), showing the temporal evolution of SSTA associated with ENSO. Observation is dashed black line.

Diagnostics are stored in esmvaltool/diag_scripts/enso_metrics/

- **matrix.py**: reads metrics in work_dir from csv file written out in climatology_diagnostic1, use for other groups of metrics
- **climatology_diagnostic1.py**: computes basic climatology for level 1 metrics
- **climatology_diagnosticlevel2.py**: creates basic climatology level 2 plots
- **climatology_diagnosticlevel3.py**: creates basic climatology level 3 plots
- **enso_diag1metrics.py**: computes basic ENSO characteristics metrics
- **enso_divedowns/ampseasskew11-13.py**: dive down scripts for ENSO amplitude, seasonality and skewness
- **enso_divedowns/durdiver_14-15.py**: dive down scripts for ENSO duration and diversity
- **enso_divedowns/lifecycle_10.py**: dive down scripts for ENSO lifecycle
- **enso_divedowns/pattern_9.py**: dive down scripts for ENSO pattern
- **feedback/feedback_metrics_lvl4.py**: feedback level4 plots
- **feedback/feedback_metrics.py**: feedback level3 plots
- **feedback/feedback_nhf_lvl3_4.py**: feedback NHF level3-4 plots
- **feedback/feedback_nhf_metric.py**: feedback NHF metrics
- **teleconnections_metrics.py**: computes and plots teleconnections metrics

User settings in recipe

1. Script: **matrix.py**

   Required settings for script

   - **diag_metrics**: diagnostic name and script name in *yml* of the diagnostic that computes all the metrics so it can find the *csv* in the *work_dir* - eg. diagnostic_metrics/plot_script

Variables

- tos (Omon, monthly)
- areacello (Ofx)
- pr (Amon, monthly)
- ts (Amon, monthly)
- areacella (fx)
- tauu (Amon, monthly)
- zos (Omon, monthly)
- hfns (hfls+hfss, Amon, monthly)

Observations and reformat scripts

- HadISST
- TropFLUX
- GPCP-SG
- ERA-Interim

References

- [1] https://pcmdi.llnl.gov/pmp-preliminary-results/interactive_plot/portrait_plot/enso_metric/enso_metrics_interactive_portrait_plots_v20231121.html
- [2] https://github.com/CLIVAR-PRP/ENSO_metrics/

Example plots

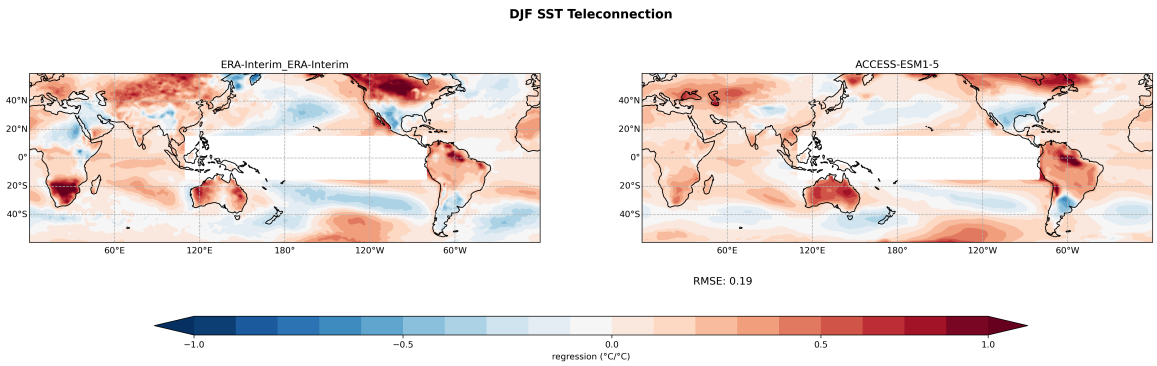


Fig. 265: PR or SST anomalies on Earth (between 60°S-60°N), showing the location associated with ENSO.

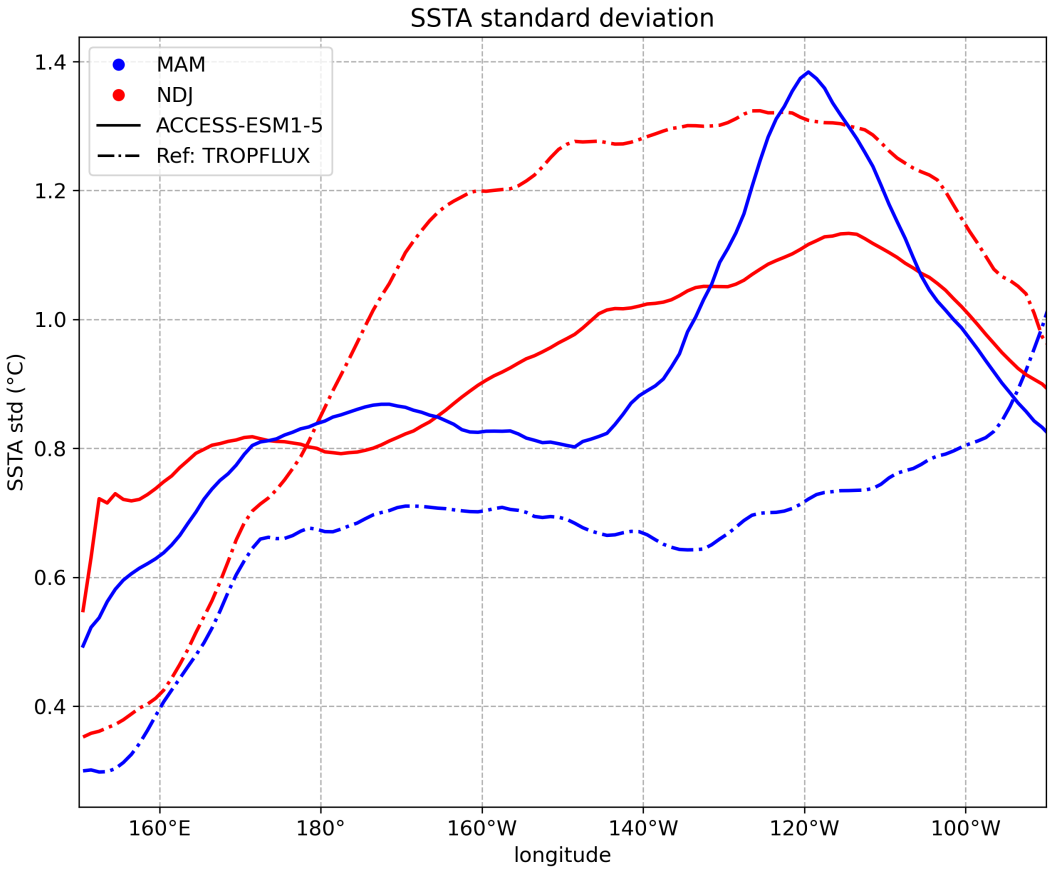


Fig. 266: Seasonality level 4 plot of boreal summer/winter SSTA standard deviation.

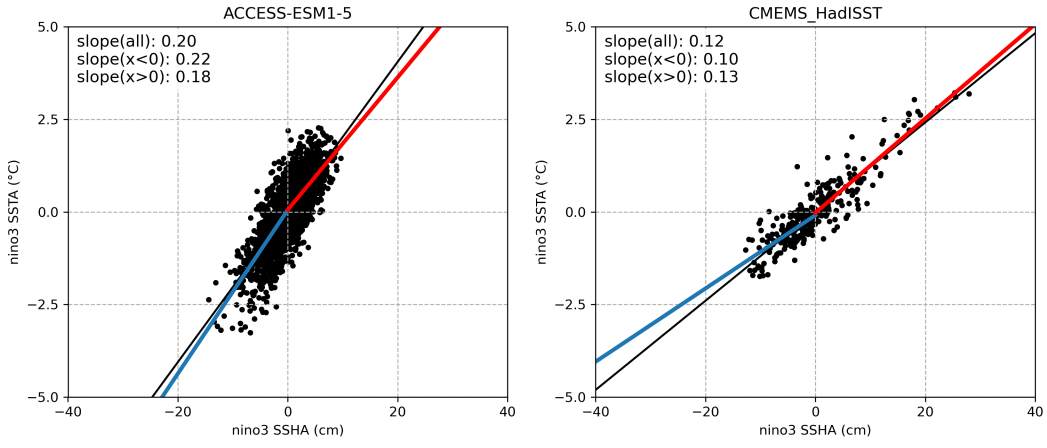


Fig. 267: Feedback level 2 plot of SSH and SST anomalies.

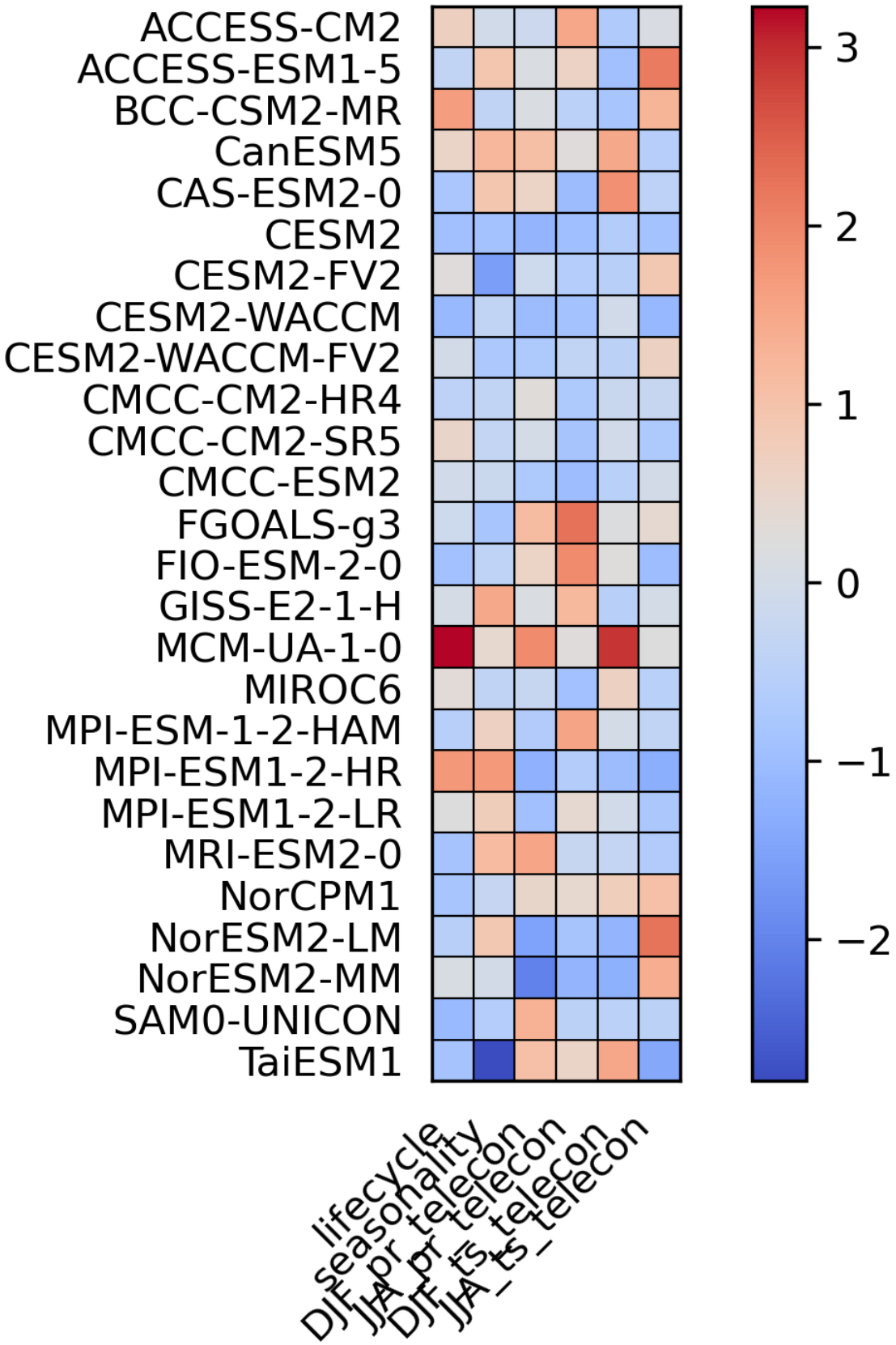


Fig. 268: Portrait plot of normalized metrics for models against reference observation.

5.7.7 Ocean diagnostics

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.

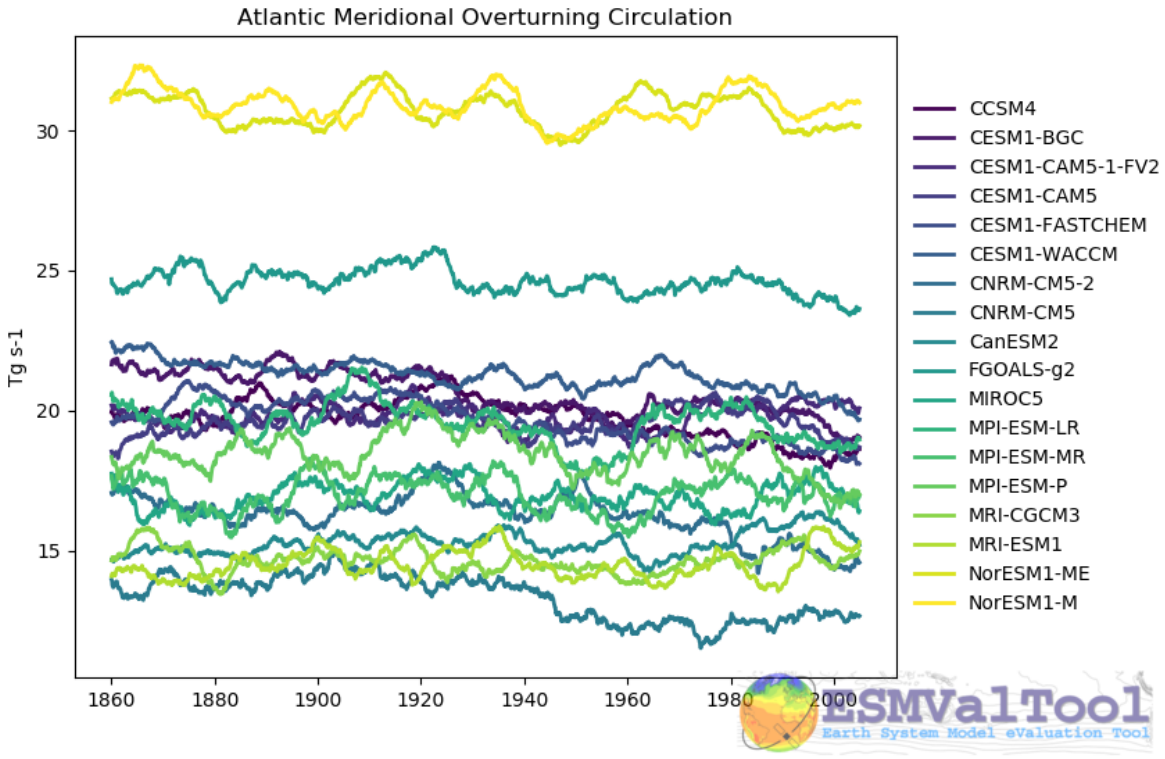
Available recipes

- 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

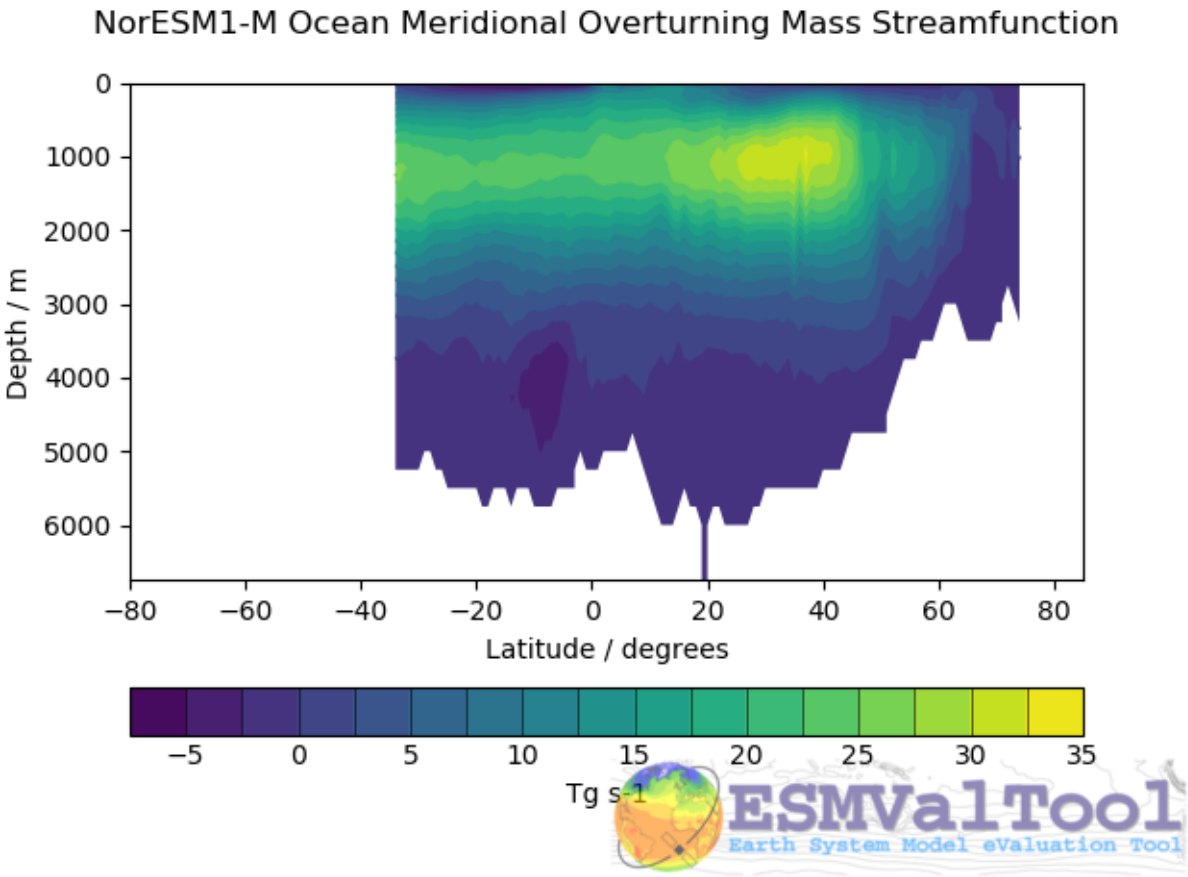
The recipe_ocean_amoc.yml is an recipe that produces figures describing the Atlantic Meridional Overturning Circulation (AMOC) and the drake passage current.

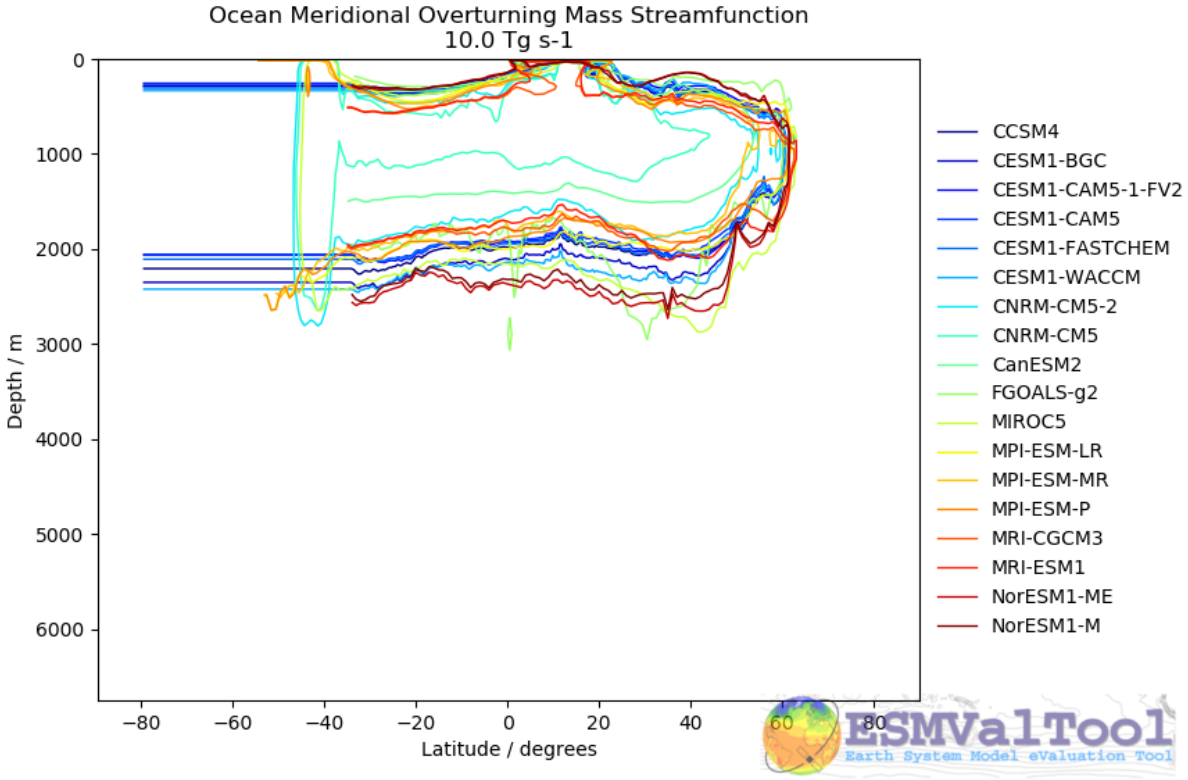
The recipes produces time series of the AMOC at 26 north and the drake passage current.



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:


NorESM1-M Ocean Meridional Overturning Mass Streamfunction
Depth / m
0
1000
2000
3000
4000
5000
6000
−80
−60
−40
−20
0
20
40
60
80
Latitude / degrees
−5
0
5
10
15
20
25
30
35
Tg s-1
ESMValTool
Earth System Model eValuation Tool



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.ncei.noaa.gov/products/world-ocean-atlas (last access 02/08/2021)

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 figures 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 within each figure are:

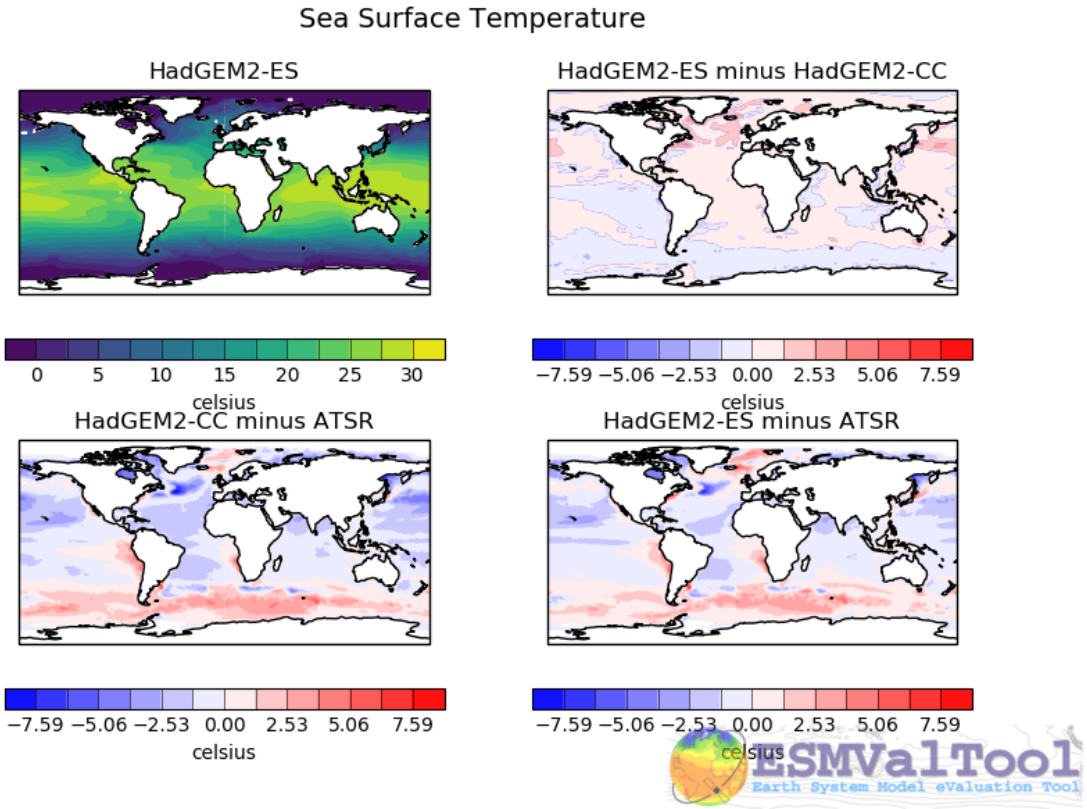
model1	model 1 minus model2
model2 minus obs	model1 minus obs

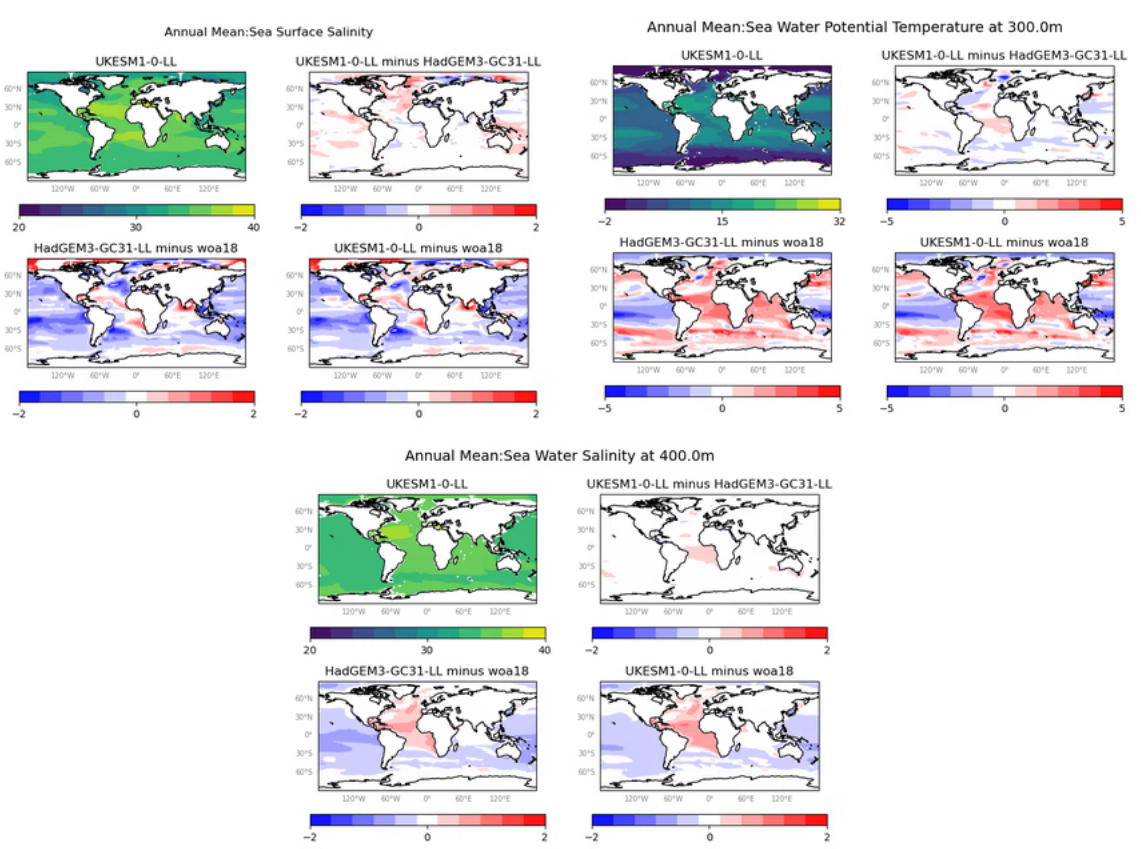
These figures are also known as Model vs Model vs Obs plots.

The figures produced by this recipe are intended to replicate Marine Assess plots. Plots produced are:

- Sea Surface Temperature
- Sea Surface Salinity
- Sea Water Temperature (at various depths)
- Sea Water Salinity (at various depths)

The figures produced by this recipe compare a HadGEM3 model, UKESM1 model and a WOA observational dataset:





These kind of figures can be very useful when developing a model, as they allow model developers to quickly see the impact of any 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

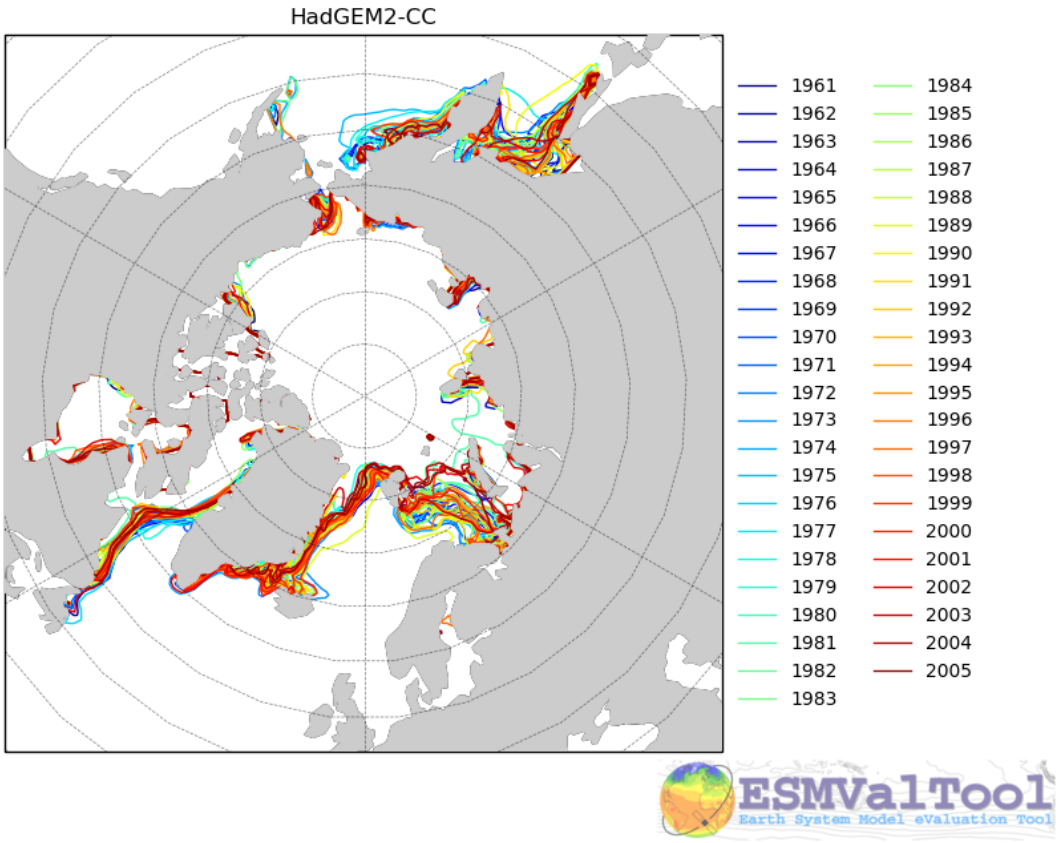
- Regions: Northern or Southern Hemisphere
- Seasons: December-January-February or June-July-August

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.

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
```

An appropriate preprocessor for a 3D field would be:

```
prep_quad_map_depth:
  climate_statistics:
    operator: mean
  extract_levels:
    levels: [100., 200., 300., 400., 500., 1000]
    scheme: linear
```

and an example of an appropriate diagnostic section of the recipe would be:

```
sea_surface_temperature:
    description: Global surface quad plots tos
    variables:
      tos:  # Temperature ocean surface
        preprocessor: prep_quad_map
        mip: Omon
    additional_datasets:
      - {dataset: WOA, project: obs4MIPs, level: L3,
         start_year: 2000, end_year: 2000, tier: 2}
    scripts:
      Global_Ocean_map:  &Global_Ocean_map
        script: ocean/diagnostic_maps_quad.py
        control_model: {dataset: HadGEM3-GC31-LL, project: CMIP6, mip: Omon,
                        exp: historical, ensemble: r1i1p1f3}
        exper_model: {dataset: UKESM1-0-LL, project: CMIP6, mip: Omon,
                      exp: historical, ensemble: r1i1p1f2}
        observational_dataset: {dataset: WOA, 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 minimum 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 necessary to define the overall layout through 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 minimum 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.
```

(continues on next page)

(continued from previous page)

```
    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
```

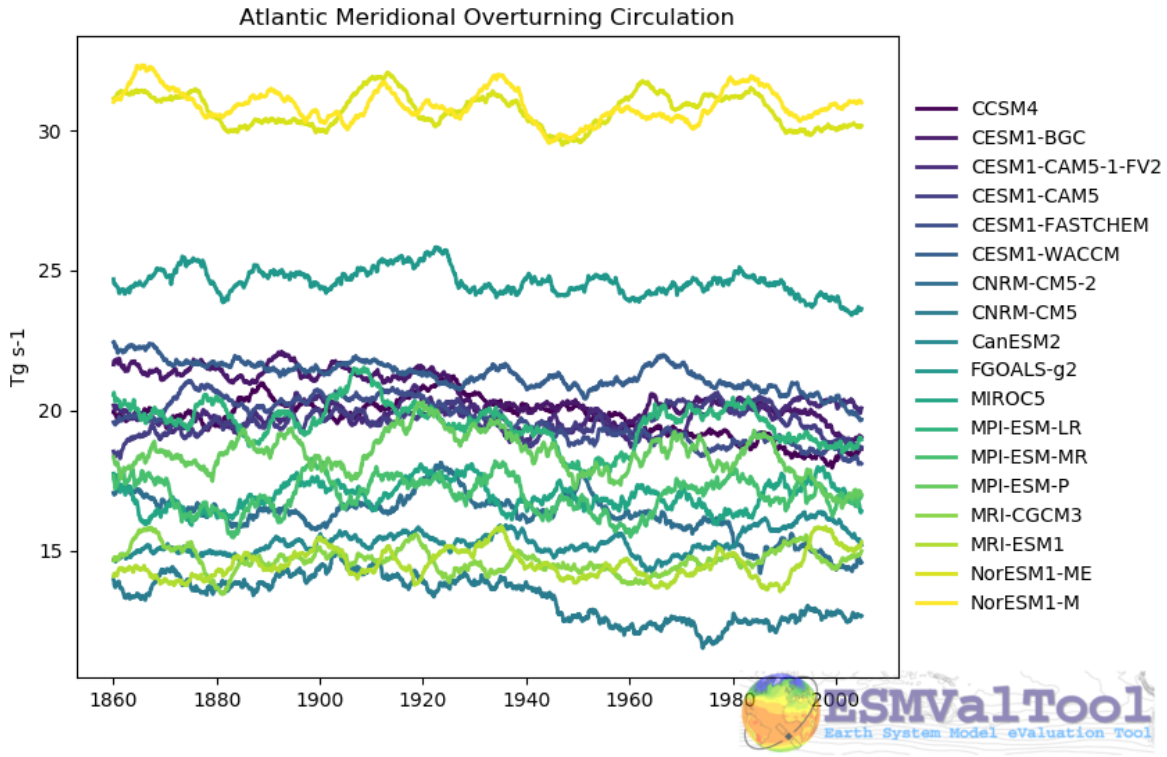
For a global volume-weighted average 3D field:

```
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

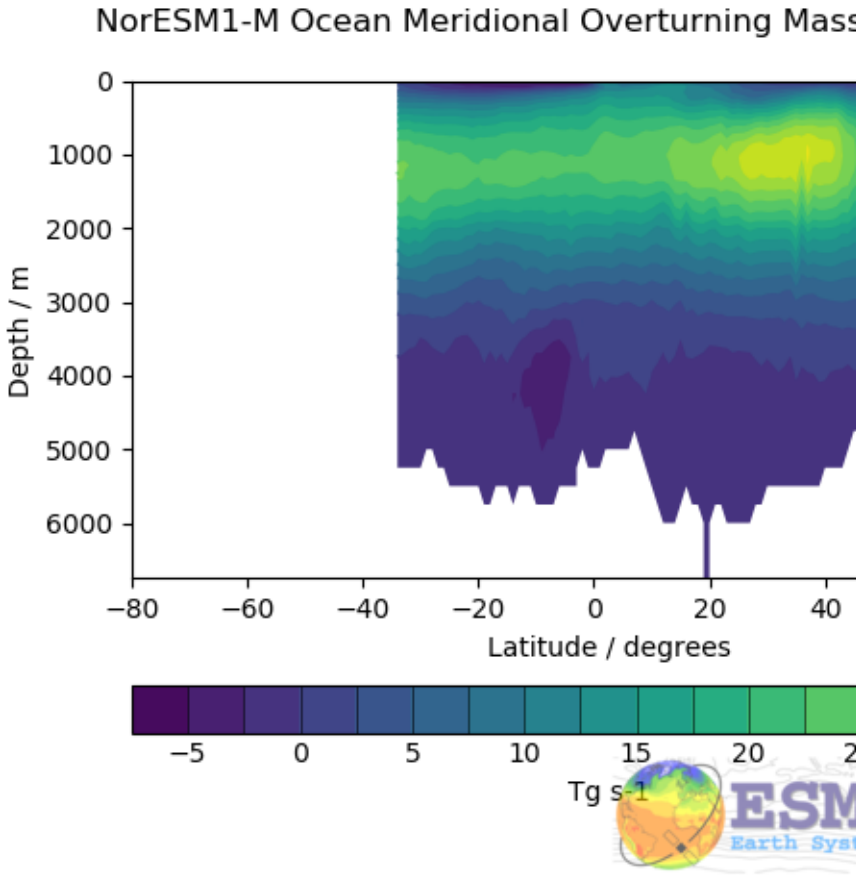
The diagnostic_transects.py diagnostic produces images of a transect, typically along a constant latitude or longitude.

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.

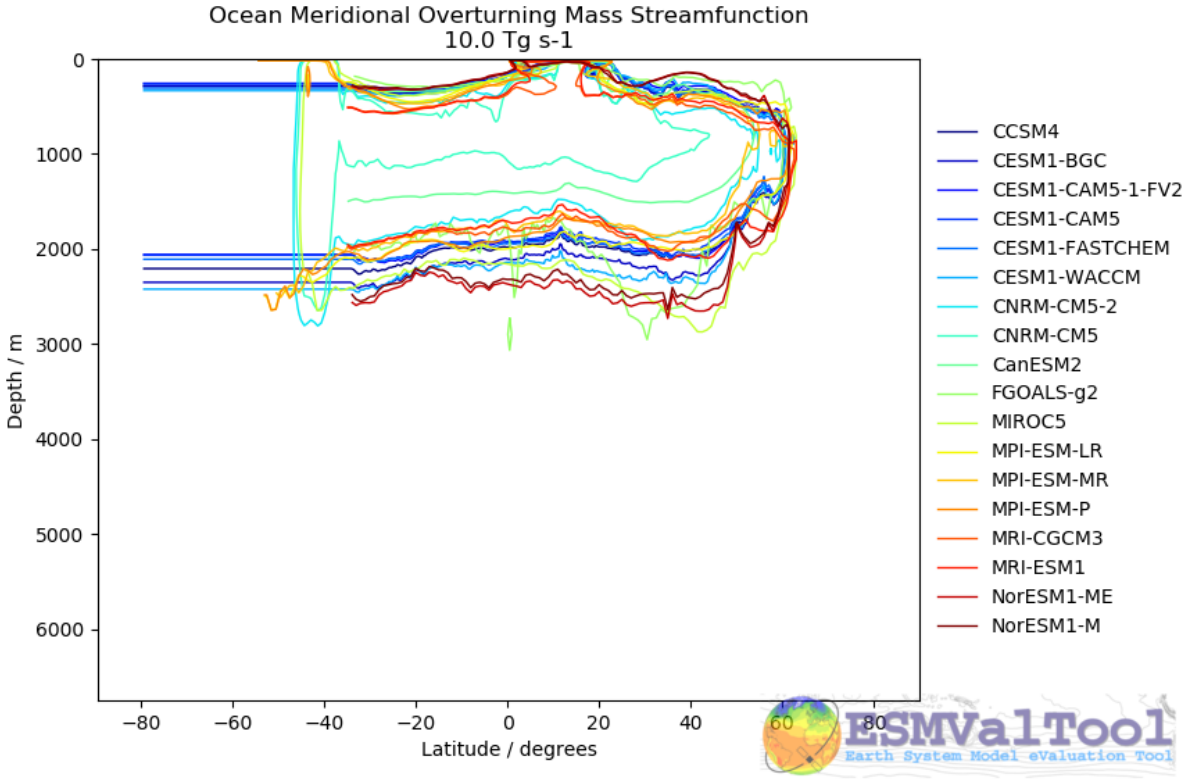
An appropriate preprocessor for a 3D+time field would be:

```
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)

(continued from previous 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: T02M
      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 configuration.
- 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.

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 configuration option `auxiliary_data_dir`.

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
```

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 *esmvaltool data format WOA* command.

Landschuetzer 2016

These data can be downloaded from: https://www.nodc.noaa.gov/archive/arc0105/0160558/3.3/data/0-data/spco2_1982-2015_MPI_SOM-FFN_v2016.nc (last access 09/20/2022)

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 *esmvaltool data format Landschuetzer2016* command.

5.7.8 Sea Surface Salinity Evaluation

Overview

This recipe compares the regional means of sea surface salinity with a reference dataset (ESACCI-SEA-SURFACE-SALINITY v1 or v2 by default). To do this, the recipe generates plots for the timeseries of each region and a radar plot showing (i) the mean state bias, and (ii) the ratio between the simulated and observed standard deviations of different regional averages of sea surface salinity, calculated in the temporal window for which observations and simulations overlap.

Preprocessor requirements:

The recipe is created in a way that should make possible (although is not tested) to use it for other variables and datasets, even for more than one at a time. The diagnostic only expects variables with dimensions *time* and *depth_id* and it does not assume any other constraint.

It is therefore mandatory to keep the *extract_shape* preprocessor for more than one region and any form of region operation (*mean*, *max*, *min* ...) to collapse the *latitude* and *longitude* coordinates. In case you want to try with variables that have extra dimensions (i.e. *depth*) you must add an extra preprocessor call to collapse them (i.e. *depth_integration*)

The recipe can be used with any shapefile. As it is, it uses the IHO Sea Areas (version 3) downloaded from https://marineregions.org/downloads.php, but any shapefile containing marine regions can be used.

Any number of regions can be choosed also, even though plots may look odd if too few or too many are selected.

Regions available on IHO Sea Areas file:

- Adriatic Sea
- Aegean Sea
- Alboran Sea
- Andaman or Burma Sea
- Arabian Sea
- Arafura Sea
- Arctic Ocean
- Baffin Bay
- Balearic (Iberian Sea)
- Bali Sea
- Baltic Sea
- Banda Sea
- Barentsz Sea
- Bass Strait
- Bay of Bengal
- Bay of Biscay
- Bay of Fundy
- Beaufort Sea
- Bering Sea
- Bismarck Sea

- Black Sea
- Bristol Channel
- Caribbean Sea
- Celebes Sea
- Celtic Sea
- Ceram Sea
- Chukchi Sea
- Coral Sea
- Davis Strait
- East Siberian Sea
- Eastern China Sea
- English Channel
- Flores Sea
- Great Australian Bight
- Greenland Sea
- Gulf of Aden
- Gulf of Alaska
- Gulf of Aqaba
- Gulf of Boni
- Gulf of Bothnia
- Gulf of California
- Gulf of Finland
- Gulf of Guinea
- Gulf of Mexico
- Gulf of Oman
- Gulf of Riga
- Gulf of St. Lawrence
- Gulf of Suez
- Gulf of Thailand
- Gulf of Tomini
- Halmahera Sea
- Hudson Bay
- Hudson Strait
- Indian Ocean
- Inner Seas off the West Coast of Scotland
- Ionian Sea

- Irish Sea and St. George's Channel
- Japan Sea
- Java Sea
- Kara Sea
- Kattegat
- Labrador Sea
- Laccadive Sea
- Laptev Sea
- Ligurian Sea
- Lincoln Sea
- Makassar Strait
- Malacca Strait
- Mediterranean Sea - Eastern Basin
- Mediterranean Sea - Western Basin
- Molukka Sea
- Mozambique Channel
- North Atlantic Ocean
- North Pacific Ocean
- North Sea
- Norwegian Sea
- Persian Gulf
- Philippine Sea
- Red Sea
- Rio de La Plata
- Savu Sea
- Sea of Azov
- Sea of Marmara
- Sea of Okhotsk
- Seto Naikai or Inland Sea
- Singapore Strait
- Skagerrak
- Solomon Sea
- South Atlantic Ocean
- South China Sea
- South Pacific Ocean
- Southern Ocean

- Strait of Gibraltar
- Sulu Sea
- Tasman Sea
- The Coastal Waters of Southeast Alaska and British Columbia
- The Northwestern Passages
- Timor Sea
- Tyrrhenian Sea
- White Sea
- Yellow Sea

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_sea_surface_salinity.yml

Diagnostics are stored in diag_scripts/sea_surface_salinity/

- compare_salinity.py: plot timeseries for each region and generate radar plot.

User settings in recipe

1. compare_salinity.py

   Required settings for script

   none

   Optional settings for script

   none

   Required settings for variables

   - ref_model: name of reference data set

   Optional settings for variables

   none

Variables

- sos (ocean, monthly, time latitude longitude)

Observations and reformat scripts

- ESACCI-SEA-SURFACE-SALINITY (sos)

References

- Diagnostic: please contact authors
- ESACCI-SEA-SURFACE-SALINITY dataset: Boutin, J., J.-L. Vergely, J. Koehler, F. Rouffi, N. Reul: ESA Sea Surface Salinity Climate Change Initiative (Sea_Surface_Salinity_cci): Version 1.8 data collection. Centre for Environmental Data Analysis, 25 November 2019. doi: 10.5285/9ef0ebf847564c2eabe62cac4899ec41. http://dx.doi.org/10.5285/9ef0ebf847564c2eabe62cac4899ec41

Example plots

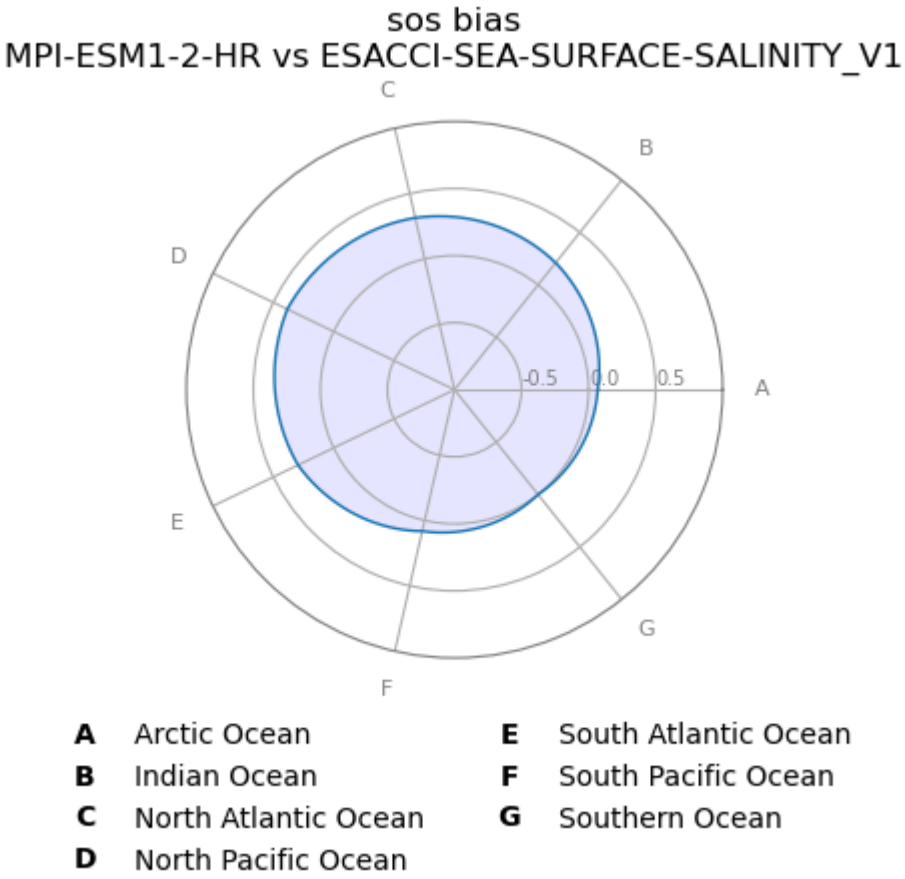


Fig. 269: Radar plot showing the mean state biases (simulation minus observations) for the regional averages of sea surface salinity in the selected ocean basins and seas.

5.7.9 Ocean metrics

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.

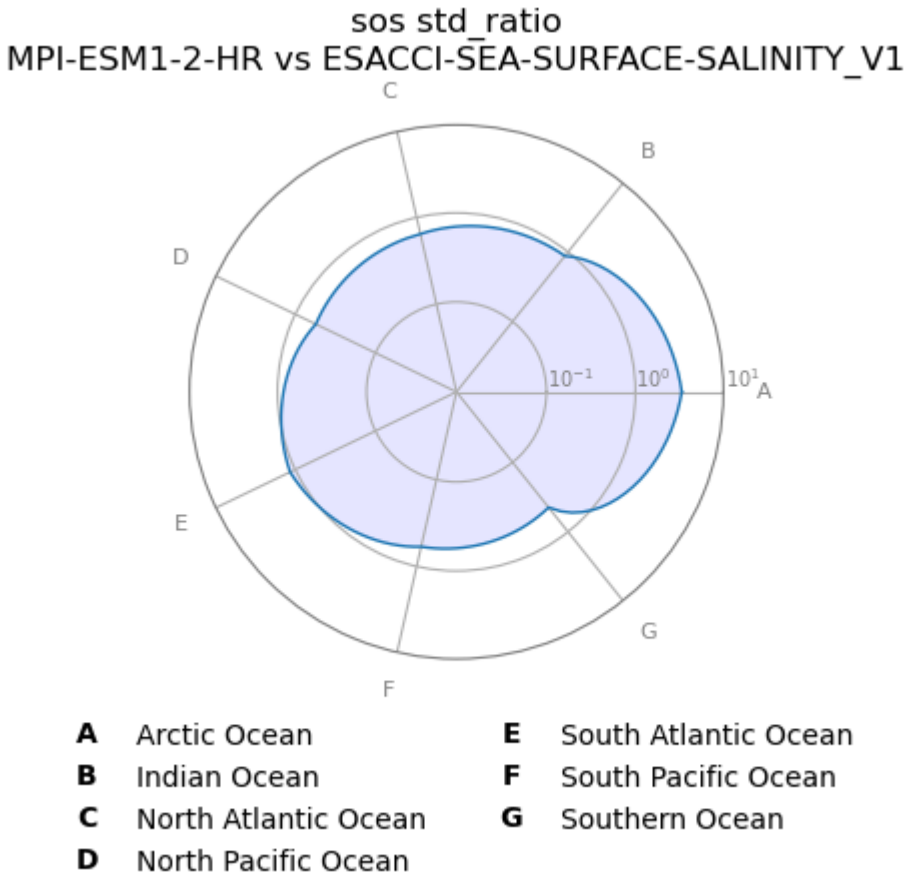


Fig. 270: Radar plot showing the ratio between the simulated and observed standard deviations of the regional averages of sea surface salinity in the selected ocean basins and seas.

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.

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

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)

Observations and reformat scripts

Note: WOA data has not been tested with reciepe_russell18jgr.yml and corresponding diagnostic scripts.

- WOA (thetao, so - esmvaltool/cmorizers/data/formatters/datasets/woa.py)

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

Example plots

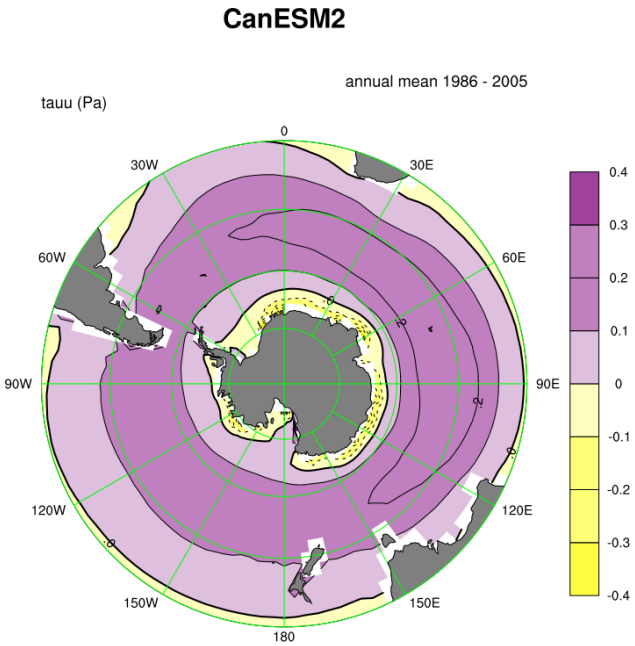


Fig. 271: Figure 1: Annual-mean zonal wind stress (tauu - N/m^2) with eastward wind stress as positive plotted as a polar contour map.

5.8 Other

5.8.1 Capacity factor of wind power: Ratio of average estimated power to theoretical maximum power

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.

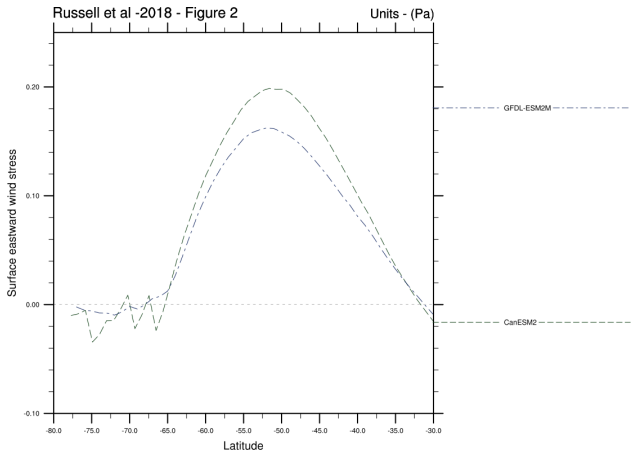


Fig. 272: Figure 2: The zonal and annual means of the zonal wind stress (N/m^2) plotted in a line plot.

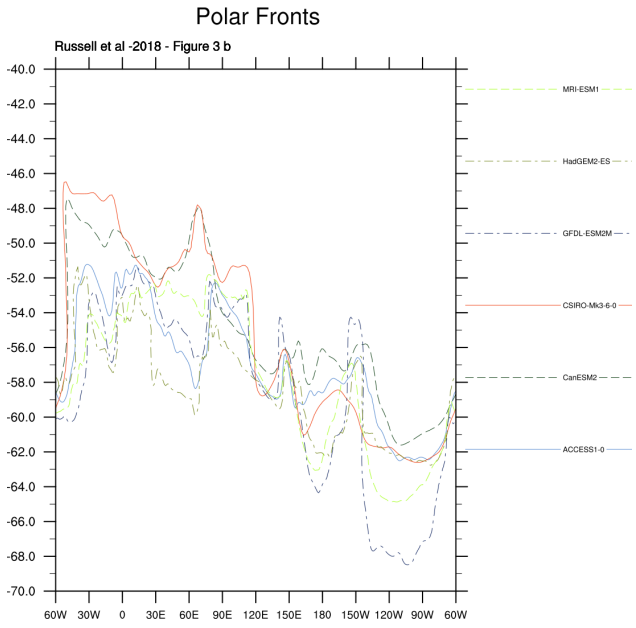


Fig. 273: Figure 3a: The latitudinal position of Subantarctic Front using definitions from Orsi et al (1995).

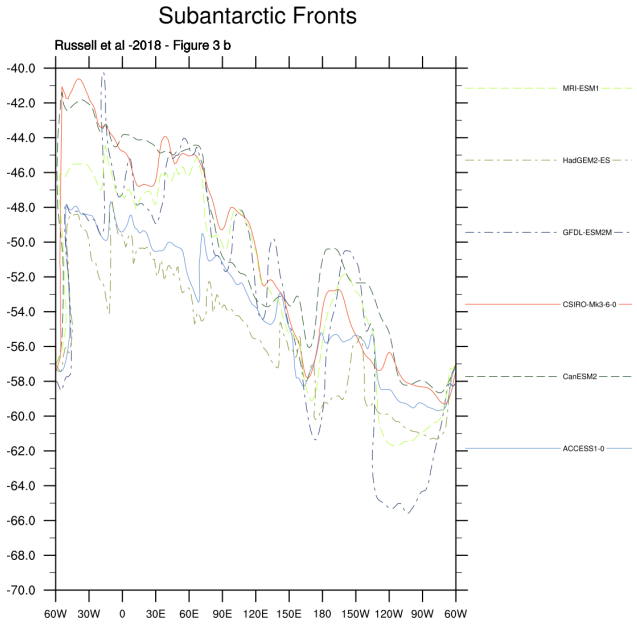


Fig. 274: Figure 3b: The latitudinal position of Polar Front using definitions from Orsi et al (1995).

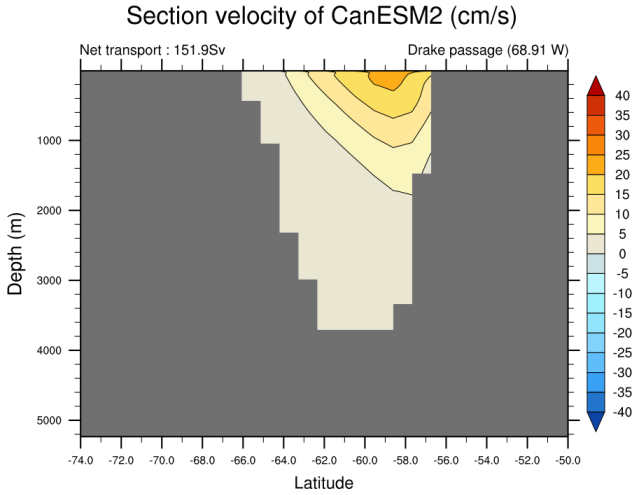


Fig. 275: Figure 4: Time averaged zonal velocity through Drake Passage (at 69W, in cm/s, eastward is positive). The total transport by the ACC is calculated if volcello file is available.

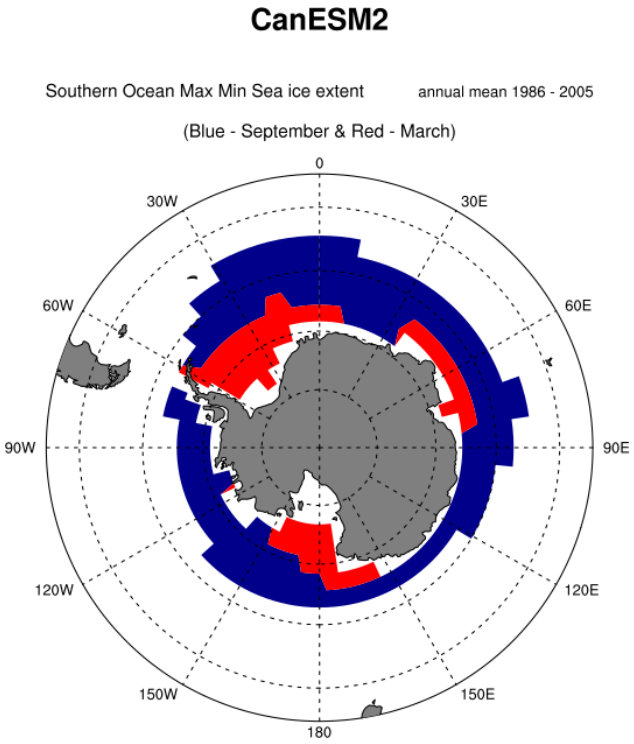


Fig. 276: Figure 5: Mean extent of sea ice for September(max) in blue and February(min) in red plotted as polar contour map.

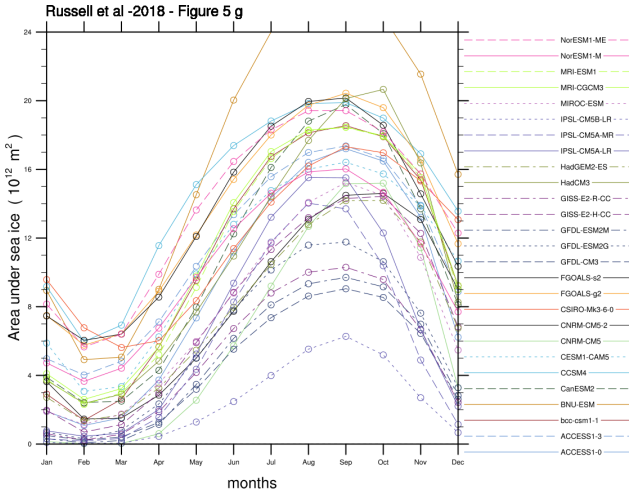


Fig. 277: Figure 5g: Annual cycle of sea ice area in southern ocean as a line plot (monthly climatology).

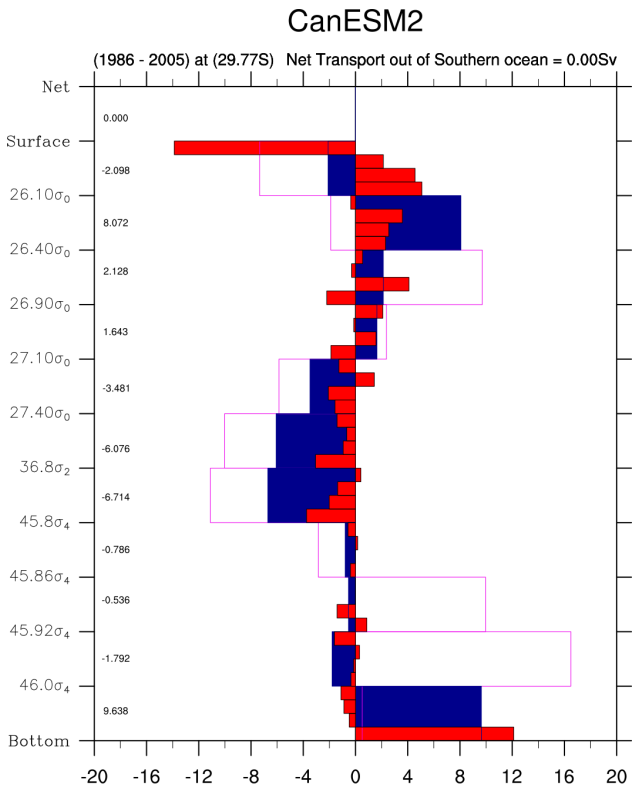


Fig. 278: Figure 6a: Density layer based volume transport (in Sv) across 30S based on the layer definitions in Talley (2008).

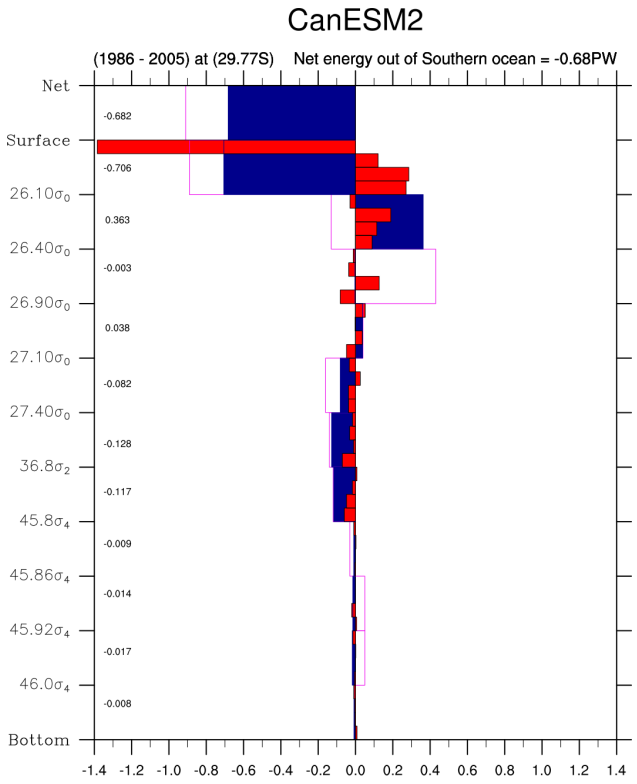


Fig. 279: Figure 6b: Density layer based heat transport(in PW) across 30S based on the layer definitions in Talley (2008).

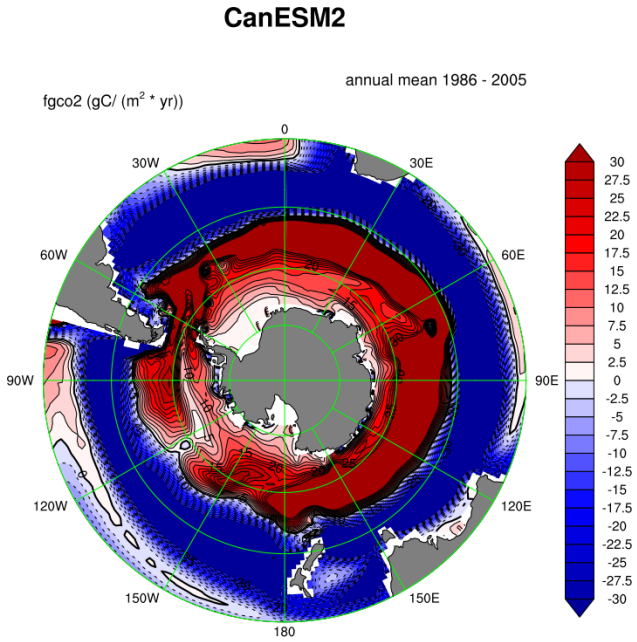


Fig. 280: Figure 7: Annual mean CO_2 flux (sea to air, gC/(yr * m^2), positive (red) is out of the ocean) as a polar contour map.

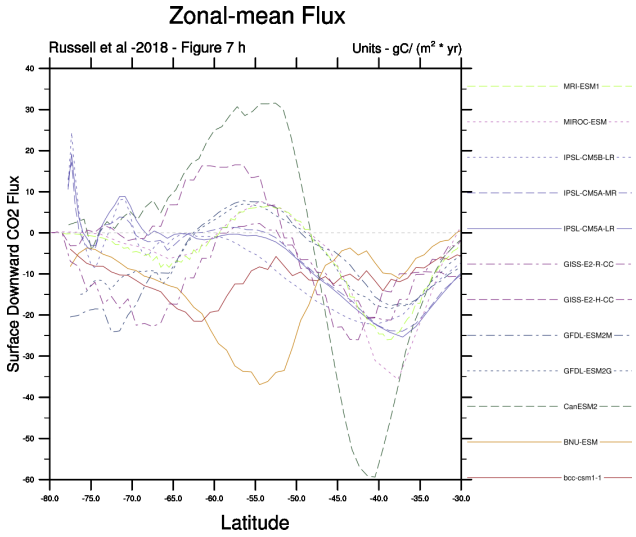


Fig. 281: Figure 7h: the time and zonal mean flux of CO_2 in gC/(yr * m^2) plotted as a line plot.

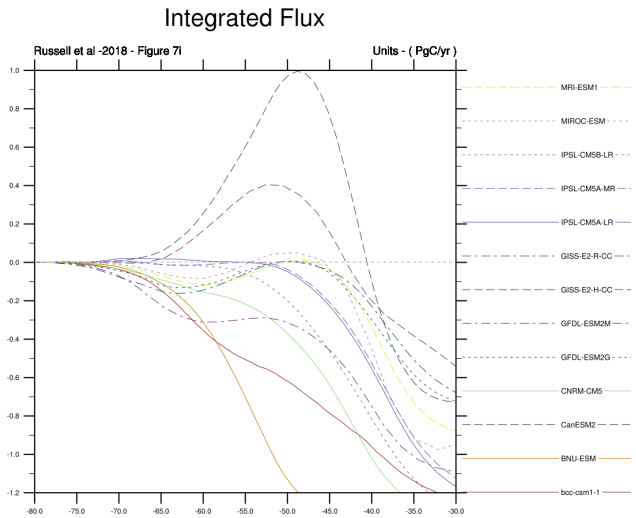


Fig. 282: Figure 7i is the cumulative integral of the net CO_2 flux from 90S to 30S (in PgC/yr) plotted as a line plot.

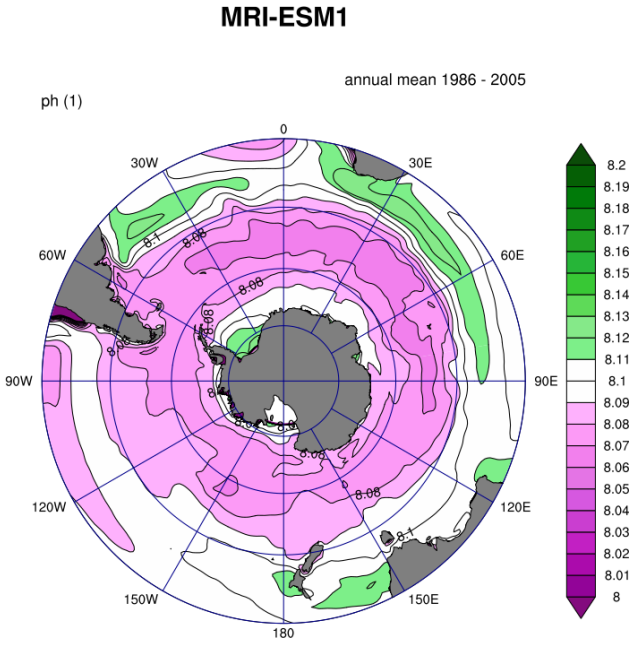


Fig. 283: Figure 8: Annual-mean surface pH plotted as a polar contour map.

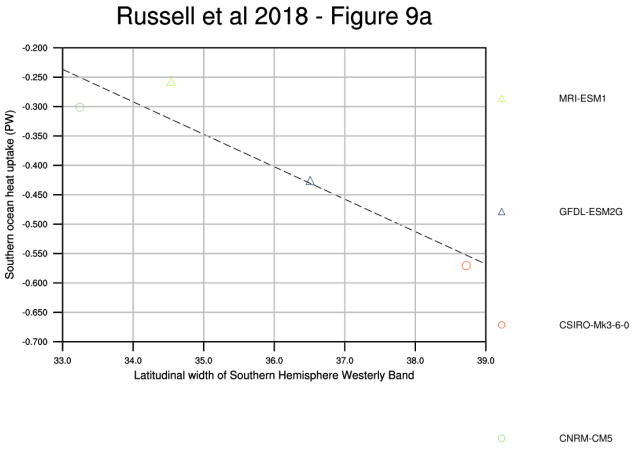


Fig. 284: Figure 9a: Scatter plot of the width of the Southern Hemisphere westerly wind band (in degrees of latitude) against the annual-mean integrated heat uptake south of 30S (in PW—negative uptake is heat lost from the ocean) along with the best fit line.

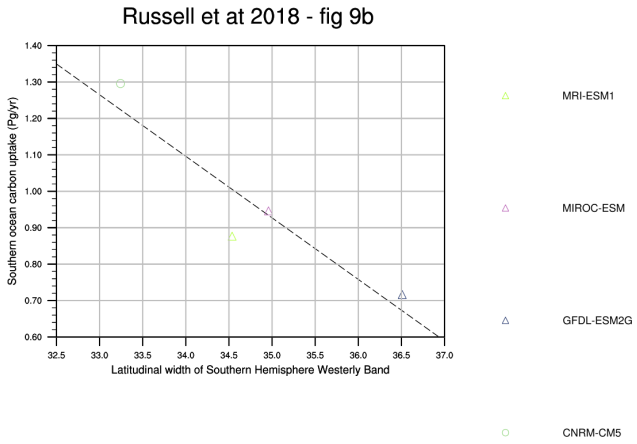


Fig. 285: Figure 9b: Scatter plot of the width of the Southern Hemisphere westerly wind band (in degrees of latitude) against the annual-mean integrated carbon uptake south of 30S (in Pg C/yr), along with the best fit line.

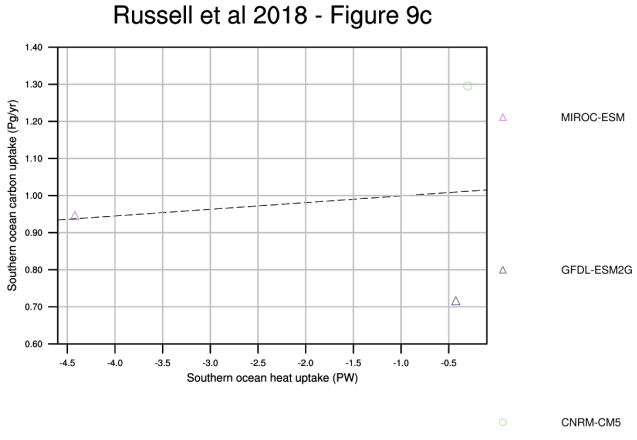


Fig. 286: Figure 9c: Scatter plot of the net heat uptake south of 30S (in PW) against the annual-mean integrated carbon uptake south of 30S (in Pg C/yr), along with the best fit line.

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.

User settings

User setting files are stored in recipes/

1. recipe_capacity_factor.yml

   Required settings for script

   - power_curves: (should not be changed)

Variables

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

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

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]

Example plots

5.8.2 CMORizer recipes

Overview

These are CMORizer recipes calling CMORizer diagnostic scripts.

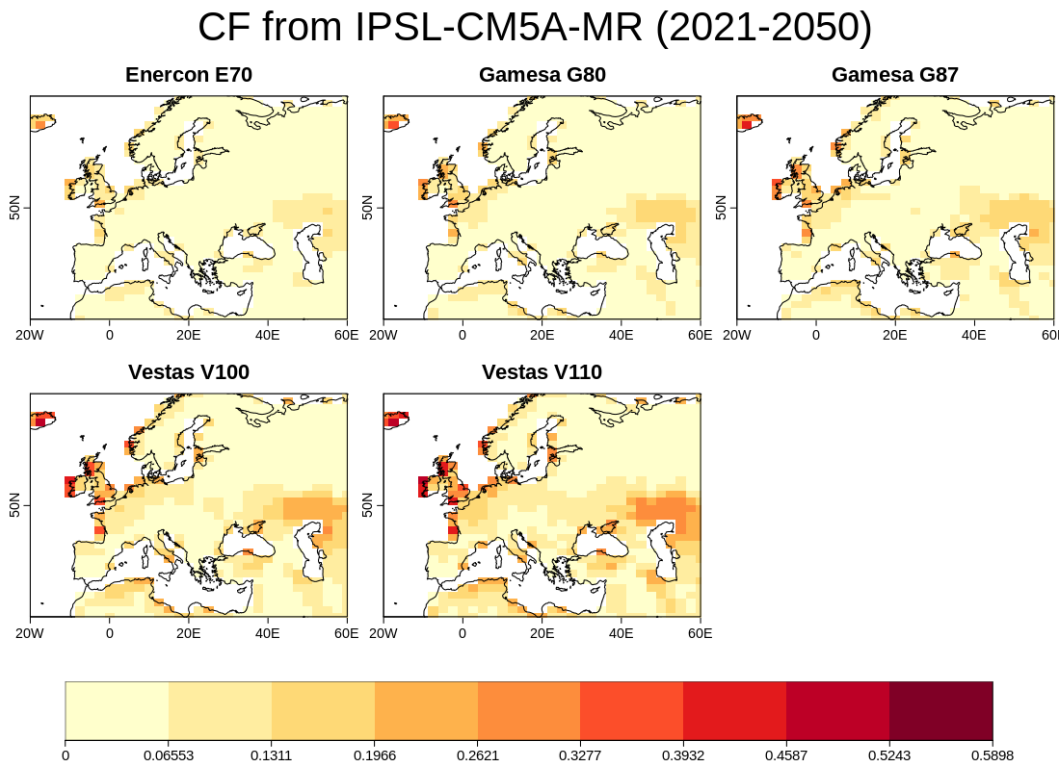


Fig. 287: Wind capacity factor for five turbines: Enercon E70 (top-left), Gamesa G80 (middle-top), Gamesa G87 (top-right), Vestas V100 (bottom-left) and Vestas V110 (middle-bottom) using the IPSL-CM5A-MR simulations for the r1p1i1 ensemble for the rcp8.5 scenario during the period 2021-2050.

ESMValCore supports ERA5 hourly and monthly datasets in their native format, see *Datasets in native format*. and ERA5 data documentation. It may be useful in some cases to create ERA5 daily CMORized data. This can be achieved by using a CMORizer *recipe*, see recipe_daily_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 files to the standard OBS6 file names. The output are thus daily, CMORized ERA5 data, that can be used through the OBS6 project. As such, this example recipe creates 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 run cmorizers/recipe_daily_era5.yml
```

Note that the `recipe_daily_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 the time axis of non-instantaneous variables half an hour back in time, resulting in a missing record on the last day of the year. ERA5 data can be downloaded using era5cli.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- cmorizers/recipe_daily_era5.yml

Diagnostics are stored in esmvaltool/diag_scripts/

- cmorizers/era5.py: generates output filename

User settings in recipe

1. cmorizers/recipe_daily_era5.yml

   Required add_one_day preprocessor settings:

   - start_year: 1990
   - start_month: 1
   - start_day: 1
   - end_year: 1991
   - end_month: 1
   - end_day: 1

These settings should not be changed

- **daily_mean:**
  operator: mean
- **daily_min:**
  operator: min
- **daily_max:**
  operator: max

Variables

1. cmorizers/recipe_daily_era5.yml
   - clt
   - evspsbl
   - evspsblpot

- mrro
- pr
- prsn
- ps
- psl
- rlds
- rls
- rsds
- rsdt
- rss
- tas
- tasmax
- tasmin
- tdps
- ts
- tsn
- uas
- vas

References

- Hersbach, H., et al., Quarterly Journal of the Royal Meteorological Society, 730, 1999-2049, doi:10.1002/qj.3803, 2020.

5.8.3 Ensemble Clustering - a cluster analysis tool for climate model simulations (EnsClus)

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.

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).

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

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

Variables

- chosen by user (e.g., precipitation as in the example)

Observations and reformat scripts

None.

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

Example plots

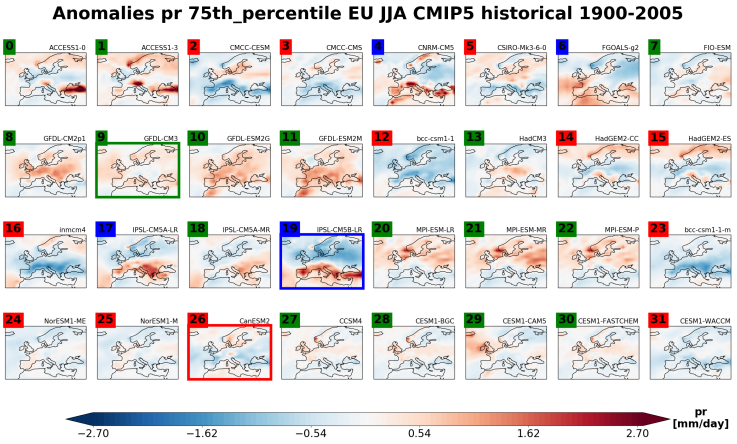


Fig. 288: Clustering based on the 75th percentile of historical summer (JJA) precipitation rate for CMIP5 models over 1900-2005. 3 clusters are computed, based on the principal components explaining 80% of the variance. The 32 models are grouped in three different clusters. The green cluster is the most 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.

5.8.4 ESA CCI LST comparison to Historical Models

Overview

This diagnostic compares ESA CCI LST to multiple historical emsemble members of CMIP models. It does this over a defined region for monthly values of the land surface temperature. The result is a plot showing the mean differnce of CCI LST to model average LST, with a region of +/- one standard deviation of the model mean LST given as a measure of model variability.

The recipe and diagnostic need the all time average monthly LST from the CCI data. We use the L3C single sensor monthy data. A CMORizing script calculates the mean of the day time, and night time overpasses to give the all time average LST. This is so that the Amon output from CMIP models can be used. We created such a dataset from the Aqua MODIS data from CCI.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- `recipe_esacci_lst.yml`

Diagnostics are stored in esmvaltool/diag_scripts/lst/

- `lst.py`

User settings in recipe

1. Script `recipe_esacci_lst.yml`

   No required settings for script

   No user defined inputs to the diagnostic

   Required settings for variables

   - The diagnostic works with all data sources on having the same start_year and end_year, and hence that data is also available.

   Required settings for preprocessor

   - start_longitude, end_longitude The western and eastern bounds of the region to work with.
   - start_latitude, end_latitude The southern and northern bounds of the region to work with.
   - target_grid This should be one of the model grids.

Variables

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

Observations and reformat scripts

This recipe and diagnostic is written to work with data created from the CMORizer esmvaltool/cmorizers/obs/cmorize_obs_esacci_lst.py. This takes the orginal ESA CCI LST files for the L3C data from Aqua MODIS DAY and NIGHT files and creates a the all time mean data this diagnostic uses. Advice from the CCI LST team is to use the monthly not daily files to create the all time average to avoid th epossibility of biasing towards night time LST values being more prevalent because of how the cloud screening algorithms work.

References

- ESA CCI LST project https://climate.esa.int/en/projects/land-surface-temperature/

Example plots

5.8.5 Example recipes

Overview

These are example recipes calling example diagnostic scripts.

The recipe `examples/recipe_python.yml` produces time series plots of global mean temperature and for the temperature in Amsterdam. It also produces a map of global temperature in January 2020.

The recipe `examples/recipe_easy_ipcc.yml` reproduces part of figure 9.3a from IPCC AR6 - Climate Change 2021: The Physical Science Basis. It demonstrates how ESMValTool can be used to conveniently analyze many models on their native grid and is described in detail in the blog post Analysis-ready climate data with ESMValCore.

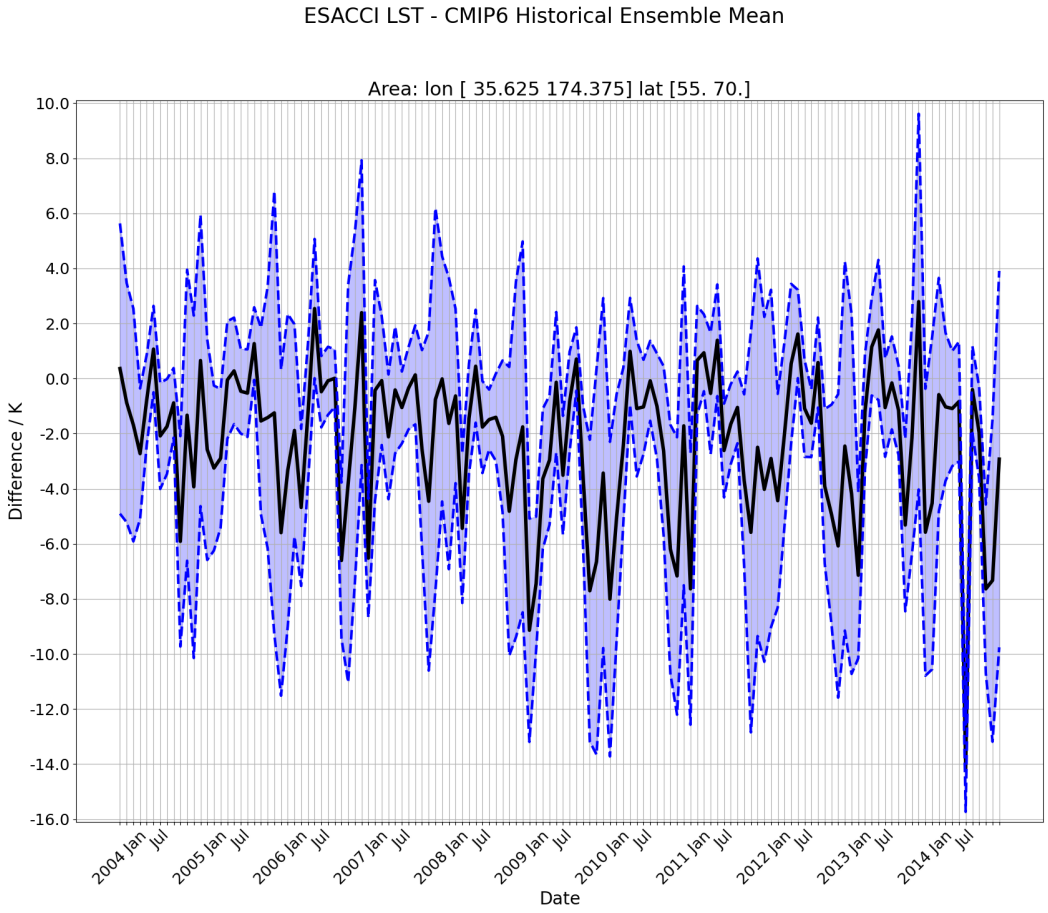


Fig. 289: Timeseries of the ESA CCI LST minus mean of CMIP6 ensembles. The selected region is 35E-175E, 55N-70N. The black line is the mean difference, and the blue shaded area denotes one standard deviation either way of the individual ensemble member's differecen in LST. Models used for this are UKESM1 members r1i1p1f2 and r2i1p1f2, and CESM members r2i1p1f1 and r3i1p1f1. We have used the entire timeseries of available CCI data 2004-2014 inclusive, noting we have not written the CMORizer to process the incomplete year of 2003 for the Aqua MODIS data.

The recipe `examples/recipe_extract_shape.yml` produces a map of the mean temperature in the Elbe catchment over the years 2000 to 2002. Some example shapefiles for use with this recipe are available here, make sure to download all files with the same name but different extensions.

The recipe `examples/recipe_decadal.yml` showcases how the `timerange` tag can be used to load datasets belonging to the DCPP activity. Produces timeseries plots comparing the global mean temperature of a DCPP dataset with an observational dataset.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/:

- examples/recipe_python.yml
- examples/recipe_easy_ipcc.yml
- examples/recipe_extract_shape.yml
- examples/recipe_decadal.yml

Diagnostics are stored in esmvaltool/diag_scripts/:

- examples/diagnostic.py: visualize results and store provenance information
- examples/make_plot.py: Create a timeseries plot with likely ranges
- examples/decadal_example.py: visualize results and store provenance information

User settings in recipe

1. Script `examples/diagnostic.py`

   Required settings for script

   - `quickplot: plot_type`: which of the `iris.quickplot` functions to use. Arguments that are accepted by these functions can also be specified here, e.g. `cmap`. Preprocessors need to be configured such that the resulting data matches the plot type, e.g. a timeseries or a map.

Variables

- tas (atmos, monthly, longitude, latitude, time)
- tos (ocean, monthly, longitude, latitude, time)

Example plots

5.8.6 Timeseries for Arctic-Midlatitude Teleconnections

Overview

The recipe produces the timeseries of selected variables to study Arctic-midlatitude teleconnections for further application of Causal Model Evaluation (CME) described in Galytska et al. (2023).

The output of the recipe consists of the .nc files named after the data source (e.g. ERA5, ACCESS-CM2.nc etc.). Each file contains the area-weighted spatial average of climatological monthly anomalies of selected variables. The recipe also applies the CVDP package.

Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_galytska23jgr.yml

Diagnostics are stored in esmvaltool/diag_scripts/

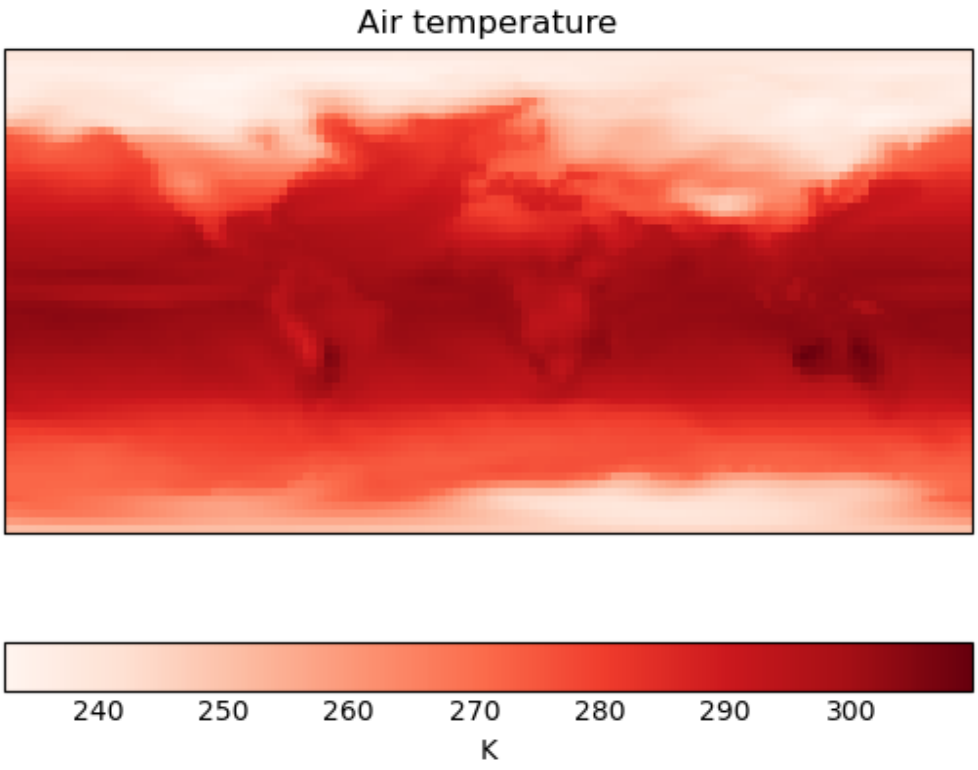


Fig. 290: Air temperature in January 2000 (BCC-ESM1 CMIP6).

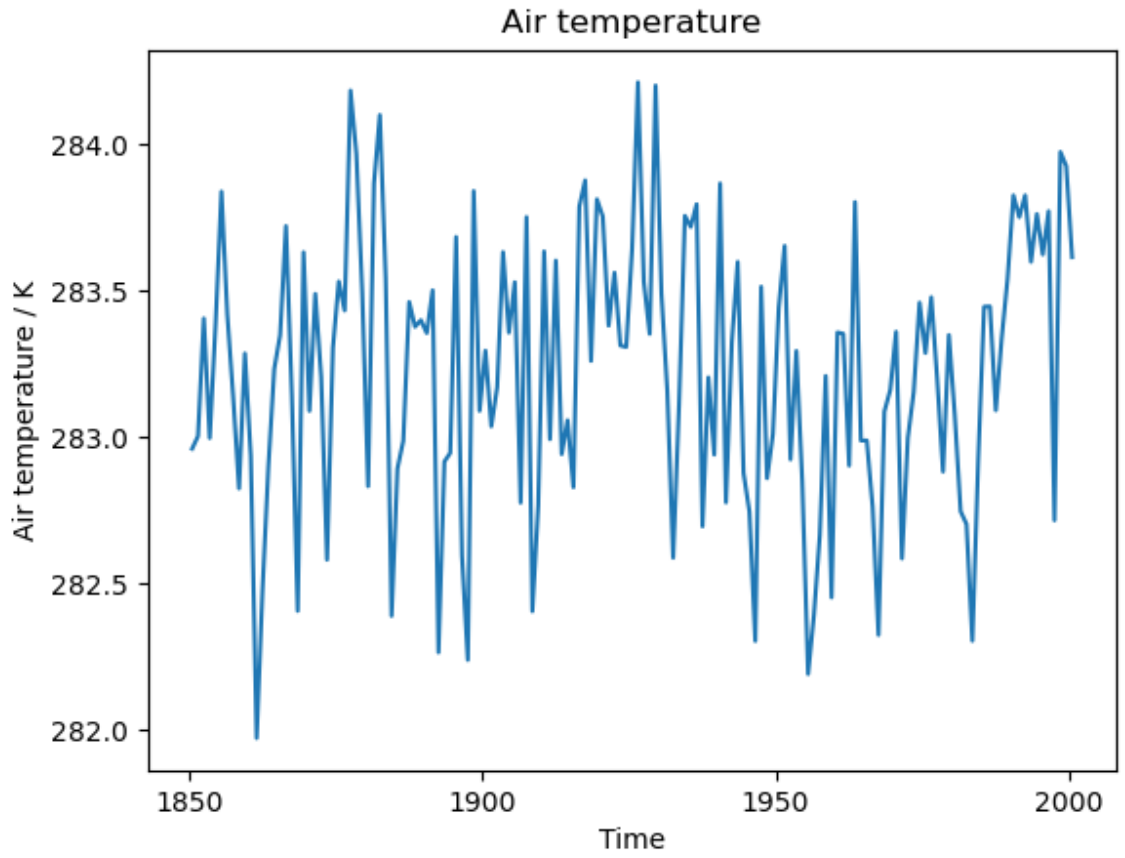


Fig. 291: Amsterdam air temperature (multimodel mean of CMIP5 CanESM2 and CMIP6 BCC-ESM1).

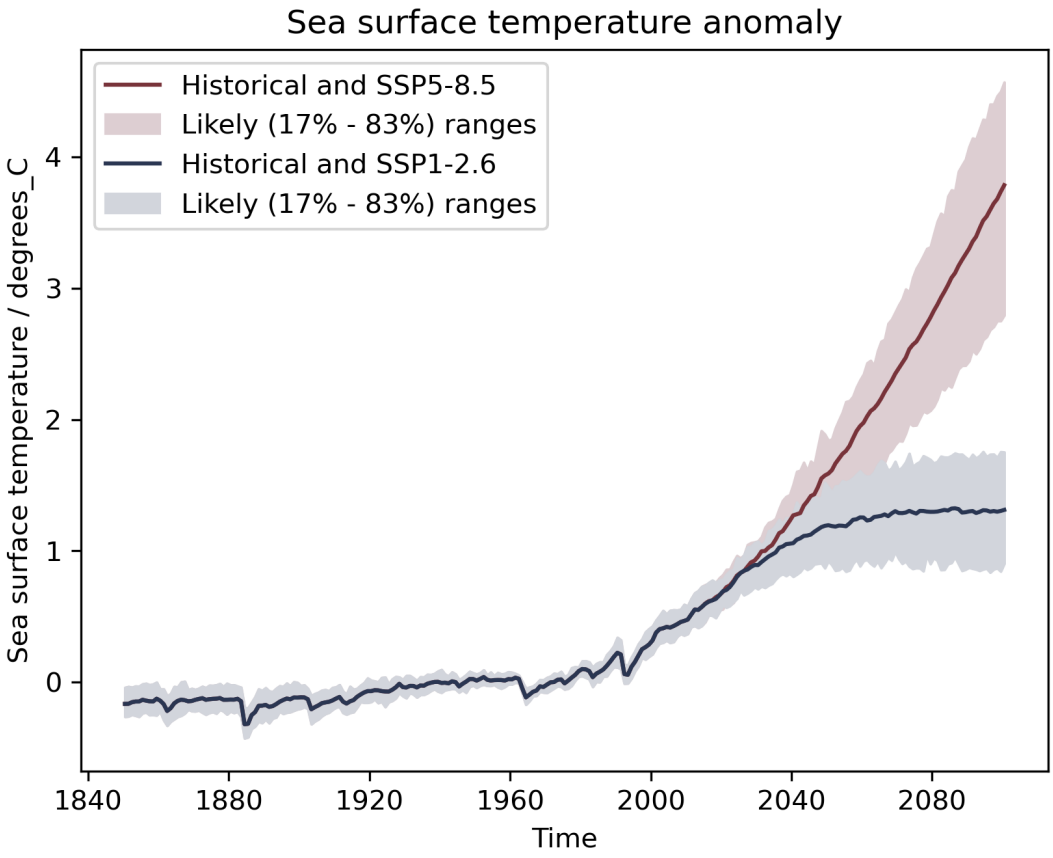


Fig. 292: Mean sea surface temperature anomaly (part of figure 9.3a from IPCC AR6).

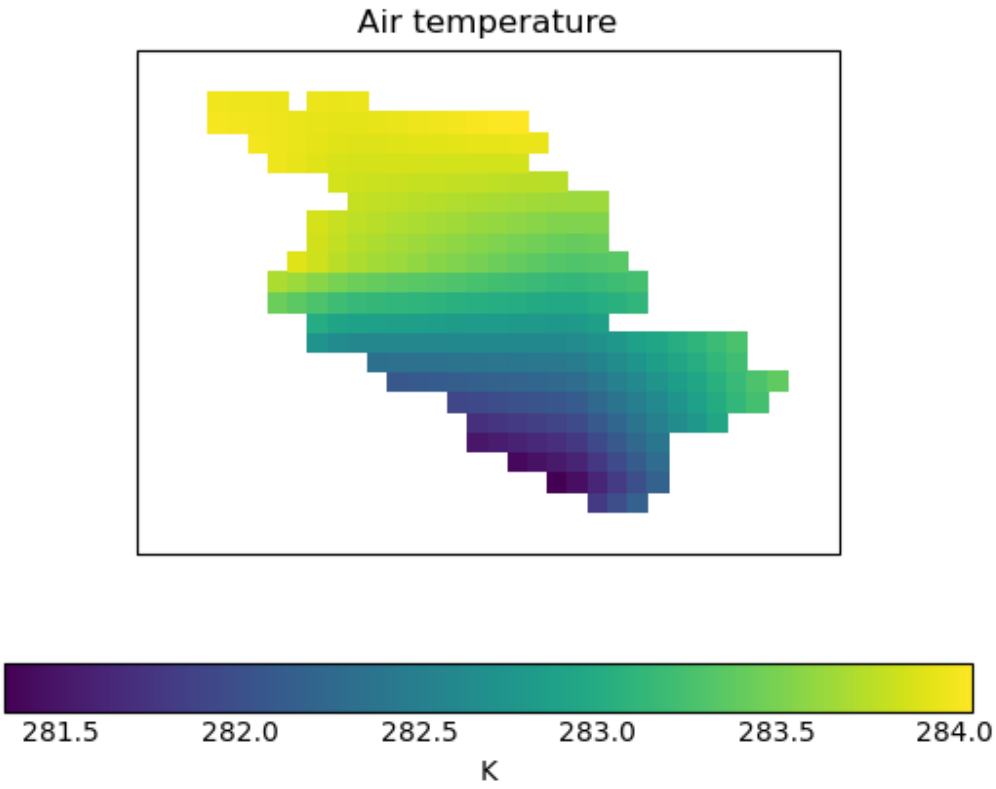


Fig. 293: Mean air temperature over the Elbe catchment during 2000-2002 according to CMIP5 CanESM2.

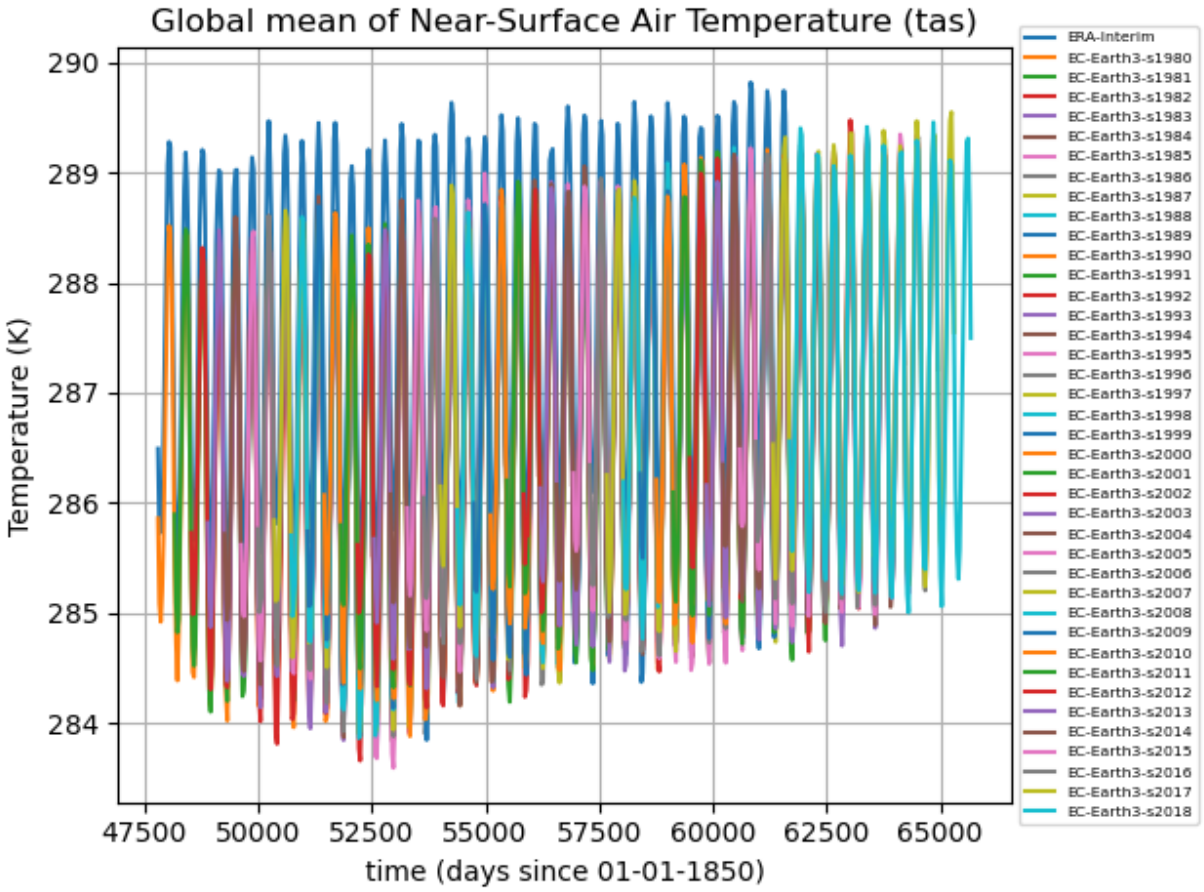


Fig. 294: Global mean temperature of CMIP6 dcppA-hindcast EC-Earth3 and OBS ERA-Interim.

- galytska23/select_variables_for_tigramite.py
- cvdp/cvdp_wrapper.py

User settings in recipe

1. Preprocessor
   - `anomalies` (*period: monthly*): Calculate anomalies for selected variables
   - `regrid` (*target_grid: 1x1*): Linear regridding of all datasets to the uniform grid
   - `area_statistics` (*operation: mean*): Calculate mean over defined regions
2. Script <select_variables_for_tigramite.py>

   none
3. Script <cvdp_wrapper.py>

   none
4. Script <cvdp/cvdp/driver.ncl>
   - `modular` (*modular: True*): Run the diagnostics that is selected in `modular_list`
   - `modular_list` (*modular_list: psl.nam_nao*): Calculate only NAO diagnostics

Variables

- zg (atmos, monthly mean, longitude latitude time)
- tas (atmos, monthly mean, longitude latitude time)
- psl (atmos, monthly mean, longitude latitude time)
- va (atmos, monthly mean, longitude latitude time)
- ta (atmos, monthly mean, longitude latitude time)
- sic/siconc (seaice, monthly mean, longitude latitude time)
- ts (atmos, monthly mean, longitude latitude time)
- pr (atmos, monthly mean, longitude latitude time)

Observations and reformat scripts

- ERA5 (pr, psl, ta, tas, ts, va, zg - ERA5 data can be used via the native6 project)
- HadISST - (sic - esmvaltool/cmorizers/data/formatters/datasets/hadisst.ncl)

References

- Galytska, E., Weigel, K., Handorf, D., Jaiser, R., Köhler, R. H., Runge, J., & Eyring, V.: Causal model evaluation of Arctic-midlatitude teleconnections in CMIP6. Authorea Preprints. https://doi.org/10.1002/essoar.10512569.1.
- Copernicus Climate Change Service (C3S), 2017: ERA5: Fifth generation of ECMWF atmospheric reanalyses of the global climate, edited, Copernicus Climate Change Service Climate Data Store (CDS). https://cds.climate.copernicus.eu/cdsapp#!/home
- http://www.cesm.ucar.edu/working_groups/CVC/cvdp/

Example plots

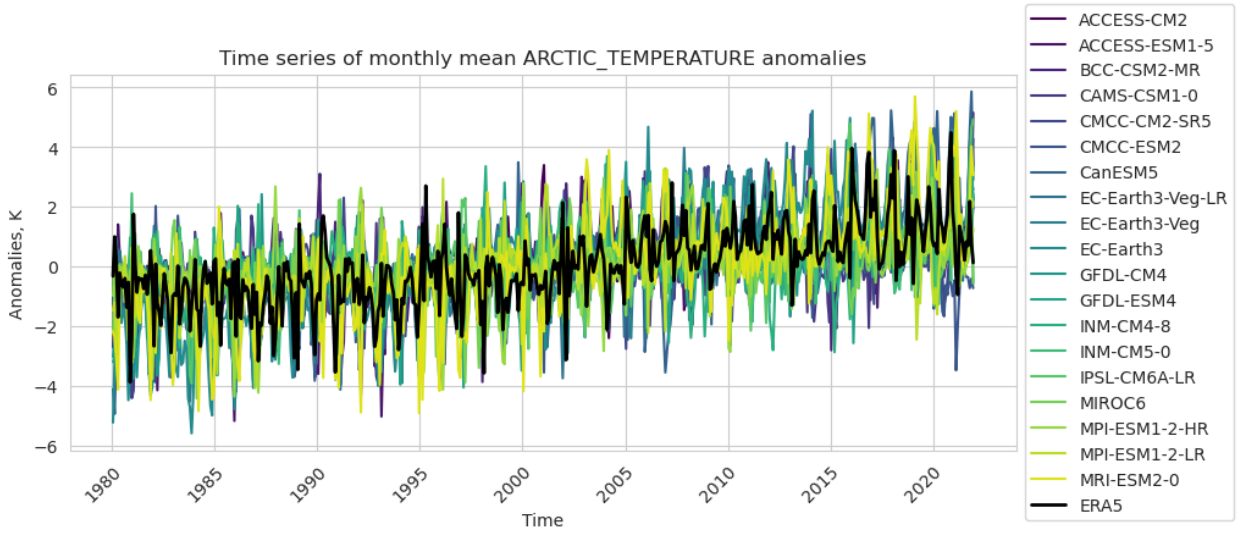


Fig. 295: Monthly mean temperature anomalies in the Arctic (65°–90°N) from observations and selected CMIP6 models during 1980-2021.

5.8.7 Multi-model products

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.

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.

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).

Variables

- any Amon variable (atmos, monthly mean, longitude latitude time)

Observations and reformat scripts

None

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

Example plots

5.8.8 Capacity factor for solar photovoltaic (PV) systems

Overview

This diagnostic computes the photovoltaic (PV) capacity factor, a measure of the fraction of the maximum possible energy produced per PV grid cell. It uses the daily incoming surface solar radiation and the surface temperature with a method described in Bett and Thornton (2016). The user can select temporal range, season, and region of interest.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_pv_capacity_factor.yml

Diagnostics are stored in diag_scripts/pv_capacityfactor/

- pv_capacity_factor.R: prepares data and plots results.
- PV_CF.R: calculates the daily capacity factor.

User settings

User setting files are stored in recipes/

1. recipe_capacity_factor.yml

   Required settings for script

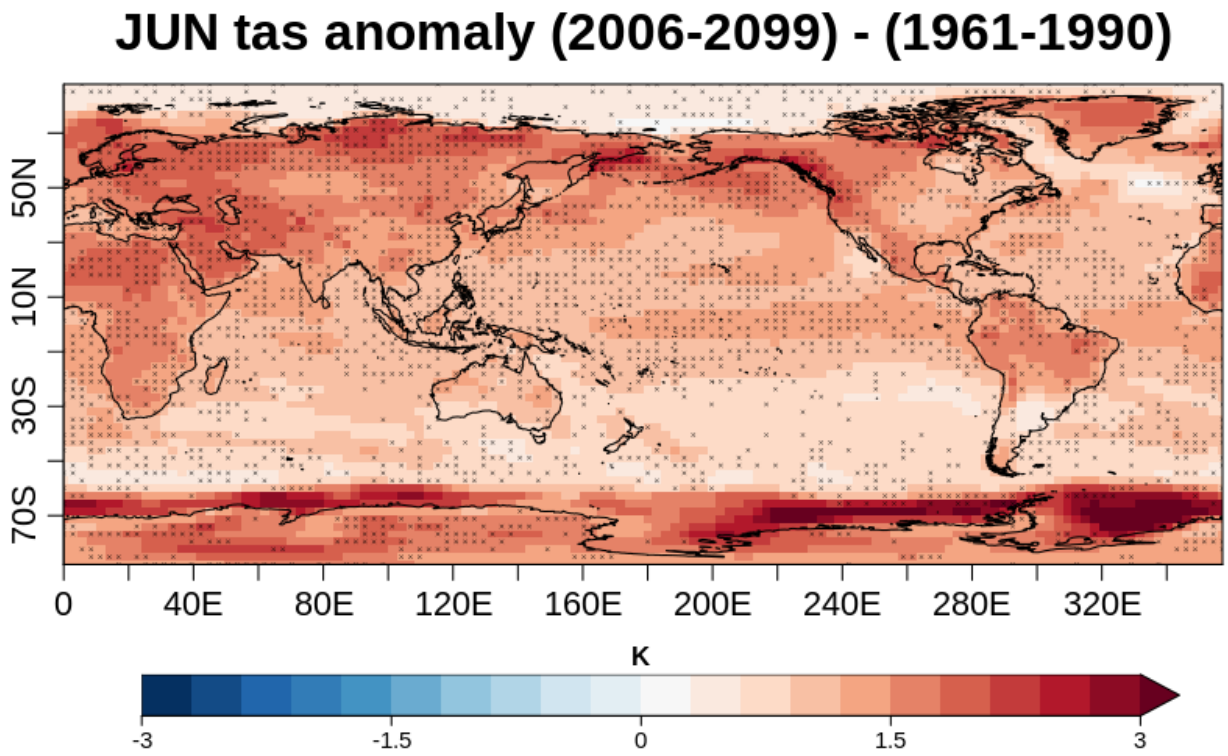


Fig. 296: Multi-model mean anomaly of 2-m air temperature during the future projection 2006-2099 in June considering the reference period 1961-1990 (colours). Crosses indicate that the 80% of models agree in the sign of the multi-model mean anomaly. The models selected are BCC-CSM1-1, MPI-ESM-MR and MIROC5 in the r1i1p1 ensembles for the RCP 2.6 scenario.

- season: String to include shortcut for season in plot title and name (e.g. "djf"). It will be converted to upper case. This season should be the one set in the preprocessor, since it is only used as a string and does not affect the data in the diagnostic. In the default recipe this is solved through a node anchor.

Optional settings for script

- maxval_colorbar: Optional upper limit for the colorbar.

Variables

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

Observations and reformat scripts

- ERA-Interim

References

- Bett, P. E. and Thornton, H. E.: The climatological relationships between wind and solar energy supply in Britain, Renew. Energ., 87, 96–110, https://doi.org/10.1016/j.renene.2015.10.006, 2016.

Example plots

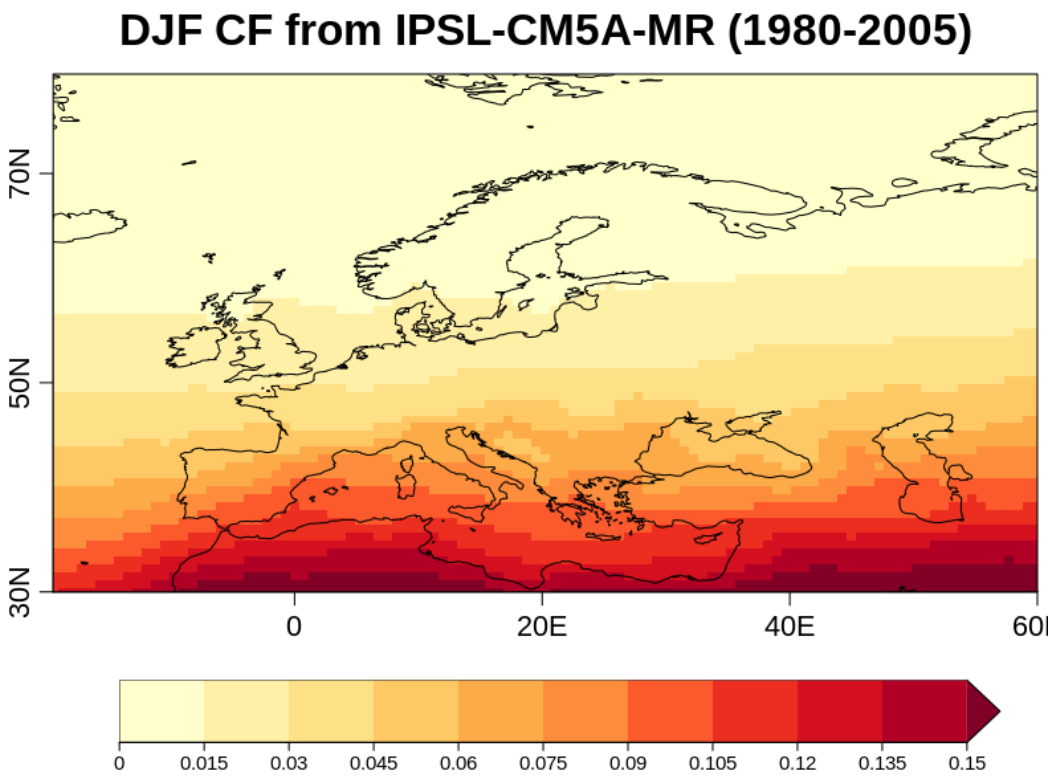


Fig. 297: PV capacity factor calculated from IPSL-CM5-MR during the DJF season for 1980–2005.

5.8.9 Sea Ice

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.

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

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

Variables

- sic (ocean-ice, monthly mean, longitude latitude time)
- areacello (fx, longitude latitude)

Observations and reformat scripts

Note: (1) obs4MIPs data can be used directly without any preprocessing; (2) use `esmvaltool data info DATASET` or see headers of cmorization scripts (in esmvaltool/cmorizers/data/formatters/datasets/) for non-obs4MIPs data for download instructions.

- HadISST (sic - esmvaltool/cmorizers/data/formatters/datasets/hadisst.ncl)

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.

Example plots

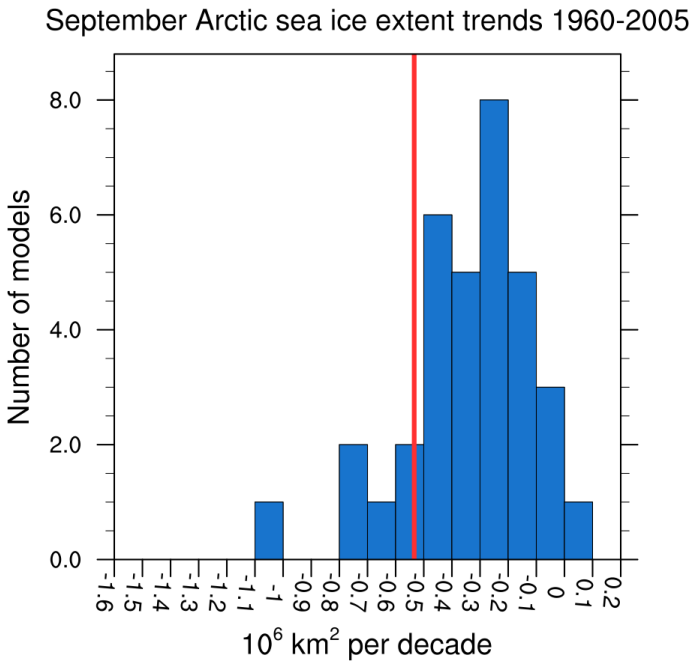


Fig. 298: Sea ice extent trend distribution for the Arctic in September (similar to IPCC AR5 Chapter 9, Fig. 9.24c). [seaice_trends.ncl]

5.8.10 Seaice drift

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.

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

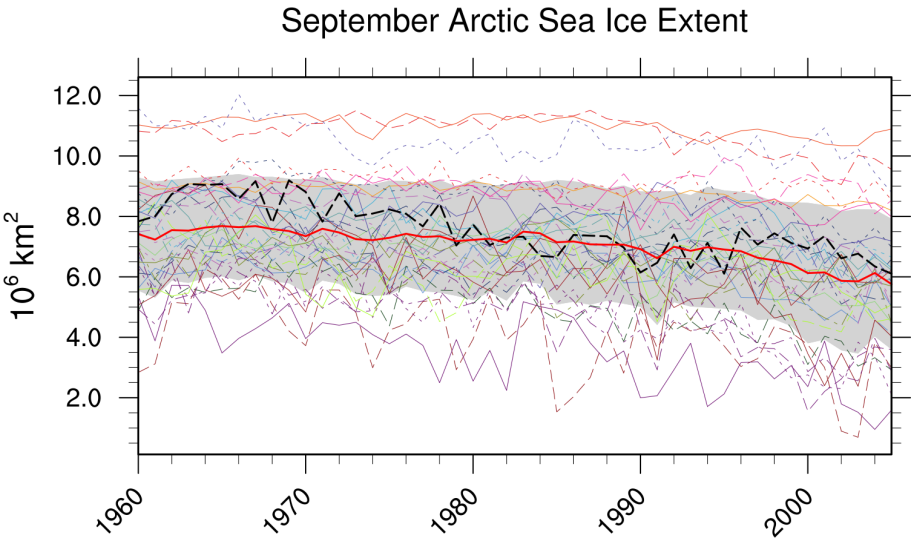


Fig. 299: Time series of total sea ice area and extent (accumulated) for the Arctic in September including multi-model mean and standard deviation (similar to IPCC AR5 Chapter 9, Fig. 9.24a). [seaice_tsline.ncl]

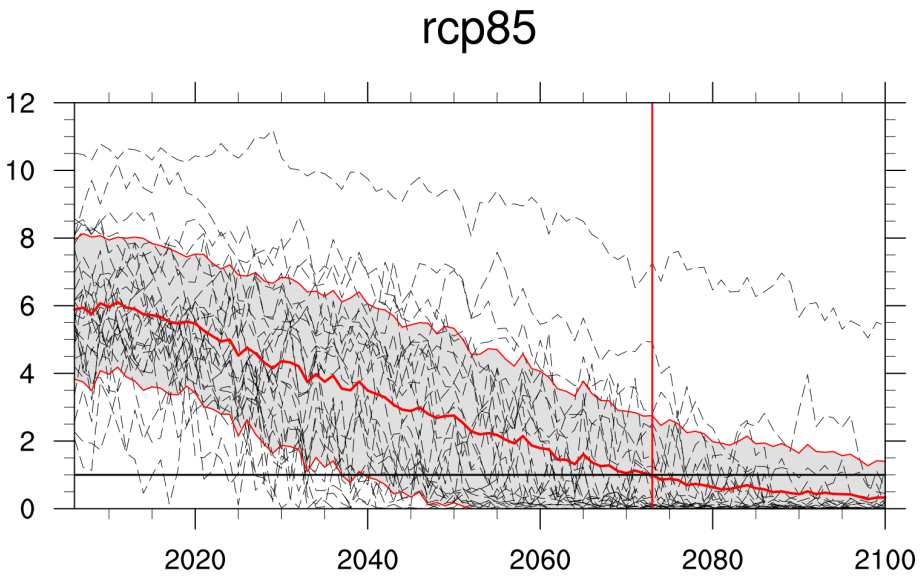


Fig. 300: Time series of September Arctic sea ice extent for individual CMIP5 models, multi-model mean and multi-model standard deviation, year of disappearance (similar to IPCC AR5 Chapter 12, Fig. 12.31e). [seaice_yod.ncl]

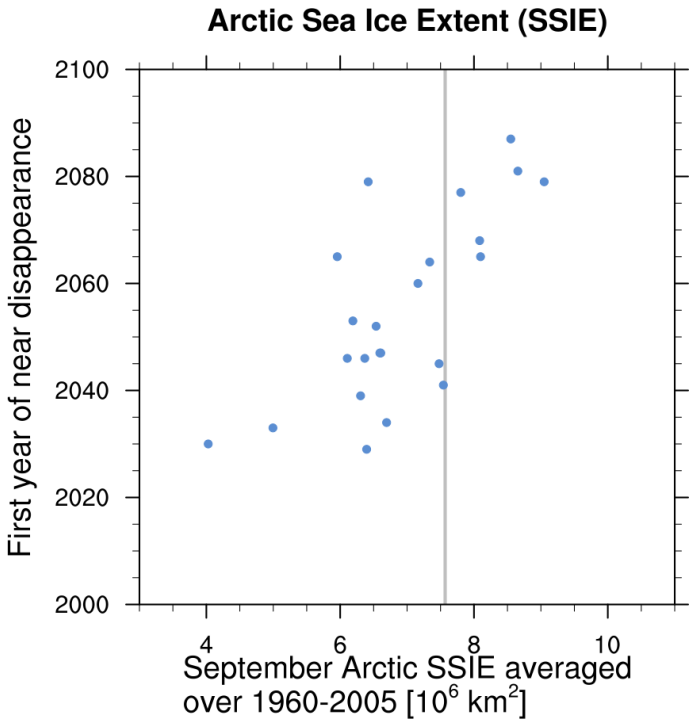


Fig. 301: Scatter plot of mean historical September Arctic sea ice extent vs 1st year of disappearance (RCP8.5) (similar to IPCC AR5 Chapter 12, Fig. 12.31a). [seaice_ecs.ncl]

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

Variables

- sispeed, sithick, siconc (daily)

Example plots

5.8.11 Seaice feedback

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

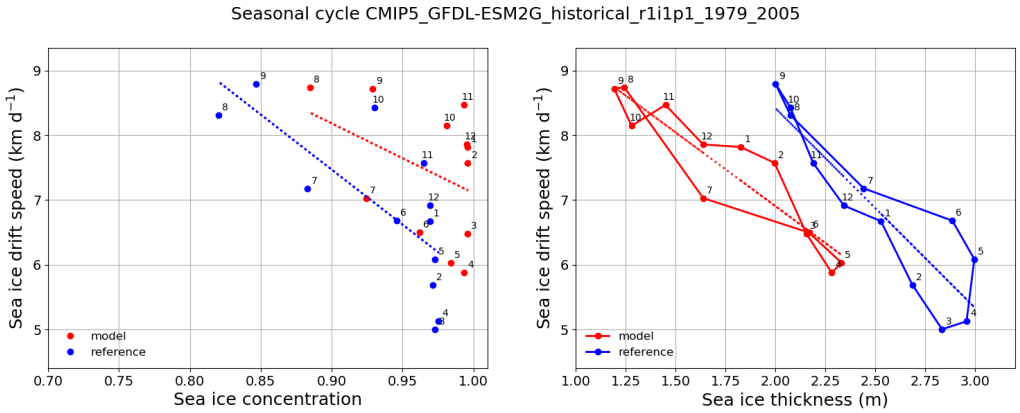


Fig. 302: Scatter plots of modelled (red) and observed (blue) monthly mean sea-ice drift speed against sea-ice concentration (left panel) and sea-ice thickness (right panel) temporally averaged over the period 1979–2005 and spatially averaged over the SCICEX box.

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.

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

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)

Variables

- sit (seaice, monthly mean, time latitude longitude)

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.

Example plots

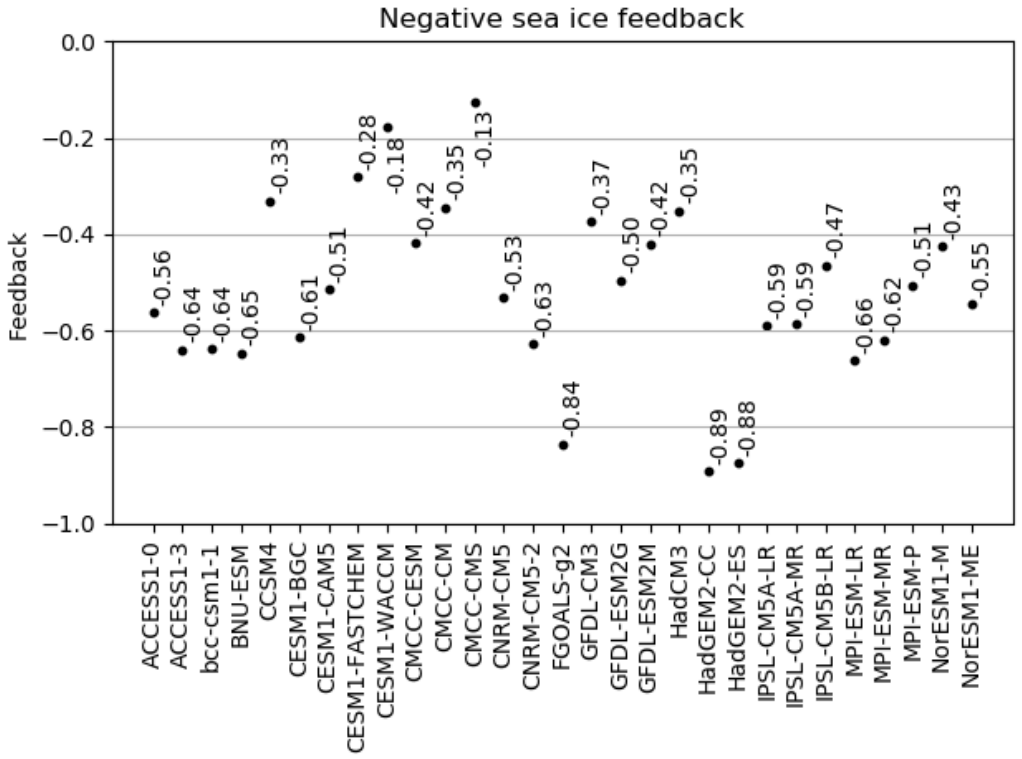


Fig. 303: Seaice negative feedback values (CMIP5 historical experiment 1979-2004).

5.8.12 Sea Ice Sensitivity

Overview

This recipe and diagnostic calculates the rate of sea ice area loss per degree of warming, as in plot 1d of Notz et al. and figure 3e of Roach et al.. The figures used in the plots are output to csv files in the work directory.

Available recipe and diagnostic

Recipe is in *recipes*

- recipe_seaice_sensitivity.yml

Diagnostic is in *diag_scripts/seaice/*

- seaice_sensitivity.py (Plotting the sensitivity of sea ice to mean global temperature).

Recipe settings

Years to be evaluated are specified in the `extract_test_period` preprocessor. Arctic sea ice is evaluated using data from September, and Antarctic sea ice is evaluated using annually meaned data. This can be amended by changing the preprocessor for each variable (shown below).

```
pp_arctic_sept_sea_ice:
  extract_time:
    start_day: 1
    start_month: 1
    start_year: 1979
    end_day: 31
    end_month: 12
    end_year: 2014
  extract_month:
    month: 9
  extract_region:
    start_longitude: 0
    end_longitude: 360
    start_latitude: 0
    end_latitude: 90
  area_statistics:
    operator: sum
  convert_units:
    units: 1e6 km2
```

Datasets

The recipe tries to use as many datasets as possible from the list in Table S3 of https://agupubs.onlinelibrary.wiley.com/action/downloadSupplement?doi=10.1029%2F2019GL086749&file=grl60504-sup-0002-Table_SI-S01.pdf Only one ensemble member is used for each model.

All, some or no datasets may be labelled in the plots, using `label_dataset: True` in the recipe settings.

References

- SIMIP_2020: Scripts for the processing, analysis and plots of the SIMIP Community Paper (Notz et al. 2020), https://github.com/jakobdoerr/SIMIP_2020/tree/master

Example plots

mean	-4.01 million km2 K-1
std_dev	0.32 million km2 K-1
plausible	1.28 million km2 K-1

5.8.13 Shapeselect

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

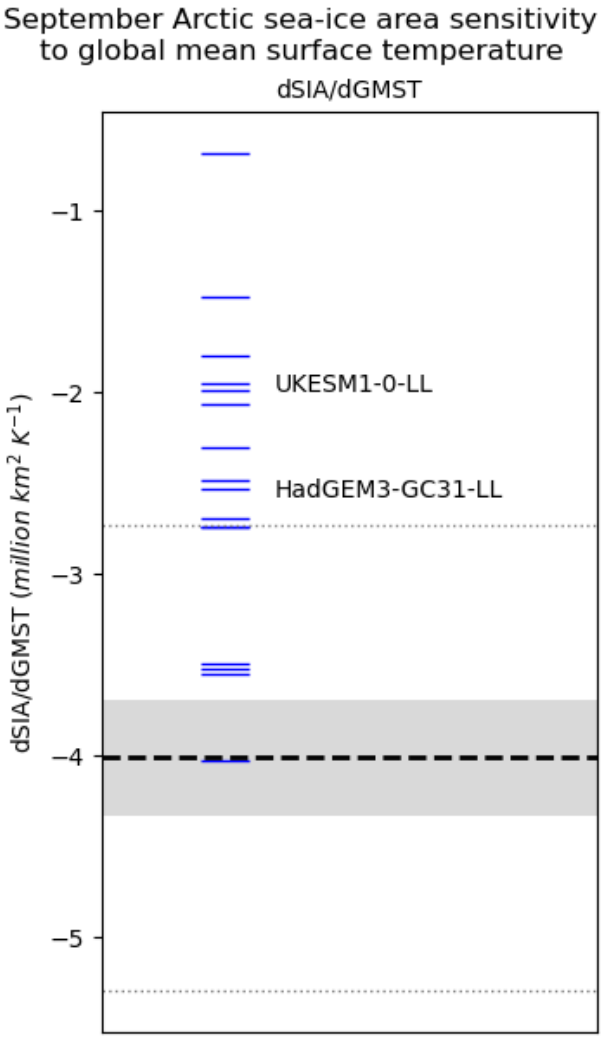


Fig. 304: Plot of sensitivity of northern hemisphere sea ice area loss (millions of square kilometres) in the month of September to the annual mean global temperature change (K).

The dashed black line shows the observational mean, the shaded area denotes one one standard deviation of observational uncertainty, as calculated by Notz et al (2020). The dotted grey lines reflect Notz et al estimate of a plausible range incorporating both internal variability and observational uncertainty. These values are configurable in the recipe, with the default values taken from Notz et al (2020):

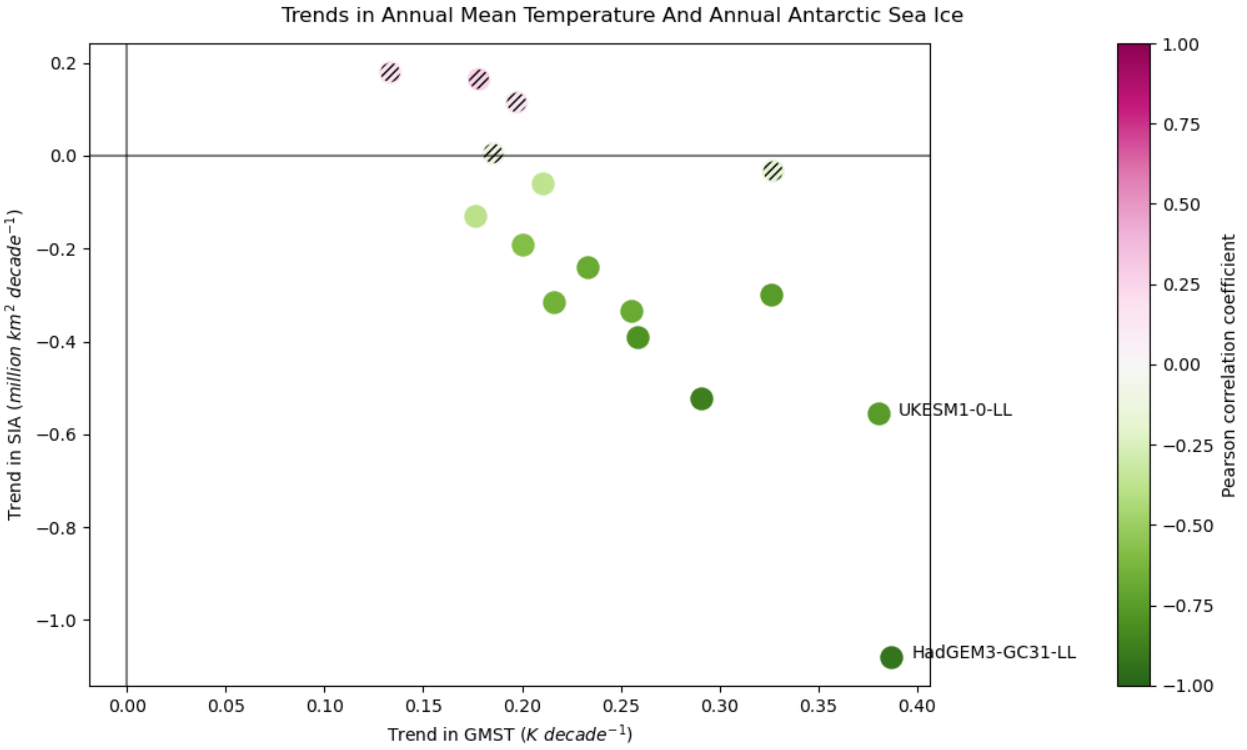


Fig. 305: Plot of the trend of annually averaged southern hemisphere sea ice area (millions of square kilometres) over time against the trend of annually and globally averaged air temperature near the surface (degrees Kelvin) over time. The values plotted are 10 times the annual trend, which was calculated using `scipy.stats.linregress()`, for consistency with the decadal values used in the published plot.
The colour of each point is determined by the Pearson correlation coefficient between the two variables, and the hatching indicates a `p_value` greater than 0.05, both calculated using `scipy.stats.linregress()`.

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.

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.

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 configuration option `auxiliary_data_dir`.
   - 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.

Variables

- pr,tas (daily)

Example plots

5.8.14 Short test versions of scientific recipes to check for backward compatibility.

Overview

These recipes are created to cover typical functionalities in the ESMValTool and allow to test them quickly. Each recipe should run less than 5 minutes to facilitate fast tests.

Available recipes and diagnostics

Recipes are stored in recipes/testing/

- recipe_deangelis15nat_fig1_fast.yml

Diagnostics are stored in diag_scripts/

- deangelis15nat/deangelisf1b.py

Other recipes used for the continuous integration (CI) tests; these are run only by the automated tests:

- recipe_python_for_CI.yml

This is identical to the example Python recipe, with the only difference being the location extraction preprocessor, which is replaced here with `extract_point`. The reason for this is that this recipe is used solely for CI tests, and the geolocator/Nominatim engine for location extraction should not be used in CI runs as per their usage policy.

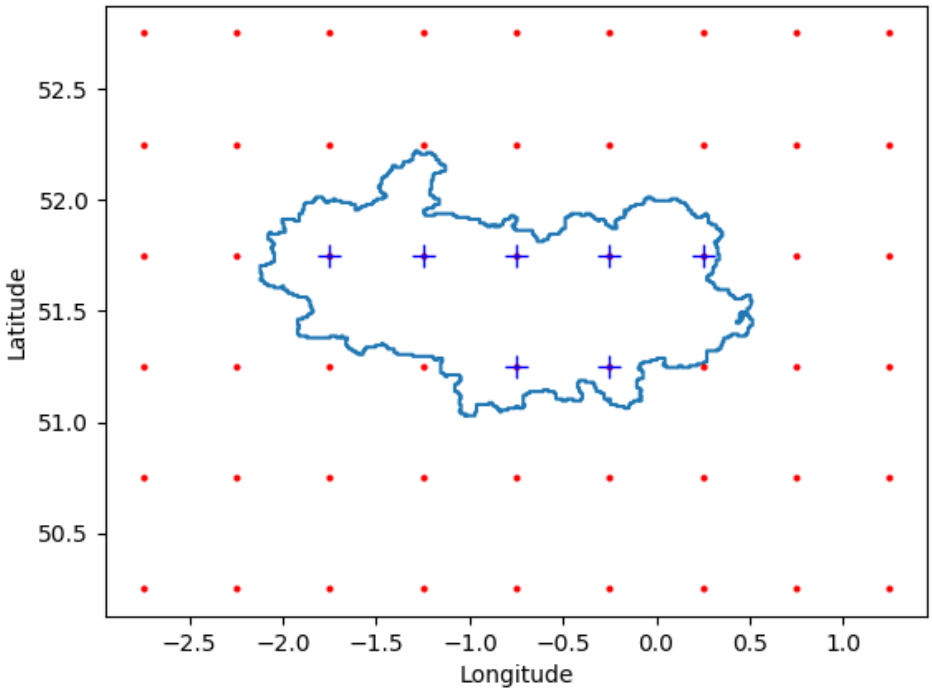


Fig. 306: Example of the selection of model grid points falling within (blue pluses) and without (red dots) a provided shapefile (blue contour).

User settings in recipes

The recipe recipe_deangelis15nat_fig1_fast.yml calls the first diagnostic (deangelisf1b.py) from the original recipe recipe_deangelis15nat.yml. It can be run with CMIP5 and CMIP6 models for any duration. Several flux variables (W m^{-2}) and up to 6 different model experiments can be handled. Each variable needs to be given for each model experiment. The same experiments must be given for all models. For testing purpose it was reduce to two models, 3 experiments and one year. For a more detailed documentation see *Evaluate water vapor short wave radiance absorption schemes of ESMs with the observations.*

5.8.15 Toymodel

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.

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

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)

Variables

- any variable (atmos/ocean, daily-monthly, longitude, latitude, time)

Observations and reformat scripts

None

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.

Example plots

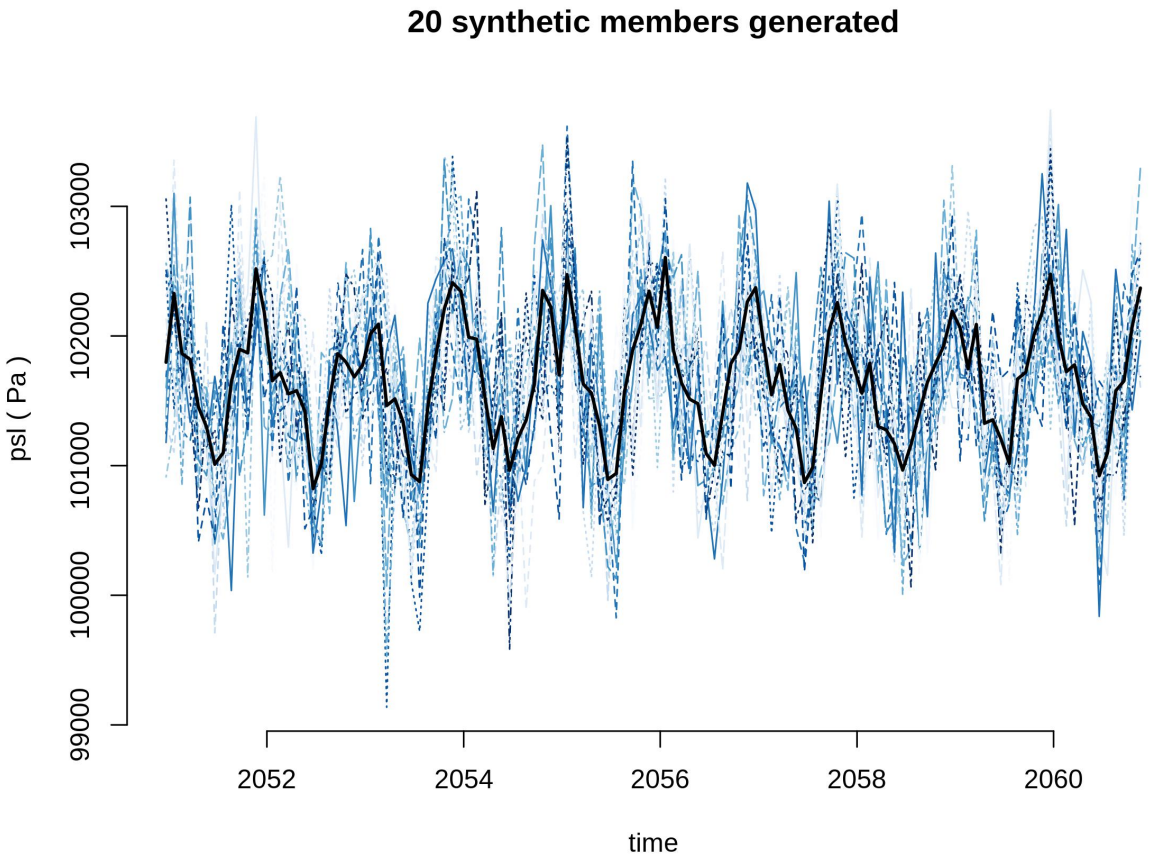


Fig. 307: Twenty synthetic single-model ensemble generated by the recipe_toymodel.yml (see Section 3.7.2) for the 2051-2060 monthly data of r1i1p1 RCP 4.5 scenario of BCC_CSM1-1 simulation.

5.9 Legacy recipes

Recipes that have been retired and are included for documentation purposes only. Typically, these recipes require a legacy version of ESMValTool to run.

5.9.1 Legacy Recipes

Recipes that have been retired and are included for documentation purposes only. Typically, these recipes require a legacy version of ESMValTool to run.

Legacy recipe: Diagnostics of stratospheric dynamics and chemistry

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.13.0.

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.

Available recipes and diagnostics

Recipes are stored in archive/legacy_recipes

- recipe_eyring06jgr.yml

Diagnostics are stored in archive/legacy_diag_scripts/eyring06jgr/

- eyring06jgr_fig01.ncl
- eyring06jgr_fig05a.ncl
- eyring06jgr_fig05b.ncl
- eyring06jgr_fig15.ncl

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.

Variables

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

Example plots

Legacy recipe: Ozone and associated climate impacts

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.13.0.

Overview

This recipe is implemented into the ESMValTool to evaluate atmospheric chemistry and the climate impact of stratospheric ozone changes. It reproduces selected plots from Eyring et al. (2013).

The following plots are reproduced:

- Zonal mean of long-term zonal wind with linear trend

Available recipes and diagnostics

Recipes are stored in archive/legacy_recipes

- recipe_eyring13jgr_12.yml

Diagnostics are stored in archive/legacy_diag_scripts/eyring13jgr/

- eyring13jgr_fig12.ncl

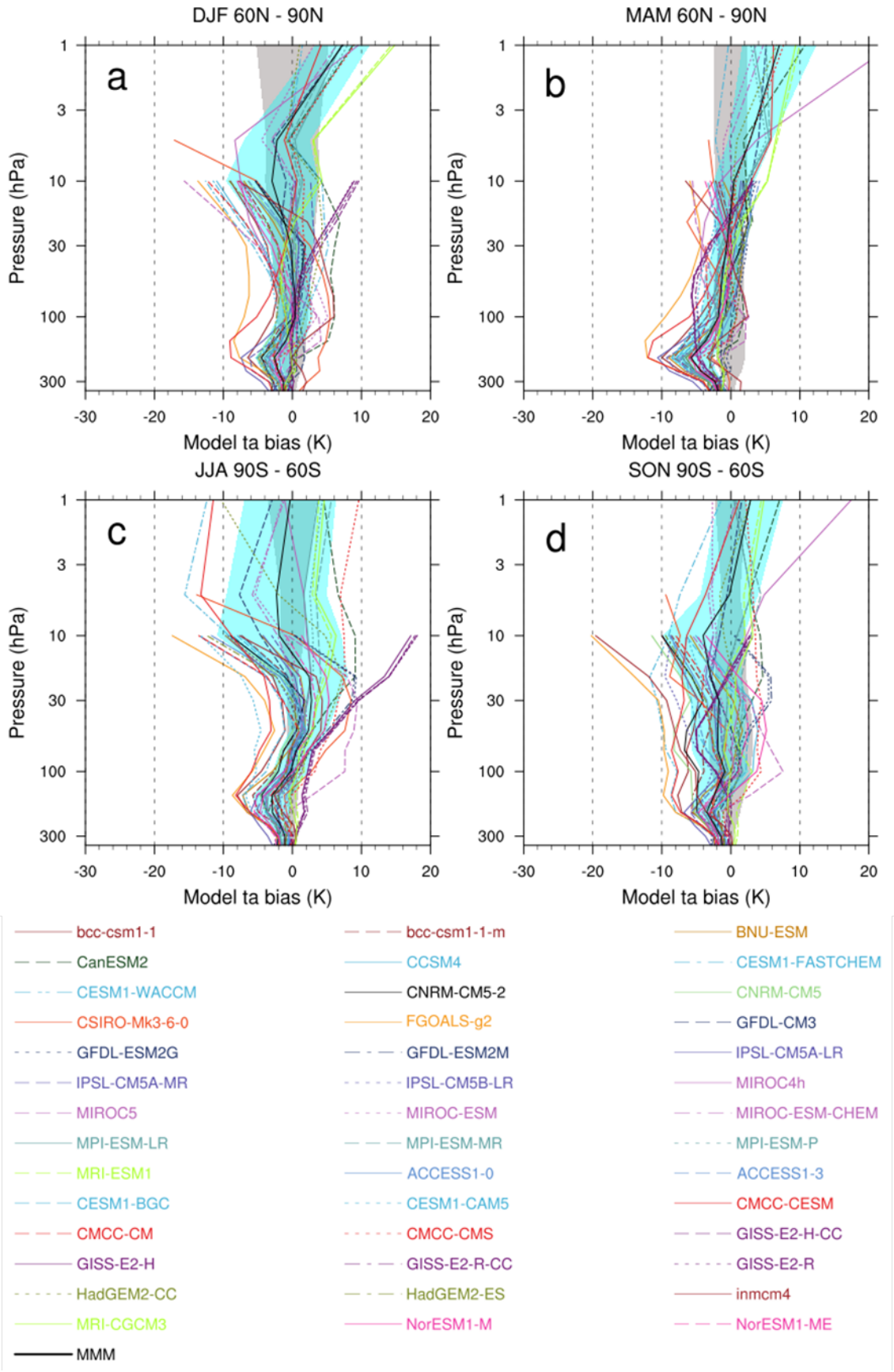


Fig. 308: Climatological mean temperature biases for (top) 60–90N and (bottom) 60–90S for the (left) winter and (right) spring seasons. The climatological means for the CCMs and ERA-Interim data from 1980 to 1999 are included. Biases are calculated relative to ERA-Interim reanalyses. The grey area shows 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.

User settings in recipe

1. Preprocessor
   - `zonal` : Regridding and zonal mean used by eyring13jgr_fig12
2. Script <eyring13jgr_fig12.ncl>

   Required settings for script
   - `e13fig12_exp_MMM`: name of the experiments for the MMM

   Optional settings for script
   - `e13fig12_start_year`: year when to start the climatology calculation
   - `e13fig12_end_year`: year when to end the climatology calculation
   - `e13fig12_multimean`: calculate multimodel mean (default: False)
   - `e13fig12_season`: season (default: ANN (annual))

   Required settings for variables
   - `preprocessor`: zonal
   - `reference_dataset`: name of the reference model or observation for regridding and bias calculation (e.g. ERA5).
   - `mip`: Amon.

Variables

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

Observations and reformat scripts

- ERA5 *Reformatting with:* recipes/cmorizers/recipe_era5.yml

Example plots

Legacy recipe: Psyplot Diagnostics

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.12.0.

Overview

These recipes showcase the use of the Psyplot diagnostic that provides a high-level interface to Psyplot for ESMValTool recipes.

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_psyplot.yml

Diagnostics are stored in archive/legacy_diag_scripts

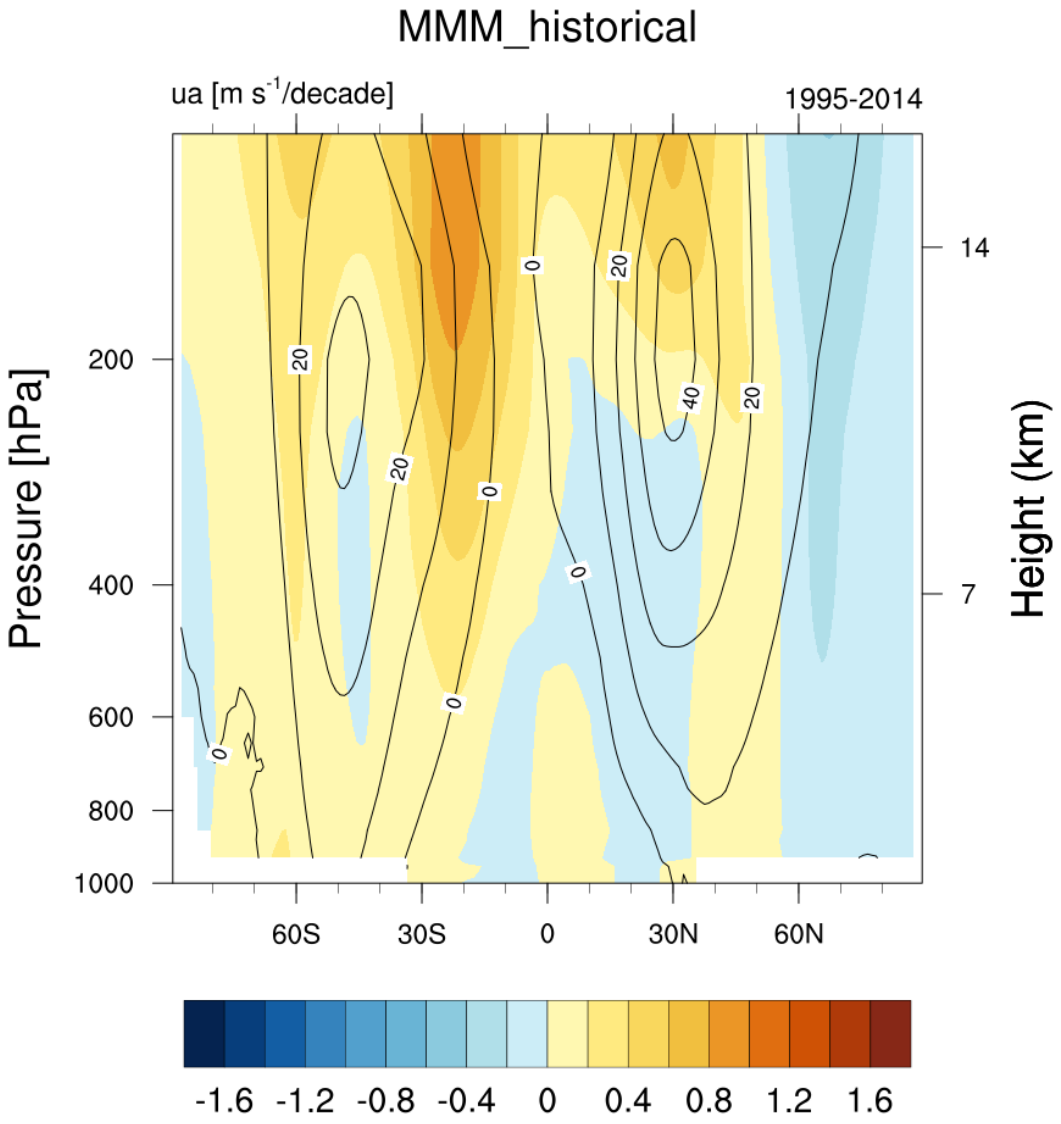


Fig. 309: Long-term mean (thin black contour) and linear trend (colour) of zonal mean DJF zonal winds for the multi-model mean CMIP6 over 1995-2014

- psyplot_diag.py

Variables

Arbitrary variables are supported.

Observations and reformat scripts

Arbitrary datasets are supported.

References

- Sommer, (2017), The psyplot interactive visualization framework, Journal of Open Source Software, 2(16), 363, doi:10.21105/joss.00363

Example plots

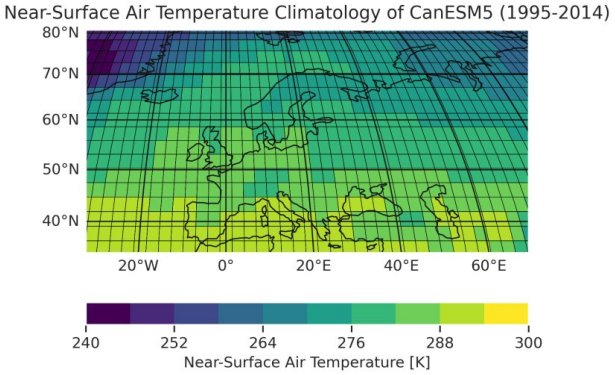


Fig. 310: Historical near-surface air temperature climatology over Europe simulated by CanESM5 between 1995 and 2014. The plot visualizes the invidividual rectangular grid cells of the model's regular grid.

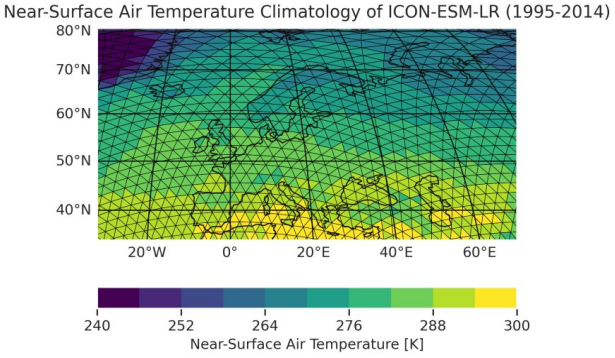


Fig. 311: Historical near-surface air temperature climatology over Europe simulated by ICON-ESM-LR between 1995 and 2014. The plot visualizes the invidividual triangular grid cells of the model's unstructured grid.

Legacy recipe: RainFARM stochastic downscaling

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.11.0.

Note

Rainfarm is also published as an R package: https://github.com/jhardenberg/rainfarmr However, we don't know if it is up to date and provides the same features. Similar implementations in Python are also available in the pysteps library at https://github.com/pySTEPS/pysteps/blob/master/examples/rainfarm_downscale.py

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).

Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_rainfarm.yml

Diagnostics are stored in diag_scripts/rainfarm/

- rainfarm.jl

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).

Variables

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

Observations and reformat scripts

None.

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

Example plots

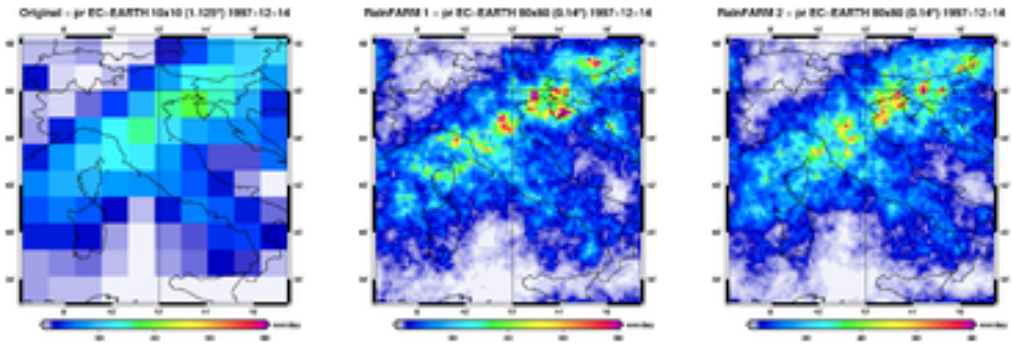


Fig. 312: Example of daily cumulated precipitation from the CMIP5 EC-EARTH model on a specific day, downscaled using RainFARM from its original resolution (1.125°) (left panel), increasing spatial resolution by a factor of 8 to 0.14°; Two stochastic realizations are shown (central and right panel). A 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).

Legacy recipe: Constraining uncertainty in projected gross primary production (GPP) with machine learning

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.12.0.

Warning

Not all datasets necessary to run these recipes are available on ESGF. The following datasets are missing:

- Dataset: co2, Amon, CMIP5, HadGEM2-ES, esmHistorical, r1i1p1
- Dataset: gpp, Lmon, CMIP5, MIROC-ESM, esmFixClim1, r1i1p1
- Supplementary: sftlf, fx, CMIP5, MIROC-ESM, esmFixClim1, r0i0p0

Overview

These recipes reproduce the analysis of Schlund et al., JGR: Biogeosciences (2020). In this paper, a machine learning regression (MLR) approach (using the MLR algorithm Gradient Boosted Regression Trees, GBRT) is proposed to constrain uncertainties in projected gross primary production (GPP) in the RCP 8.5 scenario using observations of process-based diagnostics.

Recipes and diagnostics

Recipes are stored in archive/legacy_recipes/

- schlund20jgr/recipe_schlund20jgr_gpp_abs_rcp85.yml
- schlund20jgr/recipe_schlund20jgr_gpp_change_1pct.yml
- schlund20jgr/recipe_schlund20jgr_gpp_change_rcp85.yml

Diagnostics are stored in archive/legacy_diag_scripts/

- mlr/evaluate_residuals.py
- mlr/main.py
- mlr/mmm.py
- mlr/plot.py
- mlr/postprocess.py
- mlr/preprocess.py
- mlr/rescale_with_emergent_constraint.py

Variables

- *co2s* (atmos, monthly, longitude, latitude, time)
- *gpp* (land, monthly, longitude, latitude, time)
- *gppStderr* (land, monthly, longitude, latitude, time)
- *lai* (land, monthly, longitude, latitude, time)
- *pr* (atmos, monthly, longitude, latitude, time)
- *rsds* (atmos, monthly, longitude, latitude, time)

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

Observations and reformat scripts

- CRU (*pr*, *tas*)
- ERA-Interim (*rsds*)
- LAI3g (*lai*)
- MTE (*gpp*, *gppStderr*)
- Scripps-CO2-KUM (*co2s*)

References

- Schlund, M., Eyring, V., Camps-Valls, G., Friedlingstein, P., Gentine, P., & Reichstein, M. (2020). Constraining uncertainty in projected gross primary production with machine learning. Journal of Geophysical Research: Biogeosciences, 125, e2019JG005619, https://doi.org/10.1029/2019JG005619.

Example plots

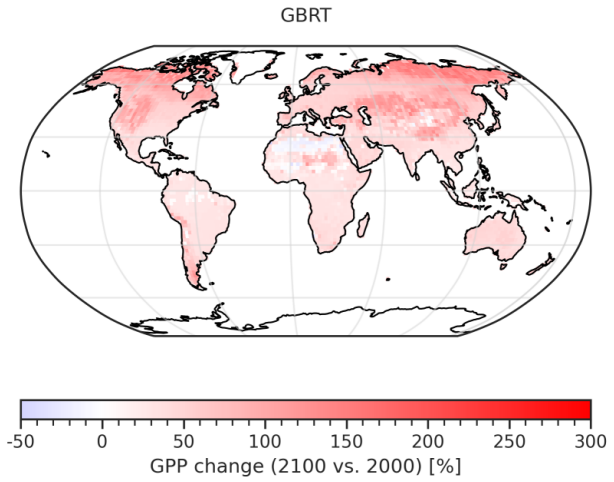


Fig. 313: GBRT-based prediction of the fractional GPP change over the 21st century (= GPP(2091-2100) / GPP(1991-2000)).

Legacy Recipe: Standardized Precipitation-Evapotranspiration Index (SPEI)

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.12.0. Most of its functionality is available in more general diagnostics under "droughts". See *Standardized Precipitation-Evapotranspiration Index (SPEI)* for a similar recipe based on the updated diagnostics.

Overview

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

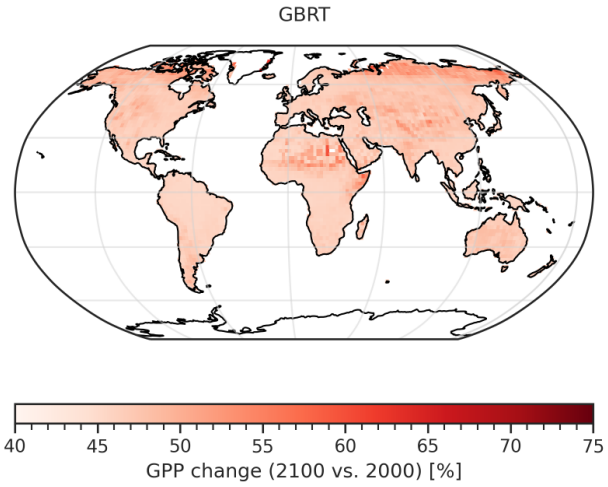


Fig. 314: Corresponding error of the GBRT-based prediction of the fractional GPP change over the 21st century (considering errors in the MLR model and errors in the predictors).

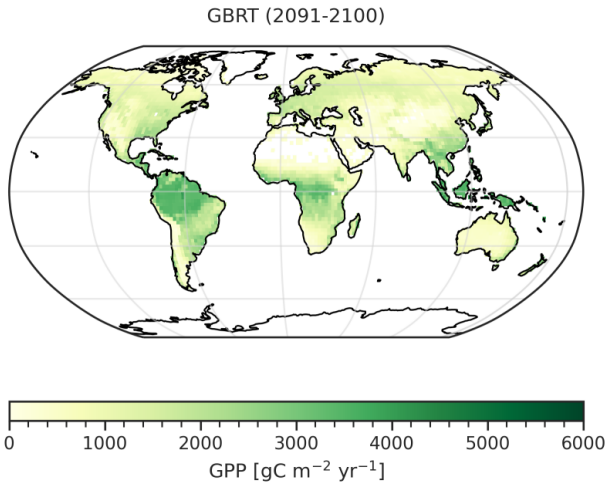


Fig. 315: GBRT-based prediction of the absolute GPP at the end of the 21st century (2091-2100).

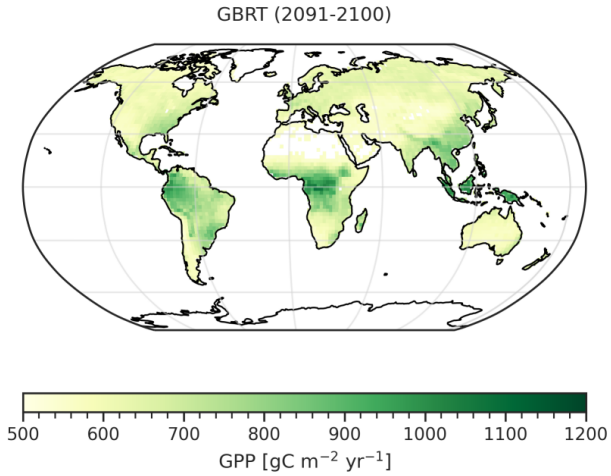


Fig. 316: Corresponding error of the GBRT-based prediction of the absolute GPP at the end of the 21st century (considering errors in the MLR model and errors in the predictors).

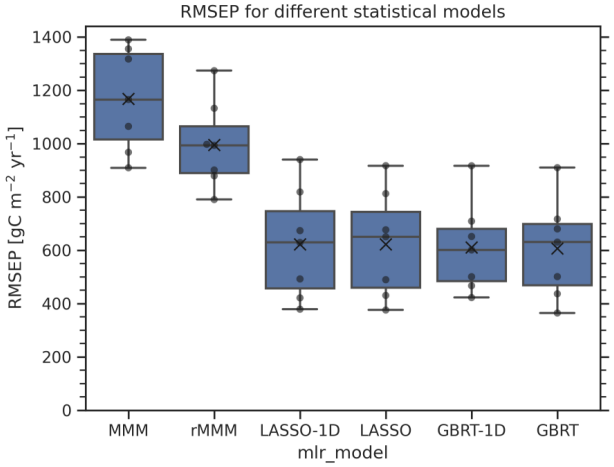


Fig. 317: Boxplot of the root mean square error of prediction (RMSEP) distributions for six different statistical models used to predict future absolute GPP (2091-2100) using a leave-one-model-out cross-validation approach. The distribution for each statistical model contains seven points (black dots, one for each climate model used as truth) and is represented in the following way: the lower and upper limit of the blue boxes correspond to the 25% and 75% quantiles, respectively. The central line in the box shows the median, the black "x" the mean of the distribution. The whiskers outside the box represent the range of the distribution

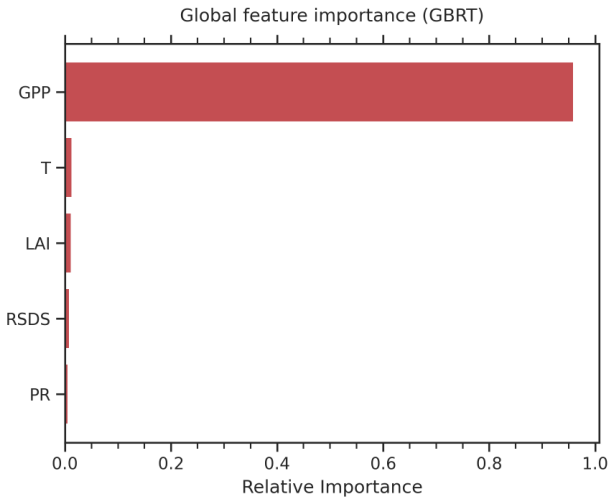


Fig. 318: Global feature importance of the GBRT model for prediction of the absolute GPP at the end of the 21st century (2091-2100).

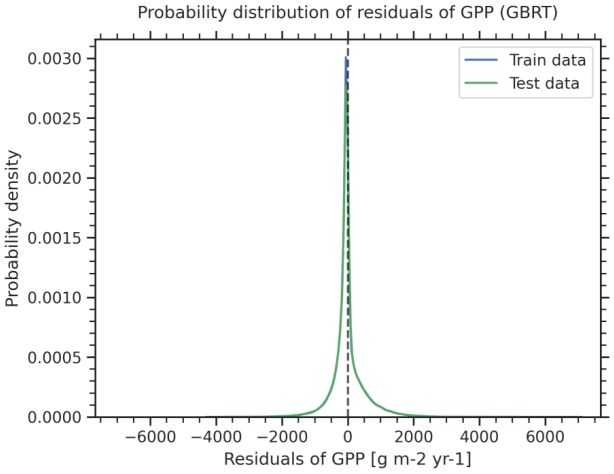


Fig. 319: Distribution of the residuals of the GBRT model for the prediction of absolute GPP at the end of the 21st century (2091-2100) for the training data (blue) and test data excluded from training (green).



Fig. 320: Training progress of the GBRT model for the prediction of absolute GPP at the end of the 21st century (2091-2100) evaluated as normalized root mean square error on the training data (blue) and test data excluded from training (green).

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 groundwater 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.

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

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.

Variables

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

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.

Example plots

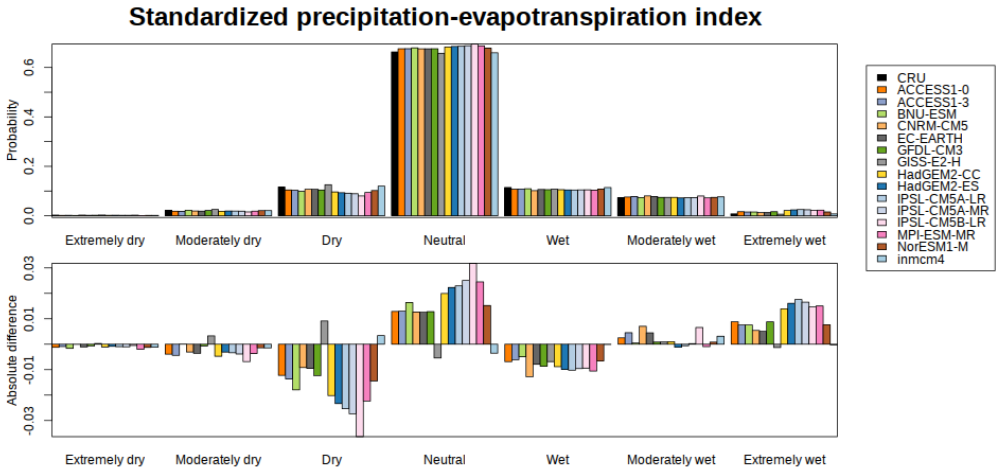


Fig. 321: (top) Probability distribution of the standardized precipitation-evapotranspiration index of a sub-set of the CMIP5 models, and (bottom) bias relative to the CRU reference data set.

Legacy recipe: Emergent constraints on carbon cycle feedbacks

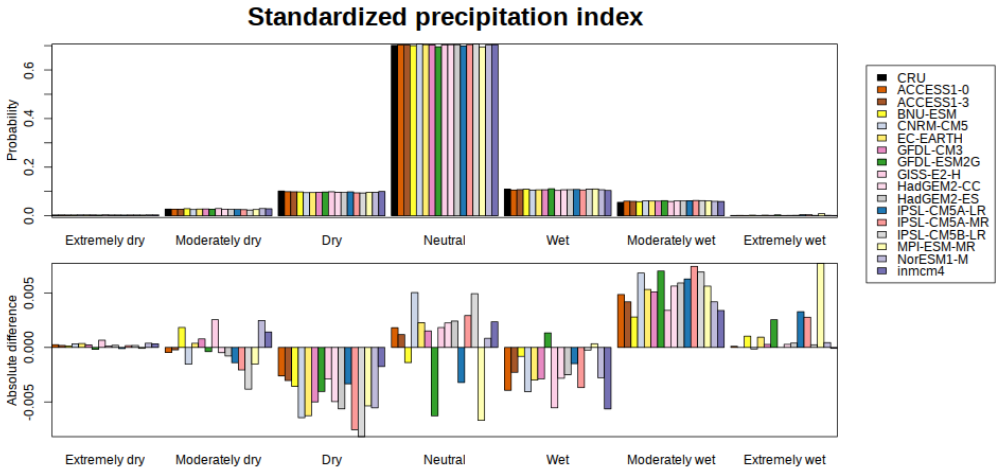


Fig. 322: (top) Probability distribution of the standardized precipitation index of a sub-set of the CMIP5 models, and (bottom) bias relative to the CRU reference data set.

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.13.0.

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.

Available recipes and diagnostics

Recipes are stored in archive/legacy_recipes

- recipe_wenzel14jgr.yml

Diagnostics are stored in archive/legacy_diag_scripts/carbon_ec

- 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

User settings

> **Note**
>
> Make sure to run this recipe with the configuration option `max_parallel_tasks: 1`.

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

Variables

- tas (atmos, monthly mean, longitude latitude time)
- nbp (land, monthly mean, longitude latitude time)
- fgco2 (ocean, monthly mean, longitude latitude time)

Observations and reformat scripts

- GCP2018: Global Carbon Budget including land (nbp) and ocean (fgco2) carbon fluxes
- NCEP-NCAR-R1: National Centers for Environmental Prediction reanalysis data for near surface temperature

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.

Example plots

Legacy recipe: Multiple ensemble diagnostic regression (MDER) for constraining future austral jet position

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.13.0.

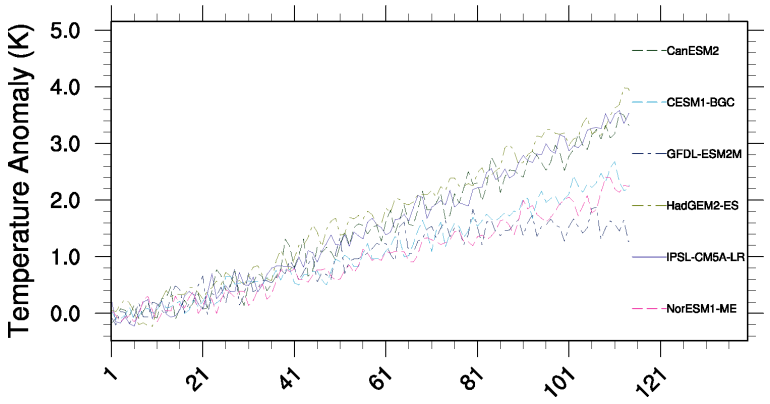


Fig. 323: Time series of tropical (30S to 30N) mean near surface temperature (tas) change between year 30 and year 110 for the CMIP5 models simulated with prescribed CO2 (1%/yr CO2 increase) coupled simulation (1pctCO2).

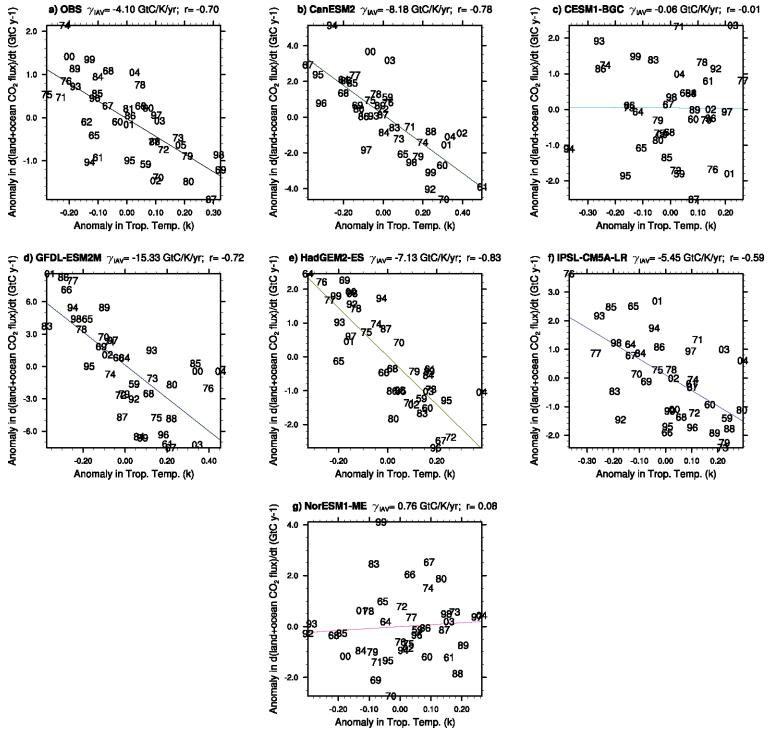


Fig. 324: Correlations between the interannual variability of global co2flux (nbp+fgco2) and tropical temperature for the individual CMIP5 models using esmHistorical simulations, and for observations.

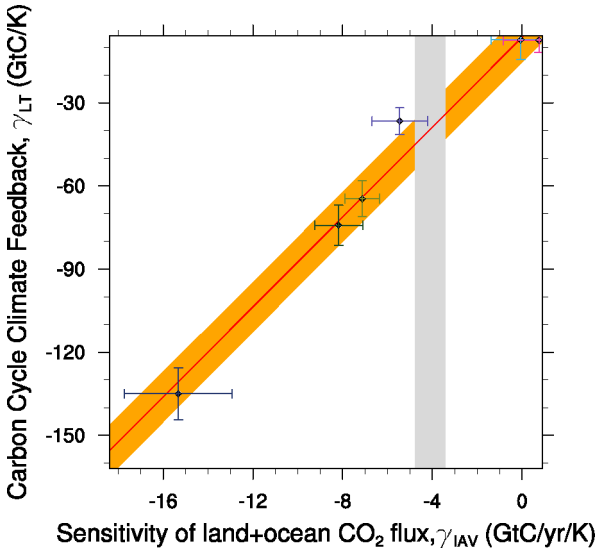


Fig. 325: Carbon cycle-climate feedback of tropical land carbon vs. the sensitivity of co2flux to interannual temperature variability in the tropics (30S to 30N). The red line shows the linear best fit of the regression together with the prediction error (orange shading) and the gray shading shows the observed range.

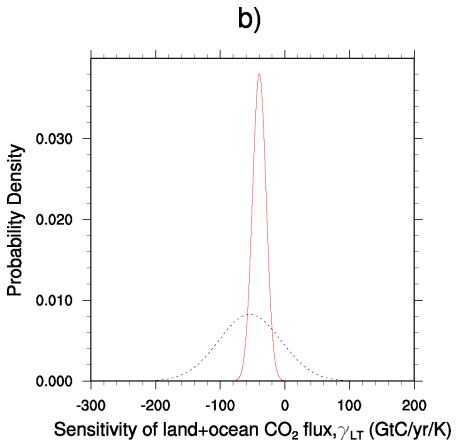


Fig. 326: Probability Density Functions for the pure CMIP5 ensemble (black dashed) and after applying the observed constraint to the models (red solid)

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.

Available recipes and diagnostics

Recipes are stored in archive/legacy_recipes

- recipe_wenzel16jclim.yml

Diagnostics are stored in archive/legacy_diag_scripts

- austral_jet/asr.ncl
- austral_jet/main.ncl
- mder/absolute_correlation.ncl
- mder/regression_stepwise.ncl
- mder/select_for_mder.ncl

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"`.

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)

Observations and reformat scripts

- ERA-Intermin (*ta*, *uajet*, *va*, *ps*)
- CERES-EBAF (*asr*)

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.

Example plots

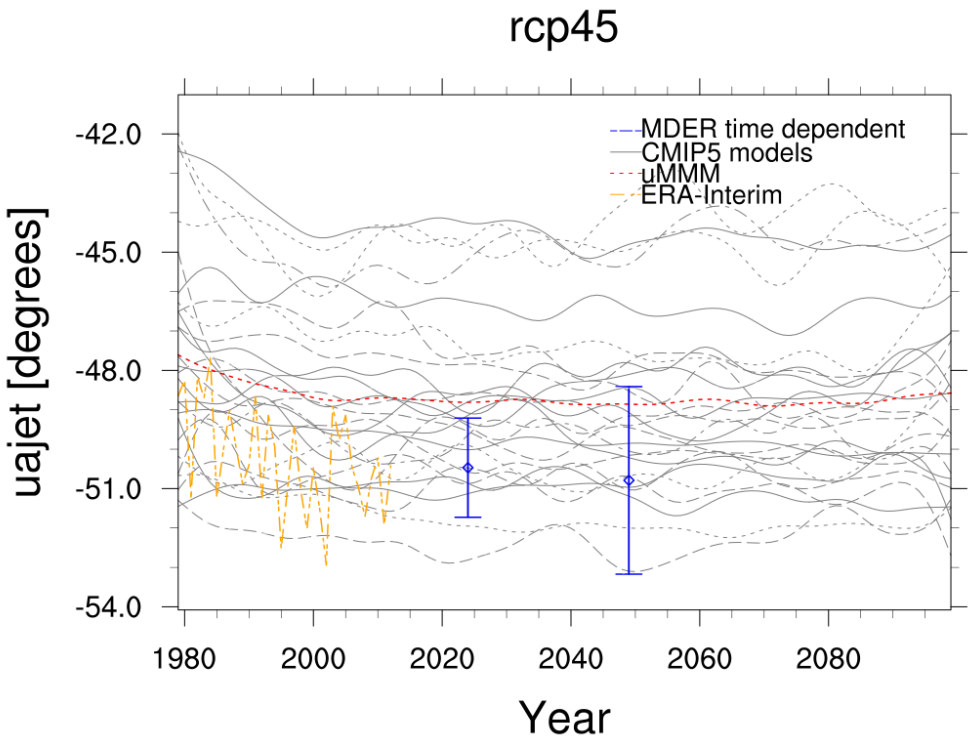


Fig. 327: Time series of the the target variable (future austral jet position in the RCP 4.5 scenario) for the CMIP5 ensemble, observations, unweighted multi-model mean projections and (MDER) weighted multi-model mean projections.

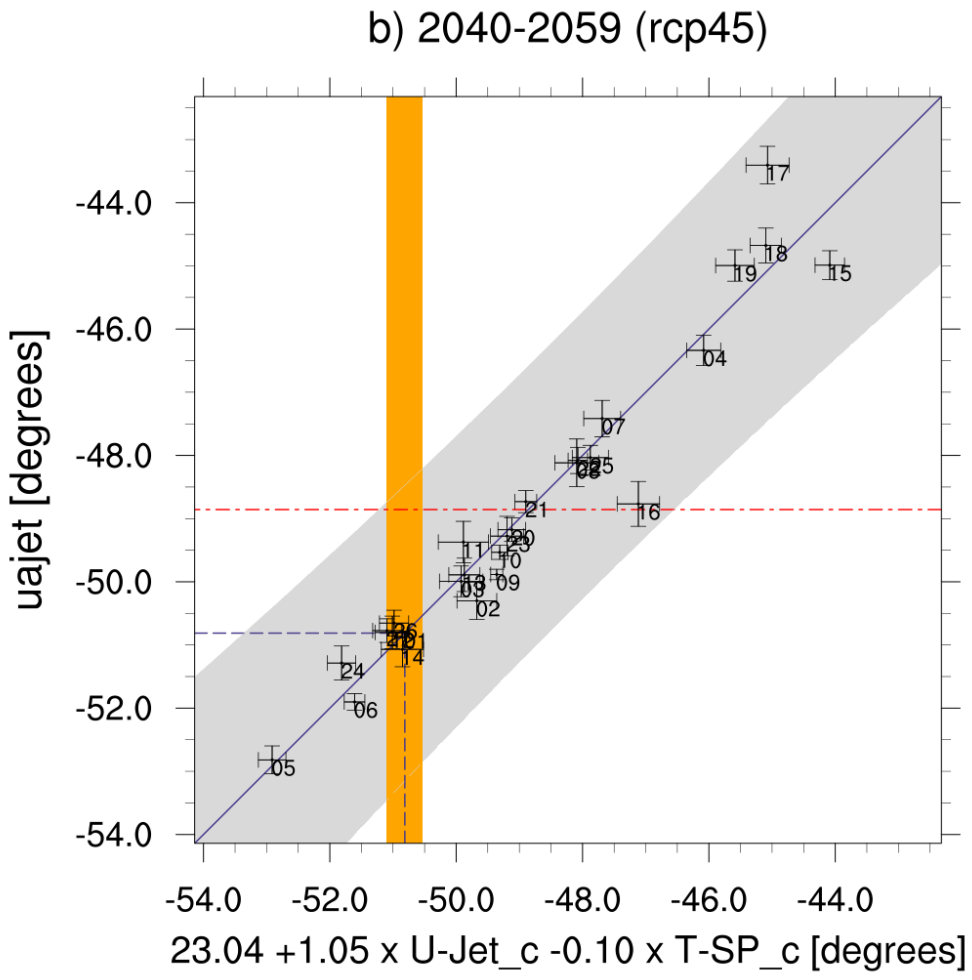


Fig. 328: Scatterplot of the target variable (future austral jet position in the RCP 4.5 scenario) vs. the MDER-determined linear combination of diagnostics for the CMIP5 ensemble.

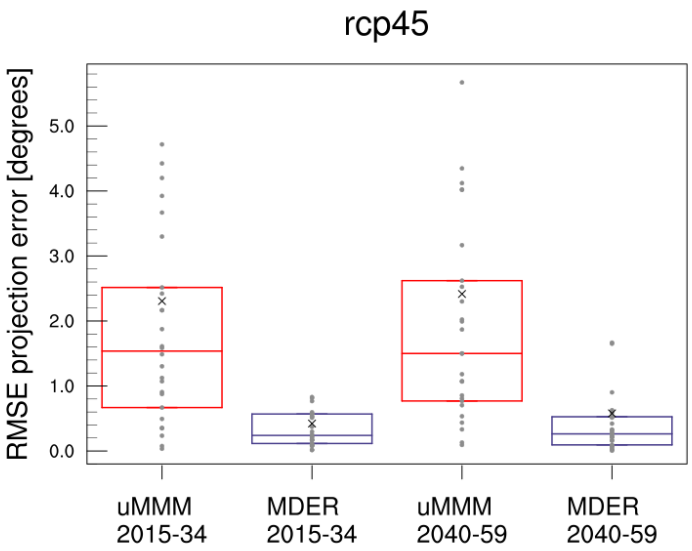


Fig. 329: Boxplot for the RMSE of the target variable for the unweighted and (MDER) weighted multi-model mean projections in a pseudo-reality setup.

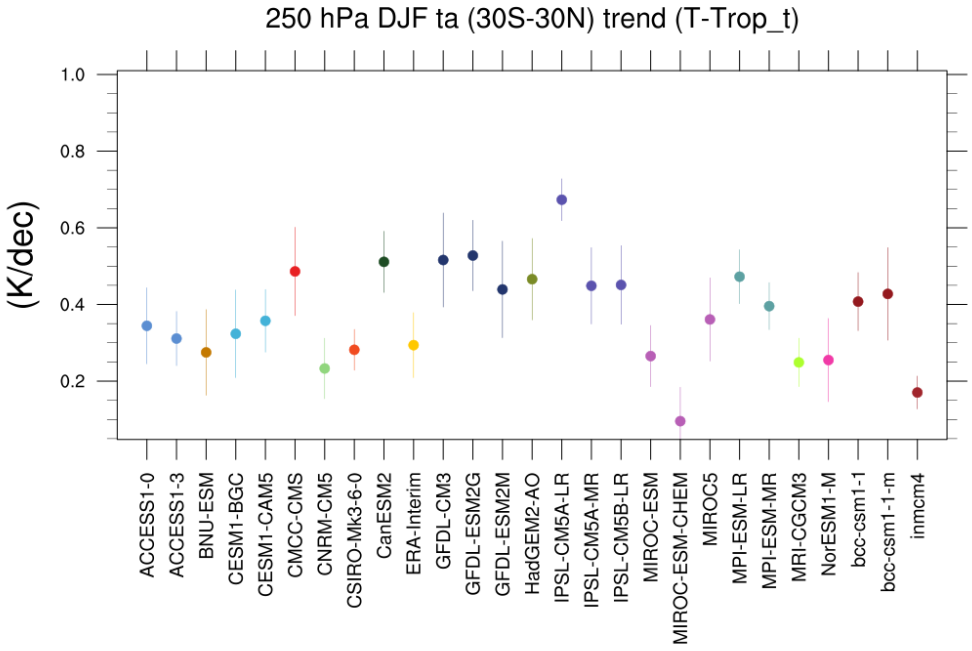


Fig. 330: Trends in tropical DJF temperature at 250hPa for different CMIP5 models and observations.

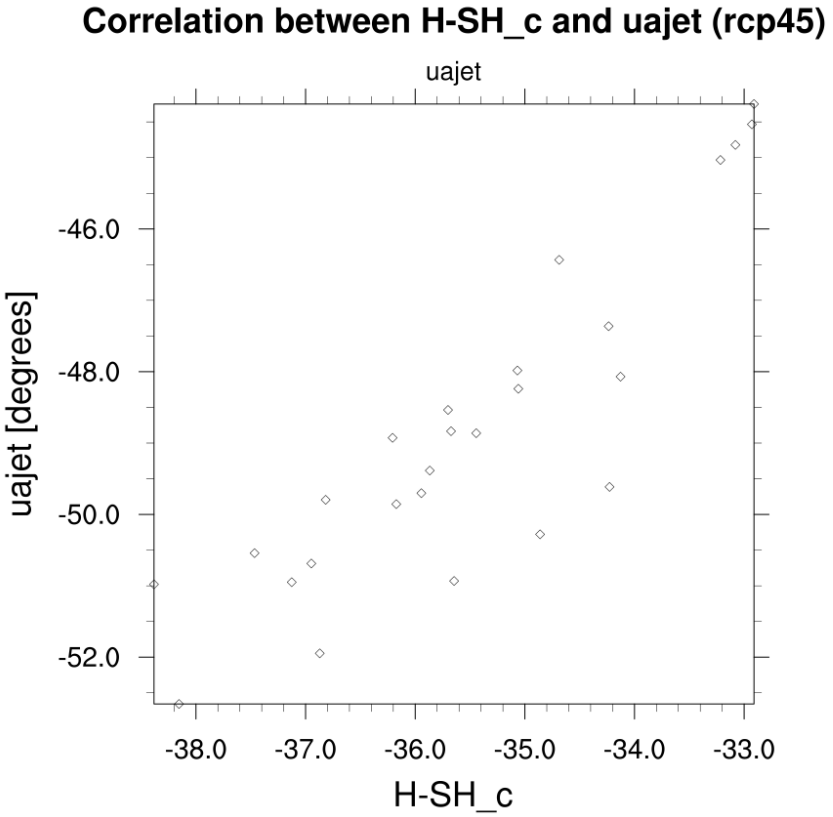


Fig. 331: Scatterplot of the target variable (future austral jet position in the RCP 4.5 scenario) vs. a single diagnostic, the historical location of the Southern hemisphere Hadley cell boundary for the CMIP5 ensemble.

Legacy recipe: Projected land photosynthesis constrained by changes in the seasonal cycle of atmospheric CO_2

Note

This recipe has been retired and moved to the folder "archive". This recipe is only included for documentation purposes and not maintained any more. The last known ESMValTool version supporting this recipe is version 2.13.0.

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.

Available recipe and diagnostics

Recipes are stored in archive/legacy_recipes

- recipe_wenzel16nat.yml

Diagnostics are stored in archive/legacy_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

User settings

Note

Make sure to run this recipe with the configuration option `max_parallel_tasks: 1`.

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)

   none

   Optional settings (variables)

none

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 dataset (observations)

   Optional settings (variables)

   none

Variables

- co2s (atmos, monthly mean, plev longitude latitude time)
- gpp (land, monthly mean, longitude latitude time)

Observations and reformat scripts

- ESRL: Earth System Research Laboratory, ground-based CO_2 measurements

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

Example plots

5.10 Broken recipe list

5.10.1 Broken recipe list

This table gives an overview of the recipes that are known to have issues. The table is always valid for the latest stable release of ESMValTool. More details can be found in the *broken recipe policy*.

Table 1: Broken recipes

Broken recipe	Affected diagnostics	Broken since release	Problem	GitHub issue
`recipe_kcs.yml`	`kcs/local_resampling.py`	v2.14.0	Diagnostic error (related to datetimes)	Issue #4353

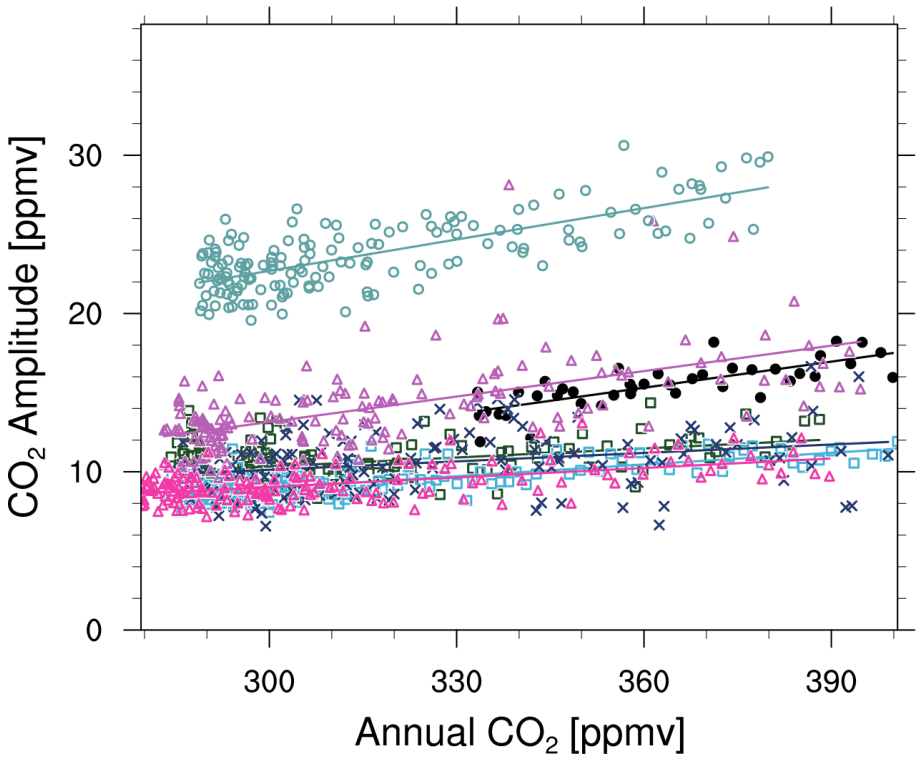


Fig. 332: Comparison of CO_2 seasonal amplitudes for CMIP5 historical simulations and observations showing annual mean atmospheric CO_2 versus the amplitudes of the CO_2 seasonal cycle at Pt. Barrow, Alaska (produced with carbon_co2_cycle.ncl, similar to Fig. 1a from Wenzel et al. (2016)).

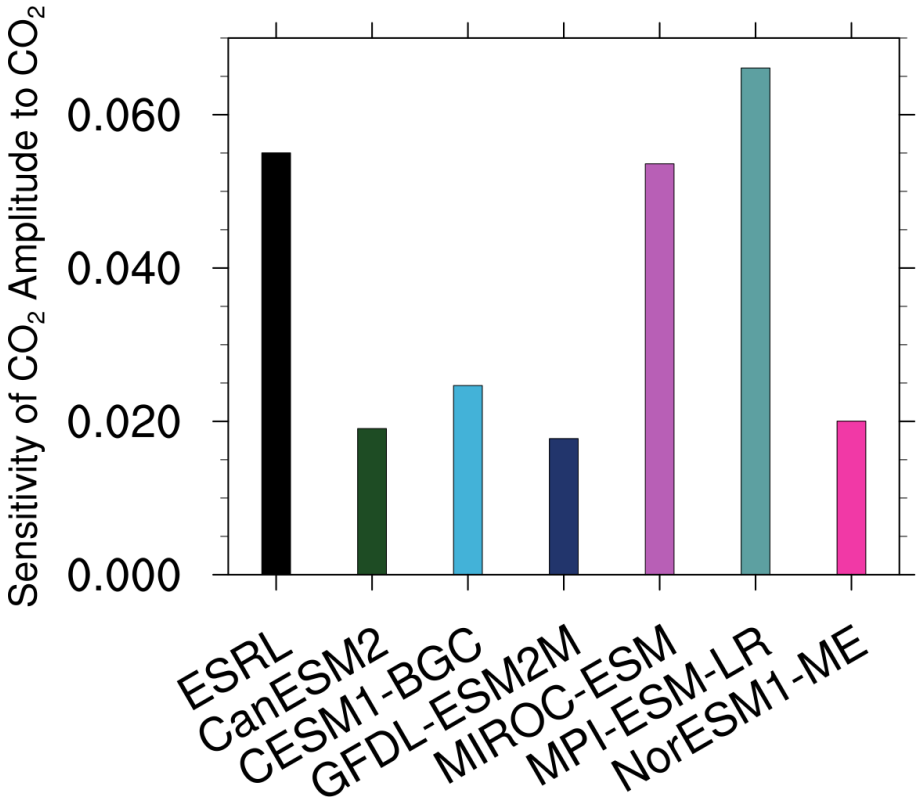


Fig. 333: Barchart showing the gradient of the linear correlations for the comparison of CO_2 seasonal amplitudes for CMIP5 historical for at Pt. Barrow, Alaska (produced with carbon_co2_cycle.ncl, similar to Fig. 1b from Wenzel et al. (2016)).

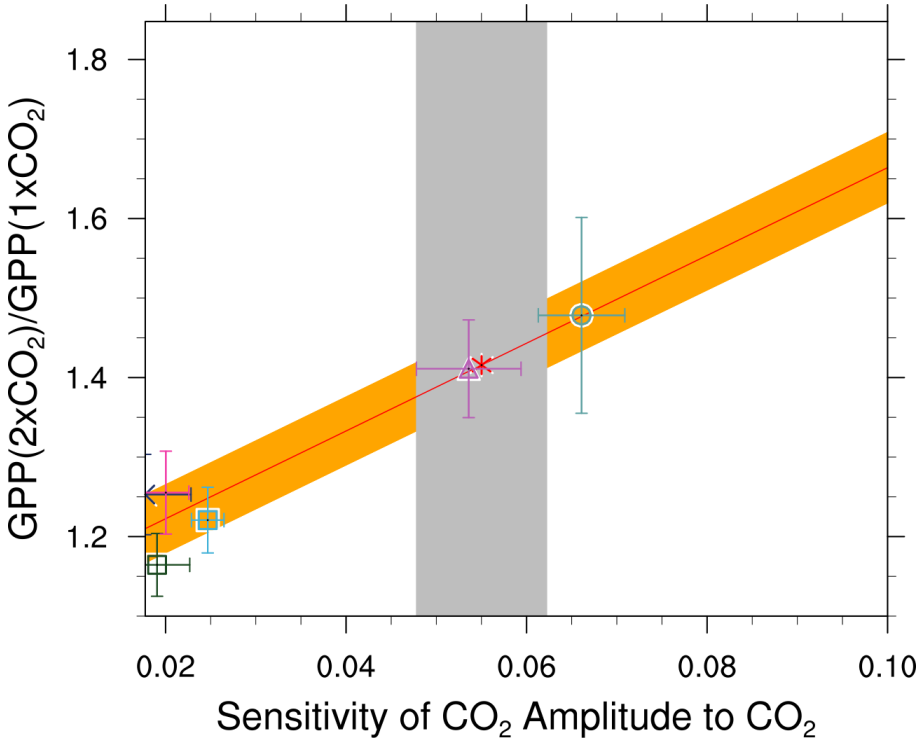


Fig. 334: Emergent constraint on the relative increase of large-scale GPP for a doubling of CO_2, showing the correlations between the sensitivity of the CO_2 amplitude to annual mean CO_2 increases at Pt. Barrow (x-axis) and the high-latitude (60N - 90N) CO_2 fertilization on GPP at $2xCO_2$. The red 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)).

CHAPTER

SIX

ESMVALTOOL CODE API DOCUMENTATION

ESMValTool is mostly used as a command line tool. However, it is also possible to use (parts of) ESMValTool as a library. This section documents the public API of ESMValTool.

6.1 Shared Diagnostic Code

6.1.1 Shared diagnostic script code

Code that is shared between multiple diagnostic scripts.

Classes:

Datasets(cfg)	Class to easily access a recipe's datasets in a diagnostic script.
ProvenanceLogger(cfg)	Open the provenance logger.
Variable(short_name, standard_name, ...)	Variable class containing all relevant information.
Variables([cfg])	Class to easily access a recipe's variables in a diagnostic.

Functions:

apply_supermeans(ctrl, exper, obs_list)	Apply supermeans on data components ie MEAN on time.
extract_variables(cfg[, as_iris])	Extract basic variable information from configuration dictionary.
get_cfg([filename])	Read diagnostic script configuration from settings.yml.
get_control_exper_obs(short_name, ...[, ...])	Get control, exper and obs datasets.
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.
group_metadata(metadata, attribute[, sort])	Group metadata describing preprocessed data by attribute.
run_diagnostic()	Run a Python diagnostic.
save_data(basename, provenance, cfg, cube, ...)	Save the data used to create a plot to file.
save_figure(basename, provenance, cfg[, ...])	Save a figure to file.
select_metadata(metadata, **attributes)	Select specific metadata describing preprocessed data.
sorted_group_metadata(metadata_groups, sort)	Sort grouped metadata.
sorted_metadata(metadata, sort)	Sort a list of metadata describing preprocessed data.
variables_available(cfg, short_names)	Check if data from certain variables is available.

class esmvaltool.diag_scripts.shared.**Datasets**(*cfg*)

Bases: object

Class to easily access a recipe's datasets in a diagnostic script.

Note

This class has been deprecated in version 2.2 and will be removed two minor releases later in version 2.4.

Examples

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)
```

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.

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

- **key** (*str*) – Desired dictionary key.
- ****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.

Example

Use as a context manager:

```
record = {
    'caption': "This is a nice plot.",
    'statistics': ['mean'],
    'domain': ['global'],
    'plot_type': ['zonal'],
    'authors': [
        'first_author',
        'second_author',
    ],
    'references': [
        'author20journal',
    ],
    '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)
```

Methods:

log(filename, record)	Record provenance.

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:

  - ancestors
  - authors
  - caption
  - domain
  - plot_type
  - references
  - statistics

> **Note**
>
> See the provenance documentation for more information.

class esmvaltool.diag_scripts.shared.**Variable**(*short_name*, *standard_name*, *long_name*, *units*)

Bases: *Variable*

Variable class containing all relevant information.

> **Note**
>
> This class has been deprecated in version 2.2 and will be removed two minor releases later in version 2.4.

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*, */ (Positional-only parameter separator (PEP 570))*)

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.

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

class esmvaltool.diag_scripts.shared.**Variables**(*cfg=None*, ***names*)

Bases: object

Class to easily access a recipe's variables in a diagnostic.

Note

This class has been deprecated in version 2.2 and will be removed two minor releases later in version 2.4.

Examples

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')
```

Methods:

add_vars(**names)	Add custom 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*.
units(var)	Access units.
var_name(var)	Access var name.

continues on next page

Table 7 – continued from previous page

vars_available(*args)	Check if given variables are available.

add_vars(***names*)

Add custom 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=None*)

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 cmip_type: optional, CMIP project type (CMIP5 or CMIP6)

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.

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:

```
def main(cfg):
    # Your diagnostic code goes here.
    print(cfg)

if __name__ == '__main__':
    with run_diagnostic() as cfg:
        main(cfg)
```

To prevent the diagnostic script from using the Dask Distributed scheduler, set `no_distributed: true` in the diagnostic script definition in the recipe or in the resulting settings.yml file.

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.**save_data**(*basename*, *provenance*, *cfg*, *cube*, ***kwargs*)

Save the data used to create a plot to file.

Parameters

- **basename** (*str*) – The basename of the file.
- **provenance** (*dict*) – The provenance record for the data.
- **cfg** (*dict*) – Dictionary with diagnostic configuration.
- **cube** (*iris.cube.Cube*) – Data cube to save.
- ****kwargs** – Extra keyword arguments to pass to iris.save.

See also

ProvenanceLogger
For an example provenance record that can be used with this function.

esmvaltool.diag_scripts.shared.**save_figure**(*basename*, *provenance*, *cfg*, *figure=None*, *close=True*, ***kwargs*)

Save a figure to file.

Parameters

- **basename** (*str*) – The basename of the file.
- **provenance** (*dict*) – The provenance record for the figure.
- **cfg** (*dict*) – Dictionary with diagnostic configuration.
- **figure** (*matplotlib.figure.Figure*) – Figure to save.
- **close** (*bool*) – Close the figure after saving.
- ****kwargs** – Keyword arguments to pass to `matplotlib.figure.Figure.savefig`.

See also

ProvenanceLogger
For an example provenance record that can be used with this function.

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
dict 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

Iris helper functions

Convenience functions for iris objects.

Functions:

check_coordinate(cubes, coord_name)	Compare coordinate of cubes and raise error if not identical.
convert_to_iris(dict_)	Change all appearances of short_name to var_name.
get_mean_cube(datasets)	Get mean cube of a list of datasets.
intersect_dataset_coordinates(cubes)	Compare dataset coordinates of cubes and match them if necessary.
iris_project_constraint(projects, input_data)	Create iris.Constraint to select specific projects from data.
prepare_cube_for_merging(cube, cube_label)	Prepare single iris.cube.Cube in order to merge it later.
unify_1d_cubes(cubes, coord_name)	Unify 1D cubes by transforming them to identical coordinates.
unify_time_coord(cube[, target_units])	Unify time coordinate of cube in-place.

esmvaltool.diag_scripts.shared.iris_helpers.**check_coordinate**(*cubes*, *coord_name*)

Compare coordinate of cubes and raise error if not identical.

Parameters

- **cubes** (*iris.cube.CubeList*) – Cubes to be compared.
- **coord_name** (*str*) – Name of the coordinate.

Returns
Points of the coordinate.

Return type
numpy.array

Raises
- **iris.exceptions.CoordinateNotFoundError** – Coordinate `coord_name` is not a coordinate of one of the cubes.
- **ValueError** – Given coordinate differs for the input cubes.

esmvaltool.diag_scripts.shared.iris_helpers.**convert_to_iris**(*dict_*)

Change all appearances of `short_name` to `var_name`.

Parameters
dict (*dict*) – Dictionary to convert.

Returns
Converted dictionary.

Return type
dict

Raises
KeyError – `dict` contains keys```'short_name'``` **and** `'var_name'`.

esmvaltool.diag_scripts.shared.iris_helpers.**get_mean_cube**(*datasets*)

Get mean cube of a list of datasets.

Parameters
datasets (*list of dict*) – List of datasets (given as metadata `dict`).

Returns
Mean cube.

Return type
iris.cube.Cube

esmvaltool.diag_scripts.shared.iris_helpers.**intersect_dataset_coordinates**(*cubes*)

Compare dataset coordinates of cubes and match them if necessary.

Use intersection of coordinate 'dataset' of all given cubes and remove elements which are not given in all cubes.

Parameters
cubes (*iris.cube.CubeList*) – Cubes to be compared.

Returns
Transformed cubes.

Return type
iris.cube.CubeList

Raises
- **iris.exceptions.CoordinateNotFoundError** – Coordinate `dataset` is not a coordinate of one of the cubes.
- **ValueError** – At least one of the cubes contains a `dataset` coordinate with duplicate elements or the cubes do not share common elements.

esmvaltool.diag_scripts.shared.iris_helpers.**iris_project_constraint**(*projects*, *input_data*, *negate=False*)

Create `iris.Constraint` to select specific projects from data.

Parameters

- **projects** (*list of str*) – Projects to be selected.
- **input_data** (*list of dict*) – List of dataset metadata used to extract all relevant datasets belonging to given `projects`.
- **negate** (*bool, optional (default: False)*) – Negate constraint (`False`: select all elements that fit `projects`, *True* `: select all elements that do **not** fit `projects`).

Returns
constraint for coordinate `dataset`.

Return type
iris.Constraint

esmvaltool.diag_scripts.shared.iris_helpers.**prepare_cube_for_merging**(*cube*, *cube_label*)

Prepare single `iris.cube.Cube` in order to merge it later.

Parameters

- **cube** (*iris.cube.Cube*) – Cube to be pre-processed.
- **cube_label** (*str*) – Label for the new scalar coordinate `cube_label`.

esmvaltool.diag_scripts.shared.iris_helpers.**unify_1d_cubes**(*cubes*, *coord_name*)

Unify 1D cubes by transforming them to identical coordinates.

Use union of all coordinates as reference and transform other cubes to it by adding missing values.

Parameters

- **cubes** (*iris.cube.CubeList*) – Cubes to be processed.
- **coord_name** (*str*) – Name of the coordinate.

Returns
Transformed cubes.

Return type
iris.cube.CubeList

Raises
ValueError – Cubes are not 1D, coordinate name differs or not all cube coordinates are subsets of longest coordinate.

esmvaltool.diag_scripts.shared.iris_helpers.**unify_time_coord**(*cube*, *target_units='days since 1850-01-01 00:00:00'*)

Unify time coordinate of cube in-place.

Parameters

- **cube** (*iris.cube.Cube*) – Cube whose time coordinate is transformed in-place.
- **target_units** (*str or cf_units.Unit, optional*) – Target time units.

Raises
iris.exceptions.CoordinateNotFoundError – Cube does not contain coordinate `time`.

Plotting

Module that provides common plot functions.

Functions:

get_dataset_style(dataset[, style_file])	Retrieve the style information for the given dataset.
get_path_to_mpl_style([style_file])	Get path to matplotlib style file.
global_contourf(cube[, cbar_center, ...])	Plot global filled contour plot.
global_pcolormesh(cube[, cbar_center, ...])	Plot global color mesh.
multi_dataset_scatterplot(x_data, y_data, ...)	Plot a multi dataset scatterplot.
quickplot(cube, plot_type[, filename])	Plot a cube using one of the iris.quickplot functions.
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.**global_contourf**(*cube*, *cbar_center=None*, *cbar_label=None*, *cbar_range=None*, *cbar_ticks=None*, ***kwargs*)

Plot global filled contour plot.

> **Note**
>
> This is only possible if the cube is 2D with dimensional coordinates *latitude* and *longitude*.

Parameters

- **cube** (*iris.cube.Cube*) – Cube to plot.
- **cbar_center** (*float, optional*) – Central value for the colormap, useful for diverging colormaps. Can only be used if `cbar_range` is given.
- **cbar_label** (*str, optional*) – Label for the colorbar.
- **cbar_range** (*list of float, optional*) – Range of the colorbar (first and second list element) and number of distinct colors (third element). See `numpy.linspace`.
- **cbar_ticks** (*list, optional*) – Ticks for the colorbar.
- ****kwargs** – Keyword argument for `iris.plot.contourf()`.

Returns

Plot object.

Return type

matplotlib.contour.QuadContourSet

Raises

- **iris.exceptions.CoordinateNotFoundError** – Input `iris.cube.Cube` does not contain the necessary dimensional coordinates `'latitude'` and `'longitude'`.
- **ValueError** – Input `iris.cube.Cube` is not 2D.

esmvaltool.diag_scripts.shared.plot.**global_pcolormesh**(*cube*, *cbar_center=None*, *cbar_label=None*, *cbar_ticks=None*, ***kwargs*)

Plot global color mesh.

> **Note**
>
> This is only possible if the cube is 2D with dimensional coordinates *latitude* and *longitude*.

Parameters

- **cube** (*iris.cube.Cube*) – Cube to plot.
- **cbar_center** (*float, optional*) – Central value for the colormap, useful for diverging colormaps. Can only be used if `vmin` and `vmax` are given.
- **cbar_label** (*str, optional*) – Label for the colorbar.
- **cbar_ticks** (*list, optional*) – Ticks for the colorbar.
- ****kwargs** – Keyword argument for `iris.plot.pcolormesh()`.

Returns
Plot object.

Return type
matplotlib.contour.QuadContourSet

Raises

- **iris.exceptions.CoordinateNotFoundError** – Input `iris.cube.Cube` does not contain the necessary dimensional coordinates `'latitude'` and `'longitude'`.
- **ValueError** – Input `iris.cube.Cube` is not 2D.

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*, *plot_type*, *filename=None*, ***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.

6.2 Diagnostic Scripts

6.2.1 Drought Diagnostics

The droughts module contains diagnostics for calculation and plotting of drought indices and metrics. A general overview is given on the *Droughts documentation page*.

Diagnostic scripts

Drought Metrics following Martin (2018)

Compares SPI/SPEI data from models with observations/reanalysis.

Description

This diagnostic applies drought characteristics based on Martin (2018) to data produced by spei.R. These characteristics and differences to a reference dataset or between different time periods are plotted for each dataset and multi-model mean. The calculated frequency, duration and severity of drought events are saved to netcdf files for further use. It expects multiple datasets for a particular index as input. The reference dataset can be specified with `reference_dataset` and is not part of the multi-model mean.

Note

Previous Version: With ESMValTool v2.12 and previous, multiple collect_drought_*.py diagnostics and a collect_drought_func.py diagnostic existed in the *droughtindex* folders. Those have been archived and replaced by this diagnostic in v2.13.

Configuration options

indexname: str
The indexname is used to generate filenames, plot titles and captions. Should be `SPI` or `SPEI`.

reference_dataset: str
Dataset name to use for comparison (excluded from MMM). With `compare_intervals=True` this option has no effect.

threshold: float, optional (default: -2.0)
Threshold for an event to be considered as drought.

compare_intervals: bool, false
If true, start and end of the time periods are compared instead of models and reference. The lengths of start and end period is given by `comparison_period`.

comparison_period: int
Number of years from start and end of the full period to be compared. Should be < (end_year - start_year)/2. If `compare_intervals=False` this option has no effect.

plot_models: bool, false
Save plots for each individual model, in addition to multi-model mean.

start_year: int
This option is used to select the time slices for comparison if `compare_intervals=True`.

end_year: int
This option is used to select the time slices for comparison if `compare_intervals=True`.

Difference Maps

Plot relative and absolute differences between two time intervals.

A global map is plotted for each dataset with an index (must be unique). The map shows the difference of the first and last N years (N = comparison_period). For multiple datasets a multi-model mean is calculated by default. This can be disabled using *plot_mmm: False*. To plot only mmm and skip maps for individual datasets use *plot_models: False*. The diagnostic is applied to each variable by default, but for single variables another meta key can be chosen for grouping like *group_by: project* to treat observations and models separately. The produced maps can be clipped to non polar landmasses (220, 170, -55, 90) with *clip_land: True*.

Configuration options in recipe

basename: str, optional
Format string for the plot filename. Can use meta keys and diffmap_metric. For multi-model mean the dataset will be set to "MMM". Data will be saved as same name with .nc extension. By default: "{short_name}_{exp}_{diffmap_metric}_{dataset}"

clip_land: bool, optional (default: False)
Clips map plots to non polar land area (220, 170, -55, 90).

comparison_period: int, optional (default: 10)
Number of years to compare (first and last N years). Must be less or equal to half of the total time period.

filters: dict or list, optional
Filter for metadata keys to select datasets. Only datasets with matching values will be processed. This can be useful, if ancestors or preprocessed data is available, that should not be processed by the diagnostic. If a list of dicts is given, all datasets matching any of the filters will be considered. By default None.

group_by: str, optional (default: short_name)
Meta key to loop over for multiple datasets.

metrics: list, optional
List of metrics to calculate and plot. For the difference ("percent" and "diff") the mean over two comparison periods ("first" and "last") is calculated. The "total" periods mean can be calculated and plotted as well. By default ["first", "last", "diff", "total", "percent"]

mdtol: float, optional (default: 0.5)
Tolerance for missing data in multi-model mean calculation. 0 means no missing data is allowed. At 1 the mean is calculated if any data is available.

plot_kwargs: dict, optional
Kwargs passed to diag_scripts.shared.plot.global_contourf function. The "cbar_label" parameter is formatted with meta keys. So placeholders like "{short_name}" or "{units}" can be used. By default {"cmap": "RdYlBu", "extend": "both"}

plot_kwargs_overwrite: list, optional (default: [])
List of plot_kwargs dicts for specific metrics (diff, first, latest, total) and group_by values (ie. pr, tas for group_by: short_name). *group* and *metric* can either be strings or lists of strings to be applied to all matching plots. Leave any of them empty to apply to all. All other given keys are applied to the plot_kwargs dict for this plot. Settings will be applied in order of the list, so later entries can overwrite previous ones.

plot_mmm: bool, optional (default: True)
Calculate and plot the average over all datasets.

plot_models: bool, optional (default: True)
Plot maps for each dataset.

strip_plots: bool, optional (default: False)
Removes titles, margins and colorbars from plots (to use them in panels).

titles: dict, optional
Customize plot titles for different metrics. Possible dict keys are "first", "last", "trend", "diff", "total", "percent". The values are formatted using meta data. Placeholders like "{short_name}" can be used. By default {"first": "Mean Historical", "last": "Mean Future", "trend": "Future - Historical", "diff": "Future - Historical", "total": "Mean Full Period", "percent": "Relative Change"}.

6.2.2 Emergent constraints diagnostics

This module provides various tools to evaluate emergent constraints for arbitrary input variables.

Examples

- *Emergent constraints for equilibrium climate sensitivity*
- *Emergent constraint on equilibrium climate sensitivity from global temperature variability*
- *Emergent constraints on equilibrium climate sensitivity in CMIP5: do they hold for CMIP6?*

Diagnostic scripts

Emergent constraint on ECS from global temperature variability

Diagnostic script to reproduce emergent constraint of Cox et al. (2018).

Description

Plot equilibrium climate sensitivity ECS vs. temperature variability metric to establish an emergent relationship for ECS.

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

confidence_level: float, optional (default: 0.66)
Confidence level for ECS error estimation.

Calculation of emergent constraints on ECS

Diagnostic script to calculate various emergent constraints for ECS.

Description

Calculate the X axis of various emergent constraints for the equilibrium climate sensitivity (ECS).

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

diag: str
Emergent constraint to calculate (must be one of `'brient_shal'`, `'su'`, `'volodin'`, `'zhai'`).

metric: str, optional (default: 'regression_slope')
Metric to measure model error. Only relevant for Su et al. (2014) constraint. Must be one of `'regression_slope'`, `'correlation_coefficient'`.

n_jobs: int, optional (default: 1)
Maximum number of jobs spawned by this class.

output_attributes: dict, optional
Write additional attributes to netcdf files.

pattern: str, optional
Pattern matched against ancestor file names.

savefig_kwargs: dict
Keyword arguments for `matplotlib.pyplot.savefig()`.

seaborn_settings: dict
Options for `seaborn.set_theme()` (affects all plots).

Evaluate multiple emergent constraints simultaneously

Diagnostic script to evaluate multiple emergent constraints simultaneously.

Description

Establish multiple emergent constraints for arbitrary input variables and an arbitrary target variable. All input datasets need to be one-dimensional and must include a coordinate `'dataset'` or `'model'` (thus, the data describes a single scalar value for each dataset). All input datasets must be marked with a `var_type` (either `feature`, `label`, `prediction_input` or `prediction_input_error`) and a `tag`, which describes the type of data. This diagnostic supports only a single `tag` for `label` and an arbitrary number of `tag` s for `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'`), multiple datasets are considered as `prediction_input` (in this case `'OBS1'` and `'OBS2'`).

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

additional_data: list of dict, optional
Additional datasets given as list of metadata.

all_data_label: str, optional (default: 'all')
Label used in plots when all input data is considered. Only relevant if `group_by` is not used.

combine_groups: bool, optional (default: False)
Add results to plots for data generated by combining the data of all individual groups.

confidence_level: float, optional (default: 0.66)
Confidence level for estimation of target variable.

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

ignore_patterns: list of str, optional
Patterns matched against ancestor files. Those files are ignored.

merge_identical_pred_input: bool, optional (default: True)
Use identical prediction_input values as single value.

numbers_as_markers: bool, optional (default: False)
Use numbers as markers in scatterplots.

patterns: list of str, optional
Patterns matched against ancestor files.

plot_regression_line_mean: bool, optional (default: False)
Plot means of regression lines in scatterplots.

read_external_file: str, optional
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: dict
Keyword arguments for `matplotlib.pyplot.savefig()`.

seaborn_settings: dict
Options for `seaborn.set_theme()` (affects all plots).

Evaluate single emergent constraint

Diagnostic script to evaluate a single emergent constraint.

Description

Establish a single emergent constraint for an arbitrary input variable and an arbitrary target variable. All input datasets need to be one-dimensional and must include a coordinate `'dataset'` or `'model'` (thus, the data describes a single scalar value for each dataset). All input datasets must be marked with a `var_type` (either `feature`, `label`, `prediction_input` or `prediction_input_error`) and a `tag`, which describes the type of 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'`), multiple datasets are considered as `prediction_input` (in this case `'OBS1'` and `'OBS2'`).

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

additional_data: list of dict, optional
Additional datasets given as list of metadata.

all_data_label: str, optional (default: 'all')
Label used in plots when all input data is considered. Only relevant if `group_by` is not used.

combine_groups: bool, optional (default: False)
Add results to plots for data generated by combining the data of all individual groups.

confidence_level: float, optional (default: 0.66)
Confidence level for estimation of target variable.

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

ignore_patterns: list of str, optional
Patterns matched against ancestor files. Those files are ignored.

merge_identical_pred_input: bool, optional (default: True)
Use identical prediction_input values as single value.

numbers_as_markers: bool, optional (default: False)
Use numbers as markers in scatterplots.

patterns: list of str, optional
Patterns matched against ancestor files.

plot_regression_line_mean: bool, optional (default: False)
Plot means of regression lines in scatterplots.

read_external_file: str, optional
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: dict
Keyword arguments for `matplotlib.pyplot.savefig()`.

seaborn_settings: dict
Options for `seaborn.set_theme()` (affects all plots).

Auxiliary scripts

Auxiliary functions for emergent constraints scripts

Convenience functions for emergent constraints diagnostics.

Functions:

cdf(data, pdf)	Calculate cumulative distribution function for a 1-dimensional PDF.
check_metadata(metadata[, allowed_var_types])	Check metadata.
combine_groups(groups)	Combine `list` of groups to a single `str`.
constraint_info_array(x_data, y_data, ...[, ...])	Get array with all relevant parameters of emergent constraint.
create_simple_scatterplot(x_data, y_data, ...)	Create simple scatterplot of an emergent relationship (without saving).
export_csv(data_frame, attributes, basename, cfg)	Export CSV file.
get_caption(attributes, feature, label[, group])	Construct caption from plotting attributes for (feature, label) pair.
get_colors(cfg[, groups])	Get color palette.
get_constraint(x_data, y_data, obs_mean, obs_std)	Get constraint on target variable.

continues on next page

Table 10 – continued from previous page

get_constraint_from_df(training_data, ...[, ...])	Get constraint on target variable from pandas.DataFrame.
get_groups(training_data[, add_combined_group])	Extract groups from training data.
get_input_data(cfg)	Extract input data.
get_input_files(cfg[, patterns, ignore_patterns])	Get input files.
get_provenance_record(attributes, tags, **kwargs)	Get provenance record.
get_xy_data_without_nans(data_frame, ...)	Get (X, Y) data for `(feature, label)` combination without nans.
pandas_object_to_cube(pandas_object[, ...])	Convert pandas object to iris.cube.Cube.
plot_individual_scatterplots(training_data, ...)	Plot individual scatterplots for the different groups.
plot_merged_scatterplots(training_data, ...)	Plot merged scatterplots (all groups in one plot).
plot_target_distributions(training_data, ...)	Plot distributions of target variable for every feature.
regression_line(x_data, y_data[, n_points])	Return x and y coordinates of the regression line (mean and error).
set_plot_appearance(axes, attributes, **kwargs)	Set appearance of a plot.
standard_prediction_error(x_data, y_data)	Return a function to calculate standard prediction error.
target_pdf(x_data, y_data, obs_mean, obs_std)	Calculate probability density function (PDF) for target variable.

esmvaltool.diag_scripts.emergent_constraints.**cdf**(*data*, *pdf*)

Calculate cumulative distribution function for a 1-dimensional PDF.

Parameters

- **data** (*numpy.ndarray*) – Data points (1D array).
- **pdf** (*numpy.ndarray*) – Corresponding probability density function (PDF).

Returns

Corresponding cumulative distribution function (CDF).

Return type

numpy.ndarray

esmvaltool.diag_scripts.emergent_constraints.**check_metadata**(*metadata*, *allowed_var_types=None*)

Check metadata.

Parameters

- **metadata** (*dict*) – Metadata to check.
- **allowed_var_types** (*list of str, optional*) – Allowed var_types, defaults to ALLOWED_VAR_TYPES.

Raises

- **KeyError** – Metadata does not contain necessary keys `'var_type'` and `'tag'`.
- **ValueError** – Got invalid value for key `'var_type'`.

esmvaltool.diag_scripts.emergent_constraints.**combine_groups**(*groups*)

Combine `list` of groups to a single `str`.

Parameters

groups (*list of str*) – List of group names.

Returns

Combined `str`.

Return type
str

esmvaltool.diag_scripts.emergent_constraints.**constraint_info_array**(*x_data*, *y_data*, *obs_mean*, *obs_std*, *n_points=1000*, *necessary_p_value=None*)

Get array with all relevant parameters of emergent constraint.

Parameters

- **x_data** (*numpy.ndarray*) – X data of the emergent constraint.
- **y_data** (*numpy.ndarray*) – Y data of the emergent constraint.
- **obs_mean** (*float*) – Mean of observational data.
- **obs_std** (*float*) – Standard deviation of observational data.
- **n_points** (*int, optional (default: 1000)*) – Number of sampled points for PDF of target variable.
- **necessary_p_value** (*float, optional*) – If given, replace constrained mean and standard deviation with unconstrained values when *p*-value of emergent relationship is greater than the given necessary *p*-value.

Returns

Array of shape (8,) with the elements:

0. Constrained mean of target variable.
1. Constrained standard deviation of target variable.
2. Unconstrained mean of target variable.
3. Unconstrained standard deviation of target variable.
4. Slope of emergent relationship.
5. Intercept of emergent relationship.
6. Correlation coefficient *r* of emergent relationship.
7. *p*-value of emergent relationship.

Return type
numpy.ndarray

esmvaltool.diag_scripts.emergent_constraints.**create_simple_scatterplot**(*x_data*, *y_data*, *obs_mean*, *obs_std*)

Create simple scatterplot of an emergent relationship (without saving).

Parameters

- **x_data** (*numpy.ndarray*) – X data of the emergent constraint.
- **y_data** (*numpy.ndarray*) – Y data of the emergent constraint.
- **obs_mean** (*float*) – Mean of observational data.
- **obs_std** (*float*) – Standard deviation of observational data.

esmvaltool.diag_scripts.emergent_constraints.**export_csv**(*data_frame*, *attributes*, *basename*, *cfg*, *tags=None*)

Export CSV file.

Parameters

- **data_frame** (*pandas.DataFrame*) – Data to export.
- **attributes** (*dict*) – Plot attributes for the different features and the label data. Used to retrieve provenance information.
- **basename** (*str*) – Basename for the name of the file.
- **cfg** (*dict*) – Recipe configuration.
- **tags** (*iterable of str, optional*) – Tags for which provenance information should be retrieved (using `attributes`). If not specified, use (last level of) columns of the given `data_frame`.

Returns
Path to the new CSV file.

Return type
str

esmvaltool.diag_scripts.emergent_constraints.**get_caption**(*attributes*, *feature*, *label*, *group=None*)

Construct caption from plotting attributes for (feature, label) pair.

Parameters

- **attributes** (*dict*) – Plot attributes.
- **feature** (*str*) – Feature.
- **label** (*str*) – Label.
- **group** (*str, optional*) – Group.

Returns
Caption.

Return type
str

Raises
KeyError – `attributes` does not include necessary keys.

esmvaltool.diag_scripts.emergent_constraints.**get_colors**(*cfg*, *groups=None*)

Get color palette.

Parameters

- **cfg** (*dict*) – Recipe configuration.
- **groups** (*list, optional*) – Use to check whether color for combining groups has to be added.

Returns
List of colors that can be used for `matplotlib`.

Return type
list

esmvaltool.diag_scripts.emergent_constraints.**get_constraint**(*x_data*, *y_data*, *obs_mean*, *obs_std*, *confidence_level=0.66*)

Get constraint on target variable.

Parameters

- **x_data** (*numpy.ndarray*) – X data of the emergent constraint.

- **y_data** (*numpy.ndarray*) – Y data of the emergent constraint.
- **obs_mean** (*float*) – Mean of observational data.
- **obs_std** (*float*) – Standard deviation of observational data.
- **confidence_level** (*float, optional (default: 0.66)*) – Confindence level to estimate the range of the target variable.

Returns
Lower confidence limit, best estimate and upper confidence limit of target variable.

Return type
tuple of float

esmvaltool.diag_scripts.emergent_constraints.**get_constraint_from_df**(*training_data*, *pred_input_data*, *confidence_level=0.66*)

Get constraint on target variable from pandas.DataFrame.

Parameters

- **training_data** (*pandas.DataFrame*) – Training data (features, label).
- **pred_input_data** (*pandas.DataFrame*) – Prediction input data (mean and error).
- **confidence_level** (*float, optional (default: 0.66)*) – Confindence level to estimate the range of the target variable.

Returns
Lower confidence limit, best estimate and upper confidence limit of target variable.

Return type
tuple of float

esmvaltool.diag_scripts.emergent_constraints.**get_groups**(*training_data*, *add_combined_group=False*)

Extract groups from training data.

Parameters

- **training_data** (*pandas.DataFrame*) – Training data (features, label).
- **add_combined_group** (*bool, optional (default: False)*) – Add combined group of all other groups at the beginning of the returned list.

Returns
Groups.

Return type
list of str

esmvaltool.diag_scripts.emergent_constraints.**get_input_data**(*cfg*)

Extract input data.

Return training data, prediction input data and corresponding attributes.

Parameters
cfg (*dict*) – Recipe configuration.

Returns
A tuple containing the training data (pandas.DataFrame), the prediction input data (pandas.DataFrame) and the corresponding attributes (dict).

Return type
tuple

esmvaltool.diag_scripts.emergent_constraints.**get_input_files**(*cfg*, *patterns=None*, *ignore_patterns=None*)

Get input files.

Parameters

- **cfg** (*dict*) – Recipe configuration.
- **patterns** (*list of str, optional*) – Use only ancestor files that match these patterns as input files.
- **ignore_patterns** (*list of str, optional*) – Ignore input files that match these patterns.

Returns
Input files.

Return type
list of str

esmvaltool.diag_scripts.emergent_constraints.**get_provenance_record**(*attributes*, *tags*, ***kwargs*)

Get provenance record.

Parameters

- **attributes** (*dict*) – Plot attributes. All provenance keys need to start with `'provenance_'`.
- **tags** (*list of str*) – Tags used to retrieve data from the `attributes` `dict`, i.e. features and/or label.
- ****kwargs** (*Keyword arguments*) – Additional `key:value` pairs directly passed to the provenance record `dict`. All values may include the format strings `{feature}` and `{label}`.

Returns
Provenance record.

Return type
dict

esmvaltool.diag_scripts.emergent_constraints.**get_xy_data_without_nans**(*data_frame*, *feature*, *label*)

Get (X, Y) data for `(feature, label)` combination without nans.

Parameters

- **data_frame** (*pandas.DataFrame*) – Training data.
- **feature** (*str*) – Name of the feature data.
- **label** (*str*) – Name of the label data.

Returns
Tuple containing a `pandas.DataFrame` for the X axis (feature) and a `pandas.DataFrame` for the Y axis (label) without missing values.

Return type
tuple

esmvaltool.diag_scripts.emergent_constraints.**pandas_object_to_cube**(*pandas_object*, *index_droplevel=None*, *columns_droplevel=None*, ***kwargs*)

Convert pandas object to `iris.cube.Cube`.

Parameters

- **pandas_object** (*pandas.DataFrame or pandas.Series*) – Data to convert.
- **index_droplevel** (*int or list of int, optional*) – Drop levels of index if not `None`.
- **columns_droplevel** (*int or list of int, optional*) – Drop levels of columns if not `None`. Can only be used if `pandas_object` is a `pandas.DataFrame`.
- ****kwargs** (*Keyword arguments*) – Keyword arguments used for the cube metadata, e.g. `standard_name`, `var_name`, etc.

Returns
Data cube.

Return type
iris.cube.Cube

Raises
TypeError – `columns_droplevel` is used when `pandas_object` is not a `pandas.DataFrame`.

esmvaltool.diag_scripts.emergent_constraints.**plot_individual_scatterplots**(*training_data*, *pred_input_data*, *attributes*, *basename*, *cfg*)

Plot individual scatterplots for the different groups.

Plot scatterplots for all pairs of `(feature, label)` data (Separate plot for each group).

Parameters

- **training_data** (*pandas.DataFrame*) – Training data (features, label).
- **pred_input_data** (*pandas.DataFrame*) – Prediction input data (mean and error).
- **attributes** (*dict*) – Plot attributes for the different features and the label data.
- **basename** (*str*) – Basename for the name of the file.
- **cfg** (*dict*) – Recipe configuration.

esmvaltool.diag_scripts.emergent_constraints.**plot_merged_scatterplots**(*training_data*, *pred_input_data*, *attributes*, *basename*, *cfg*)

Plot merged scatterplots (all groups in one plot).

Plot scatterplots for all pairs of `(feature, label)` data (all groups in one plot).

Parameters

- **training_data** (*pandas.DataFrame*) – Training data (features, label).
- **pred_input_data** (*pandas.DataFrame*) – Prediction input data (mean and error).
- **attributes** (*dict*) – Plot attributes for the different features and the label data.

- **basename** (*str*) – Basename for the name of the file.
- **cfg** (*dict*) – Recipe configuration.

esmvaltool.diag_scripts.emergent_constraints.**plot_target_distributions**(*training_data*, *pred_input_data*, *attributes*, *basename*, *cfg*)

Plot distributions of target variable for every feature.

Parameters

- **training_data** (*pandas.DataFrame*) – Training data (features, label).
- **pred_input_data** (*pandas.DataFrame*) – Prediction input data (mean and error).
- **attributes** (*dict*) – Plot attributes for the different features and the label data.
- **basename** (*str*) – Basename for the name of the file.
- **cfg** (*dict*) – Recipe configuration.

esmvaltool.diag_scripts.emergent_constraints.**regression_line**(*x_data*, *y_data*, *n_points=1000*)

Return x and y coordinates of the regression line (mean and error).

Parameters

- **x_data** (*numpy.ndarray*) – X data used to fit the linear regression.
- **y_data** (*numpy.ndarray*) – Y data used to fit the linear regression.
- **n_points** (*int, optional (default: 1000)*) – Number of points for the regression lines.

Returns

numpy.ndarray s for the keys `'x'`, `'y'`, `'y_minus_err'`, `'y_plus_err'`, `'slope'`, `'intercept'`, `'pvalue'` and `'rvalue'`.

Return type

dict

esmvaltool.diag_scripts.emergent_constraints.**set_plot_appearance**(*axes*, *attributes*, ***kwargs*)

Set appearance of a plot.

Parameters

- **axes** (*matplotlib.axes.Axes*) – Matplotlib Axes object which contains the plot.
- **attributes** (*dict*) – Plot attributes.
- ****kwargs** (*Keyword arguments*) – Keyword arguments of the form `plot_option=tag` where `plot_option` is something like `plot_title`, `plot_xlabel`, `plot_xlim`, etc. and `tag` a key for the plot attributes `dict` that describes which attributes should be considered for that `plot_option`.

esmvaltool.diag_scripts.emergent_constraints.**standard_prediction_error**(*x_data*, *y_data*)

Return a function to calculate standard prediction error.

The standard prediction error of a linear regression is the error when predicting a data point which was not used to fit the regression line in the first place.

Parameters

- **x_data** (*numpy.ndarray*) – X data used to fit the linear regression.

- **y_data** (*numpy.ndarray*) – Y data used to fit the linear regression.

Returns
Function that takes a `float` as single argument (representing the X value of a new data point) and returns the standard prediction error for that.

Return type
callable

esmvaltool.diag_scripts.emergent_constraints.**target_pdf**(*x_data*, *y_data*, *obs_mean*, *obs_std*, *n_points=1000*, *necessary_p_value=None*)

Calculate probability density function (PDF) for target variable.

Parameters

- **x_data** (*numpy.ndarray*) – X data of the emergent constraint.
- **y_data** (*numpy.ndarray*) – Y data of the emergent constraint.
- **obs_mean** (*float*) – Mean of observational data.
- **obs_std** (*float*) – Standard deviation of observational data.
- **n_points** (*int, optional (default: 1000)*) – Number of sampled points for PDF of target variable.
- **necessary_p_value** (*float, optional*) – If given, return unconstrained PDF (using Gaussian distribution with unconstrained mean and standard deviation) when *p*-value of emergent relationship is greater than the given necessary *p*-value.

Returns
x and y values for the PDF.

Return type
tuple of numpy.ndarray

6.2.3 Calculating and displaying multiple representations of the CH4 lifetime

Lifetime diagnostic to show multiple datasets in one plot.

Description

This diagnostic can be used to visualize lifetime of multiple datasets in one plot.

Input data needs to be 4D (time, air_pressure/atmosphere_hybrid_sigma_pressure_coordinate, latitude, longitude).

For some plot types, a reference dataset can be defined. For this, use the facet `reference_for_lifetime_diags: true` in the definition of the dataset in the recipe. Note that at most one reference dataset is supported.

Currently supported plot types (use the option `plots` to specify them):

- Time series (plot type `timeseries`): all datasets are plotted in one single figure.

Author

Franziska Winterstein (DLR, Germany)

Configuration options in recipe

facet_used_for_labels: str, optional (default: 'dataset')
Facet used to label different datasets in plot titles and legends. For example, `facet_used_for_labels:`

`dataset` will use dataset names in plot titles and legends; `facet_used_for_labels: exp` will use experiments in plot titles and legends. In addition, `facet_used_for_labels` is used to select the correct `plot_kwargs` for the different datasets (see configuration options for the different plot types below).

figure_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.figure()`. By default, uses `constrained_layout: true`.

m_air: float, optional (default: 28.970)
Molar mass of air in g/mol.

molarmass: float
Molar mass of reactant in g/mol.

oxidant: dict
Oxidant reaction rates for chemical species, e.g., {"oh": {"A": 1.85e-20, "ER": 987.0, "b": 2.82}}.

plots: dict, optional
Plot types plotted by this diagnostic (see list above). Dictionary keys must be `timeseries`. Dictionary values are dictionaries used as options for the corresponding plot. The allowed options for the different plot types are given below.

plot_filename: str, optional
Filename pattern for the plots. Defaults to `{plot_type}_{real_name}_{dataset}_{mip}_{exp}_{ensemble}`. All tags (i.e., the entries in curly brackets, e.g., `{dataset}`, are replaced with the corresponding tags).

plot_folder: str, optional
Path to the folder to store figures. Defaults to `{plot_dir}/../../{dataset}/{exp}/{modeling_realm}/{real_name}`. All tags (i.e., the entries in curly brackets, e.g., `{dataset}`, are replaced with the corresponding tags). `{plot_dir}` is replaced with the default ESMValTool plot directory (i.e., `output_dir/plots/diagnostic_name/script_name/`, see Output).

reactant: str
Name of reactant, e.g., "CH4".

regions: list of str, optional (default: ["TROP"])
Regions considered for line plots. Currently, only "TROP" (troposphere) and "STRA" (stratosphere) are supported.

savefig_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.savefig()`. By default, uses `bbox_inches: tight, dpi: 300, orientation: landscape`.

seaborn_settings: dict, optional
Options for `seaborn.set_theme()` (affects all plots). By default, uses `style: ticks`.

units: str, optional (default: "years")
Target units of the lifetime.

weight_type: str
Type of weighting. If "mass CH4", use CH4 mass as weights. Otherwise, use equal weights.

Configuration options for plot type `timeseries`

annual_mean: str, optional (default: False)
Optional switch to turn on annual means to be displayed 'only' or additional 'both' to the original timeseries. If not set or set to `False` only the original timeseries is shown.

annual_mean_kwargs: dict, optional
Optional keyword arguments for `iris.plot.plot()` for plotting annual means. These keyword arguments

update (and potentially overwrite) the `plot_kwargs` for the annual mean plots. Use `annual_mean_kwargs` to not show annual means.

by_timestep: bool, optional (default: False)
Calculate lifetime for each time step individually. Slower, but less memory-intensive.

display_mean: bool, optional (default: False)
Show mean over entire period in plot label.

gridline_kwargs: dict, optional
Optional keyword arguments for grid lines. By default, `color: lightgrey, alpha: 0.5` are used. Use `gridline_kwargs: false` to not show grid lines.

legend_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.legend()`. Use `legend_kwargs: false` to not show legends.

plot_kwargs: dict, optional
Optional keyword arguments for `iris.plot.plot()`. Dictionary keys are elements identified by `facet_used_for_labels` or `default`, e.g., `CMIP6` if `facet_used_for_labels: project` or `historical` if `facet_used_for_labels: exp`. Dictionary values are dictionaries used as keyword arguments for `iris.plot.plot()`. String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., `{project}`, `{short_name}`, `{exp}`. Examples: `default: {linestyle: '-', label: '{project}'}, CMIP6: {color: red, linestyle: '--'}, OBS: {color: black}`.

pyplot_kwargs: dict, optional
Optional calls to functions of `matplotlib.pyplot`. Dictionary keys are functions of `matplotlib.pyplot`. Dictionary values are used as single argument for these functions. String arguments can include facets in curly brackets which will be derived from the datasets plotted in the corresponding plot, e.g., `{short_name}`, `{exp}`. Facets like `{project}` that vary between the different datasets will be transformed to something like `ambiguous_project`. Examples: `title: 'Awesome Plot of {long_name}'`, `xlabel: '{short_name}'`, `xlim: [0, 5]`.

Hint

Extra arguments given to the recipe are ignored, so it is safe to use yaml anchors to share the configuration of common arguments with other monitor diagnostic script.

6.2.4 Monitor Diagnostic

This module provides various tools to monitor climate model simulations. It can be used to plot arbitrary variables from arbitrary datasets.

Examples

- *Monitor*
- *General model evaluation*

Diagnostic scripts

Monitoring diagnostic to plot arbitrary preprocessor output

Diagnostic to plot preprocessor output.

Description

This diagnostic can be used to visualize arbitrary preprocessor output.

Currently supported plot types (use the option `plots` to specify them):

- Climatology (plot type `clim`): Plots climatology. Supported coordinates: (*latitude*, *longitude*, *month_number*).
- Seasonal climatologies (plot type `seasonclim`): It produces a multi panel (2x2) plot with the seasonal climatologies. Supported coordinates: (*latitude*, *longitude*, *month_number*).
- Monthly climatologies (plot type `monclim`): It produces a multi panel (3x4) plot with the monthly climatologies. Can be customized to show only certain months and to rearrange the number of columns and rows. Supported coordinates: (*latitude*, *longitude*, *month_number*).
- Time series (plot type `timeseries`): Generate time series plots. It will always generate the full period time series, but if the period is longer than 75 years, it will also generate two extra time series for the first and last 50 years. It will produce multi panel plots for data with *shape_id* or *region* coordinates of length > 1. Supported coordinates: *time*, *shape_id* (optional) and *region* (optional).
- Annual cycle (plot type `annual_cycle`): Generate an annual cycle plot (timeseries like climatological from January to December). It will produce multi panel plots for data with *shape_id* or *region* coordinates of length > 1. Supported coordinates: *time*, *shape_id* (optional) and *region* (optional).
- Diurnal cycle (plot type `diurnal_cycle`): Generate a diurnal cycle plot (timeseries like climatological from 0 to 24 hours). It will produce multi panel plots for data with *shape_id* or *region* coordinates of length > 1. Supported coordinates: *time*, *shape_id* (optional) and *region* (optional).

Configuration options in recipe

cartopy_data_dir: str, optional (default: None)
Path to cartopy data dir. Defaults to None. See https://scitools.org.uk/cartopy/docs/latest/.

config_file: str, optional
Path to the monitor configuration file. Defaults to `monitor_config.yml` in the same folder as the diagnostic script. More information on the monitor configuration file can be found *here*.

plots: dict, optional
Plot types plotted by this diagnostic (see list above). Dictionary keys must be `clim`, `seasonclim`, `monclim`, `timeseries`, `annual_cycle` or `diurnal_cycle`. Dictionary values are dictionaries used as options for the corresponding plot. The allowed options for the different plot types are given below.

plot_filename: str, optional
Filename pattern for the plots. Defaults to `{plot_type}_{real_name}_{dataset}_{mip}_{exp}_{ensemble}`. All tags (i.e., the entries in curly brackets, e.g., `{dataset}`, are replaced with the corresponding tags).

plot_folder: str, optional
Path to the folder to store figures. Defaults to `{plot_dir}/../../{dataset}/{exp}/{modeling_realm}/{real_name}`. All tags (i.e., the entries in curly brackets, e.g., `{dataset}`, are replaced with the corresponding tags). `{plot_dir}` is replaced with the default ESMValTool plot directory (i.e., `output_dir/plots/diagnostic_name/script_name/`, see Output).

rasterize_maps: bool, optional (default: True)
If `True`, use rasterization for map plots to produce smaller files. This is only relevant for vector graphics (e.g., `output_file_type=pdf,svg,ps`).

In the variable definitions, users can set the attribute `plot_name` to fix the variable name that will be used for the plot's title. If it is not set, `mapgenerator` will try to choose a sensible one from the name attributes (`long_name`, `standard_name` and `var_name`).

Configuration options for plot type `clim`

maps: list of str, optional (default: ['global'])
List of maps to plot, as defined in the monitor configuration file.

Configuration options for plot type `seasonclim`

maps: list of str, optional (default: ['global'])
List of maps to plot, as defined in the monitor configuration file.

Configuration options for plot type `monclim`

maps: list of str, optional (default: ['global'])
List of maps to plot, as defined in the monitor configuration file.

months: list of int, optional
Select only specific months. Defaults to `None` (i.e. show all months).

plot_size: tuple of int, optional (default: (5, 4))
Size of each individual figure.

columns: int, optional (default: 3)
Number of columns in the plot.

rows: int, optional (default: 4)
Number of rows in the plot.

Configuration options for plot type `timeseries`

None

Configuration options for plot type `annual_cycle`

None

Configuration options for plot type `diurnal_cycle`

None

Hint

Extra arguments given to the recipe are ignored, so it is safe to use yaml anchors to share the configuration of common arguments with other monitor diagnostic script.

Monitoring diagnostic to plot EOF maps and associated PC timeseries

Diagnostic to compute and plot the first EOF of an arbitrary input.

Description

This diagnostic can be used to compute and show Empirical Orthogonal Functions (EOFs) and Principal Components (PCs) of arbitrary input. It creates a map plot of the first EOF and the associated PC time series.

Configuration options in recipe

cartopy_data_dir: str, optional (default: None)
Path to cartopy data dir. Defaults to None. See https://scitools.org.uk/cartopy/docs/latest/.

config_file: str, optional
Path to the monitor configuration file. Defaults to `monitor_config.yml` in the same folder as the diagnostic script. More information on the monitor configuration file can be found *here*.

plot_filename: str, optional
Filename pattern for the plots. Defaults to `{plot_type}_{real_name}_{dataset}_{mip}_{exp}_{ensemble}`. All tags (i.e., the entries in curly brackets, e.g., `{dataset}`, are replaced with the corresponding tags).

plot_folder: str, optional
Path to the folder to store figures. Defaults to `{plot_dir}/../../{dataset}/{exp}/{modeling_realm}/{real_name}`. All tags (i.e., the entries in curly brackets, e.g., `{dataset}`, are replaced with the corresponding tags). `{plot_dir}` is replaced with the default ESMValTool plot directory (i.e., `output_dir/plots/diagnostic_name/script_name/`, see Output).

rasterize_maps: bool, optional (default: True)
If `True`, use rasterization for map plots to produce smaller files. This is only relevant for vector graphics (e.g., `output_file_type=pdf,svg,ps`).

Hint

Extra arguments given to the recipe are ignored, so it is safe to use yaml anchors to share the configuration of common arguments with other monitor diagnostic script.

Monitoring diagnostic to show multiple datasets in one plot (incl. biases)

Monitoring diagnostic to visualize multiple datasets.

Description

This diagnostic can be used to visualize multiple datasets in a single plot. A large variety of different plot types is supported.

Currently supported plot types

Plot types can be specified with the *recipe option* `plots`. All plots can be customized with additional *plot-specific options*.

- `1d_profile` (1D plot): plot 1D profile. Input data needs single dimension *altitude* or *air_pressure*.
- `annual_cycle` (1D plot): plot annual cycle. Input data needs single dimension *month_number*.
- `benchmarking_annual_cycle` (1D plot): benchmarking version of `annual_cycle`. Input data needs single dimension *month_number*.
- `benchmarking_boxplot` (boxplot): plot boxplot. Input data needs to be scalar.
- `benchmarking_diurnal_cycle` (1D plot): benchmarking version of `diurnal_cycle`. Input data needs single dimension *hour*.
- `benchmarking_map` (2D plot): benchmarking version of `map`. Input data needs dimensions *(longitude, latitude)*.
- `benchmarking_timeseries` (1D plot): benchmarking version of `timeseries`. Input data needs single dimension *time*.

- `benchmarking_zonal` (2D plot): benchmarking version of `zonal_mean_profile`. Input data needs dimensions *(latitude, altitude)* or *latitude, air_pressure*.
- `diurnal_cycle` (1D plot): plot diurnal cycle. Input data needs single dimension *hour*.
- `hovmoeller_anncyc_vs_lat_or_lon` (2D plot): plot Hovmoeller plot. Input data needs dimensions *(month_number, latitude)* or *(month_number, longitude)*.
- `hovmoeller_time_vs_lat_or_lon` (2D plot): plot Hovmoeller plot. Input data needs dimensions *(time, latitude)* or *(time, longitude)*.
- `hovmoeller_z_vs_time` (2D plot): plot Hovmoeller plot. Input data needs dimensions *(time, altitude)* or *(time, air_pressure)*.
- `map` (2D plot): plot map plot. Input data needs dimensions *(longitude, latitude)*.
- `timeseries` (1D plot): plot time series. Input data needs single dimension *time*.
- `variable_vs_lat` (1D plot): plot variable vs. latitude. Input data needs single dimension *latitude*.
- `zonal_mean_profile` (2D plot): plot zonal mean profile. Input data needs dimensions *(latitude, altitude)* or *latitude, air_pressure*.

Each plot type belongs to one or more classes which define how the data is plotted:

- Class *1D plot*: for each variable separately, all datasets are plotted in one single figure. Input data needs to be 1D.
- Class *2D plot*: for each variable and dataset, an individual figure is plotted. Input data needs to be 2D. A single reference dataset can be defined by setting the facet `reference_for_monitor_diags: True` in the dataset definition in the recipe. In this case, three panels are plotted, incl. a bias. Note that if a reference dataset is defined, all input datasets need to be given on the same horizontal and vertical grid (you can use the preprocessors `esmvalcore.preprocessor.regrid()` and `esmvalcore.preprocessor.extract_levels()` for this).
- Class *boxplot*: one figure for all datasets is plotted. Input data needs to be 0D.

In addition, for many plot types, a benchmarking version exists (`benchmarking_*`). These plots include envelopes (1D plots) or hatching (2D plots) to visualize data ranges (e.g., multi-model percentiles) in addition to the data itself. For these plots, at least one benchmarking dataset needs to be defined by setting the facet `benchmark_dataset: True` in the dataset definition in the recipe. Reference datasets are not supported for 2D benchmarking diagnostics.

Author

Manuel Schlund (DLR, Germany)

Configuration options in recipe

facet_used_for_labels: str, optional (default: 'dataset')
Facet used to label different datasets in plot titles and legends. For example, `facet_used_for_labels: 'dataset'` will use dataset names in plot titles and legends; `facet_used_for_labels: 'exp'` will use experiments in plot titles and legends. In addition, `facet_used_for_labels` is used to select the correct `plot_kwargs` for the different datasets (see configuration options for the different plot types below).

figure_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.figure()`. By default, uses `{constrained_layout: True}`.

group_variables_by: str, optional (default: 'short_name')
Facet or coordinate which is used to create variable groups. For each variable group, an individual plot is created. Specifying a coordinate allows to create one plot for each point along a dimension. For example, when used

in combination with the preprocessor function esmvalcore.preprocessor.extract_shape() the *shape_id* coordinate can be used to create one plot for each shape.

matplotlib_rc_params: dict, optional
Optional matplotlib.RcParams used to customize matplotlib plots. Options given here will be passed to matplotlib.rc_context() and used for all plots produced with this diagnostic. Note: fontsizes specified here might be overwritten by the plot-type-specific option `fontsize` (see below).

plots: dict
Plot types plotted by this diagnostic (see list above). Dictionary keys must be elements of the list above. Dictionary values are dictionaries used as options for the corresponding plot. The allowed options for the different plot types are given below.

plot_filename: str, optional
Filename pattern for the plots. By default, uses `'{plot_type}_{real_name}_{dataset}_{mip}_{exp}_{ensemble}'`. All tags (i.e., the entries in curly brackets, e.g., `'{dataset}'`, are replaced with the corresponding tags).

plot_folder: str, optional
Path to the folder to store figures. By default, uses `'{plot_dir}/../../{dataset}/{exp}/{modeling_realm}/{real_name}'`. All tags (i.e., the entries in curly brackets, e.g., `'{dataset}'`, are replaced with the corresponding tags). `'{plot_dir}'` is replaced with the default ESMValTool plot directory (i.e., `output_dir/plots/diagnostic_name/script_name/`, see Output).

savefig_kwargs: dict, optional
Optional keyword arguments for matplotlib.pyplot.savefig(). By default, uses `{bbox_inches: 'tight', dpi: 300, orientation: 'landscape'}`.

seaborn_settings: dict, optional
Options for seaborn.set_theme() (affects all plots). By default, uses `{style: 'ticks'}`.

Configuration options for 1D plots

aspect_ratio: float, optional (default: None)
Aspect ratio of the plot.

axes_kwargs: dict, optional
Optional calls to methods of the corresponding matplotlib.axes.Axes instance. Dictionary keys are functions of matplotlib.axes.Axes. Dictionary values are used as argument(s) for these functions (if values are dictionaries, these are interpreted as keyword arguments; otherwise a single argument is assumed). String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., `{project}`, `{short_name}`, `{exp}`. Examples: `{set_title: 'Awesome Plot of {long_name}'}`, `{set_xlabel: '{short_name}'}`, `{set_xlim: [0, 5]}`.

caption: str, optional
Figure caption used for provenance tracking. Can include facets in curly brackets which will be derived from the corresponding dataset, e.g., `{project}`, `{short_name}`, `{exp}`. By default, uses a very basic caption.

envelope_kwargs: dict, optional
Optional keyword arguments for iris.plot.fill_between(). By default, uses `{alpha: 0.8, facecolor: 'lightblue', linewidth: 0.0, zorder: 1.0}`. Only relevant for benchmarking plots.

gridline_kwargs: dict, optional
Optional keyword arguments for grid lines. By default, uses `{color: 'lightgrey', alpha: 0.5}`. Use `gridline_kwargs: False` to not show grid lines.

hlines: list of dict, optional
Horizontal lines to show in plot. Each list element corresponds to one line, and each list element should contain a dictionary with keywords arguments passed to matplotlib.axes.Axes.axhline(). Example: `[{y: 0}, {y: 1, color: 'red'}]`.

legend_kwargs: dict, optional
Optional keyword arguments for matplotlib.pyplot.legend(). Use legend_kwargs: False to not show legends.

log_x: bool, optional (default: False)
Use logarithmic X-axis.

log_y: bool, optional (default: False)
Use logarithmic Y-axis.

plot_kwargs: dict, optional
Optional keyword arguments for iris.plot.plot(). Dictionary keys are elements identified by facet_used_for_labels or 'default', e.g., 'CMIP6' if facet_used_for_labels: 'project' or 'historical' if facet_used_for_labels: 'exp'. Dictionary values are dictionaries used as keyword arguments for iris.plot.plot(). String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project}, {short_name}, {exp}. Examples: {default: {linestyle: '-', label: '{project}'}, CMIP6: {color: 'red', linestyle: '--'}, OBS: {color: 'black'}}.

pyplot_kwargs: dict, optional
Optional calls to functions of matplotlib.pyplot. Dictionary keys are functions of matplotlib.pyplot. Dictionary values are used as argument(s) for these functions (if values are dictionaries, these are interpreted as keyword arguments; otherwise a single argument is assumed). String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project}, {short_name}, {exp}. Examples: {title: 'Awesome Plot of {long_name}'}, {xlabel: '{short_name}'}, {xlim: [0, 5]}.

rasterize: bool, optional (default: False)
If True, use rasterization for plots to produce smaller files. This is only relevant for vector graphics (e.g., output_file_type: 'pdf').

time_format: str, optional (default: None)
strftime() format string that is used to format the time axis using matplotlib.dates.DateFormatter. If None, use the default formatting imposed by the iris plotting function.

transpose_axes: bool, optional (default: False)
Swap X- and Y-axis.

x_major_formatter: str, optional (default: None)
Format string for matplotlib.ticker.FormatStrFormatter used to format major tick labels of X-axis.

x_minor_formatter: str, optional (default: None)
Format string for matplotlib.ticker.FormatStrFormatter used to format minor tick labels of X-axis.

y_major_formatter: str, optional (default: None)
Format string for matplotlib.ticker.FormatStrFormatter used to format major tick labels of Y-axis.

y_minor_formatter: str, optional (default: None)
Format string for matplotlib.ticker.FormatStrFormatter used to format minor tick labels of Y-axis.

Configuration options for 2D plots

aspect_ratio: float, optional (default: None)
Aspect ratio of the plot.

axes_kwargs: dict, optional
Optional calls to methods of the corresponding matplotlib.axes.Axes instance. Dictionary keys are functions of matplotlib.axes.Axes. Dictionary values are used as argument(s) for these functions (if values are dictionaries, these are interpreted as keyword arguments; otherwise a single argument is assumed). String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project},

{short_name}, {exp}. Examples: {set_title: 'Awesome Plot of {long_name}'}, {set_xlabel: '{short_name}'}, {set_xlim: [0, 5]}.

caption: str, optional
Figure caption used for provenance tracking. Can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project}, {short_name}, {exp}. By default, uses a very basic caption.

cbar_label: str, optional (default: '{short_name} [{units}]')
Colorbar label. Can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project}, {short_name}, {exp}.

cbar_label_bias: str, optional (default: '{short_name} [{units}]')
Colorbar label for plotting biases. Can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project}, {short_name}, {exp}. Only relevant for plots including reference datasets.

cbar_kwargs: dict, optional
Optional keyword arguments for matplotlib.pyplot.colorbar(). By default, uses {orientation: 'vertical'}.

cbar_kwargs_bias: dict, optional
Optional keyword arguments for matplotlib.pyplot.colorbar() for plotting biases. These keyword arguments update (and potentially overwrite) the cbar_kwargs for the bias plot. Only relevant for plots including reference datasets.

common_cbar: bool, optional (default: False)
Use a common colorbar for the top panels (i.e., plots of the dataset and the corresponding reference dataset) when using a reference dataset. If neither vmin and vmax nor levels is given in plot_kwargs, the colorbar bounds are inferred from the dataset in the top left panel, which might lead to an inappropriate colorbar for the reference dataset (top right panel). Thus, specify vmin and vmax or levels when using common_cbar: True. Only relevant for plots including reference datasets.

fontsize: int, optional (default: None)
Fontsize used for ticks, labels and titles. For the latter, use the given fontsize plus 2. Does not affect suptitles. If not given, use default matplotlib values. For a more fine-grained definition of fontsizes, use the option matplotlib_rc_params (see above).

gridline_kwargs: dict, optional (default: False)
Optional keyword arguments for grid lines. Use gridline_kwargs: False to not show grid lines.

log_x: bool, optional (default: False)
Use logarithmic X-axis.

log_y: bool, optional (default: False)
Use logarithmic Y-axis.

plot_func: str, optional (default: 'contourf')
Plot function used to plot the maps. Must be a function of iris.plot that supports plotting of 2D data.

plot_kwargs: dict, optional
Optional keyword arguments for the plot function defined by plot_func. Dictionary keys are elements identified by facet_used_for_labels or 'default', e.g., 'CMIP6' if facet_used_for_labels: 'project' or 'historical' if facet_used_for_labels: 'exp'. Dictionary values are dictionaries used as keyword arguments for the plot function defined by plot_func. String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project}, {short_name}, {exp}. Examples: {default: {levels: 2}, CMIP6: {vmin: 200, vmax: 250}}. In addition to the normalization options supported by the plot function, the option {norm: 'centered'} can be specified. In this case, the keywords vcenter and halfrange should be used instead of vmin or vmax (see CenteredNorm).

plot_kwargs_bias: dict, optional
Optional keyword arguments for the plot function defined by plot_func for plotting biases. These keyword argu-

ments update (and potentially overwrite) the `plot_kwargs` for the bias plot. By default, uses `{cmap: 'bwr', norm: 'centered'}`. Only relevant for plots including reference datasets.

projection: str, optional (default: None)
Projection used for the plot. Needs to be a valid projection class of `cartopy.crs`. Keyword arguments can be specified using the option `projection_kwargs`. For map plots, `'Robinson'` is used as default.

projection_kwargs: dict, optional
Optional keyword arguments for the projection given by `projection`. For map plots, the default keyword arguments `{central_longitude: 10}` are used.

pyplot_kwargs: dict, optional
Optional calls to functions of `matplotlib.pyplot`. Dictionary keys are functions of `matplotlib.pyplot`. Dictionary values are used as argument(s) for these functions (if values are dictionaries, these are interpreted as keyword arguments; otherwise a single argument is assumed). String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., `{project}`, `{short_name}`, `{exp}`. Examples: `{title: 'Awesome Plot of {long_name}'}`, `{xlabel: '{short_name}'}`, `{xlim: [0, 5]}`.

rasterize: bool, optional (default: False)
If `True`, use rasterization for plots to produce smaller files. This is only relevant for vector graphics (e.g., `output_file_type: 'pdf'`).

show_stats: bool, optional (default: True)
Show basic statistics on the plots.

time_format: str, optional (default: None)
`strftime()` format string that is used to format the time axis using `matplotlib.dates.DateFormatter`. If `None`, use the default formatting imposed by the iris plotting function.

transpose_axes: bool, optional (default: False)
Swap X- and Y-axis.

x_major_formatter: str, optional (default: None)
Format string for `matplotlib.ticker.FormatStrFormatter` used to format major tick labels of X-axis.

x_minor_formatter: str, optional (default: None)
Format string for `matplotlib.ticker.FormatStrFormatter` used to format minor tick labels of X-axis.

x_pos_stats_avg: float, optional (default: 0.01)
Text X-position of average (shown on the left) in Axes coordinates. Can be adjusted to avoid overlap with the figure. Only relevant if `show_stats: True`.

x_pos_stats_bias: float, optional (default: 0.7)
Text X-position of bias statistics (shown on the right) in Axes coordinates. Can be adjusted to avoid overlap with the figure. Only relevant if `show_stats: True`.

y_major_formatter: str, optional (default: None)
Format string for `matplotlib.ticker.FormatStrFormatter` used to format major tick labels of Y-axis.

y_minor_formatter: str, optional (default: None)
Format string for `matplotlib.ticker.FormatStrFormatter` used to format minor tick labels of Y-axis.

Configuration options for boxplots

axes_kwargs: dict, optional
Optional calls to methods of the corresponding `matplotlib.axes.Axes` instance. Dictionary keys are functions of `matplotlib.axes.Axes`. Dictionary values are used as argument(s) for these functions (if values are dictionaries, these are interpreted as keyword arguments; otherwise a single argument is assumed). String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., `{project}`,

{short_name}, {exp}. Examples: {set_title: 'Awesome Plot of {long_name}'}, {set_xlabel: '{short_name}'}, {set_xlim: [0, 5]}.

caption: str, optional
Figure caption used for provenance tracking. Can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project}, {short_name}, {exp}. By default, uses a very basic caption.

fontsize: int, optional (default: None)
Fontsize used for ticks, labels and titles. For the latter, use the given fontsize plus 2. Does not affect suptitles. If not given, use default matplotlib values. For a more fine-grained definition of fontsizes, use the option matplotlib_rc_params (see above).

plot_kwargs: dict, optional
Optional keyword arguments for seaborn.boxplot().

pyplot_kwargs: dict, optional
Optional calls to functions of matplotlib.pyplot. Dictionary keys are functions of matplotlib.pyplot. Dictionary values are used as argument(s) for these functions (if values are dictionaries, these are interpreted as keyword arguments; otherwise a single argument is assumed). String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., {project}, {short_name}, {exp}. Examples: {title: 'Awesome Plot of {long_name}'}, {xlabel: '{short_name}'}, {xlim: [0, 5]}.

scatter_kwargs: dict, optional
Optional keyword arguments for matplotlib.pyplot.scatter() (used to plot marks for benchmarking dataset). By default, uses {marker: 'x', s: 200, linewidths: 2, color: 'red', zorder: 3}.

var_order: list of str, optional (default: None)
Order in which variables are plotted. Must be elements identified via group_variables_by. If None, use order in which data is given.

Base class for monitoring diagnostics

Base class for monitoring diagnostics

Base class for monitoring diagnostics.

Classes:

MonitorBase(config)	Base class for monitoring diagnostic.

class esmvaltool.diag_scripts.monitor.monitor_base.**MonitorBase**(*config*)

Bases: object

Base class for monitoring diagnostic.

It contains the common methods for path creation, provenance recording, option parsing and to create some common plots.

Methods:

get_plot_folder(var_info)	Get plot storage folder from variable info.
get_plot_name(plot_type, var_info[, add_ext])	Get plot filename from variable info.
get_plot_path(plot_type, var_info[, add_ext])	Get plot full path from variable info.
get_provenance_record(ancestor_files, **kwargs)	Create provenance record for the diagnostic data and plots.

continues on next page

Table 12 – continued from previous page

plot_cube(cube, filename[, linestyle])	Plot a timeseries from a cube.
plot_timeseries(cube, var_info[, period])	Plot timeseries from a cube.
record_plot_provenance(filename, var_info, ...)	Write provenance info for a given file.

get_plot_folder(*var_info*)

Get plot storage folder from variable info.

Parameters
var_info (*dict*) – Variable information from ESMValTool

get_plot_name(*plot_type*, *var_info*, *add_ext=True*)

Get plot filename from variable info.

Parameters

- **plot_type** (*str*) – Name of the plot
- **var_info** (*dict*) – Variable information from ESMValTool
- **add_ext** (*bool, optional (default: True)*) – Add filename extension from configuration file.

get_plot_path(*plot_type*, *var_info*, *add_ext=True*)

Get plot full path from variable info.

Parameters

- **plot_type** (*str*) – Name of the plot
- **var_info** (*dict*) – Variable information from ESMValTool
- **add_ext** (*bool, optional (default: True)*) – Add filename extension from configuration file.

static get_provenance_record(*ancestor_files*, ***kwargs*)

Create provenance record for the diagnostic data and plots.

plot_cube(*cube*, *filename*, *linestyle='-'*, ***kwargs*)

Plot a timeseries from a cube.

Supports multiplot layouts for cubes with extra dimensions *shape_id* or *region*.

plot_timeseries(*cube*, *var_info*, *period=''*, ***kwargs*)

Plot timeseries from a cube.

It also automatically smoothes it for long timeseries of monthly data:

- Between 10 and 70 years long, it also plots the 12-month rolling average along the raw series
- For more than ten years, it plots the 12-month and 10-years rolling averages and not the raw series

record_plot_provenance(*filename*, *var_info*, *plot_type*, ***kwargs*)

Write provenance info for a given file.

6.2.5 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

Diagnostic for fig 3.24 in Chapter 3 of IPCC AR6 WGI.

This diagnostic either calculates bias with respect to the reference dataset or plots the data and the reference dataset. The diagnostic accepts the groups of the 1D datasets (with exception of the resolved mask). The additional keywords for the diagnostic provided through the recipe: bias: an option boolean flag showing if bias is calculated, default: false mask: an optional dictionary if the data should be masked, default: false mask is a dictionary and should contain keywords: flag: flag showing if the data should be masked type: type of mask, accepted values: 'simple' and 'resolved' group: name of the variable group with resolved mask (needed only if type='resolved') statistics: a dictionary with the statistics which will be plotted. Needs keywords 'best_guess': a string with the statistical operator, and 'borders' a list with the statistics to be the borders for teh shading. mpl_style: name of the matplotlib style file (optional) caption: figure caption (optional) color_style: optional name of the color style, colors are defined for variable groups.

The resolved mask can have as many dimensions as one needs.

Initial development by: Lee de Mora (PML)
ledm@pml.ac.uk

Revised and corrected (15.01.2021): Elizaveta Malinina (CCCma)
elizaveta.malinina-rieger@ec.gc.ca

Rewritten and adapted to main branch v2.12+ (08.03.2025):
Elizaveta Malinina (CCCma) elizaveta.malinina-rieger@ec.gc.ca

Classes:

Data4Analyis(name, group, cfg, mask_meta)	Data class which later will be used for plotting.

Functions:

create_provenance(caption)	Create provenance dictionary.
main(cfg)	Diagnostic itself.
plot_bias_plot(data_list, cfg)	Plot the diagnostic figure.

class esmvaltool.diag_scripts.ocean.diagnostic_biases.**Data4Analyis**(*name: str*, *group: list*, *cfg: dict*, *mask_meta: list[dict] | None*)

Bases: object

Data class which later will be used for plotting.

name

name of the data group for which class is created

Type
str

bias

flag, showing if bias should be calculated

Type
bool

mask

flag if data should be masked

Type
bool

mask_type

type of the mask: 'simple' or 'resolved'

Type
str

reference

name of the reference dataset

Type
str

ref_cube

iris cube with the reference dataset data

Type
iris.cube.Cube

data

iris cubelist with the data

Type
iris.cube.CubeList

best_guess

iris cube with the best guess to be plotted

Type
iris.cube.Cube

border1

iris cube with the bottom/top for the shading

Type
iris.cube.Cube

border2

iris cube with the top/bottom for the shading

Type
iris.cube.Cube

Methods:

calculate_statistics(stats, cfg)	Calculate statistics which later will be plotted.
determine_reference(group)	Determine the reference dataset from the data group.
mask_data(mask_meta)	Masks data.
obtain_data(group, mask_meta)	Obtain data for further statistics calculation.

calculate_statistics(*stats: dict*, *cfg: dict*)

Calculate statistics which later will be plotted.

Parameters

- **stats** – dictionary with the statistics which will be calculated. Dictionary should have keywords 'best_guess' and 'borders'.

- **cfg** – Config dictionary coming from ESMValCore

determine_reference(*group: list*)

Determine the reference dataset from the data group.

Parameters
group – List with the metadata of the variable group

Raises
ValueError – if more than one or no reference datasets have been provided

mask_data(*mask_meta: list[dict] | None*)

Masks data.

Parameters
mask_meta – List with the mask_metadata in case mask_type='resolved'

Raises
ValueError – if more than one datasets for resolved mask are provided or if data cubes have more than one dimension or if an unsupported type of mask is provided

obtain_data(*group: list*, *mask_meta: list[dict] | None*)

Obtain data for further statistics calculation.

Parameters

- **group** – List with the metadata of the variable group
- **mask_meta** – List with the mask_metadata in case mask_type='resolved'

esmvaltool.diag_scripts.ocean.diagnostic_biases.**create_provenance**(*caption: str*)

Create provenance dictionary.

esmvaltool.diag_scripts.ocean.diagnostic_biases.**main**(*cfg: dict*)

Diagnostic itself.

esmvaltool.diag_scripts.ocean.diagnostic_biases.**plot_bias_plot**(*data_list: list[*Data4Analyis*]*, *cfg: dict*)

Plot the diagnostic figure.

Parameters

- **data_list** – List with the data classes which will be plotted
- **cfg** – Config dictionary coming from ESMValCore

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., ]
```

(continues on next page)

(continued from previous page)

```
      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.

Model 1 vs Model 2 vs Observations diagnostics.

Diagnostic to produce an image showing four maps, based on a comparison of two different 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.

This diagnostic also requires the `exper_model`, `control_model` and `observational_dataset` keys in the recipe.

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.

Original script: Author: Lee de Mora (PML): ledm@pml.ac.uk

Refactored script: (Alphabetical order) Author: Sophie Hall (Met Office): sophie.hall@metoffice.gov.uk Author: Emma Hogan (Met Office): emma.hogan@metoffice.gov.uk Author: Dave Storkey (Met Office): dave.storkey@metoffice.gov.uk

Functions:

create_plotting_data(control, experiment, ...)	Calculate the diff between the ctr, exp & obs to prepare data for plots.
create_quadmap(exp_single_level, ...)	Add all subplots to a main plot, positions of subplots are pre-set.
load_data(config)	Load all necessary data to output experiment, control, observation.
main(config)	Load the config file and running through the order of operation.
plot_global_single_level(axis, cube, ...)	Create each individual plot before being added to create_quadmap.
save_cube(cube, field_name, config, ancestors)	Produces a provenance record and saves data for each cube.

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**create_plotting_data**(*control*, *experiment*, *observation*)

Calculate the diff between the ctr, exp & obs to prepare data for plots.

Parameters

- **control** (*iris cube*) – Data as defined as control_model from the recipe.
- **experiment** (*iris cube*) – Data as defined as exper_model from the recipe.

- **observation** (*iris cube*) – Data as defined as observational_dataset from the recipe.

Returns

- **exp** (*iris cube*) – Untouched experimental input.
- **exp_minus_ctr** (*iris cube*) – Experiment model minus control model.
- **ctr_minus_obs** (*iris cube*) – Control model minus observational dataset.
- **exp_minus_obs** (*iris cube*) – Experimental model minus observational dataset.

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**create_quadmap**(*exp_single_level*, *exp_minus_ctr_single_level*, *ctr_minus_obs_single_level*, *exp_minus_obs_single_level*, *config*)

Add all subplots to a main plot, positions of subplots are pre-set.

Parameters

- **exp_single_level** (*iris cube*) – Extracted single level of experiment cube.
- **exp_minus_ctr_single_level** (*iris cube*) – Extracted single level of exp_minus_ctr cube.
- **ctr_minus_obs_single_level** (*iris cube*) – Extracted single level of ctr_minus_obs cube.
- **exp_minus_obs_single_level** (*iris cube*) – Extracted single level of exp_minus_obs cube.
- **config** (*dictionary*) – configuration dictionary that contains all the necessary information for the function to run. It includes details about the models, observational datasets, file paths, and other settings.

Returns

Make the four pane model vs model vs obs comparison plot

Return type

quadmap

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**load_data**(*config*)

Load all necessary data to output experiment, control, observation.

Parameters

config (*dictionary*) – configuration dictionary that contains all the necessary information for the function to run. It includes details about the models, observational datasets, file paths, and other settings.

Returns

- **experiment** (*iris cube*) – Data as defined as exper_model from the recipe.
- **control** (*iris cube*) – Data as defined as control_model from the recipe.
- **observation** (*iris cube*) – Data as defined as observational_dataset from the recipe.

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**main**(*config*)

Load the config file and running through the order of operation.

Parameters

config (*dictionary*) – configuration dictionary that contains all the necessary information

for the function to run. It includes details about the models, observational datasets, file paths, and other settings.

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**plot_global_single_level**(*axis*, *cube*, *contour_levels*, *title*)

Create each individual plot before being added to create_quadmap.

Parameters

- **axis** (*matplotlib 'ax'*) – Represents one (sub-)plot in a figure. It contains the plotted data, axis ticks, labels, title, legend, etc.
- **cube** (*iris cube*) – This is a data structure that contains the climate data to be plotted. Including information like temperature values, latitude, longitude, and depth.
- **contour_levels** (*numpy.array*) – Used to set the ticks on the colour bar and used to define levels for the contour plot.
- **title** (*str*) – This is a string that will be used as the title of the subplot.

esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.**save_cube**(*cube*, *field_name*, *config*, *ancestors*)

Produces a provenance record and saves data for each cube.

Parameters

- **cube** (*iris cube*) – This is a data structure that contains the climate data to be plotted. Including information like temperature values, latitude, longitude, and depth.
- **field_name** (*str*) – A string that contains the cube name with the corresponding extracted depth level.
- **config** (*dictionary*) – configuration dictionary that contains all the necessary information for the function to run. It includes details about the models, observational datasets, file paths, and other settings.
- **ancestors** (*list*) – A list of keys from the input_files dictionary, representing the provenance of the data. This list helps track the origin and transformation history of the data used in the cube

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 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
```

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.
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 sig significant digits & return it as a string.

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.

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 sig significant digits & 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

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*, *metadata*, *filename*, *obs_metadata=None*, *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.

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

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, filename)	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*, *metadata*)

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*, *region=''*)

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.

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.

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.
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 dictionary layer:cube
match_model_to_key(model_type, cfg_dict, ...)	Match up model or observations dataset dictionairies from config file.
prepare_provenance_record(cfg, ...)	Prepare informations to feed provenance

esmvaltool.diag_scripts.ocean.diagnostic_tools.**add_legend_outside_right**(*plot_details*, *ax1*, *column_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.

Parameters
cube (*iris.cube.Cube*) – the opened dataset as a 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

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*, *default='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 dictionary, 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 dictionary (for the individual dataset file)

Parameters

- **cfg** (*dict*) – the opened global config dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary 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 dictionary, passed by ESMValTool.
- **index** (*int*) – the index of the file in the cfg file.

Returns
A dictionary 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – the metadata dictionary

Returns
List of thresholds

Return type
list

esmvaltool.diag_scripts.ocean.diagnostic_tools.**make_cube_layer_dict**(*cube*)

Take a cube and return a dictionary 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 dictionary 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 dictionary in the cfg metadata.

Parameters

- **model_type** (*str*) – The string model_type to match (only used in debugging).
- **cfg_dict** (*dict*) – the config dictionary item for this model type, parsed directly from the diagnostics/ scripts, part of the recipe.
- **input_files_dict** (*dict*) –

  The input file dictionary, loaded directly from the get_input_files()
  function, in diagnostics_tools.py.

Returns
A dictionary of the input files and their linked details.

Return type
dict

esmvaltool.diag_scripts.ocean.diagnostic_tools.**prepare_provenance_record**(*cfg*, ***provenance_record*)

Prepare informations to feed provenance

Parameters

- **cfg** (*dict*) – the opened global config dictionary, passed by ESMValTool.
- **provenance_record** (*dict*) – dictionary for a specific diagnostic provenance details.

6.2.6 Seaborn Diagnostic

Create arbitrary Seaborn plots.

Description

This diagnostic provides a high-level interface to Seaborn. For this, the input data is arranged into a single `pandas.DataFrame`, which is then used as input for the Seaborn function defined by the option *seaborn_func*.

Caveats

All datasets of a given variable must have the same units (e.g., it is not allowed to use datasets with units *K* and datasets with units *°C* for the variable *tas*).

Author

Manuel Schlund (DLR, Germany)

Configuration options in recipe

add_ancillary_variables: bool, optional (default: False)
Add `ancillary_variables()` to the main data frame. Note that this will assume that ancillary variables are identical across cubes within a group (see option *groupby_facet*). This equality is not checked!

add_aux_coords: bool, optional (default: False)
Add `aux_coords` to the main data frame. Note that this will assume that auxiliary coordinates are identical across cubes within a group (see option *groupby_facet*). This equality is not checked!

add_cell_measures: bool, optional (default: False)
Add `cell_measures()` to the main data frame. Note that this will assume that cell measures are identical across cubes within a group (see option *groupby_facet*). This equality is not checked!

data_frame_ops: dict, optional
Perform additional operations on the main data frame. Allowed operations are `pandas.DataFrame.query()` (dict key *query*) and `pandas.DataFrame.eval()` (dict key *eval*). Operations are defined by strings (dict values). Examples: `{'query': 'latitude > 80', 'eval': 'longitude = longitude - 180.0'}`.

dropna_kwargs: dict, optional
Optional keyword arguments for `pandas.DataFrame.dropna()` to drop missing values in the input data. If not given, do not drop NaNs. Note: NaNs are dropped after potential *data_frame_ops*.

facets_as_columns: list of str, optional
Facets that will be added as a columns to the main data frame. Values for these facets must be identical across all datasets within a group (see option *groupby_facet*).

groupby_facet: str, optional (default: 'alias')
Facet which is used to group input datasets when creating the main data frame. All datasets within a group are expected to have the same index after calling `iris.pandas.as_data_frame()` on them. These datasets within a group will then get merged (combined along axis 1, i.e., columns) into a single data frame per group. Finally,

the data frames for all groups are concatenated (combined along axis 0, i.e., rows) into one main data frame. *groupby_facet* is also added as a column to this main data frame.

legend_title: str, optional (default: None)
Title for legend. If `None`, Seaborn will determine the legend title (if possible).

plot_filename: str, optional
Filename for the final plot. By default, uses 'seaborn_(*seaborn_func*)'.

plot_object_methods: dict, optional
Execute methods of the object returned by the plotting function (*seaborn_func*). This object will either be a `matplotlib.axes.Axes` (e.g., `scatterplot()`, `lineplot()`), a `seaborn.FacetGrid` (e.g., `relplot()`, `displot()`), a `seaborn.JointGrid` (e.g., `jointplot()`), or a `seaborn.PairGrid` (e.g., `pairplot()`). Dictionary keys are method names, dictionary values function arguments (use a `dict` to specify keyword arguments). Example (for `relplot()`): `{'set': {'xlabel': 'X [km]'}, 'set_titles': 'Model {col_name}'}`.

reset_index: bool, optional (default: False)
Put coordinate information of datasets into columns instead of (multi-) indices. This avoids the deletion of coordinate information if different groups of datasets have different dimensions but increases the memory footprint of this diagnostic.

write_netcdf: bool, optional (default: False)
Output netCDF file for plotted data. What is written into the file is decided based on the pandas data frame and the seaborn kwargs: "x", "y", "hue" and "col". Because there is no direct way to write panda data frames into netCDF and to make data CF complient, the data frame is converted to an iris CubeList first. This is not possible for all data frames and often reset_index: true is required. Therefore the default is set to False.

savefig_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.savefig()`. By default, uses `bbox_inches: tight, dpi: 300, orientation: landscape`.

seaborn_func: str
Function used to plot the data. Must be a function of Seaborn. An overview of Seaborn's plotting functions is given here.

seaborn_kwargs: dict, optional
Optional keyword arguments for the plotting function given by *seaborn_func*. Must not include an argument called *data*. Example: `{'x': 'variable_1', 'y': 'variable_2', 'hue': 'coord_1'}`. Note: variables (here: *variable_1* and *variable_2* are identified by their *variable_group* in the recipe, i.e., the keys that specify variable groups in *variables*.

seaborn_settings: dict, optional
Options for `seaborn.set_theme()` (affects all plots).

suptitle: str or None, optional (default: None)
Suptitle for the plot (see `matplotlib.pyplot.suptitle()`). If `None`, do not create a suptitle. If the plot shows only a single panel, use *plot_object_methods* with `{'set': {'title': 'TITLE'}}` instead.

6.2.7 Plotting 1D Histograms

Diagnostic script that creates a 1D histogram plot for several datasets.

Description

This script plots 1D histograms for each dataset provided in the input metadata. It uses precomputed histogram data from the preprocessor "histogram".

Authors

- Lisa Bock (DLR)

Configuration parameters through recipe

- log_y: Boolean to set y-axis to logarithmic scale (default: False).
- ylim: List of two floats to set y-axis limits (default: None).
- suptitle: String for the figure's title (default: None).
- dpi: Integer for the resolution of the saved figure (default: 300).
- xlabel: String for the x-axis label (default: None, which uses the variable name and units from the cube).
- ylabel: String for the y-axis label (default: None).
- plot_filename: String for the base name of the saved plot and data files (default: "histogram_diag").

6.2.8 Portrait Plot

Portrait Plot Diagnostic.

Plot performance metrics of multiple datasets vs up to four references A *documented example recipe* to use this diagnostic is provided as `recipes/recipe_portrait_CMIP.yml`.

Description

This diagnostic provides plot functionalities for performance metrics, and is written to be as flexible as possible to be adaptable to further use cases. X and Y axis, grouping parameter and splits for each rectangle can be configured in the recipe. All `*_by` parameters can be set to any metadata key. To split by 'reference' this key needs to be set as extra_facet in recipe.

Authors

- Lukas Lindenlaub (Universität Bremen, Germany)
- Diego Cammarano

Configuration parameters through recipe:

axes_properties: dict, optional
: Dictionary that gets passed to `matplotlib.axes.Axes.set()`. Subplots can be widely customized. E.g. xlabel, ylabel, yticklabels, xmargin... By default {}.

cbar_kwargs: dict, optional
: Dictionary that gets passed to `matplotlib.pyplot.colorbar()`. E.g. label, ticks... By default {}.

default_split: str, optional
: Data labeled with this string, will be used as main rectangles. All other splits will be plotted as overlays. This can be used to choose the base reference, while all references are labeled for the legend. If None, the first split will be used as default. By default None.

dpi: int, optional
: Dots per inch for the figure. By default 300.

domain: str, optional
: Domain for provenance. By default 'global'.

figsize: tuple of float, optional
[width, height] of the figure in inches. The final figure will be saved with bbox_inches="tight", which can change the resulting aspect ratio. By default [7.5, 3.5].

group_by: str, optional
Split portrait groups into multiple groups (one matrix per group). By default 'project'.

legend: dict, optional
Customize, if, how and where the legend is plotted. The 'best' position and size of the legend depends on multiple parameters of the figure (i.e. lengths of labels, aspect ratio of the plots...). Might require manual adjustment of `x`, `y` and `size` to fit the figure layout. Keys (each optional) that will be handled are:

position: str or None, optional
Position of the legend. Can be 'right' or 'left'. Or set to None to disable plotting the legend. By default 'right'.

size: float, optional
Size of the legend in Inches. By default 0.3.

x_offset: float, optional
Manually adjust horizontal position to save space or fix overlap. Number given in Inches. By default 0.

y_offset: float, optional
Manually adjust vertical position to save space or fix overlap. Number given in Inches. By default 0.

matplotlib_rc_params: dict, optional
Optional `matplotlib.RcParams` used to customize the portrait plot. Options given here will be passed to `matplotlib.rc_context()`.

nan_color: str or None, optional
Matplotlib named color or hexcode for NaN values. If set to None, no triangles are plotted for NaN values. By default 'white'.

normalize: str or None, optional
('mean', 'median', 'centered_mean', 'centered_median', None). Divide by median or mean if not None. Subtract median/mean afterwards if centered. By default 'centered_median'.

plot_kwargs: dict, optional
Dictionary that gets passed as kwargs to `matplotlib.axes.Axes.imshow()`. Colormaps will be converted to 11 discrete steps automatically. Default colormap is `cmap='RdYlBu_r'` with limits `vmin=-0.5` and `vmax=0.5`.

plot_legend: bool, optional
If True, a legend is plotted, when multiple splits are given. By default True.

split_by: str, optional
The rectangles can be split into 2-4 triangles. This is used to show metrics for different references. For this case there is no need to change this parameter. Multiple variables can be set in the recipe with `split` assigned as extra_facet to label the different references. Data without a split assigned will be plotted as main rectangles, this can be changed by setting default_split parameter. By default 'split'.

x_by: str, optional
Metadata key for x coordinate. By default 'alias'.

y_by: str, optional
Metadata key for y coordinate. By default 'variable_group'.

6.2.9 Transient Climate Response to Cumulative CO_2 Emissions (TCRE)

Calculate Transient Climate Response to Cumulative CO2 Emissions (TCRE).

Description

This diagnostics calculates the Transient Climate Response to Cumulative CO2 Emissions (TCRE) and produces relevant plots.

Author

Manuel Schlund (DLR, Germany)

Configuration options in recipe

calc_tcre_period: list of int, optional (default: [90, 110])
Period considered to calculate TCRE. This needs to be a sequence of two integers, where the first integer defines the start and the second integer the end of a slice. TCRE is calculated by averaging over a subarray of the temperature change array which is determined by this slice. For example, if the input data are annual means, the values here correspond to the years (measured from simulation start) over which the temperature change is averaged (by default from years 90 to 110).

caption: str, optional
Figure caption used for provenance tracking. By default, uses "Surface air temperature anomaly versus cumulative CO2 emissions for multiple datasets."

exp_control: str, optional (default: 'esm-piControl')
Name of the control experiment.

exp_target: str, optional (default: 'esm-flat10')
Name of the target experiment.

figure_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.figure()`. By default, uses `constrained_layout: true`.

gridline_kwargs: dict, optional
Optional keyword arguments for grid lines. By default, `color: lightgrey, alpha: 0.5` are used. Use `gridline_kwargs: false` to not show grid lines.

groupby_facet: str, optional (default: 'dataset')
Facet used to group datasets. TCRE is calculated for each group element individually.

legend_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.legend()`. Use `legend_kwargs: false` to not show legends.

matplotlib_rc_params: dict, optional
Optional `matplotlib.RcParams` used to customize matplotlib plots. Options given here will be passed to `matplotlib.rc_context()` and used for all plots produced with this diagnostic.

plot_kwargs: dict, optional
Optional keyword arguments for `iris.plot.plot()`. Dictionary keys are elements identified by `groupby_facet` or `default`, e.g., `CMIP6` if `groupby_facet: project` or `CESM2` if `groupby_facet: dataset`. Dictionary values are dictionaries used as keyword arguments for `iris.plot.plot()`.

pyplot_kwargs: dict, optional
Optional calls to functions of `matplotlib.pyplot`. Dictionary keys are functions of `matplotlib.pyplot`. Dictionary values are used as argument(s) for these functions (if values are dictionaries, these are interpreted as keyword arguments; otherwise a single argument is assumed).

savefig_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.savefig()`. By default, uses `bbox_inches: tight, dpi: 300, orientation: landscape`.

seaborn_settings: dict, optional
Options for `seaborn.set_theme()` (affects all plots). By default, uses `style: ticks`.

var_emissions: str, optional (default: 'cumulative_fco2antt')
Short name of the variable describing the cumulative anthropogenic CO2 emissions. This must be the name of the variable given in the recipe. Note that using the `cumulative_sum()` preprends `cumulative_` to this name.

var_temperature: str, optional (default: 'tas')
Short name of the variable describing the temperature change. This must be the name of the variable given in the recipe.

6.2.10 Zero Emissions Commitment (ZEC)

Diagnostic script to calculate and plot ZEC.

Description

Calculate and plot ZEC (Zero Emission Commitment) Temperature. Requires input data from a base simulation (e.g. 1pctCO2, flat-10) and a dedicated ZEC simulation (e.g. esm-flat10-zec, esm-1pct-brch-1000PgC).

Configuration options in recipe

zec_year
[list, optional (default: [50])] Calculate ZEC for the 20-year average centered around year x, multiple values are possible.

experiments: dict, optional (default: {
'reference': ['esm-flat10', '1pctCO2'], 'simulation': ['esm-flat10-zec', 'esm-1pct-brch-1000PgC', 'esm-1pct-brch-750PgC', 'esm-1pct-brch-2000PgC'] }) When using non-default experiments to calculate ZEC, the experiment setting is required with values for `reference` and `simulation`.

6.2.11 Hour of Maximum Precipitation

Calculate and plot hour of daily maximum precipitation.

Description

This diagnostics calculates the hour of daily maximum precipitation and plots it. The input data needs to be subdaily and of shape (hour, latitude, longitude). The hour dimension should be in local solar time (see `local_solar_time()`).

Author

Manuel Schlund (DLR, Germany)

Configuration options in recipe

caption: str, optional
Figure caption used for provenance tracking. By default, uses "Global map of the hour of the daily maximum precipitation for {alias}.".

cbar_label: str, optional (default: "Hour of daily maximum precipitation")
Colorbar label.

cbar_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.colorbar()`. By default, uses `{orientation: "horizontal"}`.

figure_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.figure()`. By default, uses `constrained_layout: true, ticks: [0, 3, 6, 9, 12, 15, 18, 21]`.

matplotlib_rc_params: dict, optional
Optional `matplotlib.RcParams` used to customize matplotlib plots. Options given here will be passed to `matplotlib.rc_context()` and used for all plots produced with this diagnostic.

method: str, optional (default: *"dft"*)
Method to determine the hour of daily maximum precipitation. Possible options:

- If *"dft"*, apply a discrete Fourier transform (DFT) to the data and use the peak of the first component (see https://mdtf-diagnostics.readthedocs.io/en/latest/sphinx_pods/precip_diurnal_cycle.html).
- If *"harmonic_fit"*, fit a 12hr- plus 24hr-harmonic to the input data and use the peak of the 24hr-harmonic (see Dai, 2024; https://doi.org/10.1007/s00382-024-07182-6). This should give identical/very similar results to *"dft"*, but is much slower.
- If *"max"*, simply use the maximum in the input data.

plot_kwargs: dict, optional
Optional keyword arguments for `iris.plot.pcolormesh()`. By default, uses `cmap: twilight`.

projection: str, optional (default: None)
Projection used for the plot. Needs to be a valid projection class of `cartopy.crs`. Keyword arguments can be specified using the option `projection_kwargs`.

projection_kwargs: dict, optional
Optional keyword arguments for the projection given by `projection`. For map plots, the default keyword arguments `{central_longitude: 10}` are used.

pyplot_kwargs: dict, optional
Optional calls to functions of `matplotlib.pyplot`. Dictionary keys are functions of `matplotlib.pyplot`. Dictionary values are used as argument(s) for these functions (if values are dictionaries, these are interpreted as keyword arguments; otherwise a single argument is assumed). String arguments can contain format strings (e.g., `"{dataset} ({project})"`).

savefig_kwargs: dict, optional
Optional keyword arguments for `matplotlib.pyplot.savefig()`. By default, uses `bbox_inches: tight, dpi: 300, orientation: landscape`.

seaborn_settings: dict, optional
Options for `seaborn.set_theme()` (affects all plots). By default, uses `style: ticks`.

threshold: float, optional (default: 0.0)
Mask grid points where precipitation is lower than the given threshold.

CHAPTER

SEVEN

OBTAINING INPUT DATA

ESMValTool supports input data from climate models participating in CMIP6, CMIP5, CMIP3, and CORDEX as well as observations, reanalysis, and any other data, provided that it adheres to the CF conventions and the data is described in a CMOR table as used in the various Climate Model Intercomparison Projects.

Note

CORDEX support is still work in progress. Contributions, in the form of *pull request reviews* or pull requests are most welcome. We are particularly interested in contributions from people with good understanding of the CORDEX project and its standards.

This section provides an introduction to getting (access to) climate data for use with ESMValTool.

Because the amount of data required by ESMValTool is typically large, it is recommended that you use the tool on a compute cluster where the data is already available, for example because it is connected to an ESGF node. Examples of such compute clusters are Levante and Jasmin, but many more exist around the world.

If you do not have access to such a facility through your institute or the project you are working on, you can request access by applying for the *ENES Climate Analytics Service <https://is.enes.org/sdm-climate-analytics-data/>*.

If the options above are not available to you, ESMValTool also offers a feature to make it easy to download CMIP6, CMIP5, CMIP3, CORDEX, and obs4MIPs from ESGF. ESMValTool also provides support to download some observational dataset from source.

The chapter in the ESMValCore documentation on finding data explains how to configure ESMValTool so it can find locally available data and/or download it from ESGF if it isn't available locally yet.

7.1 Models

If you do not have access to a compute cluster with the data already mounted, ESMValTool can automatically download any required data that is available on ESGF. This is the recommended approach for first-time users to obtain some data for running ESMValTool. For example, run

```
esmvaltool run --search_esgf=when_missing examples/recipe_python.yml
```

to run the default example recipe and automatically download the required data to the directory `~/climate_data`. The data only needs to be downloaded once, every following run will reuse previously downloaded data stored in this directory. See ESGF configuration for a more in depth explanation and the available configuration options.

Alternatively, you can use an external tool called esgpull to download and store files from ESGF.

7.2 Observations

Observational and reanalysis products in the standard CF/CMOR format used in CMIP and required by ESMValTool 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 supports two ways to perform this reformatting (aka 'CMORization'):

1. Using a CMORizer script: The first is to use a CMORizer script to generate a local pool of reformatted data that can readily be used by ESMValTool. This method is described in detail below.
2. Using fixes for on-the-fly CMORization: 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'). Details on this second method are given at the *end of this chapter*.

7.2.1 Tiers

All observational datasets are grouped into in three tiers:

- **Tier 1**: obs4mips and ana4mips datasets. These datasets are publicly and freely available without any license restrictions. These datasets do not need any reformatting and can be used as is with ESMValTool.
- **Tier 2** other freely available datasets that are not obs4mips. There are no license restrictions. These datasets need to be reformatted to be used with ESMValTool ('CMORization', see above).
- **Tier 3** restricted datasets. Datasets which require registration to be downloaded or that can only be obtained upon request from the respective authors. License restrictions do not allow us to redistribute Tier 3 datasets. The data have to be obtained and reformatted by the user ('CMORization', see above).

[!NOTE] .. _tier3_note: For some of the Tier 3 datasets, we obtained permission from the dataset providers to share the data among ESMValTool users on HPC systems. These Tier 3 datasets are marked with an asterisk in the table in section *supported datasets below*.

An overview of the Tier 2 and Tier 3 datasets for which a CMORizing script is available in ESMValTool v2.0 is given in section *supported datasets below*.

A collection of readily CMORized OBS and OBS6 datasets can be accessed directly on CEDA/JASMIN and DKRZ. At CEDA/JASMIN OBS and OBS6 data is stored in the *esmeval* Group Workspace (GWS), and to be granted read (and execute) permissions to the GWS, one must apply at https://accounts.jasmin.ac.uk/services/group_workspaces/esmeval/ ; after permission has been granted, the user is encouraged to use the data locally, and not move it elsewhere, to minimize both data transfers and stale disk usage; to note that Tier 3 data is subject to data protection restrictions; for further inquiries, the GWS is administered by [Valeriu Predoi](mailto:valeriu.predoi@ncas.ac.uk).

7.2.2 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.

ESMValTool provides the `esmvaltool data` command line tool, which can be used to download and format datasets.

To list the available commands, run

```
esmvaltool data --help
```

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

```
esmvaltool data download --help
```

The list of datasets supported by ESMValTool through a CMORizer script can be obtained with:

```
esmvaltool data list
```

Datasets for which auto-download is supported can be downloaded with:

```
esmvaltool data download --original-data-dir ~/RAWOBS [DATASET_LIST]
```

where the `--original-data-dir` option specifies a directory where the orignal data will be stored, `~/RAWOBS` in this example but it can be chosen freely.

Note that all Tier3 and some Tier2 datasets for which auto-download is supported will require an authentication. In such cases enter your credentials in your `~/.netrc` file as explained here.

An entry to the `~/.netrc` should look like:

```
machine [server_name] login [user_name] password [password]
```

Make sure that the permissions of the `~/.netrc` file are set so only you and administrators can read it, i.e.

```
chmod 600 ~/.netrc
ls -l ~/.netrc
```

The latter command should show `-rw-------`.

For other datasets, downloading instructions can be obtained with:

```
esmvaltool data info [DATASET]
```

To CMORize one or more datasets, run:

```
esmvaltool data format --original-data-dir ~/RAWOBS [DATASET_LIST]
```

where the `--original-data-dir` option specifies a directory where the orignal data is stored, `~/RAWOBS` in this example but it can be chosen freely.

Within the directory pointed to by `--original-data-dir`, 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 or CMIP6 CMOR tables for the OBS and OBS6 projects respectively. The resulting output is saved in the output_dir, again following the Tier structure. The output file names follow the convention:

```
[project]_[dataset]_[type]_[version]_[mip]_[short_name]_YYYYMM_YYYYMM.nc
```

where `project` may be OBS (CMIP5 format) or OBS6 (CMIP6 format), `type` may be `sat` (satellite data), `reanaly` (reanalysis data), `ground` (ground observations), `clim` (derived climatologies), `campaign` (aircraft campaign).

At the moment, `esmvaltool data format` supports Python and NCL scripts.

7.2.3 Supported datasets for which a CMORizer script is available

A list of the datasets for which a CMORizers is available is provided in the following table.

Dataset	Variables (MIP)	Tier	Script language
AERONET	od440aer, od550aer, od870aer (AERmon)	3	Python
AGCD	pr (Amon)	2	Python
ANU Climate	pr, tas, tasmin, tasmax (Amon)	3	Python
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
CALIPSO-ICECLOUD*[1]	cli (AMon)	3	NCL
CAMS[6]	nbp (Lmon), fgco2 (Omon)	3	Python
CDS-SATELLITE-ALBEDO[Page 625, 6]	bdalb (Lmon), bhalb (Lmon)	3	Python
CDS-SATELLITE-LAI-FAPAR[Page 625, 6]	fapar (Lmon), lai (Lmon)	3	Python
CDS-SATELLITE-SOIL-MOISTURE[Page 625, 6]	sm (day), sm (Lmon)	3	NCL
CDS-UERRA[Page 625, 6]	sm (E6hr)	3	Python
CDS-XCH4[Page 625, 6]	xch4 (Amon)	3	NCL
CDS-XCO2[Page 625, 6]	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
CLARA-AVHRR	clt, clivi, clwvi, lwp (Amon)	3	NCL
CLOUDSAT-L2	clw, clivi, clwvi, lwp (Amon)	3	NCL
CMAP	pr (Amon)	2	Python
CowtanWay	tasa (Amon)	2	Python
CRU	tas, tasmin, tasmax, pr, clt (Amon), evspsblpot (Emon)	2	Python
CT2019	co2s (Amon)	2	Python
Duveiller2018	albDiffiTr13	2	Python
E-OBS	tas, tasmin, tasmax, pr, psl (day, Amon)	2	Python
EN4	thetao, so, tos, sos (Omon)	2	Python
Eppley-VGPM-MODIS	intpp (Omon)	2	Python
ERA5[2]	cl, clt, evspsbl, evspsblpot, mrro, pr, prsn, ps, psl, ptype, rls, rlds, rlns, rlus[3], rsds, rsns, rsus[Page 625, 3], rsdt, rss, uas, vas, tas, tasmax, tasmin, tdps, ts, tsn (E1hr/Amon), orog (fx)	3	n/a
ERA5-Land[Page 625, 2]	pr	3	n/a
ERA-Interim	cl, cli, clivi, clt, clw, clwvi, evspsbl, hfds, hur, hus, lwp, orog, pr, prsn, prw, ps, psl, rlds, rlut, rlutcs, rsds, rsdt, rss, rsut, rsutcs, sftlf, ta, tas, tasmax, tasmin, tauu, tauv, tdps, tos, ts, tsn, ua, uas, va, vas, wap, zg	3	Python
ERA-Interim-Land	sm (Lmon)	3	Python

continues on next page

Table 1 – continued from previous page

Dataset	Variables (MIP)	Tier	Script language
ESACCI-AEROSOL	abs550aer, od550aer, od550aerStderr, od550lt1aer, od870aer, od870aerStderr (aero)	2	Python
ESACCI-BIOMASS	agb (Lyr, frequency=yr)	2	Python
ESACCI-CLOUD	clivi, clt, cltStderr, clwvi, lwp, rlut, rlutcs, rsut, rsutcs, rsdt, rlus, rsus, rsuscs (Amon)	2	NCL
ESACCI-FIRE	burntArea (Lmon)	2	NCL
ESACCI-LANDCOVER v1.6.1	baresoilFrac, cropFrac, grassFrac, shrubFrac, treeFrac (Lmon)	2	NCL (CMORizer available until ESMValTool v2.11.0)
ESACCI-LANDCOVER v2.0.8	baresoilFrac, cropFrac, grassFrac, shrubFrac, treeFrac (Lmon, frequency=yr)	2	Python (CMORizer available since ESMValTool v2.12.0)
ESACCI-LST	ts (Amon)	2	Python
ESACCI-OC[Page 625, 6]	chl (Omon)	2	Python
ESACCI-OZONE[Page 625, 6]	toz, o3 (AERmon)	2	Python
ESACCI-SEAICE	siconc (SIday, SImon)	2	Python
ESACCI-SEA-SURFACE-SALINITY	sos (Omon)	2	Python
ESACCI-SNOW	snc, snw (day)	2	Python
ESACCI-SOILMOISTURE	sm (Eday, Lmon), smStderr (Eday)	2	Python
ESACCI-SST	tos (Omon, Oday), tosStderr (Oday)	2	Python
ESACCI-WATERVAPOUR	prw (Amon)	3	Python
ESDC	tas, tasmax, tasmin (Amon)	2	Python
ESRL	co2s (Amon)	2	NCL
FLUX-COM*[Page 625, 1]	gpp (Lmon)	3	Python
GCP2018	fgco2 (Omon[4]), nbp (Lmon[Page 625, 4])	2	Python
GCP2020	fgco2 (Omon[Page 625, 4]), nbp (Lmon[Page 625, 4])	2	Python
GHCN	pr (Amon)	2	NCL
GHCN-CAMS	tas (Amon)	2	Python
GISTEMP	tasa (Amon)	2	Python
GLODAP	dissic, ph, talk (Oyr)	2	Python
GLWD	wetlandFrac (Emon)	2	Python
GPCC	pr (Amon)	2	Python
GPCP-SG	pr (Amon)	2	Python
GRACE	lweGrace (Lmon)	3	Python
HadCRUT3	tas, tasa (Amon)	2	NCL
HadCRUT4	tas, tasa (Amon), tasConf5, tasConf95	2	NCL
HadCRUT5	tas, tasa (Amon)	2	Python
HadEX3	txx, tnn, rx1day, rx5day (Amon, yr)	2	Python
HadISST	sic (OImon), tos (Omon), ts (Amon)	2	NCL
HALOE	tro3, hus (Amon)	2	NCL
HWSD	cSoil (Lmon), areacella (fx), sftlf (fx)	3	Python
IAP	thetao, tos (Omon)	2	Python

continues on next page

Table 1 – continued from previous page

Dataset	Variables (MIP)	Tier	Script language
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
JRA-25	cl, clt, clwvi, hus, prw, rlut, rlutcs, rsut, rsutcs, ta, tas, tcw, wap (Amon)	2	Python
JRA-55	cli, clivi, clw, clwvi, clt, prw, rlus, rlut, rlutcs, rsus, rsuscs, rsut, rsutcs, ta, tas, wap (Amon)	2	Python
Kadow2020	tasa (Amon)	2	Python
LAI3g	lai (Lmon)	3	Python
LandFlux-EVAL	et, etStderr (Lmon)	3	Python
Landschuetzer2016	dpco2, fgco2, spco2 (Omon)	2	Python
Landschuetzer2020	spco2 (Omon)	2	Python
MAC-LWP*[Page 625, 1]	lwp, lwpStderr (Amon)	3	NCL
MERRA	cli, clivi, clt, clw, clwvi, hur, hus, lwp, pr, prw, ps, psl, rlut, rlutcs, rsdt, rsut, rsutcs, ta, tas, ts, ua, va, wap, zg (Amon)	3	NCL
MERRA2*[Page 625, 1]	sm (Lmon) clt, pr, evspsbl, hfss, hfls, huss, prc, prsn, prw, ps, psl, rlds, rldscs, rlus, rlut, rlutcs, rsds, rsdscs, rsdt, tas, tasmin, tasmax, tauu, tauv, ts, uas, vas, rsus, rsuscs, rsut, rsutcs, ta, ua, va, tro3, zg, hus, wap, hur, cl, clw, cli, clwvi, clivi (Amon)	3	Python
MLS-AURA*[Page 625, 1]	hur, hurStderr (day)	3	Python
MOBO-DIC-MPIM	dissic (Omon)	2	Python
MOBO-DIC2004-2019	dissic (Omon)	2	Python
MODIS	cliwi, clt, clwvi, iwpStderr, lwpStderr (Amon), od550aer (aero)	3	NCL
MSWEP[Page 625, 2]	pr	3	n/a
MTE*[Page 625, 1]	gpp, gppStderr (Lmon)	3	Python
NCEP-NCAR-R1	clt, hur, hurs, hus, pr, prw, psl, rlut, rlutcs, rsut, rsutcs, sfcWind, ta, tas, tasmax, tasmin, ts, ua, va, wap, zg (Amon) pr, rlut, ua, va (day)	2	Python
NCEP-DOE-R2	clt, hur, prw, ta, wap, pr, tauu, tauv, tos (Amon)	2	Python
NDP	cVeg (Lmon)	3	Python
NIWA-BS*[Page 625, 1]	toz, tozStderr (Amon)	3	NCL
NOAA-CIRES-20CR-V2	clt, clwvi, hus, prw, rlut, rsut, pr, tauu, tauv (Amon)	2	Python
NOAA-CIRES-20CR-V3	clt, clwvi, hus, prw, rlut, rlutcs, rsut, rsutcs (Amon)	2	Python
NOAA-ERSSTv3b	tos (Omon)	2	Python
NOAA-ERSSTv5	tos (Omon)	2	Python
NOAA-MBL-CH4	ch4s (Amon)	2	Python
NOAA-GML-SURFACE-FLASK-CH4	ch4s (Amon)	2	Python

continues on next page

Table 1 – continued from previous page

Dataset	Variables (MIP)	Tier	Script language
NOAA-GML-SURFACE-FLASK-N2O	n2os (Amon)	2	Python
NOAA-GML-SURFACE-FLASK-CO2	co2s (Amon)	2	Python
NOAAGlobalTemp	tasa (Amon)	2	Python
NSIDC-0116-[nh|sh][5]	usi, vsi (day)	3	Python
NSIDC-g02202-[sh]	siconc (SImon)	3	Python
OceanSODA-ETHZ	areacello (Ofx), co3os, dissicos, fgco2, phos, spco2, talkos (Omon)	2	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 (Omon[4])	2	Python
PIOMAS	sit (day)	2	Python
REGEN	pr (day, Amon)	2	Python
Scripps-CO2-KUM	co2s (Amon)	2	Python
TCOM-CH4	ch4 (Amon[4])	2	Python
TCOM-N2O	n2o (Amon[4])	2	Python
TROPFLUX	ts, rlut, rsds, tauu, tauv, hfds (Amon) tos (Omon) hfls (Lmon, Amon)	2	Python
UWisc*[1]	clwvi, lwpStderr (Amon)	3	NCL
WAD2M	wetlandFrac (Emon)	2	Python
WFDE5	tas, pr (Amon, day)	2	Python
WOA	thetao, so, tos, sos (Omon) no3, o2, po4, si (Oyr)	2	Python
Yang2020	dpn2o, no2flux (Omon) areacella (fx)	2	Python

7.3 Datasets in native format

ESMValCore also provides support for some datasets in their native format. In this case, the steps needed to reformat the data are executed as dataset fixes during the execution of an ESMValTool recipe, as one of the first preprocessor steps, see fixing data. 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. Native datasets can be hosted either under a dedicated project (usually done for native model output) or under project `native6` (usually done for native reanalysis/observational products).

A list of all currently supported native datasets is provided here. A detailed description of how to include new native datasets is given here.

To use this functionality, users need to configure their data sources to include the native datasets.

[1] We obtained permission from the dataset provider to share this dataset among ESMValTool users on HPC systems.

[6] Downloading this dataset requires an ECMWF account with an api key in `~/.cdsapirc`. The corresponding licenses need to be accepted at cds or ads.

[2] CMORization is built into ESMValTool through the native6 project, so there is no separate CMORizer script.

[3] Derived on the fly from down & net radiation.

[4] The frequency of this variable differs from the one specified in the table. The correct entry that needs to be used in the recipe can be found in the corresponding section of recipe_check_obs.yml.

[5] The cmoriser requires PROJ>=9.3. Previous version of PROJ will return an error: `Internal Proj Error: proj_create: unhandled axis direction: UNKNOWN)` You can check the version of PROJ in your conda environment by running: `conda list PROJ`.

For project `native6`, more examples of usage can be found in the diagnostics `ERA5_native6` in the recipe examples/recipe_check_obs.yml.

CHAPTER

EIGHT

MAKING A RECIPE OR DIAGNOSTIC

8.1 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.

8.2 Recipe

8.2.1 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.

There is also a lesson available in the ESMValTool tutorial that describes in a step-by-step procedure how to write your own recipe. It can be found here.

8.3 Diagnostic

8.3.1 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.

8.3.2 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.

8.3.3 Example of config dictionary

To be added (use python-style code-block).

8.4 Writing a CMORizer 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.

1. Check if your variable is CMOR standard
2. Store your dataset in the right place
2.1 Downloader script (optional)
3. Create a cmorizer for the dataset
3.1 Cmorizer script written in python
3.2 Cmorizer script written in NCL
4. Run the cmorizing script
5. Naming convention of the observational data files
6. Test the cmorized dataset

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 *Datasets in native format*.

8.4.1 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 /esmvalcore/cmor/tables, differentiated according to the `MIP` they belong to. A more extensive introduction is available in Variables and CMOR Tables. 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/cmip6-custom/.

To create a new custom CMOR table you need to follow these guidelines:

- Provide the `variable_entry` (for CMIP5-style tables) or the variable name (`short_name`) followed by an optional underscore and branding suffix as the key for the variable definition (for CMIP6-style tables);
- Provide the `modeling_realm`;
- Provide the variable attributes, but leave `standard_name` blank unless the variable has a valid standard name. 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` (usually equal to `short_name`). There are also additional variable attributes that can be defined here (see the already available custom variables).

It is recommended to extend the file esmvalcore/cmor/tables/cmip6-custom/CMIP6_custom.json for CMIP6-style tables. For CMIP5-style projects, use an existing custom table from esmvalcore/cmor/tables/cmip5-custom/ as a template, edit the content and save it in the same directory as `CMOR_<short_name>.dat`.

8.4.2 2. Store your dataset in the right place

The folder specified by `--original-data-dir` 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 in the appropriate of these three folders.

For each additional dataset, an entry needs to be made to the file datasets.yml. The dataset entry should contain:

- the correct `tier` information;
- the `source` of the raw data;
- the `last_access` date;
- the `info` that explain how to download the data.

Note that these fields should be identical to the content of the header of the cmorizing script (see Section *3. Create a cmorizer for the dataset*).

2.1 Downloader script (optional)

A Python script can be written to download raw observations from source and store the data in the appropriate tier subdirectory of the folder specified by the `--original-data-dir` flag automatically. There are many downloading scripts available in /esmvaltool/cmorizers/data/downloaders/datasets/ where several data download mechanisms are provided:

- A *wget* get based downloader for http(s) downloads, with a specific derivation for NASA datasets.
- A *ftp* downloader with a specific derivation for ESACCI datasets available from CEDA.
- A Climate Data Store downloader based on *cdsapi*.

Note that the name of this downloading script has to be identical to the name of the dataset.

Depending on the source server, the downloading script needs to contain paths to raw observations, filename patterns and various necessary fields to retrieve the data. Default `start_date` and `end_date` can be provided in cases where raw data are stored in daily, monthly, and yearly files.

The downloading script for the given dataset can be run with:

```
esmvaltool data download --original-data-dir </path/to/save/data>  <dataset-name>
```

The options `--start` and `--end` can be added to the command above to restrict the download of raw data to a time range. They will be ignored if a specific dataset does not support it (i.e. because it is provided as a single file). Valid formats are `YYYY`, `YYYYMM` and `YYYYMMDD`. By default, already downloaded data are not overwritten unless the option `--overwrite=True` is used.

8.4.3 3. Create a cmorizer for the dataset

There are many cmorizing scripts available in /esmvaltool/cmorizers/data/formatters/datasets/ where solutions to many kinds of format issues with observational data are addressed. These scripts are either written in Python or in NCL.

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.

3.1 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: mte.py.

All the necessary information about the dataset to write the filename correctly, and which variable is of interest, is stored in a separate configuration file: MTE.yml in the directory `ESMValTool/esmvaltool/cmorizers/data/cmor_config/`. Note that both the name of this configuration file and the cmorizing script have to be identical to the name of your dataset. 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 *5. Naming convention of the observational data files*). Another global attribute is `reference` which includes a `doi` related to the dataset. Please see the section adding references on how to add reference tags to the `reference` section in the configuration file. If a single dataset has more than one reference, it is possible to add tags as a list e.g. `reference: ['tag1', 'tag2']`. The third part in the configuration file defines the variables that are supposed to be cmorized.

The actual cmorizing script `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, cfg_user, start_date, end_date):
```

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, `cfg` to the dataset-specific configuration file, `cfg_user` to the configuration object (which behaves basically like a dictionary), `start_date` to the start of the period to format, and `end_date` to the end of the period to format. If not needed, the last three arguments can be ignored using underscores. The return value of this function is ignored. Note that this function will trigger a Codacy style issue because it has more than 5 arguments; you can ignore that issue in line with the *Code quality* guidance.

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 esmvaltool.cmorizers.data 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.

3.2 Cmorizer script written in NCL

Warning

Writing new NCL code is not recommended because the NCL interpreter is no longer maintained.

Find here an example of a cmorizing script, written for the `ESACCI XCH4` dataset that is available on the Copernicus Climate Data Store: 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 are imported by default into your CMORizer. 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.

8.4.4 4. Run the cmorizing script

The cmorizing script for the given dataset can be run with:

```
esmvaltool data format --config_dir </path/to/config/dir/> <dataset-name>
```

The options `--start` and `--end` can be added to the command above to restrict the formatting of raw data to a time range. They will be ignored if a specific dataset does not support it (i.e. because it is provided as a single file). Valid formats are `YYYY`, `YYYYMM` and `YYYYMMDD`.

Note

The `output_dir` 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 if that tier folder is not already available, and then a folder named after the dataset. In this folder the cmorized data set will be stored as a NetCDF file. The cmorized dataset will be automatically moved to the correct tier subfolder of your OBS or OBS6 directory if the option `--install=True` is used in the command above and no such directory was already created.

If your run was successful, one or more NetCDF files are produced in your output directory.

If a downloading script is available for the dataset, the downloading and the cmorizing scripts can be run in a single command with:

```
esmvaltool data prepare --config_dir </path/to/config/dir/> <dataset-name>
```

Note that options from the `esmvaltool data download` and `esmvaltool data format` commands can be passed to the above command.

8.4.5 5. 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 data-local-esmvaltool.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;
- 200301-201812: Is the period the dataset spans with `200301` being the start year and month, and `201812` 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 *data-local.yml <https://github.com/ESMValGroup/ESMValCore/blob/main/esmvalcore/config/configurations/data-local.yml>* file).

8.4.6 6. 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.

CHAPTER

NINE

CONTRIBUTING TO THE COMMUNITY

Contributions are very welcome!

This chapter explains how to contribute to ESMValTool. 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, maintenance of recipes, mailing list and chat participation, community help/building, education and outreach.

If you have a bug or other issue to report, please open an issue on the issues tab on the ESMValTool github repository.

In case anything is unclear feel free to contact us for more information and help, e.g. on our GitHub Discussions page.

9.1 Contributing code and documentation

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 with the implementation.

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, for example by starting a discussion on our discussions page.

9.1.1 Getting started

See *Install from source* for instructions on how to set up a development installation.

New development should preferably be done in the ESMValTool GitHub repository. However, for scientists requiring confidentiality, private repositories are available, see *Moving work from the private to the public repository* for more information. The default git branch is `main`. 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*, *Codacy to analyse your code*, and *readthedocs to build the documentation*. It's also easier to get help from other developers if your code is visible in a pull request.

Please review the results of the automatic checks below your pull request. If one of the tests shows a red cross instead of a green checkmark, please click the `Details` link behind the failing check and try to solve the issue. Ask @ESMValGroup/tech-reviewers for help if you do not know how to fix the failing check. Note that this kind of automated checks make it easier to *review code*, but they are not flawless. Preferably Codacy code quality checks pass, however a few remaining hard to solve Codacy issues are still acceptable. If you suspect Codacy may be wrong, please ask by commenting on your pull request.

9.1.2 Checklist for pull requests

To clearly communicate up front what is expected from a pull request, we have the following checklist. Please try to do everything on the list before requesting a review. If you are unsure about something on the list, please ask the @ESMValGroup/tech-reviewers or @ESMValGroup/science-reviewers for help by commenting on your (draft) pull request or by starting a new discussion.

In the ESMValTool community we use *pull request reviews* to ensure all code and documentation contributions are of good quality. The icons indicate whether the item will be checked during the *Technical review* or *Scientific review*.

All pull requests

- *The pull request has a descriptive title*
- Code is written according to the *code quality guidelines*
- *Documentation* is available
- *Tests* run successfully
- The *list of authors* is up to date
- Changed dependencies are *added or removed correctly*
- The *checks shown below the pull request* are successful

If a pull request introduces a change that causes a recipe to no longer run successfully (*breaking change*), or which results in scientifically significant changes in results (*science change*), additional requirements defined in the *backward compatibility policy* apply. These include in particular:

- Instructions for the release notes to assist *recipe developers* to adapt their recipe in light of the *backward-incompatible change* available.
- If applicable, instructions for *recipe developers* working on *user recipes* to enable them to adapt their code related to *backward-incompatible changes* available (see ESMValTool_Tutorial: issue #263) available.
- Core development team tagged to notify them of the *backward-incompatible change*, and give at least 2 weeks for objections to be raised before merging to the main branch. If a strong objection is raised the backward-incompatible change should not be merged until the objection is resolved.
- Information required for the "*backward-incompatible changes*" section in the PR that introduces the *backward-incompatible change* available.

New or updated recipe and/or diagnostic

See *Making a new diagnostic or recipe* for detailed instructions.

- *Recipe runs successfully*
- *Recipe and diagnostic documentation* is available
- *Figure(s) and data* look as expected from literature
- *Provenance information* has been added

New or updated data reformatting script

See *new dataset* for detailed instructions.

- *Dataset documentation* is available
- The dataset has been *added to the CMOR check recipe*
- Numbers and units of the data look *physically meaningful*

9.1.3 Pull request title

The title of a pull request should clearly describe what the pull request changes. If you need more text to describe what the pull request does, please add it in the description. The titles of pull requests are used to compile the *Changelog*, therefore it is important that they are easy to understand for people who are not familiar with the code or people in the project. Descriptive pull request titles also makes it easier to find back what was changed when, which is useful in case a bug was introduced.

9.1.4 Code quality

To increase the readability and maintainability or the ESMValTool source code, we aim to adhere to best practices and coding standards. 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, e.g. no more than 50 lines long.

We include checks for Python, R, NCL, and yaml files, most of which are described in more detail in the sections below. This includes checks for invalid syntax and formatting errors. *Pre-commit* is a handy tool that can run all of these checks automatically just before you commit your code. It knows which tool to run for each filetype, and therefore provides a convenient way to check your code.

To enable pre-commit checks (these will be run every time you commit something), go to the `ESMValTool` directory and run:

```
pre-commit install
```

just once, after you have cloned the ESMValTool repository.

If you have installed the pre-commit checks, but for some reason need to skip them (not recommended), use the `-n` flag:

```
git commit -n
```

To run the automatic code formatting and basic code quality checks, use the command:

```
pre-commit run -a
```

Python

The standard document on best practices for Python code is PEP8 and there is PEP257 for code documentation. We make use of numpy style docstrings to document Python functions that are visible on readthedocs.

To automatically format and check if code adheres to the standard, we use ruff. Automatic formatting and basic code quality checks are enforced through pre-commit. In addition to these basic quality checks, we use ruff to make recommendations on how your code could be even better.

To automatically format your code and perform basic quality checks, run:

```
pre-commit run -a
```

To perform the additional code quality checks and automatically fix many of the reported issues, run:

```
ruff check --fix esmvaltool/diag_scripts/your_diagnostic/your_script.py
```

Each code quality check performed by ruff is called a `rule` and has a unique code, e.g. `RET504`. For a detailed explanation of the purpose of these rules, you can visit the rules page on the ruff website or run the command:

```
ruff rule RET504
```

where `RET504` is an arbitrary example of a rule code.

When you make a pull request, adherence to the Python development best practices is checked in two ways:

1. As part of the unit tests, pre-commit is run by CircleCI, see the section on *Tests* for more information.
2. Codacy is a service that runs ruff (and other code quality tools) on changed files and reports the results. Click on the 'Codacy Static Code Analysis' check entry below your pull request and then click the 'Resolve' button or the 'View more details on Codacy Production' link to see the results of the static code analysis done by Codacy. If you need to log in, you can do so using your GitHub account. If Codacy says that there are issues, but you cannot see them on the Codacy webpage for some reason, please run the command `ruff check` on your own computer to see what the issues are.

A pull request should preferably not introduce new code quality issues. However, we understand that there is a limit to how much time can be spent on polishing code, so up to 10 new issues reported by Codacy/ruff is still an acceptable amount. However, a pull request where the CircleCI/pre-commit tests are failing cannot be merged.

Note

The automatic code quality checks by ruff are really helpful to improve the quality of your code, but they are not flawless. If you suspect ruff or Codacy may be wrong or you just need help, please ask the @ESMValGroup/tech-reviewers by commenting on your pull request.

NCL

Warning

Writing new NCL code is not recommended because the NCL interpreter is no longer maintained.

Because there is no standard best practices document for NCL, we use PEP8 for NCL code as well, with some minor adjustments to accommodate 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. More information on the `nclcodestyle` command can be found *here*.

R

Best practices for R code are described in The tidyverse style guide. We currently don't check adherence to this, but we strongly encourage any new R diagnostic to be checked against the style guide by using lintr. Please use styler to automatically format your code according to this style guide. To run these tools, uncomment these lines:

Listing 1: R configuration in `.pre-commit-config.yaml`.

```
# - repo: https://github.com/lorenzwalthert/precommit/  # Checks for R
#   rev: 'v0.4.2'
#   hooks:
#     - id: style-files  # styler
#     - id: lintr
```

and run the commands:

```
pre-commit run style-files --files esmvaltool/diag_scripts/your_script.R
pre-commit run lintr --files esmvaltool/diag_scripts/your_script.R
```

These pre-commit hooks have been disabled by default because over the past few years all diagnostics that have been contributed were written in Python. However, if R contributions pick up again, we would be happy to enable these pre-commit hooks again. In the future we would also like to make use of goodpractice to assess the quality of R code.

YAML

Please use yamllint to check that your YAML files do not contain mistakes. `yamllint` checks for valid syntax, common mistakes like key repetition and cosmetic problems such as line length, trailing spaces, wrong indentation, etc. When the tool complains about the maximum line length or too many spaces, please use your own best judgement about whether solving the issue will make your recipe more readable.

Any text file

A generic tool to check for common spelling mistakes is codespell.

9.1.5 Documentation

The documentation lives on docs.esmvaltool.org and is built using Sphinx. There are two main ways of adding documentation:

1. As written text in the directory doc/sphinx/source. When writing reStructuredText (`.rst`) files, please try to limit the line length to 80 characters and always start a sentence on a new line. This makes it easier to review changes to documentation on GitHub.
2. As docstrings or comments in code. For Python code, the docstrings of Python modules, classes, and functions that are mentioned in doc/sphinx/source/api are used to generate documentation. This results in the *ESMValTool Code API Documentation*.

What should be documented

See also *Recipe and diagnostic documentation* and *Dataset documentation*.

Any code documentation that is visible on docs.esmvaltool.org should be well written and adhere to the standards for documentation for the respective language. Note that there is no need to write extensive documentation for functions that are not visible in the online documentation. However, a short description in the docstring helps other contributors to understand what a function is intended to do and and what its capabilities are. For short functions, a one-line docstring is usually sufficient, but more complex functions might require slightly more extensive documentation.

How to build and view the documentation

Whenever you make a pull request or push new commits to an existing pull request, readthedocs will automatically build the documentation. The link to the documentation will be shown in the list of checks below your pull request, click 'Details' behind the check `docs/readthedocs.org:esmvaltool` to preview the documentation. If all checks were successful, you may need to click 'Show all checks' to see the individual checks.

To build the documentation on your own computer, go to the directory where the repository was cloned and run

```
sphinx-build doc/sphinx/source/ doc/sphinx/build/
```

or

```
sphinx-build -Ea doc/sphinx/source/ doc/sphinx/build/
```

to build it from scratch.

Alternatively, the pixi task `doc` can be used:

```
pixi run doc
```

Make sure that your newly added documentation builds without warnings or errors and looks correctly formatted. readthedocs will build the documentation with the command

```
sphinx-build -W doc/sphinx/source/ doc/sphinx/build/
```

to catch mistakes that can be detected automatically.

The configuration file for Sphinx is doc/sphinx/source/conf.py and the configuration file for ReadTheDocs is .readthedocs.yaml.

When reviewing a pull request, always check that the documentation checks shown below the pull request were successful. Successful checks have a green ✓ in front, a means the test job failed.

Integration with the ESMValCore documentation

The ESMValCore documentation is hosted as a subproject of the ESMValTool documentation on readthedocs. To link to a section from the ESMValCore documentation from the reStructuredText (`.rst`) files, use the usual `:ref:` but prefix the reference with `esmvalcore:`. For example, `:ref:`esmvalcore:recipe`` to link to The recipe format.

9.1.6 Tests

To check various aspects of the recipes and code, there tests available in the tests directory.

Whenever you make a pull request or push new commits to an existing pull request, these tests will be run automatically on CircleCI. The results appear at the bottom of the pull request. Click on 'Details' for more information on a specific test job. To see some of the results on CircleCI, you may need to log in. You can do so using your GitHub account.

To run the tests on your own computer, go to the directory where the repository is cloned and run the command `pytest` or `pixi run test`.

Have a look at *Testing recipes* for information on testing recipes.

Every night, more extensive tests are run to make sure that problems with the installation of the tool are discovered by the development team before users encounter them. These nightly tests have been designed to mimic the installation procedures described in the documentation, e.g. in the *Installation* chapter. The nightly tests are run using both CircleCI and GitHub Actions, the result of the tests ran by CircleCI can be seen on the CircleCI project page and the result of the tests ran by GitHub Actions can be viewed on the Actions tab of the repository.

The configuration of the tests run by CircleCI can be found in the directory .circleci, while the configuration of the tests run by GitHub Actions can be found in the directory .github/workflows.

When reviewing a pull request, always check that all test jobs on CircleCI were successful. Successful test jobs have a green ✓ in front, a means the test job failed.

9.1.7 List of authors

If you make a contribution to ESMValTool and you would like to be listed as an author (e.g. on Zenodo), please add your name to the list of authors in `CITATION.cff` and generate the entry for the `.zenodo.json` file by running the commands

```
pip install cffconvert
cffconvert --infile CITATION.cff --format zenodo --outfile .zenodo.json
```

Presently, this method unfortunately discards entries *communities* and *grants* from that file; please restore them manually.

Note that authors of recipes and/or diagnostics also need to be added to the file esmvaltool/config-references.yml, see *Recording provenance* for more information.

9.1.8 Dependencies

We follow the guidelines set out in SPEC-0 regarding the supported Python versions and dependencies.

Before considering adding a new dependency, carefully check that the license of the dependency you want to add and any of its dependencies are compatible with the Apache 2.0 license that applies to the ESMValTool. Note that GPL version 2 license is considered incompatible with the Apache 2.0 license, while the compatibility of GPL version 3 license with the Apache 2.0 license is questionable. See this statement by the authors of the Apache 2.0 license for more information.

When adding or removing dependencies, please consider applying the changes in the following locations in `pyproject.toml`:

- `dependencies` contains dependencies that can be installed from PyPI
- `[tool.pixi.dependencies]` contains dependencies that can be installed from conda-forge

It is strongly preferred that those two lists are kept in sync, apart from differences in how packages are named. Run the command `pixi lock --no-install` after making changes to the dependencies to update the `pixi.lock` file, which is used to make sure that the same versions of packages are installed for all ESMValTool developers.

In addition to these core dependencies, there are also optional dependencies for diagnostics that need R or NCL. These are listed in the sections `[tool.pixi.feature.r]` and `[tool.pixi.feature.ncl]` in respectively.

There are also three feature groups for development dependencies:

- `[tool.pixi.feature.dev]` contains tools that are useful for development
- `[tool.pixi.feature.doc]` contains tools that are needed to build the documentation
- `[tool.pixi.feature.test]` contains tools that are needed to run the tests

Note that packages may have a different name on conda-forge than on PyPI or CRAN.

Tip

When reviewing a pull request where dependencies are added or removed, always check that the changes have been applied to both the PyPI and the conda-forge dependencies and that the `pixi.lock` file has been updated by running `pixi lock` after the changes were made.

9.1.9 Pull request checks

To check that a pull request is up to standard, several automatic checks are run when you make a pull request. Read more about it in the *Tests* and *Documentation* sections. Successful checks have a green ✓ in front, a means the check failed.

If you need help with the checks, please ask the technical reviewer of your pull request for help. Ask @ESMValGroup/tech-reviewers if you do not have a technical reviewer yet.

If the checks are broken because of something unrelated to the current pull request, please check if there is an open issue that reports the problem and create one if there is no issue yet. You can attract the attention of the @ESMValGroup/esmvaltool-coreteam by mentioning them in the issue if it looks like no-one is working on solving the problem yet. The issue needs to be fixed in a separate pull request first. After that has been merged into the `main` branch and all checks are green again on the `main` branch, merge it into your own branch to get the tests to pass.

When reviewing a pull request, always make sure that all checks were successful. If the Codacy check keeps failing, please run prospector locally. If necessary, ask the pull request author to do the same and to address the reported issues.

See the section on *code_quality* for more information. Never merge a pull request with failing CircleCI or readthedocs checks.

9.2 Making a new diagnostic or recipe

9.2.1 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.

9.2.2 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, and R 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
- 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
- ncl: esmvaltool/diag_scripts/examples/diagnostic.ncl

For an explanation of the recipe format, you might want to read about the ESMValTool recipe and have a look at the available preprocessor functions. For further inspiration, check out the already *available recipes and diagnostics*.

There is a directory esmvaltool/diag_scripts/shared for code that is shared by many diagnostics. This directory contains code for creating common plot types, generating output file names, selecting input data, and other commonly needed functions. See *Shared diagnostic script code* for the documentation of the shared Python code.

9.2.3 Re-using existing code

Always make sure your code is or can be released under a license that is compatible with the Apache 2.0 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 reuse, 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.

9.2.4 Recipe and diagnostic documentation

This section describes how to document a recipe. For more general information on writing documentation, see *Documentation*.

On readthedocs

Recipes should have a page in the *Recipes* chapter which describes what the recipe/diagnostic calculates.

When adding a completely new recipe, please start by copying doc/sphinx/source/recipes/recipe_template.rst.template to a new file `doc/sphinx/source/recipes/recipe_<name of diagnostic>.rst` and do not forget to add your recipe to the index.

Fill all sections from the template:

- Add a brief description of the method
- Add references
- Document recipe options for the diagnostic scripts
- Fill in the list of variables required to run the recipe
- Add example images

An example image for each type of plot produced by the recipe should be added to the documentation page to show the kind of output the recipe produces. The '.png' files can be stored in a subdirectory specific for the recipe under doc/sphinx/source/recipes/figures and linked from the recipe documentation page. A resolution of 150 dpi is recommended for these image files, as this is high enough for the images to look good on the documentation webpage, but not so high that the files become large.

By default, the first example image will be used for the automatically generated gallery. To select a specific image for the gallery, you can use the following syntax for an arbitrary amount of images:

```
.. _fig_name:
..
  gallery
.. figure:: /recipes/figure/recipe_name/figure-name.png
```

To not have any figure appear in the gallery, you can include a

```
..
  no-gallery
```

anywhere in your recipe documentation file.

In the recipe

Fill in the `documentation` section of the recipe as described in Recipe section: documentation and add a `description` to each diagnostic entry. Please note that the `maintainer` entry is per se not necessary to run a recipe, but mandatory for recipes within the ESMValTool repository (enforced by a unit test). If no maintainer is available, use the single entry `unmaintained`. When reviewing a recipe, check that these entries have been filled with descriptive content.

In the diagnostic scripts

Functions implementing scientific formula should contain comments with references to the source paper(s) and formula number(s).

When reviewing diagnostic code, check that formulas are implemented according to the referenced paper(s) and/or other resources and that the computed numbers look as expected from literature.

9.2.5 Diagnostic output

Typically, diagnostic scripts create plots, but any other output such as e.g. text files or tables is also possible. Figures should be saved in the `plot_dir`, either in both `.pdf` and `.png` format (preferred), or respect the configuration option `output_file_type` . Data should be saved in the `work_dir`, preferably as a `.nc` (NetCDF) file, following the CF-Conventions as much as possible.

Have a look at the *example scripts* for how to access the value of `work_dir`, `plot_dir`, and `output_file_type` from the diagnostic script code. More information on the interface between ESMValCore and the diagnostic script is available here and the description of the *Output* may also help to understand this.

If a diagnostic script creates plots, it should save the data used to create those plots also to a NetCDF file. If at all possible, there will be one NetCDF file for each plot the diagnostic script creates. There are several reasons why it is useful to have the plotted data available in a NetCDF file:

- for interactive visualization of the recipe on a website
- for automated regression tests, e.g. checking that the numbers are still the same with newer versions of libraries

If the output data is prohibitively large, diagnostics authors can choose to implement a `write_netcdf: false` diagnostic script option, so writing the NetCDF files can be disabled from the recipe.

When doing a scientific review, please check that the figures and data look as expected from the literature and that appropriate references have been added.

9.2.6 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 provided that the provenance tree is small, also plotted in an SVG file for human inspection. In addition to provenance information, a caption is also added to the plots.

Provenance information from the recipe is automatically recorded by ESMValCore, whereas diagnostic scripts must include code specifically to record provenance. See below for documentation of provenance attributes that can be included in a recipe. When contributing a diagnostic, please make sure it records the provenance, and that no warnings related to provenance are generated when running the recipe. To allow the ESMValCore to keep track of provenance (e.g. which input files were used to create what plots by the diagnostic script), it needs the Information provided by the diagnostic script to ESMValCore.

Note

Provenance is recorded by the `esmvaltool` command provided by the ESMValCore package. No `*_provenance.xml` files will be generated when re-running just the diagnostic script with the command that is displayed on the screen during a recipe run, because that will only run the diagnostic script.

Provenance items provided by the recipe

Provenance tags can be added in several places in the recipe. The Recipe section: documentation section provides information about the entire 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.

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
- `long_names` a list of long names of used variables, e.g. Air Temperature

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, see References configuration file for an introduction. It is also possible to add custom provenance information by adding items to each category in this file. In this file, the information is written in the form

```
key:
  value: description
```

for example

```
plot_types:
  errorbar: error bar plot
```

To use these items, include them in the provenance record dictionary in the form `key: [value]` i.e. for the example above as `'plot_types': ['errorbar']`.

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:

Recording provenance in a Python diagnostic script

Always use *esmvaltool.diag_scripts.shared.run_diagnostic()* at the end of your script:

```
if __name__ == '__main__':
  with run_diagnostic() as config:
      main(config)
```

Create a `provenance_record` for each diagnostic file (i.e. image or data file) that the diagnostic script outputs. The `provenance_record` is a dictionary of provenance items, for example:

```
provenance_record = {
      'ancestors': ancestor_files,
      'authors': [
          'andela_bouwe',
          'righi_mattia',
      ],
      'caption': caption,
      'domains': ['global'],
      'plot_types': ['zonal'],
      'references': [
          'acknow_project',
      ],
      'statistics': ['mean'],
    }
```

To save a matplotlib figure, use the convenience function *esmvaltool.diag_scripts.shared.save_figure()*. Similarly, to save Iris cubes use *esmvaltool.diag_scripts.shared.save_data()*. Both of these functions take `provenance_record` as an argument and log the provenance accordingly. Have a look at the example Python diagnostic in esmvaltool/diag_scripts/examples/diagnostic.py for a complete example.

For any other files created, you will need to make use of a *esmvaltool.diag_scripts.shared.ProvenanceLogger* to log provenance. Include the following code directly after the file is saved:

```
with ProvenanceLogger(cfg) as provenance_logger:
      provenance_logger.log(diagnostic_file, provenance_record)
```

The full path of a `diagnostic_file` can be obtained using *esmvaltool.diag_scripts.shared.get_diagnostic_filename*.

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.

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
```

9.2.7 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 script provenance, see *recording-provenance*).
- 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`.

9.2.8 Testing recipes

To test a recipe, you can run it yourself on your local infrastructure or you can ask the @esmvalbot to run it for you. To request a run of `recipe_xyz.yml`, write the following comment below a pull request:

```
@esmvalbot Please run recipe_xyz.yml
```

Note that only members of the @ESMValGroup/esmvaltool-developmentteam can request runs. The memory of the @esmvalbot is limited to 16 GB and it only has access to data available at DKRZ.

When reviewing a pull request, at the very least check that a recipes runs without any modifications. For a more thorough check, you might want to try out different datasets or changing some settings if the diagnostic scripts support those. A simple *tool* is available for testing recipes with various settings.

9.2.9 Detailed checklist for reviews

This (non-exhaustive) checklist provides ideas for things to check when reviewing pull requests for new or updated recipes and/or diagnostic scripts.

Technical reviews

Documentation

Check that the scientific documentation of the new diagnostic has been added to the user's guide:

- A file `doc/sphinx/source/recipes/recipe_<diagnostic>.rst` exists
- New documentation is included in `doc/sphinx/source/recipes/index.rst`
- Documentation follows template doc/sphinx/source/recipes/recipe_template.rst.template
- Description of configuration options
- Description of variables
- Valid image files
- Resolution of image files (~150 dpi is usually enough; file size should be kept small)

Recipe

Check yaml syntax (with `yamllint`) and that new recipe contains:

- Documentation: description, authors, maintainer, references, projects
- Provenance tags: themes, realms

Diagnostic script

Check that the new diagnostic script(s) meet(s) standards. This includes the following items:

- In-code documentation (comments, docstrings)
- Code quality (e.g. no hardcoded pathnames)
- No Codacy errors reported
- Reuse of existing functions whenever possible
- Provenance implemented

Run recipe

Make sure new diagnostic(s) is working by running the ESMValTool with the recipe.

Check output of diagnostic

After successfully running the new recipe, check that:

- NetCDF output has been written
- Output contains (some) valid values (e.g. not only nan or zeros)
- Provenance information has been written

Check automated tests

Check for errors reported by automated tests

- Codacy
- CircleCI
- Documentation build

Scientific reviews

Documentation added to user's guide

Check that the scientific documentation of the new diagnostic in `doc/sphinx/source/recipes/recipe_<diagnostic>.rst`:

- Meets scientific documentation standard and
- Contains brief description of method
- Contains references
- Check for typos / broken text
- Documentation is complete and written in an understandable language
- References are complete

Recipe

Check that new recipe contains valid:

- Documentation: description, references
- Provenance tags: themes, realms

Diagnostic script

Check that the new diagnostic script(s) meet(s) scientific standards. This can include the following items:

- Clear and understandable in-code documentation including brief description of diagnostic
- References
- Method / equations match reference(s) given

Run recipe

Make sure new diagnostic(s) is working by running the ESMValTool.

Check output of diagnostic

After successfully running the new recipe, check that:

- Output contains (some) valid values (e.g. not only nan or zeros)
- If applicable, check plots and compare with corresponding plots in the paper(s) cited

9.3 ESMValTool policy on backward compatibility

9.3.1 Motivation

Development of recipes or conducting project-related work may require a rather long period of time during which new versions of the ESMValTool might become available. For a good user experience and a seamless workflow, users and developers need to know before upgrading to a new version if and how their work might be affected (backward compatibility). This includes, for instance, information about changes to technical features such as syntax of recipes and configuration files, and interfaces of shared functions, but also changes that affect the results of an ESMValTool run, e.g. modification of algorithms or changes to the order of operators. It is therefore essential that users and developers have the best advice on how and when to upgrade to new versions.

While trying to minimise the impact of new code developments on users and developers by maintaining backward compatibility where possible, this cannot always be guaranteed. A very restrictive policy might delay the ESMValTool development and make it more complex for developers to contribute.

This document outlines the key principles of an updated ESMValTool policy on backward compatibility.

9.3.2 Definitions

Release: A numbered version of ESMValCore / ESMValTool that has been released to the community, e.g. 2.4.0. This policy relates only to backward compatibility of releases, not to interim revisions of the main branch. Release numbers are of the format x.y.z, where:

- x indicates a major release
- y indicates a minor release
- z indicates a patch release

Backward-incompatible change: A change in ESMValCore or ESMValTool that causes a recipe to no longer run successfully (a *breaking change*), or which results in scientifically significant changes in results (a *science change*).

Breaking change: A change which causes a previously working recipe to no longer run successfully.

Science change: A change that alters scientific results. We do not formally distinguish between trivial science changes (e.g. from changes in the order of calculations) and more significant changes that would affect interpretation, although the detail that we communicate will share any understanding that we have regarding expected impact.

Benign third-party dependency changes: A change over which we have no control, but which we believe will only have trivial technical impacts (such as a change in font). Such changes are outside of the scope of this policy, though we will communicate about those we are aware of.

Developer of backward-incompatible change: For the purpose of this policy, developer is the individual that is responsible for the pull request (PR) that is not backward compatible.

Recipe developer: Someone who is developing a recipe that is not (yet) integrated into the repository.

Recipe user: For the purpose of this policy, a *recipe user* is anyone who runs a recipe using a *release* of ESMValTool. In this context, someone can be both a *recipe developer* and a *recipe user*, but they perform different activities in each capacity.

Recipe maintainer: First contact point for *integrated recipes* in case of problems with that recipe (see also *Maintaining a recipe*).

Integrated recipes: Recipes that are contained within the main branch of the ESMValTool repository, and can therefore be updated by any developer in line with the above guidance. Note that the recipe can be updated by someone other than the original author.

User recipes: Recipes developed by any developer outside of the main branch of the repository (i.e. on a dev/feature branch or outside the repository completely), and therefore cannot be updated by anyone else.

9.3.3 Scope

The ESMValTool and ESMValCore policy on backward compatibility aims at balancing two competing needs: the occasional need of improvements or maintenance to break backward compatibility and the need for stability for existing users and developers. The following aspects are covered by this policy:

- Key principles and approaches to backward compatibility
- Guidelines and requirements for *developers of backward-incompatible changes*
- Communication with users and developers about *backward-incompatible changes*

Not within the scope of this policy are:

- Versioning scheme of ESMValTool
- Breakage of recipes due to changes in input data or dependencies. This is covered by the *broken recipe policy*.

9.3.4 Expectations of developers, users & funders

Stakeholders and their expectations and aims:

Projects / Funders

- Aim to facilitate scientific discovery
- Expect deliverables, e.g. new features/recipes
- Expect reproducible results

Recipe users

- Expect the recipe to work
- Expect the recipe to be easy to run
- Expect reproducible results
- Expect easy installation of ESMValTool

Recipe developers

- Develop recipes
- Expect their recipe to keep working with every new *release* of ESMValCore
- Expect ESMValCore bugfixes and new features to become available quickly
- Expect reproducible results
- Expect easy installation of ESMValTool

Core developers and *recipe maintainers*

- Fix bugs
- Add ESMValCore features requested by *recipe developers*
- Try to accommodate ESMValCore features contributed to by *recipe developers*
- Maintain existing recipes
- Add new recipes
- Try to help (other) *recipe developers* with contributing their recipe
- Try to make installation as easy as possible

There is a tension between making new features available and keeping everything as is. New features facilitate scientific discovery because they enable *recipe developers* to do new research (e.g. analyse more data, new data, or perform a different analysis). Ensuring that every recipe ever made works with every new feature is technically a lot of work, more than we have funding for. Therefore we need to make sure that new features are added regularly, but we respect the timescale on which *recipe developers* work when removing outdated features. Writing a paper and getting it published may take up to a year, so this seems a good timescale for larger changes. For changes that only affect a few users, shorter timescales could be acceptable. It is also good to note that we are part of a large software ecosystem (ESMValTool currently depends on over 500 different software packages), so we may not always be able to control at what pace changes are made to the software that we depend upon.

Two-way communication about new and removed features is needed to make this work. This requires active involvement from both the people developing the new features and the *recipe developers*. ESMValTool core developers and ESMValCore core developers need to make sure they clearly communicate changes. In the first place, this is done by writing good descriptions in issues and pull requests on GitHub, but some of this material also makes it to the changelog (where the GitHub pull requests are linked). It is highly recommended to communicate a relevant selection (e.g. important new, scheduled for removal, and removed features) also by other means, to ensure we reach as many people potentially affected as possible (see *Guidance on handling *backward-incompatible changes** section below). We organize *monthly community* meetings where *recipe developers* can learn about the latest developments and everyone is welcome to join, ask questions, and provide feedback.

To meet the needs of users and funders, we should take reproducibility of older results seriously, but this should not hold us back from advancing our tools. We can support this by uploading a well tested container image to an archive that provides a DOI and by providing clear instructions on how to use such containers.

9.3.5 Helping developers to upgrade

Recipe users of ESMValTool should be able to successfully run *integrated recipes* using a *release*, since all *backward-incompatible changes* introduced between *releases* will have been fixed before the *release* is created. Please note the *broken recipe policy*.

However, *recipe developers* working on *user recipes* must be provided with information to enable them to adapt their code to resolve issues related to *backward-incompatible changes* when *backward-incompatible changes* are introduced to the main branch / when a *release* of ESMValTool is created.

Guidance on handling *backward-incompatible changes*

As well as helping users to handle *backward-incompatible changes*, the policy and surrounding tools must help developers avoid making *backward-incompatible changes*. Not many ideas are developed on this yet, but components should include:

- Testing; *backward-incompatible changes* should be discovered as early in the development process as possible. This motivates continued investment in automated testing. To discover *backward-incompatible changes* early on in the development cycle, every night a selection of recipes is run on CircleCI. A recipe can be added to the test suite by adding it to the directory esmvaltool/recipes/testing. Only add recipes that require a small amount of data, i.e. considerably less than a gigabyte.
- Guidance on how to minimise the likelihood of introducing *backward-incompatible changes* and how to use deprecation warnings when needed (see developer guidance).
- *Instructions on how to provide text for the release notes* to assist *recipe developers* to adapt their recipe in light of the *backward-incompatible change*
- General instructions for *recipe developers* working on *user recipes* to enable them to adapt their code related to *backward-incompatible changes* (see ESMValTool_Tutorial: issue #263).
- The developer or reviewer must tag the core development team to notify them of the *backward-incompatible change*, and give at least 2 weeks for objections to be raised before merging to the main branch. If a strong objection is raised the backward-incompatible change should not be merged until the objection is resolved.

Guidance on releasing *backward-incompatible changes*

During the *release* process, the following information must be provided:

- **Release notes:** The *release* notes are already documented in the *ESMValTool Changelog* and ESMValCore Changelog, and "*backward-incompatible changes*" is the first section after "Highlights".
  - **backward-incompatible changes:** This section must include clear instructions detailing how a *recipe developer* should adapt their code for each item in this section, whether the adapted code would introduce a *science change*, and the list of affected or fixed *integrated recipes* that had to be updated due to the *backward-incompatible changes*, if applicable (to provide further examples to *recipe developers* working on *user recipes* of how to adapt code).
  - **Developer guidance:** *Developers of backward-incompatible changes* must:
    - write and include the information required for the "*backward-incompatible changes*" section in the PR that introduces the *backward-incompatible change*
    - share details of the *backward-incompatible change* at the next monthly ESMValTool community meeting
  - **Communication:** The *release* notes must be shared with the community (for example, via the *User mailing list* and the Community repository) at the point the first *release* candidate is made, highlighting the "*backward-incompatible changes*" section. The User Engagement Team should organise the communication of new *releases* together with the *Release manager*.

9.4 Broken/outdated recipe policy

Recipes might stop working for different reasons. Among those are, for instance, withdrawal of datasets used by the recipe (i.e. the recipe contains data that are no longer publicly available), backward incompatible development of the ESMValTool including new features or retiring old ones as well as changes to Python or used dependencies such as library functions. Recipes might also be considered outdated by the *recipe maintainer*.

In case a recipe stopped working, the *recipe maintainer* is contacted by the technical lead development team (@ESMValGroup/technical-lead-development-team) to find a solution, fixing the affected recipe and checking the scientific output after applying the fix. If no recipe maintainer is available, such recipes will be flagged by the release manager during the *Release schedule and procedure* as "broken". For this, the affected recipe will be added to the *list of broken recipes*, together with the version number of the last known release in which the recipe was still working. If a recipe continues to be broken for three releases of the ESMValTool (about one year) and no recipe maintainer could be found during this time, or the affected recipe and diagnostics will be retired. This means the recipe and diagnostic code are moved from the folders `esmvaltool/recipes` and `esmvaltool/diag_scripts` to `archive/legacy_recipes` and `archive/legacy_diag_scripts` by the release manager. In case of recipes considered outdated by the *recipe maintainer*, a pull request can be opened to move recipe and diagnostic code to the `archive` folders at any time.

In both cases, the scientific documentation of the recipe (and diagnostics) will be kept in the user and developer guide with an additional note and link to the last release in which the recipe was still fully functional and a reference to the `archive` folder. The documentation of the retired recipe in the user and developer guide is moved into the category "legacy recipes" to distinguish between currently available and functional recipes and such legacy recipes.

Special care needs to be taken when retiring diagnostic scripts to avoid unintentionally affecting other recipes or diagnostics.

9.5 Making a new dataset

If you are contributing a new dataset, please have a look at *Writing a CMORizer script for an additional dataset* for how to do so. Please always create separate pull requests for CMORizer scripts, even when introducing a new dataset or updating an existing dataset with a new recipe.

If you are updating a CMORizer script to support a different dataset version, please have a look at *Support for multiple versions of a dataset* for how to handle multiple dataset versions.

9.5.1 Dataset documentation

The documentation required for a CMORizer script is the following:

- Make sure that the new dataset is added to the list of *Supported datasets for which a CMORizer script is available* and to the file datasets.yml.
- The code documentation should contain clear instructions on how to obtain the data.
- A BibTeX file named `<dataset>.bibtex` defining the reference for the new dataset should be placed in the directory `esmvaltool/references/`, see *Adding references* for detailed instructions.

For more general information on writing documentation, see *Documentation*.

9.5.2 Testing

When contributing a new script, add an entry for the CMORized data to recipes/examples/recipe_check_obs.yml and run the recipe, to make sure the CMOR checks pass without warnings or errors.

To test a pull request for a new CMORizer script:

1. Download the data following the instructions included in the script and place it in the directory specified by the `--original-data-dir` flag followed by a subdirectory `Tier1`, `Tier2` or `Tier3`, followed by a subdirectory with the same name as the dataset.
2. If available, use the downloading script by running `esmvaltool data download --original-data-dir </path/to/save/data> <dataset>`
3. Run the cmorization by running `esmvaltool data format --original-data-dir </path/to/save/data> <dataset>`
4. Copy the resulting data to the directory specified in your configuration for the `OBS` (for CMIP5 compliant data) or `OBS6` (for CMIP6 compliant data) project.
5. Run `recipes/examples/recipe_check_obs.yml` with the new dataset to check that the data can be used

9.5.3 Scientific sanity check

When contributing a new dataset, we expect that the numbers and units of the dataset look physically meaningful. The scientific reviewer needs to check this.

9.5.4 Data availability

Once your pull request has been approved by the reviewers, ask a member of @OBS-maintainers to add the new dataset to the data pool at DKRZ and CEDA-Jasmin. This team is in charge of merging CMORizer pull requests.

9.5.5 Detailed checklist for reviews

This (non-exhaustive) checklist provides ideas for things to check when reviewing pull requests for new or updated CMORizer scripts.

Dataset description

Check that new dataset has been added to the table of observations defined in the ESMValTool guide user's guide in section *Obtaining input data* (generated from `doc/sphinx/source/input.rst`). Check that the new dataset has also been added to the file datasets.yml.

BibTeX info file

Check that a BibTeX file, i.e. `<dataset>.bibtex` defining the reference for the new dataset has been created in `esmvaltool/references/`.

recipe_check_obs.yml

Check that new dataset has been added to the testing recipe `esmvaltool/recipes/examples/recipe_check_obs.yml`

Downloader script

If present, check that the new downloader script `esmvaltool/cmorizers/data/downloaders/datasets/<dataset>.py` meets standards. This includes the following items:

- Code quality checks
    1. Code quality
    2. No Codacy errors reported

CMORizer script

Check that the new CMORizer script `esmvaltool/cmorizers/data/formatters/datasets/<dataset>.{py,ncl}` meets standards. This includes the following items:

- In-code documentation (header) contains
    1. Download instructions
    2. Reference(s)
- Code quality checks
    1. Code quality (e.g. no hardcoded pathnames)
    2. No Codacy errors reported

Config file

If present, check config file `<dataset>.yml` in `esmvaltool/cmorizers/data/cmor_config/` for correctness. Use `yamllint` to check for syntax errors and common mistakes.

Run downloader script

If available, make sure the downloader script is working by running `esmvaltool data download --config_file <config-file> <dataset>`

Run CMORizer

Make sure CMORizer is working by running `esmvaltool data format --config_file <config-file> <dataset>`

Check output of CMORizer

After successfully running the new CMORizer, check that:

- Output contains (some) valid values (e.g. not only nan or zeros)
- Metadata is defined properly

Run `esmvaltool/recipes/examples/recipe_check_obs.yml` for new dataset.

RAW data

Contact the team in charge of ESMValTool data pool (@OBS-maintainers) and request to copy RAW data to RAWOBS/Tier2 (Tier3).

CMORized data

Contact the team in charge of ESMValTool data pool (@OBS-maintainers) and request to

- Merge the pull request
- Copy CMORized dataset to OBS/Tier2 (Tier3)
- Set file access rights for new dataset

9.6 Support for multiple versions of a dataset

If you plan to update a CMORizer script to support a newer version of an existing dataset, indicate in the issue or pull request if support for previous versions should be kept. If the dataset is used in recipes, please also indicate if the recipes should be updated with the newest dataset version.

9.6.1 Policy for dropping support for older dataset versions

Support for older versions should preferably be kept as long as the data are publicly available. This ensures reproducibility and eases comparison of results of recipes using this dataset.

Even when previous dataset versions are no longer available or data issues have been fixed in a newer dataset version, it is preferable to keep support for the previous version in addition to supporting the newer version. In such cases, it is recommended to ask the recipe maintainers of recipes using the older version of the dataset to update to the newer version if possible so that support for the old version can be dropped in the future.

9.6.2 Naming conventions

If the data structure is rather similar between versions, a single CMORizer script (e.g. woa.py) and config file (e.g. WOA.yml) should be favored to handle multiple versions and avoid code duplication. Version-dependent data fixes can be applied based on the `version` keys defined in the config file.

In some cases, it can be simpler to use different names for different dataset versions (e.g. GCP2018 and GCP2020). CMORizer scripts and config files should be named accordingly.

9.7 Review of pull requests

In the ESMValTool community we use pull request reviews to ensure all code and documentation contributions are of good quality. An introduction to code reviews can be found in The Turing Way, including why code reviews are important and advice on how to have constructive reviews.

Most pull requests will need two reviews before they can be merged. First a technical review takes place and then a scientific review. Once both reviewers have approved a pull request, it can be merged. These three steps are described in more detail below. If a pull request contains only technical changes, e.g. a pull request that corrects some spelling errors in the documentation or a pull request that fixes some installation problem, a scientific review is not needed.

If you are a regular contributor, please try to review a bit more than two other pull requests for every pull request you create yourself, to make sure that each pull request gets the attention it deserves.

If a pull request contains backward-incompatible changes, the developer or reviewer must tag the @ESMValGroup/esmvaltool-coreteam team to notify them of the *backward-incompatible change*.

9.7.1 1. Technical review

Technical reviews are done by the technical review team. This team consists of regular contributors that have a strong interest and experience in software engineering.

Technical reviewers use the technical checklist from the pull request template to make sure the pull request follows the standards we would like to uphold as a community. The technical reviewer also keeps an eye on the design and checks that no major design changes are made without the approval from the technical lead development team. If needed, the technical reviewer can help with programming questions, design questions, and other technical issues.

The technical review team can be contacted by writing @ESMValGroup/tech-reviewers in a comment on an issue or pull request on GitHub.

9.7.2 2. Scientific review

Scientific reviews are done by the scientific review team. This team consists of contributors that have a strong interest and experience in climate science or related domains.

Scientific reviewers use the scientific checklist from the pull request template to make sure the pull request follows the standards we would like to uphold as a community.

The scientific review team can be contacted by writing @ESMValGroup/science-reviewers in a comment on an issue or pull request on GitHub.

9.7.3 3. Merge

Pull requests are merged by the @ESMValGroup/esmvaltool-coreteam. Specifically, pull requests containing a *CMORizer script* can only be merged by a member of @OBS-maintainers, who will then add the CMORized data to the OBS data pool at DKRZ and CEDA-Jasmin. The team member who does the merge first checks that both the technical and scientific reviewer approved the pull request and that the reviews were conducted thoroughly. He or she looks at the list of files that were changed in the pull request and checks that all relevant checkboxes from the checklist in the pull request template have been added and ticked. Finally, he or she checks that the *Pull request checks* passed and merges the pull request. The person doing the merge commit edits the merge commit message so it contains a concise and meaningful text.

Any issues that were solved by the pull request can be closed after merging. It is always a good idea to check with the author of an issue and ask if it is completely solved by the related pull request before closing the issue.

If a pull request contains *backward-incompatible changes*, the person in charge of merging must give the core development team at least 2 weeks for objections to be raised before merging to the main branch. If a strong objection is raised the backward-incompatible change should not be merged until the objection is resolved.

The core development team can be contacted by writing @ESMValGroup/esmvaltool-coreteam in a comment on an issue or pull request on GitHub.

9.7.4 Frequently asked questions

How do I request a review of my pull request?

If you know a suitable reviewer, e.g. because your pull request fixes an issue that they opened or they are otherwise interested in the work you are contributing, you can ask them for a review by clicking the cogwheel next to 'Reviewers' on the pull request 'Conversation' tab and clicking on that person. When changing code, it is a good idea to ask the original authors of that code for a review. An easy way to find out who previously worked on a particular piece of code is to use git blame. GitHub will also suggest reviewers based on who previously worked on the files changed in a pull request. All recipes contain a list of the recipe authors (and some of them in addition a list of recipe maintainers). It is a good idea to ask these people for a review.

If there is no obvious reviewer, you can attract the attention of the relevant team of reviewers by writing to @ESMValGroup/tech-reviewers or @ESMValGroup/science-reviewers in a comment on your pull request. You can

also label your pull request with one of the labels looking for technical reviewer or looking for scientific reviewer, though asking people for a review directly is probably more effective.

How do I optimize for a fast review?

When authoring a pull request, please keep in mind that it is easier and faster to review a pull request that does not contain many changes. Try to add one new feature per pull request and change only a few files. For the ESMValTool repository, try to limit changes to a few hundred lines of code and new diagnostics to not much more than a thousand lines of code. For the ESMValCore repository, a pull request should ideally change no more than about a hundred lines of existing code, though adding more lines for unit tests and documentation is fine.

If you are a regular contributor, make sure you regularly review other people's pull requests, that way they will be more inclined to return the favor by reviewing your pull request.

How do I find a pull request to review?

Please pick pull requests to review yourself based on your interest or expertise. We try to be self organizing, so there is no central authority that will assign you to review anything. People may advertise that they are looking for a reviewer by applying the label looking for technical reviewer or looking for scientific reviewer. If someone knows you have expertise on a certain topic, they might request your review on a pull request though. If your review is requested, please try to respond within a few days if at all possible. If you do not have the time to review the pull request, notify the author and try to find a replacement reviewer.

How do I actually do a review?

To do a review, go to the pull request on GitHub, the list of all pull requests is available here https://github.com/ESMValGroup/ESMValCore/pulls for the ESMValCore and here https://github.com/ESMValGroup/ESMValTool/pulls for the ESMValTool, click the pull request you would like to review.

The top comment should contain (a selection of) the checklist available in the pull request template. If it is not there, copy the relevant items from the pull request template. Which items from the checklist are relevant, depends on which files are changed in the pull request. To see which files have changed, click the tab 'Files changed'. Please make sure you are familiar with all items from the checklist by reading the content linked from *Checklist for pull requests* and check all items that are relevant. Checklists with some of the items to check are available: *recipe and diagnostic checklist* and *dataset checklist*.

In addition to the items from the checklist, good questions to start a review with are 'Do I understand why these changes improve the tool?' (if not, ask the author to improve the documentation contained in the pull request and/or the description of the pull request on GitHub) and 'What could possibly go wrong if I run this code?'.

To comment on specific lines of code or documentation, click the 'plus' icon next to a line of code and write your comment. When you are done reviewing, use the 'Review changes' button in the top right corner to comment on, request changes to, or approve the pull request.

What if the author and reviewer disagree?

When the author and the reviewer of a pull request have difficulty agreeing on what needs to be done before the pull request can be approved, it is usually both more pleasant and more efficient to schedule a meeting or co-working session, for example using Google meet or Jitsi meet.

When reviewing a pull request, try to refrain from making changes to the pull request yourself, unless the author specifically agrees to those changes, as this could potentially be perceived as offensive.

If talking about the pull requests in a meeting still does not resolve the disagreement, ask a member of the @ESMValGroup/esmvaltool-coreteam for their opinion and try to find a solution.

9.8 Maintaining a recipe

As development of the ESMValTool continues, new features may be added, old ones replaced or retired or the interface of library functions may change when updating to new versions. This or for example the withdrawal of datasets used by a recipe can result in an existing recipe to stop working. Such "broken" recipes might require some work to fix such problems and make the recipe fully functional again.

A first **contact point** for the technical lead development team (@ESMValGroup/technical-lead-development-team) in such cases is the recipe "maintainer". The recipe maintainer is then asked to check the affected recipe and if possible, fix the problems or work with the technical lead development team to find a solution. Ideally, a recipe maintainer is able to tell whether the results of a fixed recipe are scientifically valid and look as expected. Being a recipe maintainer consists of the following tasks:

- answering timely to requests from the technical lead development team, e.g. if a recipe is broken
- if needed, checking and trying to fix their recipe(s) / working with the technical lead development team (e.g. fixing a recipe might include updating the actual recipe, diagnostic code or documentation)
- if needed, checking the output of the fixed recipe for scientific validity (asking science lead development team for advice if needed)
- If needed, change the documentation to reflect that some differences from the original results might appear (for reproducibility reasons. e.g. some missing datasets in the fixed recipe produce slight differences in the results but do not modify the conclusions)
- informing the core development team when no longer available as maintainer

Ideally, a recipe maintainer is named when contributing a new recipe to the ESMValTool. Recipe maintainers are asked to inform the core development team (@ESMValGroup/esmvaltool-coreteam) when they are no longer able to act as maintainer or when they would like to step down from this duty for any reason. The core development team will then try to find a successor. If no recipe maintainer can be found, the *policy on unmaintained broken (non-working) recipes* might apply eventually leading to retirement of the affected recipe.

9.9 Upgrading a namelist (recipe) or diagnostic to ESMValTool v2

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 `main`. 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).

9.9.1 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.

9.9.2 Create your own branch

Create your own branch from `main` for each namelist (recipe) to be ported:

```
git checkout main
git pull
git checkout -b <recipe>
```

`main` contains only v2.0 under the `./esmvaltool/` directory.

9.9.3 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.

9.9.4 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.

9.9.5 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 changes 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

continues on next page

Table 1 – continued from previous page

Name in v1.0	Name in v2.0	Affected code
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
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.

9.9.6 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 configuration 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].

9.9.7 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.

9.9.8 Make sure the diagnostic script writes NetCDF output

Each diagnostic script is required to write the output of the analysis 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).

9.9.9 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.

9.9.10 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 statements, the `error_msg` function can be used, which automatically include an exit statement.

9.9.11 Update the documentation

If necessary, add or update the documentation for your recipes in the corresponding 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.

9.9.12 Open a pull request

Create a pull request on github to merge your branch back to `main`, provide a short description of what has been done and nominate one or more reviewers.

9.10 GitHub Workflow

9.10.1 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:

- *MAIN 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>).

9.10.2 Access rights

- Write access to the *MAIN* 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 *MAIN BRANCH* of the PRIVATE repository will be regularly synchronized with the *MAIN 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.

9.10.3 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.

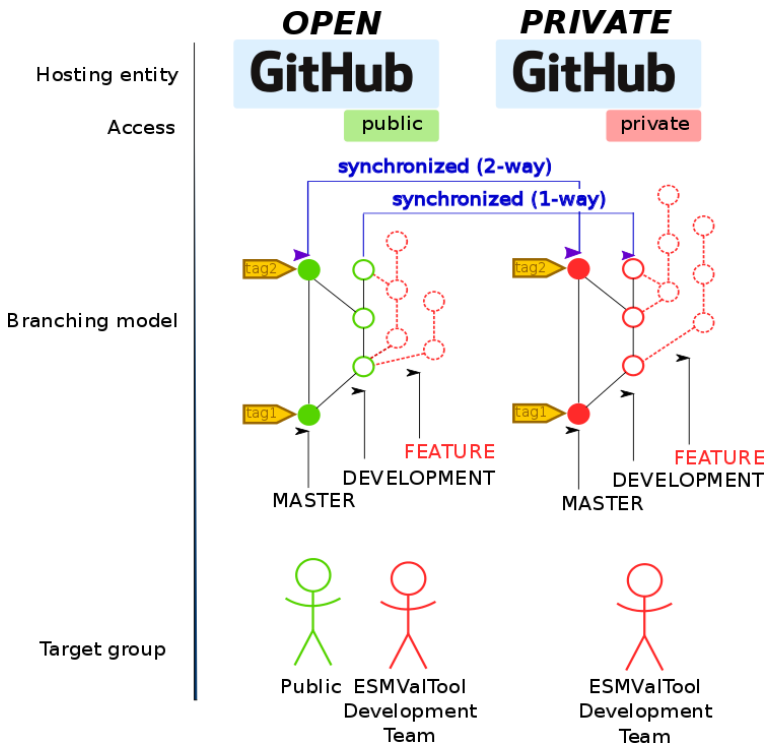


Fig. 1: Schematic diagram of the ESMValTool GitHub repositories.

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

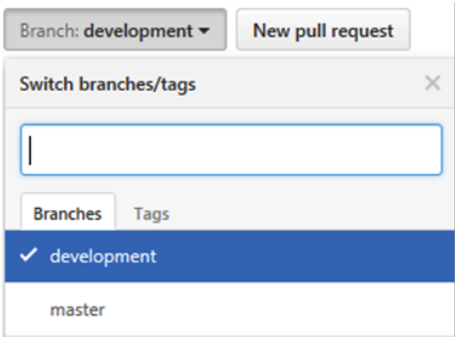
For working on GitHub you need to create an account and login to https://github.com/.

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 main 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/

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 CONTRIBUTING.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.

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.

9.10.4 General do-s and don't-s

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.

Don't-s

- Do not use other programming languages than the ones currently supported (Python, R, NCL). 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*.

9.11 Moving work from the private to the public repository

In case you develop a new diagnostic with the ESMValTool, and you plan on publishing the results of the diagnostic in a peer-reviewed paper, you might want to develop the diagnostic in a slightly less open setting than the ESMValTool-public repository. That is what the ESMValTool-private repository is for. It would be great, though, if you would make the diagnostic available for the whole community after your paper was accepted. The steps that you need to take to develop a diagnostic in the private repository and then open a pull request for it in the public repository are described in the following:

9.11.1 1. Clone the private repository

For example, to clone a repository called esmvaltool-private, you would run:

```
git clone git@github.com:esmvalgroup/esmvaltool-private
```

or

```
git clone https://github.com/esmvalgroup/esmvaltool-private
```

9.11.2 2. Make a branch to develop your recipe and diagnostic

```
git checkout main
```

```
git pull
```

```
git checkout -b my-awesome-diagnostic
```

9.11.3 3. Develop your diagnostic in that branch and push it to the private repository

```
git push -u origin my-awesome-diagnostic
```

the first time and

```
git push
```

any other time

9.11.4 4. Write and submit your paper

9.11.5 5. Push your branch to the public repository

Add the public repository as a remote

```
git remote add public git@github.com:esmvalgroup/esmvaltool
```

or

```
git remote add public https://github.com/esmvalgroup/esmvaltool
```

and push your branch to the public repository

```
git push -u public my-awesome-diagnostic
```

9.11.6 6. Make a pull request in the public repository

Go to https://github.com/ESMValGroup/ESMValCore/pulls and click the 'New pull request button'. Process reviewer comments and get it merged as described in *Review of pull requests*.

9.11.7 7. Obtain a DOI for your code and add it to your paper

Wait for a new release of ESMValTool. Releases are scheduled normally every four months. Find the release schedule here: *Release schedule*. With the next release, your diagnostic recipe and source code will automatically be included in the archive on Zenodo and you can add the DOI from Zenodo to your paper: https://zenodo.org/record/3698045

9.12 Release strategy and procedure

These pages detail the release strategy and technical procedures during a release; these pages are very helpful if you are the next Release Manager.

For the release schdule, check the *Release schedule* to see upcoming and past releases.

9.12.1 Release schedule and procedure for ESMValCore and ESMValTool

This document describes the process for the release of ESMValCore and ESMValTool. By following a defined process, we streamline the work, reduce uncertainty about required actions, and clarify the state of the code for the user.

ESMValTool follows a strategy of timed releases. That means that we do releases with a regular frequency and all features that are implemented up to a certain cut-off-point can go into the upcoming release; those that are not are deferred to the next release. This means that generally no release will be delayed due to a pending feature. Instead, the regular nature of the release guarantees that every feature can be released in a timely manner even if a specific target release is missed.

Because of limited resources, only the latest released versions of ESMValTool and ESMValCore is maintained. If your project requires longer maintenance or you have other concerns about the release strategy, please contact the ESMValTool core development team, see *Support*.

Overall Procedure

Timeline

1. Contributors assign issues (and pull requests) that they intend to finish before the due date, there is a separate milestone for ESMValCore and ESMValTool
2. The ESMValCore feature freeze takes place on the ESMValCore due date
3. Some additional testing of ESMValCore takes place

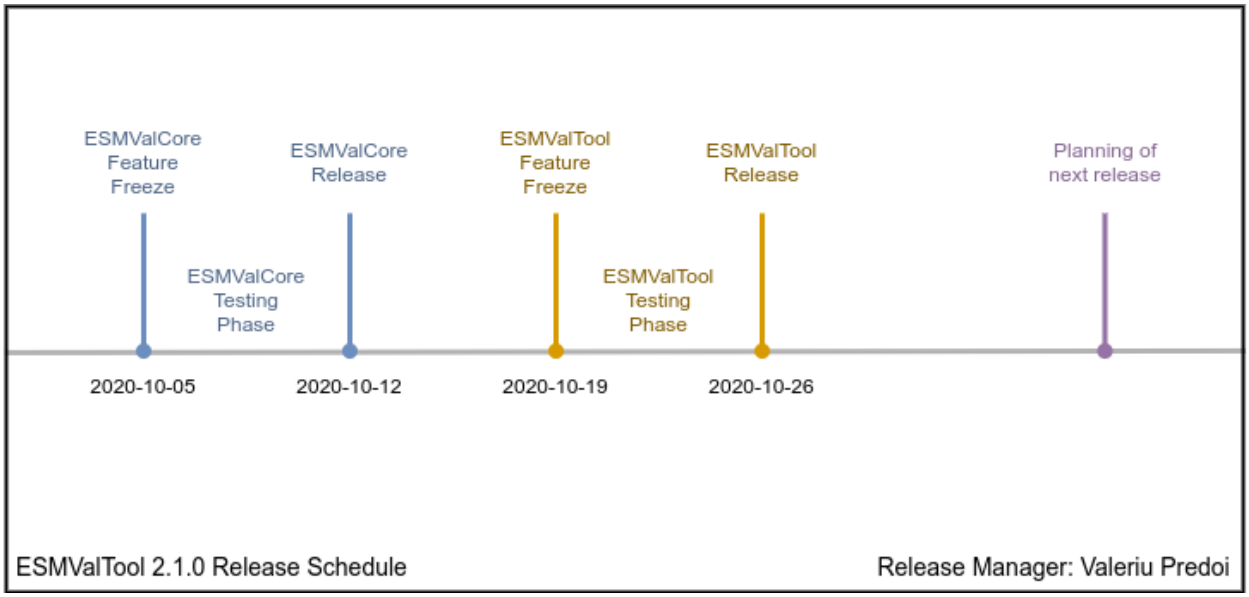


Fig. 2: Example of a Release Timeline (in this case for 2.1.0)

4. ESMValCore release
5. The ESMValTool feature freeze takes place
6. Some additional testing of ESMValTool takes place
7. ESMValTool release
8. Soon after the release, the core development team meets to coordinate the content of the milestone for the next release

Release schedule

With the following release schedule, we strive to have three releases per year and to avoid releases too close to holidays, as well as avoiding weekends.

Upcoming releases

Date	Release manager	Version
2026 Q1	DLR	2.14.0
2026 Q2	ACCESS-NRI	2.15.0
2026 Q4	Met Office	2.16.0
2027 Q1	NLeSC	2.17.0
2027 Q2	NERC (NCAS)	2.18.0
2027 Q4	BSC	2.19.0

Past releases

- 2.13.0 (Release Manager: Julien Lenhardt)

Planned	Done	Event	Changelog
2025-08-18		ESMValCore *Feature Freeze*	
2025-08-29	2025-10-16	ESMValCore Release 2.13.0	v2.13.0
2025-09-01		ESMValTool *Feature Freeze*	
2025-09-12	2025-10-20	ESMValTool Release 2.13.0	*v2.13.0*

- 2.12.0 (Release Manager: Saskia Loosveldt Tomas)

Planned	Done	Event	Changelog
2025-01-13		ESMValCore *Feature Freeze*	
2025-01-20	2025-02-27	ESMValCore v2.12.0 released	v2.12.0
2025-01-27		ESMValTool *Feature Freeze*	
2025-02-03	2025-03-05	ESMValTool v2.12.0 released	*v2.12.0*

- 2.11.0 (Release Manager: Met Office: Emma Hogan, Chris Billows, Ed Gillett)

Planned	Done	Event	Changelog
2024-04-22		ESMValCore *Feature Freeze*	
2023-05-03	2024-07-03	ESMValCore v2.11.0 released	v2.11.0
2023-05-06		ESMValTool *Feature Freeze*	
2023-05-17	2024-07-04	ESMValTool v2.11.0 released	*v2.11.0*

- 2.10.0 (Release Manager: Klaus Zimmermann)

Planned	Done	Event	Changelog
2023-10-02		ESMValCore *Feature Freeze*	
2023-10-09	2023-12-19	ESMValCore v2.10.0 released	v2.10.0
2023-10-16		ESMValTool *Feature Freeze*	
2023-10-16	2023-12-20	ESMValTool v2.10.0 released	*v2.10.0*

- 2.9.0 (Release Manager: Bouwe Andela)

Planned	Done	Event	Changelog
2023-06-05		ESMValCore *Feature Freeze*	
2023-06-12	2023-07-04	ESMValCore v2.9.0 released	v2.9.0
2023-06-19		ESMValTool *Feature Freeze*	
2023-06-26	2023-07-06	ESMValTool v2.9.0 released	*v2.9.0*

- 2.8.1 (Bugfix, Release Manager: Valeriu Predoi)

Done	Event	Changelog
2023-06-02	ESMValCore Release 2.8.1	v2.8.1

- 2.8.0 (Release Manager: Rémi Kazeroni)

Planned	Done	Event	Changelog
2023-03-03		ESMValCore Feature Freeze	
2023-03-20	2023-03-23	ESMValCore Release 2.8.0	v2.8.0
2023-03-17		ESMValTool Feature Freeze	
2023-03-27	2023-03-28	ESMValTool Release 2.8.0	*v2.8.0*

- 2.7.1 (Bugfix, Release Manager: Valeriu Predoi)

Done	Event	Changelog
2022-12-12	ESMValCore Release 2.7.1	v2.7.1

- 2.7.0 (Release Manager: Valeriu Predoi)

Planned	Done	Event	Changelog
2022-10-03		ESMValCore Feature Freeze	
2022-10-10	2022-10-13	ESMValCore Release 2.7.0	v2.7.0
2022-10-17		ESMValTool Feature Freeze	
2022-10-24	2022-10-28	ESMValTool Release 2.7.0	*v2.7.0*

- 2.6.0 (Release Manager: Saskia Loosveldt Tomas)

Planned	Done	Event	Changelog
2022-06-06		ESMValCore Feature Freeze	
2022-06-13	2022-07-15	ESMValCore Release 2.6.0	v2.6.0
2022-06-20		ESMValTool Feature Freeze	
2022-06-27	2022-07-25	ESMValTool Release 2.6.0	*v2.6.0*

- 2.5.0 (Coordinating Release Manager: Axel Lauer, team members: Manuel Schlund, Rémi Kazeroni)

Planned	Done	Event	Changelog
2022-02-07		ESMValCore Feature Freeze	
2022-02-14	2022-03-14	ESMValCore Release 2.5.0	v2.5.0
2022-02-21		ESMValTool Feature Freeze	
2022-02-28	2022-03-15	ESMValTool Release 2.5.0	*v2.5.0*

- 2.4.0 (Release Manager: Klaus Zimmermann)

Planned	Done	Event	Changelog
2021-10-04		ESMValCore Feature Freeze	
2021-10-11	2021-11-08	ESMValCore Release 2.4.0	v2.4.0
2021-10-18		ESMValTool Feature Freeze	
2021-10-25	2021-11-09	ESMValTool Release 2.4.0	*v2.4.0*

- 2.3.1 (Bugfix, Release Manager: Klaus Zimmermann)

Done	Event	Changelog
2021-07-23	ESMValCore Release 2.3.1	v2.3.1

- 2.3.0 (Release Manager: Klaus Zimmermann)

Planned	Done	Event	Changelog
2021-06-07		ESMValCore Feature Freeze	
2021-06-14	2021-06-14	ESMValCore Release 2.3.0	v2.3.0
2021-06-21		ESMValTool Feature Freeze	
2021-06-28	2021-07-27	ESMValTool Release 2.3.0	*v2.3.0*

- 2.2.0 (Release Manager: Javier Vegas-Regidor)

Planned	Done	Event	Changelog
2021-02-01		ESMValCore Feature Freeze	
2021-02-07	2021-02-09	ESMValCore Release 2.2.0	v2.2.0
2021-02-14		ESMValTool Feature Freeze	
2021-02-21	2021-02-25	ESMValTool Release 2.2.0	*v2.2.0*

- 2.1.1 (Bugfix, Release Manager: Valeriu Predoi)

Done	Event	Changelog
2020-12-01	ESMValTool Release 2.1.1	*v2.1.1*

- 2.1.0 (Release Manager: Valeriu Predoi)

Planned	Done	Event	Changelog
2020-10-05		ESMValCore Feature Freeze	
2020-10-12	2020-10-12	ESMValCore Release 2.1.0	v2.1.0
2020-10-19		ESMValTool Feature Freeze	
2020-10-26	2020-10-26	ESMValTool Release 2.1.0	*v2.1.0*

- 2.0.0 (Release Manager: Bouwe Andela)

Planned	Done	Event	Changelog
2020-07-01		ESMValCore Feature Freeze	
2020-07-20	2020-07-20	ESMValCore Release 2.0.0	v2.0.0
2020-07-22		ESMValTool Feature Freeze	
2020-08-03	2020-08-03	ESMValTool Release 2.0.0	*v2.0.0*

Detailed timeline steps

These are the detailed steps to take to make a release.

1. Populate the milestone
   - The core development team will make sure it adds issues that it intends to work on as early as possible.
   - Any contributor is welcome to add issues or pull requests that they intend to work on themselves to a milestone.
2. ESMValCore feature freeze, testing, and release candidates
   - A release branch is created and branch protection rules are set up so only the release manager (i.e. the person in charge of the release branch) can push commits to that branch.

- Make a release candidate with the release branch following the ESMValCore release instructions.
- Uncomment the release candidate channel item (i.e. `conda-forge/label/esmvalcore_rc`) in the `pyproject.toml` of ESMValTool to add it to the list of channels used. Adjust the pin on ESMValCore after each release candidate (e.g. `esmvalcore==2.8.0rc1`). Check that the environment creation of ESMValTool works fine and contains the latest release candidate version.
- Run all the recipes (optionally with a reduced amount of data) to check that they still work with the release candidate.
- If a bug is discovered that needs to be fixed before the release, a pull request can be made to the main branch to fix the bug. The person making the pull request can then ask the release manager to cherry-pick that commit into the release branch.
- Make another release candidate including the bugfix(es) and run the affected recipes again to check for further bugs.
- Make as many release candidates for ESMValCore as needed in order to fix all the detected bugs.

3. ESMValTool feature freeze
   - A release branch is created and branch protection rules are set up so only the release manager (i.e. the person in charge of the release branch) can push commits to that branch.
   - The creation of the release branch is announced to the ESMValTool development team along with the procedures to use the branch for testing and making last-minute changes (see next step).
4. Some additional testing of ESMValTool
   - *Run all the recipes to check that they still work and generate the overview HTML pages*.
   - Upload the results to the webpage at https://esmvaltool.dkrz.de/shared/esmvaltool/.
   - *Compare the results to those obtained with the previous release*.
   - Create a GitHub discussion to communicate about the results.
   - If there are differences with the previous release, ask recipe maintainers or authors to review the plots and NetCDF files of their diagnostics, for example by mentioning them in the discussion.
   - If a bug is discovered that needs to be fixed before the release, a pull request can be made to the main branch to fix the bug. The person making the pull request can then ask the release manager to cherry-pick that commit into the release branch.
   - Update the *list of broken recipes* with new recipes that could not be run successfully during the testing. Open a separate GitHub issue for each failing recipe and assign the next milestone. Open an overview issue, see Issue #3484 for an example, and review past overview issues. Take action to ensure that the broken recipe policy is followed.
5. ESMValCore release
   - Make the official ESMValCore release with the last release candidate by following the ESMValCore release instructions.
6. ESMValTool release
   - Pin ESMValCore to the same version as ESMValTool in the `pyproject.toml` and on conda-forge. This way, we make sure that ESMValTool uses the ESMValCore version with which it has been tested. Make sure to comment again the release candidate channel once ESMValCore has been released.
   - Make the release by following *How to make an ESMValTool release*.
7. Announce the releases

- Ask the user engagement team to announce the releases to the user mailing list, the development team mailing list, and on twitter.

8. Core development team meets to coordinate the content of next milestone
    - Create a doodle for the meeting or even better, have the meeting during an ESMValTool workshop
    - Prepare the meeting by filling the milestone
    - At the meeting, discuss
        - If the proposed issues cover everything we would like to accomplish
        - Are there things we need to change about the release process
        - Who will be the release manager(s) for the next release

Bugfix releases

Next to the feature releases described above, it is also possible to have bugfix releases (2.0.1, 2.0.2, etc). In general bugfix releases will only be done on the latest release, and may include ESMValCore, ESMValTool, or both.

Procedure

1. One or more issues are resolved that are deemed (by the core development team) to warrant a bugfix release.
2. A release branch is created from the last release tag and the commit that fixes the bug/commits that fix the bugs are cherry-picked into it from the main branch.
3. Some additional testing of the release branch takes place.
4. The release takes place.

Compatibility between ESMValTool and ESMValCore is ensured by the appropriate version pinning of ESMValCore by ESMValTool.

Glossary

Feature freeze

The date on which no new features may be submitted for the upcoming release. After this date, only critical bug fixes can still be included to the *Release branch*. Development work can continue in the main branch. If you are unsure whether new developments could interfere with the release, check with the *Release manager*.

Milestone

A milestone is a list of issues and pull-request on GitHub. It has a due date, this date is the date of the feature freeze. Adding an issue or pull request indicates the intent to finish the work on this issue before the due date of the milestone. If the due date is missed, the issue can be included in the next milestone.

Release manager

The person in charge of making the release, both technically and organizationally. Appointed for a single release. Check the *Release schedule* to see who is the manager of the next release.

Release branch

The release branch can be used to do some additional testing before the release, while normal development work continues in the main branch. It will be branched off from the main branch after the feature freeze and will be used to make the release on the release date. The only way to still get something included in the release after the feature freeze is to ask the release manager to cherry-pick a commit from the main branch into this branch.

How to make an ESMValTool release

Before the actual release, a number of tests, and pre-release steps must be performed, a detailed workflow description can be found here *Release: recipes runs and comparison*.

The release manager makes the release, assisted by the release manager of the previous release, or if that person is not available, another previous release manager. Perform the steps listed below with two persons, to reduce the risk of error.

> **Note**
>
> The previous release manager ensures the current release manager has the required administrative permissions to make the release. Consider the following services: conda-forge, DockerHub, PyPI, and readthedocs.

The release of ESMValTool should come after the release of ESMValCore. To make a new release of the package, follow these steps:

1. Check that all tests and builds work

- Check that the `nightly` test run on CircleCI was successful.
- Check that the GitHub Actions test runs were successful.
- Check that the documentation builds successfully on readthedocs.
- Check that the Docker images are building successfully.

All tests should pass before making a release (branch).

2. Increase the version number

The version number is automatically generated from the information provided by git using setuptools-scm, but a static version number is stored in `CITATION.cff`. Make sure to update the version number and release date in `CITATION.cff`. See https://semver.org for more information on choosing a version number. Make sure that the ESMValCore version that is being used is set to the latest version. See the *dependencies* section in order to find more details on how update the ESMValCore version. Make a pull request and get it merged into `main`.

3. Add release notes

Use the script *draft_release_notes.py* to create a draft of the release notes. This script uses the titles and labels of merged pull requests since the previous release. Open a discussion to allow members of the development team to nominate pull requests as highlights. Add the most voted pull requests as highlights at the beginning of changelog. After the highlights section, list any backward incompatible changes that the release may include. The *backward compatibility policy* lists the information that should be provided by the developer of any backward incompatible change. Make sure to also list any deprecations that the release may include, as well as a brief description on how to upgrade a deprecated feature. Review the results, and if anything needs changing, change it on GitHub and re-run the script until the changelog looks acceptable. Copy the result to the file `doc/sphinx/source/changelog.rst`. If possible, try to set the script dates to the date of the release you are managing. Make a pull request and get it merged into `main`.

4. Create a release branch

Create a branch off the `main` branch and push it to GitHub. The name of the release branch should be of the form `vX.Y.x`, where `X.Y` is the major and minor version number of the release, e.g. `v2.1.x` for the releases `v2.1.0`, `v2.1.1`, etc. Ask someone with administrative permissions to set up branch protection rules for it so only you and the person helping you with the release can push to it. Announce the name of the branch in an issue and ask the members of the ESMValTool development team to run their favourite recipe using this branch.

5. Make the release on GitHub

Do a final check that all tests on CircleCI and GitHub Actions completed successfully. Then click the releases tab and create the new release from the release branch (i.e. not from `main`). The release tag always starts with the letter `v` followed by the version number, e.g. `v2.1.0`.

6. Merge the release branch back into the main branch

When the (pre-)release is tagged, it is time to merge the release branch back into *main*. We do this for two reasons, namely, one, to mark the point up to which commits in *main* have been considered for inclusion into the present release, and, two, to inform setuptools-scm about the version number so that it creates the correct version number in *main*. However, unlike in a normal merge, we do not want to integrate any of the changes from the release branch into main. This is because all changes that should be in both branches, i.e. bug fixes, originate from *main* anyway and the only other changes in the release branch relate to the release itself. To take this into account, we perform the merge in this case on the command line using the ours merge strategy (`git merge -s ours`), not to be confused with the `ours` option to the ort merge strategy (`git merge -X ours`). For details about merge strategies, see the above-linked page. To execute the merge use following sequence of steps

```
git fetch
git checkout main
git pull
git merge -s ours v2.1.x
git push
```

Note that the release branch remains intact and you should continue any work on the release on that branch.

7. Create and upload the PyPI package

The package is automatically uploaded to the PyPI by a GitHub action. Note that for security reasons, the upload needs to be approved by someone from the ESMValGroup/technical-lead-development-team who is not the author of the release.

If the automatic build and upload has failed for some reason, do it manually by following these instructions:

- Check out the tag corresponding to the release, e.g. `git checkout tags/v2.1.0`
- Make sure your current working directory is clean by checking the output of `git status` and by running `git clean -xdf` to remove any files ignored by git.
- Install the required packages: `python3 -m pip install --upgrade build twine`
- Build the package: `python3 -m build` This command should generate two files in the `dist` directory, e.g. `ESMValTool-2.1.0-py3-none-any.whl` and `ESMValTool-2.1.0.tar.gz`.
- Upload the package: `python3 -m twine upload dist/*` You will be prompted for an API token if you have not set this up before, see here for more information.

You can read more about this in Packaging Python Projects.

8. Create the Conda package

The `esmvaltool` package is published on the conda-forge conda channel. This is done via a pull request on the esmvaltool-suite-feedstock repository.

After the upload of the PyPI package, this pull request is automatically opened by a bot. An example pull request can be found here. Follow the instructions by the bot to finalize the pull request. This step mostly contains updating dependencies that have been changed during the last release cycle. Once approved by the feedstock maintainers they will merge the pull request, which will in turn publish the package on conda-forge some time later. Contact the feedstock maintainers if you want to become a maintainer yourself.

9. Check the Docker images

There are three main Docker container images available for ESMValTool on Dockerhub:

- `esmvalgroup/esmvaltool:stable`, built from docker/Dockerfile, this is a tag that is always the same as the latest released version. This image is only built by Dockerhub when a new release is created.
- `esmvalgroup/esmvaltool:development`, built from docker/Dockerfile.dev, this is a tag that always points to the latest development version of ESMValTool. This image is built by Dockerhub every time there is a new commit to the `main` branch on Github.
- `esmvalgroup/esmvaltool:experimental`, built from docker/Dockerfile.exp, this is a tag that always points to the latest development version of ESMValTool with the latest development version of ESMValCore. Note that some recipes may not work as expected with this image because the ESMValTool development version has been designed to work with the latest release of ESMValCore (i.e. not with the development version). This image is built by Dockerhub every time there is a new commit to the ESMValTool `main` branch on Github.

In addition to the three images mentioned above, there is an image available for every release (e.g. `esmvalgroup/esmvaltool:v2.5.0`). When working on the Docker images, always try to follow the best practices.

After making the release, check that the Docker image for that release has been built correctly by

1. checking that the version tag is available on Dockerhub and the `stable` tag has been updated,
2. running some recipes with the `stable` tag Docker container, for example one recipe for Python, NCL, and R,
3. running a recipe with a Singularity container built from the `stable` tag.

If there is a problem with the automatically built container image, you can fix the problem and build a new image locally. For example, to build and upload the container image for v2.5.0 of the tool run:

```
git checkout v2.5.0
git clean -x
docker build -t esmvalgroup/esmvaltool:v2.5.0 . -f docker/Dockerfile
docker push esmvalgroup/esmvaltool:v2.5.0
```

and if it is the latest release that you are updating, also run

```
docker tag esmvalgroup/esmvaltool:v2.5.0 esmvalgroup/esmvaltool:stable
docker push esmvalgroup/esmvaltool:stable
```

Note that the `docker push` command will overwrite the existing tags on Dockerhub.

If you would like to make a small change to an existing Docker container image, it is also possible to do just that using the docker commit command. Note that this is only recommended for very small changes, as it is not reproducible and it will add an extra layer, increasing the size of the image. To do this, start the container with `docker run -it --entrypoint /bin/bash esmvalgroup/esmvaltool:v2.5.0` and make your changes. Exit the container by pressing *ctrl+d* and find it back by running `docker ps -a`. Find the *CONTAINER ID* of the image you would like to save and run `docker commit -c 'ENTRYPOINT ["conda", "run", "--name", "esmvaltool", "esmvaltool"]' 633696a8b53a esmvalgroup/esmvaltool:v2.5.0` where `633696a8b53c` is the an example of a container ID, replace it by by the actual ID.

Changelog

- 2020-09-09 Converted to rst and added to repository (future changes tracked by git)
- 2020-09-03 Update during video conference (present: Bouwe Andela, Niels Drost, Javier Vegas, Valeriu Predoi, Klaus Zimmermann)
- 2020-07-27 Update including tidying up and Glossary by Klaus Zimmermann and Bouwe Andela

- 2020-07-23 Update to timeline format by Bouwe Andela and Klaus Zimmermann
- 2020-06-08 First draft by Klaus Zimmermann and Bouwe Andela

9.12.2 Release: recipes runs and comparison

The release procedure for ESMValTool is a fairly involved process (at the moment), so it is important to be very well organized and to have documented each procedural steps, so that the next release manager can follow said steps, and finalize the release without any delays.

The workflow below assumes an ESMValCore release candidate, or a completed stable release, have been released and deployed on conda-forge and PyPI; it also assumes the release manager has access to accounts on DKRZ/Levante.

Below is a list of steps that the release manager, together with the previous release manager, should go through before the actual release; these include testing the new code by running all available recipes in the `main` branch, and comparing the output against the previous release.

Open an issue on GitHub

First, open an issue on GitHub where the testing workflow before the release is documented (see example https://github.com/ESMValGroup/ESMValTool/issues/2881). Name it something relevant like "Recipe testing and comparison for release 2.x.x", and populate the issue description with information about where the testing is taking place, what tools are used, and what versions, here are some suggestions:

- path to the output directories on DKRZ/Levante

We should document various utilities' versions so that the work can be reproduced in case there is an issue, or release work needs to be picked up mid-release by another release manager:

- documenting *conda*/*mamba* versions:

```
mamba --version
```

- documenting *git* branch and its state:

```
git status
```

Furthermore, the runtime environment needs to be documented: make a copy of the environment file, and attach it in the release testing issue; to record the environment in a yaml file use e.g.

```
conda env export > ToolEnv_v2xx_Test.txt
```

Modifications to configuration files need to be documented as well. To test recipes, it is recommended to only use the default options and DKRZ data directories, simply by uncommenting the DKRZ-Levante block of a newly generated configuration file.

Submit run scripts - test recipe runs

We are now ready to start running all the available recipes, to compare output against previous release. Running is currently done via batch scripts submitted to a schedulers (SLURM). Generate the submission scripts using the `generate.py` *utility Python script*.

You will have to set the name of your environment, your email address (if you want to get email notifications for successful/failed jobs) and the name of the directory you want to store the log files of the jobs. A compute project from which resources are billed needs to be set, and the default partition is set to *interactive*. More information on running jobs with SLURM on DKRZ/Levante can be found in the DKRZ documentation.

You can also specify the path to your configuration directory where `max_parallel_tasks` can be set in a YAML file. The script was found to work well with `max_parallel_tasks=8`. Some recipes need to be run with

`max_parallel_tasks=1` (large memory requirements, CMIP3 data, diagnostic issues, ...). These recipes are listed in *ONE_TASK_RECIPES*.

Some recipes need other job requirements, you can add their headers in the *SPECIAL_RECIPES* dictionary. Otherwise the header will be written following the template that is written in the lines below. If you want to exclude recipes, you can do so by uncommenting the *exclude* lines.

Before submitting all jobs, it is recommended to try the batch script generation with `submit = False` and check the generated files. If recipes with special runtime requirements have been added to ESMValTool since the previous release, these may need to be added to *SPECIAL_RECIPES* and/or to *ONE_TASK_RECIPES*. Other recipes should run successfully with the default SLURM settings set in this script.

The launch scripts will be saved in the same directory you execute the script from. These are named like `launch_recipe_<name>.sh`. To submit these scripts to the SLURM scheduler, use the `sbatch launch_recipe_<name>.sh` command. You can check the status of your BATCH queue by invoking:

```
squeue -u $USER
```

Also, for computationally-heavy recipes, you can require more memory and/or time, see e.g. edited batch header below (note the *compute* partition which is used for such heavy runs):

```
#SBATCH --partition=compute
#SBATCH --time=08:00:00
#SBATCH --mem=0
#SBATCH --constraint=512G
```

Note

On DKRZ/Levante, a user can't have more than 20 SLURM jobs running at a time. As soon as a job is finished, the next one should start. More information on the job handling at DKRZ here. Also note that the `--mem=0` argument needs be specified if any of the `--constraint` arguments are used for memory requests, so that the node's full memory is allocated.

Analyse the results

Once all jobs are completed, assemble some statistics so that issues with certain recipes can be followed-up, and document this information in the release issue, such as:

- number of successfully run recipes
- number of failed recipes with preprocessor errors (can they be fixed? Can the fixes be included in the release?)
- number of failed recipes with diagnostic errors (can they be fixed? Can the fixes be included in the release?)
- number of recipes that are missing data
- number of recipes that have various other issues (and document them)

To parse the output of all these runs, use the `parse_recipes_output.py` *utility Python script*. It is recommended to run the recipes with *log_level: info* in your config file to enable the parsing script to run fast.

Share the results with the community

Create the debug.html and index.html overview webpages by running the *utility script* in the directory containing the recipe runs. These two files, together with the recipe output, need to be copied to the disk of a virtual machine (VM) used to display recipe output in webpages. Do not store final release results on the VM including */preproc/* dirs, the total size for all the recipes output, including */preproc/* dirs is in the 4.5TB ballpark, much too high for the VM storage capacity! Therefore, we would recommend using the option to remove preprocessing directories upon recipe running

successfully `--remove-preproc-dir=True` at runtime, or set `remove_preproc_dir: true` in the configuration file.

Login and access to the DKRZ esmvaltool VM - results from recipe runs are stored on the VM; log in to the Levante head node and then continue to the VM with:

```
ssh user@esmvaltool.dkrz.de
```

where *user* is your DKRZ/Levante user name. Then create a new subdirectory in `/shared/esmvaltool/` that will contain recipe output. This should be named like the ESMValCore version used for the testing, e.g. `v2.x.xrcx` or `v2.x.x`. Recipe output can be copied by doing from the VM:

```
nohup rsync --exclude preproc/ -rlt /path_to_testing/esmvaltool_output/* /shared/
↪esmvaltool/v2.x.x/
```

By copying the debug.html and index.html files into /shared/esmvaltool/v2.x.x/, the output becomes available online, see for example. Before copying the recipe output to the VM, you may want to clean up your directory containing the results by removing any large `preproc` directories of failed runs and only keeping the last run for each recipe. This will help generating a clearer overview webpage. Note that the `summarize.py` script needs to be rerun if recipe runs were added or deleted from your testing directory.

Link the overview webpage to the release issue. This makes it much easier to ask for feedback from recipe developers and analyse failures.

Results produced with the final ESMValCore release candidate should be put in a VM directory named after the version number, e.g. `v2.x.x`. Once the release process is over, test results produced with previous release candidates can be deleted to save space on the VM.

Note

If you wrote recipe runs output to Levante's */scratch* partition, be aware that the data will be removed after two weeks, so you will have to quickly move the output data to the VM, using the `nohup` command above.

Running the comparison

To compare the newly produced output from running all recipes, follow these steps below.

Access to the DKRZ esmvaltool VM, then install miniconda on the VM, and if you have a Miniconda installer already downloaded in your Levante $HOME

```
scp Miniconda3-py39_4.12.0-Linux-x86_64.sh user@esmvaltool.dkrz.de:/mnt/esmvaltool_disk2/
↪work/<username>
```

Warning

conda environments should not be created in the home directory because it is on a very small disk, but rather in a directory with your username under */mnt/esmvaltool_disk2/work/<username>*

Next, we need to set up the input files

The `/work` partition is visible from the VM so you can run the compare tool straight on the VM.

The steps to running the compare tool on the VM are the following:

- run date: log the run date here

- conda env: log the name of the conda environment you are using
- ESMValTool branch: log the name of the code branch you are using (e.g. *v2.8.x*)
- prerequisite - install *imagehash*: *pip install imagehash*
- reference run (v2.7.0; previous stable release): *export reference_dir=/work/bd0854/b382109/v270* (contains *preproc/* dirs too, 122 recipes)
- current run (v2.8.0): *export current_dir=path_to_current_run*
- run the *comparison script* with:

```
nohup python ESMValTool/esmvaltool/utils/develop/compare.py --reference $reference_dir --
↪current $current_dir > compare_v280_output.txt
```

Copy the comparison txt file to the release issue. Some of the recipes will appear as having identical output to the one from previous release. However, others will need human inspection. Ask the recipe maintainers (@ESMValGroup/esmvaltool-recipe-maintainers) and ESMValTool Development Team (@ESMValGroup/esmvaltool-developmentteam) to provide assistance in checking the results. Here are some guidelines on how to perform the human inspection:

- look at plots from current run vs previous release run: most of them will be identical, but if Matplotlib has changed some plotting feature, images may look slightly different so the comparison script may report them if the difference is larger than the threshold - but Mark I eyeball inspection will show they are identical
- other plots will differ due to changes in plot settings (different colours, axes etc) due to updated settings from the diagnostic developers: if they look similar enough, then it's fine
- report (and subsequently open issues) if you notice major differences in plots; most times a simple comment on the release issue, whereby you tag the diagnostic developers leads to them having a look at the plots and OK-ing them; if that's not the case, then open a separate issue. You can example of release issues containing overview lists and tables of failures and problems in 2881 and 3076.

Appendix

Here you can find a list of utility scripts used to run recipes and analyse the results:

- *Python scripts* that create slurm submission scripts and parse slurm log files.
- *Python script* that compares one or more recipe runs to known good previous run(s).
- *Python script* that creates the `index.html` and `debug.html` overview pages.

CHAPTER

TEN

UTILITIES

This section provides information on tools that are useful when developing ESMValTool. Tools that are specific to ESMValTool live in the esmvaltool/utils directory, while others can be installed using the usual package managers.

10.1 Pre-commit

pre-commit is a handy tool that can run many tools for checking code quality with a single command. Usually it is used just before committing, to avoid accidentally committing mistakes. It knows knows which tool to run for each filetype, and therefore provides a convenient way to check your code!

To run `pre-commit` on your code, go to the ESMValTool directory (`cd ESMValTool`) and run

```
pre-commit run
```

By default, pre-commit will only run on the files that have been changed, meaning those that have been staged in git (i.e. after `git add your_script.py`).

To make it only check some specific files, use

```
pre-commit run --files your_script.py
```

or

```
pre-commit run --files your_script.R
```

Alternatively, you can configure `pre-commit` to run on the staged files before every commit (i.e. `git commit`), by installing it as a git hook using

```
pre-commit install
```

Pre-commit hooks are used to inspect the code that is about to be committed. The commit will be aborted if files are changed or if any issues are found that cannot be fixed automatically. Some issues cannot be fixed (easily), so to bypass the check, run

```
git commit --no-verify
```

or

```
git commit -n
```

or uninstall the pre-commit hook

```
pre-commit uninstall
```

Note that the configuration of pre-commit lives in .pre-commit-config.yaml.

10.2 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
```

10.3 Colormap samples

Tool to generate colormap samples for ESMValTool's default Python and NCL colormaps.

Run

```
esmvaltool colortables python
```

or

```
esmvaltool colortables ncl
```

to generate the samples.

10.4 Running multiple recipes

It is possible to run more than one recipe in one go.

This can for example be achieved by using `rose` and/or `cylc`, tools that may be available at your local HPC cluster.

In the case in which neither `rose` nor `cylc` are available at your HPC cluster, it is possible to automatically generate job submission scripts, as well as a summary of the job outputs using the scripts available in esmvaltool/utils/batch-jobs.

10.4.1 Using cylc

A cylc suite for running all recipes is available in esmvaltool/utils/testing/regression. This suite is configured to work with versions of cylc older than 8.0.0 .

To prepare for using this tool:

1. Log in to a system that uses slurm
2. Make sure the required CMIP and observational datasets are available and their `rootpath` and `drs` is properly set up in the configuration
3. Make sure the required auxiliary data is available (see *recipe documentation*)
4. Install ESMValTool

Next, get started with cylc:

1. Run `module load cylc`
2. Register the suite with cylc `cylc register run-esmvaltool-recipes ~/ESMValTool/esmvaltool/utils/testing/regression`
3. Edit the suite if needed, this allows e.g. choosing which recipes will be run
4. Validate the suite `cylc validate run-esmvaltool-recipes --verbose`, this will e.g. list the recipes in the suite

5. Run all recipes `cylc run run-esmvaltool-recipes`
6. View progress `cylc log run-esmvaltool-recipes`, use e.g. `cylc log run-all-esmvaltool-recipes examples-recipe_python_yml.1 --stdout` to see the log of an individual esmvaltool run. Once the suite has finished running, you will see the message "WARNING - suite stalled" in the log.
7. Stop the cylc run once everything is done `cylc stop run-esmvaltool-recipes`.

To generate an overview page of the recipe runs, use the `summarize.py` *utility script*.

10.4.2 Using the scripts in *utils/batch-jobs*

In utils/batch-jobs, you can find a script to generate slurm submission scripts for all available recipes in ESMValTool, as well as a script to parse the job outputs.

Using *generate.py*

The script generate.py, is a simple python script that creates slurm submission scripts, and if configured, submits them to the HPC cluster. It has been tested in DKRZ's Levante cluster.

The following parameters have to be set in the script in order to make it run:

- `env`, *str*: Name of the conda environment in which *esmvaltool* is installed.
- `mail`, *bool*: Whether or not to receive mail notifications when a submitted job fails or finishes successfully. Default is `False`.
- `submit`, *bool*: Whether or not to automatically submit the job after creating the launch script. Default value is `False`.
- `account`, *str*: Name of the DKRZ account in which the job will be billed.
- `outputs`, *str*: Name of the directory in which the job outputs (.out and .err files) are going to be saved. The outputs will be saved in */home/user/<outputs>*.
- `conda_path`, *str*: Full path to the *miniforge3/etc/profile.d/conda.sh* executable.

Optionally, the following parameters can be edited:

- `config_dir`, *str*: Path to configuration directory, by default `~/.config/esmvaltool/`.
- `partition`, *str*: Name of the DKRZ partition used to run jobs. Default is `interactive` to minimize computing cost compared to `compute` for which nodes cannot be shared.
- `memory`, *str*: Amount of memory requested for each run. Default is `64G` to allow to run 4 recipes on the same node in parallel.
- `time`, *str*: Time limit. Default is `04:00:00` to increase the job priority. Jobs can run for up to 8 hours and 12 hours on the compute and interactive partitions, respectively.
- `default_max_parallel_tasks`, *int*: Default is `8` which works for most recipes. For other cases, an entry needs to be made to the `MAX_PARALLEL_TASKS` dictionary (see below).

The script will generate a submission script for all recipes using by default the `interactive` queue and with a time limit of 4h. In case a recipe may require of additional resources, they can be defined in the `SPECIAL_RECIPES` dictionary. The recipe name has to be given as a `key` in which the values are another dictionary. The latter are used to specify the `partition` in which to submit the recipe, the new `time` limit and other `memory` requirements given by the slurm flags `--mem`, `--constraint` or `--ntasks`. In general, an entry in `SPECIAL_RECIPES` should be set as:

```
SPECIAL_RECIPES = {
 'recipe_name': {
     'partition': '#SBATCH --partition=<name_of_the_partition>',
```

(continues on next page)

(continued from previous page)

```
    'time': '#SBATCH --time=<custom_time_limit>',
    'memory': '#SBATCH --mem=<custom_memory_requirement>' # --constraint or --nstasks␣
↪can be used instead.
    },
}
```

Some recipes can only be run with a number of tasks less than `default_max_parallel_tasks` for various reasons (memory issues, diagnostic issues, CMIP3 data used). These recipes need to be added to the `MAX_PARALLEL_TASKS` dictionary with a specific `max_parallel_tasks` value.

Note that the script has been optimized to use standard SLURM settings to run most recipes while minimizing the computational cost of the jobs and tailored runtime settings for resource-intensive recipes. It is only necessary to edit this script for recipes that have been added since the last release and cannot be run with the default settings.

In the case in which `submit` is set to `True`, but you want to exclude certain recipes from being submitted, their name can be added in the `exclude` list:

```
exclude = ['recipe_to_be_excluded_1', 'recipe_to_be_excluded_2']
```

Using *parse_recipes_outputs*

You can run this script (simply as a standalone Python script) after all recipes have been run, to gather a bird's eye view of the run status for each recipe; running the script provides you with a Markdown-formatted list of recipes that succeeded, recipes that failed due to a diagnostic error, and recipes that failed due to missing data (the two most common causes for recipe run failure). You should provide the location of the output log files from SLURM (`*.out` and `*.err`) to the script as well as a list of all available recipes. To generate the list, run the command:

```
for recipe in $(esmvaltool recipes list | grep '\.yml$'); do echo $(basename "$recipe");␣
↪done > all_recipes.txt
```

To keep the script execution fast, it is recommended to use `log_level: info` in the configuration so that SLURM output files are rather small.

10.5 Overview of recipe runs

To create overview webpages of a set of recipe runs, run:

```
python esmvaltool/utils/testing/regression/summarize.py ~/esmvaltool_output/
```

This will generate 2 html files:

- `index.html` that displays a summary of each recipe run, with a title and a representative plot, a short description of the aim of the recipe, and links to each individual run.
- `debug.html` that provides an overview table of successful and failed runs with links to each individual run, and computing resources used for each run.

10.6 Comparing recipe runs

A command-line tool is available for comparing one or more recipe runs to known good previous run(s). This tool uses xarray to compare NetCDF files and difference hashing provided by imagehash to compare PNG images. All other file types are compared byte for byte.

The package imagehash is included in the standard ESMValTool environment. However, if you are using an environment that does not include imagehash, install it as follows:

```
pip install imagehash
```

Next, run the `esmvaltool develop compare` command:

```
esmvaltool develop compare ~/reference_output/recipe_python_20220109_110112/ ~/output/
↪recipe_python_20220310_180417/
```

where the first argument is a directory containing the output from a reference run and the second argument is a directory containing output from a test run to compare against the reference. This command will exit with a return code of 0 if the run outputs :w are the same, or 1 if they are different.

To compare results from a set of multiple runs to a reference set, e.g. between the current version of the code to the previous version, there is a wrapper script that loops over multiple comparisons. To use this go to the location where ESMValTool is installed and run e.g.:

```
python esmvaltool/utils/testing/regression/compare.py /shared/esmvaltool/v2.4.0 /shared/
↪esmvaltool/v2.5.0
```

To get more information on how a result is different, use the `--verbose` flag with either of the above commands.

10.7 draft_release_notes.py

draft_release_notes.py is a script for drafting release notes based on the titles and labels of the GitHub pull requests that have been merged since the previous release.

To use it, install the package pygithub:

```
pip install pygithub
```

Create a GitHub access token (leave all boxes for additional permissions unchecked) and store it in the file `~/.github_api_key`.

Edit the script and update the date and time of the previous release and run the script:

```
python esmvaltool/utils/draft_release_notes.py ${REPOSITORY}
```

`REPOSITORY` can be either `esmvalcore` or `esmvaltool` depending on the release notes you want to create.

Review the resulting output (in `.rst` format) and if anything needs changing, change it on GitHub and re-run the script until the changelog looks acceptable. In particular, make sure that pull requests have the correct label, so they are listed in the correct category. Finally, copy and paste the generated content at the top of the changelog.

10.8 Converting Version 1 Namelists to Version 2 Recipes

The xml2yml converter can turn the old xml namelists into new-style yml recipes. It is implemented as a xslt stylesheet that needs a processor that is xslt 2.0 capable. With this, you simply process your old namelist with the stylesheet xml2yml.xsl to produce a new yml recipe.

After the conversion you need to manually check the mip information in the variables! Also, check the caveats below!

10.8.1 Howto

One freely available processor is the Java based saxon. You can download the free he edition here. Unpack the zip file into a new directory. Then, provided you have Java installed, you can convert your namelist simply with:

```
java -jar $SAXONDIR/saxon9he.jar -xsl:xml2yml.xsl -s:namelist.xml -o:recipe.yml
```

10.8.2 Caveats/Known Limitations

- At the moment, not all model schemes (OBS, CMIP5, CMIP5_ETHZ…) are supported. They are, however, relatively easy to add, so if you need help adding a new one, please let me know!
- The documentation section (namelist_summary in the old file) is not automatically converted.
- In version 1, one could name an exclude, similar to the reference model. This is no longer possible and the way to do it is to include the models with another `additional_models` tag in the variable section. That conversion is not performed by this tool.

Authored by **Klaus Zimmermann**, direct questions and comments to klaus.zimmermann@smhi.se

10.9 Extracting a list of input files from the provenance

There is a small tool available to extract just the list of input files used to generate a figure from the `*_provenance.xml` files (see *Recording provenance* for more information).

To use it, install ESMValTool from source and run

```
python esmvaltool/utils/prov2files.py /path/to/result_provenance.xml
```

The tool is based on the prov library, a useful library for working with provenance files. With minor adaptations, this script could also print out global attributes of the input NetCDF files, e.g. the tracking_id.

10.10 Recipe Test Workflow (RTW)

The Recipe Test Workflow (*RTW*) is a workflow that is used to regularly run recipes so issues can be discovered during the development process sooner rather than later.

10.10.1 Recipe Test Workflow (RTW)

About

The Recipe Test Workflow (*RTW*) is a workflow that is used to regularly run recipes so issues can be discovered during the development process sooner rather than later.

Cylc v8 and *Rose* v2 are used as the workflow engine and application configuration system for the *RTW*, respectively. *Cylc* and *Rose* are not included in the ESMValTool environment as they are typically already centrally installed at sites e.g. JASMIN and the Met Office.

Currently tested recipes

The recipes tested with the *RTW* at each site are defined in the following files:

- metoffice
- dkrz
- jasmin

Hint

The *site/<site>/recipes.jinja* file provides more information.

User Guide

Quick Start Guide

- Clone the ESMValTool repository (which contains the *RTW*):

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

- Configure the *RTW*:

```
cd ESMValTool/esmvaltool/utils/recipe_test_workflow
rose edit
```

- Run the *RTW*:
  - at the Met Office:

```
export CYLC_VERSION=8
cylc vip -O metoffice
```

  - on JASMIN:
    - add the following line to your `~/.bashrc` file to ensure the Cylc and Rose executables can be found:

```
export PATH=/apps/jasmin/metomi/bin:$PATH
```

    - SSH to the Rose and Cylc server:

```
ssh -X cylc
```

    - run the RTW on JASMIN:

```
cd ESMValTool/esmvaltool/utils/recipe_test_workflow
export CYLC_VERSION=8
cylc vip -O jasmin
```

  - on DKRZ:
    - add the following line to your `~/.bashrc` file to ensure the Cylc and Rose executables can be found:

```
export PATH=/work/bd0854/metomi/bin:$PATH
```

    - run the RTW on DKRZ:

```
cd ESMValTool/esmvaltool/utils/recipe_test_workflow
cylc vip -O dkrz
```

- Optionally browse the logs using Cylc Review, a web service for browsing logs via an HTTP interface.

The workflow

An overview of the workflow

The *RTW* performs the following steps:

`install_env_file`

Description
Copies the environment file for *ESMValTool*, based on the `SITE` provided

Runs on
Localhost

Executes
The `cp` command from the *Rose* app

Details
Runs once at the start of the workflow

`get_esmval`

Description
Either clones the latest versions of *ESMValTool* and *ESMValCore* from GitHub, or gets the latest container image from DockerHub and converts to a singularity image, depending on `SITE`

Runs on
Localhost, or `COMPUTE` on JASMIN

Executes
The `clone_latest_esmval.sh` script (if cloning), or a `singularity build` command (if getting container) from the *Rose* app

Details
Runs at the start of each cycle

`configure`

Description
Creates the *ESMValTool* user configuration file and validates it

Runs on
Localhost

Executes
The `configure.py` script from the *Rose* app

Details
Runs each cycle after `get_esmval` has completed

`process`

Description
Runs the requested recipes using *ESMValTool*

Runs on
`COMPUTE`, which depends on the `SITE`

Executes
The *ESMValTool* command line script from the *Rose* app

Details
Runs each cycle for every recipe defined in the *RTW* after `configure` has completed

`compare`

Description
Compares the output from the `process` job with *KGOs*

Runs on
`COMPUTE`, which depends on the `SITE`

Executes
The `esmvaltool develop compare` command from the *Rose* app

Details
Runs each cycle for every recipe defined in the *RTW* after `process` has completed

housekeeping

Description
Removes the logs and data (including recipe outputs) from two days or more before the current cycle

Runs on
Localhost

Executes
The `rose_prune` application

Details
Runs at the end of each cycle (except the first cycle)

Design considerations

Portability

The *RTW* is portable; site-specific information can be found in the `site` and `opt` directories within the *RTW*. The files required are:

site/<site>/recipes.jinja
Contains all the recipes run at the `SITE`

Hint

- The file uses the Jinja2 templating language, which has a similar syntax to Python
- Jinja2 gives *Cylc* many powerful features; Cylc Jinja2 provides more information

site/<site>/runtime.cylc
Contains task definitions specific to the `SITE`, for example, `COMPUTE`

site/<site>/env-file
Contains details on how to set up the environment for ESMValTool at the `SITE`

opt/rose-suite-<site>.conf
Contains configuration items specific to the `SITE`, including `SITE` and where to find the required datasets

Metadata

The *RTW* uses Rose metadata. Every item defined in the suite configuration file (`rose-suite.conf`) will have an entry in the main metadata configuration file (`meta/rose-meta.conf`).

Resources

The resources used by the `process` jobs are defined in the `site/<site>/recipes.jinja` file, allowing the jobs to be configured by `SITE` as well as by recipe. This ensures only the required resources are requested when running each of the `process` jobs.

Running the RTW on DKRZ in "production mode"

"Production" is used to mean an intention to run the *RTW* on a continual, live basis to ensure recipe issues are captured when they occur.

In practice, the production option generates a web page with test results at: https://esmvaltool.dkrz.de/shared/esmvaltool/rtw/status_report.html

To run the *RTW* in production mode:

- add the following line to your `~/.bashrc` file to ensure the Cylc and Rose executables can be found:

```
export PATH=/work/bd0854/metomi/bin:$PATH
```

- run the RTW on DKRZ, installing the RTW into a `recipe_test_workflow_production` directory:

```
cd ESMValTool/esmvaltool/utils/recipe_test_workflow
cylc vip -O dkrz -O production -n recipe_test_workflow_production
```

How to add a recipe to the RTW

Overview

To add a recipe to the *RTW* you will:

- Run the recipe at your site
- Note the actual duration and memory usage
- Edit your site's recipe file
- Create the recipe's KGOs
- Request a review

The recipe will then run at your site whenever the *RTW* is run.

Preparation

1. Open a new ESMValTool issue on GitHub. Assign yourself to the issue and add the `Recipe Test Workflow (RTW)` label. ESMValTool issue #3663 provides an example.
2. Create a branch.
3. Run the recipe:
    - with the latest version of ESMValTool
    - on the compute server you use at your site

> **Hint**
>
> Your compute server is defined in the `site/<site>/runtime.cylc` file as follows:
>
> ```
> [[COMPUTE]]
>    platform = <your compute server>
> ```

4. Obtain the actual duration and memory usage of the recipe. This can be found either in the message printed to screen, or at the end of the `run/main_log.txt` file in the recipe output directory. The relevant lines will look something like:

```
YYYY-MM-DD HH:MM:SS:sss UTC [12345] INFO    Time for running the recipe was:␣
↪0:02:13.334742
YYYY-MM-DD HH:MM:SS:sss UTC [12345] INFO    Maximum memory used (estimate): 2.4 GB
[...]
YYYY-MM-DD HH:MM:SS:sss UTC [12345] INFO    Run was successful
```

Adding the recipe

1. Add the recipe in alphabetical order to either `FAST_RECIPES` or `MEDIUM_RECIPES` in the `site/<site>/recipes.jinja` file. It should look something like:

```
{
   'recipe_path': 'recipe_a_fast_recipe',
   'actual': '2m13s, 2.4 GB on YYYY-MM-DD',
   'max_time': 'PT3M',
   'max_memory': '3G',
}
```

Important

Add the recipe to `FAST_RECIPES` if it takes *less* than 10 mins to run at your site. Add the recipe to `MEDIUM_RECIPES` if it takes *more* than 10 mins.

Hint

The *site/<site>/recipes.jinja* file provides more information.

Hint

Set the `max_time` to 10-20% more than the actual duration. For actual durations of up to `1m45s`, set `max_time` to `PT2M` (2 minutes).

Hint

Try not to regularly waste more than 500 MiB in memory usage. Typically, rounding the actual memory usage up to the nearest integer is acceptable.

Create the KGOs

1. Run the *RTW*, as detailed in the *Quick Start Guide*; it is expected that the `compare` task will fail.

Important

The `compare` task fails because the *KGOs* for the recipe do not yet exist. This run of the *RTW* will generate the outputs that will be used as *KGOs*.

2. Recursively copy the recipe output directory `recipe_<recipe>_<date>_<time>/` to your site-specific *KGOs* directory:

```
cp -r ${HOME}/cylc-run/recipe_test_workflow/runN/share/cycle/<cycle>/recipe_<recipe>
↪_<date>_<time> <KGO_ROOT_PATH>
```

Hint

`<cycle>` will look something like: `20250101T0900Z`. The recipe output directory will look something like: `recipe_python_20250101_090000`

Hint

Find your site-specific `KGO_ROOT_PATH` in the `opt/rose-suite-<site>.conf` file.

Note

Cylc is typically configured such that `${HOME}/cylc-run/recipe_test_workflow/runN/share` is a symbolic link to a share directory located on a scratch disk; the recipe output directory is not stored in `${HOME}`.

3. Enable write permissions for all users on the recipe output directory in your site-specific *KGOs* directory:

```
chmod -R a+w <KGO_ROOT_PATH>/recipe_<recipe>_<date>_<time>
```

4. Stop any running `recipe_test_workflow` workflows:

```
cylc stop recipe_test_workflow/*
```

5. Run the *RTW* again, as detailed in the *Quick Start Guide*; the `compare` task should now succeed.

Important

Unless you want the *RTW* to continue running every night, repeat the previous step to stop the workflow.

Request a review

1. Commit and push your changes.
2. Create a PR. Assign yourself to the PR, and add the `Recipe Test Workflow (RTW)` label to the PR. ESMValTool PR #3664 provides an example.

Note

Reviewers will automatically be assigned to your PR.

Congratulations!

The recipe will now run at your site whenever the RTW is run.

Glossary

CMIP
Coupled Model Intercomparison Project, see the CMIP Documentation.

Cylc
Cylc ("silk") is a workflow engine for cycling systems, see the Cylc Documentation.

ESM
Earth System Model

ESMValCore
ESMValCore contains the core functionalities for the ESMValTool.

ESMValTool
ESMValTool is a community diagnostic and performance metrics tool for routine evaluation of *ESMs* in *CMIP*, see the ESMValTool Documentation.

KGOs
Known Good Outputs (KGOs) are outputs from an ESMValTool recipe that have been confirmed (typically by a scientist) to be as expected. KGOs are used in the compare task in the *RTW* to check whether any changes have been introduced to the outputs from an ESMValTool recipe after changes are made to either ESMValCore or ESMValTool.

Rose
Rose is a toolkit for writing, editing and running application configurations, see the Rose Documentation.

RTW
Recipe Test Workflow

CHAPTER

ELEVEN

FREQUENTLY ASKED QUESTIONS

11.1 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*.

11.2 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. ESMValTool uses the yamllint linter tool to check recipe syntax.

11.3 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 done successfully, provided that the diagnostic script supports this.

11.4 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.

11.5 Using multiple configuration directories

By default, ESMValTool will read YAML configuration files from the user configuration directory `~/.config/esmvaltool`, which can be changed with the `ESMVALTOOL_CONFIG_DIR` environment variable. If required, users can specify the command line option `--config_dir` to select another configuration directory, which is read **in addition** to the user configuration directory See the section on configuration YAML files for details on this.

11.6 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 run recipe_example.yml; \
ln -sfT $(ls -1d ~/esmvaltool_output/recipe_example_* | tail -1) ~/esmvaltool_output/
↪recipe_example_latest
```

11.7 Can ESMValTool plot arbitrary model output?

General model evaluation provides a set of recipes that can be used for a basic climate model evaluation with observational data. This is especially useful to get an overview of the general performance of a simulation.

Furthermore, recipe *Monitor* allows for the plotting of any preprocessed model. The plotting parameters are set through a yaml configuration file, and the type of plots to be generated are determined in the recipe.

Similarly, recipe *Seaborn Diagnostics* and the corresponding diagnostic *seaborn_diag.py* provide a high-level interface to the Seaborn package which can also be used to create a large variety of different plots.

See also *General-purpose recipes*.

11.8 Debugging FAQ

11.8.1 Something's going wrong in preprocessing, how can I find out where?

Sometimes there is a problem in the preprocessing, often because of faulty data.

To locate the problem, the first step is to edit your *config-user.yml* to set

```
log_level: debug
```

Then run *esmvaltool* again and try to understand the error message. Don't forget that you can open the error log in the run directory instead of working only with the terminal output. Usually, the error log ends in a python traceback. See for example here for an introduction to understanding tracebacks.

It can also be helpful to look in the error log directly above the traceback for more information on what caused the problem.

If the error message is not enough to diagnose the problem, you can "spread out" the effects of the different preprocessors. For that, edit again *config-user.yml* to set

```
save_intermediary_cubes: true
remove_preproc_dir: false
```

This way, you can inspect the output of every preprocessing step individually by looking at the individual outputs in the *preproc* directory. Usually, the last existing file can give you a clue on what went wrong.

Warning

This can be costly both in storage space and computing time, so be careful to use as small a dataset as possible to diagnose your problem and to deactivate the setting when you are done debugging!

11.8.2 I get a *iris.exceptions.ConcatenateError*. What's the problem?

Broadly speaking, the problem is that data that was spread over different files was supposed to be part of one dataset, but doesn't follow the same conventions. This can be due to different coordinates, different attributes, or a number of other differences. Unfortunately, it can be difficult to understand what exactly is the difference between the data.

To diagnose the problem, first, figure out which netcdf files cause the issue. If it happens in the preprocessing, refer to this FAQ for some pointers.

Next, try to understand the differences between the two files. Perhaps the error message already pointed to the area of difference, for example some attributes or particular coordinates. In that case you can focus on those first.

11.8.3 How can I compare two (or more) netcdf files?

There are three popular tools/methods to understand the differences between netcdf files. The first two, looking at the metadata of the individual files and visualizing their data, are almost always possible and can give you a quick idea of what's going on. They are, however, often not suited to detailed comparisons, for example when the actual data contains subtle problems or when the problem is in the values of coordinates. In those cases, turn to a more in-depth look at the data. This last method also works better when you have a larger collection of files and are not sure where exactly the problem lies.

Inspecting the netcdf file headers

You can use the *ncdump* utility to look at the metadata of netcdf files. More specifically, *ncdump -h my_file.nc* will output the so-called header of the file that contains information about all the variables present in the file, along with their units, etc. This includes the coordinate variables. It also will give you all the global, ie file level, and local, ie variable level attributes. It will, however, not give you the coordinate values or the data values itself. It will also not help you with comparison, so you can only compare the output for two files manually.

Nevertheless, when you want to get a quick overview, or when you have some idea what to look for, or when the other methods don't work, this is a good starting point.

Visualizing the data

Often a quick look at the data can reveal the most glaring problems, like missing or wrong masks, or wrong units. There are a number of tools that can be used for this, but one of the most basic and almost universally available ones is *ncview*. This can also be installed with conda from conda-forge.

Just do a quick *ncview my_file.nc* to open the gui. Then select the variable and use the navigation buttons to flip through the timesteps and, in the case of 3d data, the levels.

Again, this tool doesn't help you with comparison, but you can open two instances to compare two files in side-by-side.

Comparing data with iris

Iris is a powerful python framework for working with CF compliant netcdf files. It also includes some support for other formats like grib, and pp files. Iris is the foundation for esmvaltool, but it can also be used on its own. Two convenient ways to do that are ipython and jupyter notebooks. This FAQ cannot cover any of these tools in great detail, so please refer to their respective documentation for an overview and in-depth questions. Instead, here we will give just a quick way to understand differences in a set of netcdf files with iris.

Iris loads data with one of several functions whose names start with *load*. These functions take a single string or a list of strings that are interpreted as filenames. They may also contain directories and globs, like *./my_data/tas_*.nc* to load all surface temperature data from the *my_data* subdirectory of the current working directory. The iris function will return either a single Cube, or a list of cubes in the form of a CubeList.

Often, we want to combine different parts of dataset to form the whole dataset again. This is accomplished in iris using either a *merge* or a *concatenate*. Refer to the iris user guide to understand the difference.

In fact, it is this *concatenate* that has failed you when you are reading this FAQ entry.

To find out what went wrong, let's assume we have boiled down the problem to the two files *tas_1969.nc* and *tas_1970.nc*. Then one way to learn more about the issue is the following.

```
import iris
cube_list = iris.load('tas_19*.nc')
cube_list.concatenate_cube()
```

This will likely produce the exact same error that we started with. While we do want to do a concatenation, since both files contain several timesteps each, the output of the corresponding merge function is often more illuminating. So let's give that a try.

```
cube_list.merge_cube()
```

Hopefully, that gives you a better idea of the problem.

If you are dealing with more than two files it can be difficult to figure out which files are actually the culprits. In that case try

```
import iris
cube_list = iris.load('tas_*.nc')
concatenated_cube_list = cube_list.concatenate()
print(concatenated_cube_list)
```

This will concatenate all the consecutive bits it can manage and keep only those parts separate, that cannot be merged. This way, you can often boil down the problem to only two or a few cubes. Then you can diagnose those in the same way as discussed above, eg

```
concatenated_cube_list.concatenate_cube()  # or
concatenated_cube_list.merge_cube()
```

You can also inspect the attributes and coordinates of the resulting cubes with a particular focus on the areas of problems pointed to by the output of *merge_cube*.

CHAPTER

TWELVE

CHANGELOG

12.1 v2.14.0

12.1.1 Highlights

- **New diagnostics/recipes:**
  - Calculation of Lamb weathertypes (Pull request #4231)
  - Histogram plots (Pull request #4329)
  - Hour of maximum precipitation (Pull request #4334)
- **Support for new observational datasets:**
  - ESACCI SNOW (Pull request #3542)
  - Improved support for ESACCI AEROSOL (Pull request #3629)
  - EN4 (Pull request #4193)
- We modernized our development set-up and are now using Ruff for *Code quality* checks and formatting.

This release includes

12.1.2 Backwards incompatible changes

- Update the CMORizer command (Pull request #4317, Pull request #4326) by @bouweandela

> **Upgrade instructions**
>
> The `--config-file` argument to the `esmvaltool data` commands has been removed. Please use these commands with the `--config-dir` argument to specify the output directory, custom CMOR tables, or Dask configuration, and with the `--original-data-dir` argument to point the tool to the directory where downloaded original data should be saved.

12.1.3 Bug fixes

- Fix recipe_enso_climatology1 - remove collect diagnostic (Pull request #4221) by @flicj191
- Rename `modeling_realm` to `type` in CMORizers (Pull request #4319) by @bouweandela
- Fix `multi_datasets.py` so it can handle different projections (Pull request #4358) by @schlunma

12.1.4 Broken or retired recipes

- Remove `recipe_russel18jgr.yml` from list of broken recipes (Pull request #4351) by @schlunma
- Retire `recipe_climwip_brunner2019_med.yml` (Pull request #4350) by @schlunma
- Add `recipe_kcs.yml` to list of broken recipes (Pull request #4354) by @schlunma
- Retire Wenzel recipes (Pull request #4357) by @schlunma
- Remove `recipe_check_obs.yml` from list of broken recipes (Pull request #4376) by @schlunma
- Retire `recipe_eyring*jgr.yml` (Pull request #4379) by @schlunma

12.1.5 Documentation

- Separate ESMValTool and ESMValCore documentation (Pull request #3914) by @flicj191
- Fix PDF documentation build (Pull request #4236) by @bouweandela
- Update links to ESMValCore configuration (Pull request #4249) by @bouweandela
- Document that we follow SPEC 0 (Pull request #4246) by @bouweandela
- Fix stable documentation build (Pull request #4253) by @bouweandela
- Update some easy to fix links in documentation (Pull request #4256) by @katjaweigel
- Add link checker github action for documentation (Pull request #4251) by @flicj191
- Add ENES-RI Zenodo community (Pull request #4254) by @bouweandela
- Schedule for link checker and update for recent report (Pull request #4305) by @flicj191
- Pin sphinx<9 (Pull request #4315) by @valeriupredoi
- Add Matomo page view tracking for the documentation (Pull request #4320) by @bouweandela
- Fix matomo integration (Pull request #4322) by @bouweandela
- Remove mentions of the deprecated config-developer.yml file from the documentation (Pull request #4335) by @bouweandela
- Documentation update -remove gensidebar (Pull request #4345) by @flicj191
- Update documentation on defining custom variables (Pull request #4349) by @bouweandela
- Update contact info (Pull request #4348) by @axel-lauer
- Remove mention of deprecated `esmvaltool config get_config_user` command (Pull request #4366) by @schlunma
- Update contact email in documentation (Pull request #4383) by @axel-lauer

12.1.6 Diagnostics

- Calculating Lamb weathertypes for CMIP5, CMIP6, SMILES and EURO-CORDEX (Pull request #4231) by @thomaskroi1996
- Add diagnostic to plot a histogram (Pull request #4329) by @LisaBock
- Add diagnostic to calculate hour of maximum precipitation (Pull request #4334) by @schlunma
- Make emergent constraints diagnostics compatible with Pandas>=3 (Pull request #4356) by @schlunma
- Allow specifying `axes_kwargs` in `monitor/multi_datasets.py` (Pull request #4365) by @schlunma

12.1.7 Observational and re-analysis dataset support

- ESACCI SNOW CMORizer (Pull request #3542) by @axel-lauer
- Update CMORizer CERES-EBAF to v4.2 (Pull request #3360) by @axel-lauer
- Fix JRA-55 downloader (Pull request #4191) by @LisaBock
- EN4 cmorizer (Pull request #4193) by @hanellis
- New ESACCI AEROSOL CMORizer (Python version) (Pull request #3629) by @axel-lauer
- Extension of ESA CCI OZONE CMORizer (Pull request #4125) by @axel-lauer

12.1.8 Release

- Modernize release notes script (Pull request #4336) by @schlunma

12.1.9 Automatic testing

- Run Github Action install from conda with Python 3.13 also (Pull request #4229) by @valeriupredoi
- Fix WOA CMORizer tests (Pull request #4243) by @valeriupredoi
- Update recipe tests to align with ESMValCore development (Pull request #4268) by @bouweandela
- Fix tests for new esmvalcore config (Pull request #4252) by @valeriupredoi
- Clean up OSX Github Action tests (Pull request #4272) by @valeriupredoi
- Update recipe load test so it works with the latest ESMValCore (Pull request #4294) by @bouweandela
- Increase CirleCI container resource to medium+ (Pull request #4364) by @valeriupredoi
- Update batch job generation script with experiences from latest tests (Pull request #4378) by @schlunma

12.1.10 Installation

- Use `pip install --no-deps` to avoid overwiting packages from the conda environment (Pull request #4182) by @bouweandela
- Drop support for Python 3.11 (Pull request #4273) by @bouweandela
- Pin iris <3.14.1 to comply with iris-esmf-regrid 0.13 (Pull request #4284) by @valeriupredoi
- Update package build and deploy (Pull request #4289) by @bouweandela
- Drop `ruamel.yaml` as a dependency (Pull request #4295) by @bouweandela
- Unpin iris (Pull request #4304) by @valeriupredoi
- Remove R development dependencies (Pull request #4306) by @valeriupredoi
- Update environment.yml with R dependency r-r.utils for recipe_martin18grl and recipe_spei (Pull request #4355) by @katjaweigel
- Pin numpy >=2 (Pull request #4375) by @valeriupredoi

12.1.11 Improvements

- Switch from Prospector to ruff (Pull request #4209) by @bouweandela
- Fix Codacy configuration (Pull request #4262) by @bouweandela
- Apply automatic fixes for style issues (Pull request #4263) by @bouweandela

- Update obs4MIPs dataset names (Pull request #4321, Pull request #4377) by @bouweandela and @valeriupredoi
- Add `keep_group_coordinates: true` to recipes that fail without it (Pull request #4352) by @schlunma
- Load configuration before accessing CMOR tables (Pull request #4363) by @bouweandela
- Avoid failing tests on coverage upload from forks (Pull request #4371) by @bouweandela

12.2 v2.13.0

12.2.1 Highlights

Constellation of CMORizers and diagnostics updated/implemented as part of the Rapid Evaluation Framework (REF):

- CMORizers:
  - Update of the ESACCI-OZONE CMORizer
  - ESACCI-SEAICE CMORizer
  - Update of the OSI-450 (OSISAF/CCI) CMORizer
- *Diagnostics*:
  - Seasonal cycle and time series of Arctic/Antarctic sea ice area
  - *Zero Emissions Commitment (ZEC)*
  - Cloud radiative effects
  - *Transient Climate Response to Emissions (TCRE)*
  - *ENSO metrics*
  - 2D histograms and scatterplots for clouds
  - Regional historical changes
  - *Evaluation of climate variables at Global warming levels*
  - *Sea surface temperature bias (IPCC AR6 Ch3 Fig. 3.24)*
  - *Climate drivers for fire*
  - *Rate of sea ice area loss per degree of warming*
  - Ozone metrics and climatology

This release includes

12.2.2 Backwards incompatible changes

- Refactor diagnostic *monitor/multi_datasets.py* (Pull request #4002) by @schlunma Plot option *annual_mean_kwargs*: This will be ignored now. Previously, this could lead to errors in rare cases. Instead, explicitly specify a dataset that uses the annual_statistics preprocessor in the recipe, for example *esmvaltool/recipes/monitor/recipe_monitor_with_refs.yml#L132-L139*.
- Retire recipe schlund20jgr (Pull request #4022) by @axel-lauer
- Retire julia recipes (Pull request #4045) by @bettina-gier
- Retire psyplot recipe and diagnostic (Pull request #4063) by @valeriupredoi

12.2.3 Community

- Added stale action (Pull request #4100) by @schlunma

12.2.4 Documentation

- Update citation and changelog for version 2.12 (Pull request #3945) by @sloosvel
- Use Python-only environment for ReadTheDocs builds (Pull request #3955) by @valeriupredoi
- Add explanatory note in documentation advice on Codacy (Pull request #4020) by @alistairsellar
- Extend broken recipe policy (Pull request #3913) by @axel-lauer
- Remove some more Julia references from documentation (but keep some) (Pull request #4073) by @valeriupredoi
- Add footer with legal info (Pull request #4072) by @axel-lauer
- Add mention of NSC HPC servers (Pull request #4084) by @jlenh
- Update figure selection and layout of the gallery (Pull request #4060) by @bettina-gier
- Update timeline for release 2.13.0 (Pull request #4096) by @jlenh
- Refer to esmvaltool.org webpage for development team information (Pull request #3892) by @bouweandela
- Remove mention of Julia in documentation ahead of discontinued support in v2.13 (Pull request #4156) by @jlenh
- Update *cartopy* documentation link to fix documentation build fail (Pull request #4179) by @jlenh
- Add debugging help (Pull request #2638) by @zklaus

12.2.5 Diagnostics

- Add recipe for seasonal cycle and time series of Arctic/Antarctic sea ice area (REF) (Pull request #3891) by @axel-lauer
- Add recipe and diagnostic for ZEC (Zero Emissions Commitment) (Pull request #3897) by @bettina-gier
- Create scatterplots for the REF (Pull request #3923) by @LisaBock
- Evaluate key climate variables at Global warming levels (Pull request #3927) by @rswamina
- Add SST bias diagnostic for IPCC AR6 Ch3 Fig. 3.24. Can be used for REF. (Pull request #3944) by @malininae
- Update recipe_combined_indices.yml for R versions larger than 4.2 (Pull request #3967) by @bouweandela
- Added option to plot horizontal lines in monitoring line plots (Pull request #3977) by @schlunma
- Fix R recipes for R > 4.2 (Pull request #3985) by @sloosvel
- REF ozone recipe (Pull request #3988) by @hb326
- Faster and lower memory version of recipe_ref_scatterplot.yml (Pull request #3992) by @bouweandela
- Update GPCP and ERA5 obs4MIPs datasets in recipe_ref_scatterplot.yml (Pull request #4011) by @bouweandela
- Add option to specify caption in recipe for *monitor/multi_datasets.py* (Pull request #4028) by @schlunma
- Optimize caption produced by *tcr.py* (Pull request #4042) by @schlunma
- Optimize caption produced by *ecs.py* (Pull request #4040) by @schlunma
- Fix tcre metric diagnostic (Pull request #4044) by @valeriupredoi
- Optimize *tcre.py* diagnostic (caption and usability via Python API) (Pull request #4056) by @schlunma

- Add additional diagnostics to recipe_ocean_quadmap.yml (Pull request #3953) by @SophieHall2024
- New CH4 lifetime diagnostic (Pull request #3507) by @FranziskaWinterstein
- New recipe: rate of sea ice area loss per degree warming (Pull request #3983) by @alistairsellar
- Remove unused option *annual_mean_kwargs* from all recipes (Pull request #4133) by @schlunma
- Aeronet format updates and diagnostic changes (Pull request #4131) by @nchawang
- Use obs4MIPs consistently in all recipes (Pull request #4142) by @bouweandela
- Add missing method in *seaice_drift.py* (Pull request #4161) by @sloosvel
- Update the name of the *remapscon2* operator in R recipes (Pull request #4164) by @jlenh
- Trace gases (CO2, CH4, N2O) surface concentration diagnostic (Pull request #4014) by @jlenh
- Climate drivers for fire for REF (Pull request #3975) by @jlenh
- Exchange toz reference data in ozone REF recipe (Pull request #4008) by @axel-lauer
- Add captions to REF recipe for basic sea ice area diags (Pull request #4033) by @axel-lauer
- CLIVAR ENSO metrics for REF (Pull request #3972) by @flicj191
- Update and combine drought diagnostics (Pull request #3907) by @lukruh
- Regional historical changes for REF (Pull request #3926) by @katjaweigel
- Add more datasets to recipe_easy_ipcc.yml (Pull request #3920) by @bouweandela
- ACCESS-NRI ENSO diagnostics CLIVAR porting (Pull request #3905) by @flicj191
- Correct typo in "pearsonr" (Pull request #4140) by @katjaweigel
- Update time ranges in REF 'cloud radiative effect' recipe (Pull request #4057) by @LisaBock
- Fix caption and time range for REF diagnostic "Cloud scatterplots" (Pull request #4066) by @LisaBock
- Portrait plot example: remove sea surface temperature (Pull request #4099) by @lukruh
- Recipes ipccwg1ar5ch9: Update CERES and GPCP versions to match ESGF (Pull request #3440) by @rbeucher
- Recipe_schlund20esd Remove CERES-EBAF version to fix ESGF search (Pull request #3441) by @rbeucher
- Adding basic sanity checks (Pull request #4120) by @LisaBock
- Adding optional data output for seaborn_diag.py (Pull request #4144) by @katjaweigel
- Fix *recipe_williams09climdyn_CREM.yml* by replacing *np.NAN* with *np.nan* (Pull request #4160) by @jlenh
- Fix *recipe_carvalhais14nat.yml* to avoid the usage of *vars()* inside a function (Pull request #4170) by @jlenh
- Fix small matplotlib errors in *recipe_zmnam.yml* (Pull request #4177) by @jlenh
- Fix *_fix_lons* function in *recipe_flato13ipcc_figure_914.yml* due to *numpy* error [#4034] (Pull request #4178) by @jlenh
- Improve the handling of options in recipe_ref_fire.yml (Pull request #4184) by @bouweandela
- Fix small matplotlib errors in *recipe_zmnam.yml* (Pull request #4177) by @jlenh
- Fix minor NCL errors for *recipe_russel18jgr.yml* (Pull request #4183) by @jlenh
- Fix specification for WOA dataset in recipe_ocean_quadmap (Pull request #4176) by @TomasTorsvik
- Fix autoassess normalise error (Pull request #4187) by @alistairsellar

12.2.6 Observational and re-analysis dataset support

- Update ESACCI-OZONE CMORizer(formatter and downloader) for REF (Pull request #3899) by @diegokam
- Update CMORizer for OSI-450 aka OSISAF/CCI (sea ice concentration) (Pull request #3963) by @axel-lauer
- Adding HadEX3 CMORizer (Pull request #3636) by @malininae
- Downloader & cmorizer for NOAA GML surface flask trace gases (Pull request #3847) by @jlenh
- Use nested update to parse configuration options given to CMORizer interface (Pull request #4113) by @schlunma
- Aeronet-header-fix to handle V3 data (Pull request #4101) by @nchawang
- Cmorizer for the Wetland Area and Dynamics for Methane Modeling (WAD2M) dataset (Pull request #4110) by @jlenh
- Cmorizer for the Global Lakes and Wetlands Database (GLWD) version 2.0 dataset (Pull request #4112) by @jlenh
- Update ESACCI Cloud CMORizer (daily and monthly data) (Pull request #3756) by @diegokam
- Fix coords in ESACCI-CLOUD cmorizer (Pull request #4129) by @LisaBock
- Added CMORizer for Yang2020 data (Pull request #4090) by @schlunma
- Add CMORized NOAA GML surface flask concentration measurement of N2O (Pull request #4059) by @jlenh
- CMORizer for ESACCI-BIOMASS (Pull request #4121) by @axel-lauer
- Faster OSI-450-nh and OSI-450-sh CMORizer and respect Dask settings (Pull request #3990) by @bouweandela
- CMORiser for the IAPv4.2 global ocean temperature dataset (Pull request #3887) by @rbeucher
- Update recipe_check_obs.yml to match the latest AERONET data (Pull request #4171) by @hb326
- Update version facet for ESACCI-CLOUD in recipe_check_obs.yml (Pull request #4166) by @jlenh
- Remove *end_year* facet for NOAA-GML datasets in *recipe_check_obs.yml* (Pull request #4167) by @jlenh
- New version of JRA-25 CMORizer (Pull request #3470) by @axel-lauer
- Add CAMS cmorizer (Pull request #3749) by @bettina-gier

12.2.7 Automatic testing

- Use mamba>=2 for Circle CI upstream development test (Pull request #3949) by @valeriupredoi
- Use python-only environment file for Circle build documentation (Pull request #3981) by @valeriupredoi
- Use conda exec instead of mamba for circleCI *build_documentation* test (Pull request #4003) by @valeriupredoi
- Run pre-commit on main (Pull request #4026) by @schlunma
- Try set a longer no output time limit for Circle tests (Pull request #4035) by @valeriupredoi
- Only run imagehash tests on Linux machines (Pull request #4039) by @schlunma
- Do not run conda install package Github Action with Python 3.13 (yet) (Pull request #4071) by @valeriupredoi
- Fix Cicle CI tests (prepend Julia to PATH for the containerized run tests) (Pull request #4086) by @valeriupredoi
- Automatically open pull requests to update GitHub Actions (Pull request #4143) by @valeriupredoi
- Run esmvalbot test PR twice a month (Pull request #3976) by @bouweandela
- Add DKRZ resources for RTW to *dkrz-recipes.jinja* (Pull request #3986) by @ehogan

- Add JASMIN resources for RTW to *jasmin-recipes.jinja* (Pull request #3987) by @ehogan
- Use site directories in the RTW (Pull request #3984) by @ehogan
- Update MO paths in the RTW (Pull request #4006) by @ehogan
- Select directories for comparison carefully in the RTW (Pull request #4007) by @ehogan
- Implement additional user configuration options in RTW (Pull request #4009) by @ehogan
- Generate "recipe status" HTML after each Recipe Test Workflow cycle (Pull request #4013) by @chrisbillowsMO
- Update RTW to use the *git* scheme (Pull request #4030) by @ehogan
- Add housekeeping to RTW (Pull request #4019) by @ehogan
- Revert "Update RTW to use the *git* scheme" (Pull request #4041) by @ehogan
- Update location of *recipe_consecdrydays.yml* in the RTW (Pull request #4052) by @ehogan
- Monitor Tests workflow fails (Pull request #4065) by @chrisbillowsMO
- RTW status report shows missing compare tasks as succeeded (Pull request #4064) by @chrisbillowsMO
- Use compare command in recipe test workflow (Pull request #4087) by @alistairsellar
- Separate *BRANCH* variable for recipe_test_workflow cloning Core and Tool (Pull request #4070) by @alistairsellar
- Update RTW to use new user configuration (Pull request #4017) by @ehogan

12.2.8 Installation

- Remove Julia from conda dependency environment (but keep tests for *esmvaltool install Julia*) and add docs how to install from source (Pull request #3921) by @valeriupredoi
- Unpin upper limit *r-base* (Pull request #3971) by @valeriupredoi
- Drop support for Python 3.10 (Pull request #4027) by @schlunma
- Support Python 3.13 (Pull request #3805) by @valeriupredoi
- Tackle Julia in DockerHub container builds: remove nstallation of packages (and remove dependencies in Project.jl bar :beer: YAML) (Pull request #4083) by @valeriupredoi
- Use *pyproject.toml* and retire *setup.py* with NO Ruff configuration changes (Pull request #4088) by @valeriupredoi
- Update environment to use ESMValCore v2.13.0rc1 (Pull request #4169) by @bouweandela
- Update ESMValCore version also for OSX (Pull request #4174) by @bouweandela

12.2.9 Improvements

- Replace flake8, yapf, and isort with ruff (Pull request #3893) by @bouweandela
- Remove *test_recipe* command (Pull request #4031) by @alistairsellar
- Replace deprecated horizontal regridding scheme *linear_extrapolate* with generic scheme (Pull request #4081) by @schlunma
- Add compare command, via new develop command group (Pull request #4054) by @alistairsellar
- Add extra Ruff rules and switch to *requires-python = ">=3.11"* in pyproject.toml (Pull request #4085) by @valeriupredoi
- Move dependabot configuration to correct path (Pull request #4145) by @bouweandela

- Use trusted publisher routine for PyPI and retire token-based type (Pull request #4150) by @valeriupredoi
- Fix pypa Github Action version (Pull request #4151) by @valeriupredoi

12.3 v2.12.0

12.3.1 Highlights

- Recipe *recipe_portrait_CMIP.yml* allows the creation of portrait plots to visualize performance metrics.
- A set of new *recipes* allow plotting arbitrary preprocessor output to benchmark a model simulation with other datasets.

This release includes

12.3.2 Backwards incompatible changes

- Remove recipe filler utility (Pull request #3777) by @schlunma. ESMValTool accepts the use of wildcards in the recipe, and a filled version of the recipe is automatically created and stored in the *run* directory.

12.3.3 Deprecations

- Adapt ESMValTool to new configuration (Pull request #3761) by @schlunma
  - The single configuration file `config-user.yml` has been deprecated in favour of configuration directories. By default, the directory `~/.config/esmvaltool` will be considered. To switch to the new format run:

    ```
    mkdir -p ~/.config/esmvaltool && mv ~/.esmvaltool/config-user.yml ~/.
    ↪config/esmvaltool
    ```

    You can also specify the location of the configuration directory with the `--config_dir` flag. Please refer to Configuration for a detailed description on how to configure the tool.

12.3.4 Bug fixes

- Fix for setting global attributes in cmorizers (Pull request #3717) by @LisaBock

12.3.5 Community

- Add a security policy (Pull request #3456) by @ehogan

12.3.6 Documentation

- Ignore autosummary warning in documentation build (Pull request #3718) by @valeriupredoi
- Retire mambaforge - addendum to 3774 (Pull request #3778) by @valeriupredoi
- Update Docker builds badge in README (Pull request #3783) by @valeriupredoi
- Readthedocs configuration/builds: revert to miniconda before miniforge is available (Pull request #3785) by @valeriupredoi
- Add info on obs tiers to docu (Pull request #3624) by @axel-lauer
- Fix wrong version in title in Readthedocs documentation (Pull request #3815) by @valeriupredoi
- Removed wrong description of option in portrait plot docstring (Pull request #3908) by @schlunma
- Update broken recipes (Pull request #3940) by @sloosvel
- Add next release schedule (Pull request #3794) by @sloosvel

12.3.7 Diagnostics

- More flexible file loading in *monitor/multi_datasets.py* (Pull request #3728) by @schlunma
- Add option to plot time on x-axis in monitoring Hovmoeller plots (Pull request #3732) by @FranziskaWinterstein
- Use *transform_first=True* for contourf plots with Robinson projection to avoid cartopy bug (Pull request #3789) by @schlunma
- Fix contourf plots for masked data (Pull request #3797) by @schlunma
- Fix issue related to removal/change of private function imported in *diag_scripts/shared/_supermeans.py* (deprecation in iris=3.11) (Pull request #3810) by @valeriupredoi
- Allow setting *matplotlib.rcParams* in *monitor/multi_datasets.py* (Pull request #3844) by @schlunma
- Adding figures of Bock and Lauer (2024) (Pull request #3526) by @LisaBock
- Added Python portrait plot diagnostic (Pull request #3551) by @lukruh
- Fix Autoassess diagnostic for new matplotlib 3.10 api change (Pull request #3917) by @valeriupredoi
- Fix diagnostic mpqb (Pull request #3918) by @sloosvel
- Add diagnostic to calculate Transient Climate Response to Emissions (TCRE) (Pull request #3904) by @schlunma
- Benchmarking recipes (Lauer et al.) (Pull request #3598) by @axel-lauer
- Add Cloud Radiative Effects recipe for REF (Pull request #3903) by @LisaBock

12.3.8 Observational and re-analysis dataset support

- Update esacci-soilmoisture(v08.1) downloader and CMORizer (Python version) (Pull request #3676) by @diegokam
- Update ESACCI Landcover CMORizer (python version) and downloader (pft yearly data, v2.0.8) (Pull request #3727) by @diegokam
- CMORizer for JRA-55 (Pull request #3141) by @axel-lauer
- Replace underscores in facets of observational datasets with minuses (Pull request #3840) by @schlunma
- Add TropFlux CMORiser (Pull request #3863) by @rbeucher
- CMORizer for ESACCI-SEAICE (Pull request #3821) by @axel-lauer
- Update ESACCI-SST cmorizer to v3.0 (Pull request #3697) by @LisaBock
- Adding pr, tauu, tauv NOAA-CIRES-20CR-V2 CMORISER (Pull request #3763) by @max-anu
- Adding pr, tauu, tauv, tos to NCEP2 CMORISer (Pull request #3765) by @max-anu
- Adding a CMORiser for CMAP data for pr (Pull request #3766) by @max-anu
- Update NSIDC_G02202_sh CMORiser to add bounds for lat,lon and time (Pull request #3744) by @flicj191

12.3.9 Automatic testing

- Remove obsolete utility *esmvt_rose_wrapper* and its documentation and very obsolete *mip_convert* cmorizer (Pull request #3759) by @valeriupredoi
- Remove obsolete and inactive *tests/system* tests (Pull request #3760) by @valeriupredoi
- Retire Mambaforge (Pull request #3774) by @valeriupredoi
- Pin mamba<2 for conda-lock: solution by Ben Mares @maresb (Pull request #3771) by @valeriupredoi

- Update comment in conda lock creation Github action (Pull request #3788) by @valeriupredoi
- Run a periodic esmvalbot test via automated PR (Pull request #3465) by @valeriupredoi
- Add init files in package-like diagnostic directories to comply with *pytest* >=*8.3.4* (Pull request #3834) by @valeriupredoi
- Fix failing sklearn test (Pull request #3850) by @schlunma
- Fix circle ci nightly test installation from source development mode (and test mode) that times out (Pull request #3864) by @valeriupredoi
- Remove portrait plot test recipe (Pull request #3871) by @schlunma
- Restrict cron Github Actions tests to run from forks (Pull request #3894) by @valeriupredoi
- Force mamba 2 in Github Actions that solve environment (Pull request #3929) by @valeriupredoi
- Force install mamba2 in CircleCI and add ipython and pin it to <9.0 (Pull request #3947) by @valeriupredoi
- Use mamba>=2 for Circle CI upstream development test (Pull request #3949) by @valeriupredoi

12.3.10 Installation

- Retire support for Python 3.9 (Pull request #3683) by @valeriupredoi
- [Julia] pin *curl* <*8.10* to restrict *libcurl* <*8.10* so Julia installs packages correctly (Pull request #3755) by @valeriupredoi
- Add support for Python=3.12 (Pull request #3501) by @valeriupredoi
- Pin cartopy to *cartopy*<*0.24* (Pull request #3768) by @valeriupredoi
- Pin pys2index >=0.1.5 in osx environment (Pull request #3792) by @valeriupredoi
- Update environment: pin *iris*>=*3.11*, unpin *cartopy* and allow for *numpy* >=*2* (Pull request #3811) by @valeriupredoi
- Unpin pandas (Pull request #3924) by @valeriupredoi

12.3.11 Improvements

- Update ERA5 renaming script for hourly (Pull request #3630) by @malininae
- Avoid masking issues in Dask 2024.8.0 (Pull request #3736) by @bouweandela
- Dark mode compatible transparent background logo (Pull request #3751) by @lukruh
- Change authors name (Pull request #3806) by @lukruh
- Recipe Test Workflow (RTW) prototype (Pull request #3210) by @ehogan
- Update obs4MIPs names to match ESGF see #2974 (Pull request #3439) by @rbeucher
- Recipe_wenzel16jclim: Remove CERES-EBAF version to fix ESGF (Pull request #3442) by @rbeucher
- Update RTW checks (Pull request #3823) by @ehogan
- Add cccma collaborators (Pull request #3831) by @malininae
- Add DKRZ as a site to RTW (Pull request #3837) by @ehogan
- Add an option to disable using the distributed scheduler from the diagnostic script (Pull request #3787) by @bouweandela
- Use site-specific lists of recipes in the RTW (Pull request #3856) by @chrisbillowsMO
- Add links to summary page that are required for hosting at DKRZ (Pull request #3866) by @bouweandela

- Add *filename* to list of ignored attributes in *compare.py* (Pull request #3919) by @sloosvel
- Add unit conversion to *DU* to recipes that use *toz* (Pull request #3784) by @schlunma

12.4 v2.11.0

Highlights

- Two new recipes have been added:
  - Recipe *recipe_aod_aeronet_assess.yml* evaluates model aerosol optical depth (AOD) climatologies against ground based observations from the AeroNET measurement network.
  - Recipe *recipe_climate_patterns.yml* generates climate patterns from CMIP6 model datasets.
- The ESACCI-WATERVAPOUR CMORizer now includes daily data and uses the officially released CDR2 data.
- Support for 5 new datasets have been added:
  - AeroNET
  - ANU Climate 2.0 Australian data
  - Australian Gridded Climate Data(AGCD) precipitation
  - NOAA-ERSST
  - NSIDC-G02202-sh sea ice fraction
- NEW TREND: First time release manager shout-outs!
  - This is the first ESMValTool release managed by the Met Office! We want to shout this out - and for all future first time release managers to shout-out - to celebrate the growing, thriving ESMValTool community.

This release includes

12.4.1 Bug fixes

- Recipe_ocean_quadmap: Update ATSR to match ESGF name (Pull request #3443) by @rbeucher
- Fix recipe_bock20jgr_fig_8-10.yml (Pull request #3665) by @LisaBock
- Update the list of datasets used in `recipe_easy_ipcc.yml` (Pull request #3710) by @bouweandela

12.4.2 Documentation

- Improve release tools and documentation (Pull request #3462) by @bouweandela
- Fix a typo in the references file (Pull request #3499) by @bouweandela
- Fix recipe path in `recipe_perfmetrics.rst` (Pull request #3532) by @TomasTorsvik
- Improved description of model evaluation recipes (Pull request #3541) by @schlunma
- Remove double word in cmorizer documentation (Pull request #3553) by @bettina-gier
- Fix Codacy badge (Pull request #3558) by @bouweandela
- Update the release schedule for v2.11.0 (Pull request #3573) by @ehogan
- Improve the formatting of the recipe documentation template (Pull request #3652) by @mo-gill
- Add introduction material on the main documentation page (Pull request #3628) by @bouweandela
- Avoid warning in documentation build (Pull request #3675) by @bouweandela
- Update the list of broken recipes for `v2.11.0` (Pull request #3706) by @ehogan

12.4.3 Diagnostics

- `monitor/multi_dataset.py` improvements: allow data w/o `timerange` and improve text formatting (Pull request #3528) by @schlunma
- Allow datasets without `project` in multi_datasets.py (Pull request #3552) by @schlunma
- Prevent overlapping time axis tick labels in monitoring recipe (Pull request #3682) by @schlunma

12.4.4 New recipe

- Add support for aerosol optical depth climatology metrics to the AutoAssess replacement (Pull request #3048) by @catherinehardacre
- CMIP6 climate patterns (Pull request #2785) by @mo-gregmunday

12.4.5 Observational and re-analysis dataset support

- Add cmorizer scripts for NOAA-ERSST. (Pull request #1799) by @bjoernbroetz
- Update OceanSODA-ETHZ CMORizer with new source file (Pull request #3535) by @TomasTorsvik
- Add CMORizer script for NSIDC-G02202-sh sea ice fraction (Pull request #3512) by @flicj191
- CMORizer Australian Gridded Climate Data(AGCD) precipitation (Pull request #3445) by @flicj191
- Extend CMORizer NCEP-DOE-R2 (Pull request #3469) by @axel-lauer
- Add comment to recipe_lauer13jclim regarding UWisc being superseded by MAC-LWP (Pull request #3537) by @rbeucher
- Recipe_autoassess_landsurface_surfrad: Remove CERES-EBAF version to fix ESGF search (Pull request #3438) by @rbeucher
- Updating ESACCI-WATERVAPOUR cmorizer (Pull request #3282) by @malininae
- CMORiser for ANU Climate 2.0 Australian data (Pull request #3511) by @flicj191
- Add AERONET cmorizer (Pull request #3227) by @zklaus
- Update CRU CMORizer (Pull request #3381) by @lukruh
- Fix recipe_check_obs to be aligned with DKRZ (Pull request #3673) by @LisaBock
- Update AERONET data version (Pull request #3692) by @ehogan

12.4.6 Automatic testing

- Move code into function in batch job generation script (Pull request #3491) by @bouweandela
- Fix sklearn tests (Pull request #3506) by @schlunma
- Pinned sklearn>=1.4.0 (Pull request #3508) by @schlunma
- Update sklearn tests to be compatible with current pytest version (pytest >=8.0.0) (Pull request #3517) by @schlunma
- Update sklearn tests to be compatible with current pytest version (pytest >=8.0.0) Part 2 (Pull request #3518) by @schlunma
- [Circle/CI]Fix *test_installation_from_conda* Circle CI tests (Pull request #3538) by @valeriupredoi
- [Github Actions] install git in OSX and add environment inspection (Pull request #3581) by @valeriupredoi
- [CI Github Actions] Update (outdated) actions versions that produce Node.js warnings (Pull request #3586) by @valeriupredoi

- Fix `flake8==7` linting issues (Pull request #3634) by @valeriupredoi
- Use `importlib` as the import mode for `pytest` (Pull request #3672) by @ehogan

12.4.7 Installation

- Update dependencies (Pull request #3487) by @bouweandela
- Merge v2.10.x into main (Pull request #3489) by @schlunma
- Add imagehash package as an ESMValTool dependency (Pull request #3557) by @alistairsellar
- Unpin `r-akima` (Pull request #3564) by @valeriupredoi
- Adding pys2index dependency (Pull request #3577) by @ljoakim
- Pin esmpy <8.6.0 (Pull request #3585) by @valeriupredoi
- Pin R <4.3.0 (Pull request #3689) by @ehogan
- Pin importlib_metadata <8 (Pull request #3700) by @ehogan
- Pin matplotlib <3.9.0 on ESMValTool release branch (Pull request #3712) by @ehogan

12.4.8 Dependency updates

- Fix for `recipe_seaice_drift.yml`: fix CRS transformer for "North Pole Stereographic" (Pull request #3531) by @flicj191
- Fixed attribute handling in austral_jet/main.ncl for iris>=3.8 (Pull request #3603) by @schlunma
- Fixed attribute handling in emergent constraint diagnostic for iris>=3.8 (Pull request #3605) by @schlunma
- Update the name of the remapcon2 operator in R recipes (Pull request #3611) by @ehogan
- Use `iris.FUTURE.save_split_attrs = True` to remove iris warning in many diagnostics (Pull request #3651) by @schlunma
- Avoid concatenation error in recipe_pcrglobwb.yml (Pull request #3645) by @bouweandela
- Update *scipy.integrate.simps* import (Pull request #3704) by @ehogan

12.4.9 Improvements

- Add native6, OBS6 and RAWOBS rootpaths to metoffice template in config-user-example.yml and remove temporary dir (Pull request #3613) by @alistairsellar

12.5 v2.10.0

Highlights

- Add a realistic IPCC example recipe that reproduces figure 9.3 from AR6. It computes the mean sea-surface temperature anomaly between 1850-2100 over all available CMIP6 models. See the *recipe documentation* or read the blog post for more information.
- Added more plot types to monitoring diagnostic: Hovmoeller Z vs. time, Hovmoeller time vs latlon, variable vs. latitude are now available. See the *recipe documentation* for more information.
- Add support for 4 new datasets:
  - NOAA-CIRES-20CR v3 reanalysis
  - NASA MERRA reanalysis

  - NOAA marine boundary layer data for CH4
  - MOBO-DIC2004-2019

  See *Supported datasets for which a CMORizer script is available* and *Observations* for more information.

- Many recipes now have up-to-date obs4MIPs dataset names so required data can automatically be downloaded from ESGF.

This release includes

12.5.1 Bug fixes

- Update recipe shapeselect to work with shapely v2 (Pull request #3283) @lukruh
- Correctly handle ~ when reading `plot_folder` option of monitoring diagnostic (Pull request #3449) @schlunma
- Fixed provenance tracking for NCL multipanel PNGs (Pull request #3332) @schlunma
- Fixed plot paths in NCL provenance tracking (Pull request #3422) @schlunma
- Fix erroneous file_type handling in certain NCL diagnostics (Pull request #3474) @zklaus
- Fix NCL provenance tracking (Pull request #3477) @schlunma
- Fix plots and provenance in Russell diagnostics (Pull request #3479) @schlunma

12.5.2 Documentation

- Add merge instructions to release instructions (Pull request #3292) @remi-kazeroni
- Update release schedule after release of v2.9.0 (Pull request #3289) @remi-kazeroni
- Add list of failing recipes for v2.9.0 release (Pull request #3294) @remi-kazeroni
- Update `mamba` version in readthedocs configuration docs builds (Pull request #3310) @valeriupredoi
- Add Romain Beucher to citation file as contributor (Pull request #3318) @valeriupredoi
- Removed recipe_carvalhais14nat from list of broken recipes (Pull request #3319) @remi-kazeroni
- Add OBS-maintainers team to documentation on OBS data maintenance and CMORizer reviews (Pull request #3335) @remi-kazeroni
- Add Pauline Bonnet to citation file (Pull request #3347) @Paulinebonnet111
- Ensure compatible zstandard and zstd in readthedocs builds (Pull request #3362) @zklaus
- Fix documentation build (Pull request #3397) @bouweandela
- Minor updates to release tools (Pull request #3216) @bouweandela
- Enhance provenance documentation (Pull request #3305) @alistairsellar
- Re-add communities and grants in zenodo file (Pull request #3416) @valeriupredoi
- Update Anconda badge in README (Pull request #3375, Pull request #3453) @valeriupredoi

12.5.3 Diagnostics

- Slight refactoring of diagnostic script `galytska23/select_variables_for_tigramite.py` for generality and portability (Pull request #3298) @valeriupredoi and @egalytska
- Allow custom variable grouping in diagnostic script `monitor/multi_datasets.py` (Pull request #3343) @schlunma
- Extended monitor diagnostic with plot type variable vs. latitude (Pull request #3340) @ellensarauer

- Add Hovmoeller Z vs. time plot to monitoring diagnostic (Pull request #3345) @cubeme and @helgehr
- Adding Hovmoeller time vs latlon plots to monitoring recipes (Pull request #3341) @lukruh and @jeremykraftdlr
- Implied heat transport new diagnostic (Pull request #3177) @mo-abodas
- Recipe changes for new statistics preprocessors (percentiles) (Pull request #3351) @schlunma
- Add a realistic example recipe (Pull request #3356) @Peter9191 and @bouweandela
- Support `CenteredNorm` in diagnostic monitor/multidatasets.py (Pull request #3415) @schlunma
- Use new preprocessor statistics calling convention for recipe_easy_ipcc.yml (Pull request #3418) @bouweandela
- Adapt to changed style scheme name in matplotlib (Pull request #3475) @zklaus
- Add version to dataset in python example recipe to avoid "Unknown file format" issue on JASMIN (Pull request #3322) @ehogan
- Add the dataset version in the heatwaves_coldwaves recipe to avoid the "Unknown file format" issue on JASMIN (Pull request #3373) @ehogan

12.5.4 Observational and re-analysis dataset support

- Cmorizer for NOAA-CIRES-20CR v3 reanalysis (clt, clwvi, hus, prw, rlut, rlutcs, rsut, rsutcs) (Pull request #3137) @LisaBock
- CMORizer for NASA MERRA reanalysis (Pull request #3039) @axel-lauer
- Download and formatting of NOAA marine boundary layer data for CH4 (NOAA-MBL-CH4) (Pull request #3301) @FranziskaWinterstein
- Added CMORizer for MOBO-DIC2004-2019 (Pull request #3297) @schlunma
- Update obs4MIPs dataset names in quantilebias recipe (Pull request #3330) @rbeucher
- Update obs4MIPs dataset names in Schlund20esd recipe (Pull request #3329) @rbeucher
- Update obs4MIPs dataset names in flatoipcc recipes (Pull request #3328) @rbeucher
- Update obs4mips dataset names in clouds recipes (Pull request #3326) @rbeucher
- Update Obs4MIPs dataset names in ECS recipes (Pull request #3327) @rbeucher
- Update obs4mips dataset names in Bock et al recipes (Pull request #3324, Pull request #3389 and Pull request #3473) @rbeucher and @bouweandela
- Update obs4mips dataset names in radiation budget recipe (Pull request #3323) @rbeucher
- Update Obs4MIPs dataset names in perfmetrics CMIP5 recipe (Pull request #3325) @rbeucher

12.5.5 Automatic testing

- Made sklearn test backwards-compatible with sklearn < 1.3 (Pull request #3285) @schlunma
- Update conda lock creation Github Action workflow and ship updated conda-lock file (Pull request #3307, Pull request #3407) @valeriupredoi
- Compress all bash shell setters into one default option per GitHub Action workflow (Pull request #3315) @valeriupredoi
- Remove deprecated option `offline` from CI configuration (Pull request #3367) @schlunma

12.5.6 Installation

- Use ESMValCore v2.10 (Pull request #3486) @bouweandela

12.5.7 Improvements

- Merge v2.9.x into main (Pull request #3286) @schlunma
- Allow NCL unit conversion *kg s-1 -> GtC y-1* (Pull request #3300) @schlunma

12.6 v2.9.0

12.6.1 Highlights

- A new *diagnostic* has been added to provide a high-level interface to seaborn, a Python data visualization library based on matplotlib. See the *recipe documentation* for more information.
- We have included a new recipe and diagnostic that represent the major physical processes that describe Arctic-midlatitude teleconnections and provide the basis for the CMIP6 model evaluation for the further application of causal discovery. The results are discussed in the article "Causal model evaluation of Arctic-midlatitude teleconnections in CMIP6" by Galytska et al. (in review in Journal of Geophysical Research: Atmospheres).
- It is now possible to use the Dask distributed scheduler, which can significantly reduce the run-time of recipes. Configuration examples and advice are available in the ESMValCore documentation. If configured, the Dask distributed scheduler will also be used by diagnostic scripts written in Python, so make sure to use lazy data wherever it is possible in your (new) diagnostics. More work on improving the computational performance is planned, so please share your experiences, good and bad, with this new feature in ESMValGroup/ESMValCore#1763.

This release includes

12.6.2 Bug fixes

- Fixed usage of `work_dir` in some CMORizer scripts (Pull request #3192) @remi-kazeroni
- Realize data for scalar cube in *recipe_carvalhais14nat* to avert issue from dask latest (2023.6.0) (Pull request #3265) @valeriupredoi
- Fix failing `mlr` diagnostic test by adding new scikit-learn default tag (Pull request #3273) @remi-kazeroni
- Fix ordering of models in perfmetrics diagnostic script (Pull request #3275) @LisaBock

12.6.3 Documentation

- Update release schedule after v2.8.0 (Pull request #3138) @remi-kazeroni
- Added reference entry for Winterstein (Pull request #3154) @FranziskaWinterstein
- Show logo on PyPI (Pull request #3185) @valeriupredoi
- Add Release Managers for v2.9.0 and v2.10.0 (Pull request #3184) @remi-kazeroni
- Fix readthedocs build with esmpy>=8.4.0 and missing ESMFMKFILE variable (Pull request #3205) @valeriupredoi
- Add ESMValCore release v2.8.1 into the documentation (Pull request #3235) @remi-kazeroni
- Modified links to the tutorial (Pull request #3236) @remi-kazeroni
- Fix gitter badge in README (Pull request #3258) @remi-kazeroni
- Add release notes for v2.9.0 (Pull request #3266) @bouweandela

12.6.4 Diagnostics

- New plot_type 1d_profile in monitor (Pull request #3178) @FranziskaWinterstein
- Add Seaborn diagnostic (Pull request #3155) @schlunma
- New recipe and diagnostic for Arctic-midlatitude research (Pull request #3021) @egalytska
- Generate climatology on the fly for AutoAssess soil moisture (Pull request #3197) @alistairsellar
- Remove "fx_variables" from recipe_tebaldi21esd.yml (Pull request #3211) @hb326
- Remove "fx_variables" from ipccwg1ar5ch9 recipes (Pull request #3215) @katjaweigel
- Remove "fx_variables" from recipe_wenzel14jgr.yml (Pull request #3212) @hb326
- Update obs4MIPs dataset to the current naming scheme in recipe_smpi.yml (Pull request #2991) @bouweandela
- Fixed pandas diagnostics for pandas>=2.0.0 (Pull request #3209) @schlunma
- Update recipe_impact.yml to work with newer versions of *pandas* (Pull request #3220) @bouweandela
- Add variable long names to provenance record in monitoring diagnostics (Pull request #3222) @bsolino

12.6.5 Observational and re-analysis dataset support

- Add CMORizer for GPCP-SG (pr) (Pull request #3150) @FranziskaWinterstein
- Extension of NASA MERRA2 CMORizer (cl, cli, clivi, clw, clwvi) (Pull request #3167) @axel-lauer

12.6.6 Automatic testing

- Add a CircleCI-testing-specific `recipe_python_for_CI.yml` to avoid calling geolocator/Nominatim over CI (Pull request #3159) @valeriupredoi
- Check if Python minor version changed after Julia install in development installation test (Pull request #3213) @valeriupredoi
- Fix tests using deprecated `esmvalcore._config` module that has been removed in ESMValCore v2.9 (Pull request #3204) @valeriupredoi

12.6.7 Installation

- Add support for Python=3.11 (Pull request #3173) @valeriupredoi
- Drop python=3.8 support (Pull request #3193) @valeriupredoi
- Repair generation of conda lock files (Pull request #3148) @valeriupredoi
- Modernize lock creation script and repair lock generation (Pull request #3174) @valeriupredoi
- Pin numpy !=1.24.3 due to severe masking bug (Pull request #3182) @valeriupredoi
- Update xesmf to versions >= 0.4.0 (Pull request #2728) @zklaus
- Update esmpy import for ESMF version 8.4.0 or larger (Pull request #3188) @valeriupredoi
- Relax the pin on iris to allow the use of older versions for performance reasons (Pull request #3270) @bouweandela
- Use ESMValCore v2.9.0 (Pull request #3274) @bouweandela

12.6.8 Improvements

- Update pre-commit hooks (Pull request #3189) @bouweandela
- Add support for using a dask distributed scheduler (Pull request #3151) @bouweandela

12.7 v2.8.0

12.7.1 Highlights

- This release includes the diagnostics for reproducing figures 3.9, 3.19, 3.42 and 3.43 of the IPCC AR6 WG1 report. See *recipe documentation* about added recipes.
- A new set of recipes and diagnostics has been included to evaluate cloud climatologies from CMIP models as used in Lauer et al. (2023), J. Climate. See *recipe documentation* about added recipes.
- Addition of a set of recipes for extreme events, regional and impact evaluation as used in Weigel et al. (2021), GMD and in IPCC AR5. See *recipe documentation* about added recipes.

Highlights from ESMValCore v2.8.0 here:

- ESMValCore now supports wildcards in recipes and offers improved support for ancillary variables and dataset versioning.
- Support for CORDEX datasets in a rotated pole coordinate system has been added.
- Native ICON output is now made UGRID-compliant on-the-fly.
- The Python API has been extended with the addition of three modules: `esmvalcore.config`, `esmvalcore.dataset`, and `esmvalcore.local`
- The preprocessor `multi_model_statistics()` has been extended to support more use-cases.

This release includes:

12.7.2 Backwards incompatible changes

Please read the descriptions of the linked pull requests for detailed upgrade instructions.

- Deprecated features scheduled for removal in v2.8.0 or earlier have now been removed (Pull request #2941) @schlunma. Removed `esmvaltool.iris_helpers.var_name_constraint` (has been deprecated in v2.6.0; please use `iris.NameConstraint` with the keyword argument `var_name` instead). Removed *write_netcdf* and *write_plots* from *recipe_filer.py*.
- No files from the `native6` project will be found if a non-existent version of a dataset is specified (#3041) @remi-kazeroni. The tool now searches for exact `version` of `native6` datasets. Therefore, it is necessary to make sure that the version number in the directory tree matches with the version number in the recipe to find the files.
- The conversion of precipitation units from monitoring diagnostic is now done at the preprocessor stage (#3049) @schlunma. To use the unit conversion for precipitation in the new version of this diagnostic, add it as a preprocessor for the precipitation dataset to the recipe.

12.7.3 Bug fixes

- Fix for provenance records from *seaice_tsline.ncl* (Pull request #2938) @axel-lauer
- Fix in *validation.py* for resolving datasets with identical names by using distinct aliases (Pull request #2955) @FranziskaWinterstein
- Bugfix: masking of non-significant differences in *zonal.ncl* (perfmetrics) (Pull request #2957) @axel-lauer
- Fix typo in *perfmetrics/main.ncl* to add tropopause (Pull request #2966) @FranziskaWinterstein

- Fix .png bug in *wenzel16nat* diagnostics (Pull request #2976) @axel-lauer
- *Recipe_ocean_Landschuetzer2016*: Fix typo in filename to run model vs OBS diagnostics (Pull request #2997) @TomasTorsvik
- Fix read_cmor in NCL utilities (Pull request #3007) @axel-lauer
- Removed usages of deprecated features that cause diagnostic crashes (Pull request #3009) @schlunma
- Replace removed *matplotlib.pyplot.savefig* option *additional_artists* (Pull request #3075) @schlunma
- Added missing comma to *sommer17joss.bibtex* (Pull request #3078) @schlunma
- Fix call of output_type in *aux_plotting.ncl* (Pull request #3083) @LisaBock
- Remove colorbar from *bbox_extra_artists* (Pull request #3087) @schlunma
- Fix *MPI-ESM1-2-HR* entries in *recipe_tebaldi21esd* (Pull request #3093) @remi-kazeroni
- Fix bug in provenance writing of *perfmetrics* recipes v2.8.0 (Pull request #3098) @axel-lauer
- Fix *recipe_sea_surface_salinity* for v2.8 (Pull request #3102) @sloosvel
- Fix variable *short_name* and metadata for ESACCI-LST CMORizer (Pull request #3104) @remi-kazeroni
- Fix *recipe_carvalhais14*: replace outline patch with splines (Pull request #3111) @valeriupredoi
- Replace deprecated function *cm.register_cmap* with *mpl.colormaps.register* for *recipe_ arctic_ocean* (Pull request #3112) @TomasTorsvik
- Fix *recipe_extract_shape.yml* (lacking caption for provenance) (Pull request #3126) @valeriupredoi

12.7.4 Community

- Update documentation on pre-installed versions on HPC clusters (Pull request #2934) @remi-kazeroni

12.7.5 Deprecations

- Remove radiation recipes that have been superseded by *recipe_radiation_budget* along with associated diagnostic scripts (#3115) @alistairsellar

12.7.6 Documentation

- Backward compatibility policy (Pull request #2879) @alistairsellar
- Suppress installing and reinstalling dependencies with pip during readthedocs builds (Pull request #2913) @valeriupredoi
- Update installation instructions (Pull request #2939) @bouweandela
- Update documentation for *recipe_extreme_index* (Pull request #2951) @katjaweigel
- Update documentation and *recipe_check_obs* (ERA5) (Pull request #2952) @axel-lauer
- Updated ICON dataset entry in documentation (Pull request #2954) @schlunma
- Add Franziska Winterstein as collaborator in CITATION file (Pull request #3001) @valeriupredoi
- Update release schedule for v2.7.0 and v2.8.0 (Pull request #3010) @remi-kazeroni
- Add ESMValCore Bugfix release v2.7.1 to the release overview table (Pull request #3028) @valeriupredoi
- Detailed instructions for release procedure: running recipes and analyzing the output (Pull request #3032) @valeriupredoi
- Link backward compatibility policy to top level of ESMValCore changelog (Pull request #3052) @alistairsellar

- Update release instructions (Pull request #3066) @remi-kazeroni
- Updated docs and tests regarding new *search_esgf* option (Pull request #3069) @schlunma
- Update script to draft release notes (Pull request #3070) @remi-kazeroni
- Synchronize documentation table of contents with ESMValCore (Pull request #3073) @bouweandela
- Update environment handling in release documentation (Pull request #3096) @remi-kazeroni
- Clarify use (or not) of Jasmin climatology files by soil moisture & permafrost recipes (Pull request #3103) @alistairsellar
- Add link to recipe portal in the gallery page (Pull request #3113) @remi-kazeroni
- Improve stratosphere documentation (Pull request #3114) @alistairsellar
- Added note to documentation that not all datasets used in *schlund20jgr* recipes are available on ESGF (Pull request #3121) @schlunma
- Draft changelog for *v2.8.0* (Pull request #3124) @remi-kazeroni
- Documenting broken recipes after recipe testing for releases (Pull request #3129) @remi-kazeroni
- Increase ESMValTool version to 2.8.0 and update release dates (Pull request #3136) @remi-kazeroni

12.7.7 Diagnostics

- Cloud diagnostics for Lauer et al. (2023) (Pull request #2750) @axel-lauer
- Splitting of *flato13ipcc.yml* into separate recipes and adding recipes for regional Figures (Pull request #2156) @katjaweigel
- Adding IPCC AR6 Chapter 3 Figure 3.43 - Pattern Correlation (Pull request #2772) @LisaBock
- Adding IPCC AR6 Chapter 3 Fig. 3.42 - Perfmetrics (Pull request #2856) @LisaBock
- Comment missing datasets and remove deprecated argument in *recipe_climate_change_hotspot* (Pull request #2920) @sloosvel
- Add plot type *annual_cycle* to multi-dataset monitoring diagnostic (Pull request #2922) @schlunma
- Adding IPCC AR6 Chapter 3 Fig. 3.19 - Speed-Up Of Zonal Mean Wind (Pull request #2984) @LisaBock
- Adding IPCC AR6 Chapter 3 Fig. 3.9 - Attribution (Pull request #2986) @LisaBock
- Obs4mips CERES-EBAF: update version to latest available through esgf in *recipe_validation.yml* (Pull request #3002) @valeriupredoi
- Improve flexibility of cloud diagnostics (Pull request #3016) @axel-lauer
- Let *recipe_impact.yml* write a CSV file that can directly be used in C4I portal (Pull request #2258) @Peter9192
- Fix version numbers of native6 datasets in recipes (#3041) @remi-kazeroni
- Removed automatic conversion of precipitation units from monitoring diagnostic (#3049) @schlunma.
- Updated recipes for ESMValCore v2.8 (Pull request #3064) @schlunma
- Fix *cos22esd* for release of 2.8 (Pull request #3097) @sloosvel
- Diagnostic for *recipe_autoassess_stratosphere.yml*: remove unused feature incompatible with Matplotlib=3.7.1 (Pull request #3089) @valeriupredoi
- Fix numpy deprecation in *hype* diagnostic (Pull request #3101) @Peter9192
- Remove superseded radiation recipes (#3115) @alistairsellar
- Removed *fx_variables* in *recipe_mpqb_xch4* and *recipe_lauer22jclim_fig8* (Pull request #3117) @axel-lauer

- Update Python example recipe (Pull request #3119) @bouweandela
- Updated figure settings to account for newer matplotlib version (Pull request #3133) @katjaweigel

12.7.8 Observational and re-analysis dataset support

- Earth System Data Cube (ESDC) cmorizer (Pull request #2799) @bsolino
- Added CMORizer for Landschützer2020 (spco2) (Pull request #2908) @schlunma
- Added CMORizer for MOBO-DIC_MPIM (dissic) (Pull request #2909) @schlunma
- Added CMORizer for OceanSODA-ETHZ (areacello, co3os, dissicos, fgco2, phos, spco2, talkos) (Pull request #2915) @schlunma
- Extension of ERA-Interim CMORizer (cl, cli, clw, lwp, rlut, rlutcs, rsut, rsutcs) (Pull request #2923) @axel-lauer
- Add JRA-25 cmorizer (clt, hus, prw, rlut, rlutcs, rsut, rsutcs) (Pull request #2927) @LisaBock
- New CMORizers for datasets from the NCEP family (NCEP-DOE-R2, NCEP-NCAR-R1, NOAA-CIRES-20CR) (Pull request #2931) @hb326
- Updates to the recipes that use the NCEP reanalysis dataset (Pull request #2932) @hb326
- MERRA2 cmorizer convert vertical level coordinate units from hPa to Pa (Pull request #3003) @valeriupredoi
- MERRA2 cmorizer set UNLIMITED time coordinate (Pull request #3006) @valeriupredoi
- Added CMORizers for TCOM-CH4 (CH4) and TCOM-N2O (N2O) (Pull request #3014) @schlunma
- Update HadISST cmorizer to include recent years (Pull request #3027) @remi-kazeroni

12.7.9 Automatic testing

- Add DKRZ/Levante batch scripts for release recipe running (Pull request #2883) @valeriupredoi
- Remove *pytest-flake8* and call the use of *flake8* straight (Pull request #2904) @valeriupredoi
- Unpin *flake8* (Pull request #2937) @valeriupredoi
- Fix failing tests that use deprecated feature of *sklearn* (Pull request #2961) @schlunma
- Fix recipe loading tests for esmvalcore before and after version 2.8 (Pull request #3020) @valeriupredoi
- Update recipe load test for v2.8 (Pull request #3040) @bouweandela
- Test running recipes with the development version of ESMValCore (Pull request #3072) @bouweandela
- Fix *test_naming.py* so it doesn't let through directories that need be ignored (Pull request #3082) @valeriupredoi
- Conda environment files for interim use of *esmvalcore=2.8.0rc1* (Pull request #3090) @valeriupredoi
- Move *flake8* check to a step separate from installation on CircleCI (Pull request #3105) @bouweandela
- Recreate conda lock file to harpoon esmvalcore=2.8.0rc1 (Pull request #3108) @valeriupredoi
- Update batch script generation to run all recipes in one command (Pull request #3130) @remi-kazeroni

12.7.10 Installation

- Merge release branch *release_270stable* in main so we pick up unsquashed commits and set the correct version 2.7.0 for main (and up version in CITATION.cff) (Pull request #2896) @valeriupredoi
- Unpin *NetCDF4* (Pull request #2929) @valeriupredoi
- Unpin *cf-units* (Pull request #2930) @bouweandela

- Set the version number on the development branches to one minor version more than the last release (Pull request #2964) @bouweandela
- Pin *shapely<2.0.0* for linux64 (Pull request #2970) @valeriupredoi
- Unpin *matplotlib* (Pull request #3068) @valeriupredoi
- Add *packaging* as direct dependency to ESMValTool (Pull request #3099) @valeriupredoi
- Re-pin sphinx to latest (6.1.3) and add nbsphinx to the environment (Pull request #3118) @valeriupredoi
- Conda environment files for esmvalcore=2.8.0rc2 (Pull request #3120) @remi-kazeroni
- Remove rc (release candidates) conda channel and re-pin esmvalcore to new stable 2.8 (Pull request #3131) @valeriupredoi

12.7.11 Improvements

- Read *config-user.yml* using *esmvalcore.config* module (Pull request #2736) @bouweandela
- Make results of recipes *schlund20jgr_*.yml* deterministic (Pull request #2900) @schlunma
- *Recipe_gier2020bg.yml*: add sorting to SA barplot (Pull request #2905) @bettina-gier
- Add the outline of a climatological tropopause to the zonalmean_profile plots (Pull request #2947) @FranziskaWinterstein
- Update data finder imports (Pull request #2958) @bouweandela
- Add support for the upcoming ESMValCore v2.8 release to the recipe filler tool (Pull request #2995) @bouweandela
- Updated monitoring diagnostics with netCDF output and additional logging (Pull request #3029) @schlunma
- Use aliases in perfmetrics (Pull request #3058) @FranziskaWinterstein

12.8 v2.7.0

12.8.1 Highlights

- This release has seen the inclusion of the code for figures 3.3, 3.4, 3.5, 3,13 and 3.15 of the IPCC AR6 WG1 report, see them in the new documentation
- We have also included new diagnostics and recipe necessary to produce the plots and tables for the journal article "Climate model projections from the Scenario Model Intercomparison Project (ScenarioMIP) of CMIP6" by Tebaldi et al. in ESD 2020-68 from 2021; also see the recipe entry
- We have also extended the support for MERRA2 observational dataset, by adding support for a large number of variables, including 3D variables, see the table of supported obs datasets

12.8.2 Backwards incompatible changes

- Remove installation of R dependencies from the help message (Pull request #2761) @remi-kazeroni

12.8.3 Bug fixes

- Fix misplaced provenance records from IPCC AR5 Ch.12 diags (Pull request #2758) @axel-lauer
- Fix *esmvaltool.utils.testing.regression.compare* module to run with Python<3.10 too (Pull request #2778) @valeriupredoi
- Fixed small bug that could lead to wrong pr units in *monitor/multi_datasets.py* (Pull request #2788) @schlunma

- Pin *xgboost>1.6.1* so we avert documentation failing to build with *1.6.1* (Pull request #2780) @valeriupredoi
- Pin *matplotlib-base<3.6.0* to avoid conflict from *mapgenerator* that fails doc builds (Pull request #2830) @valeriupredoi
- Fixed wrong latitudes in NDP CMORizer (Pull request #2832) @schlunma
- Fix indexer in Autoassess supermeans module use a tuple of *(slice(), idx, idx)* (Pull request #2838) @valeriupredoi
- Replace xarray ufuncs with bogstandard numpy in weighting/climwip/calibrate_sigmas.py (Pull request #2848) @valeriupredoi
- Fix units MERRA2 CMORizer (Pull request #2850) @axel-lauer
- Fix bug when using log-scale y-axis for ocean transects. (Pull request #2862) @TomasTorsvik

12.8.4 Community

- Add MO-paths to config file (Pull request #2784) mo-tgeddes

12.8.5 Deprecations

- Recipe *recipe_esacci_oc.yml* replace with new regrid scheme *nearest_extrapolate* (Pull request #2841) @valeriupredoi

12.8.6 Documentation

- Update release schedule for v2.7 (Pull request #2747) @bouweandela
- Add Met Office installation method (Pull request #2751) mo-tgeddes
- Add release dates for 2023 (Pull request #2769) @remi-kazeroni
- Made *maintainer* entry mandatory for published recipes (Pull request #2703) @schlunma
- Use command with current command line opts for *cffconvert* in documentation (Pull request #2791) @valeriupredoi
- Update CMORizer documentation with command options (Pull request #2795) @remi-kazeroni
- Fixed broken link for monthly meetings (Pull request #2806) @remi-kazeroni
- Update MO obs4MIPs paths in the user configuration file (Pull request #2813) mo-tgeddes
- Fix Windows incompatible file names in documentation of recipe_climate_change_hotspot.yml (Pull request #2823) @ledm
- Update documentation for the Landschuetzer 2016 recipe. (Pull request #2801) @TomasTorsvik
- Fixed anaconda badge in README (Pull request #2866) @valeriupredoi
- Update release strategy notes (Pull request #2734) @sloosvel
- Add documentation on how to handle CMORizers for multiple dataset versions (Pull request #2730) @remi-kazeroni
- Extending documentation: recipe maintainer + broken recipe policy (Pull request #2719) @axel-lauer

12.8.7 Diagnostics

- Recipe and diagnostics for : Tebaldi et al.,ESD, 2021 (Pull request #2052) debe-kevin
- Figures for IPCC AR6 WG1 Chapter 3 (Atmosphere) (Pull request #2533) @LisaBock

12.8.8 Observational and re-analysis dataset support

- Update CERES-EBAF to Ed4.1 (Pull request #2752) @axel-lauer
- New CMORizer for CALIPSO-ICECLOUD (Pull request #2753) @axel-lauer
- New CMORizer for CLOUDSAT-L2 (Pull request #2754) @axel-lauer
- Update MERRA2 cmorizer with extra 2D and 3D variables (Pull request #2774) @valeriupredoi

12.8.9 Automatic testing

- Pin *netcdf4 != 1.6.1* since that is spitting large numbers of SegFaults (Pull request #2796) @valeriupredoi

12.8.10 Installation

- Increase esmvalcore version to 2.7.0 in environment files (Pull request #2860) @valeriupredoi
- Add iris-esmf-regrid as a dependency (Pull request #2880) @zklaus

12.8.11 Improvements

- Fix tebaldi21esd (Pull request #2749) @axel-lauer
- Added option to show basic statistics in plots of *monitor/multi_datasets.py* (Pull request #2790) @schlunma
- Remove retracted datasets from *recipe_climate_change_hotspot* (Pull request #2854) @sloosvel

12.9 v2.6.0

12.9.1 Highlights

- A new monitoring diagnostic has been added to allow the comparison of model runs against reference datasets. For details, see *Monitoring diagnostic to show multiple datasets in one plot (incl. biases)*.
- A tool has been developed to compare the output of recipe runs against previous runs, in order to detect in an automated way breaking changes between releases. Find more information in *Comparing recipe runs*.
- The recipe *Climate Change Hotspot* allows to compute hotspots in any rectangular region.

Please also note the highlights from the corresponding ESMValCore release here. Thanks to that ESMValTool has gained the following features:

- A new set of CMOR fixes is now available in order to load native EMAC model output and CMORize it on the fly.
- The version number of ESMValCore is now automatically generated using setuptools_scm, which extracts Python package versions from git metadata.

This release includes

12.9.2 Bug fixes

- Fix dtype for Marrmot recipe results (Pull request #2646) @SarahAlidoost
- Adapt test_fix_coords to new version of cf-units (Pull request #2707) @zklaus
- Fix nested axes in *recipe_martin18_grl* and *recipe_li17natcc* (Pull request #2712) @lukruh
- Update common_climdex_preprocessing_for_plots.R (Pull request #2727) @earnone

12.9.3 Community

- Collecting github user names for config-references (Pull request #2677) @lukruh

12.9.4 Deprecations

- Deprecate the function *esmvaltool.diag_scripts.shared.var_name_constraint*. This function is scheduled for removal in v2.8.0. Please use `iris.NameConstraint` with the keyword argument var_name instead: this is an exact replacement. (Pull request #2655) @schlunma

12.9.5 Documentation

- Documentation Improvements (Pull request #2580) @stacristo
- Fixed broken label in the documentation (Pull request #2616) @remi-kazeroni
- Add readthedocs configuration file (Pull request #2627) @bouweandela
- Update the command for building the documentation (Pull request #2622) @bouweandela
- Added DKRZ-Levante to *config-user-example.yml* (Pull request #2632) @remi-kazeroni
- Improved documentation on native dataset support (Pull request #2635) @schlunma
- Add documentation on building and uploading Docker images (Pull request #2662) @bouweandela
- Remove support for Mistral in *config-user-example.yml* (Pull request #2667) @remi-kazeroni
- Add note to clarify that CORDEX support is work in progress (Pull request #2682) @bouweandela
- Restore accidentally deleted text from input data docs (Pull request #2683) @bouweandela
- Add running settings note in *recipe_wenzel16nat.yml* documentation (Pull request #2692) @sloosvel
- Add a note on transferring permissions to the release manager (Pull request #2688) @bouweandela
- Update documentation on ESMValTool module at DKRZ (Pull request #2696) @remi-kazeroni
- Add note on how to run recipe_wenzel14jgr.yml (Pull request #2717) @sloosvel
- Added conda forge feedstock repo link in README (Pull request #2555) @valeriupredoi

12.9.6 Diagnostics

- Compute bias instead of correlation in *compare_salinity.py* (Pull request #2642) @sloosvel
- Update monitor diagnostics (Pull request #2608) @schlunma
- Add new Psyplot diagnostic (Pull request #2653) @schlunma
- Reduce memory usage of lisflood recipe (Pull request #2634) @sverhoeven
- Provenance in ocean diagnostics (Pull request #2651) @tomaslovato
- Extend monitor diagnostics with multi-dataset plots (Pull request #2657) @schlunma

- Recipe and diagnostics to plot climate change hotspots: Cos et al., ESD 2022 (Pull request #2614) @pepcos
- Update plots of consecutive dry days recipe (Pull request #2671) @bouweandela
- Fix the format of ids in Hype forcing files (Pull request #2679) @SarahAlidoost
- WFlow diagnostic script: remove manual rechunking (Pull request #2680) @Peter9192

12.9.7 Observational and re-analysis dataset support

- Extending the HadCRUT5 cmorizer (Pull request #2509) @LisaBock
- Cmorize Kadow2020 dataset (Pull request #2513) @LisaBock
- Cmorize NOAAGlobalTemp dataset (Pull request #2515) @LisaBock
- Add option to CMORize ts as tos in ESACCI data (Pull request #2731) @sloosvel

12.9.8 Automatic testing

- Add a tool for comparing recipe runs to previous runs (Pull request #2613) @bouweandela
- Ignore NCL interface files when comparing recipe runs (Pull request #2673) @bouweandela
- Add a short version of recipe deangelis15nat for testing (Pull request #2685) @katjaweigel
- Expanded recipe output comparison tool to better handle absolute paths in output (Pull request #2709) @schlunma
- Update development infrastructure (Pull request #2663) @bouweandela

12.9.9 Installation

- Removed *package/meta.yaml* and all references to it (Pull request #2612) @schlunma

12.9.10 Improvements

- Improved handling of weights in MLR diagnostics (Pull request #2625) @schlunma
- Fixed order of variables in perfemetrics plot of Anav13jclim recipe (Pull request #2706) @schlunma
- Added input file sorting to many diagnostic to make output exactly reproducible (Pull request #2710) @schlunma
- Removed 'ancestors' attributes before saving netcdf files in emergent constraints diagnostics (Pull request #2713) @schlunma

12.10 v2.5.0

12.10.1 Highlights

- A new recipe to plot generic preprocessor output is now available. For details, see *Monitor*.
- The CMORization of observational and other datasets has been overhauled. For many datasets, an automatic download script is now available. For details, see *Observations* and *Writing a CMORizer script for an additional dataset*.

Please also note the highlights from the corresponding ESMValCore release here. Thanks to that ESMValTool has gained the following features:

- The new preprocessor `extract_location` can extract arbitrary locations on the Earth.
- Time ranges can now be extracted using the ISO 8601 format.

- The new preprocessor `ensemble_statistics` can calculate arbitrary statistics over all ensemble members of a simulation.

This release includes

12.10.2 Backwards incompatible changes

- Streamline observations download (Pull request #1657) Javier Vegas-Regidor. This change removes the `cmorize_obs` command which has previously been used to CMORize observations and other datasets. The new command `esmvaltool data` provides many new features apart from the CMORization (`esmvaltool data format`), for example, automatic downloading of observational datasets (`esmvaltool data download`). More details on this can be found *here* and *here*.
- Dropped Python 3.7 (Pull request #2585) @schlunma. ESMValTool v2.5.0 dropped support for Python 3.7. From now on Python >=3.8 is required to install ESMValTool. The main reason for this is that conda-forge dropped support for Python 3.7 for OSX and arm64 (more details are given here).

12.10.3 Bug fixes

- Remove the use of *esmvalgroup* channel from the conda install Github Action workflow (Pull request #2420) @valeriupredoi
- Ignore .pymon-journal file in test discovery (Pull request #2491) @zklaus
- Relocate pytest-monitor outputted database *.pymon* so *.pymon-journal* file should not be looked for by *pytest* (Pull request #2501) @valeriupredoi
- Re-establish Python 3.7 compatibility (Pull request #2506) @zklaus
- Update intersphinx mapping (Pull request #2531) @zklaus
- Fixed *KeyError* in *recipe_ocean_bgc.yml* (Pull request #2540) @schlunma
- Corrected ESACCI-SEA-SURFACE-SALINITY from OBS to OBS6 (Pull request #2542) @axel-lauer
- Fixed *recipe_kcs.yml* (Pull request #2541) @schlunma
- Fix MDER diagnostic regression_stepwise (Pull request #2545) @axel-lauer
- Fix for recipe_wenzel16nat (Pull request #2547) @axel-lauer
- Fixed *recipe_carvalhais14nat* and removed deprecated use of np.float (Pull request #2558) @schlunma
- Fix *recipe_wenzel14jgr* (Pull request #2577) @remi-kazeroni
- Fixed various recipes by removing faulty or non-available datasets (Pull request #2563) @schlunma
- Remove missing CMIP5 data from 2 recipes (Pull request #2579) @remi-kazeroni
- Fix *recipe_seaice* (Pull request #2578) @remi-kazeroni
- Fix *recipe_climwip_brunner20esd* (Pull request #2581) @remi-kazeroni

12.10.4 Deprecations

- Remove *–use-feature=2020-resolver* command line option for obsolete pip 2020 solver (Pull request #2493) @valeriupredoi
- Renamed vertical regridding schemes in affected recipes (Pull request #2487) @schlunma

12.10.5 Documentation

- Update release manager for v2.5 (Pull request #2429) @axel-lauer
- Mention ENES Climate Analytics service (Pull request #2438) @bouweandela
- Add recipe overview page (Pull request #2439) @bouweandela
- Fix pointer to Tutorial lesson on preprocessor from 05 to 06 (Pull request #2473) @valeriupredoi
- Removed obsolete option *synda-download* from documentation (Pull request #2485) @schlunma
- Update CMUG XCH4 docu figure (Pull request #2502) @axel-lauer
- Add Python=3.10 to package info, update Circle CI auto install and documentation for Python=3.10 (Pull request #2503) @schlunma
- Unify user configuration file (Pull request #2507) @schlunma
- Synchronized *config-user.yml* with version from ESMValCore (Pull request #2516) @schlunma
- CITATION.cff fix and automatic validation of your citation metadata (Pull request #2517) @abelsiqueira
- Add backwards incompatible changes at the top of the release notes draft (Pull request #2431) @bouweandela
- Fixed intersphinx mapping of *scipy* (Pull request #2523) @schlunma
- Add authors to citation cff (Pull request #2525) @SarahAlidoost
- Update documentation on running a recipe (Pull request #2432) @bouweandela
- Fix recipe *hydrology/recipe_wflow.yml* (Pull request #2549) @remi-kazeroni
- Update *draft_release_notes.py* for new release (Pull request #2553) @schlunma
- Added stand with Ukraine badge (Pull request #2565) @valeriupredoi
- Updated CREM docu (recipe_williams09climdyn.yml) (Pull request #2567) @axel-lauer
- First draft for v2.5.0 changelog (Pull request #2554) @schlunma
- Replace nonfunctional Github Actions badge with cool one in README (Pull request #2582) @valeriupredoi
- Updated changelog (Pull request #2589) @schlunma
- Updated release strategy with current release and upcoming release (Pull request #2597) @schlunma
- Increased ESMValTool version to 2.5.0 (Pull request #2600) @schlunma

12.10.6 Diagnostics

- AutoAssess: Add new diagnostic for radiation budget (Pull request #2282) @Jon-Lillis
- CMUG Sea Surface Salinity dataset and diagnostic (Pull request #1832) Javier Vegas-Regidor
- Recipe with new diagnostics for ESA-CMUG H2O (Pull request #1834) @katjaweigel
- Cleaned Schlund et al. (2020) recipe and fixed small bugs in corresponding diagnostic (Pull request #2484) @schlunma
- Add ESA CCI LST cmorizer and diagnostic (Pull request #1897) @morobking
- XCH4 ESA CMUG diagnostics (subset of the MPQB diagnostics) (Pull request #1960) @hb326
- Add support for ESACCI Ocean Color (Chlorophyll) observations (Pull request #2055) ulrikaw-cloud
- Updated *recipe_zmnam.yml* with hemisphere selection (Pull request #2230) @fserva
- Add recipe and diagnostic scripts to compute figures of D9.4 of ISENES3 (Pull request #2441) @sloosvel

- Save resampled climates from KCS diagnostic local_resampling.py (Pull request #2221) @Emmadd
- Use years from KCS recipe (Pull request #2223) @Emmadd
- Recipe to plot generic output from the preprocessor (Pull request #2184) Javier Vegas-Regidor
- Fixed provenance tracking for emergent constraint diagnostics (Pull request #2573) @schlunma

12.10.7 Observational and re-analysis dataset support

- Ensure dummy data for cmorize_obs_woa test are written to the correct directory (Pull request #2451) @ehogan
- Add ESA CCI LST cmorizer and diagnostic (see previous section *Diagnostics*)

12.10.8 Automatic testing

- Run a nightly Github Actions workflow to monitor tests memory per test (configurable for other metrics too) and lists the slowest 100 tests (Pull request #2449) @valeriupredoi
- Fix individual pytest runs broken due to missing explicit imports from *iris* and adding a couple missing package markers (Pull request #2455) @valeriupredoi
- Add Python=3.10 to Github Actions and switch to Python=3.10 for the Github Action that builds the PyPi package (Pull request #2488) @valeriupredoi
- Switch all github actions from miniconda to mambaforge (Pull request #2498) @zklaus
- Pin *flake8<4* to have actual FLAKE8 error printed if tests fail and not garbage (Pull request #2492) @valeriupredoi
- Implementing conda lock (Pull request #2193) @valeriupredoi
- [Docker] Update Docker container builds with correct installations of Julia (Pull request #2530) @valeriupredoi
- Update Linux condalock file (various pull requests) github-actions[bot]

12.10.9 Installation

- Comment out release candidate channel in environment.yml (Pull request #2417) @zklaus
- Comment out rc channel in osx environment file (Pull request #2421) @valeriupredoi
- Add *python-cdo* as conda-forge dependency in environment files to ensure *cdo* gets used from conda-forge and not pip (Pull request #2469) @valeriupredoi
- Install rasterio from conda-forge and avoid issues from python=3.10 (Pull request #2479) @valeriupredoi
- Updated dependencies with new ESMValCore version (Pull request #2599) @schlunma

12.10.10 Improvements

- Remove use of OBS and use CMIP instead in *examples/recipe_ncl.yml* (Pull request #2494) @valeriupredoi
- Expanded *recipe_preprocessor_test.yml* to account for new *multi_model_statistics* features (Pull request #2519) @schlunma
- Updated piControl periods for recipes that use KACE-1-0-G (Pull request #2537) @schlunma
- Reduced time range in *recipe_globwat.yml* (Pull request #2548) @schlunma
- Removed models with missing data from recipe_williams09climdyn.yml (Pull request #2566) @axel-lauer
- Restored original versions of *recipe_schlund20esd.yml* and *recipe_meehl20sciadv.yml* (Pull request #2583) @schlunma

12.11 v2.4.0

12.11.1 Highlights

- ESMValTool is moving from Conda to Mamba as the preferred installation method. This will speed up the installation and comes with some improvements behind the scenes. Read more about it in *the installation guide*.

Please also note the highlights from the corresponding ESMValCore release here. Thanks to that ESMValTool has gained the following features:

- Download any missing data that is available on the ESGF automatically.
- Resume previous runs, reusing expensive pre-processing results.

This release includes

12.11.2 Bug fixes

- Fixed *recipe_meehl20sciadv.yml* for ESMValCore 2.3 (Pull request #2253) @schlunma
- Fix provenance of NCL figures created using the log_provenance function (Pull request #2279) @bouweandela
- Fix bug in ClimWIP brunner19 recipe when plotting (Pull request #2226) @lukasbrunner
- Pin docutils <0.17 to fix sphinx build with rtd theme (Pull request #2312) @zklaus
- Fix example recipes (Pull request #2338) @valeriupredoi
- Do not add bounds to plev (plev19) in era interim cmorizer (Pull request #2328) @valeriupredoi
- Fix problem with pip 21.3 that prevents installation from source (Pull request #2344) @zklaus
- Add title to recipe embedded in test_diagnostic_run.py (Pull request #2353) @zklaus
- Fix capitalization of obs4MIPs (Pull request #2368) @bouweandela
- Specify that areacella is needed for area statistics in the Python example recipe (Pull request #2371) @bouweandela
- Enabling variable *obs550lt1aer* in recipes (Pull request #2388) @remi-kazeroni
- Update a diagnostic to new Iris version (Pull request #2390) @katjaweigel
- Fixed bug in provenance tracking of ecs_scatter.ncl (Pull request #2391) @schlunma
- Fix provenance issue in pv_capacity_factor.R (Pull request #2392) @katjaweigel
- Remove obsolete write_plots option from R diagnostics (Pull request #2395) @zklaus
- Fix arctic ocean diagnostic (Pull request #2397) @zklaus
- Fix sea ice drift recipe and script (Pull request #2404) @sloosvel
- Adapt diagnostic script to new version of iris (Pull request #2403) @zklaus
- Fix ocean multimap (Pull request #2406) @zklaus
- Fix diagnostic that uses *xarray*: *dtype* correctly set and harmonize *xarray* and *matplotlib* (Pull request #2409) @zklaus
- Deactivate provenance logging for plots in thermodyn toolbox (Pull request #2414) @zklaus

12.11.3 Deprecations

- Removed write_plots and write_netcdf from some NCL diagnostics (Pull request #2293) @schlunma
- Fixed provenance logging of all python diagnostics by removing 'plot_file' entry (Pull request #2296) @schlunma
- Do not deprecate classes Variable, Variables and Datasets on a specific version (Pull request #2286) @schlunma
- Remove obsolete write_netcdf option from ncl diagnostic scripts (Pull request #2387) @zklaus
- Remove write plots from ocean diagnostics (Pull request #2393) @valeriupredoi
- More removals of instances of *write_plots* from Python diagnostics (appears to be the final removal from Py diags) (Pull request #2394) @valeriupredoi

12.11.4 Documentation

- List Manuel Schlund as release manager for v2.5 (Pull request #2268) @bouweandela
- GlobWat fix download links and gdal command (Pull request #2334) @babdollahi
- Add titles to recipes authored by *predoi_valeriu* (Pull request #2333) @valeriupredoi
- Added titles to recipes maintained by lauer_axel (Pull request #2332) @axel-lauer
- Update the documentation of the GRACE CMORizer (Pull request #2349) @remi-kazeroni
- Add titles in BSC recipes (Pull request #2351) @sloosvel
- Update esmvalcore dependency to 2.4.0rc1 (Pull request #2348) @zklaus
- Add titles to recipes maintained by Peter Kalverla (Pull request #2356) @Peter9192
- Adding titles to the recipes with maintainer hb326 (Pull request #2358) @hb326
- Add title for zmnam as for #2354 (Pull request #2363) @fserva
- Added recipe titles the the ocean recipes. (Pull request #2364) @ledm
- Update recipe_thermodyn_diagtool.yml - add title (Pull request #2365) @ValerioLembo
- Fix provenance of figures of several R diagnostics (Pull request #2300) @bouweandela
- Adding titles to Mattia's recipes (Pull request #2367) @remi-kazeroni
- Adding titles to wenzel recipes (Pull request #2366) @hb326
- Fix formatting of some recipe titles merged from PR 2364 (Pull request #2372) @zklaus
- Adding titles to Bjoern's recipes (Pull request #2369) @remi-kazeroni
- Add titles to ocean recipes (maintainer Lovato) (Pull request #2375) @tomaslovato
- Add titles for three c3s-magic recipes (Pull request #2378) @zklaus
- Add title for recipe maintained by Ruth Lorenz (Pull request #2379) @zklaus
- Fix toymodel recipe (Pull request #2381) Javier Vegas-Regidor
- Added titles for recipes of maintainer *schlund_manuel* (Pull request #2377) @schlunma
- Write_plots and titles for deangelis15nat, li17natcc, martin18grl, pv_capacity_factor (Pull request #2382) @katjaweigel
- Add titles for some recipes (Pull request #2383) @zklaus
- Adding titles for recipes by von Hardenberg and Arnone (Pull request #2384) @zklaus
- Last two missing titles (Pull request #2386) @valeriupredoi

- Update documentation on downloading data (Pull request #2370) @bouweandela
- Fix installation instructions for Julia (Pull request #2335) @zklaus
- Fix provenance of Julia example diagnostic (Pull request #2289) @bouweandela
- Added notes on use of mamba in the installation documentation chapter (Pull request #2236) @valeriupredoi
- Update version number for 2.4.0 release (Pull request #2410) @zklaus
- Update release schedule for 2.4.0 (Pull request #2412) @zklaus
- Update changelog for 2.4.0 release (Pull request #2411) @zklaus

12.11.5 Diagnostics

- Add all available CMIP5 and CMIP6 models to recipe_impact.yml (Pull request #2251) @bouweandela
- Add Fig. 6, 7 and 9 of Bock20jgr (Pull request #2252) @LisaBock
- Generalize *recipe_validation** diagnostic to work with identical control and experiment dataset names (Pull request #2284) @valeriupredoi
- Add missing preprocessor to recipe_gier2020bg and adapt to available data (Pull request #2399) @bettina-gier
- Removed custom version of *AtmosphereSigmaFactory* in diagnostics (Pull request #2405) @schlunma

12.11.6 Observational and re-analysis dataset support

- Replace recipe_era5.yml with recipe_daily_era5.yml (Pull request #2182) @SarahAlidoost
- Update WOA cmorizer for WOA18 and WOA13v2 (Pull request #1812) @LisaBock
- GLODAP v2.2016 ocean data cmorizer (Pull request #2185) @tomaslovato
- Updated GCP CMORizer (Pull request #2295) @schlunma

12.11.7 Automatic testing

- Add a cylc suite to run all recipes (Pull request #2219) @bouweandela
- Retire test with Python 3.6 from full development Github Actions test (Pull request #2229) @valeriupredoi
- Remove Python 3.6 tests from GitHub Actions (Pull request #2264) @valeriupredoi
- Unpin upper bound for iris (previously was at <3.0.4) (Pull request #2266) @valeriupredoi
- Pin latest esmvalcore to allow use of the bugfix release 2.3.1 always (Pull request #2269) @valeriupredoi
- Add apt update so Julia gets found and installed by Docker (Pull request #2290) @valeriupredoi
- Use mamba for environment update and creation in the Docker container build on DockerHub (Pull request #2297) @valeriupredoi
- Docker container experimental - run a full env solve with mamba instead of a conda update (Pull request #2306) @valeriupredoi
- Full use of mamba in Github Actions source install test and use generic Python 3.7 (removing the very specific 3.7.10) (Pull request #2287) @valeriupredoi
- Replace use of conda with mamba for conda_install test on Circle CI (Pull request #2237) @valeriupredoi
- Update circleci configuration (Pull request #2357) @zklaus

12.11.8 Installation

- Remove *mpich* from conda dependencies list (Pull request #2343) @valeriupredoi

12.11.9 Improvements

- Add script for extracting a list of input files from the provenance (Pull request #2278) @bouweandela
- Update github actions (Pull request #2360) @zklaus
- Removed 'write_plots' from all NCL diagnostics (Pull request #2331) @axel-lauer
- Update and modernize *config-user-example.yml* (Pull request #2374) @valeriupredoi

12.12 v2.3.0

This release includes

12.12.1 Bug fixes

- Indent block to pick up and raise exception if cmorizer data not found (TierX dir is not there) (Pull request #1877) @valeriupredoi
- Skip recipe filler tests until we have a new release since GA tests are failing (Pull request #2089) @valeriupredoi
- Fixed broken link to contributions in README (Pull request #2102) @schlunma
- Fix recipe filler for the case the variable doesn't contain short_name (Pull request #2104) @valeriupredoi
- Add fix for iris longitude bug to ClimWIP (Pull request #2107) @lukasbrunner
- Update for outdated link to reference Déandreis et al. (2014). (Pull request #2076) @katjaweigel
- Fixed recipes for ESMValCore 2.3.0 (Pull request #2203) @schlunma
- Fix the WFDE5 cmorizer (Pull request #2211) @remi-kazeroni
- Fix broken CMORizer log message if no Tier directory exists (Pull request #2207) @jmrgonza
- Fix bug in ClimWIP basic test recipe when plotting (Pull request #2225) @lukasbrunner
- Fix bug in ClimWIP advanced test recipe when plotting (Pull request #2227) @lukasbrunner
- Adjust time range for the *WDFE5* dataset in the *recipe_check_obs.yml* (Pull request #2232) @remi-kazeroni
- Fix plot and provenance of recipe_consecdrydays (Pull request #2244) @bouweandela

12.12.2 Documentation

- Improving the README.md file with a more appealing look and bit more info (Pull request #2065) @valeriupredoi
- Update plot title martin18grl (Pull request #2080) @katjaweigel
- Update contribution guidelines (Pull request #2031) @bouweandela
- Update links in pull request template to point to latest documentation (Pull request #2083) @bouweandela
- Update release schedule (Pull request #2081) @bouweandela
- Updates to contribution guidelines (Pull request #2092) @bouweandela
- Update documentation for ERA5 with new variables (Pull request #2111) @lukasbrunner
- Add OSX installation instructions to docs (Pull request #2115) @bvreede

- Instructions to use pre-installed versions on HPC clusters (Pull request #2197) @remi-kazeroni
- Add functional Autoassess diagnostics: land surface metrics: permafrost, soil moisture, surface radiation (Pull request #2170) @valeriupredoi
- Add citation info in *recipe_eady_growth_rate.yml* (Pull request #2188) @sloosvel
- Update version number to 2.3.0 (Pull request #2213) @zklaus
- Update release schedule for 2.3.0 (Pull request #2247) @zklaus
- Changelog update to v2.3.0 (Pull request #2214) @zklaus

12.12.3 Diagnostics

- Added figures 8 and 10 to recipe_bock20jgr.yml (Pull request #2074) @schlunma
- Add hydrological forcing comparison recipe (Pull request #2013) @stefsmeets
- Added recipe for Meehl et al., Sci. Adv. (2020) (Pull request #2094) @schlunma
- Add GlobWat recipe and diagnostic (Pull request #1808) @babdollahi
- Add ClimWIP recipe to reproduce Brunner et al. 2019 (Pull request #2109) @lukasbrunner
- Update Climwip recipe to reproduce brunner2020esd (Pull request #1859) @ruthlorenz
- Update recipe_thermodyn_diagtool.yml: code improvements and more user options (Pull request #1391) @ValerioLembo
- Remove model AWI-CM-1-1-MR from recipe_impact.yml (Pull request #2238) @bouweandela
- PV capacity factor for ESMValTool GMD paper (Pull request #2153) @katjaweigel

12.12.4 Observational and re-analysis dataset support

- Cmorize wfde5 (Pull request #1991) @mwjury
- Make cmorizer utils funcs public in utilities.py and add some numpy style docstrings (Pull request #2206) @valeriupredoi
- CMORizer for CLARA-AVHRR cloud data (Pull request #2101) @axel-lauer
- Update of ESACCI-CLOUD CMORizer (Pull request #2144) @axel-lauer

12.12.5 Automatic testing

- Force latest Python in empty environment in conda install CI test (Pull request #2069) @valeriupredoi
- Removed imports from private sklearn modules and improved test coverage of custom_sklearn.py (Pull request #2078) @schlunma
- Move private _(global)_stock_cube from esmvacore.preprocessor._regrid to cmorizer (Pull request #2087) @valeriupredoi
- Try mamba install esmvaltool (Pull request #2125) @valeriupredoi
- Reinstate OSX Github Action tests (Pull request #2110) @valeriupredoi
- Pin mpich to avoid default install of 3.4.1 and 3.4.2 with external_0 builds (Pull request #2220) @valeriupredoi
- Include test sources in distribution (Pull request #2234) @zklaus
- Pin *iris<3.0.4* to ensure we still (sort of) support Python 3.6 (Pull request #2246) @valeriupredoi

12.12.6 Installation

- Fix conda build by skipping documentation test (Pull request #2058) Javier Vegas-Regidor
- Update pin on esmvalcore pick up esmvalcore=2.3.0 (Pull request #2200) @valeriupredoi
- Pin Python to 3.9 for development installation (Pull request #2208) @bouweandela

12.12.7 Improvements

- Add EUCP and IS-ENES3 projects to config-references (Pull request #2066) @Peter9192
- Fix flake8 tests on CircleCI (Pull request #2070) @bouweandela
- Added recipe filler. (Pull request #1707) @ledm
- Update use of fx vars to new syntax (Pull request #2145) @sloosvel
- Add recipe for climate impact research (Pull request #2072) @Peter9192
- Update references "master" to "main" (Pull request #2172) @axel-lauer
- Force git to ignore VSCode workspace files (Pull request #2186) Javier Vegas-Regidor
- Update to new ESMValTool logo (Pull request #2168) @axel-lauer
- Python cmorizers for CDR1 and CDR2 ESACCI H2O (TCWV=prw) data. (Pull request #2152) @katjaweigel
- Remove obsolete conda package (closes #2100) (Pull request #2103) @zklaus

12.13 v2.2.0

12.13.1 Highlights

ESMValTool is now using the recently released Iris 3. We acknowledge that this change may impact your work, as Iris 3 introduces several changes that are not backward-compatible, but we think that moving forward is the best decision for the tool in the long term.

This release includes

12.13.2 Bug fixes

- Bugfix: time weights in time_operations (Pull request #1956) @axel-lauer
- Fix issues with bibtex references (Pull request #1955) @stefsmeets
- Fix ImportError for *configure_logging* (Pull request #1976) @stefsmeets
- Add required functional parameters for extract time in recipe_er5.yml (Pull request #1978) @valeriupredoi
- Revert "Fix ImportError for *configure_logging*" (Pull request #1992) @bouweandela
- Fix import of esmvalcore _logging module in cmorize_obs.py (Pull request #2020) @valeriupredoi
- Fix logging import in cmorize_obs again since last merge was nulled by pre-commit hooks (Pull request #2022) @valeriupredoi
- Refactor the functions in derive_evspsblpot due to new iris (Pull request #2023) @SarahAlidoost
- Avoid importing private ESMValCore functions in CMORizer (Pull request #2027) @bouweandela
- Fix extract_seasons in validation recipe (Pull request #2054) Javier Vegas-Regidor

12.13.3 Deprecations

- Deprecate classes Variable, Variables and Datasets (Pull request #1944) @schlunma
- Python 3.9: remove pynio as dependency and replace with rasterio and pin Matplotlib>3.3.1 and pin cartopy>=0.18 (Pull request #1997) @valeriupredoi
- Removed write_plots and write_netcdf in some python diagnostics (Pull request #2036) @schlunma

12.13.4 Documentation

- Update instructions on making a release (Pull request #1867) @bouweandela
- Update review.rst (Pull request #1917) @axel-lauer
- Add guidance on how to review a pull request (Pull request #1872) @bouweandela
- Adding tutorial links to documentation (Pull request #1927) @hb326
- Added bibtex file for schlund20jgr (Pull request #1928) @schlunma
- Documentation contact added the actual email for the mailing list (Pull request #1938) @valeriupredoi
- Make CircleCI badge specific to main branch (Pull request #1831) @bouweandela
- Documentation on how to move code from a private repository to a public repository (Pull request #1920) @hb326
- Refine pull request review guidelines (Pull request #1924) @stefsmeets
- Update release schedule (Pull request #1948) @zklaus
- Improve contact info and move to more prominent location (Pull request #1950) @bouweandela
- Add some maintainers to some recipes that are missing them (Pull request #1970) @valeriupredoi
- Update core team info (Pull request #1973) @axel-lauer
- Combine installation from source instructions and add common issues (Pull request #1971) @bouweandela
- Update iris documentation URL for sphinx (Pull request #2003) @bouweandela
- Fix iris documentation link(s) with new iris3 location on readthedocs (Pull request #2012) @valeriupredoi
- Document how to run tests for installation verification (Pull request #1847) @valeriupredoi
- List Remi Kazeroni as a code owner and sole merger of CMORizers (Pull request #2017) @bouweandela
- Install documentation: mention that we build conda package with python>=3.7 (Pull request #2030) @valeriupredoi
- Recipe and documentation update for ERA5-Land. (Pull request #1906) @katjaweigel
- Update changelog and changelog tool for v2.2.0 (Pull request #2043) Javier Vegas-Regidor
- Final update to the changelog for v2.2.0 (Pull request #2056) Javier Vegas-Regidor

12.13.5 Diagnostics

- Add mapplot diagnostic to ClimWIP (Pull request #1864) @lukasbrunner
- Add the option to weight variable groups in ClimWIP (Pull request #1856) @lukasbrunner
- Implementation of ensemble member recognition to the ClimWIP diagnostic (Pull request #1852) @lukasbrunner
- Restructure ClimWIP (Pull request #1919) @lukasbrunner

- Diagnostic for recipe_eyring13jgr.yml Fig. 12 (Pull request #1922) @LisaBock
- Added changes in shared functions necessary for schlund20esd (Pull request #1967) @schlunma
- Adding recipe and diagnostics for Gier et al 2020 (Pull request #1914) @bettina-gier
- Added recipe, diagnostics and documentation for Schlund et al., ESD (2020) (Pull request #2015) @schlunma
- Add PRIMAVERA Eady Growth Rate diagnostic (Pull request #1285) @sloosvel
- Implement shape parameter calibration for ClimWIP (Pull request #1905) @lukasbrunner

12.13.6 Observational and re-analysis dataset support

- Extended ESRL cmorizer (Pull request #1937) @bettina-gier
- Cmorizer for GRACE data (Pull request #1694) @bascrezee
- Cmorizer for latest ESACCI-SST data (Pull request #1895) @valeriupredoi
- Fix longitude in ESRL cmorizer (Pull request #1988) @bettina-gier
- Selectively turn off fixing bounds for coordinates during cmorization with utilities.py (Pull request #2014) @valeriupredoi
- Cmorize hadcrut5 (Pull request #1977) @mwjury
- Cmorize gpcc masking (Pull request #1995) @mwjury
- Cmorize_utils_save_1mon_Amon (Pull request #1990) @mwjury
- Cmorize gpcc fix (Pull request #1982) @mwjury
- Fix flake8 raised by develop test in cmorize_obs_gpcc.py (Pull request #2038) @valeriupredoi

12.13.7 Automatic testing

- Switched miniconda conda setup hooks for Github Actions workflows (Pull request #1913) @valeriupredoi
- Fix style issue (Pull request #1929) @bouweandela
- Fix mlr test with solution that works for CentOS too (Pull request #1936) @valeriupredoi
- Temporary deactivation Github Actions on OSX (Pull request #1939) @valeriupredoi
- Fix conda installation test on CircleCI (Pull request #1952) @bouweandela
- Github Actions: change time for cron job that installs from conda (Pull request #1969) @valeriupredoi
- CI upload relevant artifacts for test job (Pull request #1999) @valeriupredoi
- Github Actions test that runs with the latest ESMValCore main (Pull request #1989) @valeriupredoi
- Introduce python 39 in Github Actions tests (Pull request #2029) @valeriupredoi
- Remove test for conda package installation on Python 3.6 (Pull request #2033) @valeriupredoi
- Update codacy coverage reporter to fix coverage (Pull request #2039) @bouweandela

12.13.8 Installation

- Simplify installation of R development dependencies (Pull request #1930) @bouweandela
- Fix docker build (Pull request #1934) @bouweandela
- Use new conda environment for installing ESMValTool in Docker containers (Pull request #1993) @bouweandela
- Fix conda build (Pull request #2026) @bouweandela

12.13.9 Improvements

- Allow multiple references for a cmorizer script (Pull request #1953) @SarahAlidoost
- Add GRACE to the recipe check_obs (Pull request #1963) @remi-kazeroni
- Align ESMValTool to ESMValCore=2.2.0 (adopt iris3, fix environment for new Core release) (Pull request #1874) @stefsmeets
- Make it possible to use write_plots and write_netcdf from recipe instead of config-user.yml (Pull request #2018) @bouweandela
- Revise lisflood and hype recipes (Pull request #2035) @SarahAlidoost
- Set version to 2.2.0 (Pull request #2042) Javier Vegas-Regidor

12.14 v2.1.1

This release includes

12.14.1 Improvements

- Fix the conda build on CircleCI (Pull request #1883) @bouweandela
- Pin matplotlib to <3.3 and add compilers (Pull request #1898) @bouweandela
- Pin esmvaltool subpackages to the same version and build as the esmvaltool conda package (Pull request #1899) @bouweandela

12.14.2 Documentation

- Release notes v2.1.1 (Pull request #1932) @valeriupredoi

12.15 v2.1.0

This release includes

12.15.1 Diagnostics

- Add extra steps to diagnostic to make output of hydrology/recipe_lisflood.yml usable by the LISFLOOD model (Pull request #1737) @JaroCamphuijsen
- Recipe to reproduce the 2014 KNMI Climate Scenarios (kcs). (Pull request #1667) @Peter9192
- Implement the climwip weighting scheme in a recipe and diagnostic (Pull request #1648) @JaroCamphuijsen
- Remove unreviewed autoassess recipes (Pull request #1840) @valeriupredoi
- Changes in shared scripts for Schlund et al., JGR: Biogeosciences, 2020 (Pull request #1845) @schlunma
- Updated derivation test recipe (Pull request #1790) @schlunma
- Support for multiple model occurrence in perf main (Pull request #1649) @bettina-gier
- Add recipe and diagnostics for Schlund et al., JGR: Biogeosciences, 2020 (Pull request #1860) @schlunma
- Adjust recipe_extract_shape.yml to recent changes in the example diagnostic.py (Pull request #1880) @bouweandela

12.15.2 Documentation

- Add pip installation instructions (Pull request #1783) @bouweandela
- Add installation instruction for R and Julia dependencies tot pip install (Pull request #1787) @bouweandela
- Avoid autodocsumm 0.2.0 and update documentation build dependencies (Pull request #1794) @bouweandela
- Add more information on working on cluster attached to ESGF node (Pull request #1821) @bouweandela
- Add release strategy to community documentation (Pull request #1809) @zklaus
- Update esmvaltool run command everywhere in documentation (Pull request #1820) @bouweandela
- Add more info on documenting a recipe (Pull request #1795) @bouweandela
- Improve the Python example diagnostic and documentation (Pull request #1827) @bouweandela
- Improve description of how to use draft_release_notes.py (Pull request #1848) @bouweandela
- Update changelog for release 2.1 (Pull request #1886) @valeriupredoi

12.15.3 Improvements

- Fix R installation in WSL (Pull request #1789) Javier Vegas-Regidor
- Add pre-commit for linting/formatting (Pull request #1796) @stefsmeets
- Speed up tests on CircleCI and use pytest to run them (Pull request #1804) @bouweandela
- Move pre-commit excludes to top-level and correct order of lintr and styler (Pull request #1805) @stefsmeets
- Remove isort setup to fix formatting conflict with yapf (Pull request #1815) @stefsmeets
- GitHub Actions (Pull request #1806) @valeriupredoi
- Fix yapf-isort import formatting conflict (Pull request #1822) @stefsmeets
- Replace vmprof with vprof as the default profiler (Pull request #1829) @bouweandela
- Update ESMValCore v2.1.0 requirement (Pull request #1839) Javier Vegas-Regidor
- Pin iris to version 2 (Pull request #1881) @bouweandela
- Pin eccodes to not use eccodes=2.19.0 for cdo to work fine (Pull request #1869) @valeriupredoi
- Increase version to 2.1.0 and add release notes (Pull request #1868) @valeriupredoi
- Github Actions Build Packages and Deploy tests (conda and PyPi) (Pull request #1858) @valeriupredoi

12.15.4 Observational and re-analysis dataset support

- Added CMORizer for Scripps-CO2-KUM (Pull request #1857) @schlunma

12.16 v2.0.0

This release includes

12.16.1 Bug fixes

- Fix pep8-naming errors and fix zmnam diagnostic (Pull request #1702) @bouweandela
- Fix keyword argument in cmorize_obs (Pull request #1721) @mattiarighi
- Fixed JMA-TRANSCOM CMORizer (Pull request #1735) @schlunma

- Fix bug in extract_doi_value (Pull request #1734) @bascrezee
- Fix small errors in the arctic_ocean diagnostic (Pull request #1722) @koldunovn
- Flatten ancestor lists for diag_spei.R and diag_spi.R. (Pull request #1745) @katjaweigel
- Fix for recipe_ocean_ice_extent.yml (Pull request #1744) @mattiarighi
- Fix recipe_combined_indices.yml provenance (Pull request #1746) Javier Vegas-Regidor
- Fix provenance in recipe_multimodel_products (Pull request #1747) Javier Vegas-Regidor
- Exclude FGOALS-g2 due to ESMValCore issue #728 (Pull request #1749) @mattiarighi
- Fix recipe_modes_of_variability (Pull request #1753) Javier Vegas-Regidor
- Flatten lists for ancestors for hyint to prevent nested lists. (Pull request #1752) @katjaweigel
- Fix bug in cmorize_obs_eppley_vgpm_modis.py (#1729) (Pull request #1759) @tomaslovato
- Correct mip for clltkisccp in example derive preprocessor recipe (Pull request #1768) @bouweandela
- Update date conversion in recipe_hype.yml (Pull request #1769) @bouweandela
- Fix recipe_correlation.yml (Pull request #1767) @bouweandela
- Add attribute positive: down to plev coordinate in ERA-Interim CMORizer (Pull request #1771) @bouweandela
- Fix sispeed in recipe_preprocessor_derive_test (Pull request #1772) Javier Vegas-Regidor
- Fix extreme events and extreme index ancestors (Pull request #1774) @katjaweigel
- Correct date in output filenames of ERA5 CMORizer recipe (Pull request #1773) @bouweandela
- Exclude WOA from multi-model stats in recipe_ocean_bgc (Pull request #1778) @mattiarighi

12.16.2 Diagnostics

- Enhancement of the hyint recipe to include etccdi indices (Pull request #1133) @earnone
- Add lazy regridding for wflow diagnostic (Pull request #1630) @bouweandela
- Miles default domains to include lat=0 (Pull request #1626) @jhardenberg
- Miles: selection of reference dataset based on experiment (Pull request #1632) @jhardenberg
- New recipe/diagnostic: recipe_li17natcc.yml for Axels GMD Paper (Pull request #1567) @katjaweigel
- New recipe/diagnostics: recipe_deangelis_for_gmdpart4.yml for Axels GMD Paper (Pull request #1576) @katjaweigel
- EWaterCycle: Add recipe to prepare input for LISFLOOD (Pull request #1298) @sverhoeven
- Use area weighted regridding in wflow diagnostic (Pull request #1643) @bouweandela
- Workaround for permetrics recipe until Iris3 (Pull request #1674) @mattiarighi
- C3S_511_MPQB_bas-features (Pull request #1465) @bascrezee
- Additional Land perfmetrics (Pull request #1641) @bettina-gier
- Necessary diagnostic from eyring06jgr for the release of version2 (Pull request #1686) @hb326
- Drought characteristics based on Martin2018 and SPI for gmd paper (Pull request #1689) @katjaweigel
- Additional features and bugfixes for recipe anav13clim (Pull request #1723) @bettina-gier
- Gmd laueretal2020 revisions (Pull request #1725) @axel-lauer
- Wenzel16nature (Pull request #1692) @zechlau

- Add mask albedolandcover (Pull request #1673) @bascrezee
- IPCC AR5 fig. 9.3 (seasonality) (Pull request #1726) @axel-lauer
- Added additional emergent constraints on ECS (Pull request #1585) @schlunma
- A diagnostic to evaluate the turnover times of land ecosystem carbon (Pull request #1395) koir-su
- Removed multi_model_statistics step in recipe_oceans_example.yml as a workaround (Pull request #1779) @valeriupredoi

12.16.3 Documentation

- Extend getting started instructions to obtain config-user.yml (Pull request #1642) @Peter9192
- Extend information about native6 support on RTD (Pull request #1652) @Peter9192
- Update citation of ESMValTool paper in the doc (Pull request #1664) @mattiarighi
- Updated references to documentation (now docs.esmvaltool.org) (Pull request #1679) @axel-lauer
- Replace dead link with ESGF link. (Pull request #1681) @mattiarighi
- Add all European grants to Zenodo (Pull request #1682) @bouweandela
- Update Sphinx to v3 or later (Pull request #1685) @bouweandela
- Small fix to number of models in ensclus documentation (Pull request #1691) @jhardenberg
- Move draft_release_notes.py from ESMValCore to here and update (Pull request #1701) @bouweandela
- Improve the installation instructions (Pull request #1634) @valeriupredoi
- Improve description of how to implement provenance in diagnostic (Pull request #1750) @SarahAlidoost
- Update command line interface documentation and add links to ESMValCore configuration documentation (Pull request #1776) @bouweandela
- Documentation on how to find shapefiles for hydrology recipes (Pull request #1777) @JaroCamphuijsen

12.16.4 Improvements

- Pin flake8<3.8.0 (Pull request #1635) @valeriupredoi
- Update conda package path in more places (Pull request #1636) @bouweandela
- Remove curly brackets around issue number in pull request template (Pull request #1637) @bouweandela
- Fix style issue in test (Pull request #1639) @bouweandela
- Update Codacy badges (Pull request #1662) @bouweandela
- Support extra installation methods in R (Pull request #1360) Javier Vegas-Regidor
- Add ncdf4.helpers package as a dependency again (Pull request #1678) @bouweandela
- Speed up conda installation (Pull request #1677) @bouweandela
- Update CMORizers and recipes for ESMValCore v2.0.0 (Pull request #1699) @SarahAlidoost
- Update setup.py for PyPI package (Pull request #1700) @bouweandela
- Cleanup recipe headers before the release (Pull request #1740) @mattiarighi
- Add colortables as esmvaltool subcommand (Pull request #1666) Javier Vegas-Regidor
- Increase version to v2.0.0 (Pull request #1756) @bouweandela
- Update job script (Pull request #1757) @mattiarighi

- Read authors and description from .zenodo.json (Pull request #1758) @bouweandela
- Update docker recipe to install from source (Pull request #1651) Javier Vegas-Regidor

12.16.5 Observational and re-analysis dataset support

- Cmorize aphro ma (Pull request #1555) @mwjury
- Respectable testing for cmorizers/obs/utilities.py and cmorizers/obs/cmorize_obs.py (Pull request #1517) @valeriupredoi
- Fix start year in recipe_check_obs (Pull request #1638) @mattiarighi
- Cmorizer for the PERSIANN-CDR precipitation data (Pull request #1633) @hb326
- Cmorize eobs (Pull request #1554) @mwjury
- Update download cds satellite lai fapar (Pull request #1654) @bascrezee
- Added monthly mean vars (ta, va, zg) to era5 cmorizer via recipe (Pull request #1644) @egalytska
- Make format time check more flexible (Pull request #1661) @mattiarighi
- Exclude od550lt1aer from recipe_check_obs.yml (Pull request #1720) @mattiarighi
- PERSIANN-CDR cmorizer update: adding the capability to save monthly mean files (Pull request #1728) @hb326
- Add standard_name attribute to lon and lat in cmorize_obs_esacci_oc.py (Pull request #1760) @tomaslovato
- Allow for incomplete months on daily frequency in cmorizer ncl utilities (Pull request #1754) @mattiarighi
- Fix AURA-TES cmorizer (Pull request #1766) @mattiarighi

12.17 v2.0.0b4

This release includes

12.17.1 Bug fixes

- Fix HALOE plev coordinate (Pull request #1590) @mattiarighi
- Fix tro3 units in HALOE (Pull request #1591) @mattiarighi

12.17.2 Diagnostics

- Applicate sea ice negative feedback (Pull request #1299) Javier Vegas-Regidor
- Add Russell18jgr ocean diagnostics (Pull request #1592) @bouweandela
- Refactor marrmot recipe and diagnostic to use ERA5 daily data made by new cmorizer (Pull request #1600) @SarahAlidoost
- In recipe_wflow, use daily ERA5 data from the new cmorizer. (Pull request #1599) @Peter9192
- In wflow diagnostic, calculate PET after(!) interpolation and lapse rate correction (Pull request #1618) @jeromaerts
- Fixed wenz14jgr (Pull request #1562) @zechlau
- Update portrait_plot.ncl (Pull request #1625) @bettina-gier

12.17.3 Documentation

- Restructure documentation (Pull request #1587) @bouweandela
- Add more links to documentation (Pull request #1595) @bouweandela
- Update links in readme (Pull request #1598) @bouweandela
- Minor improvements to installation documentation (Pull request #1608) @bouweandela
- Add info for new mailing list to documentation. (Pull request #1607) @bjoernbroetz
- Update making a release documentation (Pull request #1627) @bouweandela

12.17.4 Improvements

- Avoid broken pytest-html plugin (Pull request #1583) @bouweandela
- Remove reference section in config-references.yml (Pull request #1545) @SarahAlidoost
- Various improvements to development infrastructure (Pull request #1570) @bouweandela
- Install scikit-learn from conda, remove libunwind as a direct dependency (Pull request #1611) @valeriupredoi
- Create conda subpackages and enable tests (Pull request #1624) @bouweandela

12.17.5 Observational and re-analysis dataset support

- Cmorizer for HALOE (Pull request #1581) @mattiarighi
- Add CMORizer for CT2019 (Pull request #1604) @schlunma

For older releases, see the release notes on https://github.com/ESMValGroup/ESMValTool/releases.

Part V

Indices and tables

- genindex
- search

PYTHON MODULE INDEX

e

esmvaltool.diag_scripts.climate_metrics.tcre, 616
esmvaltool.diag_scripts.climate_metrics.zec, 617
esmvaltool.diag_scripts.droughts.collect_drought, 561
esmvaltool.diag_scripts.droughts.diffmap, 562
esmvaltool.diag_scripts.emergent_constraints, 567
esmvaltool.diag_scripts.emergent_constraints.cox18nature, 564
esmvaltool.diag_scripts.emergent_constraints.ecs_scatter, 564
esmvaltool.diag_scripts.emergent_constraints.multiple_constraints, 565
esmvaltool.diag_scripts.emergent_constraints.single_constraint, 566
esmvaltool.diag_scripts.hydrology.hour_of_max_precipitation, 617
esmvaltool.diag_scripts.lifetime.lifetime, 575
esmvaltool.diag_scripts.monitor.compute_eofs, 579
esmvaltool.diag_scripts.monitor.monitor, 577
esmvaltool.diag_scripts.monitor.monitor_base, 586
esmvaltool.diag_scripts.monitor.multi_datasets, 580
esmvaltool.diag_scripts.ocean.diagnostic_biases, 588
esmvaltool.diag_scripts.ocean.diagnostic_maps, 590
esmvaltool.diag_scripts.ocean.diagnostic_maps_quad, 592
esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs, 594
esmvaltool.diag_scripts.ocean.diagnostic_profiles, 597
esmvaltool.diag_scripts.ocean.diagnostic_seaice, 603
esmvaltool.diag_scripts.ocean.diagnostic_timeseries, 598
esmvaltool.diag_scripts.ocean.diagnostic_tools, 607
esmvaltool.diag_scripts.ocean.diagnostic_transects, 600
esmvaltool.diag_scripts.plot_histogram, 613
esmvaltool.diag_scripts.portrait_plot, 614
esmvaltool.diag_scripts.seaborn_diag, 612
esmvaltool.diag_scripts.shared, 543
esmvaltool.diag_scripts.shared.iris_helpers, 556
esmvaltool.diag_scripts.shared.plot, 559

INDEX

A

add_dataset() (*esmvaltool.diag_scripts.shared.Datasets method*), 544
add_legend_outside_right() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 607
add_linear_regression() (*in module esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs*), 595
add_map_subplot() (*in module esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs*), 595
add_sea_floor() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 601
add_to_data() (*esmvaltool.diag_scripts.shared.Datasets method*), 544
add_vars() (*esmvaltool.diag_scripts.shared.Variables method*), 551
agregate_by_season() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 604
apply_supermeans() (*in module esmvaltool.diag_scripts.shared*), 553

B

best_guess (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 589
bgc_units() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 608
bias (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 588
border1 (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 589
border2 (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 589

C

calculate_area_time_series() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 604
calculate_statistics() (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis method*), 589
cdf() (*in module esmvaltool.diag_scripts.emergent_constraints*), 568
check_coordinate() (*in module esmvaltool.diag_scripts.shared.iris_helpers*), 556
check_metadata() (*in module esmvaltool.diag_scripts.emergent_constraints*), 568
CMIP, **693**
combine_groups() (*in module esmvaltool.diag_scripts.emergent_constraints*), 568
constraint_info_array() (*in module esmvaltool.diag_scripts.emergent_constraints*), 569
convert_to_iris() (*in module esmvaltool.diag_scripts.shared.iris_helpers*), 557
count() (*esmvaltool.diag_scripts.shared.Variable method*), 549
create_ice_cmap() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 605
create_plotting_data() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps_quad*), 592
create_provenance() (*in module esmvaltool.diag_scripts.ocean.diagnostic_biases*), 590
create_quadmap() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps_quad*), 593
create_simple_scatterplot() (*in module esmvaltool.diag_scripts.emergent_constraints*), 569
cube_time_to_float() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 608
Cylc, **693**

D

data (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 589
Data4Analyis (*class in esmvaltool.diag_scripts.ocean.diagnostic_biases*), 588
Datasets (*class in esmvaltool.diag_scripts.shared*), 543
decadal_average() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 608
determine_profiles_str() (*in module esmvaltool.diag_scripts.ocean.diagnostic_profiles*), 598
determine_reference() (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis method*), 590
determine_set_y_logscale() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 601
determine_transect_str() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 601

E

ESM, **693**
ESMValCore, **693**
ESMValTool, **693**
esmvaltool.diag_scripts.climate_metrics.tcre module, 616
esmvaltool.diag_scripts.climate_metrics.zec module, 617
esmvaltool.diag_scripts.droughts.collect_drought module, 561
esmvaltool.diag_scripts.droughts.diffmap module, 562
esmvaltool.diag_scripts.emergent_constraints module, 567
esmvaltool.diag_scripts.emergent_constraints.cox18nature module, 564
esmvaltool.diag_scripts.emergent_constraints.ecs_scatter module, 564
esmvaltool.diag_scripts.emergent_constraints.multiple_constraints module, 565
esmvaltool.diag_scripts.emergent_constraints.single_constraint module, 566
esmvaltool.diag_scripts.hydrology.hour_of_max_precipitation module, 617
esmvaltool.diag_scripts.lifetime.lifetime module, 575
esmvaltool.diag_scripts.monitor.compute_eofs module, 579
esmvaltool.diag_scripts.monitor.monitor module, 577
esmvaltool.diag_scripts.monitor.monitor_base module, 586
esmvaltool.diag_scripts.monitor.multi_datasets module, 580
esmvaltool.diag_scripts.ocean.diagnostic_biases module, 588
esmvaltool.diag_scripts.ocean.diagnostic_maps module, 590
esmvaltool.diag_scripts.ocean.diagnostic_maps_quad module, 592
esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs module, 594
esmvaltool.diag_scripts.ocean.diagnostic_profiles module, 597
esmvaltool.diag_scripts.ocean.diagnostic_seaice module, 603
esmvaltool.diag_scripts.ocean.diagnostic_timeseries module, 598
esmvaltool.diag_scripts.ocean.diagnostic_tools module, 607
esmvaltool.diag_scripts.ocean.diagnostic_transects module, 600
esmvaltool.diag_scripts.plot_histogram module, 613
esmvaltool.diag_scripts.portrait_plot module, 614
esmvaltool.diag_scripts.seaborn_diag module, 612
esmvaltool.diag_scripts.shared module, 543
esmvaltool.diag_scripts.shared.iris_helpers module, 556
esmvaltool.diag_scripts.shared.plot module, 559
export_csv() (*in module esmvaltool.diag_scripts.emergent_constraints*), 569
extract_variables() (*in module esmvaltool.diag_scripts.shared*), 553

F

folder() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 608

G

get_array_range() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 608
get_caption() (*in module esmvaltool.diag_scripts.emergent_constraints*), 570
get_cfg() (*in module esmvaltool.diag_scripts.shared*), 553

get_colors() (*in module esmvaltool.diag_scripts.emergent_constraints*), 570
get_colour_from_cmap() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 609
get_constraint() (*in module esmvaltool.diag_scripts.emergent_constraints*), 570
get_constraint_from_df() (*in module esmvaltool.diag_scripts.emergent_constraints*), 571
get_control_exper_obs() (*in module esmvaltool.diag_scripts.shared*), 553
get_cube_range() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 609
get_cube_range_diff() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 609
get_data() (*esmvaltool.diag_scripts.shared.Datasets method*), 545
get_data_list() (*esmvaltool.diag_scripts.shared.Datasets method*), 545
get_dataset_info() (*esmvaltool.diag_scripts.shared.Datasets method*), 545
get_dataset_info_list() (*esmvaltool.diag_scripts.shared.Datasets method*), 546
get_dataset_style() (*in module esmvaltool.diag_scripts.shared.plot*), 559
get_decade() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 609
get_diagnostic_filename() (*in module esmvaltool.diag_scripts.shared*), 553
get_groups() (*in module esmvaltool.diag_scripts.emergent_constraints*), 571
get_image_format() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 609
get_image_path() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 610
get_info() (*esmvaltool.diag_scripts.shared.Datasets method*), 546
get_info_list() (*esmvaltool.diag_scripts.shared.Datasets method*), 546
get_input_data() (*in module esmvaltool.diag_scripts.emergent_constraints*), 571
get_input_files() (*in module esmvaltool.diag_scripts.emergent_constraints*), 572
get_input_files() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 610
get_mean_cube() (*in module esmvaltool.diag_scripts.shared.iris_helpers*), 557
get_obs_projects() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 610
get_path() (*esmvaltool.diag_scripts.shared.Datasets method*), 547
get_path_list() (*esmvaltool.diag_scripts.shared.Datasets method*), 547
get_path_to_mpl_style() (*in module esmvaltool.diag_scripts.shared.plot*), 559
get_plot_filename() (*in module esmvaltool.diag_scripts.shared*), 553
get_plot_folder() (*esmvaltool.diag_scripts.monitor.monitor_base.MonitorBase method*), 587
get_plot_name() (*esmvaltool.diag_scripts.monitor.monitor_base.MonitorBase method*), 587
get_plot_path() (*esmvaltool.diag_scripts.monitor.monitor_base.MonitorBase method*), 587
get_pole() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 605
get_provenance_record() (*esmvaltool.diag_scripts.monitor.monitor_base.MonitorBase static method*), 587
get_provenance_record() (*in module esmvaltool.diag_scripts.emergent_constraints*), 572
get_season() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 605
get_time_string() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 605
get_xy_data_without_nans() (*in module esmvaltool.diag_scripts.emergent_constraints*), 572
get_year() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 605
global_contourf() (*in module esmvaltool.diag_scripts.shared.plot*), 559
global_pcolormesh() (*in module esmvaltool.diag_scripts.shared.plot*), 559
group_metadata() (*in module esmvaltool.diag_scripts.shared*), 554

guess_calendar_datetime() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 610

I

index() (*esmvaltool.diag_scripts.shared.Variable method*), 549
intersect_dataset_coordinates() (*in module esmvaltool.diag_scripts.shared.iris_helpers*), 557
iris_dict() (*esmvaltool.diag_scripts.shared.Variables method*), 551
iris_project_constraint() (*in module esmvaltool.diag_scripts.shared.iris_helpers*), 557

K

KGOs, **693**

L

load_data() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps_quad*), 593
load_thresholds() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 611
log() (*esmvaltool.diag_scripts.shared.ProvenanceLogger method*), 548
long_name (*esmvaltool.diag_scripts.shared.Variable attribute*), 549
long_name() (*esmvaltool.diag_scripts.shared.Variables method*), 551

M

main() (*in module esmvaltool.diag_scripts.ocean.diagnostic_biases*), 590
main() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps*), 591
main() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps_quad*), 593
main() (*in module esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs*), 596
main() (*in module esmvaltool.diag_scripts.ocean.diagnostic_profiles*), 598
main() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 606
main() (*in module esmvaltool.diag_scripts.ocean.diagnostic_timeseries*), 599
main() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 602
make_cube_layer_dict() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 611
make_cube_region_dict() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 602
make_depth_safe() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 602
make_map_contour() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps*), 591
make_map_extent_plots() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 606
make_map_plots() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps*), 591
make_map_plots() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 606
make_model_vs_obs_plots() (*in module esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs*), 596
make_polar_map() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 606
make_profiles_plots() (*in module esmvaltool.diag_scripts.ocean.diagnostic_profiles*), 598
make_scatter() (*in module esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs*), 596
make_time_series_plots() (*in module esmvaltool.diag_scripts.ocean.diagnostic_timeseries*), 599
make_transect_contours() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 602
make_transects_plots() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 602
make_ts_plots() (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 606
mask (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 588
mask_data() (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis method*), 590
mask_type (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyi*

attribute), 589
match_model_to_key() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 611
modify_var() (*esmvaltool.diag_scripts.shared.Variables method*), 551
module
esmvaltool.diag_scripts.climate_metrics.tcre, 616
esmvaltool.diag_scripts.climate_metrics.zec, 617
esmvaltool.diag_scripts.droughts.collect_drought, 561
esmvaltool.diag_scripts.droughts.diffmap, 562
esmvaltool.diag_scripts.emergent_constraints, 567
esmvaltool.diag_scripts.emergent_constraints.cox18nature, 564
esmvaltool.diag_scripts.emergent_constraints.ecs_scatter, 564
esmvaltool.diag_scripts.emergent_constraints.multi_model_constraint, 565
esmvaltool.diag_scripts.emergent_constraints.single_constraint, 566
esmvaltool.diag_scripts.hydrology.hour_of_max_precipitation, 617
esmvaltool.diag_scripts.lifetime.lifetime, 575
esmvaltool.diag_scripts.monitor.compute_eofs, 579
esmvaltool.diag_scripts.monitor.monitor, 577
esmvaltool.diag_scripts.monitor.monitor_base, 586
esmvaltool.diag_scripts.monitor.multi_datasets, 580
esmvaltool.diag_scripts.ocean.diagnostic_biases, 588
esmvaltool.diag_scripts.ocean.diagnostic_maps, 590
esmvaltool.diag_scripts.ocean.diagnostic_maps_quad, 592
esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs, 594
esmvaltool.diag_scripts.ocean.diagnostic_profiles, 597
esmvaltool.diag_scripts.ocean.diagnostic_seaice, 603
esmvaltool.diag_scripts.ocean.diagnostic_timeseries, 598
esmvaltool.diag_scripts.ocean.diagnostic_tools, 607
esmvaltool.diag_scripts.ocean.diagnostic_transects, 600
esmvaltool.diag_scripts.plot_histogram, 613
esmvaltool.diag_scripts.portrait_plot, 614
esmvaltool.diag_scripts.seaborn_diag, 612
esmvaltool.diag_scripts.shared, 543
esmvaltool.diag_scripts.shared.iris_helpers, 556
esmvaltool.diag_scripts.shared.plot, 559
MonitorBase (*class in esmvaltool.diag_scripts.monitor.monitor_base*), 586
moving_average() (*in module esmvaltool.diag_scripts.ocean.diagnostic_timeseries*), 599
multi_dataset_scatterplot() (*in module esmvaltool.diag_scripts.shared.plot*), 560
multi_model_contours() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps*), 591
multi_model_contours() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 603
multi_model_time_series() (*in module esmvaltool.diag_scripts.ocean.diagnostic_timeseries*), 600

N

name (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 588

O

obtain_data() (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis method*), 590

P

pandas_object_to_cube() (*in module esmvaltool.diag_scripts.emergent_constraints*), 572
plot_bias_plot() (*in module esmvaltool.diag_scripts.ocean.diagnostic_biases*), 590
plot_cube() (*esmvaltool.diag_scripts.monitor.monitor_base.MonitorBase method*), 587
plot_global_single_level() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps_quad*), 594
plot_individual_scatterplots() (*in module esmvaltool.diag_scripts.emergent_constraints*), 573

plot_merged_scatterplots() (*in module esmvaltool.diag_scripts.emergent_constraints*), 573
plot_target_distributions() (*in module esmvaltool.diag_scripts.emergent_constraints*), 574
plot_timeseries() (*esmvaltool.diag_scripts.monitor.monitor_base.MonitorBase method*), 587
prepare_cube_for_merging() (*in module esmvaltool.diag_scripts.shared.iris_helpers*), 558
prepare_provenance_record() (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 611
ProvenanceLogger (*class in esmvaltool.diag_scripts.shared*), 548

Q

quickplot() (*in module esmvaltool.diag_scripts.shared.plot*), 561

R

record_plot_provenance() (*esmvaltool.diag_scripts.monitor.monitor_base.MonitorBase method*), 587
ref_cube (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 589
reference (*esmvaltool.diag_scripts.ocean.diagnostic_biases.Data4Analyis attribute*), 589
regression_line() (*in module esmvaltool.diag_scripts.emergent_constraints*), 574
Rose, **693**
rounds_sig() (*in module esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs*), 596
RTW, **693**
run_diagnostic() (*in module esmvaltool.diag_scripts.shared*), 554

S

save_cube() (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps_quad*), 594
save_data() (*in module esmvaltool.diag_scripts.shared*), 554
save_figure() (*in module esmvaltool.diag_scripts.shared*), 555
scatterplot() (*in module esmvaltool.diag_scripts.shared.plot*), 561
select_metadata() (*in module esmvaltool.diag_scripts.shared*), 555
set_data() (*esmvaltool.diag_scripts.shared.Datasets method*), 547
set_plot_appearance() (*in module esmvaltool.diag_scripts.emergent_constraints*), 574
short_name (*esmvaltool.diag_scripts.shared.Variable attribute*), 550
short_name() (*esmvaltool.diag_scripts.shared.Variables method*), 551
short_names() (*esmvaltool.diag_scripts.shared.Variables method*), 551
sorted_group_metadata() (*in module esmvaltool.diag_scripts.shared*), 555
sorted_metadata() (*in module esmvaltool.diag_scripts.shared*), 556
standard_name (*esmvaltool.diag_scripts.shared.Variable attribute*), 550
standard_name() (*esmvaltool.diag_scripts.shared.Variables method*), 552
standard_names() (*esmvaltool.diag_scripts.shared.Variables method*), 552
standard_prediction_error() (*in module esmvaltool.diag_scripts.emergent_constraints*), 574

T

target_pdf() (*in module esmvaltool.diag_scripts.emergent_constraints*), 575
timeplot() (*in module esmvaltool.diag_scripts.ocean.diagnostic_timeseries*), 600
titlify() (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 603

U

unify_1d_cubes() (*in module esmvaltool.diag_scripts.shared.iris_helpers*), 558
unify_time_coord() (*in module esmvaltool.diag_scripts.shared.iris_helpers*), 558
units (*esmvaltool.diag_scripts.shared.Variable attribute*), 550
units() (*esmvaltool.diag_scripts.shared.Variables method*), 552

V

var_name() (*esmvaltool.diag_scripts.shared.Variables method*), 552
Variable (*class in esmvaltool.diag_scripts.shared*), 549
Variables (*class in esmvaltool.diag_scripts.shared*), 550
variables_available() (*in module esmvaltool.diag_scripts.shared*), 556

vars_available() (*esmvaltool.diag_scripts.shared.Variables method*), 552